Gesprächsforschung - Online-Zeitschrift zur verbalen Interaktion (ISSN 1617-1837)
Ausgabe 24 (2023), Seite 67-126 (www.gespraechsforschung-ozs.de)

Arbeiten mit zeitalignierten multimodalen Korpora in *R*. Vorstellung des *Aligned Corpus Toolkit* (act)[1]

Oliver Ehmer

Abstract
In der Gesprächsforschung werden seit einigen Jahren in immer stärkerem Maße digitale Methoden eingesetzt. Dies gilt sowohl für die zeitalignierte Transkription und Annotation in spezialisierten Programmen, die Vorhaltung multimedialer Daten in Datenbanken als auch deren Analyse. Während in verschiedenen geisteswissenschaftlichen Disziplinen die Programmierumgebung *R* zur Auswertung und Visualisierung von Daten bereits fest etabliert ist, stellt dies in der Gesprächsforschung und Interaktionalen Linguistik einen bislang noch relativ selten vertretenen Ansatz dar. An dieser Stelle setzt das *Aligned Corpus Toolkit* (*act*) (Ehmer 2021a) an. Es handelt sich um eine Bibliothek, die *R* um verschiedene Funktionen zur Arbeit mit multimodalen zeitalignierten Daten erweitert. Dabei ist die Bibliothek stark auf die gesprächsanalytische und interaktionslinguistische Arbeit ausgerichtet. Zu ihren Funktionen zählen u.a. der Import und Export ganzer Korpora, die Erstellung von Drucktranskripten in anpassbaren Layouts und eine differenzierte Suchfunktion, die u.a. erlaubt, in normalisierten Annotationen zu suchen und Zeichenketten zu finden, die Annotationsgrenzen überspannen. Weiterhin können für die Suchtreffer Transkript- und Medienausschnitte (mit Kontext) erstellt werden. Der Beitrag stellt das Arbeitsprinzip des *Aligned Corpus Toolkit* vor und führt anhand einfacher und ausführlich beschriebener Code-Beispiele in die Grundfunktionen ein.

Keywords: zeitalignierte multimodale Korpora – R – Korpuslinguistik.

English Abstract
In recent years, the use of digital methods in conversation research has become more widespread. This applies particularly to time-aligned transcription and annotation in specialized programs, the storage of multimedia data in databases, and data analysis. While the programming environment *R* is already well established in various disciplines of the humanities for the analysis and visualization of data, its use in conversation research and interactional linguistics is not so common. This is where the *Aligned Corpus Toolkit* (*act*) (Ehmer 2021a) comes in. The *act* library extends *R* by various functions to work with multimodal time-aligned data. The library is strongly oriented towards conversation analytic and interactional linguistic work. Among the library functions are the import and export of annotation files (for selected recordings, but also for entire corpora), the creation of print transcripts in customizable layouts, and, most importantly, a sophisticated search function that makes it possible, among other things, to search normalized annotations and to find strings that span annotation boundaries. Transcript and media excerpts (with context) can be created for the search hits. This paper introduces the working principles of the *Aligned Corpus Toolkit* and provides an introduction to its basic functions by means of simple code examples described in detail.

Keywords: time-aligned multimodal corpora – R – corpus linguistics.

[1] Für hilfreiche Kommentare zu einer früheren Version dieses Artikels bedanke mich sehr herzlich bei Florian Dreyer, Henrike Helmer und Thomas Schmidt.

1. Digitale Analysemethoden und zeitalignierte Korpora

In den letzten Jahren haben sich in der Gesprächsforschung und interaktionalen Linguistik vermehrt digitale Methoden etabliert. Dies gilt sowohl für die Annotation als auch die Auswertung multimodaler Daten. Im Bereich der Annotation werden vermehrt Programme genutzt, die auf einer zeitalignierten Annotation basieren. Zu diesen zählen u.a. *Praat* (Boersma/Weenink 2021), *ELAN* (Wittenburg et al. 2006), *EXMARaLDA* (Schmidt 2002) und *Folker* (Schmidt/Schütte 2010). Das Datenmodell dieser Programme besteht darin, dass für einzelne Annotationen jeweils Start- und Endzeitpunkt in Bezug auf eine Mediendatei erfasst werden.[2] Darüber hinaus wird jede Annotation einer bestimmten Annotationsspur (engl. *tier*) zugewiesen, d.h. jede Annotation wird in einer bestimmten 'Zeile' erfasst. In einem einfachen Fall ist dies z.B. pro Teilnehmer an einer Interaktion eine Zeile für die verbale Transkription. Andere Annotationen, die z.B. die körperliche Ebene betreffen, können dann für jeden Interaktionsteilnehmer separat in weiteren Zeilen erfasst werden. Die zeitalignierte Annotation in einem solchen Modell mit separaten Spuren bietet für die Arbeit grundlegende Vorteile. Beispielsweise können die unterschiedlichen Spuren selektiv in Transkripte oder Suchanfragen sowie die zeitliche

[2] Einen anderen Ansatz verfolgen Programme wie z.B. *Transana* und *Clan*, welche das Prinzip eines klassischen Textverarbeitungsprogrammes aufgreifen. Hier wird direkt in einem Drucktranskriptlayout gearbeitet, in das dann Zeitmarkierungen eingefügt und damit eine Alignierung realisiert werden kann. Die resultierenden Daten sind jedoch weniger stark strukturiert als die Daten aus Programmen, die ein Spurenmodell verfolgen.

Ebene systematisch in die Analyse einbezogen werden, z. B. im Hinblick auf zeitliche Kookkurrenzen in unterschiedlichen Spuren. Dabei wurden teilweise ganze Ökosysteme von Werkzeugen entwickelt, mit denen nicht nur transkribiert werden kann, sondern auch Metadaten verwaltet und Suchabfragen realisiert werden können (Schmidt 2012). Weiterhin wurden eine Vielzahl individueller Lösungen für spezifische Anwendungen geschaffen, die teilweise frei verfügbar sind (vgl. u.a. die *ELAN Third Party Resources*[3]). Während zu Beginn der Entwicklung manche Programme (*syncWriter*, *MediaAnnotator* u.a.) proprietäre Dateiformate zur Speicherung der Daten verwendeten, haben sich mittlerweile verschiedene, größtenteils XML-basierte Standardformate etabliert, wodurch nicht nur eine nachhaltige Datenhaltung, sondern auch der Austausch zwischen verschiedenen Annotationsprogrammen gewährleistet ist. Ebenso entstanden Datenbankumgebungen, über welche die entsprechenden Korpora online verfügbar gemacht werden können, z.B. die *Datenbank für Gesprochenes Deutsch (DGD)* (Schmidt 2017) und die *prosoDB* (Gilles 2001). Während diese Werkzeuge über bedienerfreundliche grafische Benutzeroberflächen verfügen und teils einen großen Funktionsumfang bieten, sind die Möglichkeiten zur Auswertung, Modifikation und Visualisierung der Daten durch diese Programme und Datenbanken aber vorgegeben und damit prinzipiell begrenzt.

In den Digitalen Geisteswissenschaften hat sich für die Auswertung von Korpusdaten in vielen Bereichen die Nutzung der Programmiersprache und Statistiksoftware *R* (R Core Team 2021) durchgesetzt, mit der ein beinahe unbegrenztes Spektrum an Analysen, Modellierung und Visualisierung möglich ist. Fest etabliert ist die Nutzung von *R* beispielsweise in der quantitativen Korpuslinguistik (Baayen 2009; Gries 2009). Aber auch in der Interaktionalen Linguistik und in der Gesprächsanalyse (Ehmer 2022; Lanwer 2020; Luginbühl et al. 2021; Rühlemann 2020) finden sich bereits Anwendungen. Das breite Anwendungsspektrum an Auswertungs- und Visualisierungsverfahren ergibt sich nicht nur daraus, dass selbst entsprechende Werkzeuge programmiert werden können. Vielmehr ist *R* auch durch sogenannte Pakete bzw. Bibliotheken erweiterbar, um Funktionalitäten bereitzustellen, die in der Programmiersprache selbst nicht enthalten sind. Hierzu zählen u.a. Bibliotheken zur Textanalyse, dem Datamining oder zur statistischen Analyse und Visualisierung. Diese Pakete werden von einer großen Community von Entwicklern erstellt und meist kostenlos auf der zentralen Plattform *The Comprehensive R Archive Network* (*CRAN*)[4] zur Verfügung gestellt. Für die Arbeit mit zeitalignierten multimodalen Korpora stehen hier bereits Bibliotheken zur Verfügung, mit denen die Dateiformate verschiedener Annotationsprogramme verarbeitet werden können (z.B. rPraat[5] und textgRid[6] für *Praat* .TextGrid-Dateien, ExmaraldaR[7] für *EXMARaLDA* .exb-Dateien und FRelan[8] für *ELAN* .eaf-Dateien). Ein Nachteil dieser Bibliotheken liegt aber darin, dass lediglich die Dateiformate importiert und exportiert werden können, aber kaum weitergehenden Funktionen bereitgestellt werden, die für die Gesprächsforschung bzw. Interaktionale Linguistik

---

3 https://archive.mpi.nl/tla/elan/thirdparty
4 *CRAN:* https://cran.r-project.org
5 *rPraat:* https://cran.r-project.org/package=rPraat
6 *textgRid:* https://cran.r-project.org/package=textgRid
7 *ExmaraldaR:* https://github.com/TimoSchuer/ExmaraldaR
8 *FRelan:* https://github.com/langdoc/FRelan

relevant sind, wie beispielsweise die Suche in den Daten. Andere Bibliotheken, wie etwa `phonfieldwork`[9] für phonetische Arbeiten, bieten ein breiteres Funktionsrepertoire, wurden aber nicht für die Anwendung in der Interaktionalen Linguistik und Gesprächsforschung entwickelt.

An dieser Stelle setzt das *Aligned Corpus Toolkit* (*act*) (Ehmer 2021a) an. Es handelt sich um eine *R*-Bibliothek, welche in ihrem Funktionsumfang auf die interaktionslinguistische und gesprächsanalytische Arbeit ausgerichtet ist. Die `act`-Bibliothek ist Open Source und kostenlos über die Plattform *CRAN* verfügbar.[10] Im folgenden Teil A werden, ausgehend von einem Fallbeispiel, zunächst die Möglichkeiten vorgesellt, die die `act`-Bibliothek bietet. Anschließend wird in Teil B eine praktische Einführung in die Nutzung der Bibliothek gegeben. Hier werden, nach einigen vorbereitenden Schritten, die Funktionen anhand einfacher Programmierbeispiele ausführlich erläutert (Abschnitt 6). Dieser Teil richtet sich insbesondere an Leser_innen, die noch keine bzw. wenige Erfahrungen im Umgang mit *R* haben. Im Anhang wird eine Übersicht der vorgestellten Befehle gegeben (Abschnitt 7). In Abschnitt 8 sind die Links zu den zitierten Ressourcen zusammengestellt.

TEIL A: VORSTELLUNG DES *ALIGNED CORPUS TOOLKIT*

Das Grundprinzip des *Aligned Corpus Toolkit* besteht darin, dass der Inhalt von beliebig vielen Annotationsdateien in *R* importiert und in einem Korpus-Objekt gespeichert wird. Mit den in diesem Korpus-Objekt gespeicherten Daten kann dann in verschiedener Weise gearbeitet werden. Die `act`-Bibliothek stellt Funktionen zur Verfügung, mit denen die Daten durchsucht, verändert und in unterschiedlichen Formaten, z.B. als Drucktranskript oder in den Formaten von verschiedenen Annotationsprogrammen, exportiert werden können. Im Korpus-Objekt werden auch Verknüpfungen zu Mediendateien gespeichert, die, wenn möglich, direkt aus den Annotationsdateien eingelesen oder anhand der Dateinamen automatisch ermittelt werden. Das macht es möglich, in *R* nicht nur mit den Annotationsdaten, sondern auch mit den Mediendateien zu arbeiten und beispielsweise Medienausschnitte für Suchtreffer zu erstellen.

Um die Anwendungsmöglichkeiten der `act`-Bibliothek zu illustrieren, wird nun zunächst ein Fallbeispiel vorgestellt (2). Danach werden zentrale Funktionen der `act`-Bibliothek beschrieben (3) und anschließend die Bearbeitung des Fallbeispiels mit diesen Funktionen dargestellt (4).

2. Fallbeispiel: *parce que bon* als univerbierter Diskursmarker

Im gesprochenen Französisch finden sich oft Verwendungen des Konnektors *parce que* 'weil', gefolgt vom Diskursmarker *bon* 'gut/naja'. In Ehmer (2022:385-442) wird herausgearbeitet, dass neben einer lokal emergenten Kombination der Elemente auch Verwendungen als univerbierter Diskursmarker vorliegen, wobei Übergänge bestehen.

---

[9] *phonfieldwork:* https://cran.r-project.org/web/packages/phonfieldwork/index.html

[10] Weiterhin ist die Bibliothek auf der Entwickler-Plattform *GitHub* verfügbar, wo Aktualisierungen in kürzeren Abständen vorgenommen werden (siehe: https://github.com/oliverehmer/act).

Zum Einstieg in den Gegenstand kann das folgende Beispiel (1) dienen. F hat gerade die Auffassung vertreten, dass es wichtig sei, zur Schule zu gehen. Nun äußert sie, dass nicht alle Menschen diese Auffassung teilen, was sie nachfolgend begründet.

Beispiel (1): *sœur* 'Schwester'[11]
(Quelle: Mochet/Wittig (1984), Aufnahme FJ23)

```
      01 F:  c'est pAs l'avis de tous les GENS hEIn,
             ‚das ist nicht die Meinung aller Leute‘
      02 I:  <<pp> hm_[HM,> ]
                    ‚hm :hm‘
-->   03 F:           [parce] que:-
                       ‚weil‘
-->   04     bOn moi je pense comme ÇA?
             ‚gut ich denke so’
      05     mais j'ai ma SŒUR, (0.7)
             ‚aber meine Schwester (lit. aber ich habe meine Schwester)‘
      06     qui ELle,
             ‚die, sie‘
      07     ne pense pas paREIL;
             ‚denkt nicht so‘
```

In 01 formuliert M, dass nicht alle Menschen ihre Meinung teilen. Die Begründung dieser Aussage leitet die Sprecherin mit dem Konnektor *parce que* 'weil' in 03 ein. In der nachfolgenden Begründung realisiert die Sprecherin einen Perspektivenkontrast: zwar denke sie selbst so (04), ihre Schwester aber denke anders (05-07). Die Begründung enthält damit, in konzessiver Weise, gegenläufige Aspekte.

In Ehmer (2022:385-442) wird herausgearbeitet, dass bei der aufeinanderfolgenden Realisierung von *parce que* und *bon* von verschieden stark konventionalisierten Verwendungen auszugehen ist. Diese unterscheiden sich sowohl funktional als auch formal hinsichtlich der phonetisch-prosodischen Zäsurierung (Barth-Weingarten 2016; Barth-Weingarten/Ogden 2021). Bei lokal emergenten Kombinationen wird durch *bon* meist eine Häsitation markiert, bevor die mit *parce que* projizierte Begründung tatsächlich formuliert wird. Dabei liegt zwischen *parce que* und *bon* eine deutliche Zäsur vor. Bei der Verwendung als Diskursmarker hingegen werden *parce* que und *bon* hier innerhalb einer prosodischen Kontur produziert, die meist von vorangehenden und nachfolgenden Segmenten separiert ist. In funktionaler Hinsicht wird durch den Diskursmarker typischerweise eine komplexe Begründung eingeleitet, die mehrere begründende Aspekte oder relevante Hintergrundinformationen für den eigentlichen Grund beinhaltet. Die folgenden Beispiele illustrieren das Kontinuum der formalen Realisierungen.

[11] Das Transkript folgt den GAT-Konventionen (Selting et al. 2009).

Beispiel	Transkript[12]	Turnkonstruktionseinheiten / Annotationen
(2)	06 F: **PARce** <<creaky> **que:::->** 07 <<p, creaky> **BON**_euh-> (.)	2
(3) entspricht Beispiel (1)	08 F: [**parce**] **que**:- 09 **bOn** moi je pense comme ÇA?	2
(4)	12 =**parce que BON**. (0.5)	1

In den Beispielen (2) und (3) sind *parce que* und *bon* in zwei separaten Turnkonstruktionseinheiten realisiert. Im Falle von (2) ist *parce que* stark gedehnt, und nach *bon* folgt die Häsitationspartikel *euh*. Die Begründung wird erst in der nächsten Turnkonstruktionseinheit fortgesetzt. Im Falle von (3) sind *parce que* und *bon* ebenfalls in separaten Turnkonstuktionseinheiten realisiert, *parce que* ist aber weniger stark gedehnt und *bon* ist prosodisch am Beginn der nachfolgenden Turnkonstruktionseinheit integriert, mit welcher die inhaltliche Formulierung der Begründung beginnt. In Beispiel (4) sind *parce que* und *bon* innerhalb einer Turnkonstruktionseinheit realisiert, die durch mehr oder weniger starke prosodische Zäsuren von der vorangegangenen und nachfolgenden Turnkonstruktionseinheit separiert ist. Bei der Transkription solcher Sequenzen in spurbasierten Annotationsprogrammen wird jede Turnkonstruktionseinheit typischerweise in einer separaten Annotation transkribiert.

Zur Korpusanalyse dieses Phänomens ist das *Aligned Corpus Toolkit* in verschiedener Weise funktional. Zunächst werden die Annotationsdateien mit Hilfe der act-Bibliothek in *R* importiert. Ein erster Analyseschritt besteht dann in der *Suche* nach dem Phänomen. Die Suchfunktion der act-Bibliothek ermöglicht es, in orthographisch normalisierten Transkriptionen zu suchen, d.h. es wird in Annotationstexten gesucht, in denen die Transkriptionskonventionen herausgefiltert sind. Auf diese Weise können dann bei einer Suche nach der orthographischen Form *parce que* auch Suchtreffer für Annotationen erzielt werden, die aufgrund der Transkriptionskonventionen von der Standardgraphie abweichen, wie z.B. **[**parce**]** que**:** - (1) mit eckiger Klammer für überlappendes Sprechen und PARce **<<creaky>** que**:::->** (2) mit der Angabe von Knarrstimme jeweils innerhalb eines Worts. Darüber hinaus ermöglicht die Suchfunktion, über die Grenzen von Annotationen hinweg zu suchen. Beispielsweise können über die Verwendung von Platzhaltern ('Wildcards') Instanzen von *parce que bon* gefunden werden, bei denen *parce que* und *bon* in zwei unterschiedlichen Annotationen bzw. als separate Turnkonstruktionseinheiten transkribiert wurden, wie dies in Beispiel (2) (Zeilen 06-07) und in Beispiel (3) (Zeilen 08-09) der Fall ist.

Mit den Ergebnissen einer solchen Suche kann nun in verschiedener Weise weitergearbeitet werden. Beispielsweise können *Drucktranskripte* für alle Suchtreffer erstellt werden, die in einzelnen Dateien oder in einer Gesamtdatei gespeichert werden. Dabei kann ein beliebiger Kontext gewählt werden, beispielsweise 20 Sekunden vor und 10 Sekunden nach dem Suchtreffer. Dies ermöglicht eine direkte Analyse der Suchtreffer im Kontext. Weiterhin können für jeden Suchtreffer *Medienausschnitte* (Audio und/oder Video) erstellt werden. Auch hier kann u.a. festgelegt

[12] Die Zeilennummern wurden aus den längeren Originaltranskripten übernommen.

werden, ob bzw. wie viel Kontext einbezogen werden soll, und das Layout der Transkripte kann angepasst werden. Im konkreten Fall von *parce que bon* wurden anhand dieser Ausschnitte die sequenziellen Analysen vorgenommen. Weiterhin ist es möglich, aus *R* die *Suchtreffer in Annotationsprogrammen zu öffnen* (*Praat* und *ELAN*). Dabei werden entweder die originalen Annotationsdateien geöffnet, oder – sollte die Originaldatei in einem anderen Format vorgelegen haben – die Annotationsdateien wird ad hoc im Format des gewählten Annotationsprogramms erstellt. Hier können beispielsweise die Transkriptionen überarbeitet und die Annotationsdateien dann erneut in *R* eingelesen werden. Auf diese Weise können die Transkriptionen in den Annotationsdateien fortlaufend aktualisiert werden. Im Falle von *parce que bon* wurde außerdem in *Praat* die Stärke der Zäsuren beurteilt. Hierzu wurden Schemata erstellt, in denen Veränderungen verschiedener prosodischer Parameter erfasst wurden (vgl. Abbildung 1).

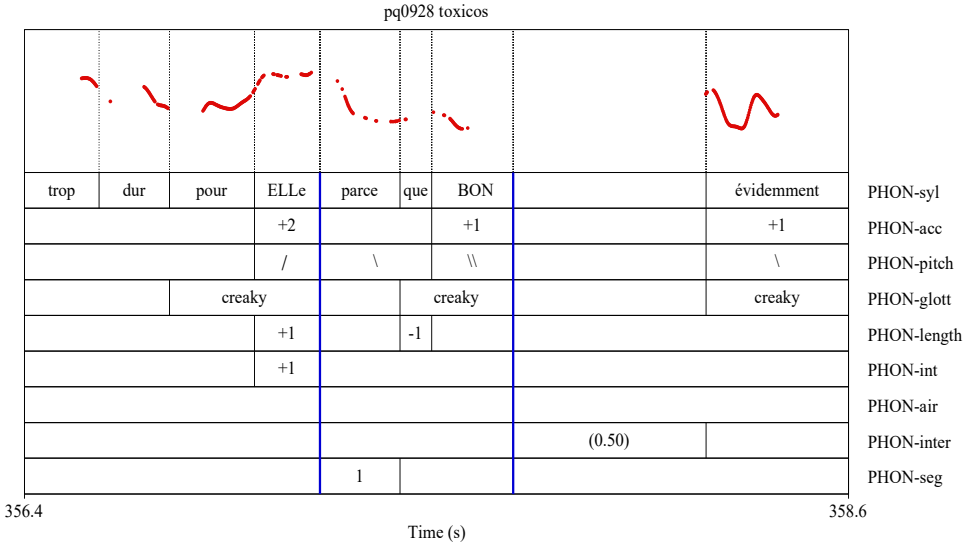


Abbildung 1: Zäsurierungsschema für Beispiel (4) (aus Ehmer 2022:422)

Eine weitere Möglichkeit besteht darin, die Suchtreffer – inklusive der Transkriptausschnitte – in einem *Tabellenformat* zu exportieren. Auf diese Weise können in einem Tabellenkalkulationsprogramm (z.B. Microsoft Excel, OpenOffice Calc) beispielsweise die Suchtreffer anhand verschiedener Kriterien klassifiziert bzw. kodiert werden (Luginbühl et al. 2021). Im konkreten Fall wurden verschiedene funktionale und formale Parameter, wie beispielsweise die Stärke der Zäsuren (vor *parce que*, zwischen *parce que* und *bon* sowie nach *bon*), erfasst. Die kodierten Beispiele bzw. die Analysedaten können dann wieder als Tabellenformat in *R* reimportiert werden. Hier kann anschließend sowohl die statistische Auswertung erfolgen als auch mit den Funktionen der `act`-Bibliothek weitergearbeitet werden.

Neben diesen Funktionen kann mit dem *Aligned Corpus Toolkit* außerdem der *Inhalt von Annotationen bearbeitet werden*, z.B. indem leere Annotationen gelöscht werden, oder es können unerwünschte Zeichen in einem gesamten Korpus ersetzt werden. Weiterhin können unterschiedliche Versionen von Annotationsdateien zusammengeführt werden, etwa wenn diese von verschiedenen Personen gleichzeitig bearbeitet wurden. Weiterhin ist der *Export von Annotationsdateien* in verschiedenen Formaten möglich.

Bevor in Abschnitt (4) die Umsetzung des Fallbeispiels in *R* illustriert wird, erfolgt zunächst in Abschnitt (3) eine detailliertere Vorstellung der verschiedenen Funktionen der act-Bibliothek.

3. Funktionen der act-Bibliothek

Die act-Bibliothek bietet verschiedene Funktionen, um in *R* mit zeitalignierten multimodalen Korpora zu arbeiten. Die wichtigsten Funktionen werden hier zusammenfassend dargestellt.

Import und Export von Annotationsdateien
Unterstützt werden aktuell die Dateiformate .eaf (*ELAN*), .exb (*EXMARaLDA*) und .TextGrid (*Praat*) für den Import und Export. Weiterhin können Untertiteldateien im .srt-Format exportiert werden. Alle Daten können ebenso auf einfache Weise in Tabellen-Formaten exportiert werden (als .csv-Datei mit Trennzeichen oder im .xlsx-Excel-Format). Es können sowohl ausgewählte Annotationsdateien als auch das gesamte Korpus exportiert werden.

Erstellung von Drucktranskripten
Es können Drucktranskripte im Stil der Gesprächsanalyse erstellt werden, wobei das Layout angepasst werden kann. Beispielsweise können die Breite des Transkripts und der Sprechersiglen festgelegt und eckige Klammern zur Markierung von Überlappungen automatisch ausgerichtet werden. Weiterhin können einzelne Annotationsspuren/-zeilen selektiv in das Transkript übernommen und zeitliche Ausschnitte gewählt werden. Drucktranskripte können für einzelne oder alle Annotationsdateien in einem Korpus-Objekt mit nur einem Befehl erstellt werden.

Suche im Korpus
Die Grundfunktion besteht in der Suche nach Zeichenketten und regulären Ausdrücken (*RegEx*)[13] in den Annotationen. Dabei kann flexibel festgelegt werden, welche Transkripte und/oder Annotationsspuren bei der Suche einbezogen oder ausgeschlossen werden sollen. Die Suchtreffer werden in einer Konkordanz-Darstellung (*Keyword in Context*/KWIC) mit der Zeitinformation ausgegeben. Darüber hinaus aber bietet die act-Bibliothek insbesondere drei Suchoptionen, die für die Interaktionale Linguistik von Relevanz sind: eine normalisierte Suche, die Suche über Annotationsgrenzen hinweg und die Suche nach zeitlichen Kookkurrenzen.

- Normalisierte Suche: Durch die Anwendung von Transkriptionskonventionen weichen Transkriptionen häufig von der Standardgraphie ab. Beispielsweise werden gemäß der GAT-Konventionen (Selting et al. 2009) Dehnungen durch die Verwendung von Doppelpunkten markiert (z.B. *a:::ber*) oder eckige Klammern zur Markierung von Überlappungen verwendet (z.B. *a::[:ber] das*). Wenn diese Konventionen innerhalb von Wörtern verwendet werden, dann werden diese Wörter bei einer Suche nach der orthographischen Normalform,

[13] Bei regulären Ausdrücken (engl. *regular expressions*, abgekürzt *RegEx*) handelt es sich um eine standardisierte Abfragesprache zur Erstellung von Suchmustern für Textsuchen. Für eine Einführung in reguläre Ausdrücke siehe z.B. Friedl (2006).

in diesem Fall *aber*, in den meisten Programmen nicht gefunden. Die act-Bibliothek bietet daher eine Funktion, bestimmte konventionsspezifische Zeichen und Zeichenkombinationen herauszufiltern (z.B. Doppelpunkte, eckige Klammern, Notation von Pausen in runden Klammern u.a.). Beim Import von Annotationsdateien werden nicht nur die originalen Inhalte der Annotationen importiert, sondern darüber hinaus eine zweite Version der Daten erstellt, in denen die Annotationskonventionen entfernt sind. Dies erlaubt dem Nutzer, sowohl in den originalen Annotationstexten als auch in der gefilterten Version zu suchen. Die Art und Weise der Filterung kann vom Nutzer angepasst werden; standardmäßig werden die GAT-Konventionen herausgefiltert. Anzumerken ist, dass die act-Bibliothek damit keine Normalisierung im engen Sinne leistet, bei der auch nicht-standard-konforme Token (z.B. "fuffzisch") ihren orthographischen Entsprechungen zugewiesen werden (z.B. "fünfzig").[14] Durch die Filterung der Annotationskonventionen kann jedoch eine weitgehende Normalisierung in Bezug auf die Notationskonventionen erzielt werden.

- Volltextsuche über Annotationsgrenzen hinweg: In zeitalignierten Annotationsprogrammen wird notwendigerweise eine Segmentierung vorgenommen. In vielen Programmen erfolgt die Suche nach Zeichenketten dann innerhalb einzelner Annotationen. Die Suche nach Zeichenketten, die sich über mehrere Annotationen erstrecken, ist dann schwieriger zu realisieren.[15] Die act-Bibliothek bietet die Möglichkeit einer sogenannten Volltextsuche, in der auch nach Zeichenketten gesucht werden kann, die sich über mehrere Annotationen erstrecken. Ein Beispiel hierfür ist die Suche mittels eines regulären Ausdrucks wie `aber.{1,10}naja`, mit dem die Zeichenketten *aber* und *naja* gefunden werden, die in einem Abstand von 1 bis 10 Zeichen aufeinanderfolgen.[16] Über diese Suche werden Abfolgen von *aber* und *naja* unabhängig davon gefunden, ob sie in derselben oder unterschiedlichen Annotationen transkribiert sind. Darüber hinaus kann in Annotationen gesucht werden, die innerhalb einer Annotationsspur aufeinander folgen und/oder sich auf verschiedenen Spuren befinden (also über 'Sprecherwechsel' hinweg). Weiterhin kann die Volltextsuche sowohl im originalen als auch im normalisierten Annotationstext erfolgen.
- Kookkurrenz-Suche: Die Ergebnisse einer ersten Suche können als Ausgangspunkt für eine weitere Suche genommen werden, um zeitliche Kookkurrenzen auf verschiedenen Annotationsspuren zu finden. Ausgehend von den Suchtreffern einer ersten Suche wird auf anderen Annotationsspuren nach Annotationen gesucht, die zeitlich überlappen und einem Suchmuster entsprechen. Sowohl die Annotationsspuren, auf denen nach einer Kookkurrenz gesucht wird, als auch das Suchmuster können angepasst werden.

---

[14] Eine solche Normalisierung bietet beispielsweise *Folker* (Schmidt/Schütte 2010).

[15] Die Suche über Annotationsgrenzen hinweg ist in *EXMARaLDA* problemlos mittels regulärer Ausdrücke möglich, in *ELAN* können Bedingungen für das Aufeinanderfolgen von Annotationen formuliert werden. In *Praat* besteht keine Suchmöglichkeit über Annotationsgrenzen hinweg.

[16] Zur Einführung in reguläre Ausdrücke siehe z.B. Friedl (2006).

Export und Re-Import von Suchergebnissen
Die Suchergebnisse können in verschiedenen Tabellenformaten exportiert und reimportiert werden, als mit Trennzeichen-getrennte Datei im .csv-Format und als Excel-Datei im .xlsx-Format. Dies ermöglich, beispielsweise die in *R* erzielten Suchergebnisse in einem Tabellenkalkulationsprogramm (z.B. Microsoft Excel, OpenOffice Calc) zu klassifizieren und danach mit den kodierten Daten in *R* weiterzuarbeiten, und zwar mit Bezug auf die originalen Annotations- und die verlinkten Mediendateien.

Erstellung von Transkript- und Medienausschnitten
Für jeden Suchtreffer können Drucktranskripte und Medienausschnitte erstellt werden. Dabei ist nicht nur das Layout der Drucktranskripte anpassbar. Vielmehr kann auch ein Zeitintervall gewählt werden, z.B. ein Kontext von 10 Sekunden vor und 20 Sekunden nach dem Suchtreffer. Die Transkripte werden für jeden Suchtreffer einzeln und als Zusammenfassung in .txt Text-Dateien gespeichert. Darüber hinaus können die Transkripte in die Tabelle mit den Suchergebnissen eingefügt werden. Auf diese Weise kann dann im Tabellenkalkulationsprogramm direkt mit dem Transkript-Kontext gearbeitet werden. Für die Erstellung der Medienausschnitte wird das kostenlose Kommandozeilenprogramm *FFmpeg*[17] genutzt, welches u.a. den Schnitt und die Konvertierung von Mediendateien erlaubt. Die Mediendateien werden also nicht direkt mit Funktionen von *R* bearbeitet. Vielmehr werden Schnittbefehle für *FFmpeg* erstellt, die dann von diesem Programm verarbeitet werden. Diese Anweisungen enthalten u.a. Informationen über den Start- und Endzeitpunkt des Ausschnitts, den Pfad der Zieldatei, die Medienqualität etc. Die Schnittbefehle können dann erstens für einzelne Suchtreffer ausgeführt werden. Zweitens werden alle Schnittbefehle in einer Stapelverarbeitungs-Datei zusammengefasst. Wenn diese Stapeldatei ausgeführt wird, werden die Schnittbefehle sukzessive abgearbeitet und Medienausschnitte für alle Suchtreffer erstellt.

Bearbeitung der Annotationsdaten
Die Daten in einem Korpus-Objekt können unter Nutzung verschiedener Funktionen bearbeitet werden, die sowohl auf ausgewählte als auch auf alle Transkripte eines Korpus-Objekts angewendet werden können. Zur Verfügung stehen beispielsweise Funktionen zum Löschen bzw. Ersetzen von Zeichenketten in Annotationen, zum Löschen leerer Annotationen und Annotationen, die einem bestimmten Suchkriterium entsprechen, zum Löschen bestimmter Annotationsspuren, zum Verschieben oder Kopieren von Annotationen, die einem bestimmten Suchkriterium entsprechen, zum Hinzufügen von Annotationsspuren und zur Veränderung ihrer Reihenfolge. Außerdem steht eine Funktion zu Verfügung, mit der unterschiedliche Annotationsdateien ein und derselben Aufnahme zusammengeführt werden können. Dies ist beispielsweise nützlich, wenn mehrere Personen gleichzeitig an unterschiedlichen Stellen einer Aufnahme gearbeitet haben und die Ergebnisse zusammengeführt werden sollen.

[17] *FFmpeg:* https://www.ffmpeg.org

Austausch mit anderen Programmen
Die act-Bibliothek erlaubt die Interaktion mit ausgewählten Programmen. Beispielsweise können einzelne Suchtreffer direkt in *Praat* oder *ELAN* geöffnet werden, wobei entweder die originalen Annotationsdateien geöffnet oder – auf der Grundlage der Daten im Korpus-Objekt – ad hoc neue Annotationsdateien erstellt werden. Unter *macOS* ist es auch möglich, die Suchtreffer direkt in *QuickTime* abzuspielen. Weiterhin können die Daten in das Format der *R*-Bibliothek rPraat (Bořil/Skarnitzl 2016) konvertiert bzw. aus diesem Format gelesen werden.
Die Einzelfunktionen des *Aligned Corpus Toolkit* sind größtenteils auch in anderen Programmen in dieser oder ähnlicher Weise verfügbar bzw. können durch die Kombination verschiedener Programme realisiert werden. Die Vorteile des *Aligned Corpus Toolkit* liegen insbesondere in den folgenden drei Aspekten:

- Zeitalignierte Annotationsdaten in verschiedenen Formaten werden *in der Programmiersprache R unmittelbar verfügbar* gemacht, ohne dass eine vorherige Konvertierung (etwa in das .csv-Format) notwendig wäre. Die act-Bibliothek stellt damit ein Bindeglied zur Realisierung interaktionsanalytischer Auswertungen und Visualisierungen in *R* dar.
- Dabei sind die zentralen Funktionen der act-Bibliothek prinzipiell miteinander kombinierbar und darüber hinaus fein konfigurierbar. Dies erlaubt, *spezifische und potentiell komplexe Problemlösungen* zu entwickeln. Durch die Arbeit in *R* als Programmiersprache können Arbeitsabläufe und Auswertungen automatisiert werden, was in Programmen mit einer grafischen Benutzeroberfläche meist nur in eingeschränktem Umfang möglich ist.
- Die Arbeit in *R* ermöglicht eine *einfache Erweiterbarkeit um neue Funktionen.* Während in den meisten Programmen der Funktionsumfang prinzipiell durch die Programmierung vorgegeben ist, können in *R* eigene Funktionen entwickelt werden. Die Funktionen und Datenstrukturen der act-Bibliothek können dabei eine Grundlage für weitergehende Funktionen darstellen. Beispielsweise ist es möglich, Funktionen für den Import und Export weiterer Annotationsformate (z.B. *Folker*), komplexe Suchen und die Integration von Metadaten zu entwickeln.

Dieser hohen Flexibilität der Arbeit in *R* steht natürlich ein anderes "Nutzererlebnis" als bei einer Desktop-Anwendung gegenüber, die über eine grafische Benutzeroberfläche verfügt. Während Desktop-Anwendungen meist sehr intuitiv bedient werden können, sind für die Nutzung von *R* grundlegende Programmierkenntnisse erforderlich.

4. Umsetzung des Fallbeispiels in *R*

Im Folgenden wird kurz illustriert, wie die in Abschnitt 3 benannten Arbeitsschritte des Fallbeispiels *parce que bon* in *R* umgesetzt werden können. Die Befehle selbst werden an dieser Stelle noch nicht im Detail erläutert, dies geschieht ausführlich in Teil B, wo dann auch mit dem vorhandenen Beispielkorpus gearbeitet wird.

In einem ersten Schritt wird mit dem Befehl `act::corpus_new()` ein neues Korpus-Objekt mit dem Namen `korpus` erstellt. In dieses Objekt werden alle Annotationsdateien importiert, die sich in einem bestimmten Ordner (und enthaltenen Unterordnern) befinden. In diesem Fall ist es der Ordner "/Users/oliverehmer/corpus". Die Pfadangaben werden im Folgenden beispielhaft für *macOS* angegeben.

```
korpus <- act::corpus_new(pathsAnnotationFiles = "/Users/oliverehmer/corpus")
```

Im Korpus-Objekt ist dann der Inhalt der einzelnen Annotationsdateien verfügbar. Mit dem Befehl `act::search_new()` kann innerhalb der Transkripte des Korpus-Objekts gesucht werden. Durch den folgenden Befehl wird mit dem regulären Ausdruck `parce que.{1,5}bon` nach Zeichenketten gesucht, in denen *bon* im Abstand von 1 bis 5 Zeichen auf *parce que* folgt.

```
suche <- act::search_new(x       = korpus,
                         pattern = 'parce que.{1,5}bon',
                         name    = "parce_que_bon")
```

Die Suchergebnisse werden im Objekt `suche` gespeichert, mit welchem dann weitergearbeitet wird. Mit dem folgenden Befehl `act::search_cuts()` werden Ausschnitte der Suchergebnisse erstellt, d.h. Drucktranskripte, Untertiteldateien und Schnittlisten zur Erstellung der Medienausschnitte mit *FFmpeg*. Diese Dateien werden unter dem angegebenen Pfad in einem neuen Ordner gespeichert, der den Namen der Suche trägt.

```
act::search_cuts(x              = korpus,
                 s              = suche,
                 outputFolder   = "/Users/oliverehmer/")
```

Die folgende Variante des Funktionsaufrufs illustriert, wie bei der Erstellung der Ausschnitte die Breite des Kontexts in Sekunden angegeben werden kann.

```
act::search_cuts(x              = korpus,
                 s              = suche,
                 cutSpanBeforesec = 20,
                 cutSpanAftersec  = 10,
                 outputFolder   = "/Users/oliverehmer/")
```

Neben den Ausschnitten werden im angegebenen Ordner auch die Suchergebnisse in verschiedenen Tabellen-Formaten (.csv und .xlsx) gespeichert. In Tabellenkalkulationsprogrammen (z.B. Microsoft Excel, Open Office Calc) können diese Dateien dann bearbeitet, d.h. die einzelnen Suchtreffer hinsichtlich formaler und/oder funktionaler Parameter kodiert werden. Anschließend können die Dateien mit dem folgenden Befehl wieder importiert werden.

```
suche2 <- act::search_results_import( path =
       "/Users/oliverehmer/parce_que_bon/searchResults_parce_que_bon.xlsx")
```

Die reimportierten Suchergebnisse befinden sich nun im Suche-Objekt `suche2`. Mit dem folgenden Befehl wird das erste Ergebnis dieser Suche in *Praat* geöffnet (vgl. Abbildung 2)

```
act::search_openresult_inpraat(x        = korpus,
                               s        = suche2,
                               resultNr = 1)
```

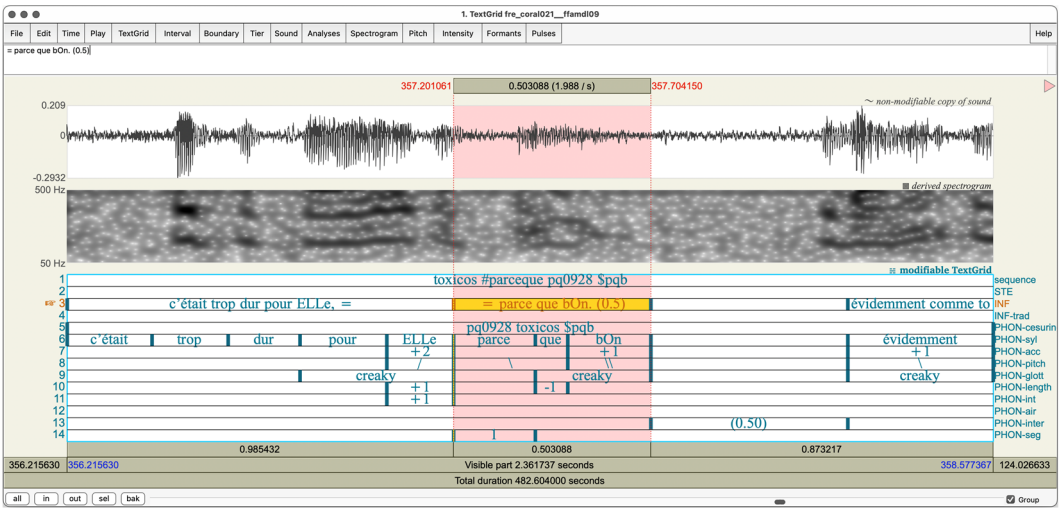


Abbildung 2: Beispiel (4) in *Praat* (Cresti/Moneglia 2005: Aufnahme ffamdl09)

TEIL B: PRAKTISCHE EINFÜHRUNG IN DAS *ALIGNED CORPUS TOOLKIT*

Die folgende praktische Einführung richtet sich insbesondere an Leser_innen mit keinen oder geringen Erfahrungen in *R*. Zunächst werden einige vorbereitende Schritte vollzogen (Abschnitt 5). Die Vorstellung der zentralen Funktionen der act-Bibliothek erfolgt dann in Abschnitt 6. Für erfahrene Nutzer von *R* und als Ergänzung steht eine umfangreiche englischsprachige Dokumentation der act-Bibliothek mit weiteren Code-Beispielen zur Verfügung. Diese Dokumentation ist sowohl als Referenz-Handbuch im .pdf-Format auf CRAN als auch über die Hilfefunktion in *R* verfügbar.

5. Vorbereitende Schritte

Im Folgenden wird zunächst die Programmoberfläche von *RStudio* vorgestellt und erläutert, wie in *R* Befehle ausgeführt werden können (5.1). Danach wird dargestellt, wie die Skriptdatei mit den Code-Beispielen zu diesem Artikel geöffnet werden kann und ggf. Probleme mit der Darstellung von Sonderzeichen behoben werden können (5.2). Anschließend wird erläutert, wie die act-Bibliothek installiert wird (5.3). Der letzte vorbereitende Schritt besteht darin, ein Beispielkorpus herunterzuladen (5.4), mit welchem anschließend gearbeitet wird.

5.1. *R*, *RStudio* und das Ausführen von Befehlen

Zur komfortablen Arbeit mit *R* kann die frei verfügbare Programmierumgebung *RStudio* genutzt werden. Die Links zum Download von *R* und *RStudio* finden sich in Abschnitt 8. *RStudio* bietet u.a. die Möglichkeit, Daten in einer grafischen Tabellendarstellung anzuzeigen und Schaubilder darzustellen (vgl. Abbildung 3).

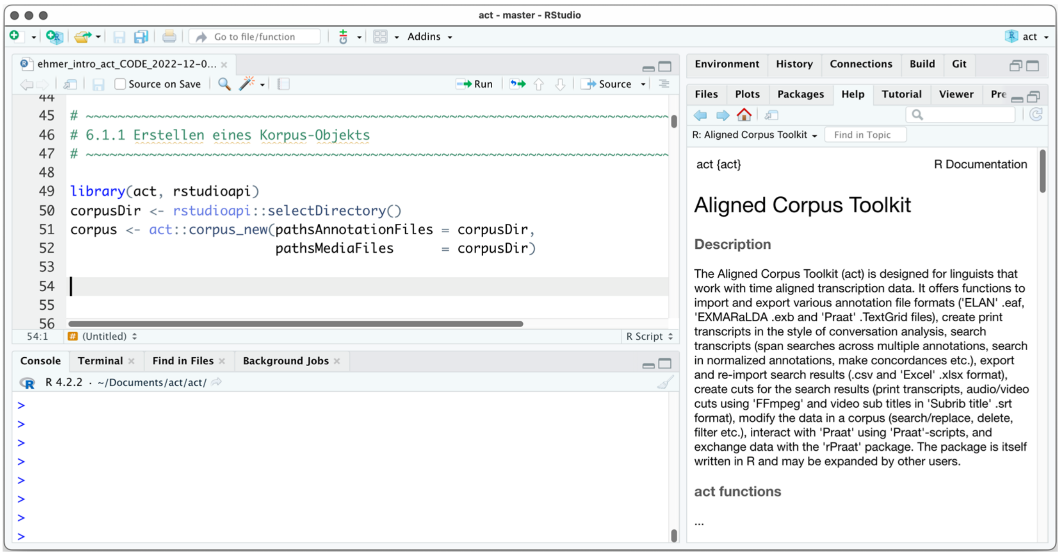


Abbildung 3: Die Benutzeroberfläche von *RStudio*

Das zentrale Fenster ist das Code-Fenster im linken oberen Panel. Hier werden die einzelnen Befehle in Form eines Texteditors eingegeben. Das *R*-Skript "ehmer_act_CODE.R" mit dem Beispiel-Code kann über den Menüpunkt "File > Open File..." im Code-Fenster geöffnet werden. Die einzelnen Befehle eines R-Skriptes können ausgeführt werden, indem der Cursor in der entsprechenden Zeile platziert und die Tastenkombination *CMD + Enter* (auf *macOS*) oder *STRG + Enter* (auf *Windows*) gedrückt wird. Zum Ausführen mehrerer Befehle wird die Tastenkombination mehrfach hintereinander gedrückt. Alternativ kann ein Code-Bereich markiert und dann die entsprechende Tastenkombination gedrückt werden.

Im Panel rechts unten ist über den Reiter "Help" eine Hilfefunktion verfügbar. Hier kann für jede installierte Bibliothek eine Dokumentation mit Code-Beispielen eingesehen werden. Über Klicken kann in der Hilfe navigiert werden. Die Hilfe zu einer bestimmten Bibliothek kann z.B. aufgerufen werden, indem im rechten Panel der Reiter "Packages" ausgewählt und dann auf den Namen der Bibliothek geklickt wird.

5.2. Korrektes Öffnen der *R*-Datei mit den Code-Beispielen

Die Code-Beispiele dieses Artikels sind in der Datei "ehmer_act_CODE.R" enthalten. Diese kann in *RStudio* über den Menüpunkt *File > Open File* geöffnet werden. Möglicherweise werden beim Öffnen die Sonderzeichen nicht korrekt dargestellt, wenn die Enkodierung der Datei als *UTF-8* nicht automatisch korrekt erkannt

wurde. In diesem Fall muss der Menüpunkt *File* > *Reopen with Encoding...* und im nachfolgenden Dialog *UTF-8* gewählt werden.[18]

5.3. Installation und Laden der act-Bibliothek

Mit den folgenden Befehlen wird die act-Bibliothek installiert und aktiviert. Die Befehle sind ebenfalls in der Datei mit den Code-Beispielen.

```
install.packages("act")
library("act")
```

Die Installation von Bibliotheken erfolgt in *R* über den Befehl `install.packages()`. In den runden Klammern wird in Anführungszeichen der Name der zu installierenden Bibliothek als Parameter angegeben, in diesem Fall "act". Beim Ausführen des Befehls wird eventuell gefragt, ob weitere Bibliotheken installiert werden sollen. Dies muss tatsächlich erfolgen, da die act-Bibliothek Befehle aus diesen Bibliotheken benutzt. Nach einmal erfolgter Installation wird die Bibliothek über den Befehl `library()` in den Arbeitsspeicher geladen und damit aktiviert. Während die Installation nur einmal notwendig ist, muss die Bibliothek jedes Mal geladen werden, bevor ihre Funktionen benutzt werden können.

Für die einfache Auswahl von Ordnern auf dem Computer verwenden wir in *RStudio* im Folgenden weiterhin die Bibliothek rstudioapi, die ebenfalls installiert und geladen werden muss.

```
install.packages("rstudioapi")
library(rstudioapi)
```

5.4. Herunterladen des Beispielkorpus

Mit den folgenden Befehlen wird das Beispielkorpus als .zip-Datei aus dem Internet heruntergeladen und entpackt.[19] Die Beispiele sind auf Spanisch und stammen aus den Publikationen Ehmer et al. (2019) und (Ehmer 2021b).

```
workDir <- rstudioapi::selectDirectory()
temp <- tempfile()
download.file("https://github.com/oliverehmer/act_examplecorpus/archive/master.zip", temp)
unzip(zipfile = temp, exdir = dirname(workDir))
```

Die erste Code-Zeile öffnet einen Dialog, über den ein bestehender Ordner ausgewählt werden muss, z.B. der Schreibtisch/Desktop, in welchen das Beispielkorpus

---

[18] Während unter *macOS* UTF-8 als Standard voreingestellt ist, wird unter *Windows* als Standard *ISO-8859-1 (Latin-1)* verwendet. Um generell Probleme beim Austausch zu vermeiden, empfiehlt es sich, auch unter *Windows* den Standard für die Enkodierung auf *UTF-8* zu setzen. Dies ist möglich über den Menüpunkt *Tools* > *Global Options*. Im Fenster, das sich dort öffnet, kann unter *Code* > *Saving* > *Default Text Encoding* als Standard Enkodierung *UTF-8* gewählt werden.

[19] Sollte der Download fehlschlagen, können die Dateien auch direkt von *GitHub* unter dem folgenden Link heruntergeladen werden: https://github.com/oliverehmer/act_examplecorpus .

gespeichert werden soll. Der Pfad zu diesem Ordner wird in der Variable `workDir` gespeichert. Die nächsten drei Zeilen laden eine .zip-Datei von der angegebenen Internetadresse und entpacken diese. Nach dem Ausführen der Befehle befindet sich im zuvor gewählten Ordner ein neuer Ordner mit dem Namen "act_example-corpus-main". Dieser enthält drei Unterordner, in denen jeweils verschiedene Annotationsdateien (in *Praat* .TextGrid und *ELAN* .eaf-Format) und Mediendateien (in den Formaten .wav und .mp4) liegen. Zusammengehörige Dateien tragen denselben Namen und unterscheiden sich lediglich in der Dateiendung (vgl. Abbildung 4).

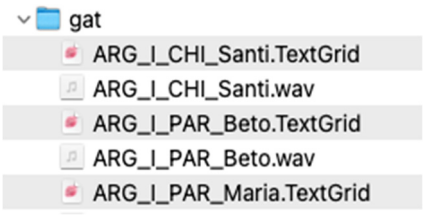


Abbildung 4: Benennung der Beispieldateien

6. Darstellung der zentralen Funktionen

Im Folgenden werden die zentralen Funktionen der `act`-Bibliothek beschrieben. Zunächst wird erläutert, wie Annotationsdateien in ein Korpus-Objekt geladen werden können (6.1). Danach wird dargestellt, wie die importierten Daten in verschiedene Annotationsdateiformate exportiert (6.2), Drucktranskripte erstellt (6.3), im Korpus-Objekt gesucht (6.4) und mit den Suchergebnissen gearbeitet (6.5) werden kann. Anschließend wird auf den Austausch mit der `rPraat`-Bibliothek eingegangen (6.6) und dargestellt, wie Filter erstellt werden können (6.7), um lediglich mit ausgewählten Transkript-Objekten/Annotationsspuren zu arbeiten.

Die Funktionen der `act`-Bibliothek werden anhand von Code-Beispielen präsentiert. Alle Code-Beispiele sind gesammelt in einer separaten *R*-Datei verfügbar (Datei "ehmer_act_CODE.R" als Zusatzmaterial zu diesem Artikel, vgl. auch 5.2). Die Code-Beispiele illustrieren jeweils nur einfache Aufrufe der Befehle. Welche weiteren Parameter die Befehle bieten, kann über die Dokumentation der `act`-Bibliothek eingesehen werden (über das Handbuch oder auch die Hilfefunktion in *RStudio*), wo dann auch weitere Beispiele gegeben werden.

6.1. Das Korpus-Objekt und der Import von Annotationsdateien

Der erste Schritt zur Arbeit mit der `act`-Bibliothek besteht darin, Annotationsdateien in ein Korpus-Objekt zu importieren, was in 6.1.1 beschrieben wird. Nach einer Darstellung der internen Struktur des Korpus-Objekts (6.1.2) wird illustriert, wie direkt auf die Daten zugegriffen werden kann (6.1.3) und wie Zusammenfassungen der Daten ausgegeben und visualisiert werden können (6.1.4). Abschließend wird erläutert, wie neue Verknüpfungen zu Mediendateien erstellt werden (6.1.5).

6.1.1. Erstellen eines Korpus-Objekts und Importieren von Annotationsdateien

Um alle Annotations-Dateien aus einem Ordner in *R* einzulesen, kann der Befehl `act::corpus_new()` genutzt werden. Beim Aufruf des Befehls wird der Ordner angegeben, in dem die Annotationsdateien liegen.

```
library(act, rstudioapi)
corpusDir <- rstudioapi::selectDirectory()

corpus <- act::corpus_new(   pathsAnnotationFiles  = corpusDir,
                             pathsMediaFiles       = corpusDir)
```

Mit der ersten Code-Zeile werden zunächst mit dem Befehl `library()` die benötigten Bibliotheken act und rstudioapi aktiviert, sollte dies nicht bereits zuvor geschehen sein. Mit der zweiten Zeile wird ein Dialogfenster geöffnet, in dem der Ordner ausgewählt werden muss, in dem die Annotationsdateien gespeichert sind. Hier muss also der zuvor heruntergeladene Ordner "act_examplecorpus-main" ausgewählt werden. Der Pfad zu diesem Ordner wird dann in der Variable `corpusDir` gespeichert.[20] In der dritten Zeile wird beim Befehl `act::corpus_new()` im Parameter `pathsAnnotationFiles` der Ordner angegeben, in dem die Annotationsdateien liegen.[21] Der Ort ist in unserem Fall in der Variable `corpusDir` gespeichert. Im Parameter `pathsMediaFiles` kann optional angegeben werden, wo die Mediendateien zu den Annotationsdateien gespeichert sind (siehe hierzu genauer 6.1.5). In unserem Fall ist das ebenfalls der zuvor gewählte Ordner, dessen Pfad in der Variable `corpusDir` gespeichert ist. Wird der Befehl `act::corpus_new()` ausgeführt, dann wird der gewählte Ordner nach Annotationsdateien durchsucht, wobei standardmäßig auch Unterordner einbezogen werden. Die Annotationsdateien werden dann importiert und ein Korpus-Objekt erstellt. Das Korpus-Objekt wird in der Variable `corpus` gespeichert.

6.1.2. Die Struktur des Korpus-Objekts

Das Korpus-Objekt ist hierarchisch strukturiert. Zentral ist dabei, dass der Inhalt jeder Annotations-Datei in Form eines Transkript-Objekts importiert wird. Die einzelnen Transkript-Objekte sind in Form einer Liste innerhalb des Korpus-Objekts gespeichert. Jedes Transkript-Objekt wiederum enthält eine Tabelle `annotations`, in welcher alle Annotationen der originalen Annotationsdatei importiert wurden, und eine Tabelle `tiers` mit den Informationen über die Annotationsspuren. Die grundlegende Struktur eines Korpus-Objekts ist in der folgenden Abbildung 5 dargestellt.

---

[20] Alternativ kann hier auch ein anderer Ordner ausgewählt werden, in dem andere Annotationsdateien liegen, z.B. das *EXMARaLDA*-Demokorpus, das hier verfügbar ist: https://exmaralda.org/de/exmaralda-demokorpus/.
Zu beachten ist bei diesem Korpus, dass dieselben Daten mehrfach in verschiedenen Annotationsdateiformaten mit ansonsten identischem Dateinamen gespeichert sind, von denen standardmäßig nur eine Version geladen wird.

[21] Im Parameter `pathsAnnotationFiles` können über den Befehl `c()` statt nur einem auch mehrere Pfade zu Ordnern oder auch einzelnen Dateien angegeben werden. Dies gilt auch für den Parameter `pathsMediaFiles`.

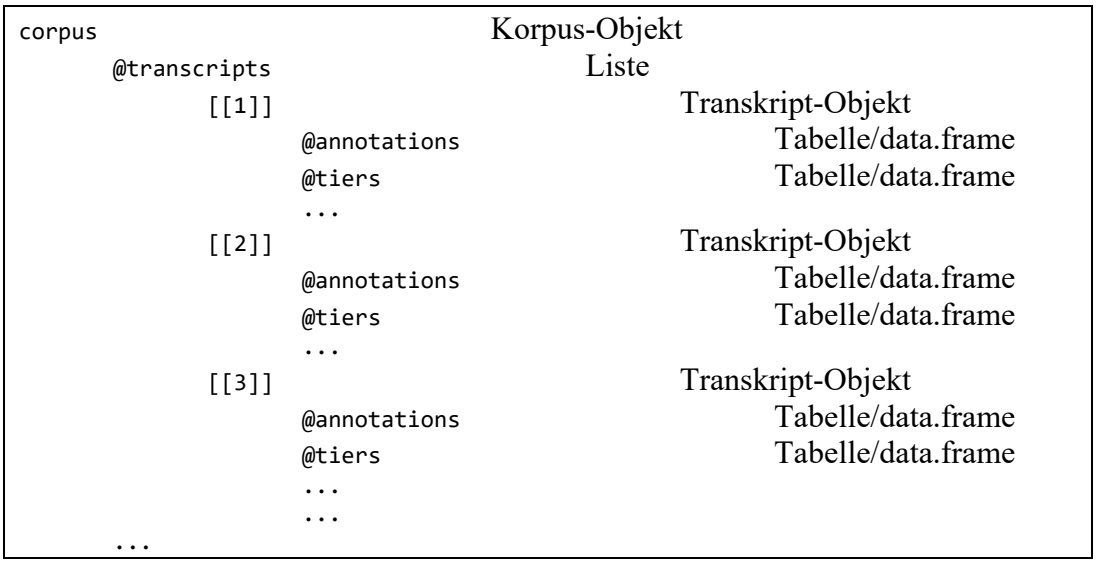


Abbildung 5: Allgemeine Struktur des Korpus-Objekts

Eine detaillierte Dokumentation des Korpus-Objekts kann über `?act::`corpus-class`` aufgerufen werden, eine Dokumentation des Transkript-Objekts über `?act::`corpus-class`.`

Für die Erstellung eines Korpus-Objekts ist wichtig, dass die einzelnen Annotationsdateien jeweils eindeutige Namen haben, denn die act-Bibliothek nutzt intern die Dateinamen als Identifizierer der Transkript-Objekte. Wenn mehrere Annotationsdateien mit demselben Namen gefunden werden – z.B. weil diese in verschiedenen Dateiformaten vorliegen oder in Unterordnern gespeichert sind –, wird im Normalfall nur eine dieser gleichnamigen Annotationsdateien geladen.[22] Dieses Verhalten kann über einen Parameter beim Aufruf von `act::corpus_new()` verändert werden.

6.1.3. Informationen über das Korpus-Objekt und Zugriff auf die Daten

Über die folgenden Befehle können grundlegende Informationen über das Korpus-Objekt abgerufen werden:

```
corpus
corpus@transcripts
corpus@transcripts[[1]]
View(corpus@transcripts[[1]]@annotations)
View(corpus@transcripts[[1]]@tiers)
View(corpus@transcripts[['ARG_I_CHI_Santi']]@tiers)
```

[22] Die Präferenz ist dabei alphabetisch, d.h. es werden die Dateien bevorzugt, die in Unterordnern liegen, deren Anfangsbuchstaben im Alphabet zuerst kommen. Bei identischen Dateinamen richtet sich die Präferenz alphabetisch nach den Dateiendungen, d.h. .eaf-Dateien (*ELAN*) werden über .exb-Dateien (*EXMARaLDA*) und diese wiederum über .TextGrid-Dateien (*Praat*) bevorzugt.

Der Aufruf des Korpus-Objekts `corpus` in der ersten Zeile gibt grundlegende Informationen über dessen Inhalt. Diese Informationen werden in *RStudio* im Konsolenfenster ausgegeben (Abbildung 6). Mit dem Befehl `corpus@transcripts` wird eine Liste aller Transkript-Objekte im Korpus angezeigt. Der Operator `@` bedeutet dabei, dass im Objekt `corpus` die Eigenschaft bzw. der 'slot' `@transcripts` abgerufen wird. Auf die einzelnen Elemente der Liste mit Transkripten kann über ein nachgestelltes `[[ ]]` zugegriffen werden. Mit `corpus@transcripts[[1]]` wird also auf das erste Transkript-Objekt im Korpus `corpus` zugegriffen. Ohne weitere Angaben werden mit dem Befehl `corpus@transcripts[[1]]` im Konsolenfenster grundlegende Informationen über dieses Transkript angezeigt (vgl. Abbildung 6).

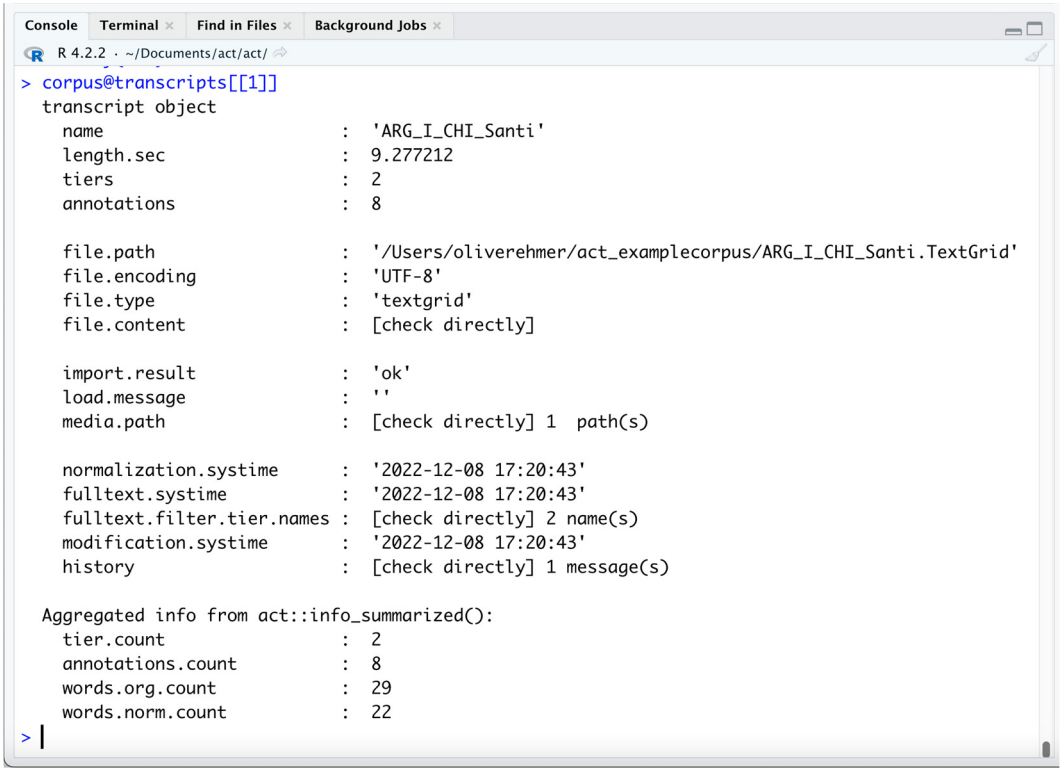
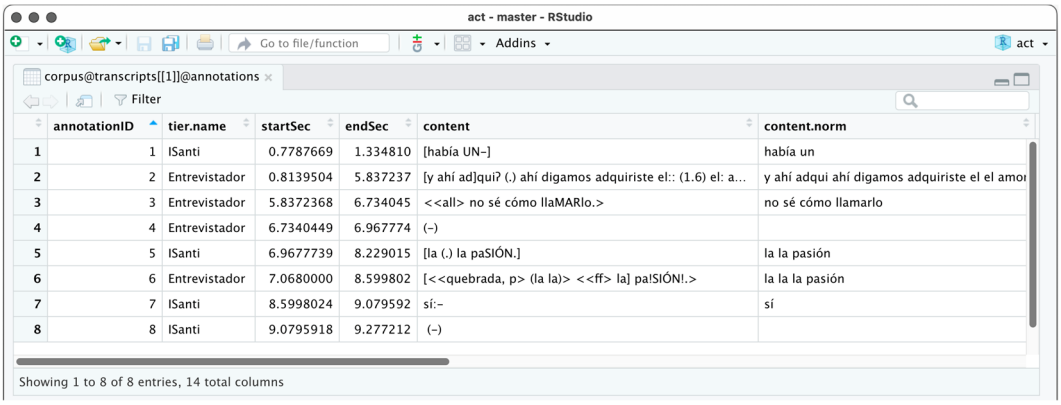


Abbildung 6: Grundlegende Informationen über `corpus@transcripts[[1]]`

Auf die einzelnen Eigenschaften des Transkript-Objekts kann mit dem Operator `@` zugegriffen werden. Mit dem Aufruf `corpus@transcripts[[1]]@annotations` beispielsweise wird auf die Annotationen des ersten Transkript-Objekts im Korpus-Objekt `corpus` zugegriffen. Die Annotationen sind in Form einer Tabelle bzw. als `data.frame` gespeichert. In *RStudio* können diese z.B. mit Befehl `View()` in grafischer Form angezeigt werden.

Der Befehl `View(corpus@transcripts[[1]]@annotations)` zeigt also alle Annotationen aus Transkript 1 an (vgl. Abbildung 7).



	annotationID	tier.name	startSec	endSec	content	content.norm
1	1	ISanti	0.7787669	1.334810	[había UN–]	había un
2	2	Entrevistador	0.8139504	5.837237	[y ahí ad]qui? (.) ahí digamos adquiriste el:: (1.6) el: a…	y ahí adqui ahí digamos adquiriste el el amor
3	3	Entrevistador	5.8372368	6.734045	<<all> no sé cómo llaMARlo.>	no sé cómo llamarlo
4	4	Entrevistador	6.7340449	6.967774	(–)	
5	5	ISanti	6.9677739	8.229015	[la (.) la paSIÓN.]	la la pasión
6	6	Entrevistador	7.0680000	8.599802	[<<quebrada, p> (la la)> <<ff> la] pa!SIÓN!.>	la la la pasión
7	7	ISanti	8.5998024	9.079592	sí:–	sí
8	8	ISanti	9.0795918	9.277212	(–)	

Abbildung 7: Inhalt der Annotationen in `corpus@transcripts[[1]]@annotations`

Analog kann die Tabelle mit den Informationen über die Annotationsspuren in einem Transkript-Objekt über `@tiers` abgerufen werden. Neben der Möglichkeit, auf Transkript-Objekte über ihre Nummer in der Liste zuzugreifen, ist dies auch über die Angabe ihres Namens in eckigen Klammern möglich.

6.1.4. Zusammenfassungen des Korpus-Objekts

Neben dem direkten Zugriff auf Transkript-Objekte und ihren Inhalt, besteht auch die Möglichkeit, eine Zusammenfassung der Informationen in einem Korpus-Objekt zu erhalten.

```
View(act::annotations_all(corpus)
View(act::tiers_all(corpus))

View(act::info(corpus)$transcripts)
View(act::info(corpus)$tiers)
```

Mit den Befehlen `act::annotations_all()` und `act::tiers_all()` werden die Annotationen bzw. Informationen aus allen Transkript-Objekten in jeweils einer Tabelle zusammengefasst. Über den Befehl `act::info()` werden Informationen über die Transkripte und die Annotationsspuren aggregiert. Als Parameter wird das Korpus-Objekt angegeben.

Es ist nun relativ einfach möglich, beispielsweise ein Schaubild zu erstellen, in dem dargestellt ist, wie viele Wörter in den einzelnen Transkripten enthalten sind (vgl. Abbildung 8).

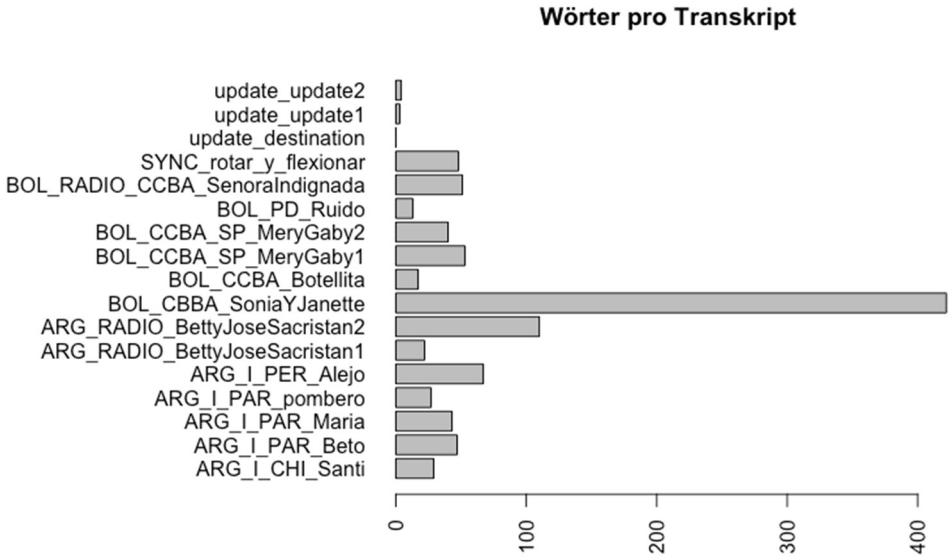


Abbildung 8: Anzahl Wörter pro Transkript im Beispielkorpus

Die folgenden Befehle erstellen dieses Diagramm:

```
info <- act::info(corpus)
par(mar = c(4,20,4,4))
barplot(      height    = info$transcripts$words.org.count,
              names.arg = info$transcripts$transcript.name,
              main      = "Wörter pro Transkript",
              horiz     = TRUE,
              las       = 2)
```

Der erste Befehl `act::info()` speichert die aggregierten Informationen über das Objekt `corpus` in der Variable `info`. Der Befehl `par()` in der zweiten Zeile passt die Darstellungsparameter des Schaubilds so an, dass die langen Namen der Transkripte ausreichend Platz haben.[23] Der Befehl `barplot()` erstellt das horizontale Balkendiagramm. Die Häufigkeitsdaten werden dabei aus der Spalte `$words.org.count` in der Tabelle `transcripts` genommen, die sich im Objekt `info` befindet. Diese werden abgerufen über `info$transcripts$words.org.count`. Analog werden als Beschriftung die Namen der Transkripte genommen, die sich in der Spalte `info$transcripts$words.org.count` befinden.

Ein weiteres Beispiel ist die Anzeige der jeweiligen Dauer der Annotationen als Histogramm (vgl. Abbildung 9). Es soll also dargestellt werden, wie häufig die Annotationen mit einer bestimmten Dauer sind. Auf der x-Achse wird dabei die Dauer der Annotationen, auf der y-Achse die Anzahl der Annotationen im Korpus mit dieser Dauer angegeben.

[23] Dabei steht der Parameter `mar` für die Angabe des Randes (engl. *margin*) im Schaubild. Dessen Breite wird in Buchstaben/Zeilen angeben. Die Werte beginnen am unteren Rand des Schaubildes, im Uhrzeigersinn folgen die drei weiteren Ränder. Die Angabe `mar = c(4,20,4,4)` steht also dafür, dass alle Ränder 4 Zeilen umfassen, bis auf den linken Rand, der 20 Buchstaben breit ist.

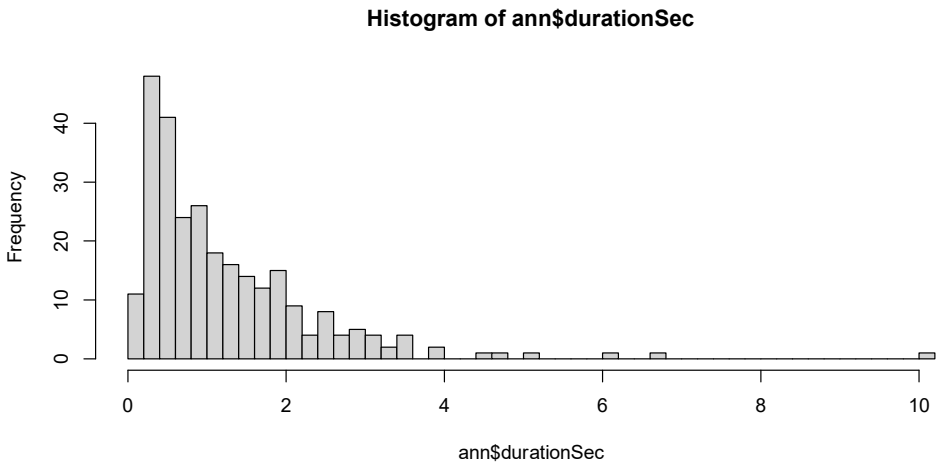


Abbildung 9: Häufigkeitsverteilung von Annotationen mit einer bestimmen Dauer

Die folgenden Befehle erstellen dieses Schaubild.

```
ann <- act::annotations_all(corpus)
ann$durationSec <- ann$endSec - ann$startSec
hist(ann$durationSec, breaks = 50)
```

Zunächst werden mit `act::annotations_all()` alle Annotationen zusammengefasst und dem Objekt `ann` zugewiesen. Dann wird anhand der Anfangs- und der Endzeitmarkierung die jeweilige Dauer der Annotationen errechnet. Der Befehl `hist()` plottet das Histogramm dieser Daten. Dabei werden die Annotationsdauern in Gruppen zusammengefasst und die Anzahl der Annotationen pro Gruppe berechnet. Der Parameter `breaks=50` gibt an, dass die Werte 50-mal unterteilt werden sollen. Es ergeben sich also 51 Gruppen bzw. Säulen im Histogramm.

6.1.5. Verknüpfen von Mediendateien

Die act-Bibliothek bietet verschiedene Möglichkeiten, außer mit den Annotationen eines Korpus auch mit den zugehörigen Mediendateien zu arbeiten. Beispielsweise können Ausschnitte für Suchergebnisse erstellt (vgl. 6.5.1.2) oder die Suchtreffer direkt in verschiedenen Programmen abgespielt werden (vgl. 6.5.4). Eine notwendige Voraussetzung hierfür ist, dass für jedes Transkript-Objekt angegeben ist, wo die zugehörigen Mediendateien gespeichert sind. Diese Mediendateien werden *nicht* direkt in *R* eingelesen, sondern lediglich – über die Angabe der Dateipfade – verknüpft. Diese Verknüpfungen bzw. Pfade sind in jedem Transkript-Objekt in der Eigenschaft `@media.path` gespeichert.

Relevant ist nun, dass die unterstützten Annotationsdateiformate bzw. Programme Verknüpfungen zu Mediendateien unterschiedlich behandeln. Im Falle von *ELAN* .eaf- und *EXMARaLDA* .exb-Dateien werden die Pfade zu den Mediendateien direkt in den Annotationsdateien gespeichert. In *Praat* .TextGrid-Dateien ist hingegen nicht gespeichert, auf welche Mediendateien sich diese beziehen. Damit können gespeicherte Verknüpfungen zu Mediendateien aus *ELAN* .eaf- und *EXMARaLDA* .exb-Dateien von der act-Bibliothek beim Import direkt eingelesen werden.

Im Falle von *Praat* .TextGrid-Dateien ist dies hingegen nicht möglich, da die Information nicht vorhanden ist.

Die act-Bibliothek stellt nun eine Möglichkeit zur Verfügung, Verknüpfungen zu Mediendateien automatisch herzustellen. Die Voraussetzung hierfür ist, dass die Annotationsdateien und die zugehörigen Mediendateien denselben Namen tragen und sich lediglich im Dateisuffix unterscheiden. Dies ist im Beispielkorpus der Fall (vgl. Abbildung 4). Die Annotationsdateien und die Mediendateien müssen dabei nicht im selben Ordner liegen, sondern können über verschiedene Ordner verteilt sein.

Die einfachste Möglichkeit zur automatischen Verlinkung von Mediendateien besteht darin, direkt beim Import der Annotationsdateien anzugeben, in welchem Ordner die Mediendateien gespeichert sind. Hierzu wird beim Aufruf des Befehls `act::corpus_new()` im Parameter `pathsMediaFiles` der Ort angegeben. Dies war bereits bei dem in 6.1.1 gegebenen Code-Beispiel der Fall, welches hier wiederholt wird.

```
library(act, rstudioapi)
corpusDir <- rstudioapi::selectDirectory()
corpus <- act::corpus_new(   pathsAnnotationFiles  = corpusDir,
                             pathsMediaFiles       = corpusDir)
```

Ist der Parameter `pathsMediaFiles` angegeben, werden die angegebenen Ordner nach Mediendateien durchsucht.[24] Voreingestellt ist, dass Unterordner ebenfalls durchsucht werden. Wenn eine Passung der Dateinamen zwischen Annotationsdatei und Mediendatei(en) vorliegt, werden die Mediendateien verlinkt. Zu beachten ist die Voreinstellung, dass bestehende Verknüpfungen zu Mediendateien entfernt werden, wenn diese Dateien nicht gefunden werden. Auf diese Weise sind im Korpus-Objekt nur Mediendateien verknüpft, die aktuell auch verfügbar sind.

Mediendateien können nicht nur beim Erstellen eines Korpus-Objekts verknüpft werden, sondern auch nachträglich. Hierzu wird der Befehl `act::media_assign()` verwendet. Dies illustriert das folgende Code-Beispiel.

```
corpus <- act::corpus_new(pathsAnnotationFiles = corpusDir, assignMedia = FALSE)
corpus@transcripts[['SYNC_rotar_y_flexionar']]@media.path

corpus@paths.media.files <- corpusDir
corpus <- act::media_assign(corpus)
corpus@transcripts[['SYNC_rotar_y_flexionar']]@media.path
```

Mit der ersten Code-Zeile werden die Annotationsdateien aus dem in `corpusDir` angegebenen Ordner importiert. Der Parameter `assignMedia = FALSE` gibt dabei an, dass zunächst *nicht* nach Mediendateien gesucht werden soll. Den Transkripten im

[24] Standardmäßig werden die Videoformate .mp4 und .mov sowie die Audioformate .wav, .aif, .aiff und .mp3 erkannt. Die Liste der erkannten Datei-Suffixe lässt sich über die Optionen `act.fileformats.video` und `act.fileformats.audio` anpassen. Das Vorgehen zur Änderung der Optionen ist in der Dokumentation beschrieben, welche mit dem Befehl `help(options_ show, package="act")` aufgerufen werden kann. Dort sind auch alle anderen Optionen der Bibliothek beschrieben. Die aktuell voreingestellten Werte lassen sich über den Befehl `act::options_show()` anzeigen.

Korpus-Objekt werden also keine Medienlinks hinzugefügt. Dies kann für das Transkript-Objekt `'SYNC_rotar_y_flexionar'` mit der zweiten Codezeile abgefragt werden. Das Ergebnis `character(0)` beim Ausführen des Befehls bedeutet, dass keine Einträge vorhanden sind.

Im Folgenden werden dann für alle Transkript-Objekte die Mediendateien nachträglich verlinkt. Zunächst wird für das Korpus-Objekt festgelegt, in welchem Ordner die zugehörigen Mediendateien gespeichert sind. Dies geschieht über die Zuweisung des in `corpusDir` enthaltenen Pfades zur Eigenschaft `@paths.media.files` des Korpus-Objekts `corpus`. Mit `act::media_assign()` werden dann Mediendateien gesucht und verlinkt und mit der letzten Code-Zeile die Pfade zu den verlinkten Dateien ausgegeben. Im Beispiel sind das eine .mp4- und eine .wav-Datei.

6.2. Export von Transkript-Objekten

Die Transkript-Objekte in einem Korpus-Objekt können in den Annotationsformaten .eaf (*ELAN*), .exb (*EXMARaLDA*) und .TextGrid (*Praat*), als .srt-Untertitel (*SubRib*-Format) sowie als Drucktranskripte im .txt-Format exportiert werden. Es ist möglich, alle oder nur ausgewählte Transkript-Objekte zu exportieren sowie das Ausgabeformat zu wählen. Die folgenden Befehle illustrieren vier unterschiedliche Möglichkeiten.

```
exportDir <- tools::file_path_as_absolute(rstudioapi::selectDirectory())

act::corpus_export(   x                   = corpus,
                      outputFolder        = file.path(exportDir,
                                            "act_export_1"))

act::corpus_export(   x                   = corpus,
                      outputFolder        = file.path(exportDir,
                                            "act_export_2"),
                      formats             = "eaf")

act::corpus_export(   x                   = corpus,
                      outputFolder        = file.path(exportDir,
                                            "act_export_3"),
                      filterTranscriptNames = c('ARG_I_CHI_Santi',
                                            'ARG_I_PAR_Beto'),
                      formats             = "printtranscript")

act::corpus_export(   x                   = corpus,
                      outputFolder        = file.path(exportDir,
                                            "act_export_4"),
                      filterTranscriptNames = 'BOL_CBBA_SoniaYJanette',
                      filterTierNames     = c('ROB', 'FRA'),
                      formats             = c("printtranscript", 'textgrid'))
```

Mit der ersten Code-Zeile wird ein Dialog zur Wahl eines Ordners geöffnet. Hier muss ein bereits bestehender Ordner gewählt werden, in welchen der Export

erfolgen soll, z.B. der Schreibtisch/Desktop. In den nun folgenden vier Varianten des Befehls `act::corpus_export()` wird in diesem Ordner jeweils ein neuer Unterordner – d.h. "act_export_1", "act_export_2" etc. – erstellt und die zu exportierenden Daten hierin gespeichert.

Die Variante 1 exportiert alle Transkript-Objekte des Korpus in allen verfügbaren Formaten. Variante 2 exportiert ebenfalls alle Transkript-Objekte, aber nur im Format .eaf, was im Parameter `formats` angegeben ist. Variante 3 exportiert lediglich zwei Transkript-Objekte, deren Namen im Parameter `filterTranscriptNames` angegeben sind. Die Namen der Transkript-Objekte stehen in einfachen Anführungsstrichen und sind mit dem Befehl `c()` zu einem Vektor zusammengefasst. Erstellt werden Drucktranskripte. Variante 4 exportiert lediglich das angegebene Transkript-Objekt als Drucktranskript und im .TextGrid-Format. Dabei wird die Ausgabe auf die beiden in `filterTierNames` angegebenen Annotationsspuren eingeschränkt.

Während in diesen Beispielen die Namen der Transkripte und Annotationsspuren direkt als Zeichenketten angegeben werden (d.h. z.B. `'ARG_I_CHI_Santi'` und `'ARG_I_PAR_Beto'`), besteht auch die Möglichkeit, nach Transkript-Objekten und Annotationsspuren zu suchen, die bestimmten Kriterien genügen. Das Erstellen solcher Filter wird in 6.8 besprochen.

Die Funktion `act::corpus_export()` kann auf ein Korpus-Objekt angewendet werden. Daneben besteht auch die Möglichkeit, einzelne Transkript-Objekte zu exportieren. Dies sind aktuell die Funktionen `act::export_eaf()`, `act::export_exb()`, `act::export_textgrid()`, `act::export_printtranscript()`, `act::export_srt()` und `act::export_rpraat()`.

Auf die Funktion `act::export_printtranscript()` wird im folgenden Abschnitt genauer eingegangen.

6.3. Erstellung von Drucktranskripten

Die Erstellung von Drucktranskripten wird innerhalb der act-Bibliothek über den Befehl `act::export_printtranscript()` realisiert. Diese Funktion wird z.B. immer dann aufgerufen, wenn alle Transkripte eines Korpus-Objekts mit `act::corpus_export()` exportiert werden (vgl. 6.2). Es ist aber auch möglich, den Befehl `act::export_printtranscript()` für ein einzelnes Transkript-Objekt aufzurufen. Als Parameter wird dann kein Korpus-Objekt, sondern ein Transkript-Objekt übergeben. Im folgenden Beispiel wird ein Drucktranskript für das Transkript-Objekt `'BOL_CBBA_SoniaYJanette'` erstellt.

```
printtrans <- act::export_printtranscript(
                            t = corpus@transcripts[['BOL_CBBA_SoniaYJanette']])
cat(printtrans)
```

Mit der ersten Zeile wird das Drucktranskript erstellt und in die Variable `printtrans` gespeichert. Mit dem Befehl `cat()` wird das Drucktranskript im Konsolenfenster von *RStudio* ausgegeben, und zwar so, dass Steuerzeichen, wie etwa Zeilenumbrüche, korrekt dargestellt werden. Die Ausgabe des Transkripts im Konsolenfenster ist in Abbildung 10 dargestellt. Zu sehen ist hier auch, dass das Transkript standardmäßig auf eine Breite von 65 Zeichen formatiert wird und dass

eckige Klammern zur Markierung des parallelen Sprechens untereinander ausgerichtet werden. Dieser Text kann im Konsolenfenster ausgewählt und beispielsweise direkt in ein Textverarbeitungsprogramm kopiert werden.

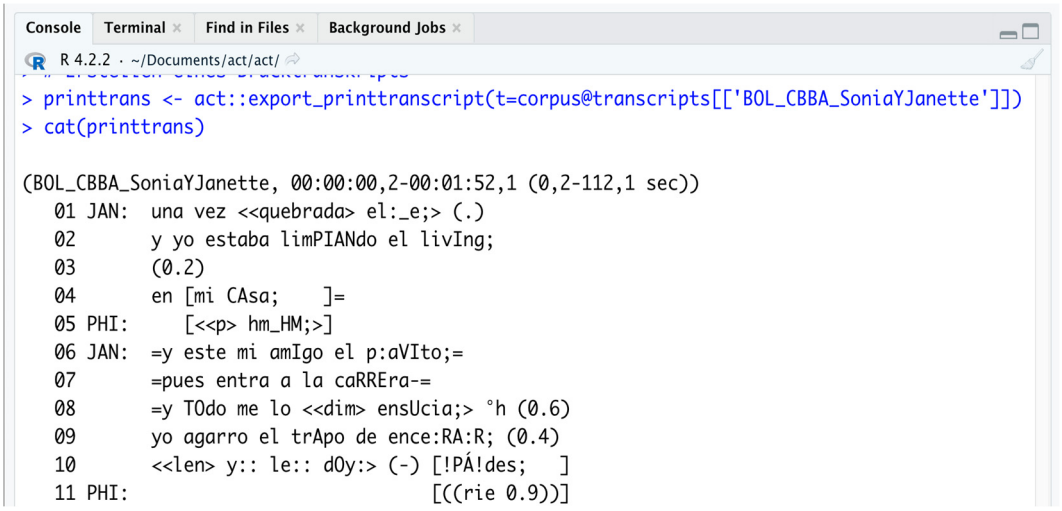


Abbildung 10: Ausgabe des Drucktranskripts im Konsolenfenster

Um das Layout des Drucktranskripts zu verändern, muss zunächst ein Layout-Objekt erstellt werden. In einem zweiten Schritt können dann die Parameter dieses Layout-Objekts angepasst werden. Bei der Erstellung von Drucktranskripten wird dann das Layout-Objekt an die Funktion `act::export_printtranscript()` übergeben und das Drucktranskript entsprechend formatiert. Dies illustrieren die folgenden Befehle.

```
meinlayout <- methods::new("layout")
meinlayout
?act::`layout-class`

meinlayout@speaker.width       <- 1
meinlayout@transcript.width    <- 50
meinlayout@brackets.tryToAlign <- FALSE

printtrans2 <-act::export_printtranscript(
                     t = corpus@transcripts[['BOL_CBBA_SoniaYJanette']],
                     l = meinlayout)
cat(printtrans2)
```

In der ersten Zeile wird mit dem Befehl `methods::new("layout")` ein neues Layout-Objekt erstellt und in der Variable `meinlayout` gespeichert. Die Eigenschaften dieses Objekts bzw. die gesetzten Werte der Parameter können über die Ausgabe des Objekts `meinlayout` eingesehen werden. Die Beschreibung der ca. 20 Parameter ist in der Hilfe dokumentiert und kann aufgerufen werden mit `?act::`layout-class``.

Mit den folgenden Befehlen werden die Eigenschaften des Layout-Objekts verändert: Die Breite der Sprechersiglen wird (von standardmäßig 3) auf 1 Zeichen reduziert, die Breite des Transkripts wird (von standardmäßig 65) auf 50 Zeichen

reduziert und die automatische Ausrichtung der eckigen Klammern wird ausgeschaltet.

Bei der Erstellung des Drucktranskripts mit `act::export_printtranscript()` wird dann im Parameter `l` das Layout-Objekt `meinlayout` übergeben. Das neu erstellte Drucktranskript `printtrans2` kann dann z.B. mit `cat()` wieder im Konsolenfenster ausgegeben werden. Die Ausgabe ist in Abbildung 11 dargestellt.

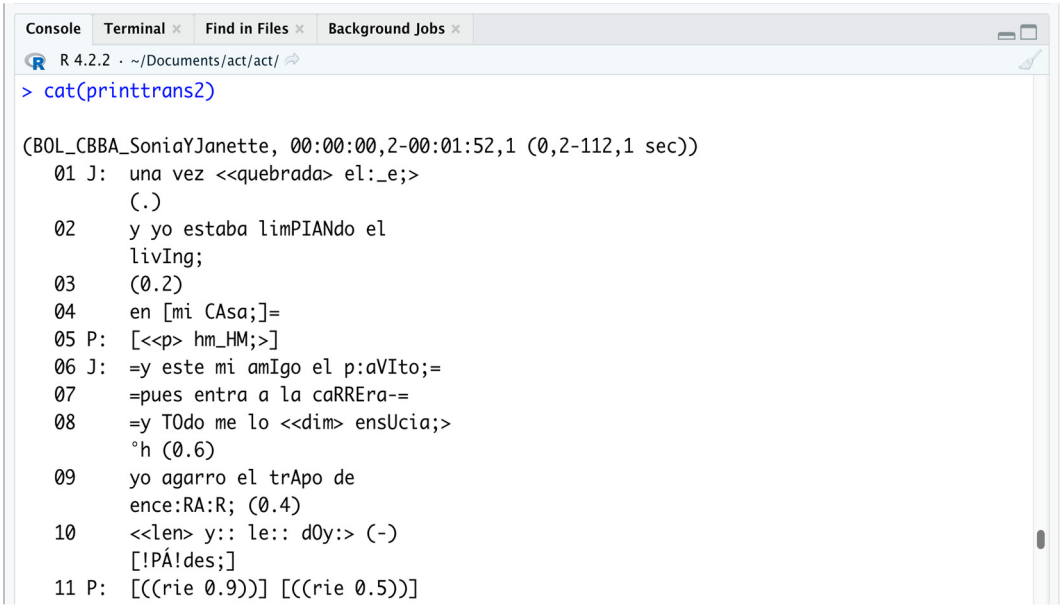


Abbildung 11: Ausgabe des Drucktranskripts mit veränderten Parametern im Konsolenfenster

Weiterhin ist es beispielsweise möglich, im Layout-Objekt festzulegen, dass nur bestimmte Annotationsspuren angezeigt werden sollen. Neben der Ausgabe des Transkriptes im Konsolenfenster ist es beim Aufruf von `act::export_printtranscript()` auch möglich, einen Pfad anzugeben, unter dem das Drucktranskript als .txt-Datei gespeichert werden soll. Weitere Optionen finden sich in der Dokumentation.

6.4. Suche im Korpus und Arbeiten mit Suchergebnissen

Die act-Bibliothek bietet die Möglichkeit, in allen Transkript-Objekten eines Korpus-Objekts zu suchen und die Suche anhand verschiedener Parameter anzupassen.

6.4.1. Erstellen und Ändern einer Suche

Das folgende Beispiel illustriert das Erstellen einer einfachen Suche.

```
suche <- act::search_new(x = corpus, pattern = "\\byo\\b")
suche
suche@results.nr
View(suche@results)
```

Durch den Befehl `act::search_new()` wird eine neue Suche erstellt. Im Parameter `x` wird angegeben, in welchem Korpus-Objekt gesucht werden soll und im Parameter `pattern` das Suchmuster. Das Suchmuster wird als regulärer Ausdruck interpretiert. In diesem Beispiel wird nach dem spanischen Personalpronomen *yo* (1. Person Singular) gesucht. Die Angabe von `\\b` vor und nach *yo* signalisiert eine Wortgrenze.[25] Hierdurch wird sichergestellt, dass keine Suchtreffer erzielt werden, die *yo* nur am Anfang (z.B. *yogur*), in der Mitte (z.B. *mayor*) oder am Ende eines Wortes (z.B. *paraguayo*) enthalten.

Das Ergebnis des Befehls ist ein Suche-Objekt, in dem sowohl die Suchparameter als auch die Ergebnisse gespeichert sind. Die einzelnen Parameter bzw. Informationen in diesem Suche-Objekt können über den @-Operator abgerufen bzw. modifiziert werden. So wird z.B. über `suche@results.nr` die Anzahl der Suchtreffer abgerufen. Die eigentlichen Suchtreffer sind in einer Tabelle bzw. einem `data.frame` in `suche@results` gespeichert. Dieses kann z.B. mit `View()` angezeigt werden. In diesen Ergebnissen sind alle Informationen zur Annotation enthalten, in denen das Suchmuster gefunden wurde (siehe Abbildung 12). Weiterhin werden die Suchtreffer als Konkordanz bzw. *Keyword in Context* (KWIC) dargestellt (siehe Abbildung 13).

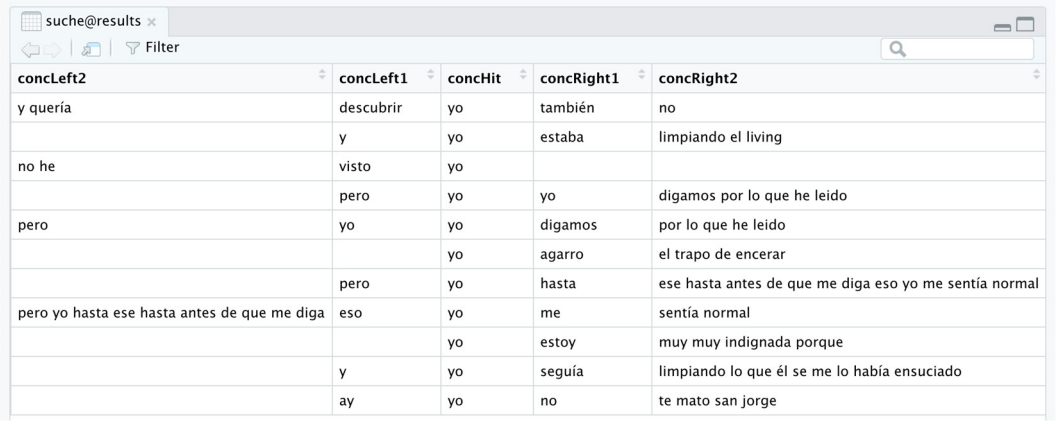
suche@results

Filter

transcript.name	annotationID	tier.name	startSec	endSec	content
BOL_PD_Ruido	1	Pr	0.1965618	2.699057	y que<<quebrada>ría::> (.) descubRIR yo
BOL_CBBA_SoniaYJanette	2	JAN	1.2417147	2.953814	y yo estaba limPIANdo el livIng;
BOL_CCBA_Botellita	8	A	4.0920174	4.683260	no he VISto_yo.
BOL_CCBA_SP_MeryGaby2	9	Mery	3.1109575	5.984671	<<f>pero: <<h>yo:_yo digA[mos (.) por l
BOL_CCBA_SP_MeryGaby2	9	Mery	3.1109575	5.984671	<<f>pero: <<h>yo:_yo digA[mos (.) por l
BOL_CBBA_SoniaYJanette	9	JAN	8.4861024	11.069998	yo agarro el trApo de ence:RA:R; (0.4)
ARG_I_PER_Alejo	8	A	8.1872142	11.779273	pero yO hasta ese (.) hasta Antes de: que
ARG_I_PER_Alejo	8	A	8.1872142	11.779273	pero yO hasta ese (.) hasta Antes de: que
BOL_RADIO_CCBA_SenoraIndignada	7	A	11.8523250	15.018290	=yo estoy muy °h muy indigNAda porque;
BOL_CBBA_SoniaYJanette	43	JAN	33.1720733	35.899136	°hh [<<f> y yo seguía limpiAndo lo que é
BOL_CBBA_SoniaYJanette	83	JAN	74.4116983	75.810224	<<:-), all> ay yo no te mAto san JOrge;>

Abbildung 12: Informationen zur Annotation in den Suchtreffern

[25] Zu beachten ist die Besonderheit bei der Verwendung von regulären Ausdrücken zwischen doppelten Anführungszeichen, dass bestimmte Elemente gekennzeichnet werden müssen. Beispielsweise wird eine Wortgrenze über den regulären Ausdruck `\b` markiert. Bei der Verwendung in *R* muss jeder umgekehrte Schrägstrich durch einen weiteren umgekehrten Schrägstrich gekennzeichnet werden. Es ergibt sich also `\\b` für die Verwendung in *R*.



suche@results

Filter

concLeft2	concLeft1	concHit	concRight1	concRight2
y quería	descubrir	yo	también	no
	y	yo	estaba	limpiando el living
no he	visto	yo		
	pero	yo	yo	digamos por lo que he leido
pero	yo	yo	digamos	por lo que he leido
		yo	agarro	el trapo de encerar
	pero	yo	hasta	ese hasta antes de que me diga eso yo me sentía normal
pero yo hasta ese hasta antes de que me diga	eso	yo	me	sentía normal
		yo	estoy	muy muy indignada porque
	y	yo	seguía	limpiando lo que él se me lo había ensuciado
	ay	yo	no	te mato san jorge

Abbildung 13: Konkordanz-Darstellung der Suchtreffer

Die Parameter der Suche können im Suche-Objekt geändert und die Suche erneut gestartet werden, wie im folgenden Beispiel.

```
suche@filter.transcript.includeRegEx <- 'ARG'
suche <- act::search_run(x = corpus, s = suche)
suche@results.nr
suche@results$transcript.name
```

Durch das Setzen des Parameters `@filter.transcript.includeRegEx` auf den Wert `ARG` wird lediglich in solchen Transkript-Objekten gesucht, deren Namen diese Zeichenkette enthält. Im Beispielkorpus sind dies Aufnahmen aus Argentinien. Die Zeichenkette wird dabei ebenfalls als regulärer Ausdruck interpretiert. Die Suche wird dann über den Befehl `act::search_run()` mit Angabe des Korpus- und des Suche-Objekts neu gestartet. Die Suche ergibt nun weniger Treffer (`suche@results.nr`) und enthält nur Ergebnisse aus den Transkripten, in deren Namen die Zeichenkette ARG vorkommt (`suche@results$transcript.name`). Die weiteren Parameter der Suche sind in der Dokumentation der Suchfunktion und im Suche-Objekt dokumentiert (Zugriff z.B. mit den Befehlen `?act::search_new` oder `?act::` `` `search-class` ``)

6.4.2. Suchmodi und Normalisierung

Die Suche in act bietet die Möglichkeit, erstens in den originalen oder normalisierten Annotationstexten zu suchen, in denen bestimmte vordefinierte Zeichen und Zeichenkombinationen (d.h. z.B. bestimmte Annotationskonventionen) herausgefiltert sind. Zweitens kann gewählt werden, ob Suchmuster lediglich *innerhalb* des Annotationstextes der einzelnen Annotation gefunden werden oder ob *über die Grenzen von Annotationen hinweg*, im Sinne einer Volltextsuche, gesucht werden soll. Die Optionen der Suche in den originalen/normalisierten Versionen und innerhalb der Annotationsgrenzen/im Volltext können frei miteinander kombiniert werden.

Die Unterschiede zwischen diesen Suchoptionen werden nun anhand eines Beispiels illustriert. Die folgende Abbildung 14 zeigt einen Ausschnitt aus der Aufnahme `'BOL_CCBA_SP_MeryGaby1'` in *Praat*, dem das nachfolgende Transkript in Beispiel (2) entspricht.

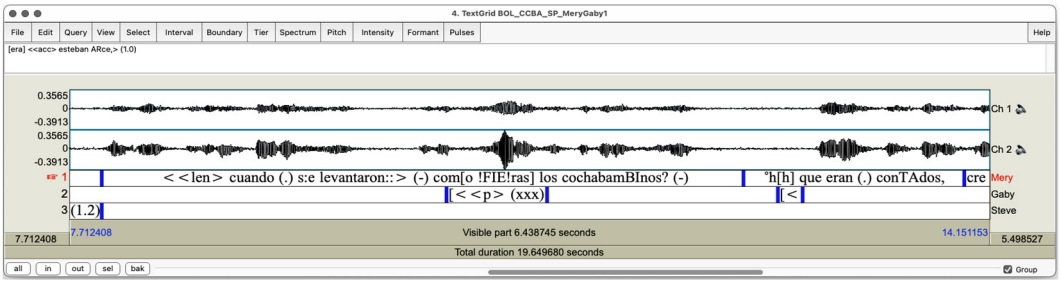


Abbildung 14: Screenshot der Aufnahme 'BOL_CCBA_SP_MeryGaby1' (7,7-14,2 sec) in *Praat*

Beispiel (5): *como fieras* 'wie die Berserker'
(Quelle: BOL_CCBA_SP_MeryGaby1, 7,7 – 14,2 sec)

```
01 M: <<len> cuando (.) s:e levantaron::> (-) com[o !FIE!ras ]
      los cochabamBInos? (-)
                  ‚als sich die Cochabambiner (Menschen aus der Stadt Cochabamba, OE)
      wie die Berserker erhoben‘
02 G:                                                  [<<p> (xxx)>]
                                                          ‚(xxx)‘
03 M: °h[h           ] que eran (.) conTAdos,
                       ‚die wenige waren‘
04 G :  [<<p> hm_HM.>]
                ‚hm_hm‘
```

6.4.2.1. Normalisierte Suche

Das Beispiel (5) ist nach den GAT-Konventionen transkribiert. So werden z.B. in Zeile 01 Längungen einzelner Laute mit dem Doppelpunkt signalisiert, welche direkt hinter einem bestimmten Graphem und damit innerhalb von Wörtern verwendet werden können, z.B. in `s:e` 'sich'. Ebenfalls werden z.B. Ausrufungszeichen innerhalb von Wörtern verwendet, um besonders starke Akzente zu markieren (z.B. `!FIE!ras` 'Berserker/Raubtiere'), oder eckige Klammern zur Markierung von gleichzeitigem Sprechen (wie z.B. in `com[o` 'wie'). Bei einer Suche nach den Standardgraphien dieser Wörter (z.B. *se*, *fieras*, oder *como*) werden diese im originalen Annotationstext nicht gefunden, da sie weitere Zeichen enthalten. Die `act`-Bibliothek bietet nun die Möglichkeit, diese Annotationszeichen herauszufiltern und eine 'normalisierte', d.h. der Standard-Orthografie angenäherte Schreibung zu erzielen. Auf diese Weise können auch Wörter gefunden werden, die aufgrund der Transkriptionskonventionen nicht gefunden werden könnten. Hervorgehoben sei an dieser Stelle, dass die `act`-Bibliothek also keine Normalisierung im engen Sinne bietet, in der auch nicht-standard-konforme Transkriptionen (z.B. "fuffzisch") ihrer orthographischen Entsprechungen (z.B. "fünfzig") zugewiesen werden.[26]

Die Normalisierung in `act` wird während des Imports von Annotationsdateien automatisch durchgeführt, wobei die Normalisierung standardmäßig auf die GAT-

[26] Diese Funktion einer Normalisierung i.e.S. bietet beispielsweise *OrthoNormal* (Schmidt 2012, 2016).

Konventionen definiert ist. Bei der Normalisierung wird neben der originalen Annotationstexte eine zweite Version erstellt. Für die Transkriptzeile 01 im obigen Beispiel sind dies die beiden folgenden Versionen:

- Original: `<<len> cuando (.) s:e levantaron::> (-) com[o !FIE!ras] los cochabamBInos? (-)`
- Normalisiert: `cuando se levantaron como fieras los cochabambinos`

In *R* können die beiden Versionen über die folgenden Befehle verglichen werden:

```
corpus@transcripts[['BOL_CCBA_SP_MeryGaby1']]@annotations$content[6]
corpus@transcripts[['BOL_CCBA_SP_MeryGaby1']]@annotations$content.norm[6]
```

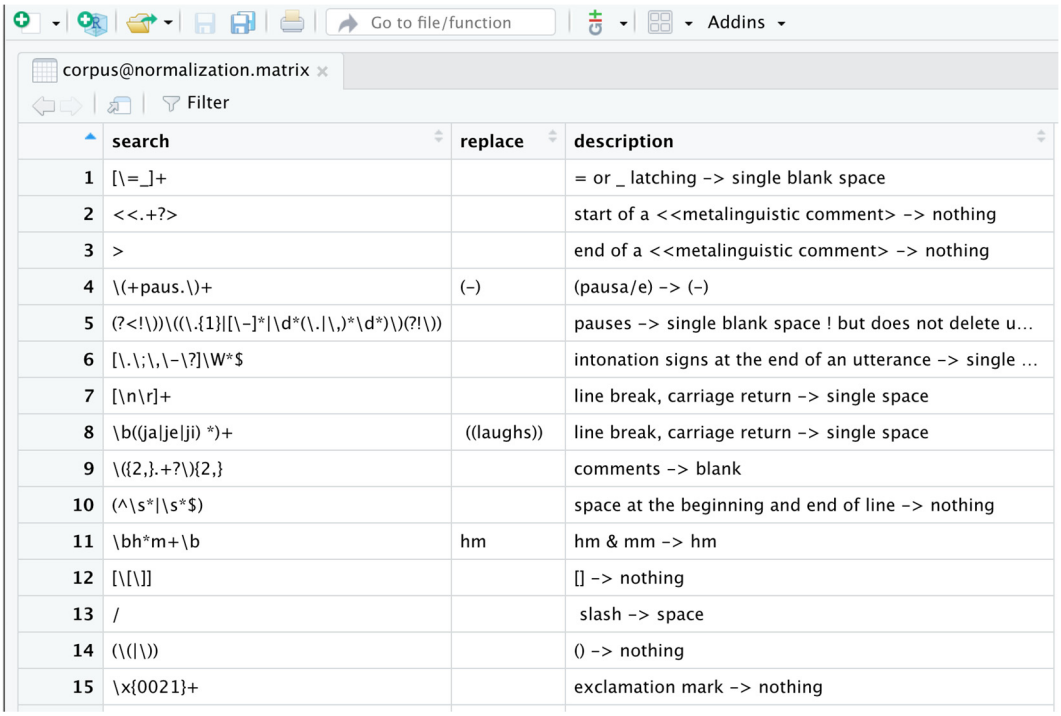
Um in den originalen bzw. normalisieren Versionen zu suchen, können die folgenden Befehle verwendet werden.

```
suche <- act::search_new(    x                   = corpus,
                             pattern             = 'fieras',
                             searchNormalized    = FALSE)
suche@results.nr
suche <- act::search_new(    x                   = corpus,
                             pattern             = 'fieras',
                             searchNormalized    = TRUE)
suche@results.nr
suche@results$hit[1]
suche@results$content[1]
```

Die erste Suche nach *fieras* mit dem Parameter `searchNormalized = FALSE` in den originalen Annotationstexten führt, wie erwartet, zu keinem Suchtreffer (`suche@results.nr` ergibt 0 im Konsolenfenster). Die normalisierte Suche mit dem Parameter `searchNormalized = TRUE` hingegen ergibt einen Suchtreffer, dessen Inhalt mit `suche@results$hit[1]` angezeigt werden kann. Der gesamte Inhalt der Annotation, die den Suchtreffer enthält, kann mit `suche@results$content[1]` ausgegeben werden.

Die Normalisierung beim Import der Annotationsdateien erfolgt angepasst auf die GAT-Konventionen. Hierfür wird eine Tabelle genutzt, in der über reguläre Ausdrücke angegeben ist, welche Konventionen bzw. Zeichenketten gesucht und in welcher Weise sie ersetzt werden sollen. Die aktuell in einem Korpus-Objekt verwendete Normalisierungsmatrix kann über den folgenden Befehl angezeigt werden.

```
View(corpus@normalization.matrix)
```



corpus@normalization.matrix

Filter

	search	replace	description
1	[\=_]+		= or _ latching -> single blank space
2	<<.+?>		start of a <<metalinguistic comment> -> nothing
3	>		end of a <<metalinguistic comment> -> nothing
4	\(+paus.\)+	(-)	(pausa/e) -> (-)
5	(?<!\))\((\.{1}|[\-]*|\d*(\.|\,)*\d*)\)(?!\))		pauses -> single blank space ! but does not delete u...
6	[\.\;\,\-\?]\W*$		intonation signs at the end of an utterance -> single ...
7	[\n\r]+		line break, carriage return -> single space
8	\b(ja|je|ji) *)+	((laughs))	line break, carriage return -> single space
9	\({2,}.+?\){2,}		comments -> blank
10	(^\s*|\s*$)		space at the beginning and end of line -> nothing
11	\bh*m+\b	hm	hm & mm -> hm
12	[\[\]]		[] -> nothing
13	/		slash -> space
14	(\(|\))		() -> nothing
15	\x{0021}+		exclamation mark -> nothing

Abbildung 15: Normalisierungsmatrix

Abbildung 15 zeigt einen Ausschnitt der standardmäßig verwendeten Normalisierungsmatrix für die GAT-Konventionen. In der ersten Spalte sind die regulären Ausdrücke für die Suche enthalten, die zweite Spalte enthält die Zeichen, durch welche die Suchtreffer ersetzt werden (in den meisten Fällen einfache oder doppelte Leerzeichen). Die dritte Spalte enthält eine kurze Beschreibung.

Eine solche Normalisierungsmatrix kann auch für andere Transkriptionskonventionen erstellt und verwendet werden. Hierzu kann beispielsweise direkt beim Import der Annotationsdateien über den Befehl `act::corpus_new()` im Parameter `pathNormalizationMatrix` der Pfad zu einer entsprechenden .csv-Datei angegeben werden.

6.4.2.2. Volltextsuche

Im Spanischen werden Relativsätze mit dem Relativpronomen *que* realisiert. Das oben angeführte Beispiel (5) enthält die Relativstruktur *los cochabambinos que eran contados* ('die Cochabambiner, die wenige waren', Z. 01 und 03). Um nach Relativstrukturen im Korpus zu suchen, kann beispielsweise ein regulärer Ausdruck verwendet werden, der den definiten maskulinen Artikel im Plural *los* gefolgt durch das Relativpronomen *que* im Abstand von 1 bis 20 Zeichen beschreibt. Ein solcher Ausdruck wäre `\blos\b.{1,20}\bque\b`. Nach diesem regulären Ausdruck kann nun erstens innerhalb der Grenzen der einzelnen Annotationen gesucht werden; dies entspricht dem Suchmodus `content` in der `act`-Bibliothek. Damit würde aber die Struktur aus dem obigen Beispiel nicht gefunden werden, da sich diese auf zwei Annotationen verteilt, dargestellt in den Transkriptzeilen 01 und 03. Zweitens ist es aber auch möglich, über die Grenzen von Annotationen hinweg zu suchen bzw. diese Grenze zu 'ignorieren'. Dies entspricht dem Suchmodus 'Volltext'. Dabei ist kann sowohl in Annotationen gesucht werden, die innerhalb *einer* Annotationsspur

direkt aufeinander folgen, als über Annotationen, die in *verschiedenen* Spuren aufeinander folgen oder sich teilweise überlappen.[27]

Die folgenden Befehle illustrieren die Unterschiede zwischen den Suchmodi `content` und `fulltext`, welche im Parameter `searchMode` angegeben sind. Zu beachten ist auch hier, dass die umgekehrten Schrägstriche \ des regulären Ausdrucks im *R*-Code verdoppelt werden müssen.

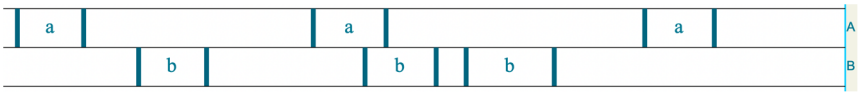
```
suche <- act::search_new(    x                    =corpus,
                             pattern              ="\\blos\\b.{1,20}\\bque\\b",
                             filterTranscriptNames = 'BOL_CCBA_SP_MeryGaby1',
                             searchMode           ='content')
suche@results.nr

suche <- act::search_new(    x                    = corpus,
                             pattern              = "\\blos\\b.{1,20}\\bque\\b",
                             filterTranscriptNames = 'BOL_CCBA_SP_MeryGaby1',
                             searchMode           = 'fulltext')
suche@results.nr
View(suche@results)
suche@results$hit[1]
suche@results$hit.span[1]
```

Der erste Befehl, mit welchem das Suchmuster lediglich innerhalb des Annotationstextes der einzelnen Annotation gesucht wird, erzielt kein Ergebnis für die Aufnahme 'BOL_CCBA_SP_MeryGaby1'. Der zweite Befehl mit dem Suchmodus `fulltext` hingegen erzielt einen Treffer. Im Suchtreffer wird dann durch das Zeichen # markiert, dass hier die Grenze einer Annotation verläuft, bzw. wird in der Spalte `hit.span` der Suchergebnisse mit `across tiers` angegeben, dass sich der Treffer über mehrere Annotationsspuren erstreckt (das ist hier der Fall, aufgrund des überlappenden Sprechens und des Beitrags von G in Zeile 02 des Transkripts).

---

[27] Technisch wird dies realisiert, indem intern mehrere Volltextversionen erstellt werden. Dies kann an der folgenden schematischen Annotationsdatei illustriert werden.



Erstellt werden die folgenden Versionen:
a) Volltext nach Annotationsspur: In einer ersten Version werden zunächst alle Annotationen einer Spur zusammengefügt und dann die Annotationen der folgenden Spuren angehängt: *a|a|a#b|b|b*.
b) Volltext nach Zeit: In der zweiten Volltextversion werden die Annotationen chronologisch nach ihrem Startzeitwert aneinandergehängt, unabhängig davon, in welcher Spur diese auftreten: *a#b#a#b|b#a*.
Standardmäßig werden Annotationen, die in *einer* Spur aufeinander folgen, mit dem | Pipe-Zeichen aneinandergehängt. Annotationen hingegen, die in verschiedenen Annotationspuren direkt aufeinanderfolgen oder sich zeitlich überlappen, werden durch das # Numeral-Zeichen getrennt. Diese Versionen werden sowohl für die originalen als auch die normalisierten Annotationstexte erstellt, um alle Suchoptionen miteinander kombinieren zu können.

6.5. Arbeiten mit den Suchergebnissen

Die act-Bibliothek bietet verschiedene Möglichkeiten, mit den Suchergebnissen weiterzuarbeiten: Für die einzelnen Suchtreffer können Transkript- und Medienausschnitte (6.5.1) erstellt werden, die Suchergebnisse können inkl. der Transkriptausschnitte in Tabellenformate exportiert und dann wieder re-importiert werden (6.5.2). Weiterhin können die Suchergebnisse als Ausgangspunkt für eine Kookkurrenz-Suche (6.5.3) genommen werden. Es ist auch möglich, die Suchergebnisse direkt in anderen Programmen (*ELAN*, *Praat* und *QuickTime*) zu öffnen und abzuspielen (6.5.4).

6.5.1. Erstellung von Ausschnitten

Im Folgenden wird dargestellt, wie erstens Transkriptausschnitte (6.5.1.1) und zweitens Medienausschnitte (6.5.1.2) erstellt werden können. Dies ist über separate Befehle möglich, die verschiedene Anpassungsmöglichkeiten bieten. In einem dritten Schritt wird vorgestellt, wie Transkript-, Medien- und Untertitelausschnitte mit nur einem Befehl erstellt werden können, der aber weniger Parameter bietet (6.5.1.3).

6.5.1.1. Erstellen von Transkriptausschnitten

Für die Analyse von Suchtreffern ist es meist unabdingbar, den Kontext einzubeziehen. Die act-Bibliothek bietet die Möglichkeit, für einen Suchtreffer einen Transkriptausschnitt zu erstellen, wobei eine Spanne angegeben werden kann, wie viele Sekunden vor dem Beginn des Suchtreffers und wie viele Sekunden nach dem Ende der Annotation einbezogen werden sollen.

```
suche <- act::search_new(x = corpus, pattern = "pero")
suche <- act::search_cuts_printtranscript(x = corpus, s = suche)

cat(suche@cuts.printtranscripts)
cat(suche@results$printtranscript)
cat(suche@results$printtranscript[2])
```

In der ersten Code-Zeile wird eine Suche nach dem spanischen Konnektor *pero* 'aber' ausgeführt und das Ergebnis im Objekt `suche` gespeichert. Die Erstellung der Transkriptausschnitte erfolgt mit dem Befehl `act::search_cuts_printtranscript()`, wobei als Parameter das Korpus-Objekt, mit dem die Suche ursprünglich ausgeführt wurde, und das Suche-Objekt selbst angegeben werden müssen. Das Ergebnis des Befehls ist eine Kopie des ursprünglichen Suche-Objekts, in das die Transkriptausschnitte eingefügt wurden. Die Transkriptausschnitte finden sich erstens gesammelt in der Eigenschaft `cuts.printtranscripts` des Suche-Objekts. Sie können über `cat()` korrekt formatiert im Konsolenfenster angezeigt werden. Zweitens wurden die Transkriptausschnitte in den Suchergebnissen selbst eingefügt, indem eine zusätzliche Spalte mit dem Namen `printtranscript` an das `data.frame` bzw. die Tabelle angehängt wurde. Diese Spalte kann mit `cat(suche@results$print`

`transcript)` komplett ausgegeben werden. Auf den Transkriptausschnitt eines einzelnen Suchtreffers kann durch Angabe der Zeile, z.B. mit `suche@results$print transcript[2]`, zugegriffen werden.

Voreingestellt ist ein Kontextfenster von 0 Sekunden, sodass lediglich Transkriptausschnitte erstellt werden, die die Annotation beinhalten, in der der Suchtreffer liegt, sowie weitere Annotationen, die innerhalb dieser Annotation beginnen. Das Fenster der Ausschnitte kann wie folgt geändert werden.

```
suche@cuts.span.beforesec <- 1
suche@cuts.span.aftersec <- 3
suche <- act::search_cuts_printtranscript(x = corpus, s = suche)
```

In der ersten Code-Zeile wird die Eigenschaft bzw. der Wert des Slots `cuts.span.beforesec` auf den Wert 1 gesetzt. Damit wird festgelegt, dass die zu erstellenden Ausschnitte 1 Sekunde vor dem Startwert der Annotation, die den Suchtreffer enthält, beginnen sollen. Analog wird über `cuts.span.beforesec` festgelegt, wie viele Sekunden nach dem Ende der Annotation, die den Suchtreffer enthält, der Ausschnitt enden soll. Wenn der Befehl `search_cuts_printtranscript()` nun mit dem Korpus-Objekt und dem modifizierten Suche-Objekt nochmals ausgeführt wird, werden die aktualisierten Werte übernommen. Zum Vergleich kann beispielsweise der Transkriptausschnitt des dritten Suchtreffers mit `cat(suche@results$print transcript[3])` im Konsolenfenster ausgegeben werden (Abbildung 16).

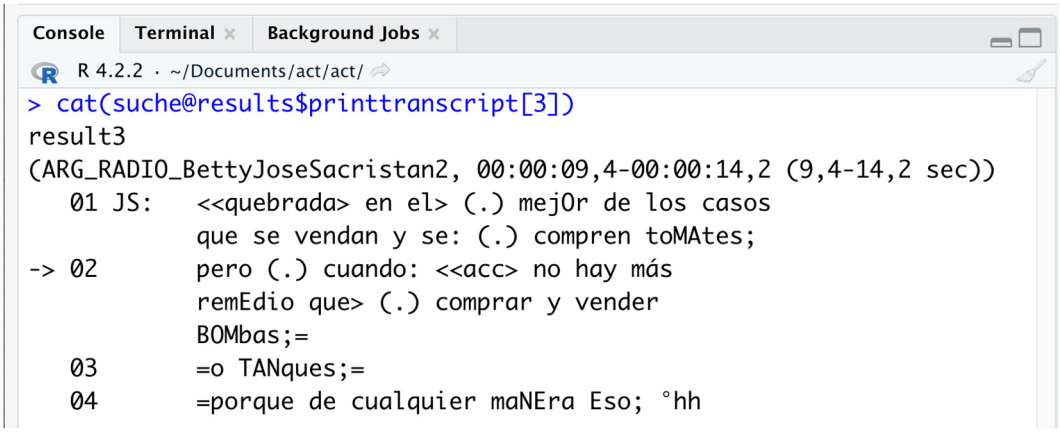


Abbildung 16: Ausgabe des Transkriptausschnitts eines Suchtreffers

Der Ausschnitt enthält nun mehr Transkriptzeilen. Die Zeile mit dem Suchtreffer ist durch den Pfeil -> hervorgehoben.

Während bei den bisherigen Funktionsaufrufen die Transkriptausschnitte lediglich in das Suche-Objekt eingetragen und von dort im Konsolenfenster ausgegeben wurden, ist es auch möglich, diese direkt in einem Ordner auf dem Computer zu speichern. Das folgende Code-Beispiel illustriert dies.

```
suche@name <- 'act_search_pero'
searchDir <- tools::file_path_as_absolute(rstudioapi::selectDirectory())
suche <- act::search_cuts_printtranscript( x            = corpus,
                                           s            = suche,
                                           outputFolder = searchDir)
```

Mit dem ersten Befehl wird der Name der Suche in `'act_search_pero'` geändert. Dieser Name wird später als Name eines neuen Ordners genutzt, in dem die Transkriptausschnitte gespeichert werden. Mit der zweiten Code-Zeile kann über das Dialogfenster der Zielordner ausgewählt werden, in dem dieser Ordner erstellt wird. Beim erneuten Aufruf der Funktion `act::search_cuts_printtranscript()` wird im Parameter `outputFolder` der Zielordner angegeben. Beim Ausführen des Befehls wird nun im Zielordner ein Unterordner mit dem Namen `'act_search_pero'` erstellt, in dem verschiedene Dateien gespeichert werden (vgl. Abbildung 17).

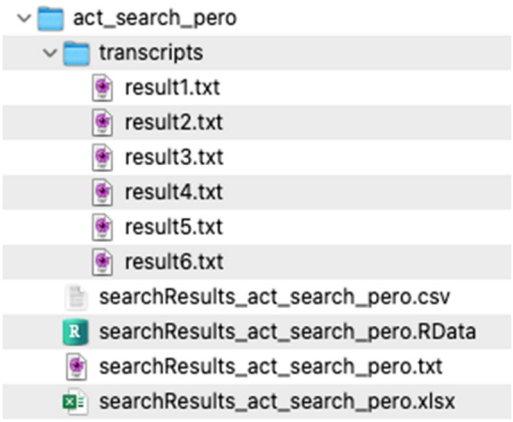
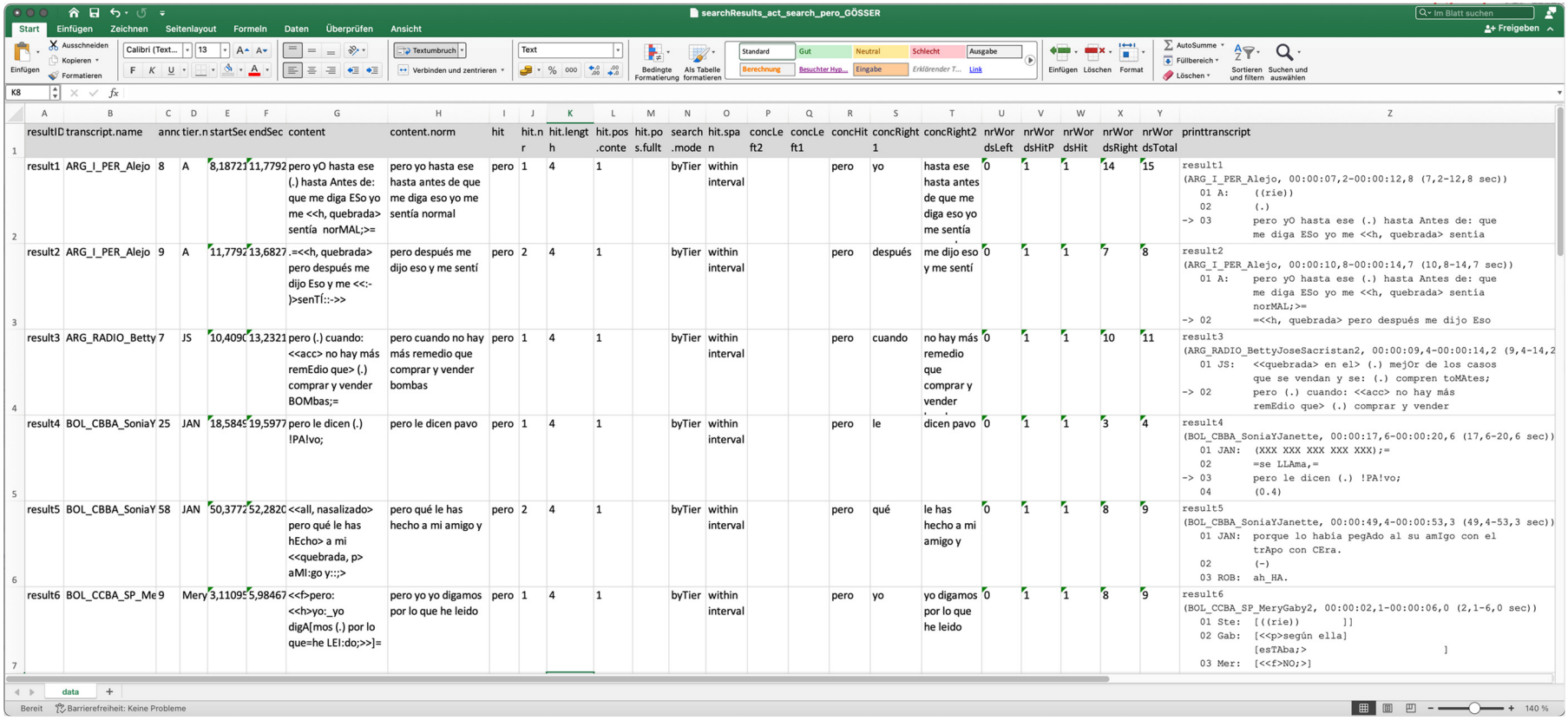


Abbildung 17: Inhalt des Ordners mit exportierten Transkripten

Der Unterordner 'transcripts' enthält mehrere .txt-Dateien, in denen die Transkriptausschnitte der einzelne Suchtreffer gespeichert sind. Weiterhin sind vier Dateien mit dem Namen 'searchResults_act_search_pero' enthalten, die sich lediglich durch die Endung unterscheiden. In der .txt-Datei sind alle Transkriptausschnitte gesammelt gespeichert. Die .csv- und die .xlsx-Datei enthalten Tabellen mit den Suchergebnissen, wobei die Transkripte in die letzte Spalte eingefügt sind (vgl. Abbildung 18). Die .RData-Datei enthält die Suchergebnisse im *R*-Datenformat.

6.5.1.2. Erstellung von Medienausschnitten

Medienausschnitte werden nicht direkt mit den Funktionen von *R* erstellt, sondern es wird die kostenlos verfügbare Software *FFmpeg* genutzt, die eine Vielzahl von Datenformaten unterstützt. Bei *FFmpeg* handelt es sich um ein Kommandozeilenprogramm, das nicht über eine Benutzeroberfläche, sondern direkt über Befehle in Textform bedient wird. Das Grundprinzip der Erstellung von Medienausschnitten mit der act-Bibliothek besteht darin, dass Schnittbefehle für *FFmpeg* erstellt werden, die dann in einem zweiten Schritt von *FFmpeg* ausgeführt werden. Verarbeitet werden sowohl Audiodateien als auch Videodateien.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	resultID	transcript.name	annc	tier.n	startSec	endSec	content	content.norm	hit	hit.nr	hit.length	hit.pos.conte	hit.pos.fullt	search.mode	hit.span	concLeft2	concLeft1	concHit	concRight1	concRight2	nrWordsLeft	nrWordsHitP	nrWordsHit	nrWordsRight	nrWordsTotal	printtranscript
2	result1	ARG_I_PER_Alejo	8	A	8,18721	11,7792	pero yO hasta ese (.) hasta Antes de: que me diga ESo yo me <<h, quebrada> sentía norMAL;>=	pero yo hasta ese hasta antes de que me diga eso yo me sentía normal	pero	1	4	1		byTier	within interval			pero	yo	hasta ese hasta antes de que me diga eso yo me sentía	0	1	1	14	15	result1 (ARG_I_PER_Alejo, 00:00:07,2-00:00:12,8 (7,2-12,8 sec)) 01 A: ((rie)) 02 (.) -> 03 pero yO hasta ese (.) hasta Antes de: que me diga ESo yo me <<h, quebrada> sentía
3	result2	ARG_I_PER_Alejo	9	A	11,7792	13,6827	.=<<h, quebrada> pero después me dijo Eso y me <<:-)>senTí::->>	pero después me dijo eso y me sentí	pero	2	4	1		byTier	within interval			pero	después	me dijo eso y me sentí	0	1	1	7	8	result2 (ARG_I_PER_Alejo, 00:00:10,8-00:00:14,7 (10,8-14,7 sec)) 01 A: pero yO hasta ese (.) hasta Antes de: que me diga ESo yo me <<h, quebrada> sentía norMAL;>= -> 02 =<<h, quebrada> pero después me dijo Eso
4	result3	ARG_RADIO_Betty	7	JS	10,4090	13,2321	pero (.) cuando: <<acc> no hay más remEdio que> (.) comprar y vender BOMbas;=	pero cuando no hay más remedio que comprar y vender bombas	pero	1	4	1		byTier	within interval			pero	cuando	no hay más remedio que comprar y vender	0	1	1	10	11	result3 (ARG_RADIO_BettyJoseSacristan2, 00:00:09,4-00:00:14,2 (9,4-14,2 01 JS: <<quebrada> en el> (.) mejOr de los casos que se vendan y se: (.) compren toMAtes; -> 02 pero (.) cuando: <<acc> no hay más remEdio que> (.) comprar y vender
5	result4	BOL_CBBA_SoniaY	25	JAN	18,5845	19,5977	pero le dicen (.) !PA!vo;	pero le dicen pavo	pero	1	4	1		byTier	within interval			pero	le	dicen pavo	0	1	1	3	4	result4 (BOL_CBBA_SoniaYJanette, 00:00:17,6-00:00:20,6 (17,6-20,6 sec)) 01 JAN: (XXX XXX XXX XXX);= 02 =se LLAma,= -> 03 pero le dicen (.) !PA!vo; 04 (0.4)
6	result5	BOL_CBBA_SoniaY	58	JAN	50,3772	52,2820	<<all, nasalizado> pero qué le has hEcho> a mi <<quebrada, p> aMI:go y:::;>	pero qué le has hecho a mi amigo y	pero	2	4	1		byTier	within interval			pero	qué	le has hecho a mi amigo y	0	1	1	8	9	result5 (BOL_CBBA_SoniaYJanette, 00:00:49,4-00:00:53,3 (49,4-53,3 sec)) 01 JAN: porque lo había pegAdo al su amIgo con el trApo con CEra. 02 (-) 03 ROB: ah_HA.
7	result6	BOL_CCBA_SP_Me	9	Mery	3,11095	5,98467	<<f>pero: <<h>yo:_yo digA[mos (.) por lo que=he LEI:do;>>]=	pero yo yo digamos por lo que he leido	pero	1	4	1		byTier	within interval			pero	yo	yo digamos por lo que he leido	0	1	1	8	9	result6 (BOL_CCBA_SP_MeryGaby2, 00:00:02,1-00:00:06,0 (2,1-6,0 sec)) 01 Ste: [((rie))]] 02 Gab: [<<p>según ella] [esTAba;>] 03 Mer: [<<f>NO;>]

Abbildung 18: Tabellenkalkulationsprogramm mit den eingefügten Transkriptausschnitten

Um Medienausschnitte zu erstellen muss *FFmpeg* zunächst installiert werden. Eine Anleitung bzw. *Vignette* zur Installation kann mit dem folgenden Befehl aufgerufen werden.

```
vignette("install_ffmpeg", package="act")
```

Um die Schnittbefehle zu erstellen, wird der Befehl `act::search_cuts_media()` verwendet. Der Befehl `act::search_cuts_media()` wird mit denselben Parametern wie der Befehl `act::search_cuts_printtranscript()` zur Erstellung der Drucktranskripte aufgerufen: dem Korpus-Objekt, dem Suche-Objekt und einem optionalen Ausgabeordner.

```
suche <- act:: search_cuts_media(   x             = corpus,
                                    s             = suche,
                                    outputFolder  = searchDir)
```

Wenn beim Aufruf des Befehls im Parameter `outputFolder` ein Zielverzeichnis angegeben ist, dann finden sich nach der Ausführung dort zwei weitere Dateien mit den Namen 'FFMPEG_cutlist_act_search_pero_win.cmd' für *Windows* und 'FFMPEG_cutlist_act_search_pero_mac' für *macOS* (vgl. Abbildung 19).

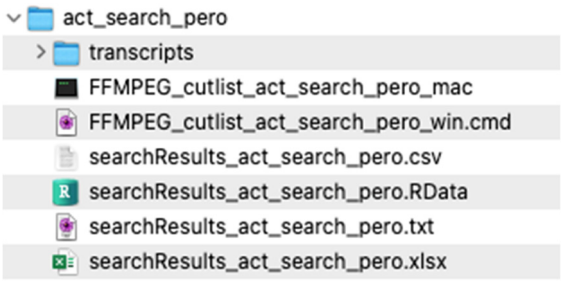


Abbildung 19: Schnittlisten-Dateien für *macOS* und *Windows*

Diese Dateien enthalten eine 'Schnittliste' mit Befehlen zum Erstellen aller Ausschnitte. Es handelt sich um Skriptdateien, die durch das jeweilige Betriebssystem – *macOS* oder *Windows* – ausgeführt werden können. Ein Doppelklick auf die Dateien öffnet diese und die Schnittliste wird dann abgearbeitet.

Neben der Ausführung der erstellten Schnittlisten-Dateien können die Schnitt-Befehle noch auf eine weitere Weise direkt in *RStudio* verwendet werden. Die Schnittbefehle wurden bei der Ausführung des Befehls `act::search_cuts_media()` im Suche-Objekt gespeichert. Diese befinden sich erstens für alle Suchtreffer gesammelt in der Eigenschaft `@cuts.cutlist.win` bzw. `@cuts.cutlist.mac` des Korpus-Objekts. Zweitens sind die Schnittbefehle für jeden Suchtreffer einzeln in der Tabelle mit den Suchergebnissen, jeweils in den Spalten `cuts.cutlist.mac` bzw. `cuts.cutlist.win` gespeichert. Das folgende Code-Beispiel illustriert, wie diese angezeigt werden können.

```
cat(suche@cuts.cutlist.mac)
cat(suche@cuts.cutlist.win)

cat(suche@results$cuts.cutlist.mac[3])
cat(suche@results$cuts.cutlist.win[3])
```

Die ersten beiden Zeilen zeigen jeweils die gesamte Schnittliste im Konsolenfenster an. Die beiden folgenden Zeilen geben die Schnittbefehle lediglich für den Suchtreffer 3 aus. Für *macOS* sehen die Befehle wie in Abbildung 20 dargestellt aus.

```
#!/bin/sh
mkdir -p "/Users/oliverehmer/Downloads/act_search_pero/result3"

if [ -f
        "/Users/oliverehmer/Desktop/act_examplecorpus/gat/ARG_RADIO_BettyJoseSacristan2.wa
        v"]
then
   ffmpeg -i
        "/Users/oliverehmer/Desktop/act_examplecorpus/gat/ARG_RADIO_BettyJoseSacristan2.wa
        v" -ss 9.40 -t 6.82 -y
        "/Users/oliverehmer/Downloads/act_search_pero/result3/result3.wav" -hide_banner
fi
```

Abbildung 20: Schnittbefehle für einen Suchtreffer

Der eigentliche *FFmpeg*-Befehl – im Code-Beispiel beginnend mit `ffmpeg` – besteht im Prinzip aus der Angabe, welches die Datei ist, aus der ausgeschnitten werden soll, dem Start- und Endzeitpunkt des Ausschnitts und der Angabe, wo die Ausschnitte gespeichert werden sollen. Die Angaben der Quell-Mediendatei werden aus dem Transkript-Objekt entnommen, aus dem der Suchtreffer stammt. In jedem Transkript-Objekt sind in der Eigenschaft `@media.path` identifizierte Mediendateien verlinkt.[28] Der Befehl `act::search_cuts_media()` erstellt Schnittbefehle jeder Mediendatei, die mit dem Transkript-Objekt verknüpft ist, aus dem ein Suchtreffer stammt. Das heißt, wenn sowohl eine Audio- als auch eine Videodatei verlinkt sind, werden ein Audio- und ein Video-Ausschnitt erstellt. Die Zeitangaben werden direkt aus dem Suche-Objekt übernommen, d.h. die Zeitangaben werden aus den Start- und Endzeiten jedes Suchtreffers übernommen (in `@results$startSec` und `@results$startSec`) und mit den Angaben zur Fensterbreite der Ausschnitte (in `@cuts.span.beforesec` und `@cuts.span.aftersec`) kombiniert. Neben dem eigentlichen *FFmpeg*-Befehl in Zeile 6 sind noch weitere Befehle enthalten, mit denen ein Zielordner erstellt wird (Zeile 2), und Angaben, mit denen überprüft wird, ob die Mediendatei verfügbar ist, aus der ausgeschnitten werden soll (Zeile 4). Während die Abbildung die Befehle für *macOS* darstellt, finden sich in der *Windows*-Variante die analogen Befehle.

Diese Befehle können direkt im Terminal (*macOS*) bzw. der Eingabeaufforderung (*Windows*) ausgeführt werden. Hierzu kann der Text einfach in das Terminal bzw. in die Eingabeaufforderung kopiert werden.

---

[28] Die mit dem Transkript 'ARG_RADIO_BettyJoseSacristan2' verlinkten Mediendateien lassen sich beispielsweise wie folgt abrufen:
`corpus@transcripts[['ARG_RADIO_BettyJoseSacristan2']]@media.path`

Eine weitere Möglichkeit besteht darin, den *R*-Befehl `rstudioapi::terminalExecute()` zu verwenden. Mit diesem *R*-Befehl können Betriebssystembefehle direkt an ein neues Terminal in *RStudio* gesendet werden. Unter *macOS* wäre das Code-Beispiel wie folgt:

```
# unter macOS:
termId = rstudioapi::terminalExecute(suche@results$cuts.cutlist.mac[3])
rstudioapi::terminalKill(termId)
```

Als Parameter des Befehls `rstudioapi::terminalExecute()` werden die gewünschten Schnittbefehle übergeben, im Beispiel `suche@results$cuts.cutlist.mac[3]` sind dies die Schnittbefehle für den dritten Suchtreffer. Der zweite Befehl schließt das geöffnete Terminal wieder. Nach dem Ausführen findet sich im oben gewählten Zielverzeichnis ein Ordner mit dem Namen 'result3', der den Medienausschnitt enthält.

Das Vorgehen unter *Windows* benötigt einen Zwischenschritt. Hier können die Befehle nicht direkt an das Terminal gesendet werden, sondern müssen in einer temporären .cmd-Datei zwischengespeichert werden, die dann ausgeführt wird. Dies kann wie folgt realisiert werden.

```
# unter Windows:
temp <- tempfile(fileext=".cmd")
write(suche@results$cuts.cutlist.win[3], file=temp)
rstudioapi::terminalExecute(temp)
rstudioapi::terminalKill(termId)
```

In der ersten Zeile nach dem Kommentar wird mit `tempfile()` der Pfad zu einer temporären Datei erstellt. In der zweiten Zeile werden die Schnittbefehle für den Dritten Suchtreffer in diese Datei geschrieben. Im dritten Befehl `rstudioapi::terminalExecute()` wird dann angegeben, dass diese Datei ausgeführt werden soll. Der letzte Befehl `rstudioapi::terminalKill()` schließt auch hier das zuvor geöffnete Terminal wieder.

6.5.1.3. Erstellen von Transkript-, Medien- und Untertitelausschnitten

Anstatt die Drucktranskripte und die *FFmpeg*-Schnittbefehle für die Suchergebnisse nacheinander zu erstellen, kann dies auch mit nur einem Befehl erfolgen. Mit `act::search_cuts()` werden Drucktranskripte, Schnittbefehle und .srt-Untertitel erstellt. Dabei ruft der Befehl `act::search_cuts()` nacheinander die Befehle `act::search_cuts_printtranscript()`, `act::search_cuts_media()` und `act::search_cuts_srt()` auf. Die Zeitangaben in den .srt-Untertiteldateien werden dabei an die Medienausschnitte angepasst. Der folgende Code gibt ein Beispiel hierfür. Der Code sucht nach dem spanischen Pronomen *yo* 'ich' (1. Person Singular).

```
outputDir <- rstudioapi::selectDirectory()
suche.1PSG <- act::search_new(      x                = corpus,
                                    pattern          = "\\byo\\b",
                                    name             = "act_search_Pron-1PSG")

suche.1PSG <- act::search_cuts(     x                = corpus,
                                    s                = suche.1PSG,
                                    cutSpanBeforesec = 1,
                                    cutSpanAftersec  = 3,
                                    outputFolder     = outputDir)

#unter macOS:
termId = rstudioapi::terminalExecute(suche.1PSG@cuts.cutlist.mac)
rstudioapi::terminalKill(termId)

#unter Windows:
temp <- tempfile(fileext=".cmd")
write(suche.1PSG@results$cuts.cutlist.win, file=temp)
rstudioapi::terminalExecute(temp)
rstudioapi::terminalKill(termId)
```

Die beiden ersten Befehle illustrieren, wie zunächst eine Suche ausgeführt und in einem zweiten Schritt dann die Ausschnitte bzw. Schnittlisten erstellt werden. Zu beachten ist, dass der Befehl `act::search_cuts()` weniger Parameter bietet, als die separaten Funktionen `act::search_cuts_printtranscript()` und `act::search_cuts_media()`. Wenn diese Parameter benötigt werden, dann müssen die Befehle hintereinander verwendet werden. Für das Ausführen der Schnittlisten müssen auf *macOS* und *Windows* unterschiedliche Code-Zeilen ausgeführt werden, wie im Beispiel mit den Kommentaren markiert.

6.5.2. Export und Re-Import von Suchergebnissen

Der Import und Export von Daten in Tabellenformaten (.csv und .xlsx) ist in *R* allgemein sehr einfach über Standardfunktionen möglich. In der act-Bibliothek stehen aber spezifische Befehle zum Export und Re-import von Suchergebnissen zur Verfügung. Der technische Hintergrund ist, dass in Transkripten nach den GAT-Konventionen häufig Gleichheitszeichen am Anfang von Annotationstexten verwendet werden, um einen schnellen Anschluss zu markieren. Diese Gleichheitszeichen werden von Tabellenkalkulationsprogrammen als Beginn einer mathematischen Formel interpretiert, weshalb sich beim Öffnen der Tabellen-Dateien Fehler ergeben. Die Export-Funktion in act ersetzt diese Gleichheitszeichen durch die Zeichenkombination "'=" (Apostroph mit Gleichheitszeichen). Beim Re-Import mit act wird diese Ersetzung rückgängig gemacht. Darüber hinaus wird beim Import überprüft, ob alle notwendigen Spalten für die Arbeit mit der act-Bibliothek vorhanden sind. Die Funktionen können wie folgt angewendet werden:

```
suche <- act::search_new(x=corpus, pattern="\\bpero\\b")
path<- rstudioapi::selectFile(existing = FALSE)
if(tolower(tools::file_ext(path))!="xlsx"){
              path<- paste(path, "xlsx", sep = ".")}
act::search_results_export(suche, path = path)

suche.reimport <- act::search_results_import(path = path)
View(suche.reimport@results)
```

Ausgehend von einer Suche wird ein Dialog geöffnet, der nach dem Pfad fragt, wohin die Suchergebnisse exportiert werden sollen. Hier kann ein beliebiger Dateiname angegeben werden. Wenn die Datei noch nicht existiert, wird sie später automatisch erstellt. Da die Funktion `act::search_results_export()` im Parameter `path` einen kompletten Dateipfad mit Datei-Suffix benötigt, wird in der dritten Code-Zeile überprüft, ob dieses vorhanden ist. Die vierte Code-Zeile exportiert die Suchergebnisse in eine .xlsx-Datei. Mit den nachfolgenden Zeilen wird diese .xlsx-Datei wieder eingelesen. Das Ergebnis ist ein Suche-Objekt.

6.5.3. Kookkurrenz-Suche

Die act-Bibliothek ermöglicht die Suche nach zeitlichen Kookkurrenzen von Suchausdrücken auf verschiedenen Annotationsspuren. Ausgehend von einer ersten Suche wird eine weitere Suche ausgeführt, die für jeden Suchtreffer der ersten Suche überprüft, ob es auf einer anderen Annotationsspur eine Annotation gibt, die den Suchkriterien entspricht.

Das folgende Code-Beispiel illustriert dies für gleichzeitiges Lachen in der Aufnahme. Dafür wird als Suchmuster der reguläre Ausdruck `\br[ií]e\w*` verwendet, der alle Worte findet, die mit der Buchstabenkombination *rie* oder *ríe* beginnen, wodurch finite Varianten des spanischen Verbs *reír* 'lachen' gefunden werden.[29] Wichtig ist, dass über die Option `searchNormalized = FALSE` in den originalen Annotationstexten gesucht wird, da *((ríe))* bzw. <<*riendo* > gemäß der GAT-Konventionen bei der Normalisierung herausgefiltert werden. Die Kookkurrenz-Suche wird mit dem Befehl `act::search_sub()` realisiert.

```
pattern.laugh <- "\\br[ií]e\\w*"

suche <- act::search_new(    x                  = corpus,
                             pattern            = pattern.laugh,
                             searchNormalized   = FALSE)

suche <- act::search_sub(    x                  = corpus,
                             s                  = suche,
                             pattern            = pattern.laugh,
                             searchNormalized   = FALSE)
```

[29] Tatsächlich könnte der reguläre Ausdruck noch genauer sein, indem die spitzen und runden Klammern in die Suche einbezogen werden, damit keine literal transkribierten Worte, die mit *rie* bzw. *ríe* beginnen, gefunden werden. Ein solcher Ausdruck könnte lauten: `(<<|\(\().{0,20}?\br[ií]e\w*.{0,20}?(>|\)\)).`

```
View(suche@results)
```

Das Ergebnis der ersten Suche, d.h. ein Suche-Objekt, wird im Parameter `s` an den Befehl `act::search_sub()` übergeben. Das Ergebnis dieses Befehls ist eine aktualisierte Version dieses Suche-Objekts. Im `data.frame` mit den Suchergebnissen ist nun eine weitere Spalte mit dem Namen `subsearch` vorhanden. Diese enthält den Inhalt von zeitlich überlappenden Annotationen, die dem Suchmuster entsprechen. Wenn in einer Zeile (d.h. für eines der ursprünglichen Suchergebnisse) in der Spalte `subsearch` kein Eintrag vorhanden ist, wurde keine passende zeitliche Kookkurrenz gefunden.

Für das aktuelle Beispiel illustriert dies die Abbildung 21, in der ein Ausschnitt aus der Tabelle `suche@results` dargestellt ist. Zu sehen ist, dass für die ersten beiden Suchtreffer, in der Spalte `subsearch` nichts eingetragen ist. Hier wurden also keine zeitlich überlappenden Suchtreffer gefunden. Für die ursprünglichen Suchtreffer in den Zeilen 3 und 4 hingegen ist in der Spalte `subsearch` zu sehen, welchen Inhalt überlappende Annotationen auf anderen Annotationsspuren haben, die dem Suchmuster entsprechen.

suche

Filter

	resultID	content	subsearch
1	result01	y ahí me explicÓ: (.) la importAn<<riendo>cia real>…	
2	result02	((rie))	
3	result03	[((rie))]	[<<riendo> clA:ro CLAro.>]
4	result04	[<<riendo> clA:ro CLAro.>]	[((rie))]
5	result05	[((rie 0.9))] [((rie 0.5))] [((rie 0.4))]	

Abbildung 21: Suchergebnisse mit zeitlichen Kookkurrenzen

In der Abbildung 22 ist der Suchtreffer 3 in Praat zu sehen, in Abbildung 23 als Drucktranskript im Konsolenfenster.

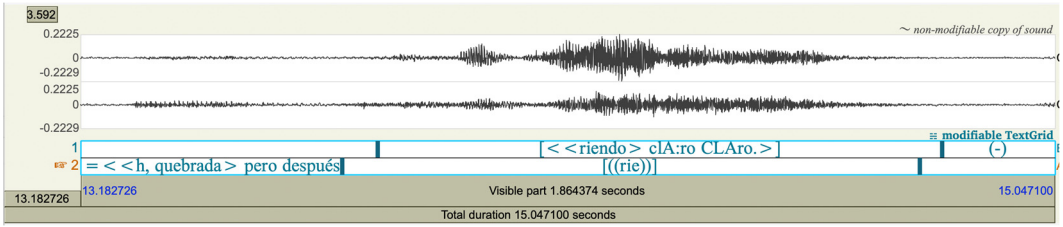


Abbildung 22: Darstellung eines Suchtreffers in *Praat*

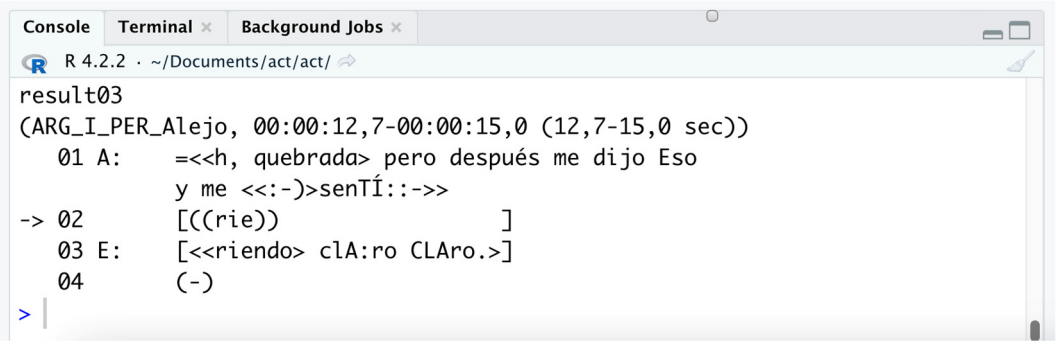


Abbildung 23: Transkriptausschnitt eines Suchtreffers im Konsolenfenster

6.5.4. Öffnen von Suchergebnissen in anderen Programmen

Die Suchergebnisse können in anderen Programmen geöffnet werden. In den Programmen *Praat* und *ELAN* können sowohl die Annotationsdateien und die verknüpften Mediendateien geöffnet werden. Unter *macOS* ist es möglich, die Mediendatei in *QuickTime* zu öffnen und abzuspielen. Die Voraussetzung für das Öffnen von Mediendateien ist, dass Verknüpfungen zu diesen bestehen (vgl. 6.1.5)

6.5.4.1. Öffnen eines Suchergebnisses in *ELAN*

Um ein Suchergebnis in *ELAN* zu öffnen, muss die act-Bibliothek wissen, wo *ELAN* auf dem Computer gespeichert ist. Danach kann mit dem Befehl `act::search_openresult_inelan()` ein Suchergebnis in *ELAN* geöffnet werden.

```
options(act.path.elan=rstudioapi::selectFile())
act::search_openresult_inelan(x = corpus, s = suche, resultNr = 1)
```

Mit der ersten Code-Zeile wird ein Dateiauswahldialog geöffnet, in dem die *ELAN*-Programmdatei ausgewählt werden muss. Dieses befindet sich normalerweise im Ordner "Programme". Diese Information wird dann permanent in den Voreinstellungen der act-Bibliothek gespeichert, sodass dieser Schritt auch beim Neustart von *RStudio* bzw. *R* nicht mehr vollzogen werden muss.

Der folgende Befehl `act::search_openresult_inelan()` öffnet ein Suchergebnis in *ELAN*, wobei ein Suche-Objekt, das Korpus-Objekt, auf dem die Suche basiert, und die Nummer des Suchergebnisses als Parameter angegeben werden müssen. Wenn die ursprüngliche Annotationsdatei, aus der der Treffer stammt, eine *ELAN* .eaf-Datei ist, dann wird versucht, diese Originaldatei zu öffnen. Sollte die Datei nicht gefunden werden oder das ursprüngliche Annotationsdateiformat ein anderes gewesen sein, dann wird ad hoc eine .eaf-Datei erstellt.[30]

Sollte das automatische Öffnen nicht funktionieren, kann der Fehler in manchen Fällen dadurch behoben werden, dass das Programm *ELAN* vor dem Ausführen des Befehls schon gestartet wurde.

[30] Um tatsächlich den gewünschten Suchtreffer in der Datei auszuwählen, wird ad hoc eine .pfsx-Datei mit den entsprechenden Informationen erstellt. Vielen Dank an dieser Stelle an Han Sloetjes für wertvolle Hinweise zum .pfsx-Dateiformat.

6.5.4.2. Öffnen eines Suchergebnisses in *Praat*

Für das Öffnen von Suchergebnissen in *Praat* ist nicht nur *Praat* selbst notwendig, sondern weiterhin das Hilfsprogramm *sendpraat* (für *Windows*) bzw. *sendpraat_carbon* (für *macOS*). Dieses Hilfsprogramm wird benötigt, um *Praat* aus *R* heraus fernzusteuern und muss zuerst installiert werden. Die Installation ist in einer Anleitung bzw. *Vignette* in der act-Bibliothek beschrieben. Diese kann mit dem folgenden Befehl angezeigt werden.

```
vignette("install_sendpraat", package="act")
```

Das Öffnen von Suchergebnissen in *Praat* erfolgt analog zum Öffnen von Ergebnissen in *ELAN* mit dem Befehl `act::search_openresult_inpraat()`. Bevor dies möglich ist, muss ebenfalls einmalig angegeben werden, wo sich die Programme *Praat* und *sendpraat* bzw. *sendpraat_carbon* auf dem Computer befinden.

```
options(act.path.praat       = rstudioapi::selectFile())
options(act.path.sendpraat   = rstudioapi::selectFile())
act::search_openresult_inpraat(x = corpus, s = suche, resultNr = 1)
```

Mit den ersten beiden Code-Zeilen werden Dateiauswahldialoge geöffnet, mit denen zunächst *Praat* und dann *sendpraat* bzw. *sendpraat_carbon* ausgewählt werden müssen. Der Befehl `act::search_openresult_inpraat()` öffnet das erste Suchergebnis in *Praat* und spielt dieses ab, wenn eine Audiodatei verfügbar ist. Auch hier gilt wieder, dass, wenn möglich, die originale .TextGrid-Datei geöffnet wird und andernfalls ad hoc eine temporäre .TextGrid-Datei erstellt wird.

Im Normalfall startet der Befehl `act::search_openresult_inpraat()` das Programm *Praat*, wenn dieses noch nicht geöffnet ist. Sollte es aber zu Problemen mit dem Befehl kommen, dann kann der Parameter `delayBeforeOpen` angepasst werden. Mit diesem wird das Öffnen des Suchtreffers verzögert. Alternativ kann *Praat* zuvor von Hand gestartet werden.

6.5.4.3. Öffnen eines Suchergebnisses in *QuickTime* (nur *macOS*)

Unter *macOS* können Suchtreffer direkt in *QuickTime* abgespielt werden, bzw. werden verlinkte Mediendateien geöffnet und abgespielt.

```
act::search_openresult_inquicktime(x = corpus, s = suche, resultNr = 1)
```

Es ist auch möglich, alle Suchtreffer nacheinander abzuspielen. Hierzu wird der Befehl `act::search_playresults_inquicktime()` verwendet. Wenn für die Suchtreffer bereits Transkriptausschnitte erstellt wurden, so werden diese in *RStudio* angezeigt. Das folgende Code-Beispiel illustriert dies.

```
suche <- act::search_new(    x                  = corpus,
                             pattern            = "\\bpero\\b",
                             cutSpanBeforesec   = 1,
                             cutSpanAftersec    = 3)
suche <- act::search_cuts_printtranscript(x = corpus, s = suche)
act::search_playresults_inquicktime(x = corpus, s = suche)
```

6.6. Interoperabilität mit der rPraat-Bibliothek

Für den Austausch von Daten mit der rPraat-Bibliothek (Bořil/Skarnitzl 2016) stehen in der act-Bibliothek die Befehle `act::export_rpraat()` und `act::import_rpraat()`. Das folgende Code-Beispiel illustriert, wie ein Transkript-Objekt zunächst in ein `rPraat`-TextGrid-Objekt konvertiert und dann eine Funktion `rPraat::tg.plot()` genutzt wird, um die Annotationen in *RStudio* anzuzeigen (vgl. Abbildung 24).

```
install.packages("rPraat")
library(rPraat)
rPraatTextgrid <- act::export_rpraat(corpus@transcripts[[1]])
rPraat::tg.plot(rPraatTextgrid)
```

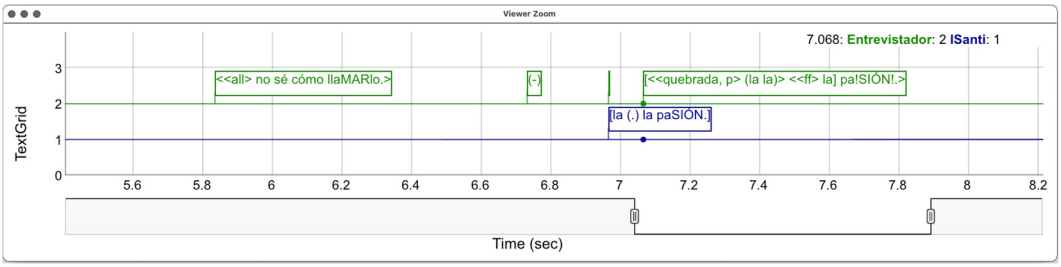


Abbildung 24: Anzeige von Transkript-Objekten mit `rPraat::tg.plot()`

6.7. Bearbeiten von Daten

Das *Aligned Corpus Toolkit* bietet verschiedene Funktionen zur Bearbeitung der Daten in einem Korpus-Objekt. Zunächst werden einige der Funktionen zur Bearbeitung von Annotationen (6.7.1), Annotationsspuren (engl. *tiers*, 6.7.2) und Transkript-Objekten (6.7.3) vorgestellt. Abschließend wird erläutert, wie die Inhalte zweier oder mehrerer Transkript-Objekte zusammengeführt werden können (6.7.4). Dies ist beispielsweise sinnvoll, wenn überarbeitete Ausschnitte aus einer oder mehreren Annotationsdateien in eine andere Annotationsdatei übernommen werden sollen.

6.7.1. Bearbeiten von Annotationen

Mit der act-Bibliothek können sowohl der Inhalt von Annotationen verändert, als auch Annotationen insgesamt kopiert oder gelöscht werden. Verschiedene Funktionen realisiert der Befehl `act::annotations_replace_copy()`. Das folgende Beispiel illustriert, wie in den Annotationstexten nach Zeichenkombinationen gesucht und diese ersetzt werden können.

```
corpus2 <- act::annotations_replace_copy(
                                x           = corpus,
                                pattern     = "(?i)\\b(mery|manuela)\\b",
                                replacement = "NAME")

View(corpus2@transcripts[['BOL_CCBA_SP_MeryGaby2']]@annotations)
```

Mit dem ersten Befehl wird nach dem regulären Ausdruck `(?i)\b(mery|manuela)\b` gesucht, mit dem die Namen 'Mery' und 'Manuela' erfasst werden. Die Option `(?i)` legt dabei fest, dass Groß- und Kleinschreibung ignoriert werden sollen. Diese beiden Namen werden dann durch die Zeichenkette 'NAME' ersetzt. Mit dem zweiten Befehl werden die Annotationen des Transkripts `BOL_CCBA_SP_MeryGaby2` angezeigt, das durch den Befehl verändert wurde. In Abbildung 25 ist ersichtlich, dass in den Annotationen 3 und 10 der originale Annotationstext (Spalte 'content') nun die Zeichenkette 'NAME' enthält.[31]

Filter

	annotationID	tier.name	startSec	endSec	content	content.norm
1	1	Mery	0.0000000	0.4154908	[<<:–)> BUEno.>]	bueno
2	2	Steve	0.0000000	0.4243826	[((rie))][((rie))]	
3	3	Steve	0.4243826	1.9571169	[<<:–)> la NAME] ya esTAba n[o:–>]	la mery ya estaba no
4	4	Mery	0.5577606	0.9364190	[por lo–]	por lo
5	5	Mery	1.7918104	1.9578422	[no,]	no
6	6	Steve	1.9571169	2.7266257	[((rie))]]	
7	7	Gaby	2.1197675	3.1109575	[<<p>según ella] [esTAba;>]	según ella estaba
8	8	Mery	2.7266257	3.1109575	[<<f>NO;>]	no
9	9	Mery	3.1109575	5.9846712	<<f>pero: <<h>yo:_yo digA[mos (.) por lo que=he LEI:do;>>]=	pero yo yo digamos por lo que he leido
10	10	Gaby	4.5740401	5.9846712	[(iba (xxx) arriba) NAME;]	iba xxx arriba manuela

Abbildung 25: Ergebnis des Suchens und Ersetzens von Zeichenketten

Es ist weiterhin möglich, Annotationen, die einem bestimmten Suchmuster entsprechen, auf eine andere Annotationsspur zu kopieren. Hierzu wird ebenfalls der Befehl `act::annotations_replace_copy()` genutzt.

```
corpus2 <- act::annotations_replace_copy(  x            = corpus,
                                           pattern      = "\\bpero\\b",
                                           destTier     = "sequence")
View(corpus2@transcripts[['ARG_I_PER_Alejo']]@annotations)
```

Mit dem ersten Befehl wird nach dem regulären Ausdruck `\bpero\b` gesucht. Alle Annotationen, die diesen regulären Ausdruck enthalten, werden auf die Annotationsspur kopiert, deren Namen im Parameter `destTier` angegeben ist. Ist diese Annotationsspur noch nicht im jeweiligen Transkript-Objekt enthalten, so wird sie standardmäßig neu erstellt. Der Inhalt der kopierten Annotation ist dann nicht der komplette Text der ursprünglichen Annotation, sondern lediglich der Suchtreffer.

[31] Zu beachten ist, dass durch den Befehl `act::annotations_replace_copy()` lediglich der originale Annotationstext verändert wird, nicht aber der normalisierte Annotationstext (Spalte 'content.norm'). In Abbildung 25 der Spalte 'content.norm' ist zu sehen, dass in den Zeilen 3 und 10 immer noch die Namen 'mery' und 'manuela' vorhanden sind. Die Normalisierung sowie die Erstellung der korrespondierenden Volltexte erfolgen automatisiert immer dann, wenn eine Suche ausgeführt wird, und zwar nur für die veränderten Transkript-Objekte. Alternativ kann die Normalisierung manuell mit dem Befehl `act::transcripts_update_normalization()` durchgeführt werden und die Erstellung der Volltexte mit dem Befehl `act::transcripts_update_fulltexts()`.

Mit dem zweiten Befehl des Beispiels werden die Annotationen im Transkript-Objekt `'ARG_I_PER_Alejo'` angezeigt, und es ist ersichtlich, dass zwei neue Annotationen in der Spur `"sequence"` hinzugefügt wurden.

Neben der Veränderung des Inhalts von Annotationen können auch alle Annotationen in einem Korpus-Objekt gelöscht werden, deren Inhalt einem bestimmten Suchmuster entspricht. Dies ist möglich mit dem Befehl `act::annotations_delete()`.

Änderungen, die an einem Korpus-Objekt vorgenommen werden, werden in der Eigenschaft `@history` des Objektes dokumentiert. Dies gilt aber lediglich für Änderungen, die mit den Funktionen der act-Bibliothek vorgenommen werden, und nicht für solche Änderungen, die durch einen direkten Zugriff auf die Daten erfolgen. Das folgende Beispiel gibt alle Änderungen am Objekt `corpus2` im Konsolenfenster aus.

```
corpus2@history
```

6.7.2. Bearbeiten von Annotationsspuren

Mit dem Befehl `act::tiers_add()` können allen oder ausgewählten Transkript-Objekten Annotationsspuren hinzugefügt werden.

```
corpus2 <- act::tiers_add(corpus, tierName="TEST")
corpus@transcripts[[1]]@tiers
corpus2@transcripts[[1]]@tiers
```

Im Beispiel wird allen Transkript-Objekten eine Annotationsspur mit dem Namen `"TEST"` hinzugefügt. Die beiden folgenden Code-Zeilen zeigen die Tabelle mit den *tiers* im originalen und im modifizierten Korpus-Objekt an. Neue Annotationsspuren werden standardmäßig an letzter Position eingefügt. Dies kann über die Angabe weiterer Parameter im Befehl `act::tiers_add()` verändert werden. Es ist aber auch möglich, die Reihenfolge von Annotationsspuren in den Transkript-Objekten eines Korpus mit dem Befehl `act::tiers_sort()` zu verändern.

```
corpus2 <- act::tiers_sort(x = corpus2, sortVector = c("Entrevistador", "TEST"))
corpus2@transcripts[[1]]@tiers
corpus2@transcripts[[2]]@tiers
```

Der erste Befehl des Beispiels sortiert die Annotationsspuren in allen Transkripten so, dass die Annotationsspur `"Entrevistador"` an erster und die Spur `"TEST"` an zweiter Stelle steht, vorausgesetzt, dass diese im jeweiligen Transkript-Objekt enthalten sind. Weitere in den Transkript-Objekten enthaltene Annotationsspuren werden in ihrer bestehenden Reihenfolge danach eingefügt.

Das folgende Code-Beispiel illustriert, wie Annotationsspuren umbenannt werden können mit `act::tiers_rename()`.

```
corpus2 <- act::tiers_rename(       x                 = corpus2,
                                    searchPattern     = "TEST",
                                    searchReplacement = "XXX")
corpus2@transcripts[[1]]@tiers
```

Mit diesem Befehl wird in allen Annotationsspuren die im Parameter `searchPattern` angegebene Zeichenkette durch die Zeichenkette ersetzt, die in `searchReplacement` angegeben ist.

Das Löschen von Annotationsspuren ist über den Befehl `act::tiers_delete()` möglich.

```
corpus2 <- act::tiers_delete(x = corpus2, tierNames = "XXX")
corpus2@transcripts[[1]]@tiers
```

Im Parameter `tierNames` ist hier nur der Name `"XXX"` angegeben. Es können jedoch auch mehrere Namen von zu löschenden Annotationsspuren angegeben werden, z.B. mit dem Befehl `c()`. Zum Ermitteln der Namen von Annotationsspuren, die eine bestimmte Zeichenkette enthalten, kann der Befehl `act::search_filter()` genutzt werden (vgl. 6.8).

6.7.3. Bearbeiten von Transkript-Objekten

In diesem Abschnitt werden grundlegende Funktionen zum Ändern von Transkript-Objekten erläutert. Der Befehl `act::transcripts_rename()` benennt Transkript-Objekte in einem Korpus-Objekt um. Dabei wird in den Transkript-Namen nach regulären Ausdrücken gesucht und die Suchtreffer werden durch andere Zeichenketten ersetzt.

```
corpus2 <- act::transcripts_rename( x                 = corpus,
                                    searchPatterns    = c("ARG_", "BOL_"),
                                    searchReplacements = c("Argentina_",
                                                           "Bolivia_"))
act::info_summarized(corpus)$transcripts.names
act::info_summarized(corpus2)$transcripts.names
```

Im Parameter `searchPatterns` sind hier zwei reguläre Ausdrücke angegeben. Die Suchtreffer dieser Ausdrücke werden durch die korrespondierenden Zeichenketten in `searchReplacements` ersetzt. Im konkreten Beispiel werden die Länderangaben in den Transkriptnamen `ARG_` durch `Argentina_` und `BOL_` durch `Bolivia_` ersetzt. Das Löschen von Transkripten erfolgt durch den Befehl `act::transcripts_delete()`, wobei die Namen der Transkripte im Parameter `transcriptNames` angegeben werden.

```
corpus2 <- act::transcripts_delete( x               = corpus,
                                    transcriptNames = c('ARG_I_CHI_Santi',
                                                        'ARG_I_PAR_Beto'))
act::info_summarized(corpus2)$transcripts.names
```

In diesem Beispiel werden die Transkripte mit den Namen `'ARG_I_CHI_Santi'` und `'ARG_I_PAR_Beto'` gelöscht, indem die Namen mit dem Befehl `c()` zu einem Vektor verbunden werden. Um die Namen von Transkript-Objekten zu ermitteln, die eine bestimmte Zeichenkette enthalten, kann der Befehl `act::search_filter()` genutzt werden (vgl. 6.8).

Um einem Korpus-Objekt Transkripte hinzuzufügen, kann der Befehl `act::transcripts_add()` verwendet werden.

```
transkripte <- corpus@transcripts[1:2]
corpus2 <- act::transcripts_add(x = corpus2, transkripte)
act::info_summarized(corpus2)$transcripts.names
```

In diesem Code-Beispiel werden in der ersten Zeile die ersten beiden Transkript-Objekte aus dem Korpus-Objekt `corpus` in die Variable `transkripte` gespeichert. Zu beachten ist, dass einfache eckige Klammern `[]` verwendet werden müssen. Dies ist anders als beim Zugriff auf nur ein einzelnes Transkript-Objekt mit doppelten eckigen Klammern `[[]]`. Mit der zweiten Code-Zeile werden diese Transkripte dem Korpus-Objekt `corpus2` hinzugefügt. Mit der dritten Code-Zeile werden über den Befehl `act::info_summarized()` zusammengefasste Informationen über das Korpus-Objekt `corpus2` abgerufen. In der Eigenschaft `$transcripts.names` befinden sich die Namen aller Transkripte. Die neuen Transkript-Objekte wurden am Ende der Liste der Transkript-Objekte hinzugefügt.

6.7.4. Zusammenführen der Inhalte von Transkript-Objekten

Das *Aligned Corpus Toolkit* stellt einen Befehl zur Verfügung, mit dem die Inhalte mehrerer Annotationsdateien bzw. Transkript-Objekte zusammengeführt werden können. Dabei werden ausgewählte Bereiche aus einem oder mehreren Transkript-Objekten in ein anderes Transkript-Objekt eingefügt. Eine praktische Anwendung hierfür ist beispielsweise, wenn für eine Mediendatei mehrere Annotationsdateien vorliegen, in denen unterschiedliche Bereiche transkribiert sind, die in eine Datei überführt werden sollen. Eine weitere Anwendungsmöglichkeit liegt vor, wenn mehrere Personen gleichzeitig verschiedene Bereiche einer Transkription überarbeiten und die Ergebnisse am Ende zusammengefügt werden sollen.

Das Grundprinzip der Zusammenführung besteht darin, dass ein Transkript-Objekt als Zielobjekt gewählt wird. Ein oder mehrere andere Transkript-Objekte, aus denen selektiv zeitliche Bereiche übernommen werden sollen, werden als Quellen bestimmt. In den Quelltranskripten müssen die zu übernehmenden Bereiche in einer separaten Annotationsspur mit markiert sein. Voreingestellt ist, dass die Annotationsspur den Namen "update" trägt, wobei der Text in den Markierungen selbst beliebig ist. Diese Werte sind anpassbar.

Das folgende Beispiel basiert auf den Annotationsdateien des Beispielkorpus im Unterordner "update". Abbildung 26 zeigt einen *Praat*-Sceenshot des Zieltranskripts. Im Bereich der roten Markierung sind noch keine Annotationen vorhanden. Abbildung 27 zeigt einen *Praat*-Screenshot des Quelltranskripts, aus dem Annotationen übernommen werden sollen. Zu sehen ist, dass im rot markierten Bereich Annotationen vorhanden sind. Abbildung 28 zeigt das Ergebnis der Zusammenführung.

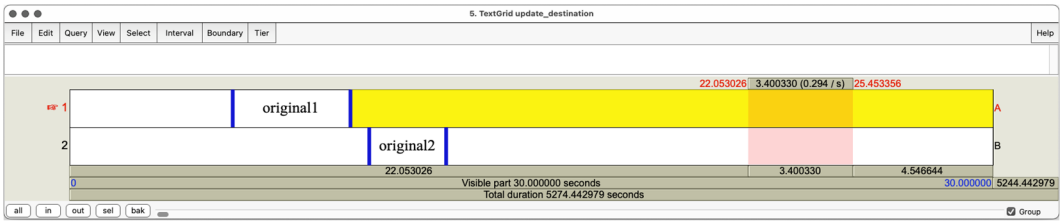


Abbildung 26: Screenshot der .TextGrid-Datei 'update_destination' in *Praat*

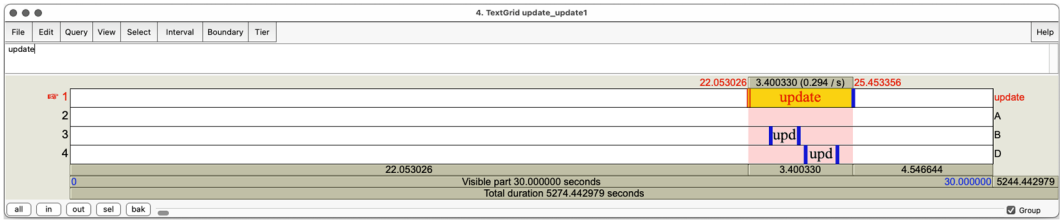


Abbildung 27: Screenshot der .TextGrid-Datei 'update_update1' in *Praat*

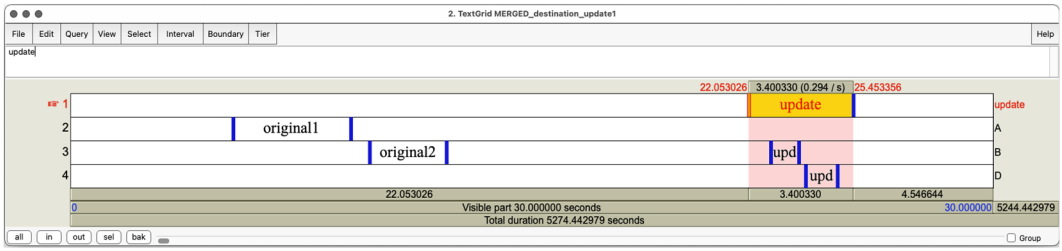


Abbildung 28: Ergebnis der Zusammenführung in *Praat*

Betrachtet man das Zieltranskript in Abbildung 26 genauer, so ist zu sehen, dass das *Praat* .TextGrid zwei Annotationsspuren mit den Namen "A" und "B" enthält, in denen jeweils eine Annotation vorhanden ist. Das Quelltranskript in Abbildung 27 enthält vier Annotationsspuren. In der ersten Spur mit dem Namen "update" ist über eine Annotation (hier mit dem Text "update") markiert, dass dieser Bereich in die Zieldatei übernommen werden soll. Weiterhin befinden sich im Quelltranskript die Annotationsspuren "A", "B" und "D", in denen zwei Annotationen vorhanden sind. Eine der Annotationen befindet sich in der Annotationspur "B", die auch im Zieltranskript vorhanden ist. Die andere Annotation befindet sich in der Annotationsspur "D", also einer Spur, für die im Zieltranskript bislang keine Entsprechung vorliegt.

Betrachtet man das Ergebnis der Zusammenführung in Abbildung 28, so ist zu sehen, dass hier ebenfalls vier Annotationsspuren vorhanden sind. In der ersten Annotationsspur mit dem Namen "update" ist markiert, welcher bzw. welche Bereiche aktualisiert wurden. In den Zeilen darunter befinden sich die zusammengeführten Annotationen aus den ursprünglichen Transkript-Objekten. Zu beachten ist, dass in diesem .TextGrid nun alle Annotationsspuren – d.h. "A" "B" und "D" – vorhanden sind, die in den verschiedenen Ausgangsdateien vorlagen. D.h. die Annotationen wurden bei der Zusammenführung entweder in die gleichnamige Spur im Zieltranskript eingefügt oder eine entsprechende Annotationsspur wurde erstellt.

Die Zusammenführung der Annotationsdateien erfolgt mit dem Befehl `act::transcripts_merge()`. Die Annotationsdateien müssen dabei bereits als Transkript-Objekte in einem Korpus-Objekt vorliegen.

```
corpus2 <- act::transcripts_merge(
                    x                           = examplecorpus,
                    destinationTranscriptName   = 'update_destination',
                    updateTranscriptNames       = 'update_update1')

View(corpus@transcripts[['update_destination']]@annotations)
View(corpus@transcripts[['update_update1']]@annotations)
View(corpus2@transcripts[['update_destination']]@annotations)

act::info_summarized(corpus)$transcripts.names
act::info_summarized(corpus2)$transcripts.names
```

Im Befehl `act::transcripts_merge()` wird im Parameter `destinationTranscriptName` der Name des Transkript-Objekts angegeben, das als Ziel fungiert bzw. *in* welches eingefügt werden soll. Im Parameter `updateTranscriptNames` wird der Name des Transkript-Objekts angegeben, das als Quelle fungiert bzw. aus dem markierte Bereiche übernommen werden sollen. Die Standardeinstellung besteht darin, dass die zu übernehmenden Bereiche in einer Annotationsspur mit dem Namen "update" eingetragen sind und einen beliebigen Text enthalten.[32]

Das Ergebnis des Befehls `act::transcripts_merge()` ist ein aktualisiertes Korpus-Objekt. Das Transkript-Objekt, das als Ziel-Objekt ausgewählt wurde, enthält nun die zusammengefügten Daten. Die folgenden Code-Zeilen im Beispiel zeigen an, welchen Inhalt das ursprüngliche Ziel-Transkript und das Update-Transkript hatten und dann, welchen Inhalt das aktualisierte Ziel-Transkript hat. Die letzten beiden Zeilen im Beispiel zeigen, dass im modifizierten Korpus-Objekt, die Transkript-Objekte mit den Updates nicht mehr vorhanden sind.

Mit dem Befehl `act::transcripts_merge()` ist es möglich, nicht nur zwei, sondern beliebig viele Transkript-Objekte zusammenzuführen. Die Abbildung 29 zeigt einen *Praat*-Screenshot einer weiteren Annotationsdatei. Hier ist zu beachten, dass zwei der Annotationen in den Zeilen "A" und "C" über den in der Spur "update" markierten Bereich hinausragen. Bei der Zusammenführung werden diese Annotationen übernommen, jedoch in Bezug auf deren Start- bzw. Endzeitpunkt abgeschnitten. Das Ergebnis der Zusammenführung aller drei Dateien ist in Abbildung 30 dargestellt.

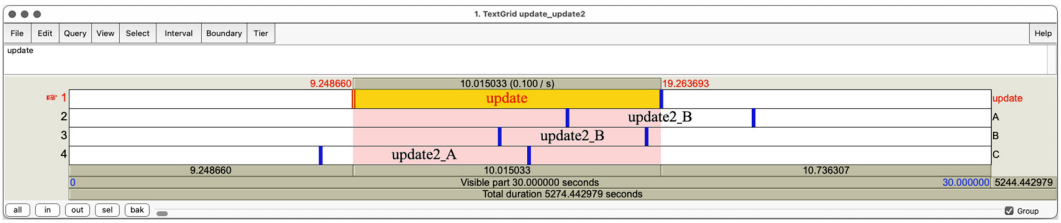


Abbildung 29: Screenshot der .TextGrid-Datei 'update_update2' in *Praat*

[32] Diese Voreinstellungen können angepasst werden. Der Name der Annotationsspur, welche die Markierungen enthält, kann über die Angabe des Parameters `identifierTier` festgelegt werden. Der Annotationstext, mit dem die zu übernehmenden Bereiche markiert werden, kann im Parameter `identifierPattern` angepasst werden, der als regulärer Ausdruck interpretiert wird.

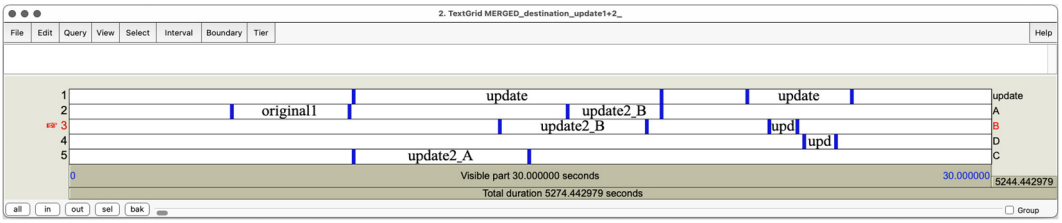


Abbildung 30: Ergebnis der Zusammenführung aller drei Dateien in *Praat*

Das folgende Code-Beispiel illustriert die Zusammenfügung aller drei Annotationsdateien bzw. Transkript-Objekte.

```
corpus2 <- act::transcripts_merge(
             x                        = examplecorpus,
             destinationTranscriptName = 'update_destination',
             updateTranscriptNames     = c('update_update1', 'update_update2'))
View(corpus2@transcripts[['update_destination']]@annotations)
```

Als Parameter `updateTranscriptNames` des Befehls `act::transcripts_merge()` sind nun die Namen zweier Transkript-Objekte angegeben, die mit dem Befehl `c()` zusammengefasst sind.

Um das neu erstellte Transkript-Objekt in eine .TextGrid-Datei zu exportieren, kann der Befehl `act::export_textgrid()` verwendet werden.

```
path <- rstudioapi::selectFile(existing = FALSE)
if(tolower(tools::file_ext(path))! = "textgrid") {
               path <- paste(path, "TextGrid", sep= ".")}
act::export_textgrid(t = corpus2@transcripts[['update_destination']], outputPath = path)
```

In der ersten Zeile des Code-Beispiels wird ein Dateiauswahl-Dialog geöffnet, in dem die Datei gewählt werden muss, in die gespeichert werden soll. Hier kann der Pfad zu einer noch nicht existierenden Datei angegeben werden, die dann erstellt wird. Die zweite Zeile überprüft, ob das Suffix der Zieldatei korrekt ist und fügt ggf. `.TextGrid` an. Mit dem dritten Befehl `act::export_textgrid()` wird das im Parameter `t` angegebene Transkript-Objekt unter dem Zielpfad gespeichert, der im Parameter `outputPath` angegeben ist.

6.8. Erstellen von Filtern

Die meisten Funktionen des *Aligned Corpus Toolkit* werden auf ein gesamtes Korpus-Objekt angewendet, z.B. die Funktionen zur Suche im Korpus, für den Export und das Bearbeiten der Daten. Dabei ist es aber auch möglich, anzugeben, dass die Funktionen lediglich einige Transkript-Objekte oder Annotationsspuren einbeziehen sollen. Hierzu werden die Namen der Transkript-Objekte und/oder Annotationsspuren als Filter angegeben, und zwar in den Parametern `filterTranscriptNames` und/oder `filterTierNames`. Dabei müssen in diesen Parametern die kompletten Namen angegeben werden.

Es ist nun zum einen möglich, die Namen der entsprechenden Transkript-Objekte und/oder Annotationsspuren einzeln zu definieren. Zum anderen können Filter aber auch über die Funktion `act::search_makefilter()` erstellt werden. Diese

Funktion sucht in einem Korpus nach allen Transkript-Objekten und Annotationsspuren, deren Namen einem bestimmten Muster (d.h. einem regulären Ausdruck) entsprechen. Dies ist beispielsweise dann sinnvoll, wenn im Namen der Transkript-Objekte eine bestimmte Sprache oder Region systematisch erfasst ist. Im Beispielkorpus stammen alle Aufnahme mit dem Kürzel 'BOL_' im Dateinamen aus Bolivien, solche mit dem Kürzel 'ARG_' aus Argentinien. Filter für Annotationsspuren anhand des Namens zu erstellen kann beispielsweise sinnvoll sein, um Kommentarzeilen aus der Suche auszuschließen, die alle die Buchstabenkombination "comment" im Namen tragen. Das Ergebnis der Funktion `act::search_makefilter()` sind dann die Namen aller Transkript-Objekte und Annotationsspuren, die den angegebenen Suchkriterien entsprechen. Ein Beispiel hierfür ist das folgende, in dem nach allen Aufnahmen und Annotationsspuren aus Argentinien gesucht wird.

```
filters <- act::search_makefilter(  x                          = corpus,
                                    filterTranscriptIncludeRegEx = "ARG")
filters
filters$transcripts.names
filters$tiers.names
```

Über die erste Code-Zeile wird im Korpus `corpus` nach allen Transkripten gesucht, die die Zeichenkette `ARG` im Transkriptnamen tragen, was im Parameter `filterTranscriptIncludeRegEx` angegeben wird. Die Ausgabe wird im Objekt `filters` gespeichert. Es handelt sich dabei um eine Liste mit den zwei benannten Listenelementen. Im Element `$transcripts.names` werden alle Transkriptnamen gespeichert, die den angegebenen Parametern entsprechen (d.h. deren Namen "ARG" enthält). Im Element `$tiers.names` sind die Namen aller Annotationsspuren zusammengefasst, die in diesen Transkripten enthalten sind.

Der Befehl `act::search_makefilter()` akzeptiert dabei eine ganze Reihe von Parametern, die alle miteinander kombiniert werden. Das Ergebnis ist dann jeweils die Schnittmenge an Transkripten bzw. Annotationsspuren, die allen angegebenen Parametern entsprechen.

```
act::search_makefilter(     x                          = corpus,
                            filterTierIncludeRegEx     = "A")

act::search_makefilter(     x                          = corpus,
                            filterTierNames            = c("A","B"))

act::search_makefilter(     x                          = corpus,
                            filterTranscriptIncludeRegEx = "ARG",
                            filterTierIncludeRegEx     = "A")

act::search_makefilter(     x                          = corpus,
                            filterTranscriptExcludeRegEx = "ARG",
                            filterTierIncludeRegEx     = "A")
```

Mit dem ersten Befehl wird nach Annotationsspuren gesucht, die ein "A" im Namen enthalten. Ausgegeben werden dann deren Namen und die Namen der Transkript-Objekte, in denen diese vorkommen. Der zweite Befehl sucht nach allen Transkript-

Objekten, die Annotationsspuren mit exakt den Namen "A" und "B" enthalten. Im dritten Befehl werden Suchparameter kombiniert. Das Ergebnis sind die Namen der Transkripte und Annotationsspuren, bei denen die Transkripte ARG im Namen enthalten und in denen die Annotationsspur den Buchstaben "A" enthält. Der vierte Befehl kombiniert ebenfalls Parameter. Hier dürfen die Transkriptnamen die Zeichenkette "ARG" *nicht* enthalten (`ExcludeRegEx`), die Annotationsspuren müssen ein "A" im Namen enthalten (`IncludeRegEx`).

7. Anhang: Übersicht der vorgestellten Befehle

Die Reihenfolge der folgenden Nennung der Befehle folgt ihrer Vorstellung in diesem Artikel.

Erstellen eines Korpus-Objekts und Import von Annotationsdateien

`corpus_new()`	Erstellen eines neuen Korpus-Objekts und Importieren von Annotationsdateien	6.1.1.
`annotations_all()`	Erstellen einer Tabelle mit allen Annotationen eines Korpus-Objekts	6.1.4.
`tiers_all()`	Erstellen einer Tabelle mit allen Annotationsspuren eines Korpus-Objekts	6.1.4.
`info()`	Zusammengefasste Informationen über den Inhalt eines Korpus-Objekts	6.1.4.
`media_assign()`	Erstellen von Verknüpfungen zu Medien-Dateien	6.1.5.

Export von Daten

`corpus_export()`	Exportieren von Transkript-Objekten in verschiedenen Annotationsdatei-Formaten, als Drucktranskript und Untertiteldatei.	6.2.

Erstellung von Drucktranskripten

`export_printtranscript()`	Exportieren eines einzelnen Transkript-Objekts als Drucktranskript	6.3.

Suche im Korpus und Arbeiten mit Suchergebnissen

`act::search_new()`	Erstellen einer neuen Suche	6.4.1.
`search_cuts_printtranscript()`	Erstellen von Transkript-Ausschnitten zu einer Suche	6.5.1.1.
`search_cuts_media()`	Erstellen von Medien-Ausschnitten (über *FFmpeg* Schnitt-Befehle)	6.5.1.2.
`act::search_cuts()`	Erstellen von Transkript, Medien- und Untertitelausschnitten.	6.5.1.3.
`search_results_export()`	Exportieren von Suchergebnissen	6.5.2.
`search_results_import()`	Importieren von Suchergebnissen	6.5.2.
`search_sub()`	Kookkurrenz-Suche, ausgehend von einer ersten Suche	6.5.3.
`search_openresult_inelan()`	Öffnen von Suchergebnissen in *ELAN*	6.5.4.1.
`search_openresult_inpraat()`	Öffnen von Suchergebnissen in *Praat*	6.5.4.2.
`search_openresult_inquicktime()`	Öffnen von Suchergebnissen in *QuickTime* (nur *macOS*)	6.5.4.3.

Interoperabilität mit der `rPraat`-Bibliothek

`act::export_rpraat()`	Exportieren eines Transkript-Objekts im Format der `rPraat`-Bibliothek	6.6

Bearbeiten von Daten

`annotations_replace_copy()`	Suchen & Ersetzen im Annotationstext und selektives Kopieren von Annotationen auf andere Annotationsspuren	6.7.1.
`annotations_delete()`	Löschen von Annotationen	6.7.1.

`tiers_add()`	Hinzufügen von Annotationsspuren	6.7.2.
`tiers_sort()`	Sortieren der Annotationsspuren	6.7.2.
`tiers_rename()`	Umbenennen von Annotationsspuren	6.7.2.
`tiers_delete()`	Löschen von Annotationsspuren	6.7.2.
`transcripts_rename()`	Umbenennen von Transkript-Objekten	
`transcripts_delete()`	Löschen von Transkript Objekten	6.7.3.
`info_summarized()`	Kondensierte Informationen über den Inhalt eines Korpus-Objekts	6.7.3.
`transcripts_add()`	Hinzufügen eines bestehenden Transkript-Objekts zu einem Korpus-Objekt	6.7.3.
`transcripts_merge()`	Zusammenführen der Inhalte von mehreren Transkript-Objekten	6.7.4.
`export_textgrid()`	Exportieren eines einzelnen Transkript-Objekts im .TextGrid Format	6.7.4.

Erstellen von Filtern

`search_makefilter()`	Erstellen von Filtern für ein Korpus-Objekt, ausgehend von Suchkriterien	6.8.

8. Links

R und RStudio

R und CRAN	https://cran.r-project.org
RStudio	https://www.rstudio.com/products/rstudio/download/

Annotationswerkzeuge

ELAN	https://archive.mpi.nl/tla/elan/download
EXMARaLDA	https://exmaralda.org/
Folker	http://agd.ids-mannheim.de/folker.shtml
Praat	http://www.praat.org

R-Bibliotheken

`act`	auf CRAN und auf GitHub
`ExmaraldaR`	auf GitHub
`FRelan`	auf GitHub
`phonfieldwork`	auf CRAN und auf GitHub
`rPraat`	auf CRAN und auf GitHub
`textgRid`	auf CRAN

Weitere Ressourcen

ELAN Third Party Resources	https://archive.mpi.nl/tla/elan/thirdparty
FFmpeg	https://www.ffmpeg.org
Sendpraat	https://www.fon.hum.uva.nl/praat/sendpraat.html
Transformer	http://www.romanistik.uni-freiburg.de/ehmer/transformer/

9. Literatur

Baayen, Rolf H. (2009): Analyzing linguistic data: A practical introduction to statistics using R, Cambridge: Cambridge University Press.

Barth-Weingarten, Dagmar (2016): Intonation Units Revisited. Cesuras in talk-in-interaction, Amsterdam: Benjamins.

Barth-Weingarten, Dagmar / Ogden, Richard (Hgg.) (2021): Weak cesuras: what fuzzy boundaries can accomplish in talk-in-interaction, Special Issue in Open Linguistics 7 (1).

Boersma, Paul / Weenink, David (2021): Praat: doing phonetics by computer [Computer program]. Version 6.1.40, retrieved 27 February 2021 from http://www.praat.org/.

Bořil, Tomáš / Skarnitzl, Radek (2016): Tools rPraat and mPraat. In: Sojka, Petr / Horák, Aleš / Kopeček, Ivan / Pala, Karel (Hgg.): Text, Speech, and Dialogue, Cham: Springer International Publishing, 367-374.

Cresti, Emanuela / Moneglia, Massimo (2005): C-ORAL-ROM: Integrated Reference Corpora for Spoken Romance Languages, Amsterdam / Philadelphia: Benjamins.

Ehmer, Oliver (2021a): act: Aligned Corpus Toolkit, R package version 1.1.9. https://CRAN.R-project.org/package=act.

Ehmer, Oliver (2021b): Synchronization in demonstrations. Multimodal practices for instructing body knowledge, Linguistics Vanguard 7 (s4), 1-18.

Ehmer, Oliver (2022): Makrokonstruktionen. Komplexe Adverbialstrukturen zwischen lokaler Emergenz und Sedimentierung im gesprochenen Französisch, Berlin/Boston: de Gruyter.

Ehmer, Oliver / Satti, Luis Ignacio / Martínez, Angelita / Pfänder, Stefan (2019): Un sistema para transcribir el habla en la interacción: GAT 2, Gesprächsforschung - Online-Zeitschrift zur verbalen Interaktion 20, 64-114.

Gilles, Peter (2001): prosoDB: Eine multimediale Datenbankumgebung für konversationelle und prosodische Analysen, Gesprächsforschung – Online-Zeitschrift zur verbalen Interaktion (2), 75-89.

Gries, Stefan Th. (2009): Quantitative Corpus Linguistics with R. A Practical Introduction.

Lanwer, Jens Philipp (2020): Appositive Syntax oder appositive Prosodie? In: Imo, Wolfgang / Lanwer, Jens Philipp (Hgg.): Prosodie und Konstruktionsgrammatik, Berlin: De Gruyter, 233-281.

Luginbühl, Martin / Mundwiler, Vera / Kreuz, Judith / Müller-Feldmeth, Daniel / Hauser, Stefan (2021): "Quantitative and Qualitative Approaches in Conversation Analysis: Methodological Reflections on a Study of Argumentative Group Discussions", Gesprächsforschung - Online-Zeitschrift zur verbalen Interaktion 22, 179-236.

Mochet, Marie-Anne / Wittig, Gilberte (1984): Français des années 80. Entretiens sociolinguistiques,
http://clapi.univ-lyon2.fr: Groupe ICOR / Plateforme CLAPI.

R Core Team (2021): R: A language and environment for statistical computing, Vienna, Austria: R Foundation for Statistical Computing,
https://www.R-project.org/.

Rühlemann, Christoph (2020): Visual Linguistics with R. A practical introduction to quantitative Interactional Linguistics, Amsterdam: Benjamins.

Schmidt, Thomas (2002): "Gesprächstranskription auf dem Computer: das System EXMARaLDA", Gesprächsforschung 3, 1-23.

Schmidt, Thomas (2012): EXMARaLDA and the FOLK tools – two toolsets for transcribing and annotating spoken language. In: Declerck, Thierry / Choukri, Khalid / Calzolari, Nicoletta (Hgg.): Proceedings of the Eighth conference on International Language Resources and Evaluation (LREC'12), 236-240.

Schmidt, Thomas (2016): Construction and Dissemination of a Corpus of Spoken Interaction - Tools and Workflows in the FOLK project, Corpus Linguistic Software Tools, Journal for Language Technology and Computational Linguistics (JLCL 31/1), Kupietz, Marc / Geyken, Alexander (Hgg.), 127-154.

Schmidt, Thomas (2017): DGD – Die Datenbank für Gesprochenes Deutsch. Mündliche Korpora am Institut für Deutsche Sprache (IDS) in Mannheim, Zeitschrift für Germanistische Linguistik 45 (3), 451-463.

Schmidt, Thomas / Schütte, Wilfried (2010): FOLKER. An annotation tool for efficient transcription of natural, multi-party interaction. In: Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC 10), may 19-21, 2010, Valletta, Malta. European Language Resources Association (ELRA). Calzolari, Nicoletta/ Choukri, Khalid / Maegaard, Bente / Mariani, Joseph / Odjik, Jan / Piperidis, Stelios / Rosner, Mike / Tapias, Daniel (Hgg.), Valletta, Malta: European Language Resources Association (ELRA), 2091-2096.

Selting, Margret / Auer, Peter / Barth-Weingarten, Dagmar / Bergmann, Jörg / Bergmann, Pia / Birkner, Karin / Couper-Kuhlen, Elizabeth / Deppermann, Arnulf / Gilles, Peter / Günthner, Susanne / Hartung, Martin / Kern, Friederike / Mertzlufft, Christine / Meyer, Christian / Morek, Miriam / Oberzaucher, Frank / Peters, Jörg / Quasthoff, Uta / Schütte, Wilfried / Stukenbrock, Anja / Uhmann, Susanne (2009): "Gesprächsanalytisches Transkriptionssystem 2 (GAT 2)", Gesprächsforschung – Online-Zeitschrift zur verbalen Interaktion 10, 353-402.

Wittenburg, Peter / Brugman, Hennie / Russel, Albert / Klassmann, Alex / Sloetjes, Han (2006): ELAN: a Professional Framework for Multimodality Research. In: Calzolari, Nicoletta / Choukri, Khalid / Gangemi, Aldo / Maegaard, Bente / Mariani, Joseph / Odijk, Jan / Tapias, Daniel (Hgg.): Proceedings of LREC 2006, Fifth International Conference on Language Resources and Evaluation, 1556-1559.

Prof. Dr. Oliver Ehmer
Universität Osnabrück
Institut für Romanistik und Latinistik
Neuer Graben 40
49074 Osnabrück

oliver.ehmer@uos.de | http://www.oliverehmer.de

Veröffentlicht am 11.6.2023