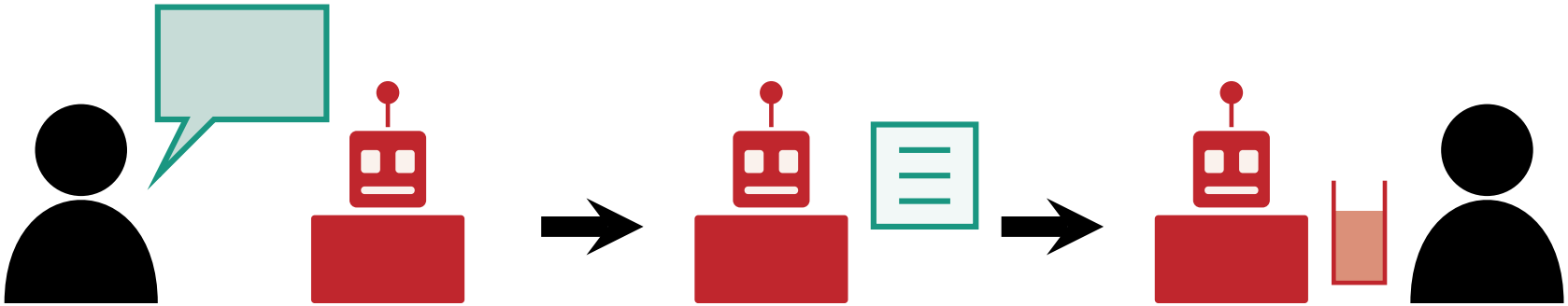
Karlsruher Institut für Technologie

Masterarbeit: Zielsystemunabhängige Quelltextsynthese aus natürlicher Sprache

Viktor Kiesel, betreut von Sebastian Weigelt

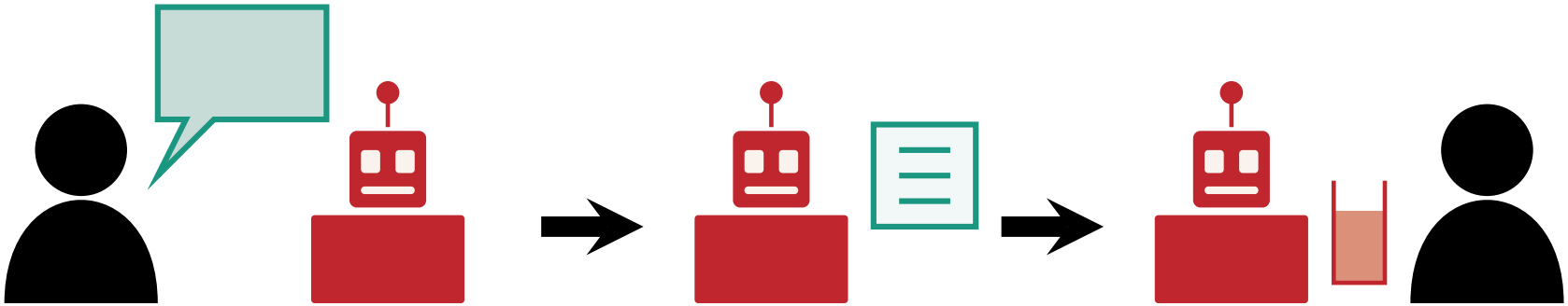
IPD Tichy, Fakultät für Informatik

Motivation



- Probleme:
  - Sprache ist oft mehrdeutig
  - Verständnis von Kontext und implizitem Wissen abhängig
  - $\Longrightarrow$ Für Maschienen schwer verständlich

Motivation



- Probleme:
  - Sprache ist oft mehrdeutig
  - Verständnis von Kontext und implizitem Wissen abhängig
  - $\Longrightarrow$ Für Maschienen schwer verständlich

Zielsetzung und Anforderungen

- Zielsetzung:
  - Quelltextsynthese aus natürlicher Sprache
  - Zielsystemunabhängigkeit
    - Minimierung des Einflusses des Zielsystems
    - Auf andere Zielsysteme transferierbar

- Anforderungen:
  - Extrahierte Sprachinformationen
  - Zielsystemunabhängigkeit vorheriger Werkzeuge

Zielsetzung und Anforderungen

- Zielsetzung:
  - Quelltextsynthese aus natürlicher Sprache
  - Zielsystemunabhängigkeit
    - Minimierung des Einflusses des Zielsystems
    - Auf andere Zielsysteme transferierbar

- Anforderungen:
  - Extrahierte Sprachinformationen
  - Zielsystemunabhängigkeit vorheriger Werkzeuge

Verwandte Arbeiten

- Quelltextsynthese:
  - Sprachverarbeitungsfließbänder [12] [7] [5]
  - Neuronale Netze [11] [8]
  - Robotersteuerung [9] [4] [10] [6]
  - Abstrakte Syntaxbäume [15] [13]
- Einschränkungen...
  - ...in der Eingabesprache (Formulierung, Komplexität)
  - ...im Quelltext (Befehlssatz, Kontrollstrukturen)
  - ...in der Domäne
  - ...im Zielsystem
- $\Longrightarrow$ Keine Quelltextsynthese aus natürlicher Sprache ohne Einschränkungen
- Programmtransformation [2] [3]

Verwandte Arbeiten

- Quelltextsynthese:
  - Sprachverarbeitungsfließbänder [12] [7] [5]
  - Neuronale Netze [11] [8]
  - Robotersteuerung [9] [4] [10] [6]
  - Abstrakte Syntaxbäume [15] [13]
- Einschränkungen...
  - ...in der Eingabesprache (Formulierung, Komplexität)
  - ...im Quelltext (Befehlssatz, Kontrollstrukturen)
  - ...in der Domäne
  - ...im Zielsystem
- $\Longrightarrow$ Keine Quelltextsynthese aus natürlicher Sprache ohne Einschränkungen
- Programmtransformation [2] [3]

Verwandte Arbeiten

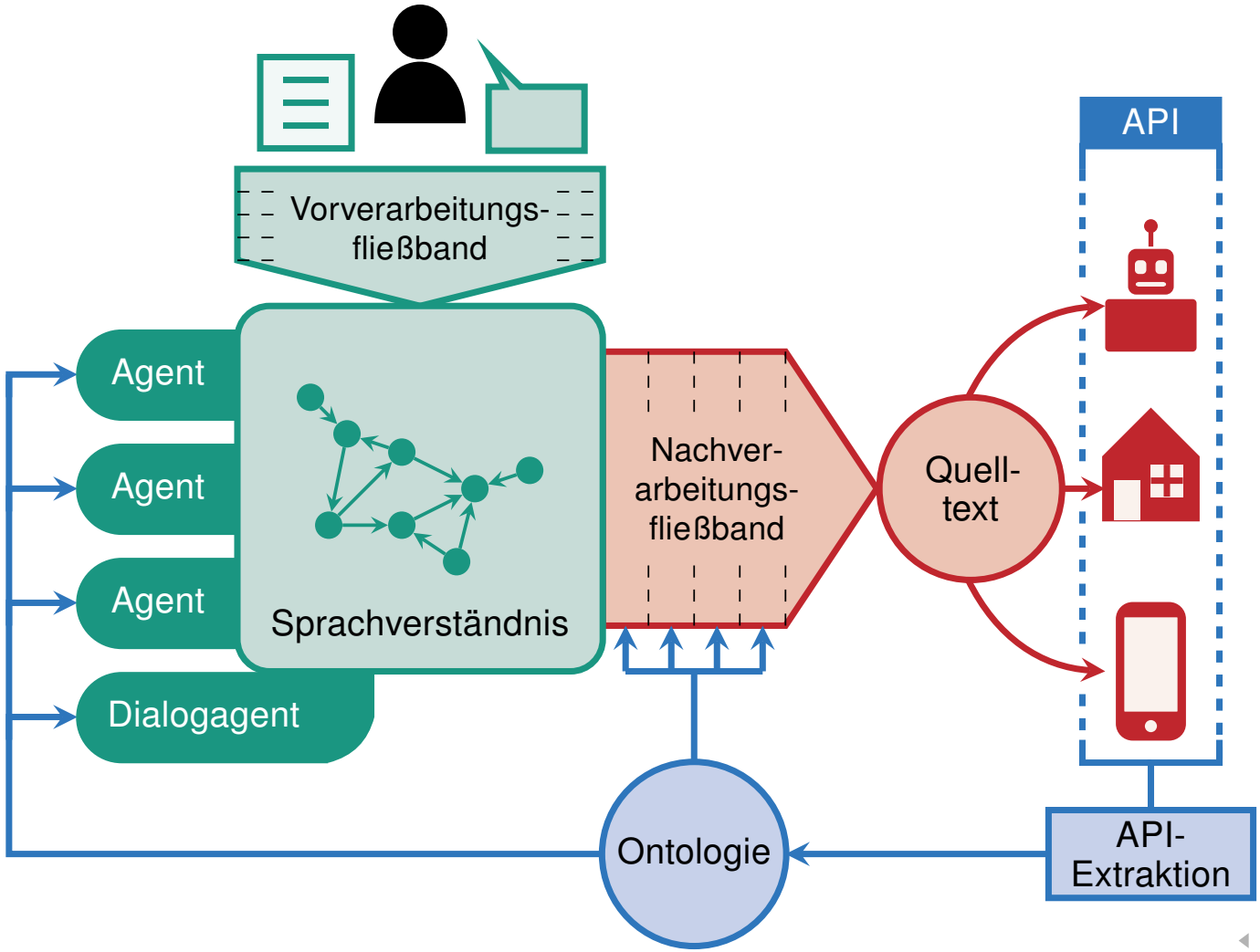
- Quelltextsynthese:
  - Sprachverarbeitungsfließbänder [12] [7] [5]
  - Neuronale Netze [11] [8]
  - Robotersteuerung [9] [4] [10] [6]
  - Abstrakte Syntaxbäume [15] [13]
- Einschränkungen...
  - ...in der Eingabesprache (Formulierung, Komplexität)
  - ...im Quelltext (Befehlssatz, Kontrollstrukturen)
  - ...in der Domäne
  - ...im Zielsystem
- $\Longrightarrow$ Keine Quelltextsynthese aus natürlicher Sprache ohne Einschränkungen
- Programmtransformation [2] [3]

Verwandte Arbeiten

- Quelltextsynthese:
  - Sprachverarbeitungsfließbänder [12] [7] [5]
  - Neuronale Netze [11] [8]
  - Robotersteuerung [9] [4] [10] [6]
  - Abstrakte Syntaxbäume [15] [13]
- Einschränkungen...
  - ...in der Eingabesprache (Formulierung, Komplexität)
  - ...im Quelltext (Befehlssatz, Kontrollstrukturen)
  - ...in der Domäne
  - ...im Zielsystem
- $\Longrightarrow$ Keine Quelltextsynthese aus natürlicher Sprache ohne Einschränkungen
- Programmtransformation [2] [3]

PARSE

- Rahmenarchitektur für Sprachverarbeitung [14]

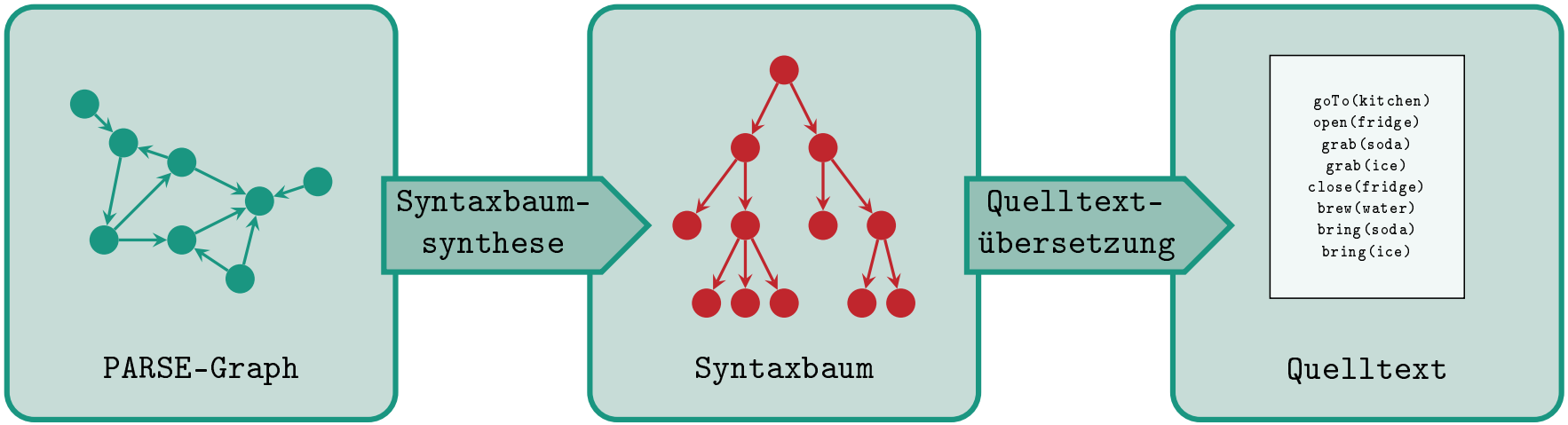


Analyse: Ansatz

- Abstrakter Syntaxbaum als Zwischenrepräsentation:
  - Lässt sich als Graph repräsentieren
  - Abstrakte Repräsentation von Quelltext
- Vorgehen:
  1. PARSE-Graph in abstrakten Syntaxbaum überführen
  2. Syntaxbaum wird in konkreten Quelltext für Zielsystem übersetzt

Analyse: Ansatz

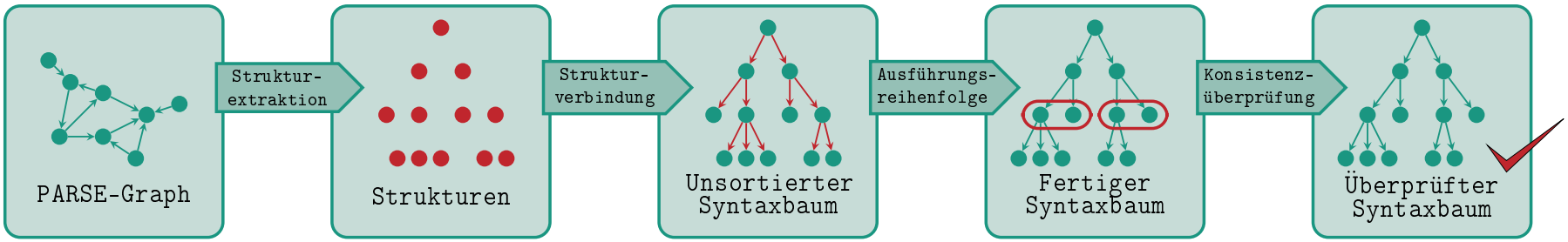
- Abstrakter Syntaxbaum als Zwischenrepräsentation:
  - Lässt sich als Graph repräsentieren
  - Abstrakte Repräsentation von Quelltext
- Vorgehen:
  1. PARSE-Graph in abstrakten Syntaxbaum überführen
  2. Syntaxbaum wird in konkreten Quelltext für Zielsystem übersetzt



Syntaxbaumextraktion: Ansatz

- **Ziel:** Synthese eines abstrakten Syntaxbaumes aus PARSE-Graphen

1. Extrahiere alle Strukturen (Quelltextbausteine) des Syntaxbaumes
2. Verbinde gefundenen Strukturen zu Syntaxbaum
3. Lege Ausführungsreihenfolge der Strukturen fest
4. Überprüfe Syntaxbaum auf Konsistenz

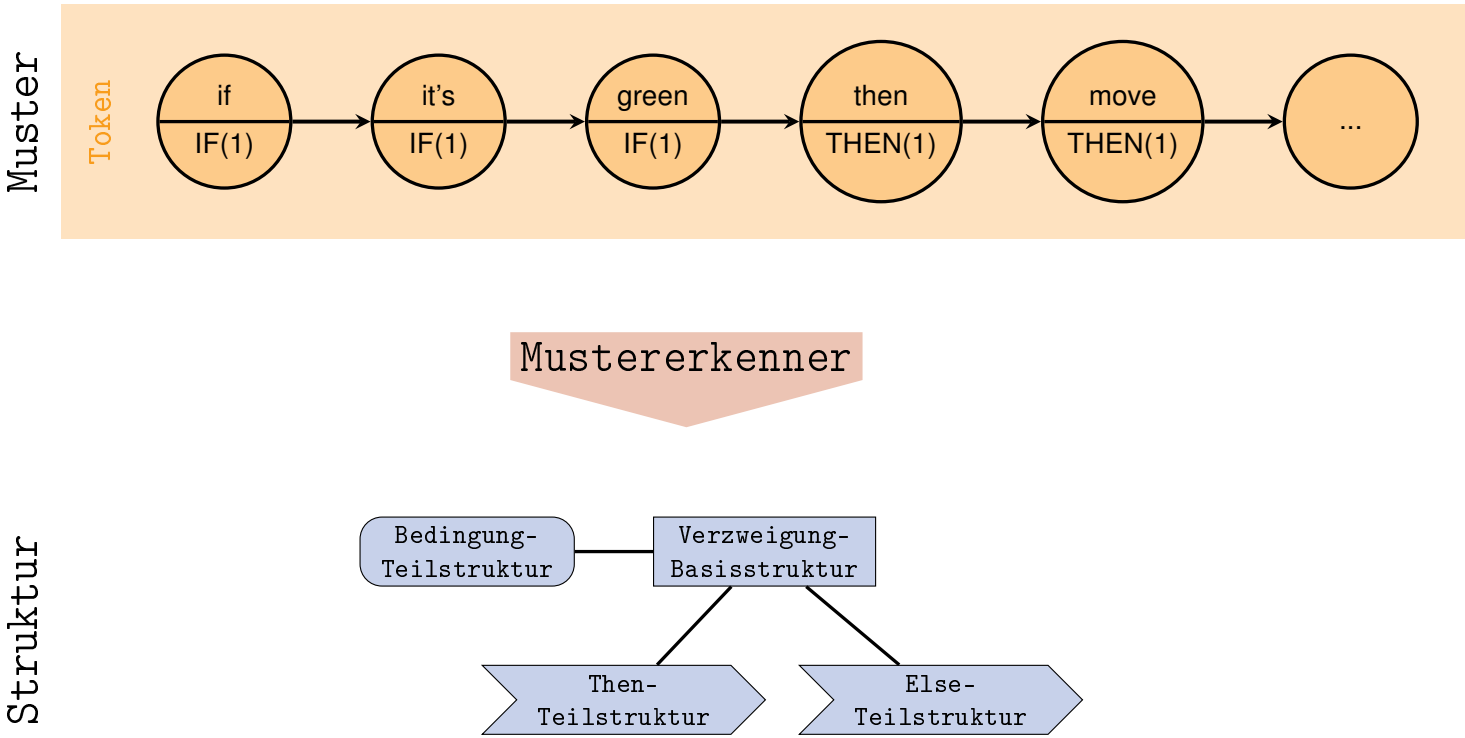


Syntaxbaumextraktion: Mustererkennung

- Mustererkennung: Durchsuche Graph auf Muster (d.h. Regelmäßigkeiten) und führe Aktion aus

Syntaxbaumextraktion: Mustererkennung

- Mustererkennung: Durchsuche Graph auf Muster (d.h. Regelmäßigkeiten) und führe Aktion aus



Syntaxbaumextraktion: Verbindung der Strukturen

- Strukturen nach der Mustererkennung separiert
- Syntaxbaum: Strukturen müssen verbunden werden
- Vorgehen:
  1. Baue aus den Strukturen alle möglichen vollständigen Pfade auf
  2. Verbinde Pfade zu Syntaxbaum

Syntaxbaumextraktion: Verbindung der Strukturen

- Strukturen nach der Mustererkennung separiert
- Syntaxbaum: Strukturen müssen verbunden werden
- Vorgehen:
  1. Baue aus den Strukturen alle möglichen vollständigen Pfade auf
  2. Verbinde Pfade zu Syntaxbaum

Syntaxbaumextraktion: Aufbau von Pfaden

„*If the dishwasher is full* *then start it* *else fill it with dirty dishes*“

- **Idee:** Teilsätze implizieren Strukturen
- Teilsatz impliziert unterschiedliche Strukturen → Beziehung zwischen den Strukturen
- Vorgehen:
  1. Sammle alle vom selben Teilsatz implizierte Strukturen
  2. Sortiere Strukturen eines Pfades anhand ihrer Wortanzahl
     - Spezifische Struktur: Nahe Blätter
     - Allgemeine Struktur: Nahe Wurzel

Syntaxbaumextraktion: Aufbau von Pfaden

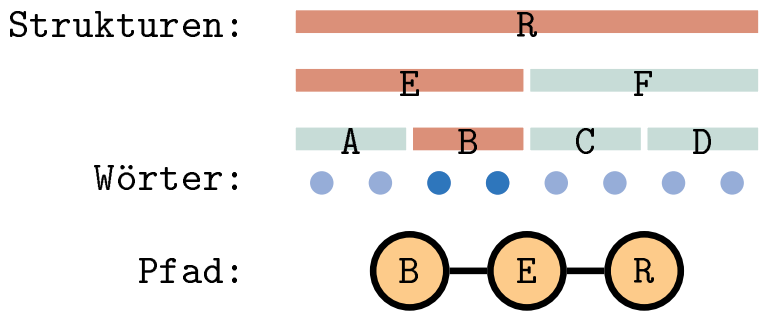
„ *If the dishwasher is full* *then start it* *else fill it with dirty dishes* “

- **Idee:** Teilsätze implizieren Strukturen
- Teilsatz impliziert unterschiedliche Strukturen → Beziehung zwischen den Strukturen
- Vorgehen:
  1. Sammle alle vom selben Teilsatz implizierte Strukturen
  2. Sortiere Strukturen eines Pfades anhand ihrer Wortanzahl
     - Spezifische Struktur: Nahe Blätter
     - Allgemeine Struktur: Nahe Wurzel

Syntaxbaumextraktion: Aufbau von Pfaden

„ *If the dishwasher is full* *then start it* *else fill it with dirty dishes* “

- **Idee:** Teilsätze implizieren Strukturen
- Teilsatz impliziert unterschiedliche Strukturen → Beziehung zwischen den Strukturen
- Vorgehen:
  1. Sammle alle vom selben Teilsatz implizierte Strukturen
  2. Sortiere Strukturen eines Pfades anhand ihrer Wortanzahl
     - Spezifische Struktur: Nahe Blätter
     - Allgemeine Struktur: Nahe Wurzel

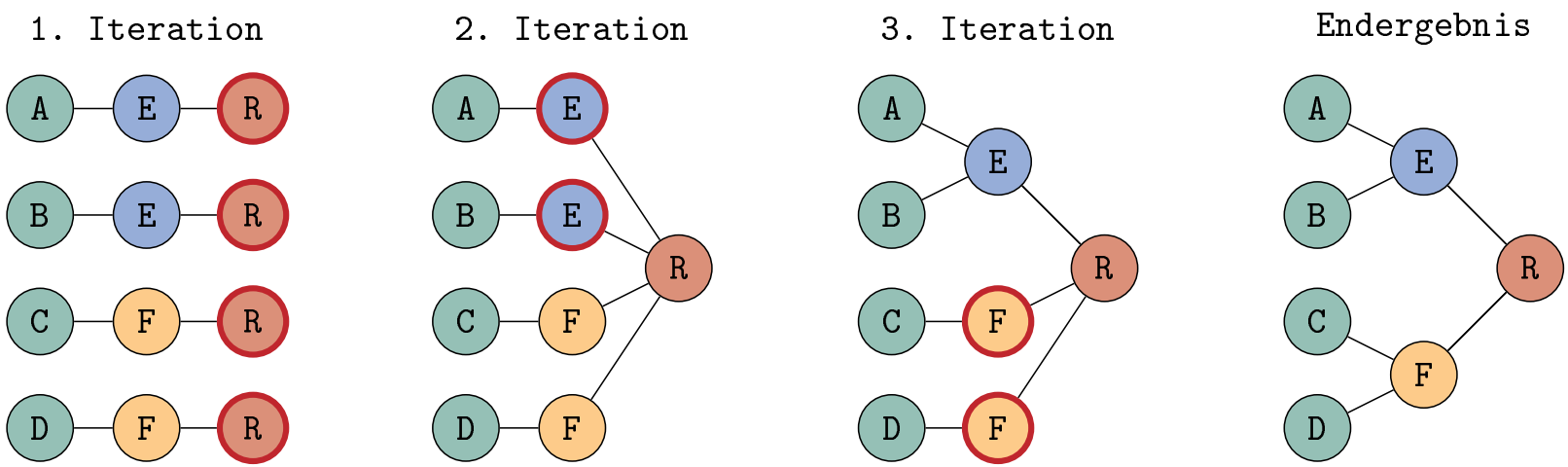


Syntaxbaumextraktion: Verbindung der Pfade

- Identische Strukturen kommen in verschiedenen Pfaden vor
  - Wurzel in jedem Pfad vorhanden
  - Aktionen/Blätter einzigartig
- Verschmelzen von identischen Strukturen führt zum Verschmelzen von Pfaden
- Wiederhole bis alle Pfade verschmolzen und Syntaxbaum erzeugt
- Mehrdeutigkeit: Mehrheitswahl

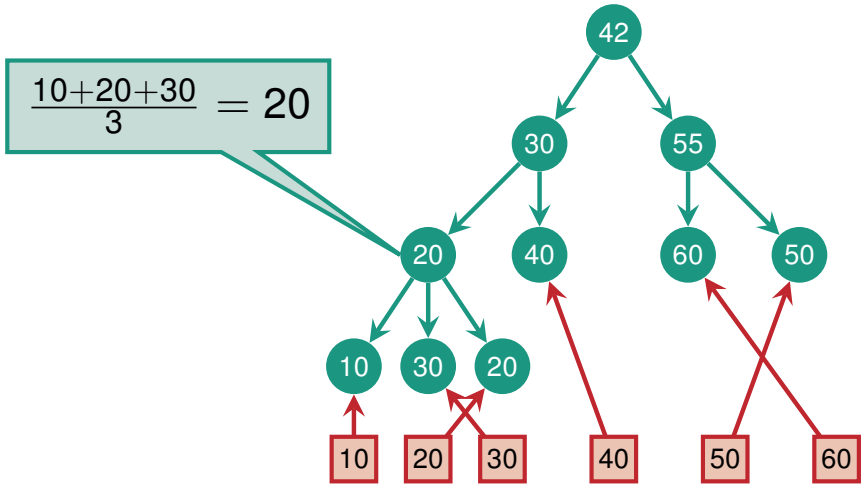
Syntaxbaumextraktion: Verbindung der Pfade

- Identische Strukturen kommen in verschiedenen Pfaden vor
  - Wurzel in jedem Pfad vorhanden
  - Aktionen/Blätter einzigartig
- Verschmelzen von identischen Strukturen führt zum Verschmelzen von Pfaden
- Wiederhole bis alle Pfade verschmolzen und Syntaxbaum erzeugt
- Mehrdeutigkeit: Mehrheitswahl



Syntaxbaumextraktion: Ausführungsreihenfolge

- Aufgebauter Baum enthält nur Elter-Kind-Beziehungen
- Für Geschwister-Strukturen muss eine Ausführungsreihenfolge festgelegt werden
- Vorgehen:
  1. Blätter übernehmen Position des Teilsatzes
  2. Elter-Strukturen übernehmen iterativ durch Mitteln Position von Kind-Strukturen



Syntaxbaumextraktion: Überprüfung des Baumes

- Syntaxbaum vollständig aufgebaut $\rightarrow$ Übersetzbar in Quelltext
- Kann Fehler enthalten
- Semantische Überprüfung: Andere PARSE-Agenten
- Daher nur syntaktische Überprüfung:
  - Fehlende Aktionen
- Unterscheidung zwischen Fehlern und Warnungen
  - Fehler: Keine Übersetzung des Syntaxbaumes in Quelltext möglich
  - Warnung: Möglicher Fehler, Syntaxbaum lässt sich aber in Quelltext übersetzen
- Dialogagent kann gefundene Fehler durch Nutzer korrigieren lassen

Syntaxbaumextraktion: Überprüfung des Baumes

- Syntaxbaum vollständig aufgebaut $\rightarrow$ Übersetzbar in Quelltext
- Kann Fehler enthalten
- Semantische Überprüfung: Andere PARSE-Agenten
- Daher nur syntaktische Überprüfung:
  - Fehlende Aktionen
- Unterscheidung zwischen Fehlern und Warnungen
  - Fehler: Keine Übersetzung des Syntaxbaumes in Quelltext möglich
  - Warnung: Möglicher Fehler, Syntaxbaum lässt sich aber in Quelltext übersetzen
- Dialogagent kann gefundene Fehler durch Nutzer korrigieren lassen

Syntaxbaumextraktion: Überprüfung des Baumes

- Syntaxbaum vollständig aufgebaut $\rightarrow$ Übersetzbar in Quelltext
- Kann Fehler enthalten
- Semantische Überprüfung: Andere PARSE-Agenten
- Daher nur syntaktische Überprüfung:
  - Fehlende Aktionen
- Unterscheidung zwischen Fehlern und Warnungen
  - Fehler: Keine Übersetzung des Syntaxbaumes in Quelltext möglich
  - Warnung: Möglicher Fehler, Syntaxbaum lässt sich aber in Quelltext übersetzen
- Dialogagent kann gefundene Fehler durch Nutzer korrigieren lassen

Syntaxbaumextraktion: Überprüfung des Baumes

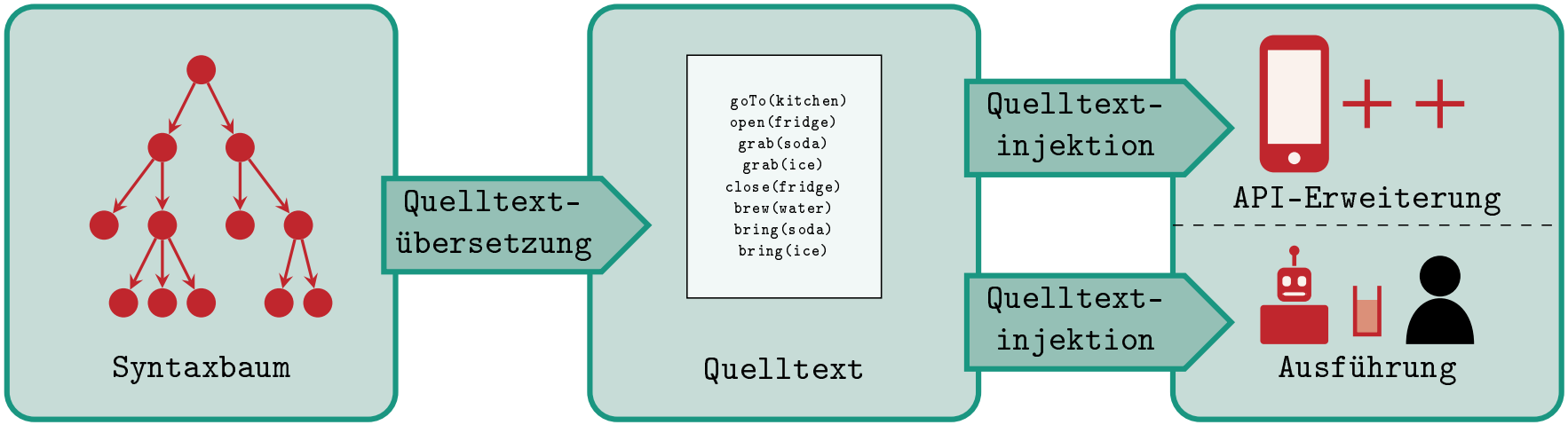
- Syntaxbaum vollständig aufgebaut $\rightarrow$ Übersetzbar in Quelltext
- Kann Fehler enthalten
- Semantische Überprüfung: Andere PARSE-Agenten
- Daher nur syntaktische Überprüfung:
  - Fehlende Aktionen
- Unterscheidung zwischen Fehlern und Warnungen
  - Fehler: Keine Übersetzung des Syntaxbaumes in Quelltext möglich
  - Warnung: Möglicher Fehler, Syntaxbaum lässt sich aber in Quelltext übersetzen
- Dialogagent kann gefundene Fehler durch Nutzer korrigieren lassen

Quelltextübersetzung: Vorgehen

- Quelltext abhängig vom Zielsystem
- Ziele:
  - Übersetzung des Syntaxbaumes zu konkretem Quelltext
  - Einfacher Transfer auf andere Zielsysteme
- Vorgehen:
  1. Übersetze Syntaxbaum in Quelltext einer konkreten Programmiersprache
  2. Füge erzeugten Quelltext in Ausführungsschablone des Zielsystems ein

Quelltextübersetzung: Vorgehen

- Quelltext abhängig vom Zielsystem
- Ziele:
  - Übersetzung des Syntaxbaumes zu konkretem Quelltext
  - Einfacher Transfer auf andere Zielsysteme
- Vorgehen:
  1. Übersetze Syntaxbaum in Quelltext einer konkreten Programmiersprache
  2. Füge erzeugten Quelltext in Ausführungsschablone des Zielsystems ein



Quelltextübersetzung: Quelltexterzeugung

- Syntaxbäume sind abstrakte hierarchische Repräsentation von Quelltext
- $\Longrightarrow$ Syntaxbäume sind äquivalent zu Quelltext
- Durch rekursives Ablaufen des Syntaxbaumes kann Syntaxbaum in Quelltext übersetzt werden
- Besucher hierfür geeignet
  - Methoden des Besuchers spiegeln Strukturen des Syntaxbaums wieder
  - Besucher nur von Programmiersprache abhängig
  - Besucher lassen sich für Zielsysteme mit gleicher Programmiersprache wiederverwenden
  - Programmiersprachen mit ähnlicher Syntax lassen sich durch Vererbung bündeln
- Es wurden Besucher für Java, Python, C und PlantUML implementiert

Quelltextübersetzung: Quelltexterzeugung

- Syntaxbäume sind abstrakte hierarchische Repräsentation von Quelltext
- $\Longrightarrow$ Syntaxbäume sind äquivalent zu Quelltext
- Durch rekursives Ablaufen des Syntaxbaumes kann Syntaxbaum in Quelltext übersetzt werden
- Besucher hierfür geeignet
  - Methoden des Besuchers spiegeln Strukturen des Syntaxbaums wieder
  - Besucher nur von Programmiersprache abhängig
  - Besucher lassen sich für Zielsysteme mit gleicher Programmiersprache wiederverwenden
  - Programmiersprachen mit ähnlicher Syntax lassen sich durch Vererbung bündeln
- Es wurden Besucher für Java, Python, C und PlantUML implementiert

Quelltextübersetzung: Quelltexterzeugung

- Syntaxbäume sind abstrakte hierarchische Repräsentation von Quelltext
- $\Longrightarrow$ Syntaxbäume sind äquivalent zu Quelltext
- Durch rekursives Ablaufen des Syntaxbaumes kann Syntaxbaum in Quelltext übersetzt werden
- Besucher hierfür geeignet
  - Methoden des Besuchers spiegeln Strukturen des Syntaxbaums wieder
  - Besucher nur von Programmiersprache abhängig
  - Besucher lassen sich für Zielsysteme mit gleicher Programmiersprache wiederverwenden
  - Programmiersprachen mit ähnlicher Syntax lassen sich durch Vererbung bündeln
- Es wurden Besucher für Java, Python, C und PlantUML implementiert

Erzeugter Python-Quelltext

- *„hey armar check if the dish is dirty then wash the dish twice otherwise read the news for me“*

```
if (isDirty(Dishes)):
        for x in range(2):
                # then wash the dish twice
                wash(Dishes)

else :
        # otherwise read the news for me
        read(News)
```

Erzeugter C-Quelltext

- *„hey armar check if the dish is dirty then wash the dish twice after that get me orange juice while reading the news for me“*

```
if (isDirty(Dishes)) {
    for(int iter0=0;iter0 < 2; iter0++) {
        wash(Dishes);
    }
} else {}
#pragma omp parallel sections
{
    #pragma omp section
    {
        get1(OrangeJuice);
    }
    #pragma omp section
    {
        read(News);
    }
}
```

Erzeugter Java-Quelltext I

- *„hey armar check if the dish is dirty then wash the dish twice after that get me orange juice while reading the news for me“*

```
if (isDirty(Dishes)) {
    for(int iter0=0;iter0 < 2; iter0++) {
        wash(Dishes);
    }
} else {}
Thread t0= new Thread() { public void run() {
    get1(OrangeJuice);
}};
t0.start();
Thread t1= new Thread() { public void run() {
    read(News);
}};
t1.start();
t0.join();
t1.join();
```

Quelltextübersetzung: Quelltextinjektion

- Probleme:
  - Zielsystem kann Vorgaben an die Ausführungsdatei stellen (Main()- vs Execute()-Methode)
  - Quelltextausführung oder API-Erweiterung
- $\Longrightarrow$ Trennung von Quelltexterzeugung und Ausführungsdatei
- Schablonen-System verwendet, um erzeugten Quelltext mit Ausführungsdatei zu kombinieren

Quelltextübersetzung: Quelltextinjektion

- Probleme:
  - Zielsystem kann Vorgaben an die Ausführungsdatei stellen (Main()- vs Execute()-Methode)
  - Quelltextausführung oder API-Erweiterung
- $\Longrightarrow$ Trennung von Quelltexterzeugung und Ausführungsdatei
- Schablonen-System verwendet, um erzeugten Quelltext mit Ausführungsdatei zu kombinieren

Evaluation

- PARSE insgesamt evaluiert
  - $\Longrightarrow$ Fehler vorheriger Werkzeugen nicht zu beseitigen
- Quelltext mehrdeutig: Automatische Evaluation nicht möglich
- Evaluation auf zwei Arten:
  - Online-Studie mit PlantUML-Aktivitätsdiagrammen
  - Manuelle Evaluation der Methoden
- PARSE-Korpus [1]:
  - 238 Befehlsbeschreibungen für den Küchenroboter ARMAR-III
  - 13 Szenarios unterschiedlicher Komplexität und Länge
  - Enthält Kontrollstrukturen und Disfluenzen

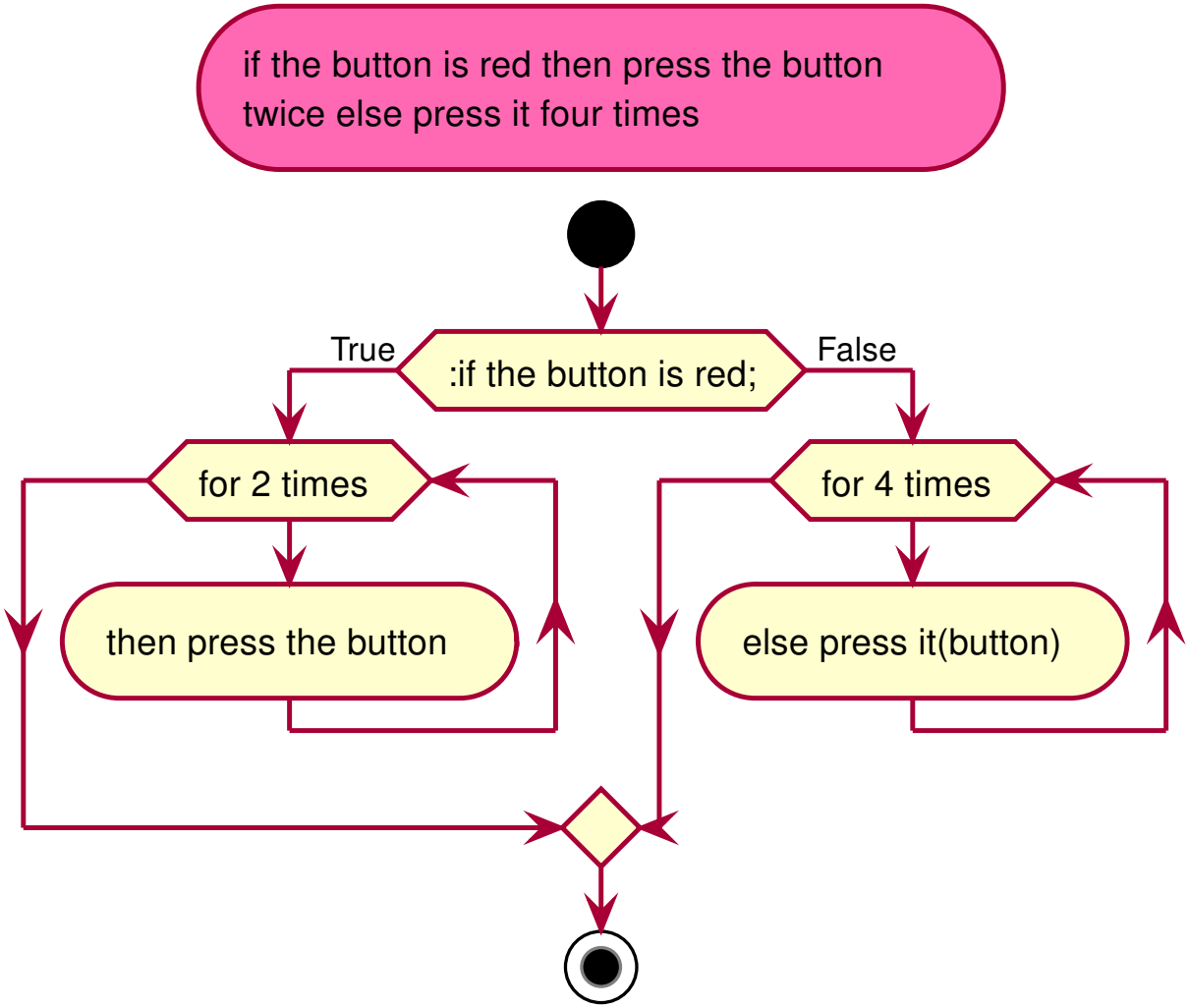
Evaluation

- PARSE insgesamt evaluiert
  - $\Longrightarrow$ Fehler vorheriger Werkzeugen nicht zu beseitigen
- Quelltext mehrdeutig: Automatische Evaluation nicht möglich
- Evaluation auf zwei Arten:
  - Online-Studie mit PlantUML-Aktivitätsdiagrammen
  - Manuelle Evaluation der Methoden
- PARSE-Korpus [1]:
  - 238 Befehlsbeschreibungen für den Küchenroboter ARMAR-III
  - 13 Szenarios unterschiedlicher Komplexität und Länge
  - Enthält Kontrollstrukturen und Disfluenzen

Evaluation

- PARSE insgesamt evaluiert
  - $\Longrightarrow$ Fehler vorheriger Werkzeugen nicht zu beseitigen
- Quelltext mehrdeutig: Automatische Evaluation nicht möglich
- Evaluation auf zwei Arten:
  - Online-Studie mit PlantUML-Aktivitätsdiagrammen
  - Manuelle Evaluation der Methoden
- PARSE-Korpus [1]:
  - 238 Befehlsbeschreibungen für den Küchenroboter ARMAR-III
  - 13 Szenarios unterschiedlicher Komplexität und Länge
  - Enthält Kontrollstrukturen und Disfluenzen

Aktivitätsdiagramm



Evaluation: Aufbau der Online-Studie

- 237 Aktivitätsdiagramme mit natürlich sprachlichen Aktivitäten
- Formular über *prolific.co*
  - Einleitung bringt Probanden auf gleiches Wissensniveau
  - Insgesamt 24 Formulare
  - Zehn zufällig ausgewählte Aktivitätsdiagramme pro Formular
  - Proband soll Fehlerfälle ankreuzen
  - 5 Probanden pro Formular
- Abgaben manuell überprüft
  - 19 Abgaben aussortiert
  - 120 Abgaben akzeptiert
- Inter-Annotator-Übereinstimmung: $\kappa = 0.32 (\sigma = 0.10)$
- $\Longrightarrow$ Konkretes Ergebnis: Mehrheitwahl

Evaluation: Aufbau der Online-Studie

- 237 Aktivitätsdiagramme mit natürlich sprachlichen Aktivitäten
- Formular über *prolific.co*
  - Einleitung bringt Probanden auf gleiches Wissensniveau
  - Insgesamt 24 Formulare
  - Zehn zufällig ausgewählte Aktivitätsdiagramme pro Formular
  - Proband soll Fehlerfälle ankreuzen
  - 5 Probanden pro Formular
- Abgaben manuell überprüft
  - 19 Abgaben aussortiert
  - 120 Abgaben akzeptiert
- Inter-Annotator-Übereinstimmung: $\kappa = 0.32 (\sigma = 0.10)$
- $\Longrightarrow$ Konkretes Ergebnis: Mehrheitwahl

Evaluation: Aufbau der Online-Studie

- 237 Aktivitätsdiagramme mit natürlich sprachlichen Aktivitäten
- Formular über *prolific.co*
  - Einleitung bringt Probanden auf gleiches Wissensniveau
  - Insgesamt 24 Formulare
  - Zehn zufällig ausgewählte Aktivitätsdiagramme pro Formular
  - Proband soll Fehlerfälle ankreuzen
  - 5 Probanden pro Formular
- Abgaben manuell überprüft
  - 19 Abgaben aussortiert
  - 120 Abgaben akzeptiert
- Inter-Annotator-Übereinstimmung: $\kappa = 0.32 (\sigma = 0.10)$
- $\Longrightarrow$ Konkretes Ergebnis: Mehrheitwahl

Evaluation: Online-Studie - Ergebnisse I

Fehlerfall	Mehrheit
Vollständig korrekt	113 (47,68%)
Anweisung inkorrekt	21 (8,86%)
Fehlende Anweisungen	3 (1,27%)
Anweisungen zuviel	9 (3,80%)
Reihenfolge falsch	5 (2,11%)
Koreferenz inkorrekt	10 (4,22%)
Verzweigung inkorrekt	3 (1,27%)
Schleife fehlerhaft	4 (1,69%)
Parallelität falsch	0
Summe	168

Diagramme	Mehrheit
Korrekt	113 (47,68%)
Fehlerhaft	50 (21,10%)
Bestimmt	163 (68,78%)
Unbestimmt	74 (31,22%)
Gesamt	237 (100%)

Evaluation: Online-Studie - Ergebnisse I

Fehlerfall	Mehrheit
Vollständig korrekt	113 (47,68%)
Anweisung inkorrekt	21 (8,86%)
Fehlende Anweisungen	3 (1,27%)
Anweisungen zuviel	9 (3,80%)
Reihenfolge falsch	5 (2,11%)
Koreferenz inkorrekt	10 (4,22%)
Verzweigung inkorrekt	3 (1,27%)
Schleife fehlerhaft	4 (1,69%)
Parallelität falsch	0
Summe	168

Diagramme	Mehrheit
Korrekt	113 (47,68%)
Fehlerhaft	50 (21,10%)
Bestimmt	163 (68,78%)
Unbestimmt	74 (31,22%)
Gesamt	237 (100%)

Evaluation: Online-Studie - Ergebnisse II

Fehlerkategorie	Mehrheit
Vollständig korrekt	113 (47,68%)
Aktivitätsfehler	35 (14,77%)
Reihenfolgefehler	31 (13,08%)
Kontrollstrukturfehler	14 (5,91%)
Summe	193

Diagramme	Mehrheit
Korrekt	113 (47,68%)
Fehlerhaft	68 (28,69%)
Bestimmt	181 (76,37%)
Unbestimmt	56 (23,62%)
Gesamt	237 (100%)

Evaluation: Online-Studie - Ergebnisse II

Fehlerkategorie	Mehrheit
Vollständig korrekt	113 (47,68%)
Aktivitätsfehler	35 (14,77%)
Reihenfolgefehler	31 (13,08%)
Kontrollstrukturfehler	14 (5,91%)
Summe	193

Diagramme	Mehrheit
Korrekt	113 (47,68%)
Fehlerhaft	68 (28,69%)
Bestimmt	181 (76,37%)
Unbestimmt	56 (23,62%)
Gesamt	237 (100%)

Evaluation: Methodenaufrufe

- 234 Befehlsbeschreibungen mit Methodenaufrufen
- Fehler in Methodenaufrufen manuell ausgewertet
- Zielsystem: Schnittstelle des ARMAR-III
  - 92 Methoden
  - 59 Parameter
  - 18 Datentypen
  - 70 Umgebungsobjekte (als Parameter verwendbar)
- 77 (32,49%) Befehlsbeschreibungen vollständig korrekt

Typ	Gesamt	r_p	f_p	f_n	Präzision	Ausbeute	F_1
Methoden	1402	1156	95	151	92,41%	88,45%	90,38%
Parameter	1572	1249	259	64	82,82%	95,13%	88,55%
Aufrufe	1402	920	331	151	73,53%	85,90%	79,24%

- Fehler in Kontrollstrukturen

Korrekt	Bedingung	Bezugsrahmen	Beides
20 (27,40%)	51 (69,86%)	21 (28,77%)	19 (26,03%)

Evaluation: Methodenaufrufe

- 234 Befehlsbeschreibungen mit Methodenaufrufen
- Fehler in Methodenaufrufen manuell ausgewertet
- Zielsystem: Schnittstelle des ARMAR-III
  - 92 Methoden
  - 59 Parameter
  - 18 Datentypen
  - 70 Umgebungsobjekte (als Parameter verwendbar)
- 77 (32,49%) Befehlsbeschreibungen vollständig korrekt

Typ	Gesamt	r_p	f_p	f_n	Präzision	Ausbeute	F_1
Methoden	1402	1156	95	151	92,41%	88,45%	90,38%
Parameter	1572	1249	259	64	82,82%	95,13%	88,55%
Aufrufe	1402	920	331	151	73,53%	85,90%	79,24%

- Fehler in Kontrollstrukturen

Korrekt	Bedingung	Bezugsrahmen	Beides
20 (27,40%)	51 (69,86%)	21 (28,77%)	19 (26,03%)

Evaluation: Methodenaufrufe

- 234 Befehlsbeschreibungen mit Methodenaufrufen
- Fehler in Methodenaufrufen manuell ausgewertet
- Zielsystem: Schnittstelle des ARMAR-III
  - 92 Methoden
  - 59 Parameter
  - 18 Datentypen
  - 70 Umgebungsobjekte (als Parameter verwendbar)
- 77 (32,49%) Befehlsbeschreibungen vollständig korrekt

Typ	Gesamt	r_p	f_p	f_n	Präzision	Ausbeute	F_1
Methoden	1402	1156	95	151	92,41%	88,45%	90,38%
Parameter	1572	1249	259	64	82,82%	95,13%	88,55%
Aufrufe	1402	920	331	151	73,53%	85,90%	79,24%

- Fehler in Kontrollstrukturen

Korrekt	Bedingung	Bezugsrahmen	Beides
20 (27,40%)	51 (69,86%)	21 (28,77%)	19 (26,03%)

Zusammenfassung und Ausblick

- Zielsystemunabhängige Quelltextsynthese aus natürlicher Sprache
- Syntaxbaumsynthese:
  1. Strukturextraktion durch Mustererkennung
  2. Verbindung der Strukturen
  3. Ausführungsreihenfolge festlegen
  4. Konsistenzüberprüfung
- Quelltextübersetzung:
  1. Quelltexterzeugung durch Besucher
  2. Quelltextinjektion durch Schablonen-System
- Evaluation:
  1. Online-Studie (47,68% korrekt)
  2. Evaluation der Methodenaufrufe (F_1 für Aufrufe: 79,24%)
- Ausblick:
  - Automatischer Vergleich von Verfahrensbeschreibungen
  - Automatische Dokumentation durch Invertierung der Verfahren

Besucher: Quelltextzeilen

Besucher	Quelltextzeilen	Bemerkungen
`PseudoCodeVisitor`	146	
`PlantUMLCodeVisitor`	212	
`AbstractVisitor`	71	Implementiert Einrückung
`PythonCodeVisitor`	157	Erbt von *`AbstractVisitor`* und ohne Parallelität
`CStyleVisitor`	142	Erbt von *`AbstractVisitor`*
`CCodeVisitor`	25	Erbt von *`CStyleVisitor`*
`JavaCodeVisitor`	30	Erbt von *`CStyleVisitor`*

Beispiel für Ausführungsdatei-Schablone I

```
import edu.kit.ipd.lego.*;
// {{ header }}

class MyLegoRobot extends LegoRobot  {

// {{ classHead }}

    public static void main(String[] args) {
        LegoRobot robo = new MyLegoRobot();
        try {
            robo.execute();
        } catch (Exception e) {
            System.err.println("Error");
        }
        robo.close();
    }

@Override
```

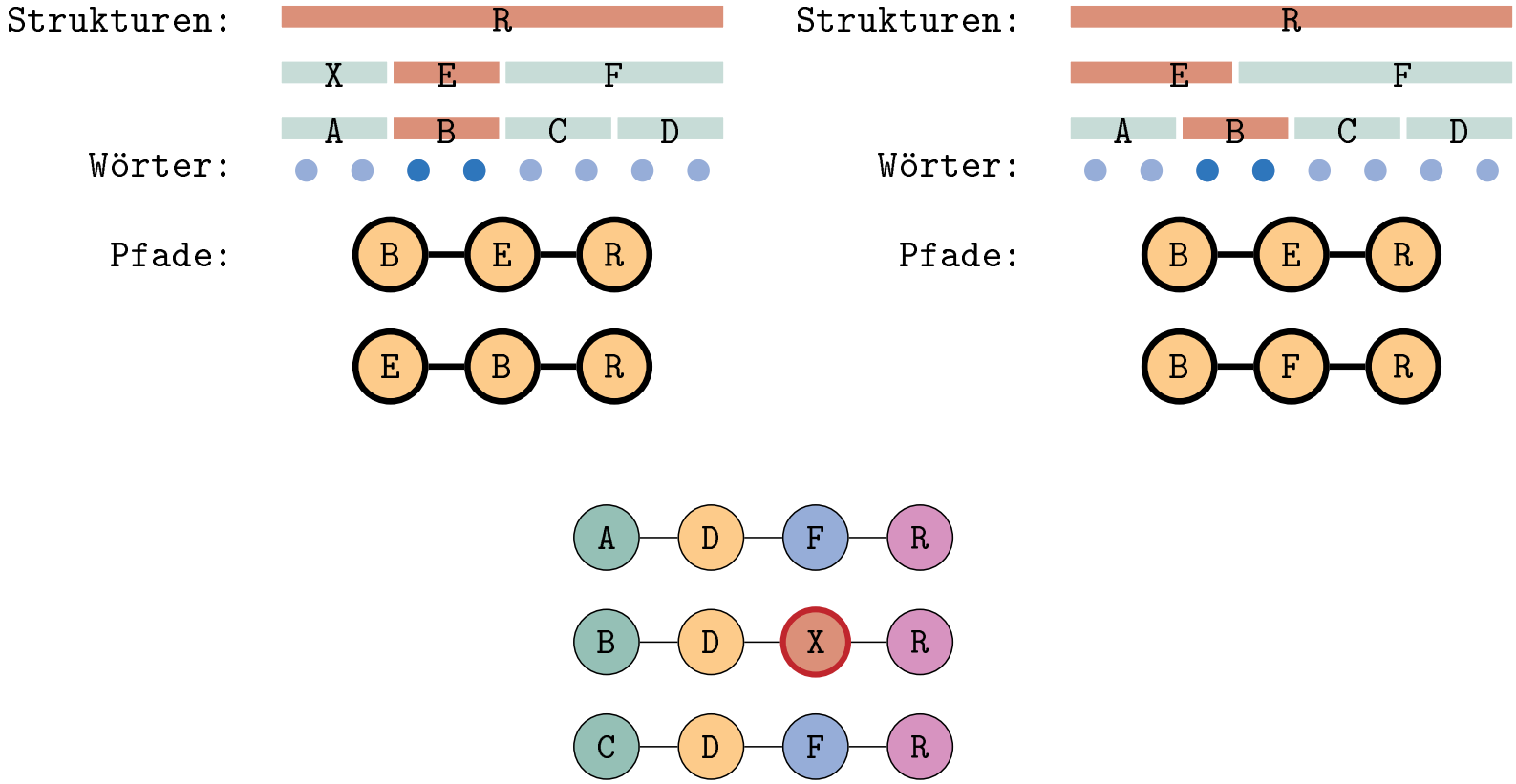
Beispiel für Ausführungsdatei-Schablone II

```
public void execute(){

// {{ topComment }}
// {{ code }}

}

// New Methods: {{newMethod}}

}
```

Besucher Methode für Verzweigung

```
public String toCode(ASTBranch astBranch) {
    String code = "if␣(";
    for (ASTNode c : astBranch.getConditions().getExpressions())
        code += c.visit(this) + "␣&&␣";
    if (!astBranch.getConditions().getExpressions().isEmpty())
        code = code.substring(0, code.length() - 4);
    code += ")\n";
    //Then-Block
    code += "{\n" + astBranch.getThenBlock().visit(this) + "}";
    //Else-Block
    code += "␣else␣{" + astBranch.getElseBlock().visit(this) + "}"
        ;
    return code;
}
```

Fehler bei Verbindung der Strukturen



References I

Zeynep Günes. „Aufbau Eines Sprachkorpus Zur Programmierung Autonomer Roboter Mittels Natürlicher Sprache“. Bachelor’s Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Mai 2015. URL: https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/guenes_ba.

Tero Hasu. „Programmatic Building of Models Just for Pretty Printing“. In: *Proceedings of OOPSLA Workshop on Domain-Specific Modeling (DSM)*. Portland, Oregon, USA, Okt. 2006.

M. de Jonge. „Pretty-Printing for Software Reengineering“. In: *International Conference on Software Maintenance, 2002. Proceedings.* International Conference on Software Maintenance, 2002. Proceedings. Okt. 2002, S. 550–559. DOI: 10.1109/ICSM.2002.1167816.

References II

Huda Khayrallah, Sean Trott und Jerome Feldman. „Natural Language for Human Robot Interaction“. In: *Proceedings of the Workshop on Human-Robot Teaming at the ACM/IEEE Conference on Human-Robot Interaction (HRI)* (Portland, Oregon). 2015. URL: http://www.bradhayes.info/hri15/papers/2.pdf.

Mathias Landhäußer, Sebastian Weigelt und Walter F. Tichy. „NLCI: A Natural Language Command Interpreter“. In: *Automated Software Engineering* 24.4 (1. Dez. 2017), S. 839–861. ISSN: 1573-7535. DOI: 10.1007/s10515-016-0202-1. URL: https://doi.org/10.1007/s10515-016-0202-1 (besucht am 28. 03. 2019).

References III

S. Lauria u. a. „Converting Natural Language Route Instructions into Robot Executable Procedures“. In: *11th IEEE International Workshop on Robot and Human Interactive Communication Proceedings*. 11th IEEE International Workshop on Robot and Human Interactive Communication Proceedings. Sep. 2002, S. 223–228. DOI: 10.1109/ROMAN.2002.1045626.

References IV

Vu Le, Sumit Gulwani und Zhendong Su. „SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language“. In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (Taipei, Taiwan). MobiSys '13. New York, NY, USA: ACM, 2013, S. 193–206. ISBN: 978-1-4503-1672-9. DOI: 10.1145/2462456.2464443. URL: http://doi.acm.org/10.1145/2462456.2464443 (besucht am 20. 03. 2019).

Xi Victoria Lin u. a. *Program Synthesis from Natural Language Using Recurrent Neural Networks*. UW-CSE-17-03-01. Seattle, WA, USA: University of Washington Department of Computer Science and Engineering, März 2017.

References V

Nicholas K. Lincoln und Sandor M. Veres. „Natural Language Programming of Complex Robotic BDI Agents“. In: *Journal of Intelligent & Robotic Systems* 71.2 (1. Aug. 2013), S. 211–230. ISSN: 1573-0409. DOI: 10.1007/s10846-012-9779-1. URL: https://doi.org/10.1007/s10846-012-9779-1 (besucht am 20. 03. 2019).

Cynthia Matuszek u. a. „Learning to Parse Natural Language Commands to a Robot Control System“. In: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Hrsg. von Jaydev P. Desai u. a. Springer Tracts in Advanced Robotics. Heidelberg: Springer International Publishing, 2013, S. 403–415. ISBN: 978-3-319-00065-7. URL: https://doi.org/10.1007/978-3-319-00065-7_28 (besucht am 20. 03. 2019).

References VI

Lili Mou u. a. „On End-to-End Program Generation from User Intention by Deep Neural Networks“. In: (25. Okt. 2015). arXiv: 1510.07211 [cs]. URL: http://arxiv.org/abs/1510.07211 (besucht am 19. 03. 2019).

David Price u. a. „NaturalJava: A Natural Language Interface for Programming in Java“. In: *Proceedings of the 5th International Conference on Intelligent User Interfaces* (New Orleans, Louisiana, USA). IUI '00. New York, NY, USA: ACM, 2000, S. 207–211. ISBN: 978-1-58113-134-5. DOI: 10.1145/325737.325845. URL: http://doi.acm.org/10.1145/325737.325845 (besucht am 19. 03. 2019).

References VII

Maxim Rabinovich, Mitchell Stern und Dan Klein. „Abstract Syntax Networks for Code Generation and Semantic Parsing“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vancouver, Canada: Association for Computational Linguistics, 2017, S. 1139–1149. DOI: 10.18653/v1/P17-1105. URL: http://aclweb.org/anthology/P17-1105 (besucht am 19. 03. 2019).

References VIII

S. Weigelt und W.F. Tichy. „Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language“. In: *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference On*. Bd. 2. Mai 2015, S. 819–820. DOI: 10.1109/ICSE.2015.264.

Pengcheng Yin und Graham Neubig. „A Syntactic Neural Model for General-Purpose Code Generation“. In: (5. Apr. 2017). arXiv: 1704.01696 [cs]. URL: http://arxiv.org/abs/1704.01696 (besucht am 20. 03. 2019).