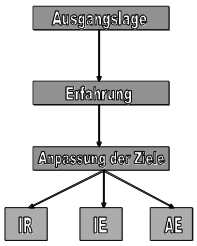
ANSWER EXTRACTION (AE)

ExtrAns: Eine Anwendung von AE

AUFBAU DES REFERATS

1. Einführung
2. Vorgeschichte/Hintergrund
   - AI, computergestütztes Suchen (ultimatives Hilfesystem)
   - TREC, IR,IE, QA & AE
3. Answer Extraction (AE)
   - AE zwischen QA, IE und IR
   - Ziel, Methode, Anwendung (ExtrAns)
4. ExtrAns
   - Anwendungsfelder: Unix (manpages), Airbus (AMM)
   - Architektur: Aufbau, Komponenten, Schwierigkeiten etc.

2. Vorgeschichte/Hintergrund



AI: Beantwortung von Benutzerfragen ohne Einschränkungen

Zu hohe Erwartungen!

Information Retrieval (IR)

Konzept	Dokumentensuche mit entsprechenden Infos zur Benutzerfrage (DR)
Mittel	„Keyword-Suche" (Salton 1983) (od. Volltextsuche, mit Stoppwortliste)

Vorteile:	Nachteile:
-willkürliche Fragestellungen	-vollständige Dokus angegeben
-Grosse Doku-Sammlungen & willkürliche Bereiche	-Ignorieren von linguistischen Infos!

Information Extraction (IE)

1. Technik ähnlich wie bei IR/DR: grosse Textsammlungen und Breite an Themen
2. Unterschied zu IR/DR: Identifizierung von sogenannten "messages". "messages" beziehen sich auf spezifische Themen, aus denen eine limitierte Menge von hochspezifischen Daten extrahiert werden.

Vorteil: präzisere Informationen als bei DR (Effizienz)

Nachteil: erlaubt keine willkürliche Fragen, diese sind vordefiniert und sehr spezifisch

Question Answering & Answer Extraction

TREC (-QA)

Alljährliche Konferenz

Bedürfnis nach präziseren & uneingeschränkteren Systemen/Methoden als IR & IE

TREC hat gezeigt, dass bei kleinen Mengen von Texten die IR Techniken jenen mit Sprachverarbeitung/linguistischen Methoden unterlegen sind, d.h. ineffizientere Ergebnisse liefern.

Question Answering (QA)

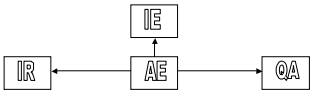
Konzept

1. Liest Texte und passt ihren Inhalt an sogenannte „knowledge basis“ an.
2. Generiert Antworten auf willkürliche, in natürlicher Sprache formulierte Fragen
3. Ermöglicht einen “natürlichen” Dialog zwischen Maschine und User

Probleme mit QA: - schwierige Implementation
- hohe Entwicklungskosten

→ QA läuft nur auf extrem eingeschränkten (an Volumen) Texten.

Answer Extraction (AE)



Konzept

- erlaubt willkürliche, in natürlicher Sprache formulierte Fragen
- Filtert/extrahiert die exakten Stellen (Sätze) aus der Dokumentensammlung heraus

→ explizite Antwort auf spezifische Frage!

WICHTIG! *AE generiert nicht eigene Antworten!!*

Extracting Answers from Technical Texts (ExtrAns) (1)

Fallbeispiel: Airbus hat nach dem Start ein technisches Problem.
Aufgabe: Diagnose des Problems anhand der Benutzeranleitung des Airbussystems (AMM)
Problem: Das AMM hat ca. 15 000 Seiten!!!

→ Answer Extraction

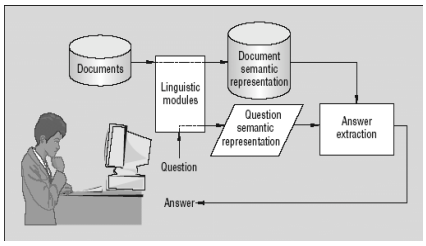
Extracting Answers from Technical Texts (ExtrAns) (2)

AE: nutzt hochentwickelte Technik, um Quelltext und Benutzerfragen zu analysieren und anschliessend automatisch Antworten zu extrahieren

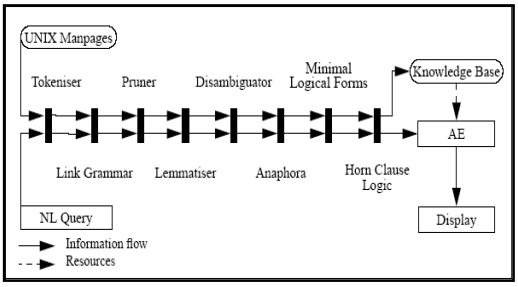
→ ExtrAns

Ein AE-System für technische Bereiche/Domänen. ExtrAns ist ein experimentelles System, welches das Konzept von AE nach seiner Anwendungsfähigkeit auf ganz bestimmte Texte und Sprachen testet.

ExtrAns: Architektur (1)



ExtrAns: Architektur (2)



ExrAns: Einführung (1)

Anwendungsbeispiele

-**Unix manpages**

-Airbus AMM

Analyse

-Analoge Analyse von Benutzerfragen und des Quelltextes

Komponenten

-Verschiedene linguistische Module interagieren miteinander

-Die wichtigsten linguistischen Module sind der "syntactic analyzer" & "semantic interpreter"

ExrAns: Einführung (2)

Ziel

-Finden jener relevanten Passagen in Unix manpages, welche die Frage direkt beantworten

Verarbeitung

- Textsammlung wird ***offline***, Benutzerfrage ***online*** verarbeitet

ExrAns: Einführung (3)

Vorgehensweise/Schritte bei der Analyse der Daten

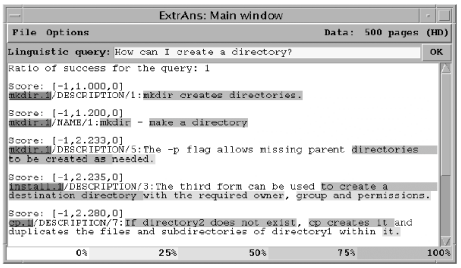
1. Daten/Text von Unix manpages werden vorverarbeitet (tokenised) und in ein vom Parser verstandenen Format konvertiert
2. Die Daten werden geparst
3. Ambigiutäten aufgehoben (so weit als möglich)
4. In logische Formen konvertiert, welche die semantische Information und die Abhängigkeiten zwischen den Wörtern darstellen.

ExrAns: Einführung (4)

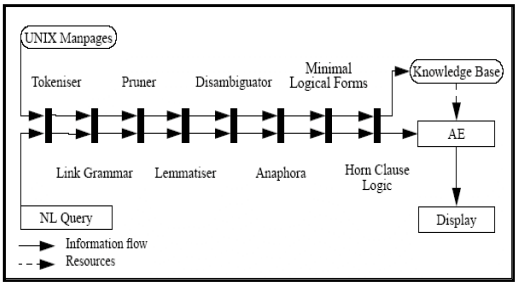
Vorgehensweise bei einer Abfrage

1. Konstruktion der logischen Formen des Fragesatzes, wie oben beschrieben.
2. System sucht in der Datenbank nach gleichen (ähnlichen) logischen Formen
3. Anzeige der gefundenen, relevanten Ergebnisse (Sätze) in einer provisorischen Darstellung
4. Benutzer wählt (klickt) eine der Ergebnisse

ExrAns: Einführung (5)



ExtrAns: Architektur



ExtrAns: Module

Die wichtigsten Module sind:
- Parsing
- Disambiguierung
- Anaphernresolution
- Generierung der logischen Formen

Syntaktische Analyse: Parsing (1)

verwendeter Parser

„***Link Grammar***“ (LG) von Sleator und Temperley (1993)

Vorteile

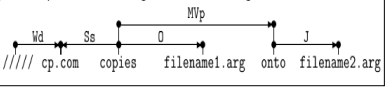
1. Hohe Parsgeschwindigkeit
2. Robustheit
3. Umfassende englische Grammatik

Syntaktische Analyse: Parsing (2)

Der Link Grammar (LG) gibt das Resultat als „dependency relations“ (dependenzbasiert), in so genannten „linkages“ zurück.

Beispiel

(1) *cp.com copies filename1.arg onto filename2.arg*



Disambiguator (Disambiguierung)

disambiguator von Brill und Resnik (1994)

-eine Korpus-basierte Methode zur Desambiguierung von Präpositionalphrasen(anbindungen)

-Entscheidung basiert auf “four tuples” & “two fields”

Anaphora resolution (Anaphernresolution)

Ziel

Prüfung und Korrektur der von LG erstellten link-Typen!!

Output

Eine Liste von äquivalenten Klassen, in denen Wörter gruppiert werden, die zum gleichen Objekt gehören.

Minimale logische Formen (MLF) 1

Kernstück des Systems

Anforderungen an die Semantikrepräsentation:

1. Einfache Herleitung der Syntaxstruktur
2. Einfache Verarbeitung
3. Ausreichende/genügende Ausdrucksstärke (semantische Kernbedeutung)

Minimale logische Formen (MLF) 2

MLF → Flache existentiell geschlossene Formeln

Erste vereinfachte Darstellung

Satz: *„cp copies files“*

```
object(cp,x1).
evt(copy,[x1,x2]).
object(file,x2).
```

MLF: Reification 1

Definition: *Reification* is the technical term for introducing new entities that refer to abstract concepts.

drei Prädikatstypen (predicates)

- object → **Objekte**: alles, worauf ein NP referieren kann
- evt → **Eventualitäten** (events): alles, was ein Verb beschreibt (Zustände, Prozesse, Ereignisse
- prop → **Eigenschaften**: Attribute, Adjektive etc.

MLF: Reification 2

Eventualität als Individuum (Davidson 1967)

Satz: *„cp copies files quickly“*
(Adverbialmodifikation)

```
evt(copy,e1,[x1,x2]).
```

e1 bedeutet das Konzept, dass x1 x2 kopiert. Somit lässt sich der Satz cp copies files quickly darstellen als:

```
object(cp,x1).
evt(copy,e1,[x1,x2]). % Verb-Reifikation e1
object(file,x2).
quickly(e1). % Modifikation von e1
```

MLF: Reification 3

untergeordnete Sätze

„cp refuses to copy files onto itself“

```
object(cp,x1).
evt(copy,e1,[x1,x2]).   % Verb-Reifikation e1
evt(refuse,e2,[x1,e1]). % Modifikation von e1; Reifikation e2
object(file,x2).
onto(e1,x2).
```

MLF: Reification 4

Darstellung einer Objekprädikation modifiziert durch Adjektive

Darstellung eines Objektes: object (name, o1, x1)

→ *o1 is the cocept that the object x1 is name.*

Satz: *„mkdir creates new directories“*

```
object(mkdir,o1,x1).
evt(create,e1,[x1,x2]).
object(directory,o2,x2).
prop(new,o2). % new modifiziert ein Konzept
```

MLF: Reification 5

Modifizierung des Objektes

Satz: *„cp copies long files.“*

```
object(file,o2,x2).
prop(long,x2).
```

Modifizierung der Eigenschaften

Satz: *„cp copies very long files.“*

```
prop(long,p1,x2).
prop(very,p2,p1).
```

MLF: Reification 6

Unterspezifikation

Satz: *„cp refuses to copy files onto itself“*

Zusatzprädikat: ***holds***

```
holds(e2).          % Refuse existiert auch in der Realität.
object(cp,x1).
evt(refuse,e2,[x1,e1]).
evt(copy,e1,[x1,x2]).
object(file,x2).
onto(e1,x2).
```

MLF: Benutzerfrage 1

Frage: *„Which command copies files?“*

MLF des Fragesatzes:

```
object(command, 01,X1),
evt(copy,E1,[X1,X2]),
object(file,02,X2).
```

→ Variablen, # Konstanten!!

MLF: Benutzerfrage 2

Query: „Which command copies files?“

```
object(command,O1,x1). evt(copy,E1,[X1,X2]).
object(file,O2,X2).
```

Folgende Antworten werden durch Prolog-Unifikation bewiesen, also die Variablen können erfolgreich gebunden werden:

findet

Answer1: „cp copies files“

```
holds(e2). object(cp,o1,x1). evt(copy,e2,[x1,x2]).
object(file,o2,x2).                    % LogForm
object(command,o3,x1).                 % from domain knowledge
```

Answer2: „cp refuses to copy a file onto itself“

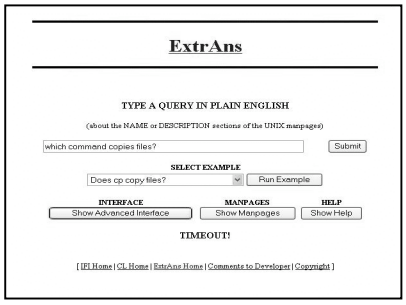
```
holds(e1). object(cp,o1,x1). evt(copy,e2,[x1,x2]).
object(file,o2,x2). evt(refuse,e1,[x1,e2]).
onto(e2,x2).                           % LogForm
object(command,o3,x1).                 % from domain knowledge
```

Answer3: „If the user types y, then cp copies the files“

```
if(e1,e2). object(cp,o1,x1). evt(copy,e2,[x1,x2]).
object(file,o2,x2). object(user,o3,x3).
evt(type,e1,[x3,x4]). object(y,04,x4).  % LogForm
object(command,o3,x1).                 % from domain knowledge
```

Abbildung 3: Auffinden der Antworten durch Prolog-Unifikation

Demoversion von ExtrAns auf dem Web



http://www.ifi.unizh.ch/CL/extrans