Kanten-Subsumtion

Übersicht

- Die vorgestellten Chart-Parser haben bei bestimmten Grammatiken Probleme
- die Symbole der Grammatik dürfen nicht unterspezifiziert sein
  - OK: Prolog-Atome
  - OK: komplexe Terme, deren Argumente jederzeit voll spezifiziert sind
  - nicht erlaubt: komplexe Terme mit freien Variabeln

Ziel

- Verstehen, wo das Problem liegt
- Verstehen, wie das Problem behoben wird

Kanten-Subsumtion

Chart-Parser fügen nicht immer eine Kante in die Chart ein.

- eine Kante mit Kategorie `np` sei in der Chart
- der Parser will eine Kante mit `np` einfügen
- Kante wird ***nicht*** eingefügt, da beide Terme unifizierbar sind

Das ist richtig so.

- es ist ja bereits eine NP gefunden worden

Kanten-Subsumtion

Chart-Parser fügen nicht immer eine Kante in die Chart ein.

- eine Kante mit »Kategorie« `np(singular)` sei in der Chart
- der Parser will eine Kante mit `np(singular)` einfügen
- Kante wird ***nicht*** eingefügt, da beide Terme unifizierbar sind

Das ist richtig so.

- es ist ja bereits eine Singular-NP gefunden worden

Kanten-Subsumtion

Chart-Parser fügen nicht immer eine Kante in die Chart ein.

- eine Kante mit »Kategorie« `np(X)` sei in der Chart
- der Parser will eine Kante mit `np(singular)` einfügen
- Kante wird ***nicht*** eingefügt, da beide Terme unifizierbar sind

Das ist richtig so.

- Singular-NPs sind ein Spezialfall von (allgemeinen) NPs.
- Dass eine Singular-NP gefunden wurde, ist also nichts Neues.

Kanten-Subsumtion

Chart-Parser fügen nicht immer eine Kante in die Chart ein.

- eine Kante mit »Kategorie« `np(singular)` sei in der Chart
- der Parser will eine Kante mit `np(X)` einfügen
- Kante wird ***nicht*** eingefügt, da beide Terme unifizierbar sind

Das ist *falsch*!

- `X` könnte ja auch für `plural` stehen
- eigentlich wollen wir gar nicht wissen, ob die beiden Terme unifizierbar sind, sondern ob der in der Chart *allgemeiner* ist als der einzufügende

Kanten-Subsumtion

Was wir wirklich wollen:

- Eine Kante wurde »bereits gefunden«, wenn …
  - der Term in der Chart *allgemeiner* ist als jener der neuen Kante
- nur dann sollte die neue Kante ignoriert werden

Chart	neue Kante	neue Kante ignorieren?
`f(X)`	`f(X)`	*ja*
`f(X)`	`f(a)`	*ja*
`f(a)`	`f(X)`	*nein*

subsumes_chk/2

Chart	neue Kante	neue Kante ignorieren?
`f(X)`	`f(X)`	*ja*
`f(X)`	`f(a)`	*ja*
`f(a)`	`f(X)`	*nein*

Das Prolog-Prädikat subsumes_chk/2 führt gerade diesen Test durch.

- `subsumes_chk(f(X), f(X))` → `yes`
- `subsumes_chk(f(X), f(a))` → `yes`
- `subsumes_chk(f(a), f(X))` → `no`

Wann wird also eine Kante ignoriert?

Eine neue Kante *E* ist also zu ignorieren, wenn eine andere Kante *C* bereits in der Chart ist, wobei

- Startknoten von *C* = Startknoten von *E*
- Endknoten von *C* = Endknoten von *E*
- die Kategorie von *C*
  ist allgemeiner als (»subsumiert«) die Kategorie von *E*
- der Teil vor dem Punkt von *C*
  subsumiert den Teil vor dem Punkt von *E*
- der Teil nach dem Punkt von *C*
  subsumiert den Teil nach dem Punkt von *E*

add_edge

Dieser Prolog-Code läuft so unter SICStus-Prolog

- bei anderen Interpretern sind evtl. Anpassungen nötig
- für beliebige Prologs: → [Covington, 1994], S. 174/175

```
:- use_module(library(terms), [subsumes_chk/2]).

add_edge(edge(Vfrom, Vto, Cat, Found, ToFind)) :-
  \+ (edge(Vfrom, Vto, ChartCat,
           ChartFound, ChartToFind),
      subsumes_chk(ChartCat, Cat),
      subsumes_chk(ChartFound, Found),
      subsumes_chk(ChartToFind, ToFind)),
  asserta(edge(Vfrom, Vto, Cat, Found, ToFind)).
```

Aufgaben: Kanten-Subsumtion

Programmiertechniken der Computerlinguistik 2 · Sommersemester 1999

1. Kanten-Subsumption

Die nachfolgende Grammatik mit komplexen Nichtterminal-Symbolen wird von [Covington, 1994] als Beispiel angegeben, wieso der Subsumptions-Test bei Chart-Parsern nötig ist. Benutze den Earley-Parser aus der letzten Lektion und lasse den Satz »the cat sees a dog« als S parsen.

```
rule(s,  [np, vp]).
rule(np, [det, n]).
rule(vp, [verbal(0)]).
rule(vp, [verbal(X), rest_of_vp(X)]).
rule(verbal(X), [v(X)]).
rule(rest_of_vp(1), [np]).
rule(rest_of_vp(2), [np, vp]).

word(det, the).
word(det, a).
word(n, dog).
word(n, cat).
word(v(0), sings).    % Verb mit 0 Objekten
word(v(1), chases).   % Verb mit 1 Objekt
word(v(1), sees).     % Verb mit 1 Objekt
word(v(2), gives).    % Verb mit 2 Objekten
```

a) Überlege Dir, warum das Parsing fehlschlägt, obwohl es eigentlich gelingen sollte.

b) Verbessere den Earley-Parser, indem Du den Subsumtions-Test an geeigneter Stelle einbaust.

c) Wieso funktioniert dieselbe Grammatik mit dem Bottom-Up-Chart-Parser aus einer früheren Lektion, obwohl dieser ebenfalls mittels Unifikation testet, ob eine Kante bereits in der Chart ist?