Chart parsing with non-atomic categories

L545

Spring 2013

(With thanks to Detmar Meurers)
By utilizing unification as we parse, we can eliminate parses that don’t work in the end

- e.g., eliminate NPs that don’t match in agreement features with their VPs as we parse, instead of as a filter
Changes to the chart representation

Each state will be extended to include the LHS feature structure (FS), which can get augmented as it goes along

▶ i.e., Add a feature structure (in DAG form) to each state
  ▶ So, $S \rightarrow \bullet \text{NP VP, [0,0]}$
  ▶ Becomes $S \rightarrow \bullet \text{NP VP, [0,0], FS}_S$

The predictor, scanner, and completer have to pass the FS, so all three operations have to be altered
Earley parser with atomic categories

**Prediction:**
for each $i[A \rightarrow \alpha \bullet_j B \beta]$ in chart
for each $B \rightarrow \gamma$ in rules
add $j[B \rightarrow \bullet_j \gamma]$ to chart

**Scanning:**
let $w_1 \ldots w_j \ldots w_n$ be the input string
for each $i[A \rightarrow \alpha \bullet_{j-1} w_j \beta]$ in chart
add $i[A \rightarrow \alpha w_j \bullet_j \beta]$ to chart

**Completion (fundamental rule of chart parsing):**
for each $i[A \rightarrow \alpha \bullet_k B \beta]$ and $k[B \rightarrow \gamma \bullet_j]$ in chart
add $i[A \rightarrow \alpha B \bullet_j \beta]$ to chart
Earley parser with unification

**Prediction:**

for each $i[A \rightarrow \alpha \bullet_j B \beta]$ in chart

for each $B' \rightarrow \gamma$ in rules

add $j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma = \text{mgu}(B, B')$ to chart

**Completion (fundamental rule of chart parsing):**

for each $i[A \rightarrow \alpha \bullet_k B \beta]$ and $k[B' \rightarrow \gamma \bullet_j \gamma]$ in chart

add $i[\sigma(A \rightarrow \alpha B \bullet_j \beta)]$ with $\sigma = \text{mgu}(B, B')$ to chart
Prediction:

for each \( i[A \rightarrow \alpha \bullet_j B \beta] \) in chart

for each \( B' \rightarrow \gamma \) in rules

add \( j[\sigma(B \rightarrow \bullet_j \gamma)] \) with \( \sigma = \text{mgu}(B, B') \) to chart

The predictor takes the specification of \( B \) (i.e., FS) and finds the most general unifier (mgu) of \( B \) with \( B' \)

- If \( B \) & \( B' \) do not unify, the rule for \( B' \) is not added to the chart
- Initially (i.e., at position 0), all that happens is that a dotted rule with a FS is added to the chart
Completion (fundamental rule of chart parsing):

for each \( i[A \rightarrow \alpha \bullet_k B \beta] \) and \( k[B' \rightarrow \gamma \bullet_j] \) in chart

add \( i[\sigma(A \rightarrow \alpha B \bullet_j \beta)] \) with \( \sigma = \text{mgu}(B, B') \) to chart

Again, a step of unification is added.

- \( B \) and \( B' \) must unify in order for the dot to move
- The resulting (more specific) FS is added to the chart
How to use a chart with feature structures

- Use **unification** to combine categories in completion or prediction.
- Each time a rule or edge is used, a new **copy** is made.
- But how about testing whether an entry already exists in the chart?
  - Currently, we simply check to see whether a state **unifies** with something already in the chart and do not add a new state if it is already there.
  - But a more specific or a more general state may already be in the chart.
The subsumption problem (based on Covington 1994)

- S → NP VP
- NP → Det N
- VP → V'(0)
- VP → V'(X) Comps(X)
- V'(X) → V(X)
- V'(X) → Adv V(X)
- Comps(1) → NP
- Comps(2) → NP NP
- Det → the
- N → dog
- N → cat
- Adv → often
- V(0) → sings
- V(1) → chases
- V(2) → gives
The subsumption problem (2)

What happens when we try to parse *the dog chases the cat*?

- At position 2 (between *dog* and *chases*), from 2 to 2, the parser predicts:
  - $\text{VP} \rightarrow \bullet \text{V}'(0)$
  - $\text{V}'(0) \rightarrow \bullet \text{V}(0)$
  - $\text{V}'(0) \rightarrow \bullet \text{Adv V}(0)$
  - $\text{VP} \rightarrow \bullet \text{V}'(X) \text{ Comps}(X)$

- What happens when we scan *chases*?
  - We have a passive $\text{V}(1)$ edge
  - But there is no predicted $\text{V}'(1)$ edge—only $\text{V}'(0)$
Using subsumption to check the chart

Subsumption check: Do not add a state to the chart if an equivalent or more general state is already there.

- In trying to add a singular determiner state at \([x, y]\), if the chart already has a determiner state at \([x, y]\) unspecified for number, do not add it.
- Without a subsumption restriction, we could add two states at \([x, y]\), one expecting to see a singular determiner, the other just a determiner.
  - On seeing a singular determiner, the parser advances the dot on both rules, creating two edges (since singular unifies with singular and with unspecified).
  - As a result, we would get duplicate edges.
- With subsumption, if either a singular or plural determiner is encountered, we advance the dot, creating only one edge (singular or plural) at \([x, y]\).
Let’s define a function `subsumes_chk` which takes 2 arguments: more general item & more specific item

No variables:

- `subsumes_chk(V'(1),V'(1))`. → yes
- `subsumes_chk(V'(1),V'(2))`. → no

Compound terms without variables are either identical or different, i.e., here: subsumption = unification
Checking for subsumption

Case 2

Variables only in more general term:

- `subsumes_chk(V'(X),V'(1)). → yes`
- `subsumes_chk(foo(X,X),foo(1,1)). → yes`
- `subsumes_chk(foo(X,X),foo(1,2)). → no`

Succeeds if a consistent variable assignment exists, i.e., here: subsumption = unification
Checking for subsumption

Case 3

Variables in both terms:

- subsumes_chk(vbar(X),vbar(Y)). → yes
- subsumes_chk(vbar(X),vbar(foo(1,Y))). → yes
- subsumes_chk(vbar(foo(1,2)),vbar(foo(1,Y))). → no

- Succeeds if terms can be unified without further instantiating more specific term; in other words:
  - Unification should not require a particular instantiation of a variable in the more specific term.

- Idea: Identify each variable in more specific term with a unique, variable-free term; then subsumption = unification.
The restriction problem

Shieber et al 1995: Grammar accepting $ab^n$ with $N$ being instantiated to the successor representation of $n$.

```
start → r(0, N)
```

```
  r(X, N) → r(s(X), N) b
```

```
r(N, N) → a
```

Prediction step with unification will loop:

```
1 0[start → •_0 r(0, N)]
```

```
2 pred r(0, N) in 1 0[r(0, N) → •_0 r(s(0), N) b]
```

```
3 pred r(s(0), N) in 2 0[r(s(0), N) → •_0 r(s(s(0)), N) b]
```

```
4 pred r(s(s(0)), N) in 3 0[r(s(s(0)), N) → •_0 r(s(s(s(0))), N) b]
```

```
5 pred r(s(s(s(0))), N) in 3 0[r(s(s(s(0))), N) → •_0 r(s(s(s(s(0)))), N) b]
```

...
Using restriction to prevent prediction loops

- Prediction terminates for grammars with atomic categories, since a new item is only added to the chart if not already there and there is a finite number of atomic categories.
- Moving beyond atomic categories, there can be an infinite number of non-atomic categories.
- Prediction loop on left-recursive rules can be a problem again.
- Solution: restrict number of predicted categories to finitely many cases.
Prediction with restriction

for each $i[A \rightarrow \alpha \bullet_j B \beta]$ in chart

for each $B' \rightarrow \gamma$ in rules

add $j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma = restriction(mgu(B, B'))$ to chart

$restriction(mgu(B, B'))$ can be any operation reducing the number of possible substitutions to finite classes:

- depth bound on term complexity
- elimination of terms that are known to grow indefinitely
- use of only selected terms known not to grow indefinitely

This is sound since prediction only creates a hypothesis to be completed!
Example

Grammar: \[\text{start} \rightarrow r(0, N)\]
\[r(X, N) \rightarrow r(s(X), N) \ b\]
\[r(N, N) \rightarrow a\]

Parsing using a restrictor that replaces every term deeper than 2 with a variable:

1  \[0[\text{start} \rightarrow \bullet_0 r(0, N)]\]
2  pred \(r(0, N)\) in 1  \[0[r(0, N) \rightarrow \bullet_0 r(s(0), N) \ b]\]
3  pred \(r(s(0), N)\) in 2  \[0[r(s(0), N) \rightarrow \bullet_0 r(s(s(0)), N) \ b]\]
4  pred \(r(s(s(A)), N)\) in 3  \[0[r(s(s(A)), N) \rightarrow \bullet_0 r(s(s(s(A))), N) \ b]\]
5  pred \(r(s(s(A))), N\) in 4  = edge 4

...