Ling/CSE 472: Introduction to Computational Linguistics

5/11/17
Parsing, unification, parsing with unification

Overview

- Reading questions (finishing from Tuesday)
- Unification
- Parsing with unification
- Evaluating parsing
- Reading questions

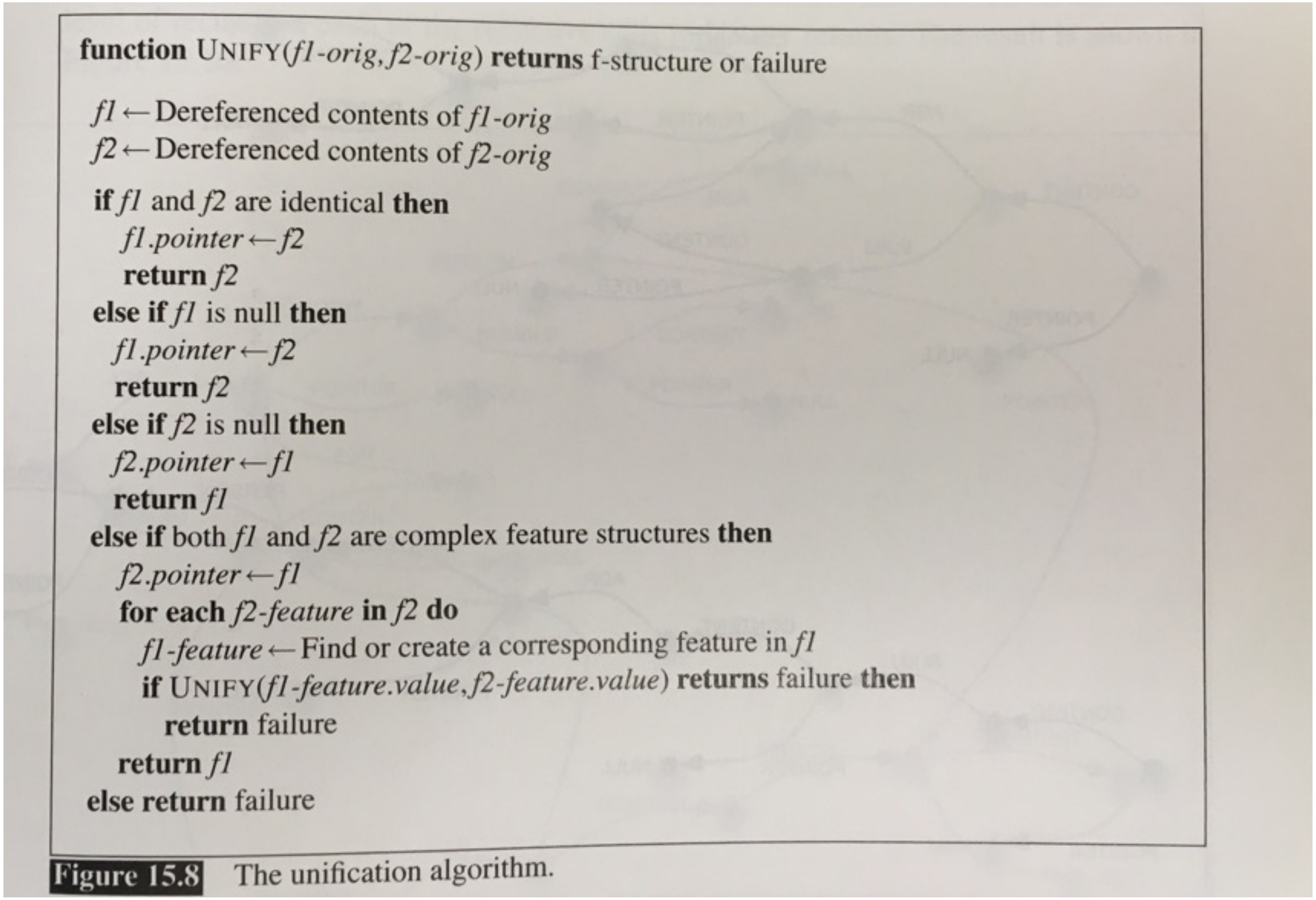
Unification

- Input: Two feature structures
- Output: Failure signal or the unique most general feature structure containing all info from both inputs

Unification algorithm

- Use graphs to represent feature structures
- Augment these structures with another layer, adding features ‘pointer’ and ‘content’
- Use pointers to merge the graphs representing the two input feature structures (why?)
- Unification is recursive (why?)
- Unification is destructive (why?)

Unification algorithm



```
function UNIFY(f1-orig, f2-orig) returns f-structure or failure

f1 ← Dereferenced contents of f1-orig
f2 ← Dereferenced contents of f2-orig

if f1 and f2 are identical then
   f1.pointer ← f2
   return f2
else if f1 is null then
   f1.pointer ← f2
   return f2
else if f2 is null then
   f2.pointer ← f1
   return f1
else if both f1 and f2 are complex feature structures then
   f2.pointer ← f1
   for each f2-feature in f2 do
      f1-feature ← Find or create a corresponding feature in f1
      if UNIFY(f1-feature.value, f2-feature.value) returns failure then
         return failure
   return f1
else return failure
```

Figure 15.8 The unification algorithm.

Unification algorithm: Example

f1:

f2:

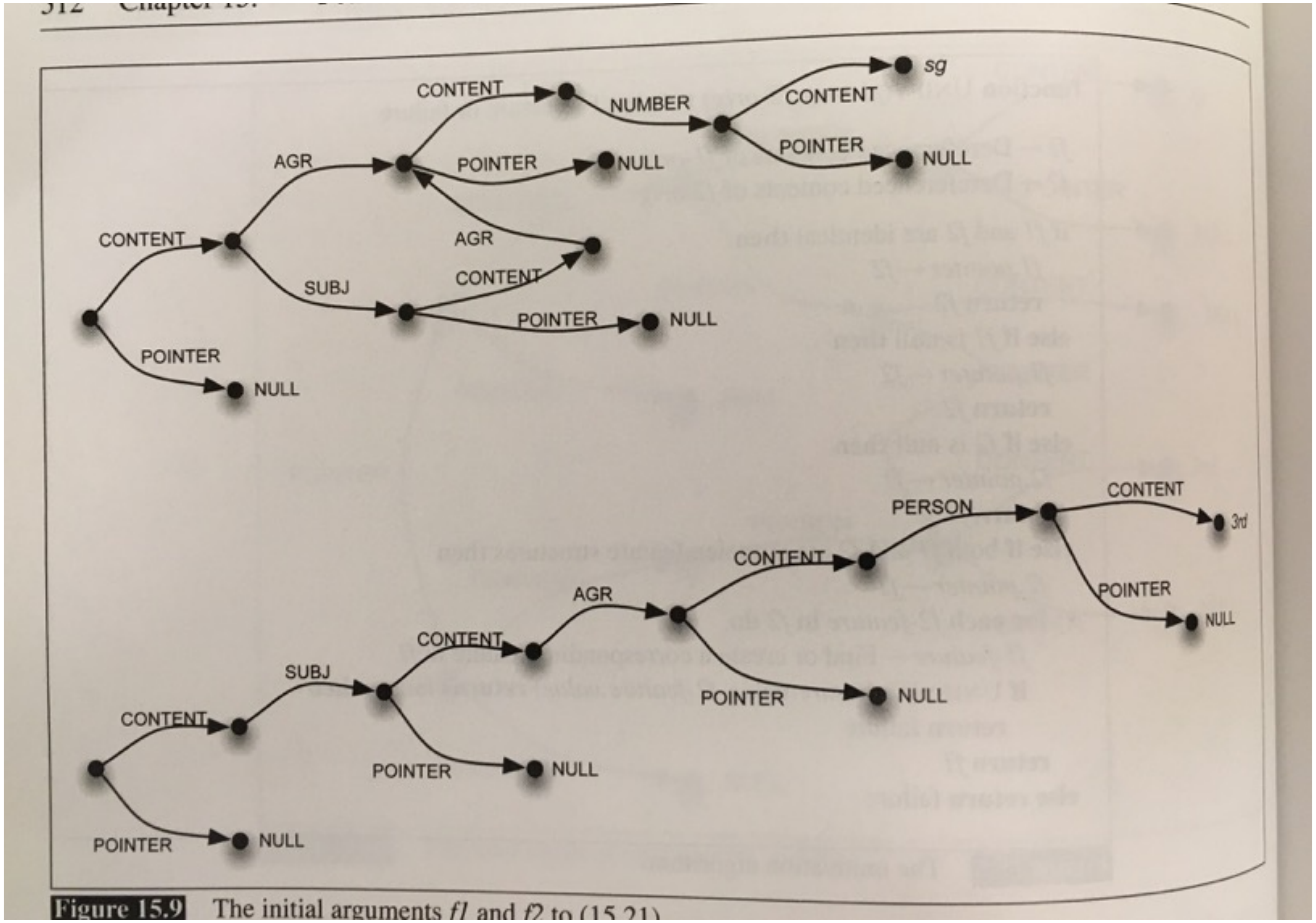


Figure 15.9 The initial arguments f1 and f2 to (15.21)

Unification algorithm: Result

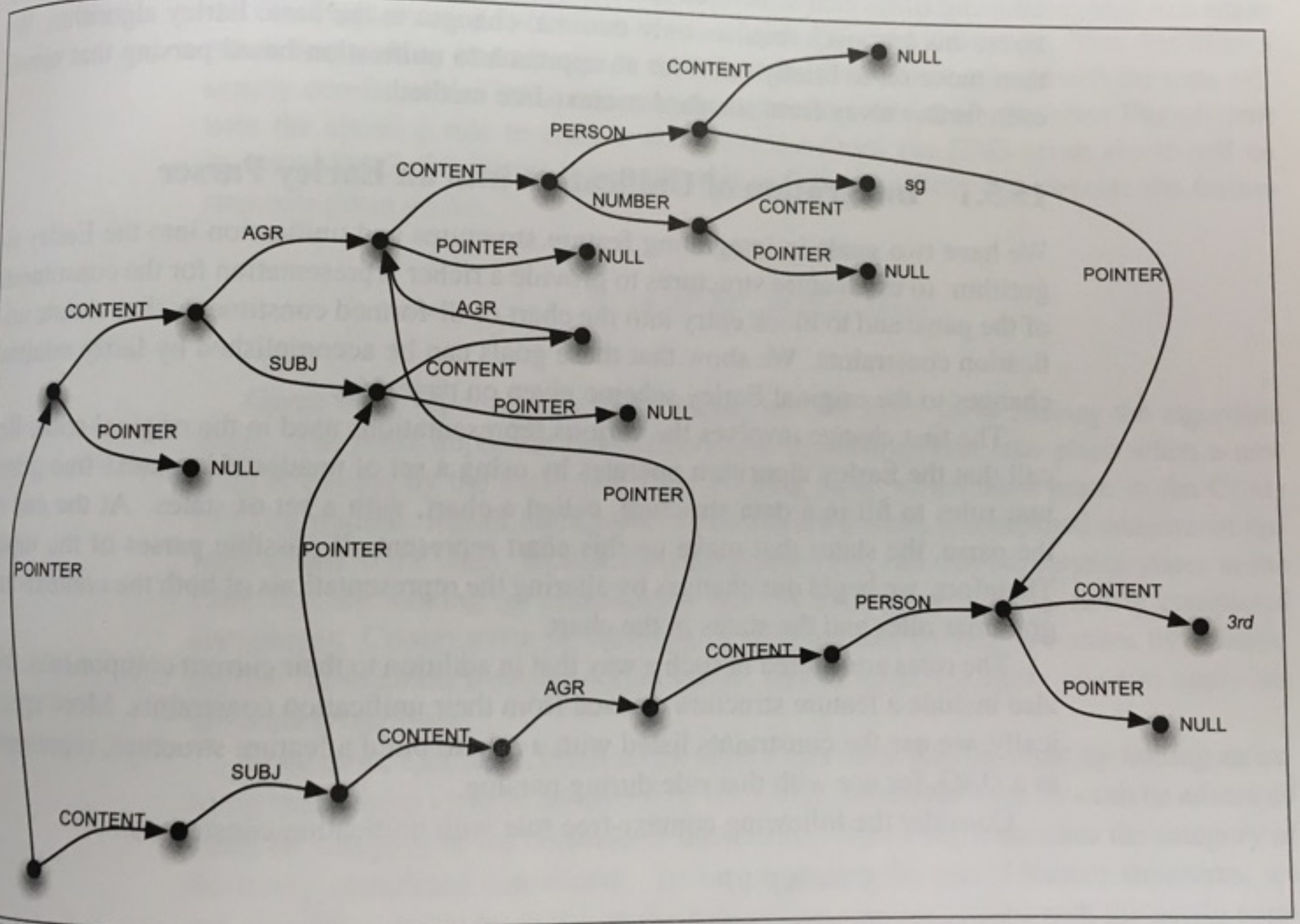


Figure 15.10 The final structures of *f1* and *f2* at the end.

Unification example

$$\begin{bmatrix} \text{A} & \boxed{1}\begin{bmatrix} \text{B} & \text{c} \end{bmatrix} \\ \text{D} & \begin{bmatrix} \text{E} & \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{D} \begin{bmatrix} \text{E} \begin{bmatrix} \text{F} \ \text{g} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- Represent each feature structure as a DAG
- Add the ‘pointer’ and ‘contents’ features
- Step through the unification algorithm to produce the result
- How would we have to alter this to handle typed feature structures?

Parsing with unification

- Associate feature structure constraints with rules
- Associate feature structures with edges
- Could just check after CFG parsing is done, but this is inefficient (why?)
- Instead: invoke unification when combining edges (COMPLETER)
- When deciding whether an edge to add is redundant, test is now *subsumption* (rather than identity): don’t add edges that are *subsumed* by something already in the chart

Evaluation slide

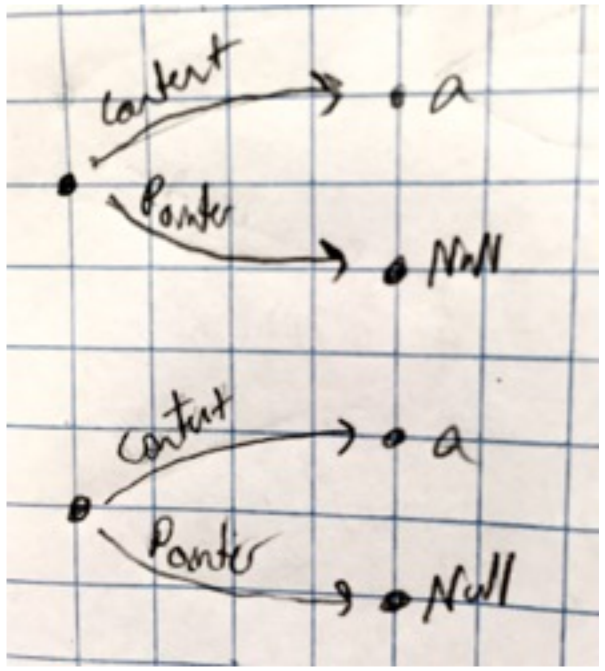
- When we are evaluating parsing algorithms (rather than parsers incorporating specific grammars), what are we evaluating them for?
- What kinds of tests could we perform?
- What inputs do we need for the tests?

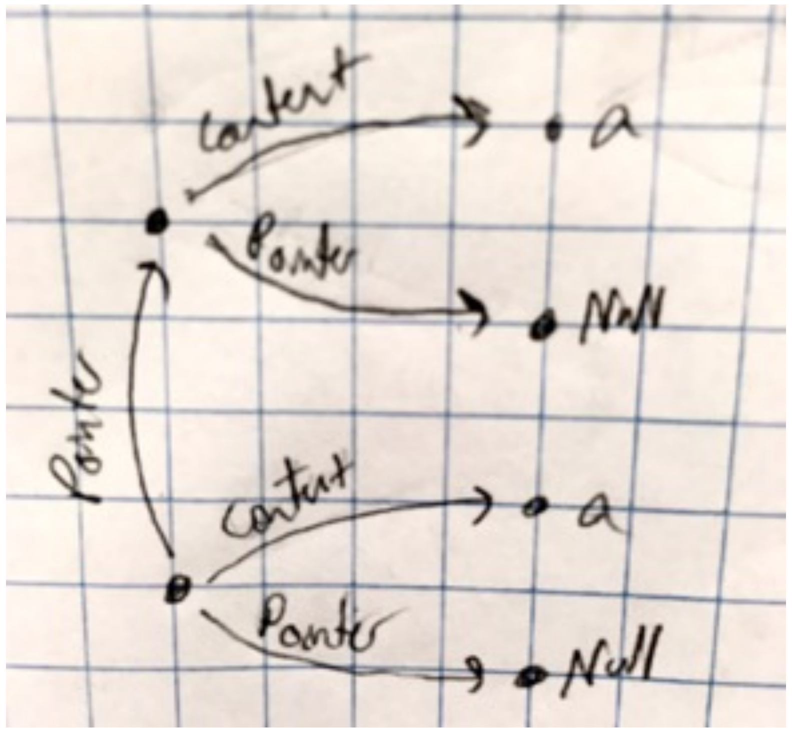
Reading questions

- "Unfortunately, this solution is inadequate since it does not meet our requirement that the two arguments be truly unified. Since the two arguments are not completely unified at the top level, future unifications involving one of the arguments would not show up in the other.” Why not?

Reading questions

- If the lower half of the diagram is changed (i.e. content -> b is added), this will not be visible by the upper half since nothing points to it. If we had a more complex structure (so the upper and lower DAGs are part of two separate larger DAGs) wouldn't this be an issue?





- If you have a node's pointer point to another node, can you delete the content node and everything that follows on that path?

Reading questions

- On page 537 it says that some people create a separate metalanguage that describes feature structures to implement operations like negation, disjunction, etc, which is distinguished from the actual feature structures. Could you explain how this works?
- How can negation be applied to unification? Does this mean one, or both features is 'negated'? If so, what does it mean to negate a feature?

Reading questions

- If/when we were to implement a type based feature structure, how would we go about assigning types? Would it not require us to manually assign the types of each element of our lexicon? Are there corpora with the most useful type information available for all English words?

- How does the type hierarchy to solve the generalization problems?

- What would be a linguistic example of a complex type with a feature whose value is also a complex type?