What we covered

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01, 12. March 2003

What we covered

1. Finite state machines and regular languages
2. Implementing finite state machines and learning Prolog along the way
3. Towards more complex grammar formalisms: Formal language theory
4. More on recursive relations in Prolog
5. From context free grammars to definite clause grammars
6. What to encode in a grammar: A DCG for English
7. How to process with a grammar: Intro to Parsing
8. Parsing strategies and their implementation

9. Remembering sub-results: Well-formed substring tables/CYK
10. Implementation of CYK
11. Remembering subcomputations: The active chart/Earley
12. More complex data structures: From atomic symbols to first order terms to feature structures
13. Term and feature structure unification
14. An brief introduction to the typed-feature based parsing system Trale
15. PATR-II and Parsing with complex categories
16. Chart-Parsing with complex categories

Finite State Techniques

Objectives:

- Introduce notions of finite state machine and finite state transducer.

Comprehension:

- What is the difference between an initial and a final state in an FSM?
- What causes non-determinism in an FSM?
- What is the difference between an FSM and an FST?
- What is the relation between regular expressions and FSMs?

Abilities:

- Translate between FSTNs, transition tables and regular expressions.
- Explain what determinism means for FSMs and FSTs.
- Specify simple FSMs and FSTs.
- Give two procedural, and one declarative, interpretation for a given FSTN.

Basic Formal Language Theory

Objectives:

- To introduce the idea that it is possible to formally describe languages and that according to what description language you choose it may or may not be possible to describe a given language in it.
- To introduce the major formal framework for describing languages that is used in mathematics and theoretical computer science: formal language theory.
- In particular, to cover the basics of set theory, the definition of what a *language* is, grammars and automata and the Chomsky hierarchy.

Comprehension:

- According to the formal definitions, how is a language (in general) related to its vocabulary?
- What has to be specified in order to define a grammar?
- Why are different kinds of automata associated with different classes of languages?
- What are the main components of an automaton?
- What does the word “generate” mean, in the technical sense used here?
- What is a type 3 device, what is the corresponding type of language and what kinds of rules does one have to use in defining such a language?

- Name a typical example of a context free language that is not a regular language.
- Name a typical example of a context sensitive language that is not context free.
- What sort of input can formal language theory give into the study of languages?

Abilities:

- Explain the relations between automata and grammars.
- Cite examples of languages of different types and give grammars for them.
- Parse and generate with grammars of types 1, 2, and 3 by hand.

DCGs as Grammar Formalism

Objectives:

- To introduce the definite clause grammar (DCG) formalism.
- To show how a DCG can be regarded as a shorthand for a description of the language written in logic.
- To show the basis of a context-free grammar of English, using X-bar theory and the DCG notation.
- To show how a feature-based notation for grammars can capture appropriate generalisations but not prevent the grammar from being context-free (because the features all take a finite number of possible values).

- To discuss the distinction between *complementation* and *modification*.

Comprehension:

- What are the five main extensions to basic DCGs that are described?
- Give three ways of representing strings and substrings in translating a DCG into logic.
- In X-bar theory, how many bars are assigned to a noun? a noun phrase? a verb? a verb phrase? a preposition? a prepositional phrase? an adjective? an adjectival phrase?
- State some possible values of the $vform$ feature and what they represent.
- Give an example of where a verb subcategorises for a given complement. Do the same for a noun and an adjective.

Abilities:

- Use DCGs to write grammars for context-free languages.
- Say whether a given DCG is equivalent to a context-free grammar.
- Explain how grammar description can be viewed as writing axioms in logic.
- Explain and exemplify the use of finite-valued features in a grammar for English, in particular the use of the *subcat* feature to mediate between lexical entries and grammar rules.
- Distinguish between complementation and modification and explain how this might be represented in phrase structure.
- Complete the grammar above by writing appropriate lexical entries and modifier rules, and extend its coverage to new lexical items.

Unbounded Dependencies in DCGs

Objectives:

- To describe what “unbounded dependencies” are (and how they differ from “bounded dependencies”) in natural language constructions.
- To show how gap threading can be used within a DCG elegantly to describe examples of unbounded dependency constructions.

Abilities:

- Explain what makes UDCs unbounded.
- Give examples of different UDCs.
- Give example gap-threading grammars, describe how they account for the data that they do, and show the patterns of gap arguments in example derivations.

Computability and Complexity

Objectives:

- To introduce standard terminology and concepts that are used to analyse whether a given task can be performed by a computer and, if so, how efficiently. These come from the areas of Computability Theory and Complexity Theory.
- To show what results this can give us about the complexity of tasks involving languages of the different Chomsky types.
- To show that efficiency considerations hence lead one to prefer grammars with types high in the Chomsky hierarchy – a preference that pulls in the opposite direction to the requirements of expressivity investigated in the last chapter.

Comprehension:

- Why is the idea of a Turing Machine introduced in computability theory?
- What is a decidable problem?
- Is logarithmic complexity better than linear complexity?
- What is an intractable algorithm?
- For which classes of languages is recognition in the worst case intractable?
- Why is the complexity of parsing (for some senses of the word) worse than that of recognition?

Abilities:

- Understand and use terms like “decidable”, “intractable” and “polynomial complexity”.
- Appreciate the relevance of these terms to Computational Linguistics and also the limits of their usefulness.
- Analyse simple programs in terms of their complexity.

Introduction to Parsing

Objectives:

- To describe the conditions that a good parser must satisfy.
- To introduce the basic distinctions between parsing strategies:
  - top-down vs. bottom-up
  - left-right vs. right-left
  - depth-first vs. breadth-first
- To locate the standard DCG parsing strategy in the space of possible parsing strategies.

Comprehension:

- Why would it be trivial to produce a parser that was correct but not complete?
- Which kind of parsing has trouble with left-recursion?
- Is shift-reduce parsing top-down or bottom-up?
- If a parsing search space is infinite but completeness if required, would it be better to use depth-first or breadth-first?
- Is standard DCG parsing top-down or bottom-up?
- Does standard DCG parsing have trouble with epsilon productions?
- What do the shift and reduce operations do in a shift-reduce recogniser?

- Why is it useful to add an “oracle” to a left-corner interpreter?

Abilities:

- List the dimensions along which Context Free parsing algorithms may be classified.
- Explain the differences between top-down and bottom-up, depth-first and breadth-first.
- Explain what kind of parser results from the standard compilation from DCGs into Prolog.
- Give the steps taken by Prolog in parsing a sentence with a given compiled DCG.

- Describe the reasons for non-termination of parsing with certain grammars and exemplify such grammars.
- Write interpreters in Prolog for left-corner and shift-reduce recognition.
- Explain how a left-corner parser can be equipped with an oracle, and why this is useful.

Well-Formed Substring Tables

Objectives:

- To explain the inefficiencies of backtrack parsing and the need for the parser to have a memory of what it has already done. This motivates the use of both well-formed substring tables introduced in this chapter and charts discussed in the next chapter.

- To describe the CKY algorithm, the standard use of a well-formed substring table.

Comprehension:

- Give an example where a backtrack recogniser will duplicate work unnecessarily.

- Is the CKY algorithm left-to-right or right-to-left?
- Is the CKY algorithm top-down or bottom-up?
- What is the computational complexity of recognition by the CKY algorithm?
- If in the CKY algorithm categories stored in the chart were replaced by parse trees, why would the complexity then be at worst exponential?

The Active Chart

Objectives:

- To introduce dotted rules as the key to generalising the CKY algorithm.
- To show Earley's algorithm for recognition using a chart.
- To show how variations on Earley's algorithm can be made.

Comprehension:

- What is a dotted rule?
- Which kinds of dotted rules correspond to the entries made in the CKY algorithm?

- What is the difference between an active and an inactive (passive) edge?
- Does prediction apply to inactive or active edges?
- Does completion apply to active or inactive edges?
- What kind of existing edges are looked for in completion?
- Does scanning apply to active or inactive edges?
- What kind of existing edges are looked for in scanning?

Abilities:

- Explain the fundamental rule and the active chart parsing algorithm
- Exemplify the algorithm parsing a simple sentence

Introduction to Unification

Objectives:

- To motivate the need for a *unification* operation for grammars with features.
- To describe how *term unification* works in Prolog and DCGs.
- To indicate that *graph unification* has some advantages over this.
- To introduce PATR-II, a DCG-like notation which uses graph unification, rather than term unification.

Comprehension:

- What does term unification do when presented with two terms with the same functor but different number of arguments?
- What does term unification do when presented with `X` and `f(X)` (where `X` is the same in both inputs)? What should it do?
- How does one draw a complex feature structure as a DAG?
- What is reentrancy?
- How are categories like S, NP dealt with in the example PATR-II grammar?

Abilities:

- Describe the term unification algorithm and how it is used in Prolog.
- Discuss the advantages and disadvantages of term and graph unification.
- Explain the PATR-II notation and give simple example grammar rules using it.

Implementing PATR-II

Objectives:

- To show how feature structures and graph unification can be implemented in Prolog.
- To show what is involved in constructing a simple Prolog implementation of PATR-II.
- To show a new approach to subcategorisation, similar in spirit to that used in Categorial Grammar, expressed in PATR-II.

Comprehension:

- What are the main predicates that need to be defined for a PATR-II interpreter in Prolog, and what arguments do they take?
- Why are feature structures represented by lists terminating in variables?
- In the approach to subcategorisation shown, what is the format of the subcategorisation list associated with a word?

Abilities:

- Explain how representing feature-structures (graphs) as lists with a variable tail allows a graph unification operator to be defined in Prolog, and give that definition.

Parsing with Unification Grammars

Objectives:

- To spell out the possible ways that a parser for unification grammars (e.g. PATR-II grammars) might be constructed.
- To show the ways in which the design of a parser that deals with UGs directly has to be more complex than a parser for context-free grammars.

Comprehension:

- What are the three possible ways discussed of implementing a parser for unification grammars?

- What does a dotted rule look like for a unification grammar?
- Where is unification used in the fundamental rule and in prediction?
- Why do chart edges have to be copied when they combine together?
- What is the appropriate test to use to see whether a new edge "is already in the chart"?
- What technique can handle loops involving repeated prediction?
- What is the purpose of indexing in a chart parser?

Abilities:

- Design the basic components and algorithms for a chart parser for PATR-II grammars.