Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Talen en Compilers

2019 - 2020, period 2

Jurriaan Hage

Department of Information and Computing Sciences
Utrecht University

2019-11-04

11. Intermediate summary

Languages

Grammar A way to describe a language inductively.

Languages

Grammar A way to describe a language inductively.

Production A rewrite rule in a grammar.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.
Derivation	Successively rewriting from a grammar until we reach a sentence.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.
Derivation	Successively rewriting from a grammar until we reach a sentence.
Parse tree	Tree representation of a derivation.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.
Derivation	Successively rewriting from a grammar until we reach a sentence.
Parse tree	Tree representation of a derivation.
Ambiguity	Multiple parse trees for the same sentence.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.
Derivation	Successively rewriting from a grammar until we reach a sentence.
Parse tree	Tree representation of a derivation.
Ambiguity	Multiple parse trees for the same sentence.
Abstract syntax	(Haskell) Datatype corresponding to a grammar.

Languages

Grammar	A way to describe a language inductively.
Production	A rewrite rule in a grammar.
Context-free	The class of grammars/languages we consider.
Nonterminal	Auxiliary symbols in a grammar.
Terminal	Alphabet symbols in a grammar.
Derivation	Successively rewriting from a grammar until we reach a sentence.
Parse tree	Tree representation of a derivation.
Ambiguity	Multiple parse trees for the same sentence.
Abstract syntax	(Haskell) Datatype corresponding to a grammar.
Semantic function	Function defined on the abstract syntax.

Languages – typical tasks

- Given a grammar, find words in the language.
- Given a language specified as a set, find a context-free grammar.
- Given a language defined in words and by means of some examples, define a context-free grammar.
- Basic set theory: empty set, union, difference, intersection.
- Difference between the empty language and the empty word and the language containing the empty word.
- Sequences, non-empty sequences, optional elements.
- Given a grammar and a word, draw a parse tree.
- Judge whether two given derivations of a word correspond to the same parse tree or not.

Ambiguity

If the semantics of the several parse trees match, or all but one reading are undesired, we can transform the grammar such that only one reading remains.

Grammar transformations

- Inlining/abstraction.
- Introducing/eliminating $\cdot^*$, $\cdot^+$, and $\cdot?$.
- Removing unreachable productions.
- Removing duplicate productions.
- Left factoring.
- Removing left-recursion.

Grammar transformations – contd.

- Associative operators/separators.
- Introducing operator priorities.

Grammar transformations – typical tasks

- Given a grammar, apply a certain transformation.
- Given a grammar, try to simplify it, or to transform it such that it is suitable for deriving a parser.
- Given a grammar, determine if you can apply a certain transformation.
- Explain how a grammar transformation works.
- Given two grammars, try to prove their equivalence by transforming one into the other, or to prove that they are not equivalent by providing an example word that can be derived by only one grammar.

Concrete and abstract syntax

(Haskell) datatypes can be constructed systematically from a grammar:

- one datatype per nonterminal, one constructor per production, arguments of constructors correspond to nonterminals on right hand sides
- often, we can simplify: use lists for $\cdot^*$ and $\cdot^+$, use Maybe for $\cdot?$.
- also, we try to use Int, Char and String where the match is "good enough"

Concrete and abstract syntax – typical tasks

- Given a grammar, give a suitable abstract syntax.
- Given a Haskell datatype, come up with a concrete syntax.

Parser combinators

- Implementation of simple parser combinators.
- Implementation of derived combinators.
- Defining your own abstractions.
- Using parser combinators: systematic derivation from grammar and performance pitfalls.
- Lexing and parsing in one or two phases, handling of spaces.
- Constructing an abstract syntax tree as a default semantic function.

Parser combinators – typical tasks

- Given a grammar, come up with a combinator parser.
- For a certain pattern, define a derived combinator.
- Analyze the efficiency of a given parser.
- Transform the grammar underlying a certain problematic parser such that performance improves.
- Plug in a certain semantic function directly into a parser.

Semantics and compositionality

- Folds abstract from the standard pattern for defining functions over algebraic datatypes (systematic pattern matching and recursion where the datatype is recursive).
- Algebras and folds can be defined for most datatypes.
- Also families of datatypes and recursive positions wrapped into lists can be handled.
- Algebras can have various return types, in particular functions.

Semantics and compositionality – typical tasks

- Given an abstract syntax, define a corresponding algebra type and fold function.
- For a desired semantics, define a directly recursive semantic function.
- For a desired semantics, define an algebra that can be used with the fold function.
- For a desired semantics, give a suitable result type for an algebra.