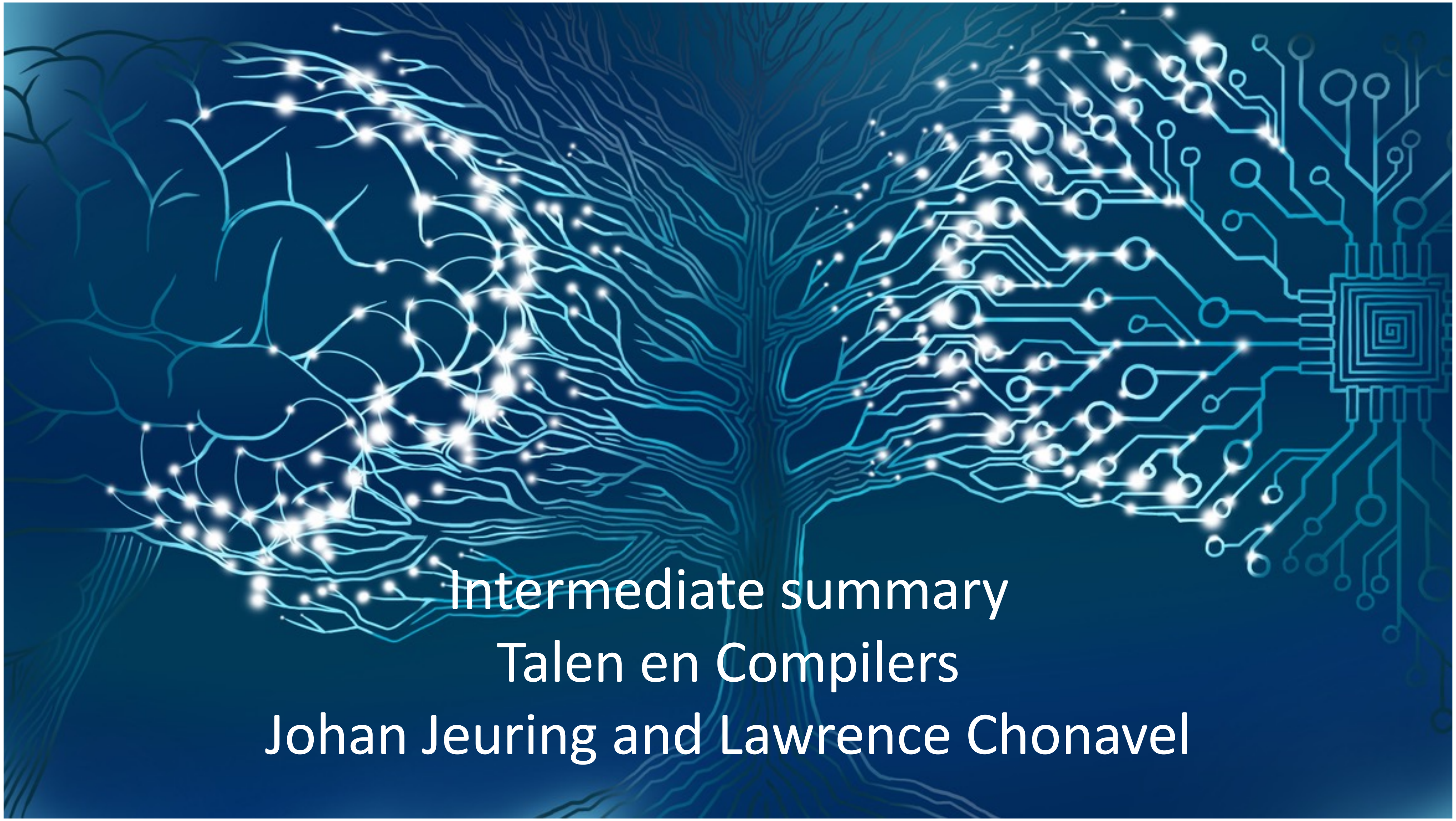

Intermediate summary
Talen en Compilers

Johan Jeuring and Lawrence Chonavel

Utrecht University

What is the midterm about?

Lecture notes: 1 – 7.3, 9.2-3,

Slides from lectures 1 (12 November 2025) – 11 (16 December 2025)

Utrecht University

Languages

Concept	Explanation
Grammar	Inductive description of a language
Production	A rewrite rule of a grammar
Context-free	Rewriting happens irrespective of the context
Nonterminal	Auxiliary symbols in a grammar
Terminal	Alphabet symbols in a grammar
Derivation	Rewrite using productions until reaching 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

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

Given a grammar and a word, add a production rule so that the word can be derived using the grammar

Utrecht University

Grammar transformations

Grammar transformation
Inlining or abstraction
Introducing or eliminating *, +, ?
Removing unreachable productions
Removing duplicate productions
Left factoring
Removing left-recursion
Associative operators or separators
Introducing operator priorities

Utrecht University

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

Utrecht University

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 * and $^+$, use Maybe for $^?$

Use Int, Char and String where the match is “good enough”

Utrecht University

Concrete and abstract syntax – typical tasks

Given a grammar, give a suitable abstract syntax

Given a Haskell datatype, come up with a concrete syntax

Utrecht University

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

Utrecht University

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 semantic function directly into a parser

Utrecht University

Semantics and compositionality

Folds abstract from the standard pattern for defining recursive functions over algebraic datatypes

Algebras and folds can be defined for most datatypes

Also families of datatypes and recursive positions wrapped into lists

Algebras can have various return types, in particular functions

Arguments represent inherited information, results synthesized

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

LR parsing

Bottom-up parsing using LR constructs a rightmost derivation of a sentence

LR parsing uses a stack, an automaton, and (often) a shift-reduce table

Utrecht University

LR parsing – typical tasks

Show a rightmost derivation of a sentence using LR parsing

Construct an LR(0) automaton for a grammar

Identify shift-reduce and reduce-reduce conflicts in an LR(0) automaton