Appendix A

Glossary

Entires marked with a star* is not part of the curriculum. Entries **in bold** are particularly important for the exam.

A.1 Term Definitions

Abstract data type*
TYPES

A type which is defined only through an interface of operations used to manipulate its value, and where the data representation is hidden.

[Wikipedia]

Abstract syntax tree
ABSTRACTION, PARSING, SYNTAX

A tree representation of the syntactic structure of a sentence; similar to a parse tree, but usually ignoring literal nonterminals and nodes corresponding to productions that don't directly contribute to the structure of the language (e.g., parenteses, productions used to encode operator priority and so on). May represent a slightly simpler language that the parse tree, for example, with operator calls desugared to function calls, and variations of a construct folded into one. Can be represented using as trees or terms, and described by an algebraic data type or a regular tree grammar.

[Wikipedia]

Abstract value*
ABSTRACTION, SEMANTICS

A value (of an abstract data type) known only through the operations used to create it. The user of an abstract data type operates on abstract values; only the implementation of the abstract data type sees the concrete value. A single abstract value may have many different concrete representations (either because there are several implementations of the data type, or because several representations mean the same – for example, $1/2 = 2/4$)

Abstraction*
ABSTRACTION

Focusing on relevant details while leaving irrelevant details unspecified or hidden away. *Data abstraction* (e.g., classes and interfaces) hides the details of how a data structure is represented, instead giving an interface to manipulate the data. *Control abstraction* (e.g., functions, methods) hides *how* something is done (the algorithm), focusing instead on *what* is done.

[Wikipedia]

Algebraic data type
SEMANTICS, TYPES

A composite data type defined inductively from other types. Typically,

each type has a number of cases or alternatives, which each case having a *constructor* with zero or more arguments.

For example, `data Expr = Int(int i) | Plus(Expr a, Expr b)`. The data type can be seen as an algebraic *signature*, with the expressions written using the constructors being *terms*. For our purposes, values may be interpreted as trees, with the constructor name being node labels, arguments being children, and atomic values and nullary constructors being leaves.

[Wikipedia]

Ambiguous grammar
AMBIGUITY, SYNTAX

A grammar for which there is a string which has more than one leftmost derivation.

– *Parsing* Requires a generalised parser, produces a parse forest.

[Wikipedia]

Analytic grammar*
PARSING, SYNTAX

A grammar which corresponds directly to the structure and semantics of a parser. For example, parser combinators and parsing expression grammars (PEGs).

[Wikipedia]

Anonymous function
SEMANTICS

A function occuring as a value, without being bound (directly) to a name. C.f. closure.

Application binary interface*
COMPILATION

Specifies how software modules or components interact with each other at the machine code level. Typically includes such things function calling conventions (whether arguments are passed on the stack or in registers, and so on), the binary layout of data structures and how system calls are done.

[Wikipedia]

Application programming interface*
ABSTRACTION

Specifies how software modules or components interact with each other.

[Wikipedia]

Associativity
AMBIGUITY, SYNTAX

A property of binary operators in parsing, indicating whether expressions such as $a+b+c$ should be interpreted as $(a+b)+b$ (left associative), $a+(b+c)$ (right associative) or as illegal (non-associative).

– *Left* Operations are grouped on the left, giving a tree which is "heavy" on the left side; typically used for arithmetic operators. Without associativity rules, grammar productions usually look like `PlusExpr ::= PlusExpr "+" MultExpr | ...`, forcing any expression containing the operator to be on the left side.

– *Right* Operations are grouped on the right, giving a tree which is "heavy" on the right side; typically used for assignment and exponentiation operators.

[Wikipedia]

Associativity rule
AMBIGUITY, SYNTAX

A disambiguation rule stating that an operator is either left-, right- or non-associative. E.g., in Rascal: `syntax Expr = left (Expr "*" Expr | Expr "/" Expr );`

Attribute grammar
PARSING, SYNTAX

A grammar where each production rule has attached attributes that are evaluated whenever the production rule is used in parsing. Can be used, for example, to build an intermediate representation directly from the

parser, or to do typechecking while parsing.

[Wikipedia]

Backend
COMPILATION

The final stage of a compiler or language processor, often tasked with low-level optimisation and code generation targeted at a particular machine architecture.

Backus-Naur form
SYNTAX

A formal notation for grammars, where productions are written `<symbol> ::= <symbol1> "literal" ...`. Often extended with support for repetition (`*`, `+`), optionality (`?` or `[]`) and alternatives (`|`).

[Wikipedia]

Bottom-up parser
PARSING

A parser that works by identifying the lowest-level details first, rather than working top-down from the start symbol. For example, an LR parser.

[Wikipedia]

Chomsky normal form*
SYNTAX

A simplified form of grammars where all the production rules are of the form $A \to BC$ or $A \to a$ or $S \to \epsilon$, where A, B and C are nonterminals (with neither B nor C being the start symbol), a is a terminal, ϵ is the empty string, and S is the start symbol. The third rule is only applicable if the empty string is in the language. Any context-free grammar can be converted to this form, and any grammar in this form is context free.

[Wikipedia]

Closure
SEMANTICS

A function (or other operation) packaged together with all the variables it can access from the surrounding scope in which it was defined. See Section 2.7.1. A related term is anonymous function, which does not necessarily imply access to variables from the surrounding scope.

[Wikipedia]

Composite data type
TYPES

A data type constructed from other types (or itself, in the case of a recursive data type), e.g., a structure or an algebraic data type.

Context-free grammar
SYNTAX

A formal grammar in which every production rule has a form of $A \to w$, where A is a single nonterminal symbol and w is a sequence of terminals and nonterminals.

[Wikipedia]

Continuation*
SEMANTICS

An abstract representation of the control state of a program. Often used in the sense of first-class continuations which let the execution state of a program be saved, passed around and then later restored.

[Wikipedia]

Cross-cutting concern*
ABSTRACTION

A programming concern, such as logging or security, which impacts many parts of a program and is difficult to decompose (cleanly separate into a library) from the rest of the system.

[Wikipedia]

Dangling else problem
AMBIGUITY, SYNTAX

A common ambiguity in programming languages (particularly those with C-like syntax) in which an optional `else` clause may be interpreted as belonging to more than one `if` sentence. Usually resolved in favour of

the closest `if`, often by an implicit disambiguation rule (at least in non-generalised parsing).

[Wikipedia]

Declarative programming*

A programming paradigm where programs are built by expressing the logic of a computation rather than the control flow; i.e., *what* should be done, rather than *how*.

[Wikipedia]

Definite clause grammar*

Syntax

A way of expressing grammars in logic programming languages such as Prolog.

[Wikipedia]

Derivation

Parsing

A sequence of production rule applications that rewrites the start symbol into the input string (i.e., by replacing a nonterminal symbol by its expansion at each step). This can be seen as a trace of a parser's actions or as a proof that the string belongs to the language.

– *Leftmost* A derivation where the leftmost nonterminal symbol is selected at every rewrite step.

– *Rightmost* A derivation where the rightmost nonterminal symbol is selected at every rewrite step.

Desugaring

Syntax, Transformation

Removal of syntactic sugar. Sometimes used in a frontend to translate convenient language constructs used by the programmer into more fundamental constructs. For example, translating Java's enhanced `for` into a more basic iterator use.

[Wikipedia]

Deterministic context-free grammar*

Ambiguity, Syntax

A context-free grammar that can be derived from a deterministic pushdown automaton (DPDA). Always unambiguous.

[Wikipedia]

Disambiguation rule

Ambiguity, Parsing, Syntax

Used to resolve ambiguities in a grammar, so that the parser yields a single unambiguous parse tree. Includes techniques such as follow restrictions, precede restrictions, priority rules, associativity rules, keyword reservation and implicit rules.

Domain-specific language

Abstraction, Languages

A language (i.e., not just a library) with abstractions targeted at a specific problem domain.

– *Benefits* Easier programming, more efficient or secure, possibly better error reports

– *Drawbacks* Lots of implementation work, language fragmentation, learning/training issues, less tooling, troublesome interoperability, possibly worse error reports

– *External DSL* A DSL defined as a separate programming language.

– *Internal or embedded DSL* A DSL defined as language-like interface to library.

[Wikipedia]

Duck typing*

Types

A typing style where the exact type of an object is not important, rather, any object is usable in any situation as long as it supports whatever methods are called on it. Used in many dynamic languages, such as Python, and in C++ templates. C.f. structural typing.

Named after the *duck test* (attributed to James Whitcomb Riley): "When I see a bird that walks like a duck and swims like a duck and

quacks like a duck, I call that bird a duck."

[Wikipedia]

Dynamic dispatch
Compilation, Languages, Semantics

The process of selecting, at runtime, which implementation of a method to call at runtime; typically based on the the actual class of the object on which the method is called (as opposed to the static type of the variable). With *multiple dispatch*, the selection is done based on some or all arguments, making it a kind of runtime overload resolution.

[Wikipedia]

Dynamic language
Languages

A language where most or all of the language semantics is processed at runtime, including aspects such as name binding and typing. May have features such as duck typing, dynamic typing, runtime reflection and introspection, and often allows code to be replaced and objects to be extended at runtime.

[Wikipedia]

Dynamic scoping
Languages, Semantics

When names are resolved by finding the closest binding in the runtime environment (i.e., the execution stack), rather than in the local lexical environment (i.e., the containing scopes at the use site). C.f. lexical scoping.

[Wikipedia]

Dynamic semantics
Semantics

Gives the meaning of a program at execution time; either in terms of values being computed, actions being performed and so on.

[Wikipedia]

Dynamic typing
Types

When type safety is enforced at runtime. Values are associated with type information, which can also be used for other purposes, such as runtime reflection. Used in languages such as Python, Ruby, Lisp, Perl, etc.

– *Benefits* Compiler may run faster; easy to load code dynamically at runtime; allows some things that are type safe but are still excluded by a static type system; easy to use duck typing to get naturally generic code with little overhead for the programmer; reflection, introspection and metaprogramming becomes easier.

– *Drawbacks* Type errors cannot be detected at compile time; rigorous testing is needed to avoid type errors; some optimisations may be difficult to perform (less of a problem with just-in-time compilation).

Dynamic typing does not imply weak typing. C.f. static typing, duck typing.

[Wikipedia]

Environment
Semantics

A mapping of names to values or types. Used in evaluation and type-checking to carry name bindings. The environment is passed around (propagated) according to the scoping rules of the language, and may use a more complicated data structure to accomodate nested and/or named scopes.

[Wikipedia]

Epsilon
Syntax

In a grammar, the empty string.

Evaluator (also Interpreter)
Compilation, Languages, Semantics

A program that executes another program.

[Wikipedia]

Extended Backus-Naur form*
Syntax

A syntax notation introduced by Niklaus Wirth, which includes notation for optionality, repetition, alternation, grouping and so on.

[Wikipedia]

Field
Types

A data member of a data structure.

Follow restriction
Ambiguity, Syntax

A disambiguation technique where a symbol is forbidden from or forced to be immediately followed by a certain terminal.

Formation rule
Semantics, Syntax

A grammar; rules for describing which strings are valid in a language. This term is used mainly in logic.

Frontend
Compilation

The first stage of a compiler or language processor, typically including a parser (possibly with a tokeniser), and a typechecker (semantic analyser). Sometimes also includes desugaring. Is typically responsible for giving the programmer feedback on errors, and translating to the internal AST or representation used by the rest of the system.

Function
Semantics

An abstraction over expressions (or more generally, over expressions, statements and algorithms).

Function type
Semantics, Types

The representation of a function in the type system. Typically includes parameter types and return type, written $t_1, ..., t_k \rightarrow t$.

Function value
Semantics

The representation of a function in an evaluator or in a dynamic semantics specification. Usually includes the parameter names and the function body. Forms a closure together with an environment giving the function's declaration scope.

Functional programming
Languages

A programming paradigm based on mathematical functions, usually without state and mutable variables. Pure functional languages have referential transparency, and typically allows Higher-order functions.

[Wikipedia]

Generalised parser
Ambiguity, Parsing

A parser that can handle the full range of context-free grammars, including nondeterministic and ambiguous grammars. For example, a GLL parser or a GLR parser.

Generative grammar*
Syntax

An approach to specifying the syntax of the language, using a set of rules that produce all strings in the language. Formalised by Noam Chomsky in the late 1950s, but the idea goes back to Pāṇini's Sanskrit grammar, 4th century BCE.

[Wikipedia]

Generative programming*
Abstraction, Languages, Transformation

Programming aided by automatic generation of code, including techniques such as generic programming, templates, aspects, code generation, etc.

[Wikipedia]

Generic programming*
ABSTRACTION, LANGUAGES

A programming style which allows the same piece of code to deal with many different types, for example through polymorphism.

[Wikipedia]

GLL parser
PARSING

An LL parsing algorithm extended to handle nondeterministic and ambiguous grammars, making it capable of parsing any context-free grammar. Unlike normal LL or recursive descent parsers, it can also handle left recursion.

[WWW]

GLR parser
PARSING

An LR parsing algorithm extended to handle nondeterministic and ambiguous grammars, making it capable of parsing any context-free grammar.

[Wikipedia]

Grammar
SYNTAX

A formal set of rules defining the syntax of a language. Formally, a tuple $\langle N, T, P, S \rangle$ of nonterminal symbols N, terminal symbols T, production rules P, and a start symbol $S \in N$. In software languages, the most frequently used kinds are context-free and regular grammars.

[Wikipedia]

Grammar in a broad sense*
SYNTAX

A structural description in software systems, and a description of structures used in software systems: a parser specification is an enriched grammar, a type definition is a grammar, an attribute grammar comprises a grammar, a class diagram can be considered a grammar, a metamodel must contain a grammar, an algebraic signature is a grammar, an algebraic data type is a grammar, a generalized algebraic data type is a very powerful grammar, a graph grammar and a tree grammar are grammars for visual concrete notation, an object grammar contains two grammars and a mapping between them.

[Paper: Toward an Engineering Discipline for Grammarware]

Higher-order function
LANGUAGES

A function which takes takes functions as arguments or returns function values.

[Wikipedia]

Imperative programming
LANGUAGES

A programming paradigm based on statements that change program state; as opposed to declarative programming. May be combined with object-oriented programming.

[Wikipedia]

Implicit disambiguation rule
AMBIGUITY, PARSING, SYNTAX

A disambiguation rule built into the parser, such as longest match for regular expressions, or resolving the dangling else problem by preferring shift over reduce in an LR parser.

Inheritance
ABSTRACTION, LANGUAGES

A technique in object-oriented programming which combines automatic code reuse with subtyping.

[Wikipedia]

Inlining
ABSTRACTION, COMPILATION, TRANSFORMATION

A technique in language processing where a call to a function or procedure is replaced by the code being called. Often used as part of code optimisation; removes abstraction introduced by the programmer.

[Wikipedia]

Island grammar*
Syntax

A grammar which describes only small parts of a language, skipping over the rest. Used, for example, to recover documentation from a program.

[Program Transformation Wiki]

Just-in-time compilation*
Compilation

A technique used in interpreted or bytecode-compiled languages where program code is compiled at runtime, during evaluation. This gives the speed advantages of compilation, while retaining the dynamic flexibility and architecture-independence of a interpreted or bytecode-based language. Can sometimes give even better performance than static compilation, since more information may be available at runtime, leading to better optimisation. Heavily used in modern implementations of JVM and .NET.

[Wikipedia]

Kleene closure
Syntax

A metasyntactic sugar for repetition: `x*` means that `x` can be repeated zero or more times. The language that the Kleene star generates, is a monoid with concatenation as the binary operation and epsilon as the identity element.

[Wikipedia]

Left factoring*
Syntax

A technique used to avoid backtracking in top-down parsing, by ensuring that the productions for a nonterminal don't have alternatives that start with the same terminals.. Used to massage a grammar to LL form.

[Wikipedia]

Left recursion*
Syntax

When production rules have the form $A \rightarrow Aa|b$, with the Nonterminal occurring directly or indirectly to the left of all terminals on the right-hand side. Must be eliminated in order to use an LL parser.

[Wikipedia]

Lexeme
Syntax

A string of characters that is significant as a group; a word or token.

Lexical analysis (also scanner, lexer or tokeniser)
Compilation, Parsing

Converting a sequence of characters (letters) to a sequence of tokens (lexemes or words).

Lexical scoping (also Static scoping)
Languages, Semantics

When names are resolved (possibly statically) by finding the closest binding in the lexical environment (i.e., by looking at the scopes that lexically contains the name). C.f. dynamic scoping.

[Wikipedia]

Lexical syntax
Syntax

Describes (often using a Regular grammar) the syntax of tokens; e.g., what constitutes an identifier, a number, different operators and the whitespace that separates them.

[Wikipedia]

Literate programming*
Languages, Syntax

A programming style where programs can be read as documents that explain the implementation, with explanations in a natural language. Tools allow programs to be compiled as either code or documents.

[Wikipedia]

LL parser
PARSING

A table-driven top-down parser, similar to a recursive descent parser. Has trouble dealing with left recursion in production rules, so the grammar must typically be left factored prior to use. The LL parser reads its input in one direction (left-to-right) and produces a leftmost derivation, hence the name LL. Often referred to as LL(k), where the k indicates the number of tokens of lookahead the parser uses to avoid backtracking.

[Wikipedia]

LL grammar
SYNTAX

A grammar that can be parsed by a LL parser.

[Wikipedia]

Logic programming

A declarative programming paradigm based on formal logic, inference and reasoning. Useful for many purposes, including formal specification of language semantics. Prolog is the most well-known logic language.

[Wikipedia]

LR parser
PARSING

A bottom-up parser that can handle deterministic context-free languages in linear time. Common variantes are LALR parsers and SLR parsers. It reads its input in one direction (left-to-right) and produces a rightmost derivation, hence the name LR.

[Wikipedia]

LR grammar
SYNTAX

A grammar that can be parsed by a LR parser.

[Wikipedia]

Massaging*
PARSING, SYNTAX, TRANSFORMATION

The act of modifying a grammar to make it fit a particular technology or purpose.

Megamodel*

A result of megamodelling — a model which elements are software languages, models, metamodels, transformations, etc

[Paper: On the Need for Megamodels]

Member (also Field)
TYPES

An element of a structure or class; a field, method or inner class/-type.

Method (also Member function)
LANGUAGES, SEMANTICS

A function which is a member of a class. Typically receives a self-reference to an object as an implicit argument.

[Wikipedia]

Mixin*

A partial class (data fields and methods) that can be used to plug functionality into another class using inheritance.

[Wikipedia]

Multi-paradigm programming*
LANGUAGES

Programming which combines several paradigms, such as functional, imperative, object-oriented or logic programming. Languages that support multiple paradigms include, for example, C++, Scala, Oz, Lisp, and many others.

[Wikipedia]

Name binding
COMPILATION, SEMANTICS

A part of language processing where names are associated with their declarations, according to scoping and namespace rules. A name's binding is

typically determined by checking the environment at the use site.

– *Static* When done statically (or *early*), name binding is often combined with typechecking.

– *Dynamic* Names are bound at runtime; also applies to dynamic dispatch (where it is sometimes called *late* or *virtual* binding), where certain properties (such as types) may be known statically, but the exact operation called is determined at runtime.

[Wikipedia]

Named tuple
TYPES

A tuple where the elements are named, like in a structure. Often exhibits structural type equivalence, even in languages that normally use nominative type equivalence

Namespace
SEMANTICS

Some kind name grouping that makes it possible to distinguish different uses of the same name. For example, having variable names be distinct from type names; or treating names in one module as distinct from the same names in another module (In this sense, namespaces are related to scope).

[Wikipedia]

Nominative type equivalence* (also Nominal/Nominative type system)
TYPES

A system where type equivalence or compatibility is determined based on the type names (or, more strictly, which declaration the names refer to) and not the structure of the type. C.f. structural type equivalence.

[Wikipedia]

Nonterminal footprint*
SYNTAX

A non-recursive measure of nonterminal symbol usage in a grammatical expression: a multiset of presence indicators (1 for the nonterminal itself, ? for its optional use, * for its Kleene closure, etc). A usefulness of a footprint for grammar matching depends on how rich the metalanguage is.

Nonterminal symbol
SYNTAX

A symbol in a grammar which is defined by a production. Can be replaced by terminal symbols by applying the production rules of the grammar. In a context-free grammar, the left-hand side of a production rule consists of a single nonterminal symbol.

Object-oriented programming*
LANGUAGES

A programming paradigm based on modelling interactions between objects. Objects have fields and methods and encapsulate state. Provides data abstraction, and usually supports inheritance, dynamic dispatch and subtype subtype polymorphism.

[Wikipedia]

Optimisation
COMPILATION, TRANSFORMATION

The process of transforming program code to make it more efficient, in terms of time or space or both.

[Wikipedia]

Overloading
SEMANTICS

When the same name is used for multiple things (of the same kind). For example, several functions with the same name, distinguished based on the parameter types. C.f. namespace, where the same name can have different meanings in different context (e.g., type names are distinct from variable names).

[Wikipedia]

Overload resolution
COMPILATION, SEMANTICS

A compilation step, usually combined with typechecking, where the name of an overloaded function is resolved based on the types of the actual arguments. C.f. dynamic dispatch, which does something similar at runtime.

[Wikipedia]

Parse forest
AMBIGUITY, PARSING, SYNTAX

The parse trees that are the result of parsing an ambiguous grammar using a generalised parser.

Parse tree
PARSING, SYNTAX

A tree that shows the structure of a string according to a grammar. The tree contains both the tokens of the original string, and a trace of the derivation steps of the parse, thus showing how the string is a valid parse according to the grammar. Typically, each leaf corresponds to a token, each interior node corresponds to a production rule, and the root node to a production rule of the start symbol.

Parser
PARSING

A program that recognises input according to some grammar, checking that it conforms to the syntax and builds a structured representation of the input.

Parser combinator*
PARSING

A way of expression a grammar and a parser using higher-order functions. Each combinator accepts parsers as arguments and returns a parser.

[Wikipedia]

Parsing expression grammar*
PARSING, SYNTAX

A form of analytic grammar, giving rules that can be directly applied to parse a string top-down. Similar to a context-free grammar, but the rules are unambiguously interpreted; for example, alternatives are tried in order. Related to parse combinators. PEGs are a useful and straightforward technique for parsing software languages.

It is suspected that there are context-free languages that cannot be parsed by a PEG, but this has not been proved.

[Wikipedia]

Parsing
PARSING

Recovering the grammatical structure of a string. The task done by a parser.

Pattern matching
LANGUAGES, TRANSFORMATION

A technique for comparing (typically algebraic) data structures, where one or both structures may contain variables (sometimes refered to as meta-variables). Upon successful match, variables are bound to the corresponding substructure from the other side. Related to unification in Prolog, but often more restricted.

[Wikipedia]

Polymorphism*
LANGUAGES, TYPES

– *Ad hoc* Another name for function or operator overloading.

– *Parametric* When a function or data type is generic and handle values of different types in the same; for instance, `List<T>` – a list with elements of an arbitrary type. The specific type in question is often given as a parameter, hence the name.

– *Subtype* When an object belonging to a subtype can be used in a place

where the supertype is expected (as with classes and inheritance in object-oriented programming).

[Wikipedia]

Precede restriction
AMBIGUITY, SYNTAX

A disambiguation technique where a symbol is forbidden from or forced to be immediately preceded by a certain terminal.

Predictive parser
PARSING

A recursive descent parser which does not require backtracking. Instead, it looks ahead a finite number of tokens and decides which parsing function should be called next. The grammar must be LL(k) for this to work, where k is the maximum lookahead.

Priority rule
AMBIGUITY, PARSING, SYNTAX

A disambiguation rule declaring an operator's priority/precedence. E.g., in Rascal: `syntax Expr = Expr "*" Expr > Expr "+" Expr;`

Procedural programming

A programming paradigm based around procedure calls. Sometimes considered the same as imperative programming and typically based on structured programming.

[Wikipedia]

Production rule
SYNTAX

A rule describing which symbols may replace other symbols in a grammar. In a context-free grammar, the left-hand side consists of a single nonterminal symbol, while the right-hand side may be any sequence of terminals and nonterminals. For example, `Expr ::= Expr "+" Expr` says that anywhere you may have an expression, you can have an expression plus another expression. The rules may be used to generate syntactically correct strings, by applying them as rewrite rules starting with the start symbol, or be used to parse strings, e.g., in a top-down parser or bottom-up parser.

Program slicing*
TRANSFORMATION

A program transformation technique where all code that is irrelevant to a certain set of inputs or outputs is removed. Applied *forwards*, any code not directly or indirectly using a selection of inputs is discarded; applied *backwards*, all code that does not contribute to the computation of the selected outputs is discarded. Mainly used in debugging (e.g., to find the code that might have contributed to an error), but sometimes also as an optimisation technique. Originally formalised by Mark Weiser in the early 1980s.

[Wikipedia]

Recogniser
PARSING

A program that recognises input according to some grammar, giving an error if it does not conform to the grammar, but does not build a data structure.

Record, Record type
TYPES

See Structure, Structure type.

[Wikipedia]

Recursive data type*
TYPES

A composite data type, such as an algebraic data type which may contain itself. Used, for example, to define data structures such as lists and trees.

Recursive descent parser
PARSING

A top-down parser built from mutually recursive functions, where each function typically implements

one production rule of the grammar.

[Wikipedia]

Referential transparency
LANGUAGES, SEMANTICS

When an expression can be replaced by its value without changing the meaning of the program; i.e., it will evaluate to the same value every time and not cause side effects. Usually a property of functional programming languages.

[Wikipedia]

Regular expression
SYNTAX

A formalism for describing a regular grammar, using the normal alphabet mixed with special *metasyntactic* symbols, such as the Kleene star. Commonly used to specify the lexical syntax of a language, and also for searching and string matching in many different applications.

[Wikipedia]

Regular grammar
SYNTAX

A formal grammar where every production rule has the form $A \rightarrow aB$ (for a *right regular grammar*), or $A \rightarrow a$ or $A \rightarrow \epsilon$, where A and B are nonterminal symbols and a is a terminal symbol, and ϵ is the empty string. Alternatively, the first production form may be $A \rightarrow Ba$, for a *left regular grammar*.

– *Limitations* Can't express arbitrary nesting, such as nested parentheses or block structure in a language.

[Wikipedia]

Reserve rule
AMBIGUITY, SYNTAX

A disambiguation rule which states that a grammar symbol cannot match some constraint. For example, identifiers could be defined as any word matching `[a-zA-Z]+` *except* `if`, `while`, ...

Scannerful parsing
PARSING

Parsing is divided into two parts; a tokeniser that deals with the lexical syntax and a parser that deals with the sentence syntax.

– *Benefits* Faster than scannerless parsing, because the lexical syntax is specified with a regular grammar which can be parsed very efficiently.

– *Drawbacks* Cannot deal with arbitrary composition of languages.

Scannerless parsing
AMBIGUITY, PARSING, SYNTAX

When scanning and parsing is unified into one process that deals with with the input characters directly.

– *Benefits* Can parse combinations of languages that have different lexical syntax. Lexical syntax can be context-free, not just regular.

– *Drawbacks* Slower than scannerful parsing. Can lead to hard to find lexical ambiguities.

Scope
SEMANTICS

A collection of identifier bindings – i.e., what is captured by the environment at some point in the code or in time.

– *Nested* With nested scopes, variables in inner scopes may *shadow* those in outer scopes, and variables are removed as control flows out of the scope. Variable shadowing may be forbidden in some languages.

– *Named* With named scopes, we can refer to names in scopes that are not ancestors of the current scope. For example, with C++ classes and namespaces and Java packages and (static) classes.

[Wikipedia]

Semantic analysis
COMPILATION, SEMANTICS

A phase of language processing that enforces the static semantics

of a language. Includes typechecking, name binding, overload resolution and checking other static constraints.

Start symbol
SYNTAX

The nonterminal symbol in a grammar that generates all valid strings in a language.

Static semantics
SEMANTICS, TYPES

The part of language semantics which is processed at compile time (statically). Often includes constraints that might be part of the syntax, but which is done separately in order to keep the grammar context-free. Includes concepts like name binding and typechecking, and is used to eliminate a large class of invalid or erroneous programs. See static typing.

Static typing
TYPES

When type safety is enforced at compile type (though some tests, such as for typecasting, may be done at runtime).

– *Benefits* Detects a large an important class of errors (type errors) at compile time; enables advanced optimisations and efficient memory use.

– *Drawbacks* Type system may become either overly complicated or overly restrictive; doesn't help with non-type errors; makes dynamic loading of code somewhat more complicated; type declarations may be cumbersome if the language lacks type inference.

Strong typing
TYPES

When a language (to some degree) enforces type safety.

[Wikipedia]

Structural type equivalence* (also Structural type system)
TYPES

A system where two types are equal or compatible if they have the same structure; e.g., have the same fields with the same types in the same order. C.f. nominative type equivalence.

[Wikipedia]

Structure, Structure type (also Record, Record type)
TYPES

A composite data type with named fields members; such as `struct` in C or `record` in Pascal. Similar to (or same as, with structural type equivalence) a named tuple.

[Wikipedia]

Structured programming*
LANGUAGES

A programming paradigm where the clarity of programs is improved by nestable language constructs like `if`, `while`, as opposed to conditional jumps.

[Wikipedia]

Syntactic sugar
SYNTAX

See Desugaring.

[Wikipedia]

Terminal symbol
SYNTAX

An elementary symbol in the language defined by a grammar, which cannot be changed/matched by the production rules in the grammar (i.e., the symbol doesn't occur (alone) on the left-hand side of a production). Corresponds to a token or an element of the alphabet of a language.

Token
PARSING

A lexeme or group of characters that forms a basic unit of parsing, categorised according to type, e.g., identifier, number, addition operator, etc.

Forms the alphabet of the parser in scannerful parsing.

[Wikipedia]

Tokeniser (also lexer or scanner)
PARSING

A program that performs lexical analysis, grouping and classifying input into tokens.

Top-down parser
PARSING

A parser using a strategy where the top-level constructs are recognised first, starting with the start symbol. The parser starts at the root of the parse tree, and builds it top-down, according to the rules of the grammar. Includes LL parsing and Recursive descent parsing.

Typechecker
COMPILATION, TYPES, SEMANTICS

A program that detects type errors, ensuring type safety in a statically typed language. Often combined with other static semantic checks and processing, such as overload resolution, name binding, and checking access restrictions on names. Can be seen as a form of abstract interpretation, where the abstract values are the types, and all operations are defined according to their static semantics (i.e., type signature) rather than their dynamic semantics. C.f. semantic analysis.

[Wikipedia]

Type inference
TYPES

Automatic deduction of the types in a language. Used with static typing to avoid having to declare types for variables and functions. Particularly useful in generic programming and type polymorphism where type expressions can become quite complicated. Used, e.g., in Haskell and Standard ML.

[Wikipedia]

Type safety
LANGUAGES, TYPES

Whether a language protects against type errors, such as when a value of one data type is interpreted as another type (e.g., an `int` as a `float` or as a pointer to a string).

[Wikipedia]

Typecasting* (also type conversion)
TYPES

Forced conversion from one type to another. In a languages with weaker type systems, may lead to data being misinterpreted.

[Wikipedia]

Unification*
LANGUAGES

A pattern matching-like operation, where the goal is to find an assignment of variables so that two terms become equal. Used heavily in Prolog.

[Wikipedia]

Weak typing
TYPES

When a language (to some degree) does not enforces type safety.

[Wikipedia]

Yield*
PARSING, TRANSFORMATION

The *yield* of a parsetree is the unparsed input text.

A.2 Tag Index

Abstraction

Ambiguity

Ambiguous grammar (p.42) — **Associativity** (p.42) — **Associativity rule** (p.42) — Dangling else problem (p.43) — Deterministic context-free grammar* (p.44) — **Disambiguation rule** (p.44) — Follow restriction (p.46) — **Generalised parser** (p.46) — Implicit disambiguation rule (p.47) — **Parse forest** (p.51) — Precede restriction (p.52) — **Priority rule** (p.52) — Reserve rule (p.53) — **Scannerless parsing** (p.53)

Compilation

Application binary interface* (p.42) — Backend (p.43) — Dynamic dispatch (p.45) — **Evaluator** (p.45) — **Frontend** (p.46) — Inlining (p.47) — Just-in-time compilation* (p.48) — Lexical analysis (p.48) — **Name binding** (p.49) — Optimisation (p.50) — Overload resolution (p.51) — **Semantic analysis** (p.53) — **Typechecker** (p.55)

Languages

Domain-specific language (p.44) — Dynamic dispatch (p.45) — Dynamic language (p.45) — **Dynamic scoping** (p.45) — **Evaluator** (p.45) — Functional programming (p.46) — Generative programming* (p.46) — Generic programming* (p.47) — Higher-order function (p.47) — Imperative programming (p.47) — Inheritance (p.47) — **Lexical scoping** (p.48) — Literate programming* (p.48) — Method (p.49) — Multi-paradigm programming* (p.49) — Object-oriented programming* (p.50) — Pattern matching (p.51) — Polymorphism* (p.51) — Referential transparency (p.53) — Structured programming* (p.54) — Type safety (p.55) — Unification* (p.55)

Parsing

Abstract syntax tree (p.41) — Analytic grammar* (p.42) — Attribute grammar (p.42) — Bottom-up parser (p.43) — Derivation (p.44) — **Disambiguation rule** (p.44) — **Generalised parser** (p.46) — GLL parser (p.47) — GLR parser (p.47) — Implicit disambiguation rule (p.47) — Lexical analysis (p.48) — LL parser (p.49) — LR parser (p.49) — Massaging* (p.49) — **Parse forest** (p.51) — **Parse tree** (p.51) — **Parser** (p.51) — Parser combinator* (p.51) — Parsing expression grammar* (p.51) — Parsing (p.51) — Predictive parser (p.52) — **Priority rule** (p.52) — **Recogniser** (p.52) — Recursive descent parser (p.52) — **Scannerful parsing** (p.53) — **Scannerless parsing** (p.53) — **Token** (p.54) — **Tokeniser** (p.55) — Top-down parser (p.55) — Yield* (p.55)

Semantics

Abstract value* (p.41) — Algebraic data type (p.41) — Anonymous function (p.42) — **Closure** (p.43) — Continuation* (p.43) — Dynamic dispatch (p.45) — **Dynamic scoping** (p.45) — **Dynamic semantics** (p.45) — Environment (p.45) — **Evaluator** (p.45) — Formation rule (p.46) — **Function** (p.46) — **Function type** (p.46) — **Function value** (p.46) — **Lexical scoping** (p.48) — Method (p.49) — **Name binding** (p.49) — **Namespace** (p.50) — Overloading (p.50) — Overload resolution (p.51) — Referential transparency (p.53) — **Scope** (p.53) — **Semantic analysis** (p.53) — **Static semantics** (p.54) — **Typechecker** (p.55)

Syntax

Abstract syntax tree (p.41) — Ambiguous grammar (p.42) —