Automatic Generation of Efficient Compilers

using a complete tool box containing:

Rex	**scanner generator**
Lalr	**LALR(1) parser generator**
Ell	**LL(1) parser generator**
Ast	**generator for abstract syntax trees**
Ag	**attribute evaluator generator (OAG)**
Estra	**transformation of attributed syntax trees**
Beg	**back end generator**
Reuse	**library of reusable modules**

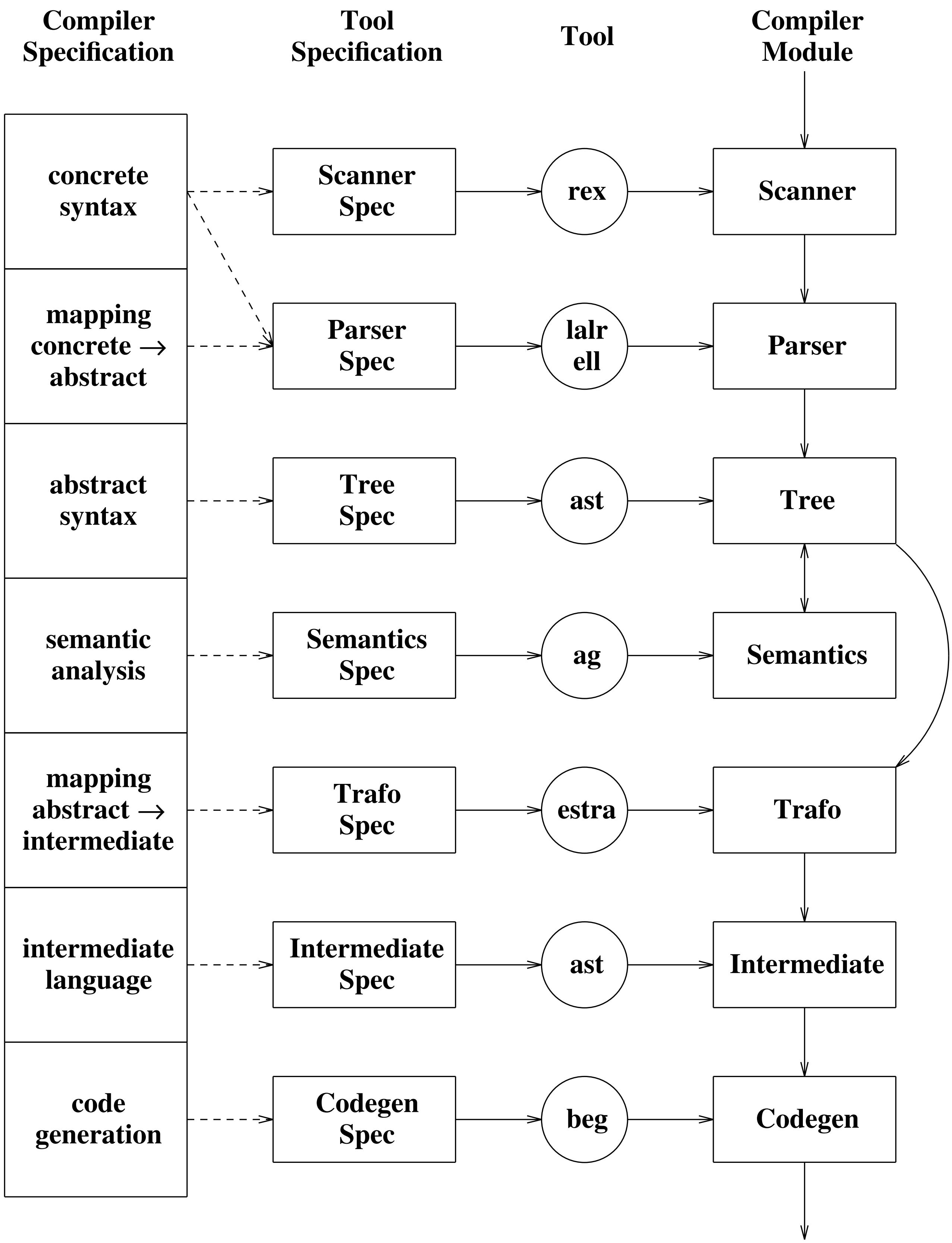
advantages:

- specification replaces implementation
- tools reduce construction effort
- consistency checks avoid errors
- program generation increases reliability
- efficiency comparable to hand-written implementations

common properties:

- implementation languages are Modula-2 or C
- target languages are Modula-2 or C

Compiler Generation



Rex

a scanner generator

- specifications are based on regular expressions

- actions are composed of target language statements

- right context is handled by an additional regular expression

- left context is handled by start states

- conflicts are resolved in favour of longest match and first rule given

- provides line and column of every token

- can normalize tokens to lower or upper case letters

- knows predefined rules to skip white space

- can handle include files

- generates table-driven scanners

- generates efficient scanners

- generates scanners in Modula-2 or C

- scanners process up to 180,000 lines/minute (on MC 68020)

- scanners are 4 times faster than those of LEX

Lalr

a parser generator

- processes LALR(1) grammars
- actions are composed of target language statements
- allows to evaluate an S-Attribution during parsing
- reports grammar conflicts by easy to understand derivation trees
- resolves grammar conflicts with precedence and associativity
- generates automatic error reporting, recovery, and repair
- generates table-driven parsers
- generates efficient parsers
- generates parsers in Modula-2 or C
- parsers process up to 35,000 tokens/sec. or 580,000 lines/min. (on MC 68020)
- parsers are 3 times faster than those of YACC

Ell

a parser generator

- processes LL(1) grammars in extended BNF
- actions are composed of target language statements
- allows to evaluate an L-Attribution during parsing
- generates automatic error reporting, recovery, and repair
- generates recursive descent parsers
- generates efficient parsers
- generates parsers in Modula-2 or C
- parsers process up to 55,000 tokens/sec. or 900,000 lines/min. (on MC 68020)

Ast

a generator for abstract syntax trees

- generates abstract data types (program modules) to handle trees

- the trees may be attributed

- besides trees graphs are handled as well

- nodes may be associated with arbitrary many attributes of arbitrary type

- specifications are based on extended context-free grammars

- common notation for concrete and abstract syntax

- as well as for attributed trees and graphs

- an extension mechanism provides single inheritance

- trees are stored as linked records

- generates efficient program modules

- generates modules in Modula-2 or C

- provides many tree operations (procedures):

- node constructors combine aggregate notation and storage management

- ascii graph reader and writer

- binary graph reader and writer

- reversal of lists

- top down and bottom up traversal

- interactive graph browser

Ag

an attribute evaluator generator

- processes ordered attribute grammars (OAGs)
- processes higher order attribute grammars (HAGs)
- operates on abstract syntax
- is based on tree modules generated by Ast
- the tree structure is fully known
- terminals and nonterminals may have arbitrary many attributes
- attributes can have any target language type
- allows tree-valued attributes
- differentiates input and output attributes
- allows attributes local to rules
- allows to eliminate chain rules
- offers an extension mechanism (single inheritance)
- attributes are denoted by unique selector names
  instead of nonterminal names with subscripts

Ag

an attribute evaluator generator (continued)

- attribute computations are expressed in the target language
- attribute computations are written in a functional style
- attribute computations can call external functions
- non-functional statements and side-effects are possible
- allows to write compact, modular, and readable specifications
- AGs can consist of several modules
- the context-free grammar is specified only once
- checks an AG for completeness of the attribute computations
- checks for unused attributes
- checks an AG for the classes SNC, DNC, OAG, LAG, and SAG
- the evaluators are directly coded using recursive procedures
- generates efficient evaluators
- generates evaluators in Modula-2 (or C)

Estra

a generator for transformations of attributed syntax trees

- is based on tree modules generated by Ast

- specifications are rule based

- a rule consists of a pattern and an action

- actions are composed of target language statements

- patterns describe tree fragments

- several transformations can be specified

- subtrees can be transformed in any order

- subtrees can be transformed several times

- subtrees can be transformed by several transformations

- inherited and synthesized attributes can be evaluated

- ambiguities are resolved using costs

- application of rules can be restricted by conditions

- pattern-matching by directly coded dynamic programming algorithm

- or by table-driven tree pattern-matcher

- generates efficient transformation modules

- generates transformation modules in Modula-2