IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS

IAE Academy free study material · Updated 19 October 2026

Compiler Design Formula Sheet

INSIDE

- 20 formulas in 5 topics
- When to use each one
- Taken from the free topic notes on iaeacademy.in

Online, one page per topic: **iaeacademy.in/study/compiler-design/formula-sheet**

FREE COUNSELLING

Preparing for GATE CS? Talk to IAE Academy.

Or call **+91 84204 36524**

Every day, 10:30 am to 7:30 pm

Chat on WhatsApp

Parsing: top-down and bottom-up parsers

REMOVING IMMEDIATE LEFT RECURSION

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$
$$\Downarrow$$
$$A \to \beta_1 A' \mid \cdots \mid \beta_n A'$$
$$A' \to \alpha_1 A' \mid \cdots \mid \alpha_m A' \mid \varepsilon$$

Conditions: no β_i starts with A, and no α_i is ε. A' is a new nonterminal. The new grammar has $n + m + 1$ alternatives in total: n for A and $m + 1$ for A'.

LEFT FACTORING

$$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \gamma$$
$$\Downarrow$$
$$A \to \alpha A' \mid \gamma$$
$$A' \to \beta_1 \mid \beta_2$$

Applies when two or more alternatives of A share a nonempty prefix α (take the longest one); A' is a new nonterminal, and an empty β_i becomes ε. Repeat until no two alternatives of any nonterminal share a prefix.

LL(1) CONDITION

$$\text{For every pair } A \to \alpha \mid \beta: \quad \text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset, \ \text{ and if } \beta \Rightarrow^* \varepsilon \text{ then } \text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$$

Also at most one alternative of A may derive ε. The condition holds for every nonterminal exactly when no cell of the LL(1) table has two entries.

MOVES OF A DETERMINISTIC PARSER

$$\text{moves} = |w| + p$$

Applies to LL(1) and LR parsers on an accepted string w of $|w|$ tokens whose parse tree has p internal nodes, not counting the augmented start S'. In LL, $|w|$ counts the matches and p the expansions; in LR, $|w|$ counts the shifts and p the reductions. A node for an ε-production counts in p. Accept, and the final match of `$`, are not counted unless the question says so.

LR(1) CLOSURE

$$[A \to \alpha \cdot B\beta,\ a] \ \Rightarrow\ \text{add } [B \to \cdot\gamma,\ b] \text{ for every production } B \to \gamma \text{ and every } b \in \text{FIRST}(\beta a)$$

If β is empty or nullable, a itself is among the new lookaheads. The initial item is $S' \to \cdot S$ with lookahead `$`. Goto works exactly as in LR(0), carrying the lookaheads along unchanged. Reduce by $A \to \alpha$ only on the item's own lookahead.

GRAMMAR-CLASS HIERARCHY

$$\text{LR}(0) \subset \text{SLR}(1) \subset \text{LALR}(1) \subset \text{LR}(1), \qquad \text{LL}(1) \subset \text{LR}(1)$$

Each inclusion is proper. A grammar class here means the set of grammars the method parses without conflict. LL(1) is **not** contained in LALR(1), SLR(1) or LR(0), and none of those is contained in LL(1): LL(1) is incomparable with each of them. More generally every LL(k) grammar is LR(k).

Syntax-directed translation

FORM OF A SYNTHESIZED RULE

$$A.s = f\,(X_1.a_1,\ \ldots,\ X_n.a_n) \quad \text{for the production } A \to X_1 X_2 \cdots X_n$$

The attribute being defined belongs to the head A; its arguments are attributes of the body symbols (or of A itself).

FORM OF AN INHERITED RULE

$$X_i.a = g\left(A.b,\ X_1.c_1,\ \ldots,\ X_n.c_n\right) \quad \text{for the production } A \to X_1 X_2 \cdots X_n$$

The attribute being defined belongs to a body symbol X_i; the arguments may be attributes of the head and of the body symbols.

L-ATTRIBUTED CONDITION

$$X_i.\text{inh} = g\left(A.\text{inh},\ X_1.\text{any},\ \ldots,\ X_{i-1}.\text{any}\right), \quad \text{no dependence on } X_{i+1}, \ldots, X_n \text{ or on } A.\text{syn}$$

Also allowed: dependence on other attributes of X_i itself, provided that creates no cycle. Not allowed: dependence on a synthesized attribute of the head A. Synthesized attributes of A may use any attribute in the production, including A 's inherited attributes, provided no cycle results.

Runtime environments: activation records and storage allocation

LIVE ACTIVATIONS

$$\text{stack depth at a point} = \text{number of nodes on the root-to-current path of the activation tree}$$

Holds under stack allocation, where every live activation has exactly one record on the stack. Counts the records on the stack at one instant; the maximum over the run is the peak stack depth. Includes the main program's record.

ACCESS-LINK HOPS FOR A VARIABLE

$$\text{links to follow to reach a variable} = n_{\text{current}} - d_{\text{var}}$$

Static scoping with access links. A procedure at nesting depth n_{current} uses a variable whose nearest enclosing declaration is at depth $d_{\text{var}} \le n_{\text{current}}$. Start at the current record and follow access links this many times; 0 means the variable is local.

ACCESS LINK OF A CALLEE

$$\text{links followed to set the callee's access link} = n_p - n_q + 1$$

Procedure p (depth n_p) calls procedure q (depth n_q), which is visible from p, so $n_q \le n_p + 1$. Follow this many access links from p's record; the record reached is the most recent activation of q's lexical parent, and q's access link is set to it. Cases: $n_q = n_p + 1$ (q declared inside p) gives 0, so the link points to p itself; $n_q = n_p$ gives 1, so the link equals p's own access link; $n_q < n_p$ gives $n_p - n_q + 1$ links.

Intermediate code generation

INSTRUCTION AND TEMPORARY COUNT FOR AN ASSIGNMENT

$$\text{temporaries} = n, \qquad \text{instructions} = n + 1$$

For $x = E$ where E has n operator occurrences and nothing is shared. The scheme gives each operator a new temporary (temporaries are never reused) and then copies the last one to x. With a DAG, replace n by the number of operator nodes in the DAG.

ONE-DIMENSIONAL ARRAY ADDRESS

$$\text{addr}(A[i]) = \text{base} + (i - \text{low}) \times w$$

w is the width of one element in bytes and low is the lower bound of the index. For a 0-based array the term is just $i \times w$.

TWO-DIMENSIONAL ARRAY ADDRESS, ROW-MAJOR

$$\text{addr}(A[i][j]) = \text{base} + \big((i - l_1)\, n_2 + (j - l_2)\big) \times w$$

$n_2 = u_2 - l_2 + 1$ is the number of columns, and l_1, l_2 are the lower bounds. Row-major is the default for C-style arrays unless stated otherwise.

TWO-DIMENSIONAL ARRAY ADDRESS, COLUMN-MAJOR

$$\text{addr}(A[i][j]) = \text{base} + \big((j - l_2)\, n_1 + (i - l_1)\big) \times w$$

$n_1 = u_1 - l_1 + 1$ is the number of rows. Use this only when the question says the layout is column-major.

ASSIGNMENTS IN SSA FORM

$$\#\text{assignments in SSA} = \#\text{assignments in the code} + \#\phi\text{-functions}$$

Each original assignment gets exactly one new name, and each ϕ is one more assignment. Since each SSA name is assigned once, this is also the number of names created by assignments and ϕ-functions. Variables that are only read and never assigned (inputs) keep their names and are not counted.

Local optimisation and data-flow analysis

LIVE VARIABLES (BACKWARD, UNION)

$$\text{IN}[B] = \text{use}[B] \cup (\text{OUT}[B] - \text{def}[B])\,, \qquad \text{OUT}[B] = \bigcup_{S \in \text{succ}(B)} \text{IN}[S]$$

A variable is live at a point if its current value may be read later before being overwritten. Initialise IN[B] = ø for all B; OUT of an exit block is ø unless the question names live-out variables.

REACHING DEFINITIONS (FORWARD, UNION)

$$\text{OUT}[B] = \text{gen}[B] \cup (\text{IN}[B] - \text{kill}[B])\,, \qquad \text{IN}[B] = \bigcup_{P \in \text{pred}(B)} \text{OUT}[P]$$

A definition reaches a point if some path from it to that point does not redefine the variable. Initialise OUT[B] = ø; IN[entry] = ø.

AVAILABLE EXPRESSIONS (FORWARD, INTERSECTION)

$$\text{OUT}[B] = e_\text{gen}[B] \cup \big(\text{IN}[B] - e_\text{kill}[B]\big)\,, \qquad \text{IN}[B] = \bigcap_{P \in \text{pred}(B)} \text{OUT}[P]$$

An expression is available at a point if on every path to it the expression has been computed and its operands not changed since. IN[entry] = ø; initialise OUT[B] to the set U of all expressions for every other block.

FROM IAE ACADEMY

Preparing for GATE CS?

IAE Academy teaches GATE CS in Ballygunge, Kolkata, and online. Counselling is free: ask us about batches, fees or your preparation.

FREE COUNSELLING

Chat on WhatsApp

Call +91 84204 36524

Open every day, 10:30 am to 7:30 pm.

Scan to chat on WhatsApp

THE GATE CS COURSE

Classroom batches in Ballygunge and online batches. Fees and details are on the course page.

Scan: course page

See the course

FREE ON IAEACADEMY.IN

Notes and solved previous-year questions for Compiler Design, topic by topic.

Scan: the notes

Open the notes

Found this useful? Send it to a friend preparing for GATE.

Send to a friend on WhatsApp

IAE Academy free study material. The formulas come from the free Compiler Design notes on iaeacademy.in. Prepared and published by IAE Academy, Kolkata. Free to read and free to share, unaltered and with attribution.

IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE coaching in Kolkata since 2012 · EC, EE, IN, CS and IT, DA, BT and CE · in the classroom and online

227A Rash Behari Ave, Ballygunge, Kolkata 700019

+91 84204 36524 · iaeacademy.in · Updated 19 October 2026

Get directions