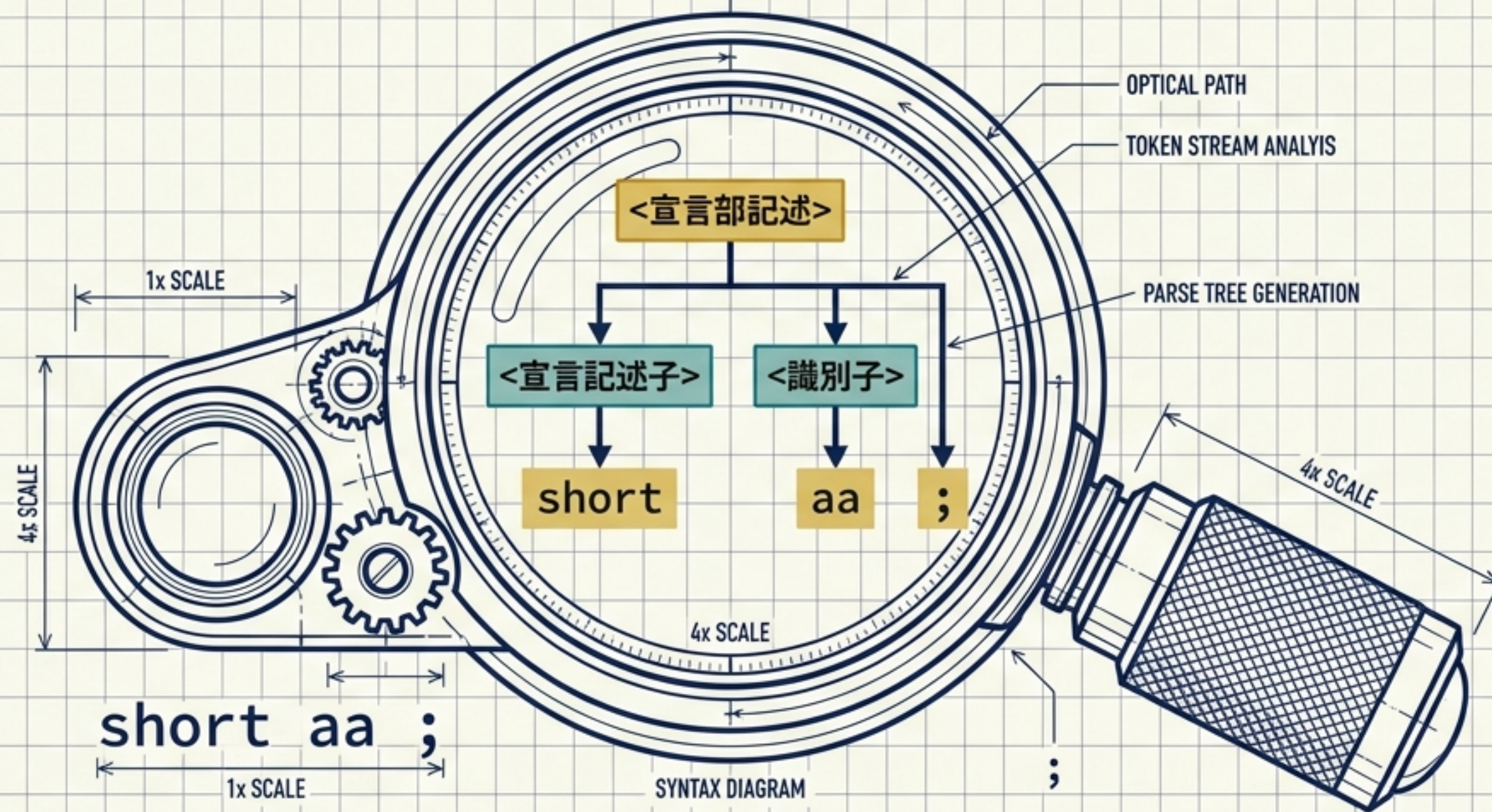
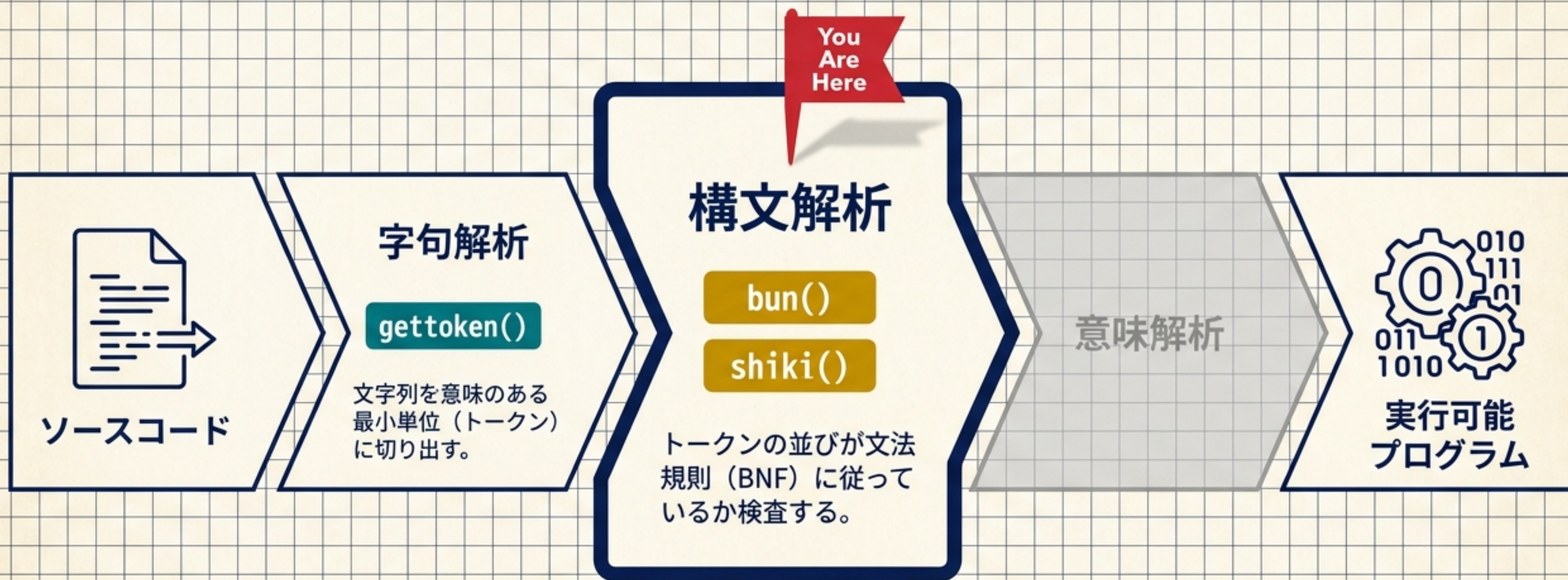


構文解析の完全攻略

解答なき解説：自力で解くための思考回路



言語処理の全貌と「構文解析」の立ち位置



捜査の基本ツール：BNF記法（拡張バックス記法）の解説

このルールブックを基準に、すべての「違反」を検知する。

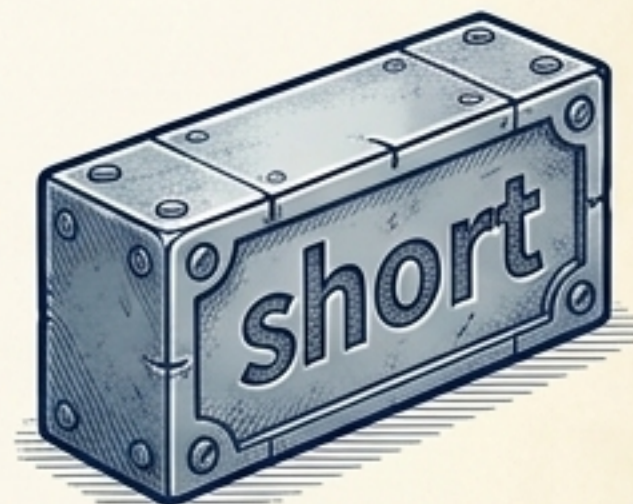
<非終端記号>

展開可能な構造の
ブロック



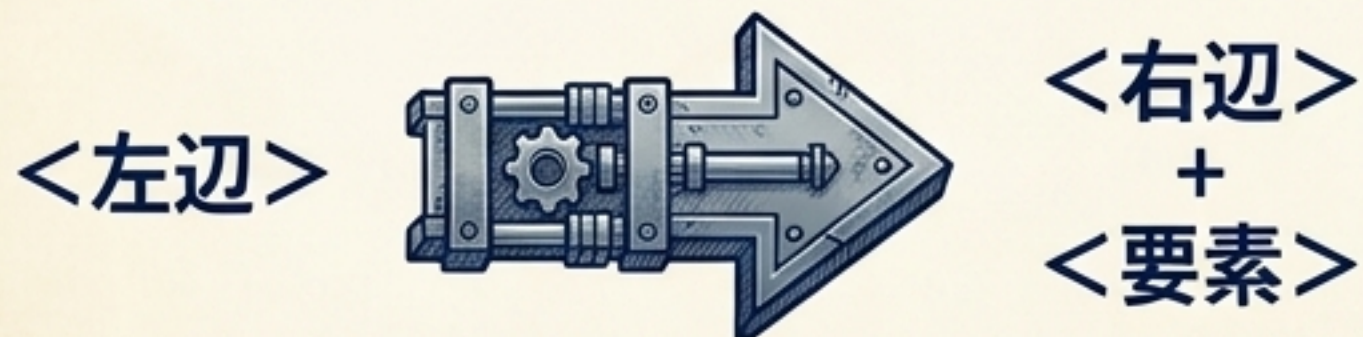
‘終端記号’

それ以上分解できない
最小単位



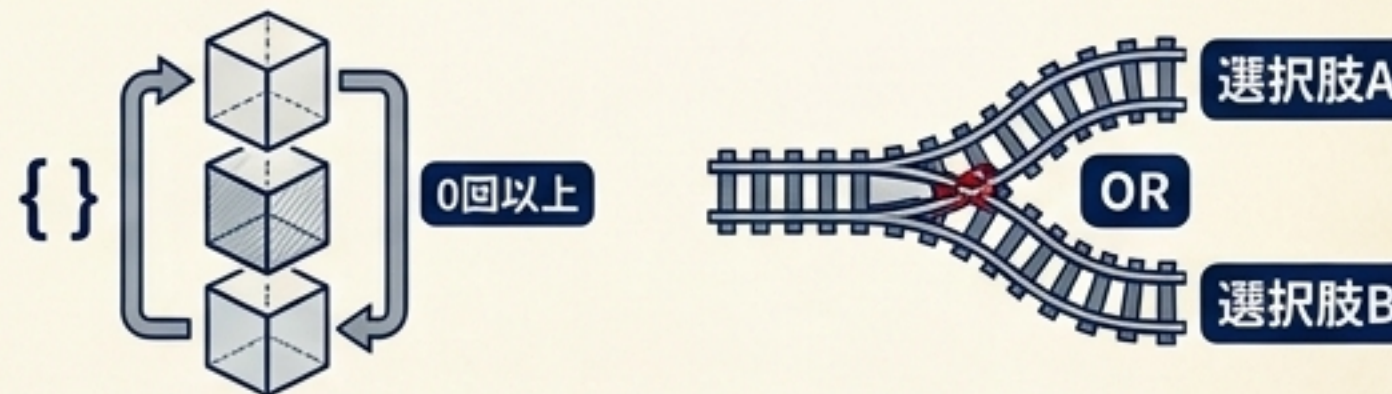
::=

定義する（左辺を右辺に置き換える）



{ } と |

{ } は0回以上の繰り返し。| は選択（または）



The Law

設問1 [ア] 攻略：繰り返される「構造」を見抜く

構文規則の { A } は「Aを0回以上繰り返す」という意味。

<プログラム> ::= <宣言部> <実行部>

<宣言部> ::= <宣言部記述> { [ア] }

The Evidence

【思考のヒント】

図のコードを見ると、宣言部には複数の宣言が並んでいる。この「1行分の宣言」という構造的単位を示す非終端記号はどれか？

short aa ;

long b1 ;

long c ;

} 宣言部

設問1 [イ] 攻略：パーツの完全なマッピング

【思考のヒント】

構文規則と実際のコードを1対1で対応させる。'aa' は英字で始まる文字列である。
字句解析のルールに従えば、この 'aa' が分類されるべき「非終端記号」は何か？

Top Layer
(BNF Rule)

<宣言部記述> ::= <宣言記述子> [イ] ';' ;

Middle Layer

Bottom Layer
(Code String)

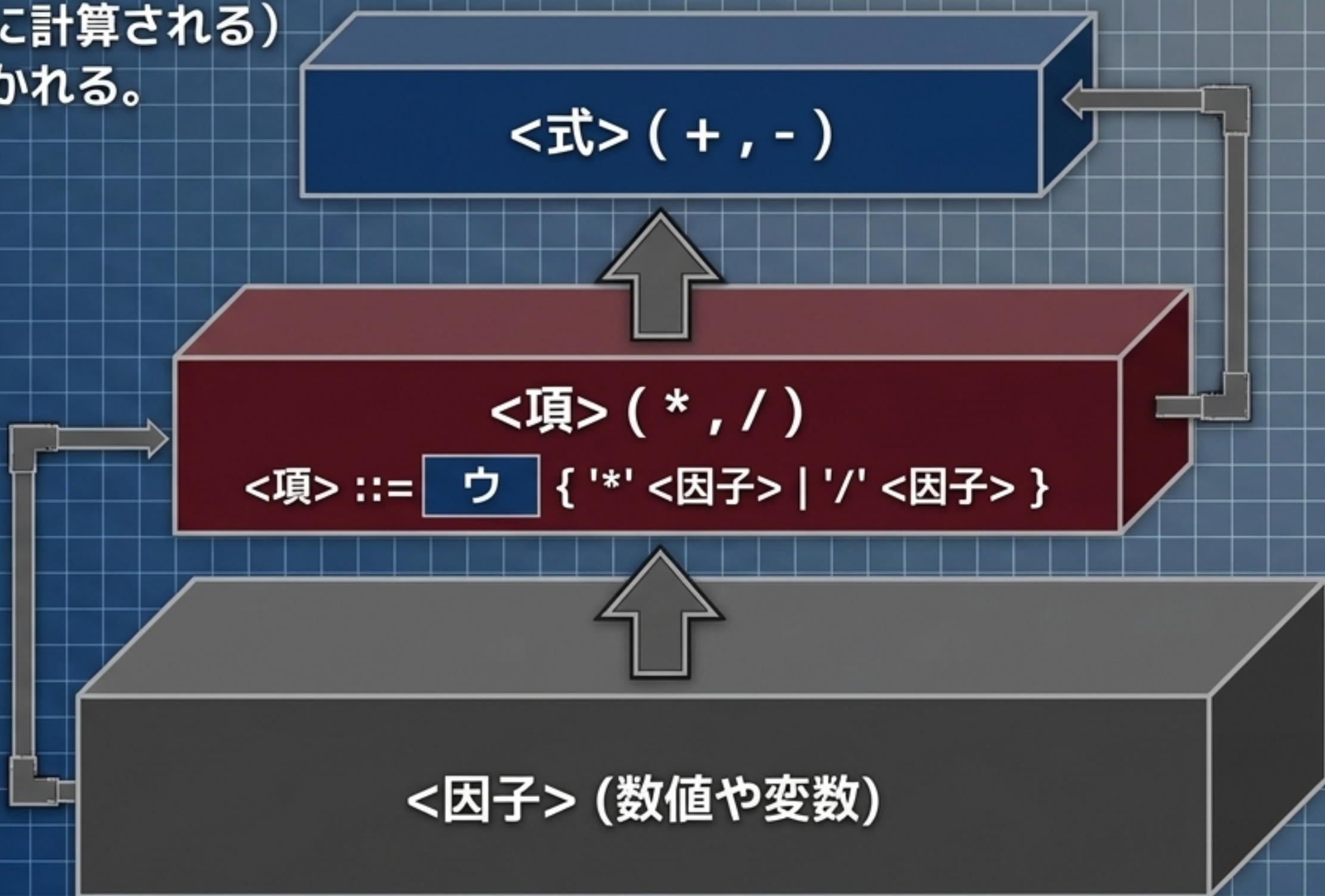
short aa ;

設問1 [ウ] 攻略：計算式の階層構造（優先順位）

BNFでは、優先順位が高い（先に計算される）ものほど、定義の深い位置に置かれる。

【思考のヒント】

<項> は掛け算・割り算の
繰り返りで構成される。
その計算のベースとなる一番
細かい「最初の要素」は何か？
下の階層を参照せよ。



設問2 攻略①：構文エラーと意味エラーの境界線

構文解析プログラムは「文法（形のルール）」だけを検査する。

「計算の意味が正しいか」は関知しない。

【思考のヒント】

行⑥ $mno = abc / 0 ;$ を評価する際、数学的なタブーに惑わされず、純粹にBNFの
＜項＞ ::= ＜因子＞ '/' ＜因子＞ に合致するかだけで判定せよ。

構文エラー (Syntax Error)

- 検査基準：BNF規則（形が合っているか）
- 例：変数の前にマイナス記号はつくか？
- 判定：BNFのルートから辿れなければエラー。

VS

意味エラー / 実行時エラー (Semantic/Runtime Error)

- 検査基準：数学的・論理的ルール
- 例：ゼロで割り算をしているか？
- 判定：BNF規則の管轄外。ここではエラーにならない。

設問2 攻略②：エラー検知のトレース手法

BNFのルートから末端まで厳密にたどる。

【思考のヒント】

① 行② short def ghi ; : 識別子を2つ
並べるルールはBNFに存在するか？

② 行⑤ ghi = - 2 * mno ; : トレースし
た結果、数値の前に単項演算子の「-」を
付けるルールは定義されているか？

<式> を開始

最初は必ず <項> から

最初は必ず <因子> から

<数> または <識別子> のみ許可

ghi = - 2 * mno ;

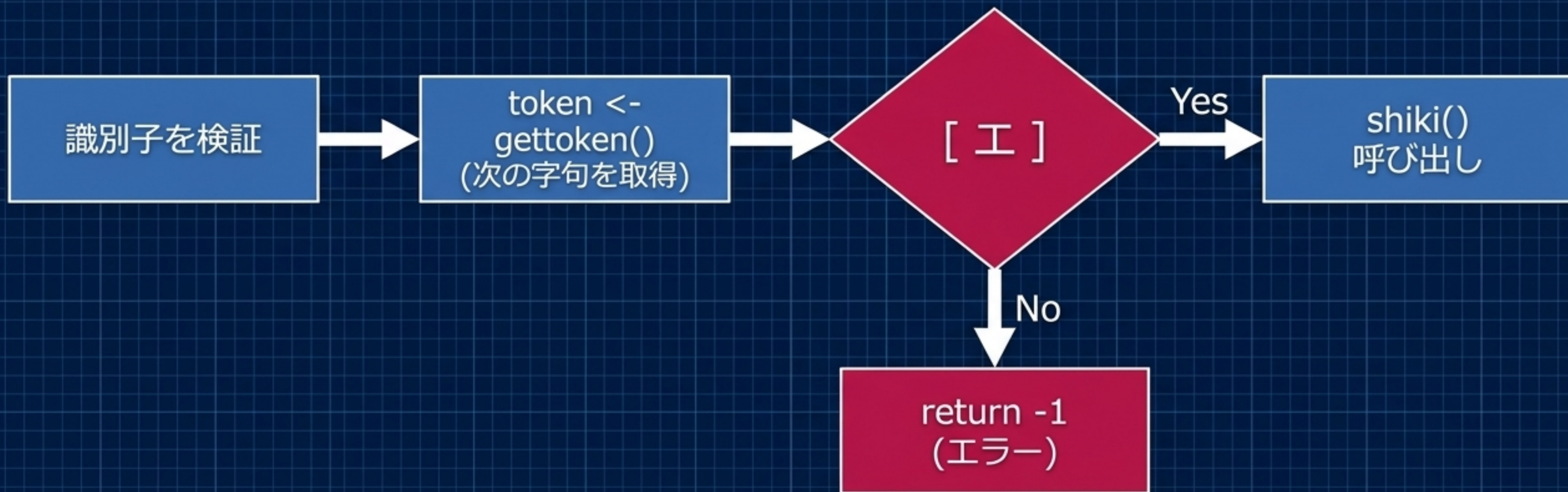
ERROR

設問3 [工] 攻略 : bun() 関数の内部ロジック

関数 bun() は $\langle \text{文} \rangle ::= \langle \text{識別子} \rangle '=' \langle \text{式} \rangle ';'$ を検証する。

【思考のヒント】

識別子を読み込んだ直後、BNFの定義によれば「必ず来なければならない特定の終端記号」がある。それは何か？ token変数と何を比較すべきか？



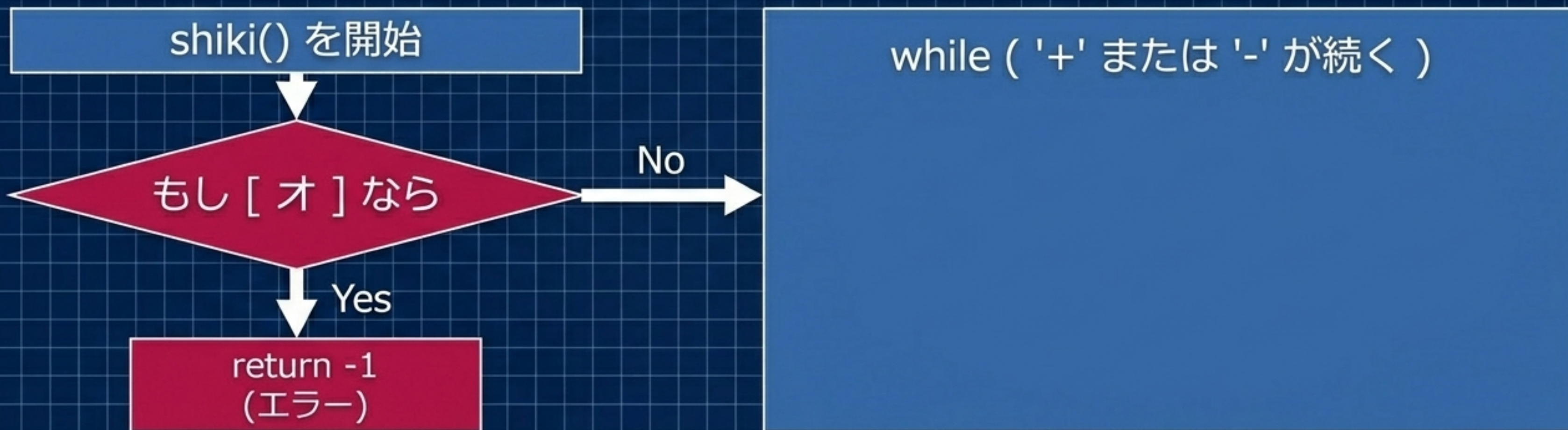
設問3 [オ] 攻略 : shiki() 関数のエラーハンドリング

関数 shiki() は $\langle \text{式} \rangle ::= \langle \text{項} \rangle \{ '+' \langle \text{項} \rangle \mid '-' \langle \text{項} \rangle \}$ を検証する。

式の開始には、まず必ず1つの $\langle \text{項} \rangle$ の解析が必要。

【思考のヒント】

プログラム冒頭のチェック。最初の $\langle \text{項} \rangle$ を解析する関数 kou() がエラー（戻り値 -1）を返した場合、式全体もエラーにすべきである。どう条件式を書くか？



設問4 (1) 攻略：カッコによる「再帰」の魔法

カッコの拡張の目的は、要素の中にさらに「複雑な計算」を入れ込むこと。

【思考のヒント】

カッコの中に「足し算や引き算を含む全体のまとまり」を入れるには、BNFの中で最も上位にある構造ルールを呼び出す必要がある。それは何か？

拡張前 / Old

$\langle \text{因子} \rangle ::= \langle \text{数} \rangle \mid \langle \text{識別子} \rangle$

$\langle \text{数} \rangle$

$\langle \text{識別子} \rangle$

拡張後 / New

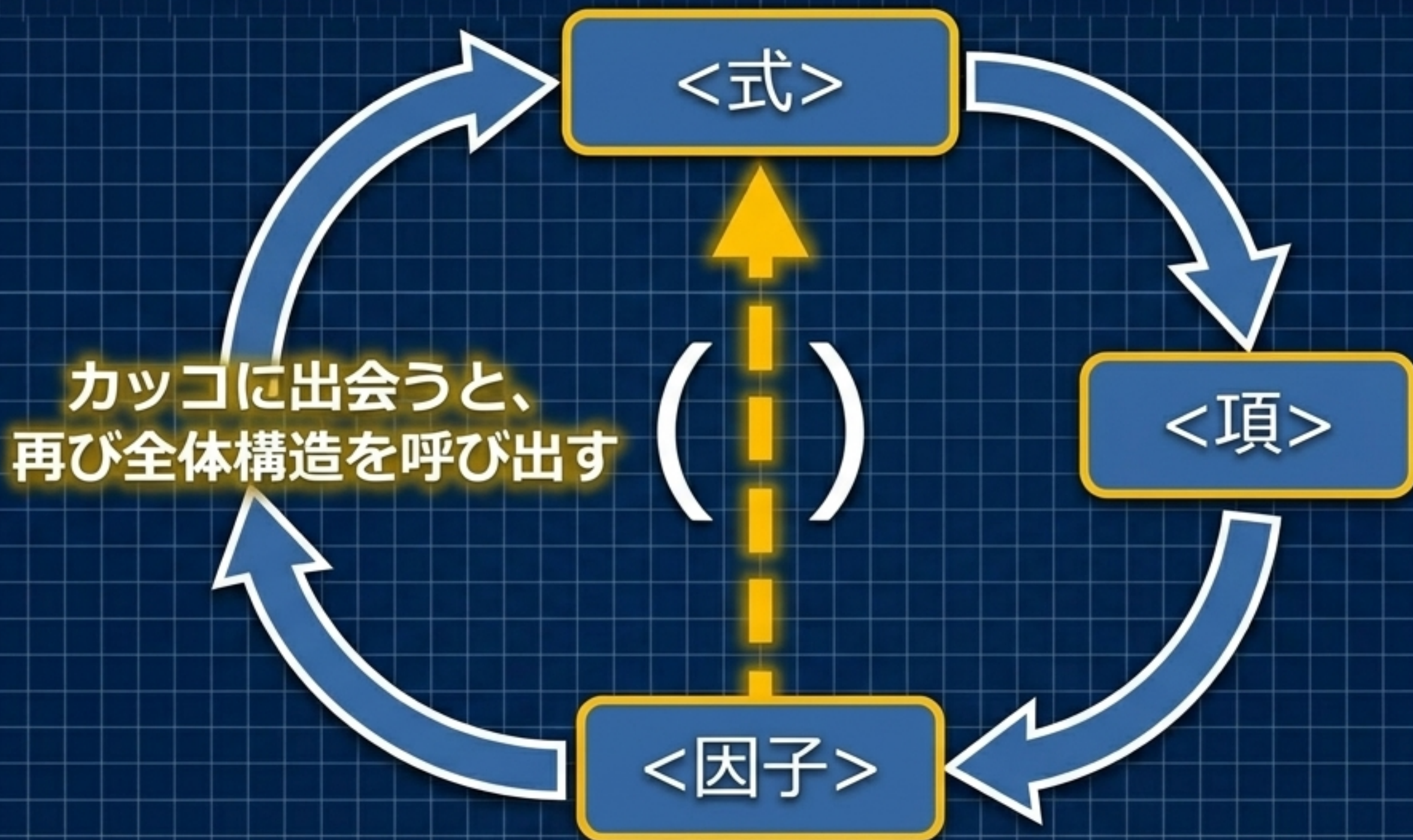
$\langle \text{因子} \rangle ::= \langle \text{数} \rangle \mid \langle \text{識別子} \rangle \mid '([\text{力}])'$

再帰

$a * (3 + b)$

設問4(1) 攻略：構文の無限ループ構造（再帰的定義）

この【力】の拡張により、深いカッコのネスト処理が可能となる。



設問4 (2) 攻略：字句解析 (Lexer) のアップグレード

構文規則に (と) という新しい終端記号が追加された。しかし、gettoken() はまだこれらを知らない。

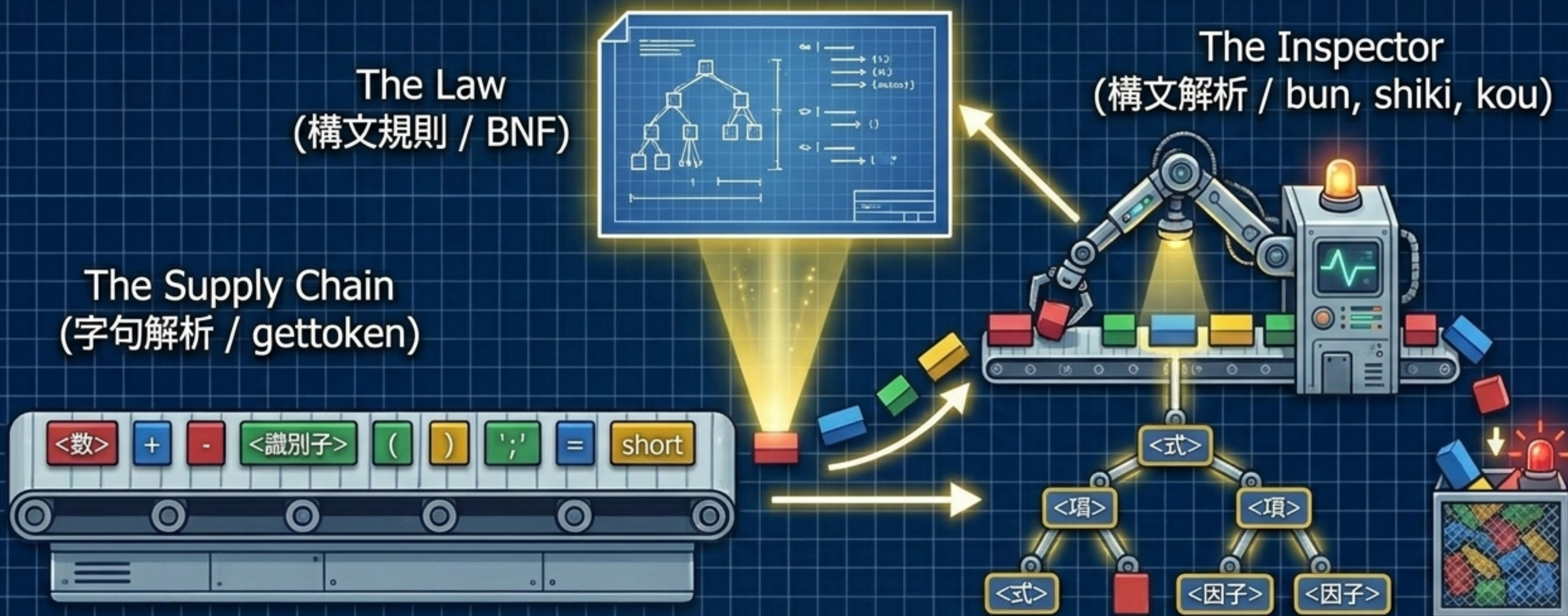
【思考のヒント】

既存の記号 (+ や = など) と同じように扱うには、gettoken() は (と) に遭遇した際、どのようなアクションを取り、何を戻り値とすべきか？ (35 字以内で言語化せよ)

字句の種類	戻り値
'short'	'S'
'long'	'L'
<数>	'N'
<識別子>	'I'
'=', '+', '-', '*', '/', ':'	左の各字句に同じ
いずれにも該当しない字句	'?'
プログラム記述の終わり	'\$'
Under Construction	'(' ')'

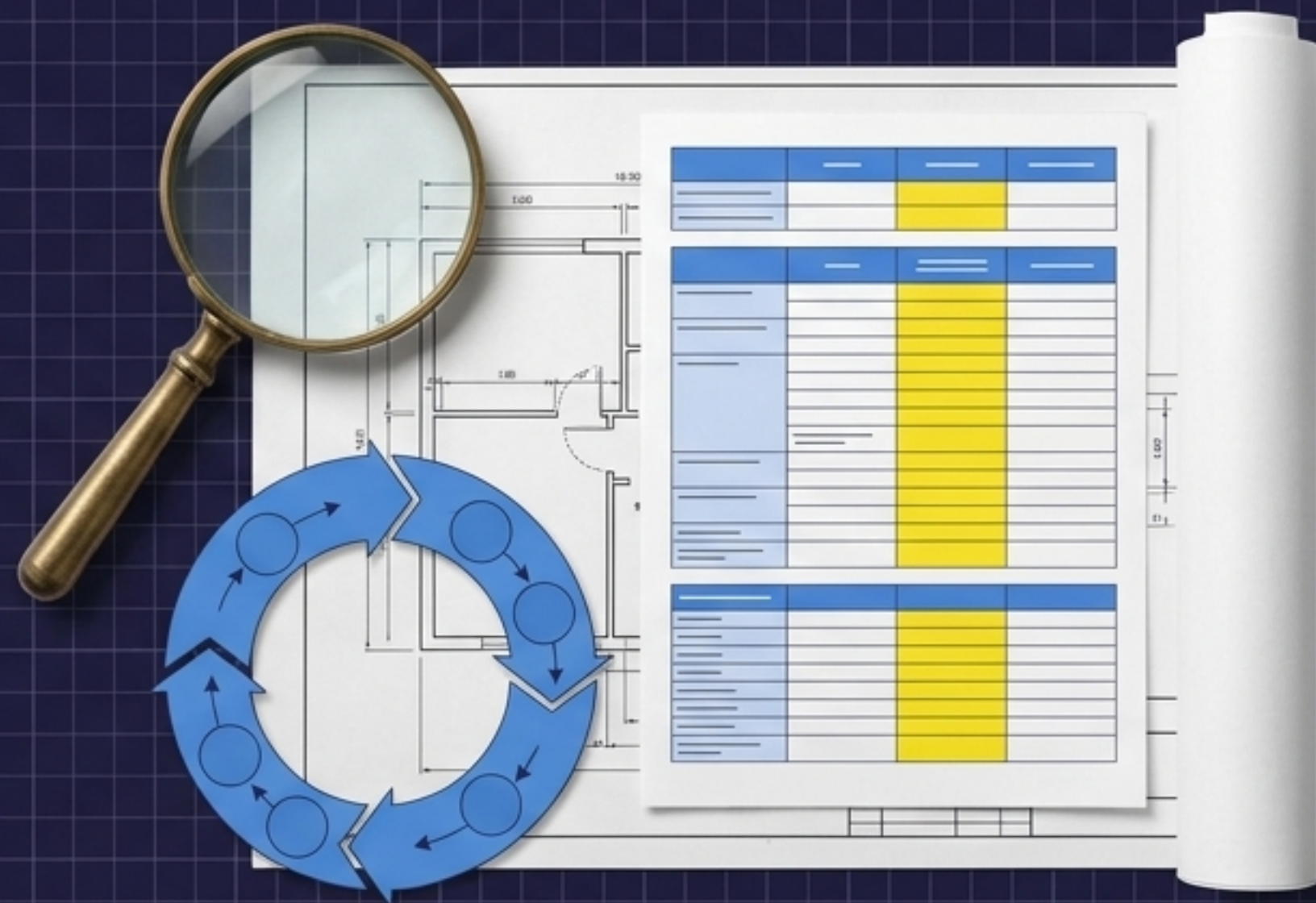
全体像の統合：構文解析エコシステム

BNFが「法」を定め、字句解析が「部品」を切り出し、関数群が法に従って部品を「組み立て・検証」する。この連携が言語プロセッサの核である。



思考回路は**完成**した。 あとは**適用**するだけ。

- ・ パターンの抽出（設問1）
- ・ 構文と意味の分離（設問2）
- ・ アルゴリズムへの翻訳（設問3）
- ・ 再帰構造の設計（設問4）



すべてのフレームワークは揃った。図面とツールを持ち帰り、自らの手で答えを導き出そう。