Na meta 2 foi pedido para realizar a análise sintática, segundo passo na realização do compilador para a linguagem Juc. Esta análise foi feita com base numa gramática ambígua inicial em notação EBNF e, a partir desta, rescrita para yacc de modo a ser feita a ligação com a análise lexical realizada na meta 1.

A rescrição da gramática foi feita através dos princípios básicos do yacc, contudo, existiam algumas restrições mediante as regras de associação dos operadores e precedências, entre outros aspetos.

Um dos aspetos que teve de ser contornado na rescrição da gramática foi o facto de existirem símbolos terminais ou não terminais que podem aparecer zero ou mais vezes ({...} na notação EBNF). Isto foi contornado no yacc através da recursividade à esquerda, criando um símbolo terminal auxiliar de modo a ser possível estes símbolos poderem aparecer numa árvore de derivação uma ou mais vezes. Por exemplo, Program -> CLASS ID LBRACE {MethodDecl|FieldDecl|SEMICOLON} RBRACE foi contornado da seguinte forma:

Program: CLASS ID LBRACE ProgramScript RBRACE  
 ;  
ProgramScript: /\*empty\*/  
 | MethodDecl ProgramScript  
 | FieldDecl ProgramScript  
 | SEMICOLON ProgramScript  
 ;

Outros dois aspetos, semelhante ao anterior, foi a existência de símbolos terminais e não terminais opcionais ([…] na notação EBNF) e símbolos que só pode ser escolhido um deles ((…) na notação EBNF). Estes aspetos foram contornados criando outras regras na gramática para um certo símbolo terminal ou criando símbolos terminais auxiliares, dependendo da complexidade da situação. Por exemplo, MethodHeader -> (Type|VOID) ID LPAR [FormalParams] RPAR foi rescrito da seguinte maneira:

MethodHeader: Type ID LPAR MethodHeader2 RPAR  
 | VOID ID LPAR MethodHeader2 RPAR  
 ;  
MethodHeader2: /\*empty\*/  
 | FormalParams  
 ;

Também teve de se ter em conta as precedências, ou seja, prevenir que se crie conflitos entre certos símbolos não terminais. Isto foi contornado no yacc identificando a prioridade de cada símbolo não terminal:

%right ASSIGN  
%left OR  
%left AND  
%left XOR  
%left EQ NE  
%left GE GT LE LT  
%left LSHIFT RSHIFT  
%left PLUS MINUS  
%left STAR DIV MOD  
%right NOT  
%left LPAR RPAR LSQ RSQ  
%right ELSE

Assim não existirão conflitos respetivamente às operações, por exemplo, a multiplicação terá sempre prioridade em relação à adição.

Contudo foi ainda necessário mudar o nível de precedência em regras da gramática específicas para não existirem outro tipo de conflitos. A dado exemplo temos a regra Expr -> (MINUS|NOT|PLUS) Expr que foi contornada usando %prec nas regras do MINUS e do PLUS.