Para demonstrar que a gramática G' que foi enviada no exercício anterior não é recursiva à esquerda, será desenvolvido uma tabela de derivação da gramática começando por PROGRAM, passando por cada não terminal e provando que sua derivação não produz uma recursão à esquerda.

PROGRAM -> STATEMENT	{, break, ;, int, float, string, print, return, for, ident, if, read
PROGRAM -> FUNCLIST	def
PROGRAM -> &	&
FUNCLIST -> FUNCDEF FUNCLISTAUX	def
FUNCLISTAUX -> FUNCLIST	def
FUNCLISTAUX -> &	&
FUNCDEF -> def ident (PARAMLIST) { STATELIST }	def
PARAMLIST -> DATATYPE ident PARAMLISTAUX	int, float, string
PARAMLIST -> &	&
DATATYPE -> int	int
DATATYPE -> float	float
DATATYPE -> string	string
PARAMLISTAUX -> , PARAMLIST	,
PARAMLISTAUX -> &	&
STATEMENT -> VARDECL	int, float, string
STATEMENT -> ATRIBSTAT	ident
STATEMENT -> PRINTSTAT	print
STATEMENT -> READSTAT	read
STATEMENT -> RETURNSTAT	return
STATEMENT -> IFSTAT	if

Continua…

STATEMENT -> FORSTAT	for
STATEMENT -> {STATELIST}	{
STATEMENT -> break	break
STATEMENT -> ;	;
VARDECL -> DATATYPE ident OPTVECTOR	int, float, string
OPTVECTOR -> [int_constant] OPTVECTOR	int_constant
OPTVECTOR -> &	&
ATRIBSTAT -> LVALUE = ATRIBSTATRIGHT	ident
ATRIBSTATRIGHT -> FUNCCALL	ident
ATRIBSTATRIGHT -> ALLOCEXPRESSION	new
FUNCCALL -> ident(PARAMLISTCALL)	ident
PARAMLISTCALL -> ident PARAMLISTCALLAUX	ident
PARAMLISTCALL -> &	&
PARAMLISTCALLAUX -> , PARAMLISTCALL	,
PARAMLISTCALLAUX -> &	&
PRINTSTAT -> print EXPRESSION	print
READSTAT -> read LVALUE	read
RETURNSTAT -> return	return
IFSTAT -> if(EXPRESSION) STATEMENT ELSESTAT	if
ELSESTAT -> else STATEMENT	else

Continua…

ELSESTAT -> &	&
FORSTAT -> for(ATRIBSTAT; EXPRESSION; ATRIBSTAT) STATEMENT	for
STATELIST -> STATEMENT OPT_STATELIST	{, break, ;, int, float, string, print, return, for, ident, if, read
OPT_STATELIST -> STATELIST	{, break, ;, int, float, string, print, return, for, ident, if, read
OPT_STATELIST -> &	&
ALLOCEXPRESSION -> new DATATYPE[NUMEXPRESSION] OPT_ALLOC_NUMEXP	new
OPT_ALLOC_NUMEXP -> [NUMEXPRESSION] OPT_ALLOC_NUMEXP	[
OPT_ALLOC_NUMEXP -> &	&
EXPRESSION -> NUMEXPRESSION OPT_REL_OP_NUM_EXPR	+, -
OPT_REL_OP_NUM_EXPR -> REL_OP NUMEXPRESSION	<, >, <=, >=, ==, /=
OPT_REL_OP_NUM_EXPR -> &	&
REL_OP -> <	<
REL_OP -> >	>
REL_OP -> <=	<=
REL_OP -> >=	>=
REL_OP -> ==	==
REL_OP -> /=	/=
NUMEXPRESSION -> TERM REC_PLUS_MINUS_TERM	+, -, int_constant
REC_PLUS_MINUS_TERM -> PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM	+, -
REC_PLUS_MINUS_TERM -> &	&

Continua…

PLUS_OR_MINUS -> +	+
PLUS_OR_MINUS -> -	-
TERM -> UNARYEXPR REC_UNARYEXPR	+, -, int_constant
REC_UNARYEXPR -> UNARYEXPR_OP TERM	*, /, %
REC_UNARYEXPR -> &	&
UNARYEXPR_OP -> *	*
UNARYEXPR_OP -> /	/
UNARYEXPR_OP -> %	%
UNARYEXPR -> PLUS_OR_MINUS FACTOR	+, -
UNARYEXPR -> FACTOR	int_constant
FACTOR -> int_constant	int_constant
FACTOR -> float_constant	float_constant
FACTOR -> string_constant	string_constant
FACTOR -> null	null
FACTOR -> LVALUE	ident
FACTOR -> NUMEXPRESSION	+, -
LVALUE -> ident OPT_ALLOC_NUMEXP	ident