RT-MAC-8905

# Grafos sintáticos simples e um grafo para a linguagem C ANSI

Valdemar W.Setzer
Roberto C.Mayer

*Agosto de 1989*

*Sumário*

Este artigo contém uma definição formal de *grafo sintático*. A seguir, é definida uma classe especial de grafos sintáticos, os *grafos sintáticos simples*, adequados a uma análise sintática tipo LL(1) bem simples, com recuperação automática de erros. Como exemplo, é apresentado o grafo sintático simples para a *linguagem C ANSI*, que pode ser empregado como especificação sintática simples para a mesma.

*Abstract*

This paper gives a formal definition of *syntax graphs* and a definition of a special class of syntax graphs, namely *simple syntax graphs*. These graphs may be used for a very simple LL(1)-type parsing that allows for automatic error recovery. As an example, the simple syntax graph for the *ANSI C language* is given, that serves as a simple syntactic definition for this language.

## *1. Introdução*

Em alguns dos cursos de compilação dados pelo primeiro autor, em que cada aluno implementa um compilador usando o algoritmo de análise sintática dado em [SM86], é exigido um compilador para a linguagem C escrito em C. Como o algoritmo de análise sintática empregado em [SM86] interpreta um grafo sintático, foi necessário desenvolver um desses grafos para a C. Depois de várias versões, cremos que finalmente chegamos a um grafo definitivo, englobando toda a C ANSI [HS87, KR88].

Para completar o trabalho, decidimos fazer uma definição formal de *grafo sintático*, já que não encontramos nenhuma na literatura. Introduzimos ainda o conceito de *grafo sintático simples*, que é aquele àceito pelo analisador sintático descrito em [SM86], dando aqui uma definição mais compacta. Apresentamos no apêndice um grafo sintático simples para a *linguagem C*, servindo como exemplo da simplicidade dos grafos deste tipo e como descrição sintática de C, bem mais simples que outras descrições. Discutimos as dificuldades encontradas e as soluções adotadas, bem como uma comparação com outras descrições de C. A linguagem C adotada foi a padrão ANSI de acordo com as gramáticas de [HS87] e [KR88].

## *2. Grafo sintático*

Nesta seção introduzimos a definição de grafo sintático, sem nenhum tipo de restrição, e as convenções que usamos para representar estes grafos.

Def$_1$: Um *grafo sintático G* é uma quádrupla ordenada $(S, V_N, V_T, C)$, onde $V_N$ e $V_T$ são conjuntos finitos, não vazios, disjuntos, de *símbolos não terminais* e *símbolos terminais*, respectivamente; $S \in V_N$ é o *símbolo não-terminal inicial*, e C é um conjunto não vazio de pares ordenados $(N, G_N)$, onde $N \in V_N$ e $G_N$ é o *grafo do não-terminal N* (cf. Def$_2$), tal que para cada $N \in V_N$ existe um único par $(N, G_N)$.

No apêndice apresentamos um grafo sintático para a linguagem C. Os símbolos não-terminais são representados por maiúsculas itálicas, enquanto os terminais são seqüências de minúsculas ou de símbolos especiais em negrito.

Def2: O *grafo $G_N$ do não-terminal N* é um grafo conexo orientado definido por uma quádrupla ordenada $(k_o, K, E_A, Es)$, onde K, $E_A$ e Es são conjuntos finitos disjuntos, denominados respectivamente de *conjunto dos nós*, *conjunto das arestas alternativas* e *conjunto das arestas sucessoras*, definidos abaixo. K é não vazio, e $k_o \in K$ é denominado *nó inicial* do grafo do não-terminal.

Por exemplo, pode-se encontrar no apêndice os grafos de *PROGRAM_MODULE*, *GLOBAL_DECLARATOR*, *PARAMETER_DECL*, etc.

Def2.1: Seja K o conjunto dos nós do grafo de um não-terminal. Associamos a K a função Con, denominada *conteúdo* do nó, definida por

$$Con: K \rightarrow V_N \cup V_T \cup \{\theta\} \quad ,$$

onde θ representa a cadeia vazia (i.é., $x\theta = \theta x = x$ para todo símbolo x).

Seja $k \in K$. Se $Con(k) = N \in V_N$, diremos que k é um *nó não-terminal*. Se $Con(k) = t \in V_T$, diremos que k é um *nó terminal*. Se $Con(k) = \theta$, diremos que k é um θ-nó (representado graficamente por θ).

Por exemplo, no grafo de *GLOBAL_DECLARATOR*, **struct** e **union** são nós terminais e *FIELD_LIST* é um nó não-terminal.

Def2.2: O conjunto $E_A$ das arestas alternativas do grafo de um não-terminal com conjunto de nós K é definido por

$$E_A = \{ (k, Alt(k)) \mid k \in K \}$$

onde Alt(k) é uma função definida como:

$$Alt: K \rightarrow K \cup \{\delta\}$$

Seja $k \in K$. Se $Alt(k) \in K$, diremos que Alt(k) é *o nó alternativo* de k, e (k, Alt(k)) é *a aresta alternativa* de k. Se $Alt(k) = \delta$, diremos que o nó k *não possui alternativa*. Graficamente, representamos

a aresta alternativa de k como uma seta saindo sempre *verticalmente para baixo imediatamente à esquerda* de k. Se k não possui alternativa, a aresta correspondente não é desenhada.

Por exemplo, no grafo de *GLOBAL_DECLARATOR*, o nó contendo union é o nó alternativa do nó que contém struct; o arco que sai para baixo à esquerda do nó (de cima para baixo) contendo '*' e vai para identifier é a aresta alternativa do primeiro; o nó que contém *DIMENSIONS* não possui alternativa.

Denotaremos a composição Alt(Alt(k)) como $Alt^2(k)$; generalizando

$$Alt^j(k) = Alt(Alt^{j-1}(k))$$

Se $Alt^{n+1}(k) = \delta$, definimos as seguintes seqüências:

$$Alt^*(k) = \{ k, Alt(k), Alt^2(k), \ldots , Alt^n(k) \}$$

(i.é., todos os nós atingíveis a partir de k usando-se apenas arestas alternativas)

$$Alt^-(k) = Alt^*(k) - \{ Alt^n(k) \}$$

(i.é., $Alt^*(k)$ sem o último nó, que não possui alternativa)

$$Alt^+(k) = Alt^*(k) - \{ k \}$$

(i.é., $Alt^*(k)$ sem o primeiro nó da seqüência)

Por exemplo, em *GLOBAL_DECLARATOR* se k é o nó contendo enum, $Alt^*(k)$ é a seqüência de nós contendo

enum, *, identifier, (

Def2.3: O conjunto Es das arestas sucessoras do grafo de um não-terminal com conjunto de nós K é definido por

$$Es = \{ (k, Suc(k)) \mid k \in K \}$$

onde Suc(k) é uma função definida como:

$$Suc: K \rightarrow K \cup \{\delta\}$$

Seja $k \in K$. Se $Suc(k) \in K$, diremos que Suc(k) é *o nó sucessor* de k, e (k, Suc(k)) é *a aresta sucessora* de k. Se $Suc(k) = \delta$, diremos que o nó k *não possui sucessor*, e é um *nó final* do grafo do não-termi-

nal. Graficamente, representamos a aresta sucessora de k como uma seta saindo sempre de k *horizontalmente à direita*. Se k é um nó final, a representação será

Por exemplo, em *GLOBAL_DECLARATOR* o nó contendo *BLOCK* é sucessor do nó contendo { ; os nós contendo ';' são nós finais.

**Def3**: O *grafo sintático* $G_N = (k_0, K, E_A, E_S)$ *do não-terminal N* é um grafo do não-terminal N, tal que as funções Con, Alt e Suc são totais em K e todo nó de K, exceto eventualmente $k_0$, é nó alternativo ou nó sucessor de algum outro nó de K, i.é., se $k \in K$, k diferente de $k_0$, existe $k' \in K$ tal que

$$k = Alt(k') \text{ ou } k = Suc(k') \quad \text{(ou ambos)}$$

Notação: Seja X a seqüência

$$X = ( x_1, x_2, \ldots , x_n )$$

Denotaremos o elemento $x_i$ de X como $X.x_i$ (inspirado na seleção de componente de **record** em Pascal ou de **struct** em C - o original provavelmente foi o **structure** de PL/I).

Nas definições 4 a 6, abaixo, consideraremos $k_1$ como sendo um nó qualquer de um grafo sintático G.

As definições seguintes aplicam-se exclusivamente à classe de grafos sintáticos que definiremos mais adiante.

**Def4**: Um *percurso de nós a partir de* $k_1$ é uma seqüência $k_1, k_2, \ldots , k_n$ ($n \geq 1$) de nós de G, tal que, se $1 \leq i < n$,

se $Con(k_i) \in V_T - \{\theta\}$, então $k_{i+1} = Alt(k_i)$ ou $k_{i+1} = Suc(k_i)$

se $Con(k_i) = \theta$, então $k_{i+1} = Suc(k_i)$

se $Con(k_i) = N \in V_N$, então $k_{i+1}, \ldots, k_n$ é formada pela concatenação do percurso de nós $k_{i+1}, \ldots k_j$ com $k_{i+1} = G_N.k_0$ e tal que $k_j$ é um nó final (de $G_N.K$), e onde nenhum $k_r$, $i+1 \leq r < j$, é um nó final e $k_{j+1} = Suc(k_i)$.

Por exemplo, no grafo de *GLOBAL_DECLARATOR*, a seqüência dos nós que contém

identifier, '(', '[', '=', *INITIALIZER*, '{', *INITIALIZER*, '{'

é um percurso de nós (note-se que ele passa pelo grafo de *INITIALIZER*).

Def5: Um *percurso de visitas de nós a partir de* $k_1$ é uma seqüência de nós **terminais** obtida de um percurso $k_1, k_2, \ldots, k_n$ $(n \geq 1)$ eliminando todos os nós $k_j$ tais que

$$k_{j+1} = Alt(k_j) \text{ ou } Con(k_j) \in V_N$$

(Notamos que $k_1$ pode não pertencer ao percurso de visitas.)

Def6: Dado um percurso de visitas a partir de $k_1$, denominamos de uma *cadeia gerada a partir de* $k_1$ à concatenação sucessiva dos conteúdos dos nós desse percurso de visitas, na ordem em que eles são percorridos.

Na seqüência do exemplo anterior, a cadeia gerada correspondente é

identifier = { {

Def7: Uma *cadeia gerada por um não-terminal* N é uma cadeia gerada a partir do nó inicial do grafo sintático de N, tal que o último nó do percurso de visitas correspondente é um nó final de $G_N$.

Def8: Uma *cadeia gerada por um grafo sintático* é uma cadeia gerada pelo símbolo inicial do grafo.

Def9: Uma *linguagem gerada por um grafo sintático* é o conjunto de todas as cadeias geradas por esse grafo.

Def10: Seja $N \in V_N$. Então,

$First(N) = \{ t \in V_T \mid N \text{ gera uma cadeia iniciada à esquerda por } t \}$

Por exemplo,

First(*DECLARATOR*) = { *, identifier, ( }

Def11: Seja $N \in V_N$, S o símbolo inicial do grafo. Então,

Follow(N) = { t ∈ $V_T$ | S gera uma cadeia através de um percurso de visitas $k_1$, ..., $k_n$ com $k_i$ nó final de N e $k_{i+1}$ = t }

### *3. Grafo sintático simples*

Introduziremos agora formalmente as restrições necessárias para que um grafo sintático possa ser dito *simples*. Notamos que, em comparação com os grafos derivados a partir de gramáticas ESLL(1) definidas em [SM86], o grafo sintático simples não possui lambda-alternativas e permite construções que não correspondem àquelas gramáticas, sendo portanto mais geral.

Def12: Um grafo sintático G = (S, $V_N$, $V_T$, C) dir-se-á *simples* ou de *classe SLL(1)*, sse dados quaisquer N ∈ $V_N$, k ∈ $G_N$.K e o inteiro n tal que $Alt^{n+1}(k) = \delta$, forem satisfeitas as seguintes condições:

1) $\{k\} \cap Alt^*(k) = \emptyset$

2) Se n > 0, para todo i tal que 0 ≤ i < n, $Con(Alt^i(k)) \in V_T - \{\theta\}$

3) Dados quaisquer i,j distintos, 0 ≤ i,j ≤ n.

$$Con(Alt^i(k)) \text{ e } Con(Alt^j(k))$$

também são distintos.

Para as condições 4) a 6), seja $k_n = Alt^n(k)$.

4) Se $Con(k_n) = M \in V_N$, então $Alt^-(k) \cap First(M) = \emptyset$

5) Se $Con(k_n) = \theta$ (ou $Con(k_n) = M \in V_N$ e M gera θ) e $Suc(k_n) = \delta$, então $Alt^-(k) \cap Follow(N) = \emptyset$

6) Se $Con(k_n) = \theta$ (ou $Con(k_n) = M \in V_N$ e M gera θ) e $Suc(k_n) \in G_N$, então a seqüência

$$Alt^-(k) \cup Alt^*(Suc(k_n))$$

deve satisfazer as condições (3) a (5).

Daqui em diante denotaremos grafo sintático simples por *SSG*, abreviatura de *Simple Syntax Graph*.

## *4. Análise sintática de grafos sintáticos simples*

A grande motivação para a construção de um SSG é que ele faz o papel de uma gramática, permitindo a execução de um algoritmo de análise sintática top-down, como o apresentado por Wirth [WI76] (essencialmente baseado na Def$_4$: caminha-se por um percurso de alternativas até que o look-ahead de 1 símbolo coincida com o conteúdo de um nó k; nesse caso o percurso continua com o sucessor de k) na versão recursiva e por Setzer [SE78, SM86] na versão não-recursiva. Esses algoritmos analisam corretamente qualquer SSG, i.é., qualquer grafo sintático que satisfaz as restrições do item anterior. Os algoritmos citados empregam objetivos (os não-terminais); as restrições dadas no item 3 garantem que um look-ahead de um símbolo é suficiente para que o analisador escolha sua próxima ação. Portanto, se fossem associadas gramáticas aos grafos, elas formariam uma subclasse das gramáticas LL(1).

Note-se que essas restrições permitem que o algoritmo seja muito mais simples do que os adequados às gramáticas LL(1), pois estes exigem conjuntos diretores (ver p.ex. [ASU86]) para, por exemplo, decidir o caminho a ser tomado quando um não-terminal tem alternativa. Na verdade, os conjuntos diretores estão implementados através das seqüências de alternativas de terminais.

Além disso, as restrições impostas permitem a emissão automática de mensagens de erro sem referência a símbolos não-terminais, ao contrário de versões consultadas de compiladores usando o analisador YACC [JO75] ou a proposta de Wirth para Pascal [JW73], em geral implementadas nos compiladores comerciais. Ainda, as restrições do SSG permitem uma boa recuperação automática de erros sintáticos [SM86].

## *5. Comentários sobre o grafo sintático simples para a linguagem C*

No Apêndice mostramos o SSG para a linguagem C, cujo símbolo inicial é *PROGRAM_MODULE*. O grafo inclui todas as extensões da linguagem introduzidas pelo comitê ANSI de padronização cf. [HS87] e [KR88], mas não estão incluídos os modificadores de ponteiros e funções exclusivos dos compiladores para IBM PC (como far, near, huge, pascal e cdecl) e de fácil introdução no nosso grafo. Também não fazem parte do grafo os comandos do preprocessador, já que o preprocessador não é considerado parte integrante da linguagem C do ponto de vista sintático [HS87] e [KR88]. Incluimos eof como símbolo terminal, denotando o fim da cadeia gerada ou reconhecida, embora não faça parte da linguagem propriamente dita.

A maior dificuldade na construção do grafo foi permitir a ocorrência dos identificadores não declarados: rótulos e funções não declaradas ou definidas previamente (para as quais o compilador deve assumir que devolvem valores inteiros). Para poder obter o grafo de *STMT* com identifier podendo ser seguido de ':' (rótulo) ou operadores posfixos, foi necessário decompor *EXPR* como aparece.

Ainda em relação às expressões, note que o grafo gera atribuições com *UNARY_EXPR* do lado esquerdo. Como existem expressões unárias que não podem receber valores (não são "lvalues" [KR88]), esta verificação deve ser feita semanticamente.

Outra verificação semântica diz respeito a *CONST_EXPR*. Embora esteja definida simplesmente como uma expressão condicional, uma expressão constante deve ter o seu valor avaliado durante a compilação. Caso isto não seja possível, deve ocorrer um erro semântico.

Finalmente, o último ponto do grafo onde a linguagem gerada pelo grafo é um superconjunto da C ANSI aparece no grafo de *GLOBAL_DECLARATOR*. Como o nome dos parâmetros das funções é opcional no caso dos protótipos de funções, *PARM_DECL_LIST* gera as declarações usando *ABSTRACT_DECLARATOR*, de forma a tornar o identificador opcional. Como conseqüência, o grafo de *GLOBAL_DECLARATOR* também gera definições de

funções sem os nomes dos parâmetros. Isto deve ser detectado como um erro semântico também.

Note que em vários pontos do grafo foi necessário "expandir" determinados não-terminais. Por exemplo, *LOCAL_DECLARATOR* e *GLOBAL_DECLARATOR* contêm *NON_EMPTY_TYPE* "expandido". Note que este método não permite resolver sintaticamente os casos citados acima, para os quais indicamos a necessidade de se efetuar uma verificação semântica.

## *6. Planaridade de grafos sintáticos simples*

Obviamente, um grafo planar é mais simples visualmente do que um grafo não-planar. No entanto, os SSG podem não ser planares, como no exemplo da Fig. 1. Esse grafo foi construído com as restrições da Def=, onde arestas alternativas são sempre representadas por setas saindo à esquerda e para baixo do nó e as arestas sucessoras são representadas por setas saindo à direita do nó.

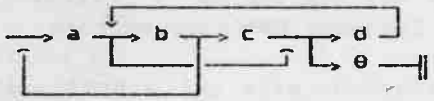



Fig. 1 - Grafo sintático simples não-planar

Se levantarmos essa restrição, o que pode ser feito colorindo diferentemente arestas sucessoras e alternativas, esse grafo torna-se planar, como na Fig. 2, onde as arestas alternativas são tracejadas e as sucessoras contínuas. Note-se que o funcionamento do algoritmo de análise sintática é também óbvio, pois o importante é saber qual a alternativa e sucessor de cada nó, que continuam sendo únicos.

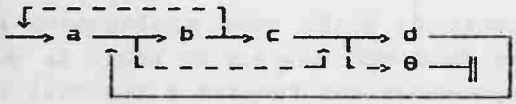



Fig. 2 - Grafo sintático simples planar equivalente ao da Fig. 1

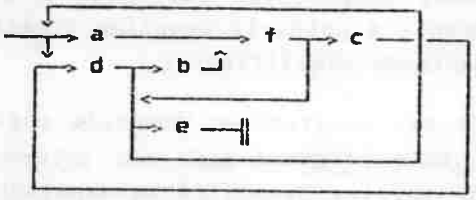



Fig. 3 - Grafo sintático simples não-planar

Existem grafos que mesmo sem a restrição da direção do desenho das arestas alternativas e sucessoras são não-planares, como o da Fig. 3, inspirado no $K_{3,3}$ da teoria dos grafos [BE73]. No entanto, ele também tem um equivalente planar no sentido de gerar a mesma linguagem, mostrado na Fig. 4, obtida duplicando-se alguns nós.

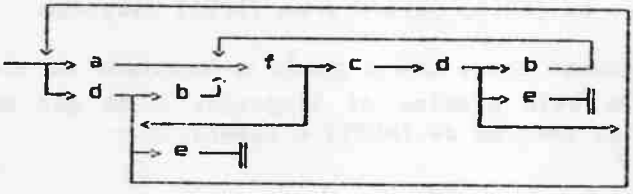



Fig. 4 - Grafo sintático simples planar equivalente ao da Fig. 3

Levantamos a conjetura de que dado um SSG não-planar, existe sempre um SSG planar equivalente.

## *7. Conclusões*

Se entendermos os SSGs como a representação gráfica de gramáticas, eles formam uma subclasse das gramáticas LL(1). Assim, o fato de existir um SSG para a linguagem C no padrão ANSI prova que existe uma gramática LL(1) para a mesma, apesar de ter sido definida tradicionalmente usando uma gramática LALR(1). O SSG do Apêndice foi testado em vários compiladores C.

Segundo a nossa experiência, é mais prático trabalhar-se com SSGs do que com as gramáticas ESLL(1); estas são extremamente compactas, pela possibilidade de representar-se expressões regulares nos lados direitos das produções, como se pode ver pela gramática de Pascal

de [SM86]. No entanto essa compactação prejudica a legibilidade. Por outro lado, o SSG aumenta a potência daquelas gramáticas e fornece um aspecto visual bastante simplificador.

Além disso, o SSG poderia ser encarado em essência como o programa do analisador sintático. Este programa pode ser interpretado, como é o caso do algoritmo de análise sintática apresentado em [SM86], ou compilado para obter uma análise sintática mais eficiente. Ainda, este algoritmo permite a recuperação automática de erros.

O SSG é uma ferramenta de aprendizado da linguagem extremamente fácil de ser comprendido mesmo por programadores que não possuam nenhum conhecimento de linguagens formais, dada a relativa simplicidade do mesmo quando comparado a grafos sem restrições, como por exemplo os apresentados em [KR83] para C e em [RP81] para Ada.

Finalmente, é interessante notar que o grafo apresentado no apêndice gera uma linguagem mais próxima da linguagem C do que muitas gramáticas disponíveis, como as de [HS87] e [KR88].

***Referências bibliográficas***

[ASU86] AHO, A.V.; SETHI, R.; ULLMAN, J.D. Compilers: Principles, Techniques and Tools. Addison-Wesley, 1986

[BE73] BERGE, C. Graphs and Hypergraphs. North-Holland, 1973

[RPS1] DE REMER, F.; PENELLO, T. Ada syntax chart. ACM Sigplan Notices, 16(9):48-59, set. 1981

Contém um grafo sintático para a linguagem Ada, com menos simbolos não terminais do que a gramática correspondente. O grafo apresentado é LALR(2), mas segundo os autores uma alteração simples, mas não muito elegante, permite transformar o grafo em LALR(1).

[HS87] HARBISON, S.; STEELE Jr, G. C - A Reference Manual, 2nd edition. Prentice-Hall, Englewood Cliffs, 1987.

[JO75] JOHNSON, S.C. Yacc - yet another compiler compiler. Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, N.J.

[JW73] JENSEN, K.; WIRTH, N. Pascal User Manual and Report. Springer-Verlag, New York, 1973.

[KR83] KERNIGHAN, B.; RITCHIE, D. Programmieren in C mit dem C reference manual. Carl Hanser Verlag, München, 1983.

Edição alemã do clássico "The C Programming Language". Inclui um grafo sintático sem restrições de nenhuma espécie, projetado por David Smith. Além do C original, inclui algumas extensões, como variáveis de enumeração.

[KR88] KERNIGHAN, B.; RITCHIE, D. The C Programming Language, 2nd edition. Addison-Wesley, 1988.

Segunda edição americana do clássico sobre a linguagem C. O texto foi alterado para refletir a definição da linguagem elaborada pelo comitê ANSI de padronização.

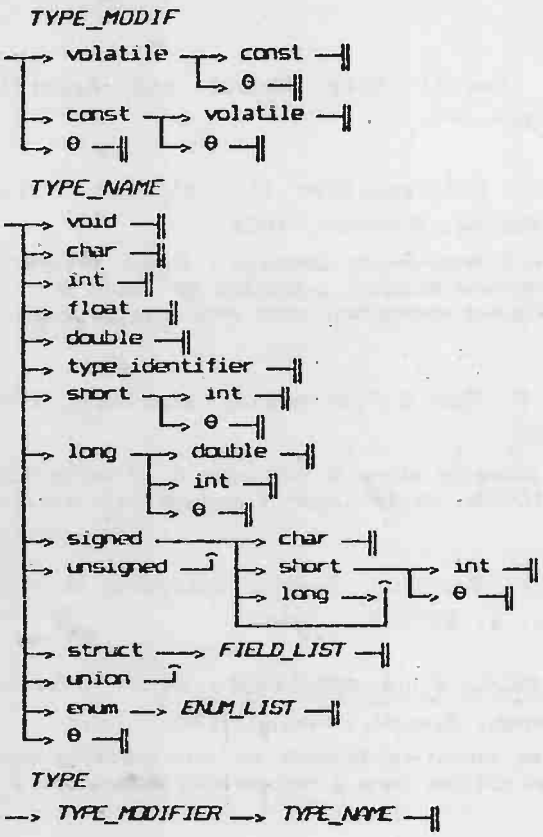
[SE79] SETZER, V.W. Non-recursive Top-down Syntax Analysis. Software- Practice and Experience, 9: 237-245, 1979

[SM86] SETZER, V.W.; HOMEM de MELO, I. A construção de um Compilador, 3ª edição. Editora Campus, Rio de Janeiro, 1986.

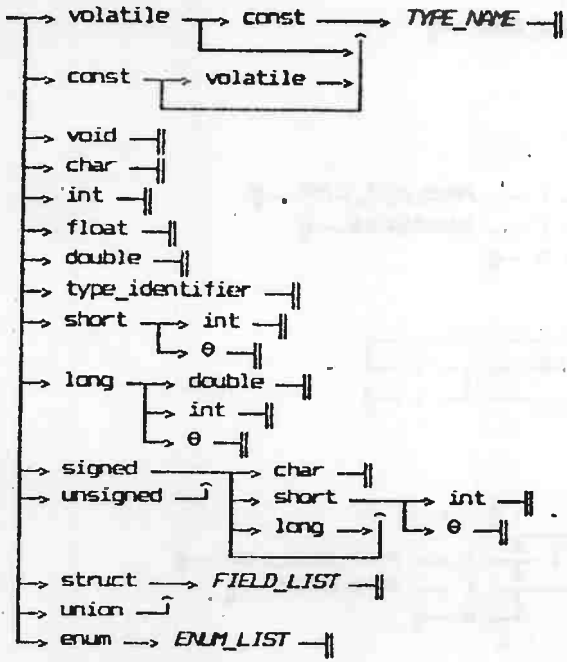
Contém o algoritmo de análise sintática baseado na interpretação dos grafos sintáticos simples e as rotinas para a recuperação automática de erros.

[WI76] WIRTH, N. Algorithms + Data Structures = Programs. Prentice-Hall, 1976.

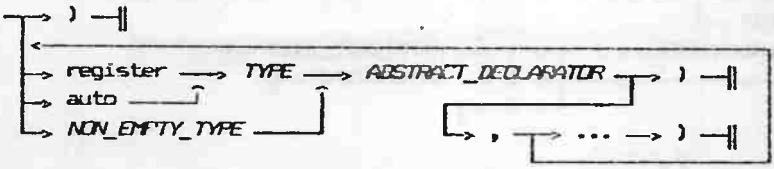
***Apêndice: grafo sintático simples para a linguagem C ANSI***



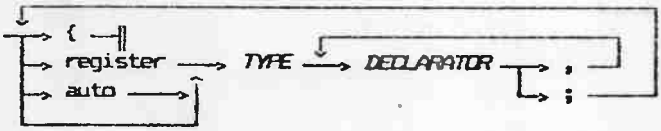
*NON_EMPTY_TYPE*



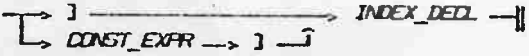

*PARM_DECL_LIST*



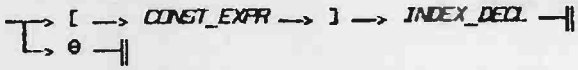

*PARAMETER_DECL*



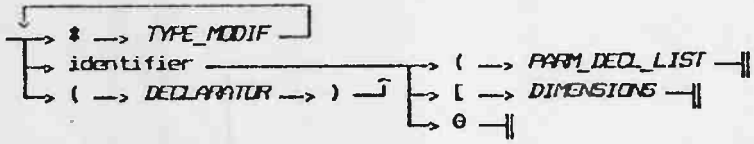

*DIMENSIONS*



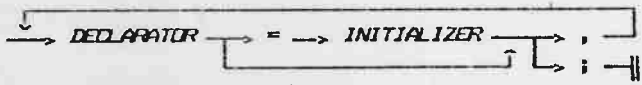
*INDEX_DECL*



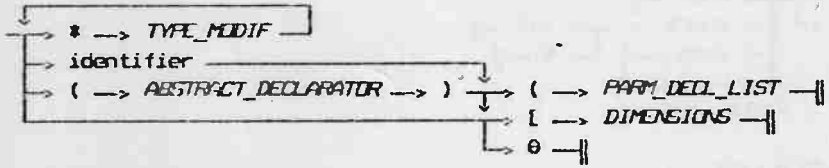

*DECLARATOR*



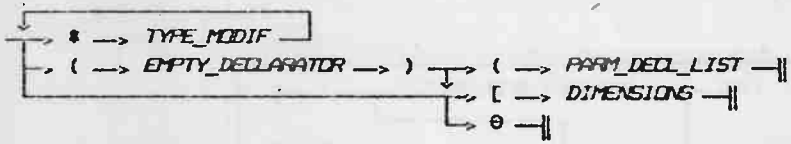

*DECLARATOR_LIST*



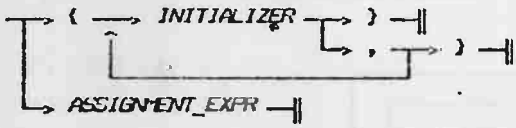

*ABSTRACT_DECLARATOR*



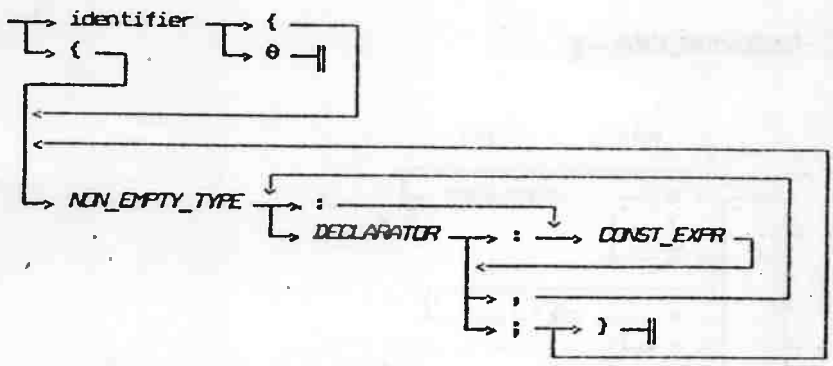

*EMPTY_DECLARATOR*



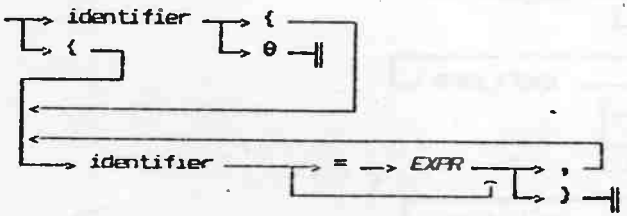

*INITIALIZER*



*FIELD_LIST*



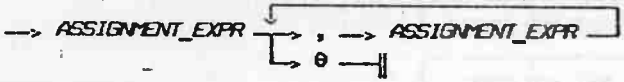

*ENUM_LIST*




*CONST_EXPR*

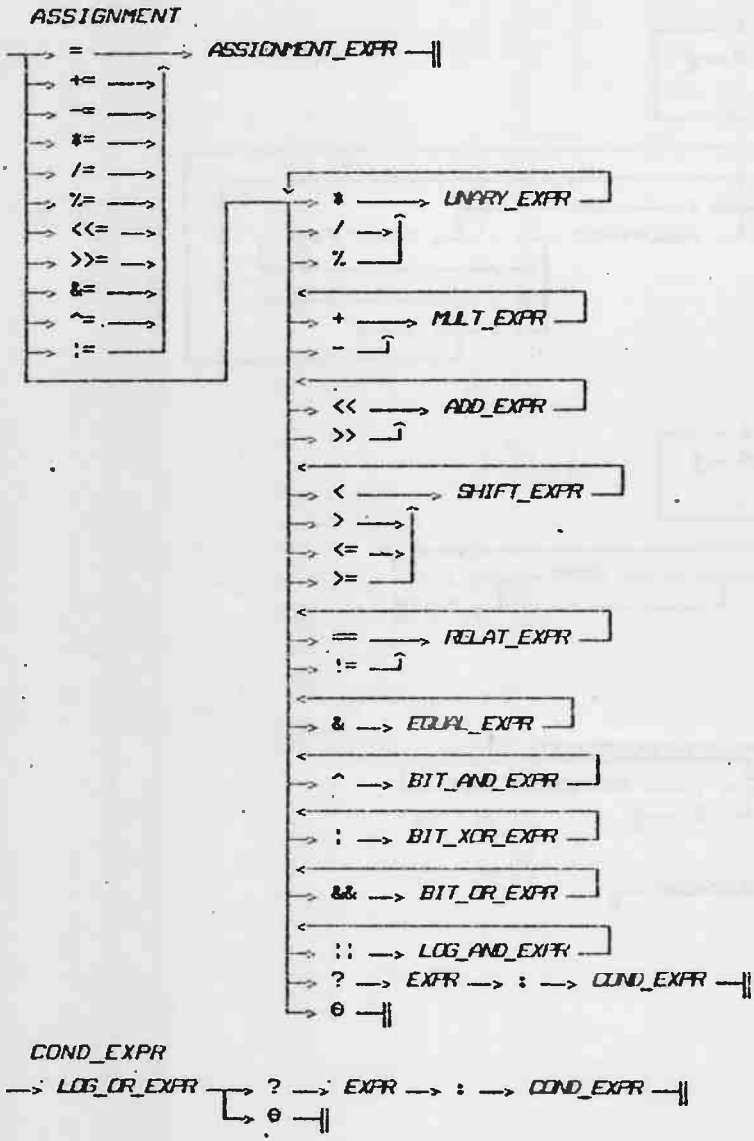
→ *COND_EXPR* ⫴

*EXPR*

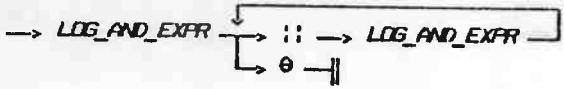



*ASSIGNMENT_EXPR*

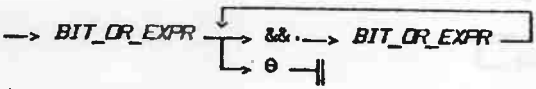
→ *UNARY_EXPR* → *ASSIGNMENT* ⫴


ASSIGNMENT
=
+=
-=
*=
/=
%=
<<=
>>=
&=
^=
!=
ASSIGNMENT_EXPR
*
/
%
UNARY_EXPR
+
-
MULT_EXPR
<<
>>
ADD_EXPR
<
>
<=
>=
SHIFT_EXPR
==
!=
RELAT_EXPR
&
EQUAL_EXPR
^
BIT_AND_EXPR
|
BIT_XOR_EXPR
&&
BIT_OR_EXPR
||
LOG_AND_EXPR
?
EXPR
:
COND_EXPR
0
COND_EXPR
LOG_OR_EXPR
?
EXPR
:
COND_EXPR
0

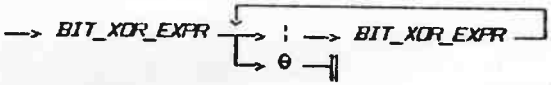
LOG_OR_EXPR



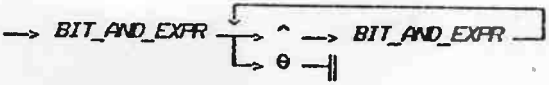

LOG_AND_EXPR



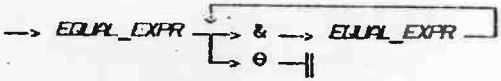

BIT_OR_EXPR



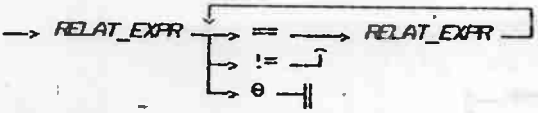

BIT_XOR_EXPR



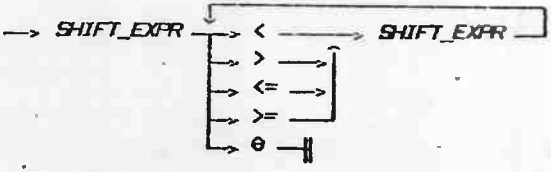

BIT_AND_EXPR



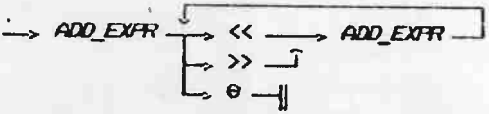

EQUAL_EXPR



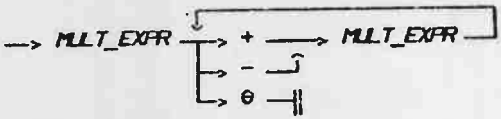

RELAT_EXPR



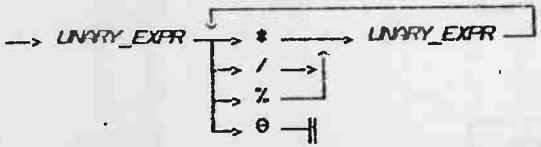

SHIFT_EXPR



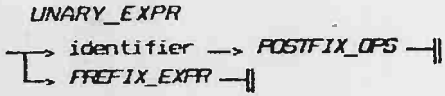
*ADD_EXPR*



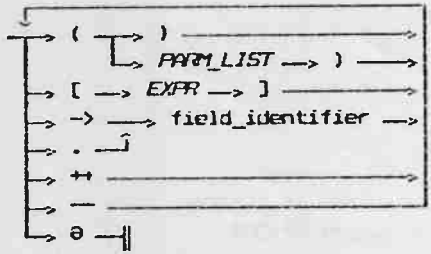

*MULT_EXPR*



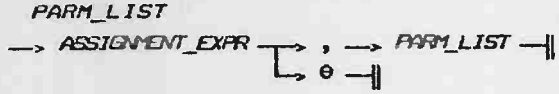

*UNARY_EXPR*



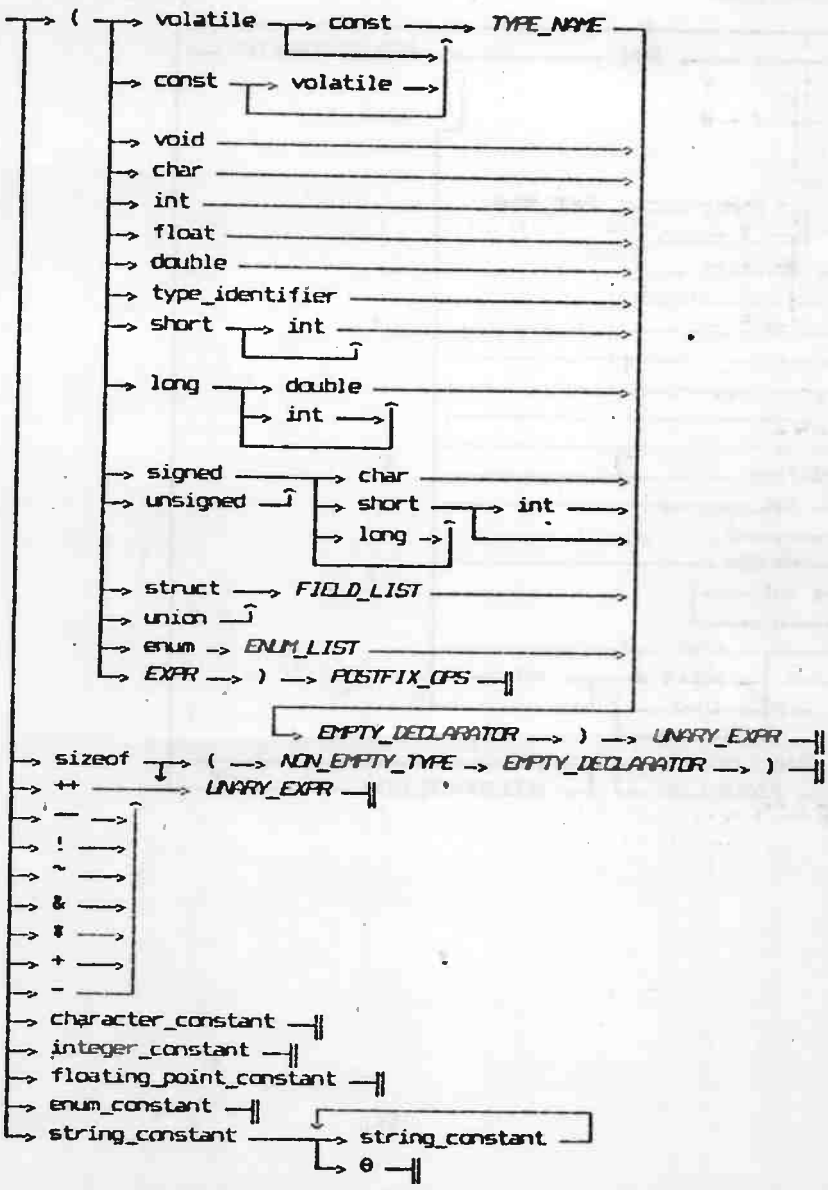

*POSTFIX_OPS*



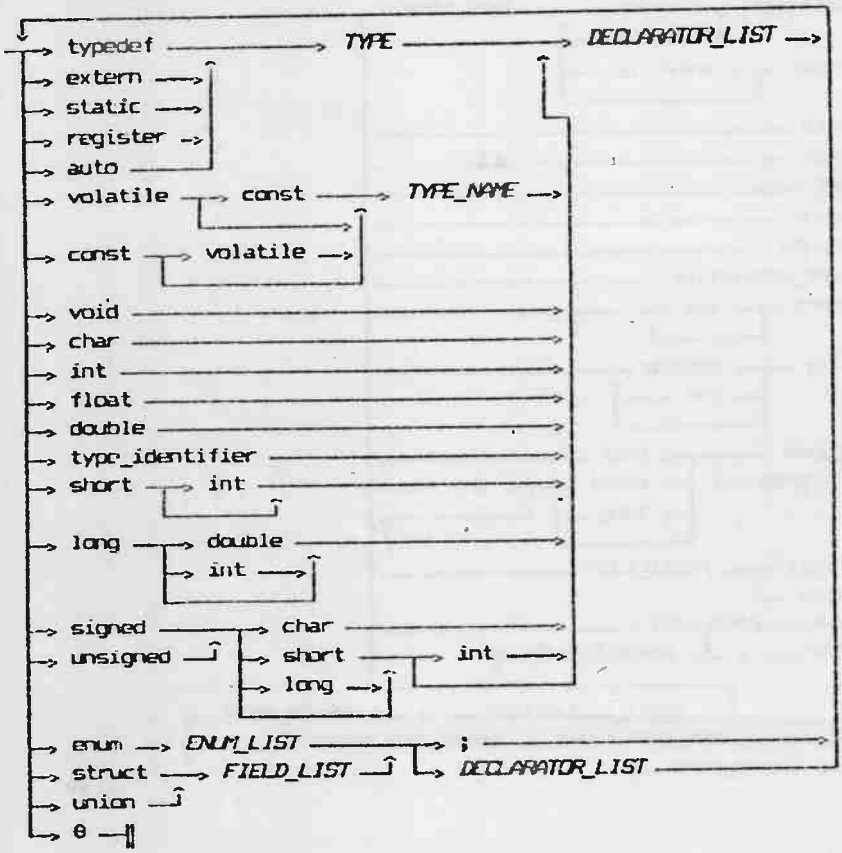

*PARM_LIST*



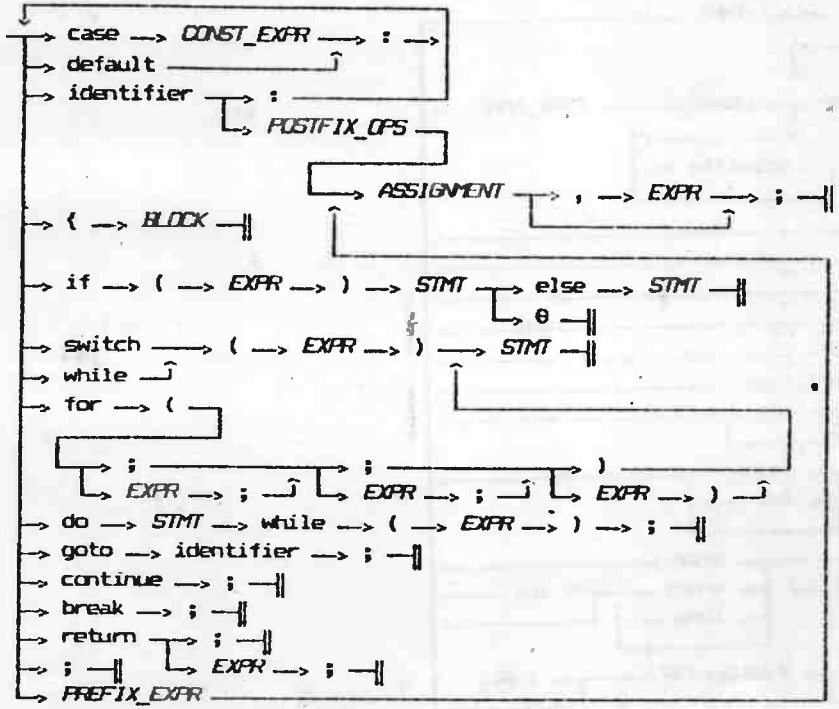
*PREFIX_EXPR*


(
volatile
const
TYPE_NAME
const
volatile
void
char
int
float
double
type_identifier
short
int
long
double
int
signed
unsigned
char
short
int
long
struct
FIELD_LIST
union
enum
ENUM_LIST
EXPR
)
POSTFIX_OPS
EMPTY_DECLARATOR
)
UNARY_EXPR
sizeof
(
NON_EMPTY_TYPE
EMPTY_DECLARATOR
)
++
UNARY_EXPR
--
!
~
&
*
+
-
character_constant
integer_constant
floating_point_constant
enum_constant
string_constant
string_constant
0

*LOCAL_DECLARATIONS*

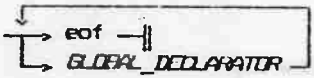


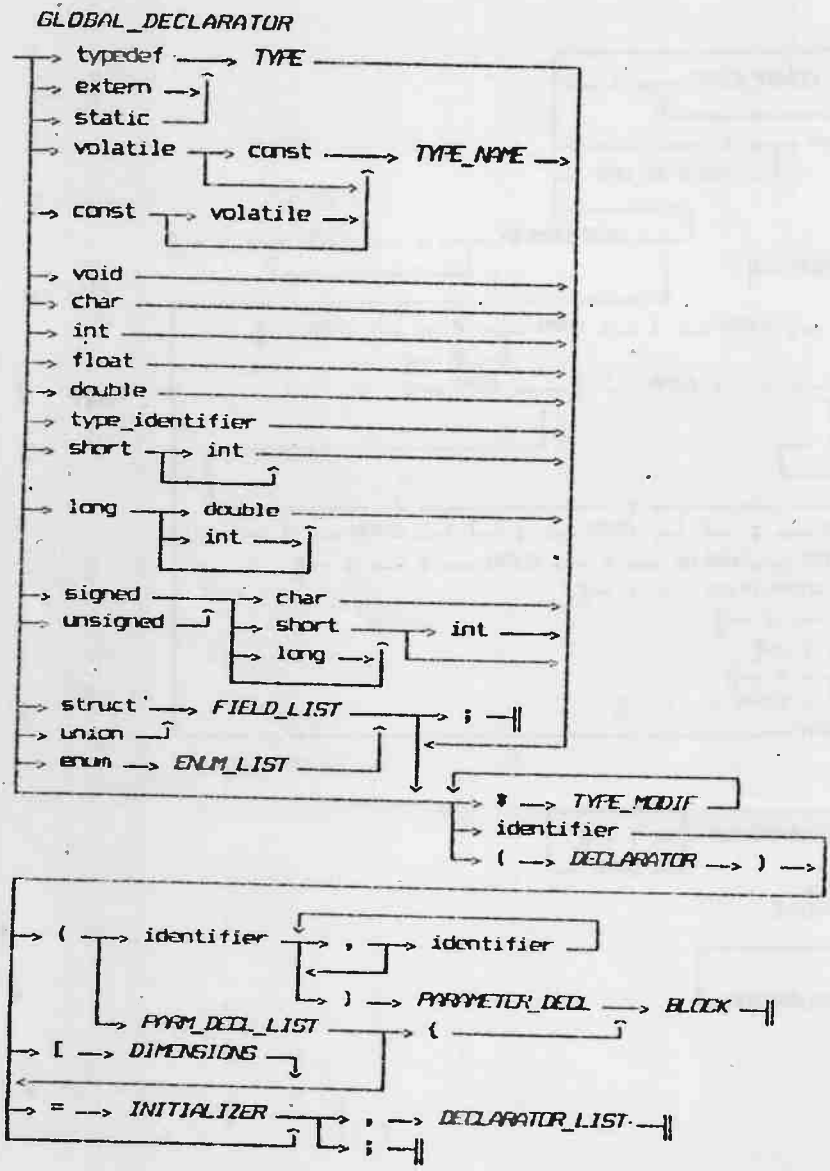
*STMT*




*BLOCK*


*PROGRAM_MODULE*




GLOBAL_DECLARATOR
typedef
TYPE
extern
static
volatile
const
TYPE_NAME
const
volatile
void
char
int
float
double
type_identifier
short
int
long
double
int
signed
unsigned
char
short
long
int
struct
union
enum
FIELD_LIST
ENUM_LIST
;
*
TYPE_MODIF
identifier
(
DECLARATOR
)
(
identifier
,
identifier
)
PARAMETER_DECL
BLOCK
PARM_DECL_LIST
(
[
DIMENSIONS
=
INITIALIZER
,
DECLARATOR_LIST
;

RELATÓRIOS TÉCNICOS
TÍTULOS PUBLICADOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA USP

RT-MAP-7702 - (Dezembro 1977)
V.W. Setzer
A Note on a Recursive Top-Down Analizer of N.Wirth

RT-MAP-7704 - (Dezembro 1977)
V.W. Setzer, M.M. Sanches
A linguagem "LEAL" para Ensino básico de Computação

RT-MAP-7802 - (Fevereiro 1978)
Sílvio Ursic, Cyro Patarra
Exact solution of Systems of Linear Equations with Iteractive Methods

RT-MAC-7803 - (Março 1978)
Martin Grötschel, Yoshiko Wakabayashi
Hypohamiltonian Digraphs

RT-MAP-7804 - (Maio 1978)
Martin Grötschel, Yoshiko Wakabayashi
Hypotraceable Digraphs

RT-MAP-7805 - (Junho 1978)
W. Hesse, V.W. Setzer
The Line-Justifier: an example of program development by transformations

RT-MAP-7809 - (Novembro 1978)
V.W. Setzer
Program development by transformations applied to relational Data-Base queries

RT-MAP-7811 - (Novembro 1978)
D.T. Fernandes e C. Patarra
Sistemas Lineares Esparsos, um Método Exato de Solução

RT-MAP-7812 - (Novembro 1978)
V.W. Setzer e G. Bressan
Desenvolvimento de Programas por Transformações: uma Comparação entre dois Métodos

RT-MAP-7814 - (Dezembro 1978)
Martin Grötschel e Yoshiko Wakabayashi
On the Complexity of the Monotone Asymetric Travelling Salesman Polytope I: HYPOHAMILTONIAN FACETS

RT-MAP-7901 - (Fevereiro 1979)
Martin Grötschel, Yoshiko Wakabayashi
On the Complexity of the Monotone Asymetric Travelling Salesman Polytope II: HYPOTRACEABLE FACETS

RT-MAP-7902 - (Junho 1979)
M.M. Sanches e V.W. Setzer
A Portabilidade do Compilador para a Linguagem LEAL

RT-MAP-7903 - (Julho 1979)
Martin Grötschel, Carsten Thomassen, Yoshiko Wakabayashi
Hypotraceable Digraphs

RT-MAP-8003 - (Setembro 1980)
Routo Terada
Fast Algoritms for NP-Hard Problems which are Optimal or near-Optimal with Probability one

RT-MAP-8004 - (Setembro 1980)
V.W. Setzer e R. Lapyda
Uma Metodologia de Projeto de Bancos de Dados para o Sistema ADABAS

RT-MAP-8005 - (Outubro 1980)
Imre Simon
On Brzozowski's Problem: $(1 \cup A)^m = A^m$

RT-MAP-8101 - (Fevereiro 1981)
Luzia Kazuko Yoshida e Gabriel Richard Bitran
Um Algoritmo para Problemas de Programação com Variáveis Zero-Um

RT-MAP-8103 - (Abril 1981)
V.W. Setzer, R. Lapyda
Design of Data Models for the ADABAS System using the Entity-Relationship Approach

RT-MAP-8105 - (Maio 1981)
U.S.R. Murty
Projective Geometries and their Truncations

RT-MAP-8106 -(Junho 1981)
V.W. Setzer, R. Lapyda
Projeto de Bancos de Dados, Usando Modelos Conceituais
(Este Relatório Técnico complementa o RT-MAP-8103. Ambos substituem o RT-MAP-8004 ampliando os conceitos ali expostos.)

RT-MAP-8107 - (Agosto 1981)
Maria Angela Gurgel, Yoshiko Wakabayashi
Embedding of Trees

RT-MAP-8201 - (Janeiro 1982)
Siang Wun Song
On a High-Performance VLSI Solution to Database Problems

RT-MAP-8202 - (Junho 1982)
Maria Angela Gurgel, Yoshiko Wakabayashi
A Result on Hamilton-Connected Graphs

RT-MAP-8205 - (Junho 1982)
Arnaldo Mandel
Topology of Oriented Matroids

RT-MAP-8206 - (Novembro 1982)
Erich J. Neuhold
Database Management Systems: A General Introduction

RT-MAP-8207 - (Novembro 1982)
Béla Bollobás
The Evolution of Random Graphs

RT-MAP-8208 - (Novembro 1982)
V.W. Setzer
Um Grafo Sintático para a Linguagem PL/M-80

RT-MAP-8209 - (Novembro 1982)
Jayme Luiz Szwarcfiter
A Sufficient Condition for Hamilton Cycles

RT-MAP-8302 - (Fevereiro 1983)
Belá Bollobás, Istvan Simon
Repeated Random Insertion into a Priority Queue

RT-MAP-8303 - (Julho 1983)
V.W. Setzer, P.C.D. Freitas e B.C.A. Cunha
Um Banco de Dados de Medicamentos

RT-MAP-8305 - (Julho 1983)
Arnaldo Mandel
The 1-Skeleton of Polytopes, oriented Matroids and some other lattices

RT-MAP-8306 - (Agosto 1983)
Arnaldo Mandel
Alguns Problemas de Enumeração em Geometria

RT-MAP-8307 - (Agosto 1983)
Siang Wun Song
Complexidade de E/S e Projetos Optimais de Dispositivos para Ordenação

RT-MAP-8402 - (Abril 1984)
V.W. Setzer
Manifesto contra o uso de computadores no Ensino de 1º Grau

RT-MAP-8404 - (Setembro 1984) (7 pgs)
Imre Simon
A Factorization of Infinite Words

RT-MAP-8405 - (Setembro 1984) (6 pgs)
Imre Simon
The Subword Structure of a Free Monoid

RT-MAP-8406 - (Setembro 1984) (25 pgs)
Jairo Z. Gonçalves e Arnaldo Mandel
Are There Free Groups in Division Rings?

RT-MAP-8407 - (Novembro 1984) (18 pgs)
Paulo Feofiloff and D.H. Younger
Vertex-Constrained Transversals in a Bipartite Graph

RT-MAP-8408 - (Novembro 1984) (126 pgs)
Paulo Feofiloff
Disjoint Transversals of Directed Coboundaries

RT-MAP-8409 - (Novembro 1984) (16 pgs)
Paulo Feofiloff e D.H. Younger
Directed cut transversal packing for source-sink connected graphs

RT-MAP-8501 - (Maio 1985) (11 pgs)
Siang Wun Song
Disposições Compactas de Árvores no Plano

RT-MAP-8502 - (Julho 1985) (11 pgs)
Paulo Feofiloff
Transversais de Cortes Orientados em Grafos Bipartidos

RT-MAP-8504 - (Setembro 1985) (110 pgs)
Christian Choffrut
Free Partially Commutative Monoids

RT-MAP-8505 - (Outubro 1985) (40 pgs)
Valdemar W. Setzer
Manifesto Against the use of Computers in Elementary Education

RT-MAP-8606 - (Agosto 1986) (105 pgs)
Júlio Michael Stern
Fatoração L - U e Aplicações

RT-MAP-8607 - (Agosto 1986) (73 pgs)
Afonso Galvão Ferreira
O Problema do Dobramento Optimal de PLAs

RT-MAP-8703 - (Fevereiro 1987) (20 pgs)
Imre Simon
The Nondeterministic Complexity of a Finite Automaton

RT-MAP-8704 - (Abril 1987) (8 pgs)
Imre Simon
Infinite Words and a Theorem of Hindman

RT-MAP-8707 - (Agosto 1987) (36 pgs)
Imre Simon
Factorization Forests of Finite Height

RT-MAP-8709 - (Março 1987) (31 pgs)
Routo Terada
Um Código Criptográfico para Segurança em Transmissão e Base de Dados

RT-MAC-8801 - (Janeiro 1988) (21 pgs)
Martin Grötschel, Yoshiko Wakabayashi
Facets of the Clique Partitioning Polytope

RT-MAC-8802 - (Fevereiro 1988) (52 pgs)
Martin Grötschel, Yoshiko Wakabayashi
A Cutting Plane Algorithm for a Clustering Problem

RT-MAC-8803 - (Março 1988) (14 pgs)
Martin Grötschel, Yoshiko Wakabayashi
Composition of Facets of the Clique Partitioning Polytope

RT-MAC-8804 - (Abril 1988) (14 pgs)
Imre Simon
Sequence Comparison: Some Theory and Some Practice

RT-MAC-8805 - (maio 88) (14 pgs)
Imre Simon
Recognizable Sets with Multiplicities in the Tropical Semiring

RT-MAC-8806 - (junho 88) (40 pgs)
Valdemar W. Setzer, Ervino Marussi
LDT: Um Gerador Universal de Aplicações para Processamento de Dados

RT-MAC-8807 - (agosto 88) (16 pgs)
Routo Terada
Probabilistic Analysis of Optimal Algorithms for Three NP-Hard Problems

RT-MAC-8808 (setembro 88) (18 pgs)
Valdemar W. Setzer
Um Sistema Simples para Documentação Semi-Automática de Programas

RT-MAC-8809 (novembro 88) (20 pgs)
Valdemar W. Setzer, R. Hirata Jr.
Hipo-PC: Um "Software" Educacional para Introdução ao Computador

RT-MAC-8901 (abril 89) (97 pgs)
Arnaldo Mandel
O editor de texto EPSILON

RT-MAC-8902 (abril 89) (11 pgs)
Valdemar W. Setzer, R. Hirata Jr.
Dia da Computação

RT-MAC-8903 (março 89) (16 pgs)
Valdemar W. Setzer, N. A. Zaguir
Um Banco de Dados para Criaçao e Seleção Zebuína

RT-MAC-8904 (junho 89) (8 pgs)
Imre Simon
Properties of Factorization Forests

RT-MAC-8905 (agosto 89) (24 pgs)
Valdemar W. Setzer, Roberto C. Mayer
Grafos Sintáticos Simples e Um Grafo para a Linguagem C ANSI