O USO DE GRAMÁTICA DE ATRIBUTOS NA FORMATAÇÃO DE NOTAÇÕES DIAGRAMÁTICAS

Favero, E.L. Esperanca, L.G. Silva, M.S. Price, R.T.

Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação
Universidade Federal do Rio Grande do Sul

Fax: (0512) 244164 - bitnet:tomprice@SBU.UFRGS.ANRS.BR

RESUMO

Este trabalho introduz o uso do mecanismo gramática de atributos na formatação de notações diagramáticas como as usadas na engenharia de software (por exemplo diagrama de fluxo de dados e diagrama de Nassi-Schneiderman). Uma gramática de atributos estende uma gramática livre de contexto com equações semânticas. Assim a linguagem é descrita tanto a nível sintático (livre de contexto) como de semântica estática (sensível ao contexto). O nível sintático compreende os aspectos relacionados com a estrutura da linguagem, estruturas de grafos (nodos/arcos) ou de árvore. O nível de semântica estática compreende todos os aspectos que não podem ser especificados na sintaxe. Neste nível serão descritas as equações que definem a formatação. Discute-se também um mecanismo para a especificação dos elementos léxicos como classes compostas de objetos geométricos.

Uma vez que o metodo é baseado em descrições gramaticais, torna-se adaptável para distintas notações diagramáticas.

1 INTRODUÇÃO

Este trabalho introduz o uso do mecanismo gramática de atributos (GA) [AHO 86] na formatação de notações diagramáticas como as usadas na engenharia de software (por exemplo diagrama de fluxo de dados e diagrama de Nassi-Schneiderman (N-S) [MAR 87]).

Gramática de atributos para diagramas

A importância da descrição das notações diagramáticas (formatação e aspectos dependentes de contexto) em **GA**, está na existência de algoritmos de avaliação de incremental de GAs [REP 83].

A partir da descrição gramatical, um editor

orientado por estrutura cria um ambiente iterativo de edição. Os itens dos menus (de inclusão e exclusão) são gerados a partir da sintaxe da linguagem descrita através de uma GLC. O processamento dependente de contexto é executado a partir das equações da GA; a formatação (posicionamento dos nodos na página) de um diagrama é especificada como um conjunto de regras que definem os valores das coordenadas para os símbolos terminais (e.g. nodos e arcos) que são exibidos na tela.

Linguagens diagramáticas

Do ponto de vista de representação interna existem dois tipos de linguagens diagramáticas: com estrutura de grafo (com nodos compartilhados) e arvore. O diagrama de fluxo de dados (DFD), figura 1, apresenta uma estrutura de grafo porque possue nodos compartilhados. O nodo **x**, por exemplo, é compartilhado por três arcos. O diagrama de Nassi-Schneiderman (figura 2) possue a representação interna (hierárquica) em forma de árvore, i.e. pode ser representado na estrutura de nodos e arcos sem nodos compartilhados. Esta estrutura hierarquica também é um grafo, porém será chamada apenas de árvore.

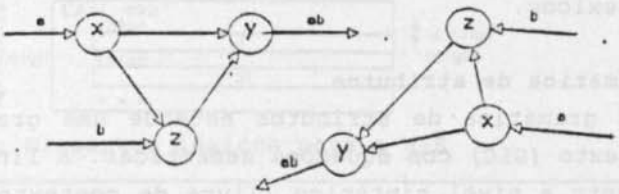


figura 1: Estrutura de grafo para um DFD; duas representações concretas para um mesmo grafo.

Quanto à formatação algumas linguagens possuem uma formatação "livre"; em um DFD, por exemplo, cada nodo pode ser criado e movimentado para qualquer posição na tela (fig. 1). Porém, algumas linguagens diagramáticas possuem uma estrutura hierárquica na sua formatação, como é o caso do diagrama de Nassi-Schneiderman - a formatação é dependente

da estrutura hierárquica do documento (fig. 2).

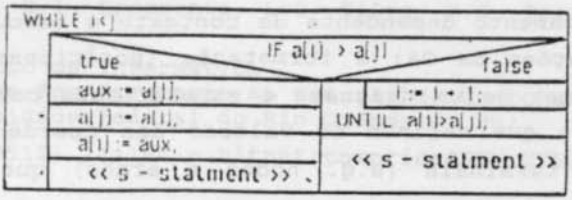


figura 2: Diagrama Nassi-Schneiderman (N-S)

Num trabalho anterior [FAV 89a] exemplificou-se o uso da metalinguagem (LDE) baseada em GA, estendida a nivel de sintaxe, para permitir a especificação de linguagens com estrutura de grafo, onde os nodos possuem formatação livre. Deu-se ênfase na descrição de estruturas de grafo e a especificação de verificações sensiveis ao contexto. Este trabalho complementa aquele apresentando a descricão de uma linguagem diagramática do tipo árvore com formatação não livre; da-se ênfase na formatação e na descrição dos componentes léxicos. Muitos dos conceitos apresentados são também utilizados na descrição das linguagens com formatação livre, por exemplo o mecanismo usado na descrição dos componentes léxicos.

Níveis da gramática de atributos

Uma gramática de atributos estende uma gramática livre de contexto (GLC) com equações semânticas. A linguagem é descrita tanto a nível sintático (livre de contexto) como de semântica estática (sensível ao contexto). Assim, uma linguagem é descrita em três níveis:

i) o **nível léxico**: neste nível são definidos os elementos léxicos terminais da descrição sintática. Por exemplo, para diagramas os léxicos são as "caixinhas" do diagrama. Para linguagens textuais, neste nível, são definidos os símbolos terminais: palavras reservadas, valores numéricos, nomes de variáveis, etc.

ii) o **nível sintático**: neste nível são definidas todas as

combinações válidas dos elementos léxicos. É usualmente definido por uma GLC. O nivel sintático compreende os aspectos relacionados com a estrutura da linguagem: grafo e árvore.

iii) o **nivel semântico**: neste nivel são definidos todos os detalhes da linguagem que não são especificáveis adequadamente no nivel sintático; é feita a verificacão de nomes declarados, verificação de tipos; pode ser gerado código intermediário; são definidas tabelas de simbolos; formatação, etc.

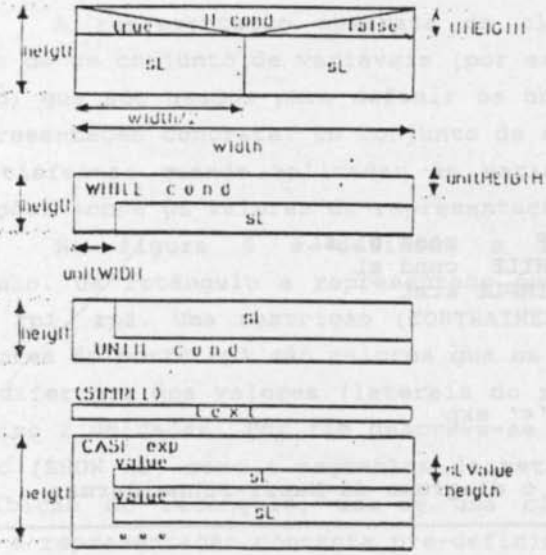


figura 3: Elementos léxicos para o N-S

2 ESPECIFICAÇÃO DA SINTAXE PARA O DIAGRAMA DE N-S

Na especificação do diagrama de N-S, abaixo, considera-se apenas uma página do diagrama. Numa implementação real deve-se considerar o problema do refinamento de elementos da página em outras páginas (ver [PRI 89]).

O diagrama N-S, figura 2, é constituído de caixinhas decoradas com textos. A especificação da sintaxe

define a combinação dos diferentes elementos léxicos que estão representados na figura 3, formando diagramas como o da figura 2. Por exemplo um comando **if** é formado pela combinação da "caixinha" !IF com a condição **Cond** e duas listas de comandos **sL**, figura 3 e 4.

```
%significado das abreviações para identificadores
s: (statement) comando
sL: (statement-list) lista de comandos
!IF, !WHILE, !SIMPLE :terminais diagramáticos
cond : expressão do tipo booleano
exp  : expressão numérica
stmt : (statement) comando textual%

D :: gr --> sL

     s  --> if
     s  --> while
     s  --> case
     s  --> repeat
     s  --> simple

     if      --> !IF     cond sL sL
     while   --> !WHILE  cond sL
     simple  --> !SIMPLE stmt

     sL --> s sL
     sL --> ^

T :: cond --> exp '<' exp
T :: stmt --> ...
```

figura 4: GLC para o diagrama de Nassi-Schneiderman

Os elementos terminais (léxicos) da gramática são pré-fixados pelo operador "!"; são escritos em letras maiúsculas (!IF, !WHILE, etc.). As produções textuais são prefixadas por **T::**, indicando que a produção deve ser processada pelo editor de texto; as produções prefixadas por **D::** são processadas pelo editor diagramático.

3 DESCRIÇÃO DOS COMPONENTES LÉXICOS

As notações diagramáticas possuem como léxicos objetos geométricos compostos como "bolhas", "caixinhas", "setas", etc. Para descrever estes elementos utiliza-se um mecanismo de classes baseado numa linguagem do tipo

"constraint-based" proposta por [BAR 87] para a modelagem de sólidos em um sistema gráfico interativo.

A notação para descrever as classes é introduzida por um exemplo. Um elemento léxico é descrito como uma classe composta de outras classes, primitivas ou não. As classes primitivas são pontos, segmentos de reta, poligonos, etc. Uma classe, como mostra a figura 5, é descrita em três secções: na primeira é definida a representação abstrata (CLASS IS); na segunda são definidas restrições (CONSTRAINED BY) sobre os valores da representação abstrata; e, na última define-se a representação concreta (SHOW AS).

A representação abstrata da classe armazena os valores de um conjunto de variáveis (por exemplo o tamanho e posição) que são usados para definir os objetos geométricos da representação concreta. Um conjunto de equações que devem ser satisfeitas quando aplicadas às variáveis, definem as restrições sobre os valores da representação abstrata.

Na figura 5 é definida a figura geométrica retângulo. Um retângulo é representado por (CLASS IS) dois pontos **rp1, rp2.** Uma restrição (CONTRAINED BY) garante que os valores do ponto **rp1** são maiores que os valores de **rp2**, e que a diferença dos valores (laterais do retângulo) seja de no mínimo 2 unidades Por fim descreve-se que o retângulo é exibido (SHOW AS) como 4 segmentos de reta **s1, s2, s3, s4.** Na exibição do retângulo, usa-se uma classe **lineSeg** que possui a representação concreta pré-definida.

```
CLASS rectangle IS
  rp1 : point = (x=1,y=1);
  rp2 : point = (x=3,y=3);
CONSTRAINED BY
   (rp1.x - rp2.x) > 2;
   (rp1.y - rp2.y) > 2;
SHOWN AS
  s1 : lineSeg = (p1=(rp1.x, rp1.y), p2=(rp1.x, rp2.y));
  s2 : lineSeg = (p1=(rp1.x, rp2.y), p2=(rp2.x, rp2.y));
  s3 : lineSeg = (p1=(rp2.x, rp2.y), p2=(rp1.x, rp2.y));
  s4 : lineSeg = (p2=(rp1.x, rp2.y), p1=(rp1.x, rp1.y));
END
```

figura 5: descrição para uma classe retângulo

A edição de classes

Para as linguagens com formatação livre (não é o caso do diagrama de Nassi-Schneiderman) são providas operações de edição de classes: criação/remoção, aumento/redução do tamanho e movimentação. A edição tem por base os conceitos de envelope e pegadores (ver por ex. [MEL 89], [FAV 89b]).

As operações de edição atualizam os valores "default" das classes. A edição de uma classe ocorre sobre a sua representação concreta; a partir da alteração de uma das subclasses da representação concreta deve-se propagar a modificação para a representação abstrata, atualizando-se a classe. A atualização da classe ocorre somente se as restrições são satisfeitas.

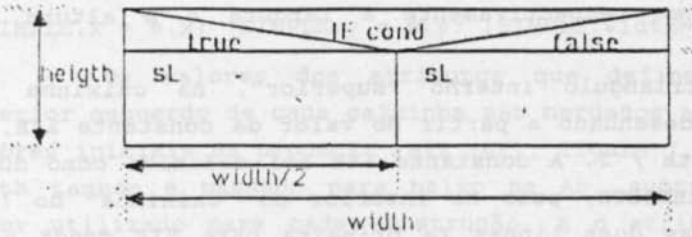
Atributos associados a léxicos

A especificação do léxico !IF ilustrada na figura 6, é discutida aqui. Os demais léxicos seriam definidos de maneira análoga.

A nível de GA cada elemento léxico é visto e tratado como uma entidade fechada - "caixa preta", com parâmetros de entrada (atributos herdados) ou de saída (atributos externos). Os atributos do tipo herdado são recebidos de nodos superiores na AS. Os atributos do tipo externo* são visíveis aos nodos superiores na AS. Na descrição do diagrama de N-S não se faz uso de atributos externos; exemplos que usam atributos externos são encontrados em [FAV 89b].

---

* Um atributo externo é similar a um atributo sintetizado. Porém, formalmente, não é o mesmo: um atributo externo é sempre associado a um léxico. Observação: na descrição da formatação, na fig. 7, são utilizados atributos sintetizados e também herdados.




```
1. ( const UnitHEIGTH = 1;
2.         ifH    = 2*UnitHEIGTH;
3.   inh    INTEGER x, y, width, heigth : !IF;)

4. CLASS ClassIf:!IF IS

5.    p1 :point = (x,y);
6.    p2 :point = (x+width, y+heigth);
7. CONSTRAINED BY
8. SHOW AS
9.     r   = rectangle(p1,p2);
10.    l1 = lineSeg(p1,(p1.x+width/2, p1.y+ifH));
11.    l2 = lineSeg((p1.x+width/2, p1.y+ifH), (p2.x,p1.y));
11.    t2 =text("true",(p1.x,p1.y+ifH/2),(p2.x/2,p1.y+ifH);
12.    t3 =text("false",(p2.x/2,p1.y+ifH/2),(p2.x,p1.y+ifH);
13. END
```

figura 6: especificação do lexico !IF

A associação entre a classe, que define o objeto geométrico no nivel léxico, e o simbolo terminal para a GA é feita pelo nome da classe seguido do identificador do símbolo terminal. Na figura 7 a construção, da linha 4,

CLASS ClassIf:!IF IS ...

associa o terminal da GA **!IF** com a classe nomeada **ClassIF.**

Além da associação dos nomes é necessário fazer um mapeamento entre variáveis e atributos; na figura 6 cada atributo, definido para o terminal !IF, está associado à variável de mesmo nome, na representação abstrada. Por exemplo, os atributos herdados **x, y, width, heigth** declarados para o terminal **!IF**, na linha 3, são associados aos pontos **p1** e **p2** que definem a representação abstrata, linha 5 e 6.

Na classe da figura 6, que define o léxico da !IF, os atributos **x** e **y** representam a posição associada ao canto

superior esquerdo da "caixinha" do **IF** e os atributos **width** e **heigth** definem respectivamente a largura e a altura da "caixinha".

O triângulo interno "superior", na caixinha da figura 6, é desenhado a partir do valor da constante **ifH**, e do valor **width / 2**. A constante **ifH** foi definida como duas vezes a **unitHEIGTH**, pois no interior da "caixinha" do !IF são utilizadas duas linhas (a primeira para "IF cond" e a segunda para "true", "false"). Os retângulos internos são desenhados pelas caixinhas **sL** internas.

A classe primitiva **text** exibe o texto centralizado no interior do envelope definido pelos dois pontos.

4 ESPECIFICAÇÃO DA FORMATAÇÃO

A especificação da formatação para o N-S descreve como são calculados os valores dos atributos que definem as posições das caixinhas na tela. O cálculo dos valores é executado pelo avaliador de GA.

O ponto de partida para a especificação da formatação é, por um lado, a definição dos atributos utilizados pelo nivel léxico e, por outro lado, a estrutura de árvore, definida na sintaxe, onde os atributos podem passar de nodo em nodo. Estas informações guiam o processo da especificação.

Os símbolos do tipo texto (**cond, stmt**, etc.) possuem os atributos (**x, y, width**) que definem o envelope para o texto. Assume-se que o texto é exibido em apenas uma linha, assim não é necessário o atributo **heigth.** O editor textual usará estes atributos para definir a janela de edição, quando é ativado.

Para denotar que todos os valores de atributos (com mesmo nome) associados a um símbolo são passados para outro símbolo usa-se a notação abreviada "*". Por exemplo a equação que define os atributos do terminal !SIMPLE

```
(inh  INTEGER width, x, y : !SIMPLE, s;
      !SIMPLE.* = s.*;)
```

é equivalente a

```
(!SIMPLE.x = s.x; !SIMPLE.y = s.y; !SIMPLE.width=s.width;).
```

Os valores dos atributos que definem o canto superior esquerdo de cada caixinha são herdados a partir dos valores iniciais da produção raiz (gr), figura 7. O atributo **width** também é passado para baixo na AS, subtraindo-se o valor utilizado para cada construção. E o atributo **heigth** (sintetizado) é calculado a partir das estruturas mais internas para as externas, figura 7.

Para se entender a especificação é necessário acompanhar, comparar, cada produção, da figura 7, com a sua respectiva caixinha na figura 3. Abaixo é mostrada a especificação da produção **if**.

```
if      --> !IF !cond sL1 sL2
 ( sL1.width = if.width / 2;
   sL2.width = if.width / 2;
   if.heigth = Max(sL1.heigth, sL2.heigth);
   sL1.y     = if.y + ifH;
   sL2.y     = if.y + ifH;
   sL1.x     = if.x;
   sL2.x     = if.x + if.width / 2;
   !IF.*     = if.*;
   cond.*    = if.*;)
```

Na produção **if** o atributo **sL1.width**, da sublista de comandos, recebe o valor **if.width/2**; metade do valor recebido pelo **if**. O valor do canto superior esquerdo da caixinha da segunda sublista (**sL2**) é **if.y+ifHEIGTH** para **y** e **if.x+if.width/2** para **x**. O atributo **if.heigth**, sintetizado, recebe a maior das alturas das sublistas (**sL1**, **sL2**).

```
(inh  INTEGER heigth, width, x, y : !IF, !WHILE, !REPEAT,
                                    !CASE, !SIMPLE;
 synt INTEGER heigth : if, while, simple, s, sl;
 inh  INTEGER width, x, y  : if, while case, simple, s, sl
                             exp, cond, stmt, !value;)

( Const unitWIDTH  = 1;
        maxWIDTH   = 80;
        unitHEIGTH = 1;
          ifH = 2 * unitHEIGTH; )

gr --> sL
```

```
  ( sL.width = maxWIDTH;
    sL.x     = 0;
    sL.y     = 0;)

s  --> if
  ( if.x = s.x;
    if.y = s.y;
    if.width = s.width;
    s.heigth = if.heigth; )

s  --> while
( while.x = s.x;
  while.y = s.y;
  while.width = s.width;
  s.heigth    = while.heigth; )

s  --> .../* case, repeat, etc. */

s --> simple
( simple.x = s.x;
  simple.y = s.y;
  simple.width = s.width;
  s.heigth     = simple.heigth; )

simple  --> !SIMPLE  stmt
  ( simple.heigth  = unitHEIGTH;
    !SIMPLE.* = simple.*;
    stmt      = simple.*;)

if      --> !IF cond sL1 sL2
 ( sL1.width = if.width / 2;
   sL2.width = if.width / 2;
   if.heigth = Max(sL1.heigth, sL2.heigth);
   sL1.y     = if.y + ifH;
   sl2.y     = if.y + ifH;
   sL1.x     = if.x;
   sL2.x     = if.x + if.width / 2;
   !IF.*     = if.*;
   cond.*    = if.*; )

while  --> !WHILE cond sL
 ( while.heigth = sl.heigth   + unitHEIGTH;
   sl.width     = while.width - unitWIDTH;
   sL.y     = while.y + unitHEIGTH;
   sL.x     = while.x - unitWIDTH;
   !WHILE.* = while.*;
   cond.*  = while.*; )

sL --> s sL
  ( sL1.heigth = s.heigth + sL2.heigth;
    sL2.width  = sL1.width;
    sL2.y = sL1.y + s.heigth;
    sL2.x = sL1.x;
    s.*   = sL1.*;)

sL --> ^
  ( sL.heigth = 0; )
```

```
cond --> exp '<' exp
cond --> ...
stmt --> ...
```

figura 7: GA para o diagrama de Nassi-Schneiderman

Na especificação define-se um valor constante para a largura do diagrama (maxWIDTH); o comprimento não tem um limite fixo, é o valor sintetizado pelo atributo **sL.heigth** no nodo raiz (Gr) da árvore.

Na hora de exibir o diagrama na tela é necessário ajustar o comprimento real do diagrama com o tamanho da tela. A altura total do diagrama (**sL.heigth**) pode ser usada para isto; o ajuste se dá pela relação entre o número de pontos da tela e o valor **sL.heigth**.

5 CONCLUSÃO

Este trabalho é resultado da experimentação no projeto ADS/UFRGS com o uso de gramáticas para especificar notações diagramáticas. Busca-se adaptar para notações diagramáticas a tecnologia desenvolvida na construção de editores orientados por estrutura para linguagens textuais. A notação utilizada aqui é processada por um protótipo de gerador de editores diagramáticos desenvolvido no projeto ADS/UFRGS (uma descrição deste gerador é encontrada em [FAV 89b]).

A implementação do avaliador de GA para o gerador de editores é descrita em [ESP89]; a descrição de vários processadores de GA é encontrada em [DER 88]; uma generalização do mecanismo de avaliação de GA é discutida em [HOO 87].

O uso do mecanismo de classes para descrição dos elementos léxicos deve-se as seguintes facilidades:

- a linguagem textual que define as classes serve como documentação da especificação e é, também, portável.

- existem algoritmos que, automaticamente, recalculam as variáveis e avaliam as restrições, das classes inter-relacionadas, após as operações de edição [BAR 87].

O uso de GA para a criação de editores para ferramentas diagramáticas apresenta algumas vantagens em relação aos métodos usuais, como o baseado em arcos e nodos [MEL 89]:

- permite a especificação dos aspectos semânticos das notações diagramáticas; identifica-se, por exemplo, nomes duplicados, nomes não declarados, etc.

- possibilita, também, a especificação da formatação como exemplificado para o N-S.

- a possibilidade de descrição de ferramentas diagramáticas em GA reduz o esforço necessário para o desenvolvimento e integração de ferramentas (textuais e diagramáticas) para a construção de Ambientes de Desenvolvimento de Software. Permite o compartilhamento de informações de tabelas de símbolos representadas em um banco de dados global [PRI 89].

- com a especificação das notações diagramáticas (vistas como linguagens) através de formalismo GA pode-se alcançar um avanço técnico similar ao alcançado com a formalização da construção de compiladores.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[AHO 86] AHO, ALFRED V.; SETHI, RAVI; ULLMAN, JEFFREY D.; **Compilers: Principles, Techniques, and Tools.** Reading, Addison-Wesley, 1986.

[BAR 87] BARFORD, LEE ALTON; ZANDEN, BRADLEY T. VANDER; **Attribute grammars constraint-based graphics systems.** Ithaca, Cornell University, June 1987.

[DER 88] DERANSART, P.; JOURDAM, M.; LORHO B.; **Attribute grammars. Definitions, Systems and Bibliography.** LNCS nro 323, Springer Verlag, 1988.

[ESP 89] ESPERANÇA, L.G.; **Um interpretador de gramáticas de atributos.** Porto Alegre, CIC/UFRGS,Dezembro, 1989. (Trabalho de diplomação)

[FAV 89a] FAVERO, E.L; PPRICE, R.T.; **A Implementação de um editor diagramático para DFD, baseada em formalismos gramaticais.** IX Congresso da SBC, Uberlândia, julho 1989.

[FAV 89b] FAVERO, E.L; **Um editor orientado por estruturas para linguagens diagramáticas.** Dezembro, 1989. CPGCC-UFRGS. (Dissertação de Mestrado).

[HOO 87] HOOVER, ROGER; **Incremental graph evaluation.** Ithaca, Cornell University, 1987. (Phd Thesis)

[MAR 87] MARTIN, JAMES; **Recommended diagramming standards for analysts and programmers: a basis for automation.** Englewood Cliffs, Prentice-Hall, 1987.

[MEL 89] MELO, W.L.M.; **Uma proposta de um editor diagramático generalizado**, Porto Alegre, CPGCC da UFRGS, Março, 1989. (Dissertação de Mestrado)

[PRI 89] PRICE, R.T.; FAVERO E.L. **Um editor híbrido (texto e diagramas) orientado por estruturas (do tipo grafo e árvore).** III Simpósio Brasileiro de Eng. de Sof. Recife - Pernambuco, Out. 1989.

[REP 83] REPS, T.; TEITELBAUM, T.; DEMERS, A. **Incremental Contex-dependent Analysis for Language-based Editors".** Acm transactions on programming languages and systems, New York, 5(3):_, July 1983.