ORDRE DE CALCUL DES ATTRIBUTS

On ne peut pas décider l'ordre de calcul des attributs par inspection des règles des noeuds voisins:

$P \rightarrow Q \quad P.X = Q.a \; Q.b = P.X$
$\quad P.Y = Q.c \; Q.d = P.Y$
$Q \rightarrow Q \quad Q^1.a = Q^2.a \; Q^2.b = Q^1.b$
$\quad Q^1.c = Q^2.c \; Q^2.d = Q^1.d$
$Q \rightarrow R \quad Q.c = Q.b \; Q.a = 0$
$Q \rightarrow S \quad Q.a = Q.d \; Q.c = 0$

Et si on ajoutait
$Q \rightarrow T \quad Q.c = Q.b \; Q.a = Q.d$
tout programme avec un T produirait une boucle dans le calcul des attributs (sauf si le T ne descendait pas d'un P...)

Donc, nécessité de trouver si un programme donné produit une boucle dans le calcul des attributs et, sinon, quel est (un) bon ordre de les calculer.

REPRESENTATION PAR UN GRAPHE

On construit un graphe orienté qui contient un sommet correspondant à chaque attribut de l'arbre syntaxique; il y a un arc d'un attribut A_1 à un autre A_2 ssi le calcul de A_1 dépend *directement* de A_2.

Il y a une boucle dans le calcul des attributs ssi il existe un circuit dans le graphe.

Le choix d'un ordre de calcul des attributs correspond au choix d'un ordre sur les sommets tel que s'il existe un arc de A_i à A_j, A_j précède A_i dans l'ordre.

Trouver un tel ordre s'appelle faire un *tri topologique* du graphe (ou de l'ordre partiel représenté par le graphe).

UN ALGORITHME DE TRI TOPOLOGIQUE

D'abord une procédure P (récursive) qui affiche dans un bon ordre tous les prédécesseurs (directs ou indirects) d'un noeud n sauf ceux qui sont déjà *marqués* ou séparés de n par un sommet marqué; P a pour effet de bord de marquer chaque sommet affiché.

$si\ n\ n'est\ pas\ marqué$
$\quad alors\ marquer\ n;$
$\qquad pour\ tout\ prédécesseur\ direct\ s\ de\ n$
$\qquad\quad faire\ P(s)$
$\qquad afficher(n)$

Maintenant l'algorithme est trivial:
$Tous\ sommets\ non\ marqués$
$\quad for\ i\ to\ nom\_sommets\ do\ P(sommet[i])$

Afin de détecter les circuits, on ajoute une deuxième marque *en_cours* à chaque sommet; elle est vraie pendant la suite d'appels récursifs sur les prédécesseurs du sommet; si P est appelé sur un sommet avec *en_cours* déjà vraie, il existe un circuit.

PLUS COMPLEXE QU'UN TRI STANDARD?

En général le temps de calcul du tri topologique est $O(E+V)$ où E et V sont les nombres d'arcs et de sommets du graphe.

Normalement $E \geq V$ et on peut dire que le temps est $O(E)$.

Mais E peut être $n(n-1)$; même la moitié dans un graphe sans circuits.

On ne peut pas espérer de faire mieux (nécessité de prendre en compte tous les arcs)

Donc une conclusion surprenante: le tri topologique a un travail plus simple à faire qu'un tri standard (plusieurs ordres possibles pour les sorties) mais son temps de calcul est forcément supérieur.

Mais pas dans notre cas parce que E le nombre d'arcs est borné par kV où k est le nombre maximum d'attributs dans la partie droite d'une règle de calcul d'un attribut.

AUTRES APPLICATION DU TRI TOPOLOGIQUE

- Planification des tâches

- Programmation dynamique irrégulière

Souvent l'algorithme de tri topologique est caché dans un algorithme récursif (on décide l'ordre dans lequel il faut faire les calculs au fur et à mesure qu'on les fait.)

FAUT-IL LE FAIRE?

Le calcul de l'ordre de calcul des attributs par le tri topologique, est-il la bonne façon de procéder?

Il permet au concepteur d'une grammaire attribuée de définir et utiliser une grammaire qui, pour certains programmes va échouer!

On préfère que tout programme construit selon les règles de la grammaire soit capable d'être compilé (on ne peut pas empêcher l'utilisateur d'écrire des programmes qui plantent à l'exécution)

Donc, il serait meilleur de tester les grammaires pour garantir qu'elles ne peuvent pas générer des programmes avec boucles dans le calcul des attributs, plutôt que tester les programmes après les avoir analysés.

SECURITE DE CALCUL DES ATTRIBUTS

Grammaires non-circulaires

Une grammaire est non-circulaire ssi pour tout programme dérivé de la grammaire il existe un ordre faisable de calcul des attributs.

C'est-à-dire qu'aucun programme dérivé de la grammaire ne possède un circuit dans son graphe des dépendances des attributs.

Il suffit de tester à chaque sommet de l'arbre syntaxique qu'il n'y a pas de circuit pour lequel ce sommet est le plus haut; mais comment le faire pour tous les programmes possibles?

ALGORITHME DE DECISION DE CIRCULARITE

Page 371(?334) du dragon.

Idée: pour chaque symbole de la grammaire construire un graphe des dépendances entre ses attributs (dépendances capables d'être produites par le sous-arbre pendant du sommet).

On déduit de nouvelles dépendances du graphe D_p des dépendances évidentes d'une production p et les graphes des symboles à droite de p; un chemin dans l'union de ces graphes montre une dépendance entre les attributs du symbole à gauche.

On fait ça pour toute production et continue tant que de nouvelles dépendances sont trouvées.

MAIS c'est trop pessimiste!

Si on a 2 productions avec S à gauche et l'une des deux peut donner une dépendance de l'attribut a de b et l'autre peut donner une dépendance de b de c, ce n'est pas forcément vrai que a peut dépendre de c.

Solution: Pour chaque symbole stocker un ensemble de graphes correspondant à chaque sous-ensemble des dépendances qui est capable d'exister simultanément; pour la production p considérer chaque combinaison possible de D_p et un des graphes de chaque symbole à droite.

TEMPS DE CALCUL

Exponentiel par rapport à la taille de la grammaire parce que le nombre de sous-ensembles des dépendances est exponentiel.

Et on sait (résultat de la théorie de la complexité) que c'est impossible de réduire ce temps à un niveau raisonnable (polynomial).

UN COMPROMIS
Les grammaires fortement non-circulaires

Celles pour lesquelles on peut associer un seul graphe à chaque symbole de façon que l'algorithme ne trouve pas de changement.

ALGORITHME DE DECISION DE NON-CIRCULARITE FORTE

Le même algorithme sauf que, au lieu de rajouter un nouveau graphe à $\mathcal{F}(X)$, on rajoute les nouveaux arcs au seul graphe $\mathcal{F}(X)$.

TEMPS DE CALCUL

Polynomial parce que chaque itération sauf la dernière ajoute une dépendance et le nombre de dépendances possibles est polynomial.

ORDRE DE CALCUL

Dès qu'on sait que la grammaire est (fortement) non-circulaire, on peut toujours calculer les attributs, sans faire le tri topologique, par un calcul simple récursif muni d'un système qui stocke les attributs calculés et ne les calcule jamais une deuxième fois.
GAG n'accepte qu'un sous-ensemble des grammaires fortement non circulaires et essaie de trouver un ordre de calcul qui permet d'économiser l'espace mémoire utilisée (utiliser une pile pour les valeurs sur un chemin de l'arbre si possible ...)