L'analyse des mots composés allemands

Martin Simon Ulmann
Département de linguistique - LATL
Université de Genève

Juin 1994

Part I

Les noms composés

1 Le fonctionnement et les problèmes de l'analyse de Thurnherr

L'analyse des mots composés allemand a déjà été traitée par Thurnherr [Thurnherr 1993, ch. 4, 17ff]. Il présente trois algorithmes, dont il choisit le deuxième qui se base sur une recherche séquentielle. Le troisième algorithme est inutilisable, parce qu'il présuppose un fichier de mots inversés. Il prétend que le premier algorithme, qui cherche lettre par lettre, est "très lent" et "demande un nombre élevé d'accès directs au fichier" qui pourrait théoriquement être "neuf fois le nombre de lettres présentes dans le mot à décomposer" ([Thurnherr 1993, ch. 4.4.1, 20]). L'algorithme choisi définit "une fourchette de recherche dans le fichier de mots" [Thurnherr 1993, ch. 4.6.1, 23] c'est-à-dire, on definit une intervalle de recherche, que l'on traverse séquentiellement . Pour *Haus*, par exemple, l'intervalle est *Ha-Hb*. Dans le lexique allmand de mots, cet intervalle contient actuellement 1247 mots et sa taille est 63250 octets.

Il paraît douteux que "la vitesse de cet algorithme est . . . plus élevée que celle de l'algorithme 1" [Thurnherr 1993, ch. 4.4.2, 20]. De plus, on ne peut pas présupposer, que la lecture séquentielle est plus vite d'un facteur considérable, car cela dépend beaucoup de la structure des données et du système d'exploitation. On a un fichier indexé avec trois clés. Il n'est pas du tout évident que le fichier est trié selon le premier clé. De plus, l'utilisation du "caché" peut influencer énormément la vitesse d'accès.

Les tests montrent, que l'algorithme actuel n'est pas suffisamment rapide. Pour le mot *Haus-eingänge*, par exemple, le temps mesuré pour le nom composé (à peu près 40 secondes) est plus que trois cents fois (!) la somme du temps mesuré pour les deux noms simples *Haus* et *Eingänge*. Si on adoptait la vue de Thurnherr que "le maximum d'accès directs est neuf fois le nombre de lettres présentes dans le mot à décomposer" [Thurnherr 1993, ch. 4.4.1, 20], on recevrait à peu près six fois plus vite le résultat avec l'algorithme de recherche lettre par lettre pour notre exemple *Hauseingänge* qu'avec l'algorithme de recherche séquentielle: Si on auvait douze lettres, cela voudrait dire que l'on aurait $9 * 12 = 108$ accès directs. Pour deux accès direct, on a utilisé a peu près $\frac{1}{320}$ du temps pour chercher le mot composé, cela veut dire

que l'on pourrait avoir environ 640 accès directs dans un même temps que l'analyse du mot composé a mis. Le facteur d'accélération est donc le nombre d' accès directs correspondant à la recherche séquentielle divisé par le nombre des accès directs nécessaires selon le prémier algorithme: $\frac{640}{108} \approx 6$ (6 fois plus vite). Toutefois, cela ne serait aussi pas suffisamment rapide, si on prennait en considération le fait que l'analyseur a mis une quarantaine de secondes pour l'analyse. Pour un analyseur de recherche lettre par lettre, cela donnerait $\frac{40}{6} \approx 6.6$ secondes.

Malheureusement, ce calcul ne serait correcte que si l'on échouait dans la recherche du mot (aucun mot lexical ne correspond au début du mot composé), parce qu'il calcule en effet le nombre d'accès pour trouver le premier mot d'un mot composé, mais on essayerait de décomposer aussi le reste du mot. Si on adopte l'hypothèse que l'on trouve dans le lexique toutes les combinations de caractère possibles (lexique complet), on recevrait le résultat suivant avec cet algorithme de retour arrière (backtracking):

$$\sum_{i=1}^{n} 9^{n-i+1} * i$$

n est le nombre de lettres du mot composé. La puissance de $n - i + 1$ vient du retour arrière, parce que cet algorithme n'empêche pas le réanalyse du même reste de mot composé. Bien entendu, notre hypotèse n'est pas réaliste; Toutefois, cette formule indique une croissance exponentielle, ce qui n'est absolument pas acceptable. La complexité d'un algorithme devrait être en $\mathcal{P}$, c'est-à-dire, polynomial, avec préférence d'un ordre $O(n^2)$ maximal.

Il est donc désirable que

1. le nombre des accès (direct ou séquenciel) dépend de la longueur du mot au lieu de la taille du lexique, parce qu'un lexique étendu est important pour l'analyse générale de la langue naturelle.
2. le nombre des accès est polynomial (d'un ordre carré $O(n^2)$ ou moins) par rapport à la longueur du mot.

Les autres problèmes de l'algorithme choisi sont: la reconnaissance des mots agrammaticaux, plusieurs analyses pour des mots non-ambigus et des règles assez compliquées.

En ce qui concerne les mots agrammaticaux, les réstrictions dcombinatoires devraient être plus strictes. Par exemple, il est difficile d'imaginer un mot qui se compose d'un superlatif et d'un substantif (ex.: **Wunderbarstetag*). Bien qu'on puisse éliminer quelques classes de mots agrammaticaux, il est douteux que l'on arrive à exclure tous les mots agrammaticaux.

Le problème du nombre d'analyses pour un mot non-ambigue apparaît surtout dans les mots à plusieurs composants.

(1)a. Bauchtanzschule

b. Bauch-tanzschule

c. Bauchtanz-schule

d. Bauch-tanz-schule

L'exemple (1) montre qu'un mot à trois composants produit trois analyses, si le lexique contient les composants et également les mot composés de deux composants; dans notre exemple, le lexique contient *Bauch*, *Tanz*, *Schule*, *Bauchtanz*, *Tanzschule*. Les deux analyses

(1b) et (1c) sont à la limite acceptables, parce qu'il est difficile à decider si une composition branche à gauche (figure 1) ou à droit (figure 2), tandis que l'analyse (1d) n'est pas acceptable. Selon Ortner, 70% des mots composés à trois composents branchent à gauche et seulement 30% branchent à droite, mais il existe aussi des mots composés de plus que trois composants [Ortner and Müller-Bollhagen 1991, 16, 18].

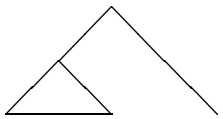

Figure 1: branche à gauche

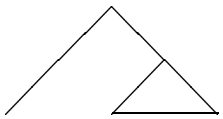

Figure 2: branche à droite

Le dernier problème mentionné ci-dessus est la difficulté des règles. Thurnherr propose un système de neuf règles pour déterminer les formes possibles d'un determinatif ([Thurnherr 1993, ch. 4.2, 19]. D'une part, il y a des règles qui ne sont pertinentes que pour un mot ou quelques mots bien connus (règles $f_6 - f_8$), d'autre part, les règles ne sont pas complètes; pour *Schwan-en-gesang*, par exemple, une règle $f_9 = e(A, en)$ devrait être utilisée (la fréquence d'application de cette règle est $9.7\% * 0.03 = 0.29\%$ (c.f. [Ortner and Müller-Bollhagen 1991, 69, 90]); [1] cette règle serait probablement plus souvent utilisée que la règle $f_3 = e(A, e)$, qui ne paraît pas être productive.[2]

2 Vers une analyse plus efficace

2.1 Stratégie d'analyse

La stratégie de l'algorithme de recherche séquentielle n'est pas satisfaisante. Les calculations montrent que l'algorithme retour arrière de recherche lettre par lettre n'est pas satisfaisante non plus. Il faut un algorithme non-retour arrière, dont la complexité dépend principalement de la longueur du mot. Pour éliminer le retour arrière, on doit observer les essais de décomposition déjà faits. L'information nécessaire est de savoir si on a essayé de décomposer le reste du mot à partir de la i-ième lettre. Si tel est le cas, on veut aussi savoir si la décomposition est possible, et si tel est aussi le cas, on veut connaître les possibilités de décomposition.

[1] Seulement les cas où aucune analyse correcte ne serait produite par les règles données par Thurnherr sont pris en considération.

[2] *Mausefalle* est probablement un cas unique, qui doit être répértorié dans le lexique (ex.: *Mäusemahlzeit*). *Tage-buch*, au contraire, peut se composer de *Tage* (nominatif pluriel) et *Buch*.

Pour réduire le domaine de la recherche, on pourrait définir le mot minimal (le début de la recherche) de la même façon que Thurnherr l'a fait pour la fourchette (c.f. [Thurnherr 1993, ch. 4.6.1, 23].

2.2 La forme du determinatif et 'Fuge'

Pour que l' analyse soit efficace, le nombre de fonctions appliquées doit être minimal. Le procédé est l'inverse de celui utilisé par l'algorithme séquentiel; on a la forme d'un substantive comprenant sa 'Fuge', puis on cherche les mots lexicaux dont ce substantif peut être dérivé. w_i^j dénote le sous-mot, qui commence par la lettre i et qui contient j lettres ($0 \leq i < n$, $0 < j \leq n - i$). m dénote la longueur du mot à découper ($m \geq 2$). Pour déterminer les mots lexicaux possibles, il y a seulement deux régles:

(2)a. $S_m \longrightarrow w_0^m$

b. $S_m \longrightarrow w_0^m e$ if $w_m^1 \neq e$

La deuxième règle correspond à la règle $f_4 = s(A, e)$.[3]

Si un mot lexical existe pour un certain m, il faut déterminer le site ou les sites de découpage, c'est-à-dire le début du mot suivant.

(3)a. $E_m \longrightarrow m$

b. $E_m \longrightarrow m + 1$ if $w_m^1 = s$

c. $E_m \longrightarrow m + 1$ if $w_m^1 = e$

d. $E_m \longrightarrow m + 2$ if $w_m^2 = es$

e. $E_m \longrightarrow m + 2$ if $w_m^2 = en$

(3a) correspond aux règles $f_0(A) = A$ et $f_4(A) = s(A, e)$. (3b) correspond aux règles $f_1(A) = e(A, s)$ et $f_5(A) = e(s(A, e), s)$. (3c) correspond à la règle $f_3(A) = e(A, e)$ et peut être supprimer. (3d) correspond à la règle $f_2(A) = e(A, es)$ et (3e) correspond à la règle $f_9(A) = e(A, en)$. Pour les règles qui ne sont pertinentes que pour un petit nombre de mots connus ($f_6 - f_8$), une comparaison directe est une solution possible.

Le calcul de complexité maximale théorique porte sur un lexique complet (un lexique qui contient tous les mots possibles d'une certaine longueur maximale) alors que le mot ne contient aucun e: [4]

$$2(n - 1 + \sum_{i=1}^{n-3} i) = 2n - 2 + (n - 2)(n - 3) = n^2 - 3n + 4$$

On voit que cet algorithme est d'ordre carré $O(n^2)$.

[3] c.f. aussi [Ortner and Müller-Bollhagen 1991, 71]

[4] Si le mot contient un ou plusieurs e, l'algorithme est plus rapide

2.3 Une solution contre toutes les solutions

Une question principale est de savoir, si l'algorithme doit fournir une seule analyse ou toutes les analyses possibles. D'une part, il existe de vrais ambiguïtés comme Thurnherr le montre avec son exemple *Spieler-folge* contre *Spiel-erfolge* [Thurnherr 1993, ch. 4.5, 22]. D'autre part, il y a des différentes possibilités pour analyser un terme composé de plusieurs mots simples ou composés.

(4)a. Bürgerkriegsschuldspruch

b. Bürgerkriegs-schuldspruch

c. Bürgerkriegs-schuld-spruch

d. Bürger-kriegsschuld-spruch

e. Bürger-kriegs-schuldspruch

f. Bürger-kriegs-schuld-spruch

Pour le mot en (4a), cinq analyses sont possibles à la condition que tous les mots simples et les mots composés de deux mots se trouvent dans le lexique. Cependent, il n'existe pas de vrais ambiguïtés. Une seule analyse est suffisante, car pour un mot composé qui est dans le lexique, on ne reçoit pas plusieurs analyses. Le mot donné en (5) est dans le lexique, et, pour cela, on renonce à l'analyser comme un mot composé, bien que cela soit possible (*Laden-diebstahl*, *Ladendieb-stahl*, *Laden-dieb-stahl*).

(5) Ladendiebstahl

De ce point de vue, la meilleure analyse de (4a) est (4b), parce qu'il n' y a que deux composants. Les analyses (4c,) (4e) et (4f) ne sont pas acceptables, parce qu'elles sont des analyses de compositions de l'analyse (4b). L'analyse (4d) n'est pas vraiment désirable, si on a déjà une analyse. Or, il est difficile de déterminer la différence entre deux analyses qui sont vraiment ambiguës, en (4b) et (4d). Il faudrait une analyse sémantique pour savoir quelle analyse est 'la correcte'.

Se borner à une seule analyse peut être justifié par le fait que les mots composés vraiment ambigus sont très rares. En ce cas, l'analyse devrait être correcte et aucune paire de ces composants ne devrait être un mot composé lexical.

2.4 L'amélioration de la strategie pour une solution

Si on se borne à une solution unique, la stratégie peut être améliorée. Du fait qu'une analyse doit donner le nombre de constituent minimale, l'idée de commencer par un mot minimal est fausse. On devrait commencer par un mot maximal w_0^m, sur lequel on applique $S(m)$ pour obtenir un ensemble de mots poteniels, que l'on doit rechercher au lexique. Si on échoue, on décrémente m. Si m est le mot minimal, on n'a pas trouvé de solution, et le mot w n'est ni dans le lexique, ni décomposable.

Pour trouver ce mot maximal w_0^m, on cherche le mot $w_{\leq}$, qui est égal à ou plus petit (alphabéthiquement) que le mot composé. (Si on trouve un mot qui est égal au mot composé, le mot recherché est alors dans le lexique, donc il est inutile d'essayer de le décomposer.) On

cherche aussi le mot séquentiellement suivant $w_>$; il est plus grand (alphabétiquement) que w. [5] Pour obtenir le mot maximal, on compare w avec $w_\leq$ et $w_>$ lettre par lettre, jusqu'à ce que l'on trouve une différence:

(6)a. $m = \max(m_\leq, m>)$

b. $m_\leq = \{\forall j < m_\leq : w[j] = w_\leq[j] \wedge w[m_\leq] \neq w_\leq[m_\leq]\}$

c. $m_> = \{\forall j < m_> : w[j] = w_>[j] \wedge w[m_>] \neq w_>[m_>]\}$

Cette stratégie peut aussi être appliqué dans le cas où l'on cherche toutes les solutions possibles, mais on doit continuer jusqu'au mot minimal; on diminue le nombre d'accès aussi, parce qu'on n'a pas besoin d'aller jusqu'à la fin du mot.

Une autre possibilité pour augmenter l'efficacité est d'appliquer des contraintes plus fortes, des heuristiques et des statistiques.

Les contraintes pour la règle (2b) peut être plus strictes. Intuitivement, on dirait que une e doit suivre un consonne pour être supprimé. [Ortner and Müller-Bollhagen 1991, 71, 74f] donne une liste de mots, qui perdent souvent ou toujours le e dans la composition:

(7)a. Adress(e), Eck(e), End(e), Farb(e), Fähr(e), Filial(e), Grenz(e), Hitz(e), Kron(e), Lymph(e), Mitt(e), Pfarr(e), Sach(e), Schul(e), Sprach(e), Violin(e), Woll(e)

b. Gebirg(e)+s, Herberg(e)+s, Hilf(e)+s, Geschicht(e)+s, Miet(e)+s

c. *b,d,f,g,h,k,l,n,r,s,t,z*

(7b) donne la dernière consonne du mot dans la composition. Si cette liste est complète, la règle (2b) peut être réformuler en (8). Le nombre d'accès peut diminuer30% à peu près (une estimation).

(8) $S_m \longrightarrow w_0^m e \qquad$ if $w_m^1 \neq e \wedge w_{m-1}^1 \in \{b, d, f, g, h, k, l, n, r, s, t, z\}$

Pour déterminer E_m, des règles pour les 'Fugen' peuvent être utilisées ([Ortner and Müller-Bollhagen 199 ch. 5.5, 68ff]). Les règles utilisées pour la génération sont un sous-ensemble de ces règles (c.f. [Ulmann 1994, ch. 3.4.13, 28]). Des heuristiques peuvent aussi aider à diminuer E_m.

Une méthode statistique peut être utilisée pour trouver le déterminatif. Si on avait des données statistiques sur les terminaisons des mots allemand, on pourrait commencer la recherche du déterminatif par un mot qui a une terminaison typique. Malheuersement, c'est difficile de trouver ce genre de statistiques et l'effort de programmation serait rélativement grand.

3 Discussion des algorithmes

Dans ce chapitre, deux algorithmes seront décrits, un qui trouve toutes les solutions et un qui ne donne qu'une seule solution.

[5] 'plus grand' ou 'plus petit' n'ont rien à voir avec la longueur du mot, mais définiissent le mot précédent ou suivant dans le lexique.

3.1 Un algorithme qui trouve toutes les solutions possibles

L'algorithme présente à peu près les éléments suivants:

1. chercher le mot maximal w_0^m
2. chercher le mot minimal w_0^l
3. pour tous les $i : l <= i < m$
   (a) cherche les mots S_i dans le lexique.
   (b) si un ou plusieurs des mots S_i qui remplissent les conditions d'un déterminatif sont trouvés, alors on applique E_i qui donne des points de découpage lesquels doivent être sauvegardés. Les mots trouvés aussi doivent être sauvegardés.
4. chercher dans la liste un point de découpage, découper et appeler la procédure récursivement. La liste des points de découpage doit être unique et absolue.

On peut eliminer le non-déterminisme et la récursivité finale dans cet algorithme, et on obtient l'algorithme suivant (une esquisse): [6]

```
PROCEDURE GetCompoundList
    (    w          : ARRAY OF CHAR; (* compound word *)
     VAR comp_list : List;
     VAR head_list : List) : BOOLEAN;
VAR
  begin,            (* begin of the determinatif in the word w *)
  w_size,           (* size of word w *)
  i, j,
  w_min,            (* minimal length of word for search *)
  w_max : CARDINAL; (* maximal length of word for search *)
BEGIN
  begin  := 0;
  w_size := Legnth(w);
  CreateChart(w_size);
  WHILE begin < w_size DO
    SubString(w, begin, w_size-begin, cur_w);
    w_min := SearchMinWord(cur_w);
    w_max := SearchMaxWord(cur_w);
    FOR i := w_min TO w_max DO
      IF ApplyS(cur_w, i, i = w_size, cur_list_S) THEN (* calculate S(m) *)
        ApplyE(cur_w, i , cur_list_E); (* calculate E(m) *)
        InsertIntoChart(cur_list_S, cur_list_E, begin);
      END;
    END;
    INC(begin)
    WHILE (j + begin < w_size) AND ChartEmptyPosition(j + begin) DO
      INC(j);
    END;
```

[6] La définition de la procédure *GetCompoundList* est identique avec celle de [Thurnherr 1993, ch 4.5, 22]

```
    INC(begin, j);
  END;
  BFSChart(comp_list, head_list); (* BFS algorithm plus elimination of
                                     false ambiguities *)
  TerminateChart();
  RETURN NOT EmptyList(headList);
END GetCompoundList;
```

Pour mémoriser les résultats intermédiares, un 'chart' est approprié.

```
TYPE
  Node  = RECORD
            outgoing_edges : List OF Edges;
            pathes         : List OF List OF CARDINAL;
            (* c.f. elimination of ambiguities, BFSChart *)
          END;

  Chart = ARRAY [0..MaxWordLength] OF nodePtr;
  Edge  = RECORD
            begin    : CARDINAL;
            end      : CARDINAL;
            lex_item : LexicalItemPtr;
          END;
```

Pour notre exemple *Hauseingänge*, on obtient un graphe comme indiqué dans la figure 3. Pour *Hau*, l'application de S_3 donne *Haue* en plus de*Hau*, qui est un mot lexical. Si E_3 est appliqué, on a les deux solutions *Hau* et *Haus*. Si on implémente la contrainte (8) pour S_4, cette analyse, qui est fausse, est exclue. Pour *Haus*, S_4 donne *Haus* et l'application de E_4 a comme résultat *Haus, Hause*. Les seules positions de découpage sont 3, 4 et 5. À partir de la position 3, on ne trouve que *sein*, qui est soit un verbe, soit un pronom possessif. Ici, l'analyse échoue. À partir de la position 4, il trouve *Ei* et *Eingänge*, qui peut faire partie d'une analyse, donc doit être inclu dans le 'chart'. L'essai d'analyse à partir de la position 5 et de la position 6 échoue. Les analyses possibles sont tous les chemins, qui commencent à la position 0 et se terminent à la dernière position du 'chart' (position 12). Dans notre cas, cela donne *Haus-eingänge*, et sans la contrainte (8), aussi $e(s(Haue, e), s)$-*eingänge*

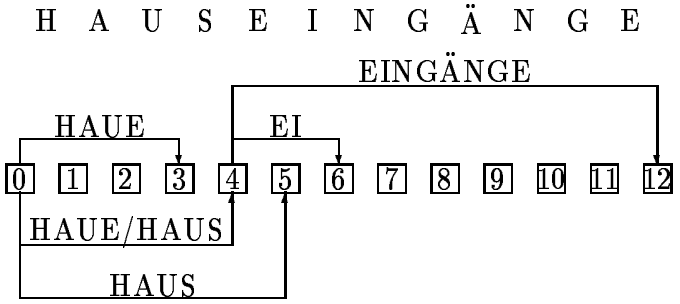

Figure 3: exemple d'un 'chart'

Pour chercher tous les chemins à travers un graphe, on peut appliquer la stratégie BFS (Breath-First-Search, c.f. [Biggs 1985, 197]). Or, du fait que les ambiguïtés fausses ne doivent pas faire partie du résultat, l'algorithme est modifié comme suit:

```
FOR i := 0 TO EndChart-1 DO
  FOR j := 1 TO Size(chart[i]^.outgoing_edges) DO
    cur_edge := Findith(chart[i]^.outgoing_edges,j);
    FOR k := 1 TO Size(chart[i]^.pathes) DO
      cur_path := Findith(chart[i]^.pathes);
      AddToTail(cur_path, i);
      IF NOT BeginningOfPathInPathList(cur_path, cur_edge^.end^.pathes) THEN
        AddToTail(cur_edge^.end^.pathes, cur_path);
      END;
    END;
  END;
END;
```

La figure 4 montre le fonctionnement de cet algorithme. A partir du noeud 0, il y a deux arrêtes. À leurs bout le chemin [0] est inséré dans leurs listes de chemins. Pour le noeud 5, on a une arrête qui termine au noeud 9, mais il n'est pas mis dans sa liste de chemins, parce qu'elle contient le chemin [0], qui est le début du sentier actuel [0,5]. Ce chemin est seulement mis dans la liste des chemins des noeuds 14 et 21. ainsi de suite, cela continue jusqu'à ce qu'on arrive au dernier noeud. Sa liste de sentiers contient toutes les analyses, qui ne sont pas identifiées comme fausse ambiguïté (cela veut dire $\forall a \in \{\text{arrête}\} | \nexists s \in \{\text{sentier avec deux ou plusieurs arrêtes}\} : a.begin = s.begin \wedge a.end = s.end$).

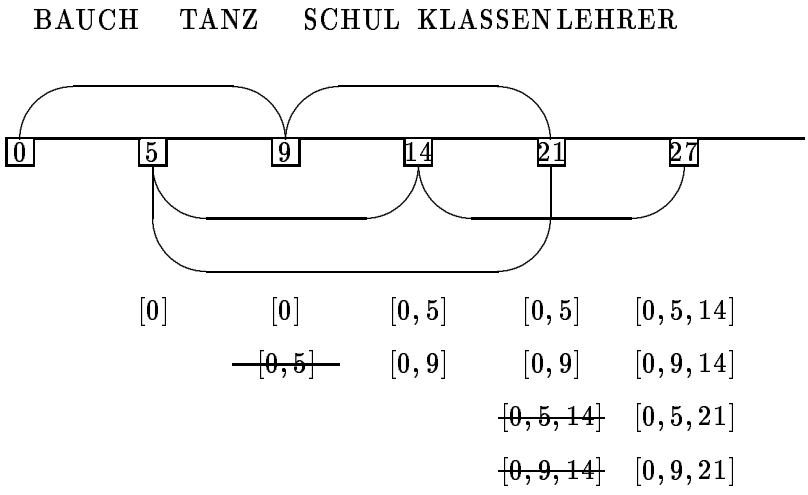

Figure 4: l'algorithme d'élimination des ambiguïtés fausses

3.2 Un algorithme qui se borne à une solution

Cet algorithme est récursif, et il sert aussi à rechercher des mots simples. De plus, il n'a pas besoin d'un 'chart'.

```
PROCEDURE GetCompoundList
    (    w         : ARRAY OF CHAR;
     VAR comp_list : List;
```

```
    VAR head      : LexicalItemPtr) : BOOLEAN;
VAR
  w_size,
  w_min,
  w_max : CARDINAL;
BEGIN
  w_size := Length(w);
  w_min  := SearchMinWord(cur_w);
  w_max  := SearchMaxWord(cur_w, found, lex_item);
  IF found THEN (* maximal word is word w => w is not a compound *)
    head := lex_item;
    RETURN TRUE;
  ELSE
    LOOP
      IF w_max <= w_min THEN  (* no solution found *)
        RETURN FALSE;
      ELSIF ApplyS(w, w_max, cur_list_S) THEN (* calculate S(m) *)
        ApplyE(w, w_max , cur_list_E); (* calculate E(m) *)
        IF SetHead(cur_list_E) THEN
          REPEAT
            cur := Retrieve(cur_list_E);
            IF NOT (cur IN CutPosition) THEN
              INCL(CutPosition, cur); (* eliminate backtracking *)
              IF GetCompoundList(w, comp_list, head_list) THEN (* recursivity *)
                MultiplyList(cur_list_S, comp_list);
                (* each the elements of cur_list_S is added/combined
                   to all lists of comp_list *)
                RETURN TRUE; (* compound w found *)
              END;
            END;
          UNTIL NOT FindNext(cur_list_E)
        END;
      END;
      DEC(w_max);
    END;
  END;
END GetCompoundList;
```

4 L'implémentation

L'algorithme qui trouve toutes les solutions a été implémentées. La recherche du mot maximal n'a pas pu réalisée comme décrit en section 2.4 (parce qu'il faudrait un clé descendent pour chercher $w_{\leq}$, le mot précédent ou égal du mot cherché). Au lieu de cela, le mot maximal est déterminé comme la longueur du mot. La fonction S_m a été implémentée selon (8) et la fonction E_m selon (4). Pour garder la compatibilité avec l'interface de Thurnherr, les éléments de la liste des chemins (le champ 'pathes' de 'Node') ne sont pas des numéros (cardinal) mais des éléments de la structure suivante:

```
Element = RECORD
            lex_index : CARDINAL;
            string    : WordString;
            cat       : Category;
          END;
```

Cette structure a l'avantage que le champ 'pathes' du dernier noeud du 'chart' correspond au paramètre 'comp_list' et la liste 'head_list' peut être remplie au moment où on arrive au dernier noeud.

Les mésures fournissent des résultats étonnants:

	Thurnherr	la nouvelle algorithme	accéleration
machine 1	13'55"	33"	25
machine 2	3'15"	13"	15
machine 3	1'50"	11"	10

Le temps mésuré correspond à la recherche et analyse d'une vingtaine de mots composés. Les résultats du test sur la machine 2 sont les suivants:

mot	CPU Thurnherr	CPU la nouvelle algorithme
Speichergeräte.	280	120
die Systemeinheit.	4940	150
Betriebssystemen.	3060	140
Bandlaufwerke.	1580	140
Peripheriegeräte.	540	150
einem Netzwerk.	580	100
der Zeichenabstand.	520	120
die Fettschrift.	1030	130
Benutzereingaben.	4860	130
der Handbuchtitel.	1510	140
ein Leerzeichen.	1270	100
Hochleistungssysteme.	46460	230
Systemeigenschaften.	230	210
Bedienungselemente.	19050	220
Kontrollanzeigen.	1440	140
Schnittstellen.	3410	170
die Systemarchitektur.	100	170
die Rechnerumgebung.	14350	180
Anzeigegeräte.	1800	90
Einsatzmöglichkeiten.	1320	280
Plattenspeicher.	2050	180
die Rechenleistung.	1780	170
Leistungsvorteile.	21950	220
moyenne μ	5831	160
$\frac{\sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}}}{\mu}$	1.79	0.478
minimum	100	90
maximum	46460	280
$\frac{\text{maximum}}{\text{minimum}}$	464.6	3.1

On voit que le temps 'CPU' utilisé par la nouvelle algorthme est 1/36 de celui utiisé par l'algorithme de Thurnherr. De plus, la différence de temps entre les mots est beaucoup plus grand pour l'algorithme de Thurnherr.

5 Conclusion

Les algorithmes proposés par Thurnherr sont définitivement trop lents. En cherchant une algorithme plus rapide, la condition laquelle le nombre d'accès sur disque doit dépendre de la longueur du mot a été élaborée. Deux algorithmes différentes ont été considérées: une qui cherche toutes les solutions correctes et un qui se borne à une solution correcte. Le premier algorithme a le désavantage d'être plus lent que le deuxième, mais il a l'avantage de trouver toutes les solutions. En ce qui concerne la performance de ces algorithmes, on peut dire, que le nombre d'accès direct est plus petit que le quarré de nombre de lettres (n^2). Les tests de l'algorithme qui cherche toutes les solutions montrent, qu'elle est plus que dix fois plus rapide que celle de Thurnherr et même au pire cas, (p. ex. si l'analyse échoue), on reçoit résultat assez vite. Il paraît que l'on a trouvé une solution satisfaisante pour l'analyse des mots composés allemands.

Part II

Les numéros composés

6 Les numéros cardinaux

En allemand, les numéros composés sont des mots composés (sans traits-d'union). Pour tester, s'il s'agit d'un numéro composé, un automate fini ou une grammaire indépendente du contexte.

6.1 Les mots

L'alphabeth terminal T contient les symboles suivants:

- null
- ein, zwei, drei ,vier, fünf, sechs, sieben, acht, neun
- zehn, elf, zwölf, dreizehn, vierzehn, fünfzehn, sechzehn, siebzehn, achtzehn, neunzehn
- zwanzig, dreissig, vierzig, fünfzig, sechzig, siebzig, achtzig, neunzig
- hundert, tausend
- und

6.2 La classification et la fonction

Pour décrire plus facilement la grammaire, des variable sont definis et chaque variable représente un élément de l'alphabeth terminal T:

- $\alpha \in \{$ ein ... neun $\}$
- $\beta \in \{$ zehn ... neunzehn, zwanzig ... neunzig $\}$
- $\gamma :=$ hundert
- $\delta :=$ tausend
- $\epsilon :=$ und
- $\zeta :=$ null
- $\eta \in$ zwanzig ... neunzig $\}$
- $\theta :=$ hundert
- $\kappa :=$ tausend

6.3 La grammaire

L'alphabeth non-terminal N contient les symboles suivants:

(9) A, B, C, D, E

E est le symbole pour commencer.

La grammaire se compose des productions suivantes:

- $A \longrightarrow \beta$
- $A \longrightarrow \alpha$
  $A \longrightarrow \alpha\epsilon\eta$

- $B \longrightarrow \theta$
- $B \longrightarrow \alpha\gamma$
- $C \longrightarrow A$
- $C \longrightarrow B$
  $C \longrightarrow B\epsilon\alpha$
  $C \longrightarrow BA$
- $D \longrightarrow \kappa$
- $D \longrightarrow C\delta$
- $E \longrightarrow C$
- $E \longrightarrow D$
  $E \longrightarrow D\epsilon A$
  $E \longrightarrow DC$

6.4 L'interprétation des numéros composés

Pour l'analyse des numéros composés, il n'est généralement pas suffisant d'avoir un automate fini, qui rend comme résultat seulement, si un numéro est correctement composé ou non. Plutôt, on veut avoir une interprétation; Dans notre cas, le numéro en chiffres sert bien. Pour cela, une fonction de valeur v est definit:

$$v : x \rightarrow v(x)$$

Pour les numéros du alphabeth terminal $x \in T$, le valeur de $v(x)$ est le chiffre correspondant (p.ex. $v(\text{acht}) = 8$, $v(\text{siebzehn}) = 17$, $v(\text{hundert}) = 100$) et pour le symbole *und*, le valeur est zéro ($v(\text{und}) = 0$). $v(nq)$ est défini comme addition ou multiplication selon les règles suivantes:

$$\begin{aligned} v(xy) &= v(x) * v(y) \quad y \in \{\gamma, \delta\} \\ v(xy) &= v(x) + v(y) \quad y \notin \{\gamma, \delta\} \end{aligned}$$

La figure 5 illustre l'analyse de l'exemple *hundertundzweitausenddreihundertfünfundsechzig*. On voit aussi, que l'introduction de deux variables pour *hundert* (γ et θ) et *tausend* (δ et κ) était necessaire pour distinguer la multiplication et l'addition pour ces éléments

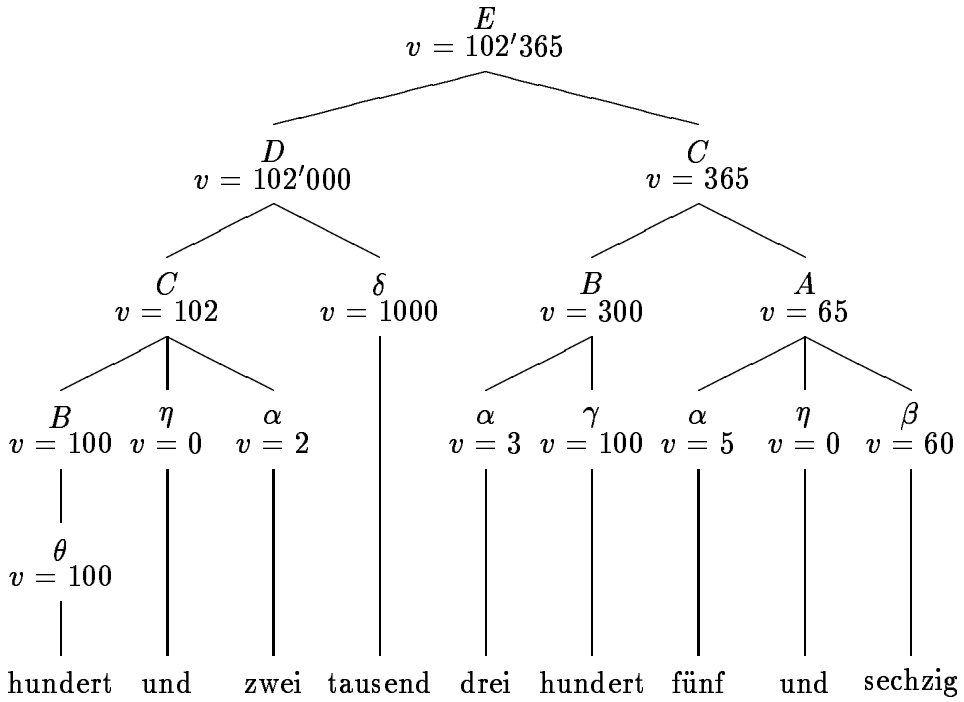

Figure 5: L'analyse de *hundertundzweitausenddreihundertfünfundsechzig*

6.5 La flexion des numéros.

On part de la supposition que tous les numéros simples (les symboles de l'alphabet terminal) sont dans le lexique. On regarde le fin du mot à partir du début du dernier composant

analysé (dans l'exemple de la figure 5, c'est 'sechzig') et le cherche au lexique. Si on le trouve, on remplace le clé par le chiffre. La plupart des numéros cardinaux n'ont qu'une seule forme. *zwei* und *drei* ont une forme génitive *zweier* et *dreier*, mais ces formes ne se conservent pas dans les numéros composés (**hundertundzweier*) et doivent être exclues. Le cas de *ein* et différent. Il peut conserver ses formes flectées et aussi le singulier dans un numéro composé (exemples (10a), (10b) et (10c)), mais aussi la forme *ein* avec pluriel est possible (exemples (10d) et (10e)) ([Drosdowski 84, para. 457, 277]).

(10)a. die Geschichte von tausendundeiner Nacht

b. tausendundein Weizenkorn

c. hundertundein Salutschuß

d. tausend[und]ein Weizenkörner

e. mit hundert[und]ein Salutschüssen

7 Les numéros ordinaux

Les numéros composés ordinaux sont à traiter de la même façon que les numéros composés cardinaux, sauf que le dernier composant a une forme différente. Ce dernier composont se compose d'un numéro cardinal et un suffix avec l'exception de *erst-*, *dritt-* et *siebt-*, qui doivent aussi être acceptées par l'automate fini, s'ils se trouvent à la fin. Cela veut dire, que l'alphabeth terminal T est élargi par ces trois symboles et le valeur est le même que celui des correspondants cardinaux (v(erst) = 1, v(dritt) = 3 et v(siebt) = 7). Parce que le reste du mot avec le dernier symbole est cherché au lexique, la forme flectée aproprieée est rendue (Les numéros cardinaux basiques sont aussi dans le lexique). et le clé est remplace par le numéro et un point (ex.: 102365.).

8 Des autres probèmes

Des autres problèmes des numéros compoés sont montrés aux exemples (11a)- (11f).

(11)a. 25jährig, das fünfundsiebzigjährige Jubiläum

b. 32armig

c. 18fach

d. 1/125, siebenundzwanzig zweihundertdreiunddreissigstel

e. Es ist fünf Uhr.

f. 35 Fass Bier.

Les exemples (12a)-(12c) sont des compositions des numéros avec des adjectives ou des suffixes, (12d) montre les fractions et (12e) et (12f) sont des exemples pour la combinaison des numéros avec des noms singuliers.

9 Conclusion

L'analyse des numéros cardinaux et ordinaux peut être réalisée par un automate fini. En même temps où on teste l'acceptibilé d'un numéro composé, le valeur (la chiffre) peut être calculé.

References

[Biggs 1985] Biggs, Norman. 1985. Discrete Mathematics. Oxford: Clarendon Press.

[Drosdowski 84] Drosdowski84, Günter et al. (ed.). 1984. *Duden* Grammatik der deutschen Gegenwartssprache. Mannheim.

[Ortner and Müller-Bollhagen 1991] Ortner, Lorelies and Elgin Müller-Bollhagen. 1991. *Deutsche Wordtbildung: Typen und Tendenzen in der Gegenwartssprache.* Berlin and New York: Walter de Gruyter.

[Thurnherr 1993] Thurnherr, Eric. 1993. "Analyse lexicographique de l'allemand." Projet de diplôme, EPFL.

[Ulmann 1994] Ulmann, Martin Simon. 1994. "Realization of a Sentence Generating System for German." Diploma Project, ETHZ.