Critical Tokenization and its Properties

Jin Guo*
National University of Singapore

Tokenization is the process of mapping sentences from character strings into strings of words. This paper sets out to study critical tokenization, a distinctive type of tokenization following the principle of maximum tokenization. The objective in this paper is to develop its mathematical description and understanding.

The main results are as follows: (1) Critical points are all and only unambiguous token boundaries for any character string on a complete dictionary; (2) Any critically tokenized word string is a minimal element in the partially ordered set of all tokenized word strings with respect to the word string cover relation; (3) Any tokenized string can be reproduced from a critically tokenized word string but not vice versa; (4) Critical tokenization forms the sound mathematical foundation for categorizing tokenization ambiguity into critical and hidden types, a precise mathematical understanding of conventional concepts like combinational and overlapping ambiguities; (5) Many important maximum tokenization variations, such as forward and backward maximum matching and shortest tokenization, are all true subclasses of critical tokenization.

It is believed that critical tokenization provides a precise mathematical description of the principle of maximum tokenization. Important implications and practical applications of critical tokenization in effective ambiguity resolution and in efficient tokenization implementation are also carefully examined.

1. Introduction

Words, and tokens in general, are the primary building blocks in almost all linguistic theories (e.g., Gazdar, Klein, Pullum, and Sag 1985; Hudson 1984) and language processing systems (e.g., Allen 1995; Grosz, Jones, and Webber 1986). Sentence, or string, **tokenization**, the process of mapping sentences from character strings to strings of words, is the initial step in natural language processing (Webster and Kit 1992).

Since in written Chinese there is no explicit word delimiter (equivalent to the blank space in written English), the problem of Chinese sentence tokenization has been the focus of considerable research efforts, and significant advancements have been made (e.g., Bai 1995; Zhang et al. 1994; Chen and Liu 1992; Chiang et al. 1992; Fan and Tsai 1988; Gan 1995; Gan, Palmer, and Lua 1996; Guo 1993; He, Xu, and Sun 1991; Huang 1989; Huang and Xia 1996; Jie 1989; Jie, Liu, and Liang 1991a, 1991b; Jin and Chen 1995; Lai et al. 1992; Li et al. 1995; Liang 1986, 1987, 1990; Liu 1986a, 1986b; Liu, Tan, and Shen 1994; Lua 1990, 1994, and 1995; Ma 1996; Nie, Jin, and Hannan 1994; Sproat and Shih 1990; Sproat et al. 1996; Sun and T'sou 1995; Sun and Huang 1996; Tung and Lee 1994; Wang, Su, and Mo 1990; Wang 1989; Wang, Wang, and Bai 1991; Wong et al. 1995; Wong et al. 1994; Wu et al. 1994; Wu and Su 1993; Yao, Zhang, and Wu 1990; Yeh and Lee 1991; Zhang, Chen, and Chen 1991).

* Institute of Systems Science, National University of Singapore, Kent Ridge, Singapore 119597; e-mail: guojin@iss.nns.sg

© 1997 Association for Computational Linguistics

The tokenization problem exists in almost all natural languages, including Japanese (Yosiyuki, Takenobu, and Hozumi 1992), Korean (Yun, Lee, and Rim 1995), German (Pachunke et al. 1992), and English (Garside, Leech, and Sampson 1987), in various media, such as continuous speech and cursive handwriting, and in numerous applications, such as translation, recognition, indexing, and proofreading.

For Chinese, sentence tokenization is still an unsolved problem, which is in part due to its overall complexity but also due to the lack of a good mathematical description and understanding of the problem. The theme in this paper is therefore to develop such a mathematical description.

In particular, this paper focuses on **critical tokenization**[1], a distinctive type of tokenization following the maximum principle. What is to be established in this paper is the notion of critical tokenization itself, together with its precise descriptions and well-proved properties.

We will prove that **critical points** are all and only unambiguous token boundaries for any character string on a complete dictionary. We will show that any **critically tokenized word string** is a minimal element in the partially ordered set of all tokenized word strings on the word string cover relation. We will also show that any tokenized string can be reproduced from a critically tokenized word string but not vice versa. In other words, critical tokenization is the most compact representation of tokenization. In addition, we will show that critical tokenization forms a sound mathematical foundation for categorizing **critical ambiguity** and **hidden ambiguity** in tokenizations, which provides a precise mathematical understanding of conventional concepts like combinational and overlapping ambiguities. Moreover, we will confirm that some important **maximum tokenization** variations, such as forward and backward maximum matching and shortest tokenization, are all subclasses of critical tokenization.

Based on a mathematical understanding of tokenization, we reported, in Guo (1997), a series of interesting findings. For instance, there exists an optimal algorithm that can identify all and only critical points, and thus all unambiguous token boundaries, in time proportional to the input character string length but independent of the size of the tokenization dictionary. Tested on a representative corpus, about 98% of the critical fragments generated are by themselves desired tokens. In other words, about 98% close-dictionary tokenization accuracy can be achieved efficiently *without disambiguation*.

Another interesting finding is that, for those critical fragments with critical ambiguities, by replacing the conventionally adopted meaning preservation criterion with the critical tokenization criterion, disagreements among (human) judges on the acceptability of a tokenization basically become non-existent. Consequently, an objective (human) analysis and annotation of all (critical) tokenizations in a corpus becomes achievable, which in turn leads to some important observations. For instance, we observed from a Chinese corpus of four million morphemes a very strong tendency to have *one tokenization per source*. Naturally, this observation suggests tokenization disambiguation strategies notably different from the mainstream best-path-finding strategy. For instance, the simple strategy of *tokenization by memorization* alone could easily exhibit critical ambiguity resolution accuracy of no less than 90%, which is notably higher than what has been achieved in the literature. Moreover, it has been observed that critical tokenization can also provide helpful guidance in identifying hidden ambiguities and in determining unregistered (unknown) tokens (Guo 1997). While these are just some of the very primitive findings, they are nevertheless promising and motivate

1 All terms mentioned here will be precisely defined later in this paper.

us to rigorously formalize the tokenization problem and to carefully explore logical consequences.

The rest of the paper is organized as follows: In Section 2, we formally define the string generation and tokenization operations that form the basis of our framework. In Section 3, we will study tokenization ambiguities and explore the concepts of critical points and critical fragments. In Section 4, we define the word string cover relation and prove it to be a partial order, define critical tokenization as the set of minimal elements of the tokenization partially ordered set, and illustrate the relationship between critical tokenization and string tokenization. Section 5 discusses the relationship between critical tokenization and various types of tokenization ambiguities, while Section 6 addresses the relationship between critical tokenization and various types of maximum tokenizations. Finally, in Sections 7 and 8, after discussing some helpful implications of critical tokenization in effective tokenization disambiguation and in efficient tokenization implementation, we suggest areas for future research and draw some conclusions.

2. Generation and Tokenization

In order to address the topic clearly and accurately, a precise and well-defined formal notation is required. What is used in this paper is primarily from elementary Boolean algebra and Formal Language Theory, which can be found in most graduate-level textbooks on discrete mathematics. This section aims at refreshing several simple terms and conventions that will be applied throughout this paper and at introducing the two new concepts of character string generation and tokenization. For the remaining basic concepts and conventions, we mainly follow Aho and Ullman (1972, Chapter 0, Mathematical Preliminaries), and Kolman and Busby (1987).

2.1 Character, Alphabet, and Character String

Definition 1

An **alphabet** $\Sigma = \{a, b, c, \ldots\}$ is a finite set of symbols. Each symbol in the alphabet is a **character**. The **alphabet size** is the number of characters in the alphabet and is denoted $|\Sigma|$. **Character strings** over an alphabet Σ are defined[2] in the following manner:

1. e is a character string over Σ. e is called the **empty character string**.
2. If S is a character string over Σ and a is a character in Σ, then Sa is a character string over Σ.
3. S' is a character string over Σ if and only if its being so follows from (1) and (2).

The **length** of a character string S is the number of characters in the string and is denoted $|S|$. A **position** in a character string is the position after a character in the string. If characters in a character string are indexed from 1 to n, then positions in the string are indexed from 0 to n, with 0 for the position before the first character and n for that after the last character.

2 This definition is adapted from Aho and Ullman (1972, 15).

Example 1
The set of 26 upper case and 26 lower case English characters forms the English alphabet $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z\}$. S = *thisishisbook* is a character string over the alphabet. Its string length is 13.

In this paper, characters are represented with small characters a, b, c, or their subscript form a_k, b_k, and c_k. The capital letter S or its expanded form $S = c_1 \ldots c_n$ is used to represent a character string. We let Σ^* denote the set containing all character strings over Σ including e, and Σ^+ denote the set of all character strings over Σ but excluding e.

2.2 Word, Dictionary, and Word String

Definition 2
Let alphabet $\Sigma = \{a, b, c, \ldots\}$ be a finite set of characters. A **dictionary** D is a set of character strings over the alphabet Σ. That is, $D = \{x, y, z, \ldots\} \subseteq \Sigma^*$. Any element in the dictionary is a **word**. The **dictionary size** is the number of words in the dictionary and is denoted $|D|$. **Word strings** over a dictionary D are defined in the following manner:

1. v is a word string over D. v is called the **empty word string**.
2. If W is a word string over D and w is a word in D, then Ww is a word string over D.
3. W' is a word string over D if and only if its being so follows from (1) and (2).

The length of a word string W is the number of words in the string and is denoted $|W|$. We let D^* denote the set containing all word strings over D, including v and let D^+ denote the set of all word strings over D but excluding v.

Example 1 (cont.)
The set $D = \{this, is, his, book\}$ is a tiny English dictionary from the English alphabet. Both *his* and *book* are words over the English alphabet. The dictionary size is 4, i.e., $|D| = 4$. *"this is his book"* is a word string. Its string length is 4.

To differentiate between *character string* and *word string*, blank spaces are added between words in word strings. For example, *"this is his book"* represents a word string of length 4 (four words concatenated) while *thisishisbook* consists of a character string of length 13 (13 characters in sequence). Slash / is sometimes used as a (hidden) word delimiter. For instance, *this/is/his/book* is an equivalent representation to *"this is his book"*.

Generally, capital letters X, Y, Z, and W, or their expanded forms such as $W = w_1 \ldots w_m$, represent word strings. Small letters x, y, z, and w, or their expanded forms such as $w = c_1 \ldots c_n$, represent both words as elements in a dictionary and character strings over an alphabet. In other words, they are both $w \in D$ and $w \in \Sigma^*$. The word string made up of the single word w alone is represented by w^1. In cases where context makes it clear, the superscript can be omitted and w is also used for representing the single word string w^1.

2.3 Character String Generation

Definition 3

Let $\Sigma = \{a, b, c, \ldots\}$ be an alphabet and $D = \{x, y, z, \ldots\}$ be a dictionary over the alphabet. The **character string generation operation** G is a *mapping* G: $D^* \rightarrow \Sigma^*$ defined as:

1. Empty word string v is mapped to empty character string e. That is, $G(v) = e$.
2. Single word string w^1 is mapped to the character string of the single word. That is, $G(w^1) = w$.
3. If W is a word string over dictionary D and w is a word in D, then, the word string Ww is mapped to the concatenation of character string $G(W)$ and $G(w)$. That is, $G(Ww) = G(W)G(w)$.

$G(W)$ is said to be the **generated character string** of the word string W from dictionary D.

Note that the character string generation operation G is a *homomorphism* (Aho and Ullman 1972, 17) with property $G(w^1) = w$.

Example 1 (cont.)

The character string *thisishisbook* is the generated character string of the word string *"this is his book"*. That is, $G($*"this is his book"*$)$ = *thisishisbook*.

2.4 Character String Tokenization

Definition 4

The **character string tokenization operation** T is a *mapping* T_D: $\Sigma^* \rightarrow 2^{D^*}$ defined as: if S is a character string in Σ^*, then $T_D(S)$ is the set of dictionary word strings mapped by the character string generation operation G to the character string S. That is, $T_D(S) = \{W | G(W) = S,\ W \in D^*\}$. Any word string W in $T_D(S)$ is a **tokenized word string**, or simply a **tokenization**, of the character string S.

Sometimes the character string tokenization operation is emphasized as the **exhaustive tokenization operation** or **ET operation** for short. In addition, the tokenized word string or tokenization is emphasized as the **exhaustively tokenized word string** or **exhaustive tokenization** or **ET tokenization** for short.

Note that the character string tokenization operation T_D is the *inverse homomorphism* (Aho and Ullman 1972, 18) of the character string generation operation G.

Example 1 (cont.)

Given character string *thisishisbook*, for the tiny English dictionary $D = \{$*this, is, his, book*$\}$, there is $T_D($*thisishisbook*$) = \{$*"this is his book"*$\}$. In other words, the word string *"this is his book"* is the only tokenization over the dictionary D.

Given dictionary $D' = \{$*th, this, is, his, book*$\}$, in which *th* is also a word, there is $T_{D'}($*thisishisbook*$) = \{$*"th is is his book", "this is his book"*$\}$. In other words, the character string has two tokenizations over the dictionary D'.

Example 2

For character string *fundsand* and the tiny English dictionary $D = \{$*fund, funds, and, sand*$\}$, there is $T_D($*fundsand*$) = \{$*"funds and", "fund sand"*$\}$. In other words, both *"funds and"* and *"fund sand"* are tokenizations of character string *fundsand*.

2.5 Discussion

Our intention, in formally defining characters and words, is to establish our mathematical system clearly and accurately. To keep discussion concise, the definitions of elementary concepts such as strings and substrings, although widely used in this paper, will be taken for granted. We limit our basic notion to what has already been defined in Aho and Ullman (1972) and Kolman and Busby (1987).

Mathematically, word strings are nothing but symbol strings, with each symbol representing a word in the dictionary. In that sense, the word string definition is redundant as it is already covered by the definition of character string. However, since the relationships between character strings and word strings are very important in this paper, we believe it to be appropriate to list both definitions explicitly.

What is new in this section is mathematical definitions for character string generation and tokenization. We consider them fundamental to our mathematical description of the string tokenization problem.

There are two points worth highlighting here. The first relates to the introduction of the character string generation operation. In the literature, the tokenization problem is normally modeled independently with no connection whatsoever with the character string generation problem. By contrast, we model tokenization and generation as inverse problems to each other. In this way, we establish a well-defined mathematical system consisting of an alphabet, a dictionary, and the (generation) homomorphism (operation) and its inverse defined on the alphabet and dictionary. As will be seen throughout this paper, the establishment of the generation operation renders various types of tokenization problems easy to describe. The generation problem is relatively simple and easy to manage, so any modeling of the tokenization problem as its inverse (that is, as the generation problem) should make it more tractable.

The second point is in regard to the tokenization definition. In the literature, the string tokenization operation is normally required to generate a *unique* tokenized word string. Following such a definition of tokenization, introducing tokenization disambiguation at the very beginning is inevitable. We believe this to be a pitfall that has trapped many researchers. In contrast, we define the character string tokenization operation as the inverse operation (inverse homomorphism) of the character string generation operation (homomorphism). Naturally, the result of the tokenization operation is a *set* of tokenizations rather than a single word string. Such treatment suggests that we could use the divide-and-conquer problem-solving strategy—to decompose the complex string tokenization problem into several smaller and, hopefully, simpler subproblems. That is the basis of our two-stage, five-step iterative problem-solving strategy for sentence tokenization (Guo 1997).

3. Critical Point and Fragment

After clarifying both sentence generation and tokenization operations, we undertake next to further clarify sentence tokenization ambiguities. Among all the concepts to be introduced, critical points and critical fragments are probably two of the most important. We will prove that, for any character string on a complete tokenization dictionary, its critical points are *all and only* unambiguous token boundaries, and its critical fragments are the longest substrings with all inner positions ambiguous.

3.1 Ambiguity

Let Σ be an alphabet, D a dictionary, and S a character string over the alphabet.

Definition 5
The character string S from the alphabet Σ has **tokenization ambiguity** on dictionary D, if $|T_D(S)| > 1$. S has **no tokenization ambiguity**, if $|T_D(S)| = 1$. S is **ill-formed** on dictionary D, if $|T_D(S)| = 0$. A tokenization $W \in T_D(S)$ has tokenization ambiguity, if there exists another tokenization $W' \in T_D(S)$, $W' \neq W$.

Example 2 (cont.)
Since $T_D(fundsand) = \{$*"funds and"*, *"fund sand"*$\}$, i.e., $|T_D(fundsand)| = 2 > 1$, the character string *fundsand* has tokenization ambiguity. In other words, it is ambiguous in tokenization. Moreover, the tokenization *"funds and"* has tokenization ambiguity since there exists another possible tokenization *"fund sand"* for the same character string.

This definition is quite intuitive. If a character string could be tokenized in multiple ways, it would be ambiguous in tokenization. If a character string could only be tokenized in a unique way, it would have no tokenization ambiguity. If a character string could not be tokenized at all, it would be ill-formed. In this latter case, the dictionary is incomplete.

Intuitively, a position in a character string is ambiguous in tokenization or is an ambiguous token boundary if it is a token boundary in one tokenization but not in another. Formally, let $S = c_1 \dots c_n$ be a character string over an alphabet Σ and let D be a dictionary over the alphabet.

Definition 6
Position p has tokenization ambiguity or is an ambiguous token boundary, if there are two tokenizations $X = x_1 \dots x_s$ and $Y = y_1 \dots y_t$ in $T_D(S)$, such that $G(x_1 \dots x_u) = c_1 \dots c_p$ and $G(x_{u+1} \dots x_s) = c_{p+1} \dots c_n$ for some index u, and for any index v, there is neither $G(y_1 \dots y_v) = c_1 \dots c_p$ nor $G(y_{v+1} \dots y_t) = c_{p+1} \dots c_n$. Otherwise, the position has no tokenization ambiguity, or is an unambiguous token boundary.

Example 1 (cont.)
Given a typical English dictionary and the character string $S =$ *thisishisbook*, all three positions after character *s* are unambiguous in tokenization or are unambiguous token boundaries, since all possible tokenizations must take these positions as token boundaries.

Example 2 (cont.)
Given a typical English dictionary and the character string $S =$ *fundsand*, the position after the middle character *s* is ambiguous in tokenization or is an ambiguous token boundary since it is a token boundary in tokenization *"funds and"* but not in another tokenization *"fund sand"*.

3.2 Complete Dictionary

To avoid ill-formedness in sentence tokenization, we now introduce the concept of a complete tokenization dictionary.

Definition 7
A dictionary D over an alphabet Σ is **complete** if for any character string S from the alphabet, $S \in \Sigma^*$, there is $|T_D(S)| \geq 1$.

That is, for any character string $S = c_1 \ldots c_n$ from the alphabet, there exists at least one word string $W = w_1 \ldots w_m$ with S as its generated character string, $G(W) = S$.

Theorem 1
A dictionary D over an alphabet Σ is complete if and only if all the characters in the alphabet are single-character words in the dictionary.

Proof
On the one hand, every single character is also a character string (of length 1). To ensure that such a single-character string is being tokenized, the single character must be a word in the dictionary. On the other hand, if all the characters are words in the dictionary, any character string can at least be tokenized as a string of single-character words. □

Theorem 1 spells out a simple way of making any dictionary complete, which calls for adding all the characters of an alphabet into a dictionary as single-character words. This is referred to as the **dictionary completion process**. If not specified otherwise, in this paper, when referring to a complete dictionary or tokenization dictionary, we mean the dictionary after the completion process.

3.3 Critical Point and Fragment

Let $S = c_1 \ldots c_n$ be a character string over the alphabet Σ and let D be a dictionary over the alphabet. In addition, let $T_D(S)$ be the tokenization set of S on D.

Definition 8
Position p in character string $S = c_1 \ldots c_n$ is a **critical point**, if for any word string $W = w_1 \ldots w_m$ in $T_D(S)$, there exists an index k, $0 \leq k \leq m$, such that $G(w_1 \ldots w_k) = c_1 \ldots c_p$ and $G(w_{k+1} \ldots w_m) = c_{p+1} \ldots c_n$. In particular, the starting position 0 and the ending position n are the two **ordinary critical points**. Substring $c_{p+1} \ldots c_q$ is a **critical fragment** of S on D, if both p and q are critical points and any other position r in between them, $p < r < q$, is not a critical point.

Example 1 (cont.)
Given a typical English dictionary, there are five critical points in the character string $S = thisishisbook$. They are 0, 4, 6, 9, and 13. The corresponding four critical fragments are *this*, *is*, *his*, and *book*.

Example 2 (cont.)
Given a typical English dictionary, there is no extraordinary critical point in the character string $S = fundsand$. It is by itself the only critical fragment of this character string.

Given a complete tokenization dictionary, it is obvious that all single-character critical fragments or, more generally, single-character strings, possess unique tokenization. That is, they possess neither ambiguity nor ill-formedness in tokenization. However, the truth of the statement below (Lemma 1) is less obvious.

Lemma 1
For a complete tokenization dictionary, all multicharacter critical fragments and all of their inner positions are ambiguous in tokenization.

Proof
Let $S = c_1 \ldots c_n$, $n > 1$, be a multicharacter critical fragment. Because the tokenization dictionary is complete, the critical fragment can at least be tokenized as a string of single-character words. On the other hand, because it is a critical fragment, for any position p, $1 \leq p \leq n-1$, there must exist a tokenization $W = w_1 \ldots w_m$ in $T_D(S)$ such that for any index k, $0 \leq k \leq m$, there is neither $G(w_1 \ldots w_k) = c_1 \ldots c_p$ nor $G(w_{k+1} \ldots w_m) = c_{p+1} \ldots c_n$. As this tokenization differs from the above-mentioned tokenization of the string of single-character words, the critical fragment has at least two different tokenizations and thus has tokenization ambiguity. □

Theorem 2
For any character string on a complete tokenization dictionary, its critical points are *all and only* unambiguous token boundaries.

Proof
By Lemma 1, all positions within critical fragments are ambiguous in tokenization. By Definition 8, critical points are unambiguous in tokenization. □

Corollary
For any character string on a complete tokenization dictionary, its critical fragments are the longest substrings with all inner positions ambiguous.

Proof
By Theorem 2. □

3.4 Discussion

In this section, we have described sentence tokenization ambiguity from three different angles: character strings, tokenizations, and individual string positions. The basic idea is conceptually simple: ambiguity exists when there are different means to the same end. For instance, as long as a character string has multiple tokenizations, it is ambiguous.

This description of ambiguity is complete. Given a character string and a dictionary, it is always possible to answer deterministically whether or not a string is ambiguous in tokenization. Conceptually, for any character string, by checking every one of its possible substrings in a dictionary, and then by enumerating all valid word concatenations, all word strings with the character string as their generated character string can be produced. Just counting the number of such word strings will provide the answer to whether or not the character string is ambiguous.

Some researchers question the validity of the complete dictionary assumption. Here we argue that, even in the strictest linguistic sense, there exists no single character that cannot be used as a single-character word in sentences. In any case, any natural language must allow us to directly refer to single characters. For instance, you could say "character x has many written forms" or "the character x in this word can be omitted" for any character x.[3]

3 Even so, some researchers might still insist that the character x here is just for temporary use and cannot be regarded as a regular word with the many linguistic properties generally associated with words. Understanding the importance of such a distinction, we will use the more generic term *token*, rather than the loaded term *word*, when we need to highlight the distinction. It must be added, however, that the two are largely used interchangeably in this paper.

The validity of the complete dictionary assumption can also be justified from an engineering perspective. To ensure a so-called soft landing, any practical application system must be designed so that every input character string can always be tokenized. In other words, a complete dictionary is an operational must. Moreover, without such a complete dictionary, it would not be possible to avoid ill-formedness in sentence tokenization nor to make the generation-tokenization system for character and words closed and complete. Without such definitions of well-formedness, any rigorous formal study would be impossible.

The concepts of critical point and critical fragment are fundamental to our sentence tokenization theory. By adopting the complete dictionary assumption, it has been proven that critical points are all and only unambiguous token boundaries while critical fragments are the longest substrings with all inner positions ambiguous.

This is a very strong and significant statement. It provides us with a precise understanding of what and where tokenization ambiguities are. Although the proof itself is easy to follow, the result has nonetheless been a surprise. As demonstrated in Guo (1997), many researchers have tried but failed to answer the question in such a precise and complete way. Consequently, while they proposed many sophisticated algorithms for the discovery of ambiguity (and certainty), they never were able to arrive at such a concise and complete solution.

As critical points are all and only unambiguous token boundaries, an identification of all of them would allow for a long character string to be broken down into several short but fully ambiguous critical fragments. As shown in Guo (1997), critical points can be completely identified in linear time. Moreover, in practice, most critical fragments are dictionary tokens by themselves, and the remaining nondictionary fragments are generally very short. In short, the understanding of critical points and fragments will significantly assist us in both efficient tokenization implementation and tokenization ambiguity resolution.

The concepts of critical point and critical fragment are similar to those of segment point and character segment in Wang (1989, 37), which were defined on a sentence word graph for the purpose of analyzing the computational complexity of his new tokenization algorithm. However, Wang (1989) neither noticed their connection with tokenization ambiguities nor realized the importance of the complete dictionary assumption, and hence failed to demonstrate their crucial role in sentence tokenization.

4. Critical Tokenization

This section seeks to disclose an important structure of the set of different tokenizations. We will see that different tokenizations can be linked by the cover relationship to form a partially ordered set. Based on that, we will establish the notion of critical tokenization and prove that every tokenization is a subtokenization of a critical tokenization, but no critical tokenization has true supertokenization.

4.1 Cover Relationship

Definition 9

Let X and Y be word strings. X **covers** Y, or X has a **cover relation** to Y, denoted $X \leq Y$, if for any substring X_s of X, there exists substring Y_s of Y, such that $|X_s| \leq |Y_s|$ and $G(X_s) = G(Y_s)$. If $X \leq Y$, then X is called a **covering word string** of Y, and Y a **covered word string** of X.

Intuitively, $X \leq Y$ implies $|X| \leq |Y|$. In other words, shorter word strings cover longer word strings. However, an absence of $X \leq Y$ does not imply the existence of

$Y \leq X$. Some word strings do not cover each other. In other words, shorter word strings do not always cover longer word strings.

Example 1 (cont.)
The word string *"this is his book"* covers the word string *"th is is his book"*, but not vice versa.

Example 2 (cont.)
The word strings *"funds and"* and *"fund sand"* do not cover each other.

Definition 9′
Let A and B be sets of word strings. A covers B, or A has a cover relation to B, denoted $A \leq B$, if for any $Y \in B$, there is $X \in A$, such that $X \leq Y$. If $A \leq B$, A is called a **covering word string set** of B, and B a **covered word string set** of A.

Example 3
Given the alphabet $\Sigma = \{a, b, c, d\}$, dictionary $D = \{a, b, c, d, ab, bc, cd, abc, bcd\}$, and character string $S = abcd$ from the alphabet, there is $T_D(S) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$. Among them, there are $\{abc/d\} \leq \{ab/c/d, a/bc/d\}$, $\{ab/cd\} \leq \{ab/c/d, a/b/cd\}$, $\{a/bcd\} \leq \{a/bc/d, a/b/cd\}$ and $\{ab/c/d, a/bc/d, a/b/cd\} \leq \{a/b/c/d\}$. Moreover, there is $\{abc/d, ab/cd, a/bcd\} \leq T_D(S)$.

4.2 Partially Ordered Set

Lemma 2
The cover relation is transitive, reflexive, and antisymmetric. That is, the cover relation is a (reflexive) partial order.

Lemma 2, proved in Guo (1997), reveals that the cover relation is a partial order—a well-defined mathematical structure with good mathematical properties. Consequently, from any textbook on discrete mathematics (Kolman and Busby [1987], for example), it is known that the tokenization set $T_D(S)$, together with the word string cover relation $\leq$, forms a partially ordered set, or simply a **poset**. We shall denote this poset by $(T_D(S), \leq)$. In case there is no confusion, we may refer to the poset simply as $T_D(S)$.

In the literature, usually a poset is graphically presented in a **Hasse diagram**, which is a digraph with vertices representing poset elements and arcs representing direct partial order relations between poset elements. In a Hasse diagram, all connections implied by the partial order's transitive property are eliminated. That is, if $X \leq Y$ and $Y \leq Z$, there should be no arc from X to Z.

Example 3 (cont.)
The poset $T_D(abcd) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$ can be graphically presented in the Hasse diagram in Figure 1.

Certain elements in a poset are of special importance for many of the properties and applications of posets. In this paper, we are particularly interested in the **minimal elements** and **least elements**. In standard textbooks, they are defined in the following manner: Let $(A, \leq)$ be a poset. An element $a \in A$ is called a minimal element of A if there is no element $c \in A$, $c \neq a$, such that $c \leq a$. An element $a \in A$ is called a least element of A if $a \leq x$ for all $x \in A$. (Kolman and Busby 1987, 195–196).

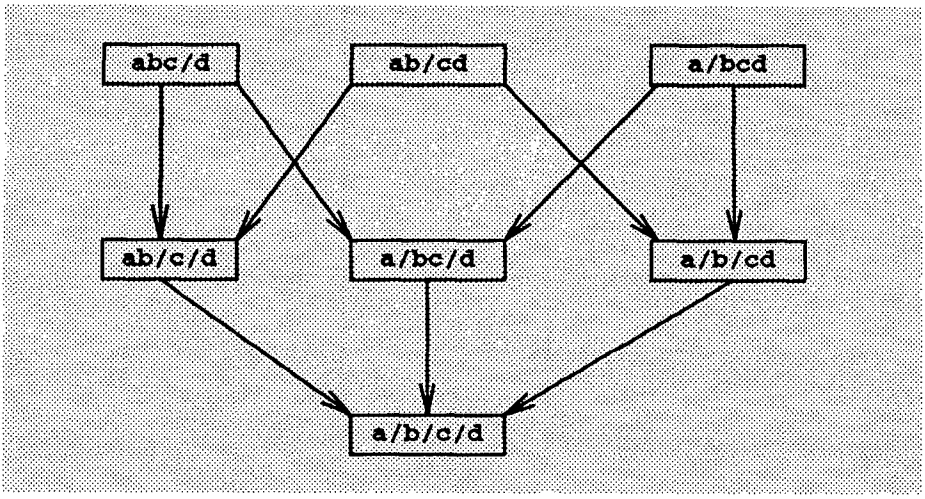

Figure 1
The Hasse diagram for the poset $T_D(abcd) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$.

Example 1 (cont.)
The word string *"this is his book"* is both the minimal element and the least element of both $T_D(thisishisbook) = \{$*"this is his book"*$\}$ and $T_{D'}(thisishisbook) = \{$*"th is is his book"*, *"this is his book"*$\}$.

Example 2 (cont.)
The poset $T_D(fundsand) = \{$*"funds and"*, *"fund sand"*$\}$ has both *"funds and"* and *"fund sand"* as its minimal elements, but has no least element.

Example 3 (cont.)
The poset $T_D(abcd) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$ has three minimal elements: abc/d, ab/cd, a/bcd. It has no least element.

Note that any finite nonempty poset has at least one minimal element. Any poset has at most one least element (Kolman and Busby 1987, 195-198).

4.3 Critical Tokenization

This section deals with the most important concept—critical tokenization. Let Σ be an alphabet, D a dictionary over the alphabet, and S a character string over the alphabet. In this case, $(T_D(S), \leq)$ is the poset.

Definition 10
The **character string critical tokenization operation** C_D is a mapping C_D: $\Sigma^* \rightarrow 2^{D^*}$ defined as: for any S in Σ^*, $C_D(S) = \{W \mid W$ *is a minimal element of the poset* $(T_D(S), \leq)\}$. Any word string W in $C_D(S)$ is a **critically tokenized word string**, or simply a **critical tokenization**, or **CT tokenization** for short, of the character string S. And $C_D(S)$ is the set of critical tokenizations.

In other words, the critical tokenization operation maps any character string to its set of critical tokenizations. A word string is critical if any other word string does not cover it.

Example 1 (cont.)
Given the English alphabet, the tiny Dictionary $D = \{th, this, is, his, book\}$, and the character string $S = thisishisbook$, there is $C_D(S) = \{$"*this is his book*"$\}$. This critical tokenization set contains the unique critical tokenization "*this is his book*". Note that the only difference between "*this is his book*" and "*th is is his book*" is that the word *this* in the former is split into two words *th* and *is* in the latter.

Example 2 (cont.)
Given the English alphabet, the tiny Dictionary $D = \{fund, funds, and, sand\}$, and the character string $S = fundsand$, there is $C_D(S) = \{$"*funds and*", "*fund sand*"$\}$.

Example 3 (cont.)
Let $\Sigma = \{a, b, c, d\}$ and $D = \{a, b, c, d, ab, bc, cd, abc, bcd\}$. There is $C_D(abcd) = \{abc/d, ab/cd, a/bcd\}$. If $D' = \{a, b, c, d, ab, bc, cd\}$, then $C_{D'}(abcd) = \{a/bc/d, ab/cd\}$.

Example 4
Given the English alphabet, the tiny Dictionary $D = \{the, blue, print, blueprint\}$, and the character string $S = theblueprint$, there are $T_D(S) = \{$"*the blueprint*", "*the blue print*"$\}$ and $C_D(S) = \{$"*the blueprint*"$\}$. Note that the tokenization "*the blue print*" is not *critical* (not a critical tokenization).

4.4 Super- and SubTokenization

Intuitively, a tokenization is a subtokenization of another tokenization if further tokenizing words in the latter can produce the former. Formally, let S be a character string over an alphabet Σ and let D be a dictionary over the alphabet. In addition, let $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_m$ be tokenizations of S on D, $X, Y \in T_D(S)$. That gives us the following definition:

Definition 11
Y is a **subtokenization** of X and X is a **supertokenization** of Y if, for any word x in X, there exists a substring Y_s of Y such that $x = G(Y_s)$. Y is a **true subtokenization** of X and X is a **true supertokenization** of Y, if Y is a subtokenization of X and $X \neq Y$.

Example 1 (cont.)
The tokenization "*th is is his book*" is a subtokenization of the critical tokenization "*this is his book*".

Example 4 (cont.)
The tokenization "*the blue print*" is a subtokenization of the critical tokenization "*the blueprint*".

4.5 Theorem

Lemma 3
Y is a subtokenization of X if and only if $X \leq Y$.

Proof
If $X \leq Y$, by definition, for any substring X_s of X, there exists substring Y_s of Y, such that $|X_s| \leq |Y_s|$ and $G(X_s) = G(Y_s)$. Also by definition, there is $x = G(x)$ for every

single word x. As any single word in a word string is also its single-word substring, it can be concluded that for any word x in X, there exists a substring Y_s of Y, such that $x = G(Y_s)$.

On the other hand, if Y is a subtokenization of X, by definition, for any word x in X, there exists a substring Y_s of Y such that $x = G(Y_s)$. Thus, given any substring X_s of X, $X_s = x_1 \ldots x_n$, for any k, $1 \leq k \leq n$, there exists a substring Y_k of Y such that $x_k = G(Y_k)$. Denote $Y_s = Y_1 \ldots Y_m$, there is $|X_s| \leq |Y_s|$ and $G(X_s) = G(Y_s)$. By definition, there is $X \leq Y$. □

Lemma 3 reveals that a word string is covered by another word string if and only if every word in the latter is realized in the former as a word string. In other words, a covering word string is in a more *compact* form than its covered word string.

Theorem 3
Every tokenization has a critical tokenization as its supertokenization, but critical tokenization has no true supertokenization.

That is, for any tokenization Y, $Y \in T_D(S)$, there exists critical tokenization X, $X \in C_D(S)$, such that X is a supertokenization of Y. Moreover, if Y is a critical tokenization and X is its supertokenization, there is $X = Y$.

Proof
By definition, for any tokenization Y, $Y \in T_D(S)$, there is a critical tokenization X, $X \in C_D(S)$, such that $X \leq Y$. By Lemma 3, it would be the same as saying that X is a supertokenization of Y. The second part of the theorem is from the definition of critical tokenization. □

Theorem 3 states that no critical tokenization can be produced by further tokenizing words in other tokenizations. However, all other tokenizations can be produced from at least one critical tokenization by further tokenizing words in it.

Example 3 (cont.)
Given $T_D(S) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$, there is $C_D(S) = \{abc/d, ab/cd, a/bcd\} \leq T_D(S)$. By splitting the word abc in $abc/d \in C_D(S)$ into $a/b/c$, ab/c or a/bc, we can make another three tokenizations in $T_D(S)$: $a/b/c/d$, $ab/c/d$ and $a/bc/d$. Similarly, from ab/cd, we can bring back $a/b/c/d$, $ab/c/d$ and $a/b/cd$; and from abc/d, we can recover $a/b/c/d$, $ab/c/d$ and $a/bc/d$. By merging all word strings produced together with word strings in $C_D(S) = \{abc/d, ab/cd, a/bcd\}$, the entire tokenization set $T_D(S)$ is reclaimed.

4.6 Discussion

Since the theory of partially ordered sets is well established, we can use it to enhance our understanding of the mathematical structure of string tokenization. One of the obvious and immediate results is the concept of critical tokenization, which is simply another name for the minimal element set of a poset. The least element is another important concept. Although it may seem trivial to the string tokenization problem, the critical tokenization is, in fact, absolutely crucial. For instance, Theorem 3 states that, from critical tokenization, any tokenization can be produced (enumerated). As the number of critical tokenizations is normally considerably less than the total amount of all possible tokenizations, this theorem leads us to focus on the study of a few critical

ones. In the next few sections, we shall further investigate certain important aspects of critical tokenizations.

5. Critical and Hidden Ambiguities

This section clarifies the relationship between critical tokenization and various types of tokenization ambiguities.

5.1 Critical Ambiguity in Tokenization

Definition 12
Let Σ be an alphabet, D a dictionary, and S a character string over the alphabet. The character string S from the alphabet Σ has **critical ambiguity in tokenization** on dictionary D if $|C_D(S)| > 1$. S has **no critical ambiguity in tokenization** if $|C_D(S)| = 1$. A tokenization $W \in T_D(S)$ has **critical ambiguity in tokenization** if there exists another tokenization $W' \in T_D(S)$, $W' \neq W$, such that neither $W \leq W'$ nor $W' \leq W$ holds.

Example 2 (cont.)
Since $C_D(fundsand) = \{$*"funds and"*, *"fund sand"*$\}$, i.e., $|C_D(fundsand)| = 2 > 1$, the character string *fundsand* has critical ambiguity in tokenization. Moreover, the tokenization *"funds and"* has critical ambiguity in tokenization since there exists another possible tokenization *"fund sand"* such that both *"funds and"* $\leq$ *"fund sand"* and *"fund sand"* $\leq$ *"funds and"* do not hold.

Example 4 (cont.)
Since $C_D(theblueprint) = \{$*"the blueprint"*$\}$, the character string *theblueprint* does not have critical ambiguity in tokenization.

It helps to clarify that the only difference between the definition of tokenization ambiguity and that of critical ambiguity in tokenization lies in the tokenization set: While tokenization ambiguity is defined on the entire tokenization set $T_D(S)$, critical ambiguity in tokenization is defined only on the critical tokenization set $C_D(S)$, which is a subset of $T_D(S)$.

As all critical tokenizations are minimal elements on the word string cover relationship, the existence of critical ambiguity in tokenization implies that the "most powerful and commonly used" (Chen and Liu 1992, 104) principle of maximum tokenization would not be effective in resolving critical ambiguity in tokenization and implies that other means such as statistical inferencing or grammatical reasoning have to be introduced. In other words, critical ambiguity in tokenization is unquestionably critical.

Critical ambiguity in tokenization is the precise mathematical description of conventional concepts such as disjunctive ambiguity (Webster and Kit [1992, 1108], for example) and overlapping ambiguity (Sun and T'sou [1995, 121], for example). We will return to this topic in Section 5.4.

5.2 Hidden Ambiguity in Tokenization

Definition 13
Let Σ be an alphabet, D a dictionary, and S a character string over the alphabet. The character string S from the alphabet Σ has **hidden ambiguity in tokenization** on dictionary D if $T_D(S) \neq C_D(S)$. A tokenization $W \in T_D(S)$ has **hidden ambiguity in tokenization** if there exists another tokenization $W' \in T_D(S)$, $W' \neq W$, such that $W \leq W'$.

Example 4 (cont.)
Let S = *theblueprint*, $T_D(S)$ = {"*the blueprint*", "*the blue print*"}, and $C_D(S)$ = {"*the blueprint*"}. Since $T_D(S) \neq C_D(S)$, the character sting *theblueprint* has hidden ambiguity in tokenization. Since "*the blueprint*" $\leq$ "*the blue print*", the character string "*the blueprint*" has hidden ambiguity in tokenization.

Intuitively, a tokenization has hidden ambiquity in tokenization, if some words in it can be further decomposed into word strings, such as "*blueprint*" to "*blue print*". They are called *hidden* or *invisible* because others cover them. The resolution of hidden ambiguity in tokenization is the aim of the principle of maximum tokenization (Jie 1989; Jie and Liang 1991). Under this principle, only covering tokenizations win and all covered tokenizations are discarded.

Hidden ambiguity in tokenization is the precise mathematical description of conventional concepts such as conjunctive ambiguity (Webster and Kit [1992, 1108], for example), combinational ambiguity (Liang [1987], for example) and categorical ambiguity (Sun and T'sou [1995, 121], for example). We will return to this topic in Section 5.4.

5.3 Ambiguity = Critical + Hidden

Let Σ be an alphabet, D a dictionary, and S a character string over the alphabet.

Theorem 4
A character string S over an alphabet Σ has tokenization ambiguity on a tokenization dictionary D if and only if S has either critical ambiguity in tokenization or hidden ambiguity in tokenization.

Proof
If S has critical ambiguity in tokenization, by definition, there is $|C_D(S)| > 1$. If S has hidden ambiguity in tokenization, by definition, there is $T_D(S) \neq C_D(S)$. In both cases, since $C_D(S) \subseteq T_D(S)$, there must be $|T_D(S)| > 1$. By definition, S has tokenization ambiguity.

If S has tokenization ambiguity, by definition, there is $|T_D(S)| > 1$. Since any finite nonempty poset has at least one minimal element, there is $|C_D(S)| > 0$. Since $C_D(S) \subseteq T_D(S)$, there is $T_D(S) \neq C_D(S)$ if $|C_D(S)| = 1$. In this case, by definition, S has hidden ambiguity in tokenization. If $|C_D(S)| > 1$, by definition, S has critical ambiguity in tokenization. □

Theorem 4 explicitly and precisely states that tokenization ambiguity is the union of critical ambiguity in tokenization and hidden ambiguity in tokenization. This result helps us in the understanding of character string tokenization ambiguity.

5.4 Discussion

By freezing the problem of token identity determination, tokenization ambiguity identification and resolution are all that is required in sentence tokenization. Consequently, it must be crucial and beneficial to pursue an explicit and accurate understanding of various types of character string tokenization ambiguities and their relationships.

In the literature, however, the general practice is not to formally define and classify ambiguities but to apply various terms to them, such as overlapping ambiguity and combinational ambiguity in their intuitive and normally fuzzy senses. Nevertheless, efforts do exist to rigorously assign them precise, formal meanings. As a representa-

tive example, in Webster and Kit (1992, 1108), both conjunctive (combinational) and disjunctive (overlapping) ambiguities are defined in the manner given below.

1. **TYPE I:** In a sequence of Chinese[4] characters $S = a_1 \ldots a_i b_1 \ldots b_j$, if $a_1 \ldots a_i, b_1 \ldots b_j$, and S are each a word, then there is **conjunctive ambiguity** in S. The segment S, which is itself a word, contains other words. This is also known as **multi-combinational ambiguity.**
2. **TYPE II:** In a sequence of Chinese characters $S = a_1 \ldots a_i b_1 \ldots b_j c_1 \ldots c_k$, if $a_1 \ldots a_i b_1 \ldots b_j$ and $b_1 \ldots b_j c_1 \ldots c_k$ are each a word, then S is an **overlapping ambiguous segment**, or in other words, the segment S displays **disjunctive ambiguity**. The segment $b_1 \ldots b_j$ is known as an **overlap**, which is usually one character long.

The definitions above contain nothing improper. In fact, conjunctive (combinational) ambiguity as defined above is a special case of hidden ambiguity in tokenization, since "$a_1 \ldots a_i b_1 \ldots b_j$" $\leq$ "$a_1 \ldots a_i / b_1 \ldots b_j$". Moreover, disjunctive (overlapping) ambiguity is a special case of critical ambiguity in tokenization, since for the character string $S = a_1 \ldots a_i b_1 \ldots b_j c_1 \ldots c_k$, both "$a_1 \ldots a_i b_1 \ldots b_j / c_1 \ldots c_k$" and "$a_1 \ldots a_i / b_1 \ldots b_j c_1 \ldots c_k$" are critical tokenizations.

The definitions above, however, are neither *complete* nor *critical*. In our opinion, a definition is complete only if any phenomenon in the problem domain can be properly described (defined). With regard to the character string tokenization problem proper, this completeness requirement can be translated as: given an alphabet, a dictionary, and a character string, the definition should be sufficient to answer the following two questions: (1) does this character string have tokenization ambiguity? (2) if yes, what type of ambiguity does it have?

The definitions above cannot fulfill this completeness requirement. For instance, if $a_1 \ldots a_i, b_1 \ldots b_j, c_1 \ldots c_k$, and $a_1 \ldots a_i b_1 \ldots b_j c_1 \ldots c_k$ are all words in a dictionary, the character string $S = a_1 \ldots a_i b_1 \ldots b_j c_1 \ldots c_k$, while *intuitively* in Type I (conjunctive ambiguity), is, in fact, captured neither by Type I nor by Type II.

We agree that, although to do so would not be trivial, it is nevertheless possible to make the definitions above complete by carefully listing and including all possible cases. However, *criticality*, which is what is being explored in this paper, would most probably still not be captured in such a carefully generalized ambiguity definition.

What we believe to be crucial is the association between tokenization ambiguity and the maximization or minimization property of the partially ordered set on the cover relation. As will be illustrated later in this paper, such an association is exceptionally important in attempting to understand ambiguities and in developing disambiguation strategies.

In short, both the cover relation and critical tokenization have given us a clear picture of character string tokenization ambiguity as expressed in Theorem 4.

6. Maximum Tokenization

This section clarifies the relationship between critical tokenization (*CT*) and three other representative implementations of the principle of maximum tokenization, i.e., forward maximum tokenization (*FT*), backward maximum tokenization (*BT*) and shortest tokenization (*ST*). It will be proven that *ST*, *FT* and *BT* are all true subclasses of *CT*.

4 Although Webster and Kit include the modifier *Chinese*, the definition has nothing to do with specific characteristics of Chinese but is general (multilingual).

6.1 Forward Maximum Tokenization

Let Σ be an alphabet, D a dictionary on the alphabet, and S a character string over the alphabet.

Definition 14

A tokenization $W = w_1 \dots w_m \in T_D(S)$ is a **forward maximum tokenization** of S over Σ and D, or **FT tokenization** for short, if, for any k, $1 \leq k \leq m$, there exist i and j, $1 \leq i \leq j \leq n$, such that[5]

1. $G(w_1 \dots w_{k-1}) = c_1 \dots c_{i-1}$,
2. $w_k = c_i \dots c_j$, and
3. For any j', $j < j' \leq n$, there is $c_i \dots c_{j'} \notin D$.

The **forward maximum tokenization** operation, or **FT operation** for short, is a mapping F_D: $\Sigma^* \rightarrow 2^{D^*}$ defined as: for any $S \in \Sigma^*$, $F_D(S) = \{W \mid W$ is a FT tokenization of S over Σ and $D\}$.

This definition is in fact a descriptive interpretation of the widely recommended conventional constructive forward maximum tokenization procedure (Liu 1986a, 1986b; Liang 1986, 1987; Chen and Liu 1992; Webster and Kit 1992).

Example 3 (cont.)

The character string $S = abcd$ has the word string abc/d as its sole FT tokenization in $T_D(S) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$, i.e., $F_D(S) = \{abc/d\}$.

Example 2 (cont.)

$F_D(fundsand) = \{$"*funds and*"$\}$, i.e., the character string *fundsand* has its sole FT tokenization "*funds and*".

Example 4 (cont.)

$F_D(S) = \{$"*the blueprint*"$\}$, i.e., the word string "*the blueprint*" is the only FT tokenization for the character string $S =$ *theblueprint*.

Lemma 4

For all $S \in \Sigma^*$, there are $|F_D(S)| \leq 1$ and $F_D(S) \subseteq C_D(S)$.

That is to say, any character string has, at most, a single FT tokenization. Moreover, if the FT tokenization exists, it is a CT tokenization.

Proof

Certain character strings do not have FT tokenization on some dictionaries, even if they have many possible tokenizations. For example, given the alphabet $\Sigma = \{a, b, c, d\}$ and the dictionary $D = \{a, abc, bcd\}$, there is $T_D(abcd) = \{a/bcd\}$. But the single tokenization does not fulfill condition (3) in the definition above for $k = 1$, because the longer word *abc* exists in the dictionary.

5 Note, as a widely adopted convention, in case $k < 1$, $w_1 \dots w_{k-1}$ represents the empty word string v and $c_1 \dots c_{k-1}$ represents the empty character string e.

Assume both $X = x_1 \dots x_m$ and $Y = y_1 \dots y_{m'}$ are *FT* tokenizations, $X \neq Y$. Then, there must exist k, $1 \leq k \leq min(m, m')$, such that $x_{k'} = y_{k'}$, for all k', $1 \leq k' < k$, but $x_k \neq y_k$. Since $G(X) = G(Y)$, there must be $|x_k| \neq |y_k|$. Consequently, either X or Y is unable to fulfill condition (3) of definition 14. By contradiction, there must be $X = Y$. In other words, any character string at most has single *FT* tokenization.

Assume the *FT* tokenization $X = x_1 \dots x_m$ is not a *CT* tokenization. By Theorem 3, there must exist a *CT* tokenization $Y = y_1 \dots y_{m'}$ such that $X \neq Y$ and $Y \leq X$. Thus, by the cover relation definition, for any substring Y_s of Y, there exists substring X_s of X, such that $|Y_s| \leq |X_s|$ and $G(X_s) = G(Y_s)$. Since $X \neq Y$, there must exist k, $1 \leq k \leq min(m, m')$, such that $x_{k'} = y_{k'}$, for all k', $1 \leq k' < k$, but $|x_k| \leq |y_k|$. This leads to a conflict with condition (3) in the definition. In other words, X cannot be an *FT* tokenization if it is not a CT tokenization. □

6.2 Backward Maximum Tokenization

Let Σ be an alphabet, D a dictionary on the alphabet, and S a character strings over the alphabet.

Definition 15

A tokenization $W = w_1 \dots w_m \in T_D(S)$ is a **backward maximum tokenization** of S over Σ and D, or **BT tokenization** for short, if for any k, $1 \leq k \leq m$, there exist i and j, $1 \leq i \leq j \leq n$, such that

1. $G(w_{k+1} \dots w_m) = c_{j+1} \dots c_n$,
2. $w_k = c_i \dots c_j$, and
3. For any i', $1 \leq i' < i$, there is $c_{i'} \dots c_j \notin D$.

The **backward maximum tokenization operation** is a mapping B_D: $\Sigma^* \rightarrow 2^{D^*}$ defined as: for any $S \in \Sigma^*$, $B_D(S) = \{W \mid W$ is a *BT* tokenization of S over Σ and $D\}$.

This definition is in fact a descriptive interpretation of the widely recommended conventional constructive backward maximum tokenization procedure (Liu 1986a, 1986b; Liang 1986, 1987; Chen and Liu 1992; Webster and Kit 1992).

Example 3 (cont.)

For the character string $S = abcd$, the word string a/bcd is the only *BT* tokenization in $T_D(S) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$. That is, $B_D(S) = \{a/bcd\}$.

Example 2 (cont.)

For the character string $S =$ *fundsand*, there is $B_D(fundsand) = \{$"*fund sand*"$\}$. That is, the word string "*fund sand*" is the only *BT* tokenization.

Example 4 (cont.)

For the character string $S =$ *theblueprint*, there is $B_D(S) = \{$"*the blueprint*"$\}$. That is, the word string "*the blueprint*" is the only *BT* tokenization.

Lemma 5

For all $S \in \Sigma^*$, there are $|B_D(S)| \leq 1$ and $B_D(S) \subseteq C_D(S)$.

That is, any character string has at most one *BT* tokenization. Moreover, if the *BT* tokenization exists, it is a *CT* tokenization.

Proof
Parallel to the proof for Lemma 4. □

6.3 Shortest Tokenization

Definition 16
The **shortest tokenization operation** S_D is a mapping S_D: $\Sigma^* \rightarrow 2^{D^*}$ defined as: for any S in Σ^*, $S_D(S) = \{W \mid |W| = min_{W' \in T_D(S)}|W'|\}$. Every tokenization W in $S_D(S)$ is a **shortest tokenization**, or **ST tokenization** for short, of the character string S.

In other words, a tokenization W of a character string S is a shortest tokenization if and only if the word string has the minimum word string length among all possible tokenizations.

This definition is in fact a descriptive interpretation of the constructive shortest path finding tokenization procedure proposed by Wang (1989) and Wang, Wang, and Bai (1991).

Example 3 (cont.)
Given the character string $S = abcd$. For the dictionary $D = \{a, b, c, d, ab, bc, cd, abc, bcd\}$, both abc/d and a/bcd are *ST* tokenizations in $T_D(S) = \{a/b/c/d, a/b/cd, a/bc/d, a/bcd, ab/c/d, ab/cd, abc/d\}$. That is, $S_D(S) = \{abc/d, a/bcd\}$. For $D' = \{a, b, c, d, ab, bc, cd\}$, however, there is $S_{D'}(S) = \{ab/cd\}$. Note, in this case, the *CT* tokenization $a/bc/d$ is not in $S_{D'}(S)$.

Example 2 (cont.)
For the character string $S = fundsand$, there is $S_D(fundsand) = \{$"*funds and*", "*fund sand*"$\}$. That is, both "*funds and*" and "*fund sand*" are *ST* tokenizations.

Example 4 (cont.)
For the character string $S = theblueprint$, there is $S_D(S) = \{$"*the blueprint*"$\}$. That is, the word string "*the blueprint*" is the only *ST* tokenization.

Lemma 6
$S_D(S) \subseteq C_D(S)$ for all $S \in \Sigma^*$. That is, every *ST* tokenization is a *CT* tokenization.

Proof
Let X be an *ST* tokenization, $X \in S_D(S)$. Assume X is not a *CT* tokenization, $X \notin C_D(S)$. Then, by Theorem 3, there exists a *CT* tokenization $Y \notin C_D(S)$, $Y \neq X$, such that $Y \leq X$. By the definition of the cover relation, there is $|Y| \leq |X|$. In fact, as $X \neq Y$, there must be $|Y| < |X|$. This is in conflict with the fact that X is an *ST* tokenization. Hence, the lemma is proven by contradiction. □

6.4 Theorem

Theorem 5
$F_D(S) \cup B_D(S) \subseteq C_D(S)$ and $S_D(S) \subseteq C_D(S)$ for all $S \in \Sigma^*$. Moreover, there exists $S \in \Sigma^*$, such that $F_D(S) \cup B_D(S) \neq C_D(S)$ or $S_D(S) \neq C_D(S)$. That is, the forward maximum tokenization, the backward maximum tokenization, and the shortest tokenization are all true subclasses of critical tokenization.

Proof
The first part is the combination of Lemma 4, 5, and 6. The second part is exemplified by Example 3 above. □

6.5 Principle of Maximum Tokenization

The three tokenization definitions in this section are essentially *descriptive* restatements of the corresponding *constructive* tokenization procedures, which in turn are realizations of the widely followed principle of maximum tokenization (e.g., Liu 1986; Liang 1986a, 1986b; Wang 1989; Jie 1989; Wang, Su, and Mo 1990; Jie, Liu, and Liang 1991a, b; Yeh and Lee 1991; Webster and Kit 1992; Chen and Liu 1992; Guo 1993; Wu and Su 1993; Nie, Jin, and Hannan 1994; Sproat et al. 1996; Wu et al. 1994; Li et al. 1995; Sun and T'sou 1995; Wong et al. 1995; Bai 1995; Sun and Huang 1996).

The first work closest to this principle, according to Liu (1986, 1988), was the *5-4-3-2-1 tokenization algorithm* proposed by a Russian MT practitioner in 1956. This algorithm is a special version of the greedy-type implementation of the forward maximum tokenization and is still in active use. For instance, Yun, Lee, and Rim (1995) recently applied it to Korean compound tokenization.

It is understood that forward maximum tokenization, backward maximum tokenization and shortest tokenization are the three most representative and widely quoted works following the general principle of maximum tokenization. However, the principle itself is not crystal-clear in the literature. Rather, it only serves as a general guideline, as different researchers make different interpretations. As Chen and Liu (1992, 104) noted, "there are a few variations of the sense of maximal matching." Hence, many variations have been derived after decades of fine-tuning and modification. As Webster and Kit (1992, 1108) acknowledged, different realizations of the principle "were invented one after another and seemed inexhaustible."

While researchers generally agree that a dictionary word should be tokenized as itself, they usually have different opinions on how a non-dictionary word (critical) fragment should be tokenized. While they all agree that a certain *form* of extremes must be attained, they nevertheless have their own understanding of what the *form* should be.

Consequently, it should come as no surprise to see various kinds of theoretical generalization or summarization work in the literature. A good representative work is by Kit and his colleagues (Jie 1989; Jie, Liu, and Liang 1991a, b; Webster and Kit 1992), who proposed a three-dimensional structural tokenization model. This model, called ASM for Automatic Segmentation Model, is capable of characterizing up to eight classes of different maximum or minimum tokenization procedures. Among the eight procedures, based on both analytical inferences and experimental studies, both forward maximum tokenization and backward maximum tokenization are recommended as good solutions. Unfortunately, in Webster and Kit (1992, 1108), they unnecessarily made the following overly strong claim:

> It is believed that all elemental methods are included in this model. Furthermore, it can be viewed as the ultimate model for methods of string matching of any elements, including methods for finding English idioms.

The shortest tokenization proposed by Wang (1989) provides an obvious counterexample. As Wang (1989) exemplified[6], for the alphabet $\Sigma = \{a, b, c, d, e\}$ and the

6 The original example is "结合成分子", a widely quoted Chinese phrase difficult to tokenize. Its

dictionary $D = \{a, b, c, d, e, ab, bc, cd, de\}$, the character string $S = abcde$ has FT set $F_D(S) = \{ab/cd/e\}$, BT set $B_D(S) = \{a/bc/de\}$ and ST set $S_D(S) = \{ab/cd/e, a/bc/de, ab/c/de\}$. Clearly, the ST tokenization $ab/c/de$, which fulfills the principle of maximum tokenization and is the desired tokenization in some cases, is neither FT nor BT tokenization. Moreover, careful checking showed that the missed ST tokenization is not in any of the eight tokenization solutions covered by the ASM model. In short, the ASM model is not a complete interpretation of the principle of maximum tokenization.

Furthermore, the shortest tokenization still does not capture all the essences of the principle. For instance, given the alphabet $\Sigma = \{a, b, c, d\}$ and the dictionary $D = \{a, b, c, d, ab, bc, cd\}$, the character string $S = abcd$ has the same tokenization set $F_D(S) = B_D(S) = S_D(S) = \{ab/cd\}$ for FT, BT and ST, but a different CT tokenization set $C_D(S) = \{ab/cd, a/bc/d\}$. In other words, the CT tokenization $a/bc/d$ is left out in all the other three sets. As the tokenization $a/bc/d$ is not a subtokenization of any other possible tokenizations, it fulfills the principle of maximum tokenization.

It is now clear that, while the principle of maximum tokenization is very useful in sentence tokenization, it lacks *precise* understanding in the literature. Consequently, no solution proposed in the literature is complete with regards to realizing the principle.

Recall that, in the previous sections, the character string tokenization operation was modeled as the inverse of the generation operation. Under the tokenization operation, every character string can be tokenized into a set of different tokenizations. The cover relationship between tokenizations was recognized and the set of tokenizations was proven to be a poset (partially ordered set) on the cover relationship. The set of critical tokenizations was defined as the set of minimum elements in the poset. In addition, it was proven that every tokenization has at least one critical tokenization as its supertokenization and only critical tokenization has no true supertokenization.

Consequently, a noncritical tokenization would conflict with the principle of maximum tokenization, since it is a true subtokenization of others. As compared with its true supertokenization, it requires the extra effort of subtokenization. On the other hand, a critical tokenization would fully realize the principle of maximum tokenization, since it has already attained an extreme form and cannot be simplified or compressed further. As compared with all other tokenizations, no effort can be saved.

Based on this understanding, it is now apparent why forward maximum tokenization, backward maximum tokenization, and shortest tokenization are all special cases of critical tokenization, but not vice versa. In addition, it has been proven, in Guo (1997), that critical tokenization also covers other types of maximum tokenization implementations such as profile tokenization and shortest tokenization.

We believe that critical tokenization is the *only* type of tokenization completely fulfilling the principle of maximum tokenization. In other words, critical tokenization is the precise mathematical description of the commonly adopted principle of maximum tokenization.

7. Further Discussion

This section explores some helpful implications of critical tokenization in effective tokenization disambiguation and in efficient tokenization implementation.

---

desired tokenization, in many contexts, is "结合/成/分子".

7.1 String Generation and Tokenization versus Language Derivation and Parsing

The relationship between the operations of sentence derivation and sentence parsing in the theory of parsing, translation, and compiling (Aho and Ullman 1972) is an obvious analogue with the relationship between the operations of character string generation and character string tokenization that are defined in this paper. As the former pair of operations is well established, and has great influence in the literature of sentence tokenization, many researchers have, either consciously or unconsciously, been trying to transplant it to the latter. We believe this worthy of reexamination.

Normally, sentence derivation and parsing are governed by complex grammars. Consequently, the bulk of the work has been in developing, representing, and processing grammar. Although it is a well known fact that some sentences may have several derivations or parses, the focus has always been either on (1) grammar enhancement, such as introducing semantic categories and consistency checking rules (selectional restrictions), not to mention those great works on grammar formalisms, or on (2) ambiguity resolution, such as introducing various heuristics and tricks including leftmost parsing and operator preferences (Aho and Ullman 1972; Aho, Sethi, and Ullman 1986; Allen 1995; Grosz, Jones, and Webber 1986).

Following this line, we observed two tendencies in tokenization research. One is the tendency to bring every possible knowledge source into the character string generation operation. For example, Gan (1995) titled his Ph.D. dissertation *Integrating Word Boundary Disambiguation with Sentence Understanding*. Here, in addition to traditional devices such as syntax and semantics, he even employed principles of psychology and chemistry, such as crystallization. Another is the tendency of enumerating almost blindly every heuristic and trick possible in ambiguity resolution. As Webster and Kit (1992, 1108) noted, "segmentation methods were invented one after another and seemed inexhaustible." For example, Chen and Liu (1992) acknowledged that the heuristic of maximum matching alone has "many variations" and tested six different implementations.

We are not convinced of the effectiveness and necessity of both of the schools of tokenization research. The principle argument is, while research is by nature trial-and-error and different knowledge sources contribute to different facets of the solution, it is nonetheless more crucial and productive to understand where the core of the problem really lies.

As depicted in this paper, unlike general sentence derivation for complex natural languages, the character string generation process can be very simple and straightforward. Many seemingly important factors such as natural language syntax and semantics do not assume fundamental roles in the process. They are definitely helpful, but only at a later stage. Moreover, as emphasized in this paper, the tokenization set has some very good mathematical properties. By taking advantage of these properties, the tokenization problem can be greatly simplified. For example, among the huge number of possible tokenizations, we can first concentrate on the much smaller critical tokenization set, since the former can be completely reproduced from the latter. Furthermore, by contrasting critical tokenizations, we can easily identify a few critically ambiguous positions, which allows us to avoid wasting energy at useless positions.

7.2 Critical Tokenization and the Syntactic Graph

It is worth noting that similar ideas do exist in natural language derivation and parsing. For example, Seo and Simmons (1989) introduced the concept of the syntactic graph, which is, in essence, a union of all possible parse trees. With this graph representation, "it is fairly easy to focus on the *syntactically ambiguous points*" (p. 19, italics added).

These syntactically ambiguous points are critical in at least two senses. First, they are the only problems requiring knowledge and heuristics beyond the existing syntax. In other words, any syntactic or semantics development should be guided by ambiguity resolution at these points. If a semantic enhancement does not interact with any of these points, the enhancement is considered ineffective. If a grammar revision in turn leads to additional syntactically ambiguous points, such a revision would be in the wrong direction.

Second, these syntactically ambiguous points are critical in efficiently resolving ambiguity. After all, these points are the only places where disambiguation decisions must be made. Ideally, we should invest no energy in investigating anything that is irrelevant to these points. However, unless all parse trees are merged together to form the syntactic graph, the only thing feasible is to check every possible position in every parse tree by applying all available knowledge and every possible heuristic, since we are unaware of the effectiveness of any checking that occurs beforehand.

The critical tokenization introduced in this paper has a similar role in string tokenization to that of the syntactic graph in sentence parsing. By Theorem 3, critical tokenization is, in essence, the union of the whole tokenization set and thus the compact representation of it. As long as the principle of maximum tokenization is accepted, the resolution of critical ambiguity in tokenization is the only problem requiring knowledge and heuristics beyond the existing dictionary. In other words, any introduction of "high-level" knowledge must at least be effective in resolving some critical ambiguities in tokenization. This should be a fundamental guideline in tokenization research.

Even if the principle of maximum tokenization is not accepted, critical ambiguity in tokenization must nevertheless be resolved. Therefore, any investment, as mentioned above, will not be a waste in any sense. What needs to be undertaken now is to substitute something more precise for the principle of maximum tokenization. It is only at this stage that we touch on the problem of identifying and resolving hidden ambiguity in tokenization. That is one of the reasons why this type of ambiguity is called *hidden*.

7.3 Critical Tokenization and Best-Path Finding

The theme in this paper is to study the problem of sentence tokenization in the framework of formal languages, a direction that has recently attracted some attention. For instance, in Ma (1996), words in a tokenization dictionary are represented as production rules and character strings are modeled as derivatives of these rules under a string concatenation operation. Although not stated explicitly in his thesis, this is obviously a finite-state model, as evidenced from his employment of (finite-) state diagrams for representing both the tokenization dictionary and character strings. The weighted finite-state transducer model developed by Sproat et al. (1996) is another excellent representative example.

They both stop at merely representing possible tokenizations as a single large finite-state diagram (word graph). The focus is then shifted to the problem of defining scores for evaluating each possible tokenization and to the associated problem of searching for the best-path in the word graph. To emphasize this point, Ma (1996) explicitly called his approach "evaluation-based."

In comparison, we have continued within the framework and established the critical tokenization together with its interesting properties. We believe the additional step is worthwhile. While tokenization evaluation is important, it would be more effective if employed at a later stage.

On the one hand, critical tokenization can help greatly in developing tokenization knowledge and heuristics, especially those tokenization specific understandings, such

as the observation of "one tokenization per source" and the trick of highlighting hidden ambiguities by contrasting competing critical tokenizations (Guo 1997).

While it may not be totally impossible to fully incorporate such knowledge and heuristics into the general framework of path evaluation and searching, they are apparently employed neither in Sproat et al. (1996) nor in Ma (1996). Further, what has been implemented in the two systems is basically a token unigram function, which has been shown to be practically irrelevant to hidden ambiguity resolution and not to be much better than some simple maximum tokenization approaches such as shortest tokenization (Guo 1997).

On the other hand, critical tokenization can help significantly in boosting tokenization efficiency. As has been observed, the tokenization of about 98% of the text can be completed in the first parse of critical point identification, which can be done in linear time. Moreover, as practically all acceptable tokenizations are critical tokenizations and ambiguous critical fragments are generally very short, the remaining 2% of the text with tokenization ambiguities can also be settled efficiently through critical tokenization generation and disambiguation (Guo 1997).

In comparison, if the best path is to be searched on the token graph of a complete sentence, while a simple evaluation function such as token unigram cannot be very effective in ambiguity resolution, a sophisticated evaluation function incorporating multiple knowledge sources, such as language experiences, statistics, syntax, semantics, and discourse as suggested in Ma (1996), can only be computationally prohibitive, as Ma himself acknowledged.

In summary, the critical tokenization is *crucial* both in knowledge development for effective tokenization disambiguation and in system implementation for complete and efficient tokenization. Further discussions and examples can be found in Guo (1997).

8. Summary

The objective in this paper has been to lay down a mathematical foundation for sentence tokenization. As the basis of the overall mathematical model, we have introduced both sentence generation and sentence tokenization operations. What is unique here is our attempt to model sentence tokenization as the inverse problem of sentence generation.

Upon that basis, both critical point and critical fragment constitute our first group of findings. We have proven that, under a complete dictionary assumption, critical points in sentences are all and only unambiguous token boundaries.

Critical tokenization is the most important concept among the second group of findings. We have proven that every tokenization has a critical tokenization as its supertokenization. That is, any tokenization can be reproduced from a critical tokenization.

Critical ambiguity and hidden ambiguity in tokenization constitute our third group of findings. We have proven that tokenization ambiguity can be categorized as either critical type or hidden type. Moreover, it has been shown that critical tokenization provides a sound basis for precisely describing various types of tokenization ambiguities.

In short, we have presented a complete and precise understanding of ambiguity in sentence tokenizations. While the existence of tokenization ambiguities is jointly described by critical points and critical fragments, the characteristics of tokenization ambiguities will be jointly specified by critical ambiguities and hidden ambiguities. Moreover, we have proven that the three widely employed tokenization algorithms, namely forward maximum matching, backward maximum matching, and shortest

length matching, are all subclasses of critical tokenization and that critical tokenization is the precise mathematical description of the principle of maximum tokenization.

In this paper, we have also discussed some important implications of the notion of critical tokenization in the area of character string tokenization research and development. In this area, our primary claim is that critical tokenization is an excellent intermediate representation that offers much assistance both in the development of effective tokenization knowledge and heuristics and in the improvement and implementation of efficient tokenization algorithms.

Besides providing a framework to better understand previous work, as has been attempted here, a good formalization should also lead to new questions and insights. While some of the findings and observations achieved so far (Guo 1997) have been mentioned here, much more work remains to be done.

Acknowledgments

The author would like to thank Ho-Chung Lui for his supervision, and Kok-Wee Gan, Zhibiao Wu, Zhendong Dong, Paul Horng Jyh Wu, Kim-Teng Lua, Chunyu Kit, and Teow-Hin Ngair for many helpful discussions. The author is also very grateful to four anonymous reviewers for their insightful comments on earlier versions of the paper. Alexandra Vaz Hugh and Ng Chay Hwee helped in correcting grammar.

References

Aho, Alfred V., R. Sethi, and Jeffrey D. Ullman. 1986. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley Publishing Co.

Aho, Alfred V. and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling, Volume 1: Parsing*. Prentice-Hall, Inc.

Allen, James. 1995. *Natural Language Understanding*, 2nd edition. The Benjamin/Cummings Publishing Co.

Bai, Shuanhu. 1995. An integrated model of Chinese word segmentation and part of speech tagging. In Liwei Chen and Qi Yuan, editors, *Advances and Applications on Computational Linguistics*. Tsinghua University Press, pages 56–61.

Chen, Keh-Jiann and Shing-Huan Liu. 1992. Word identification for Mandarin Chinese sentences. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 101–107. Nantes, France.

Chiang, Tung-Hui, Jing-Shin Chang, Ming-Yu Lin, and Keh-Yih Su. 1992. Statistical models for word segmentation and unknown word resolution. In *Proceedings of the 5th R.O.C. Computational Linguistics Conference (ROCLING V)*, pages 121–146, Taiwan.

Fan, C-K. and W-H. Tsai. 1988. Automatic word identification in Chinese sentences by the relaxation technique. *Computer Processing of Chinese and Oriental Languages* 4(1):33–56.

Gan, Kok-Wee. 1995. *Integrating Word Boundary Disambiguation with Sentence Understanding*. Ph.D. dissertation, Department of Computer Science and Information Systems, National University of Singapore.

Gan, Kok-Wee, Martha Palmer, and Kim-Teng Lua. 1996. A statistically emergent approach for language processing: Application to modeling context effects in ambiguous Chinese word boundary perception. *Computational Linguistics* 22(4):531–553.

Garside, Roger, Geoffrey Leech, and Geoffrey Sampson, editors. 1987. *The Computational Analysis of English: A Corpus-based Approach*. Longman, London.

Gazdar, G., E. Klein, G. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.

Grosz, B. J., K. S. Jones, and B. L. Webber, editors. 1986. *Readings in Natural Language Processing*. M. Kaufmann Publishers.

Guo, Jin. 1993. Statistical language modeling and some experimental results on Chinese syllables to words transcription. *Journal of Chinese Information Processing* 7(1):18–27.

Guo, Jin. 1997. *Chinese Language Modeling for Speech Recognition*. Ph.D. dissertation, Institute of Systems Science, National University of Singapore.

He, Kekang, Hui Xu, and Bo Sun. 1991. Design principles of an expert system for automatic word segmentation of written Chinese texts. *Journal of Chinese Information Processing* 5(2):1–14.

Huang, Xiangxi. 1989. A produce-test approach to automatic segmentation of written Chinese. *Journal of Chinese Information Processing* 3(4):42–49.

Huang, Changning and Ying Xia, editors.

1996. *Essays on Language Information Processing*. Tsinghua University Press, Beijing.
Hudson, R. A. 1984. *Word Grammar*. Basil Blackwell.
Jie, Chunyu. 1989. A systematic approach model for methods of Chinese automatic word segmentation and their evaluation. In *Proceedings of the Chinese Computing Conference*, pages 71–78, Beijing.
Jie, Chunyu, Yuan Liu, and Nanyuan Liang. 1991a. On the methods of Chinese automatic segmentation. *Journal of Chinese Information Processing* 3(1):1–9.
Jie, Chunyu, Yuan Liu, and Nanyuan Liang. 1991b. The design and implementation of the CASS practical automatic Chinese word segmentation system. *Journal of Chinese Information Processing* 5(4):27–34.
Jin, Wanying and Lu Chen. 1995. Identifying unknown words in Chinese corpora. In *Proceedings of the 1995 Natural Language Processing Pacific Rim Symposium (NLPRS'95)*, pages 234–239, Seoul.
Kolman, Bernard and Robert C. Busby. 1987. *Discrete Mathematical Structures for Computer Science*, 2nd edition. Prentice-Hall, Inc.
Lai, T. B. Y., S. C. Lun, C. F. Sun, and M. S. Sun. 1992. A tagging-based first-order Markov model approach to automatic word identification for Chinese sentences. In *Proceedings of the 1992 International Conference on Computer Processing of Chinese and Oriental Languages*, pages 17–23.
Li, Wen-Jie, H-H. Pan, M. Zhou, K-F. Wong, and V. Lum. 1995. Corpus-based maximum-length Chinese noun phrase extraction. In *Proceedings of the 1995 Natural Language Processing Pacific Rim Symposium (NLPRS'95)*, pages 246–251, Seoul.
Liang, Nanyuan. 1986. On computer automatic word segmentation of written Chinese. *Journal of Chinese Information Processing* 1(1).
Liang, Nanyuan. 1987. CDWS—A written Chinese automatic word segmentation system. *Journal of Chinese Information Processing* 1(2):44–52.
Liang, Nanyuan. 1990. The knowledge of Chinese words segmentation. *Journal of Chinese Information Processing* 4(2):29–33.
Liu, Yongquan. 1986a. *Language Modernization and Computer*. Wuhan University Press, Wuhan, China.
Liu, Yongquan. 1986b. On dictionary. *Journal of Chinese Information Processing* 1(1).
Liu, Yongquan. 1988. Word re-examination. *Journal of Chinese Information Processing* 2(2):47–50.
Liu, Yuan, Qiang Tan, and Xukun Shen. 1994. *Contemporary Chinese Language Word Segmentation Specification for Information Processing and Automatic Word Segmentation Methods*. Tsinghua University Press, Beijing.
Lua, Kim Teng. 1990. From character to word—An application of information theory. *Computer Processing of Chinese and Oriental Languages* 4(4):304–313.
Lua, Kim Teng. 1994. Application of information theory binding in word segmentation. *Computer Processing of Chinese and Oriental Languages* 8(1):115–124.
Lua, Kim Teng. 1995. Experiments on the use of bigram mutual information in Chinese natural language processing. In *Proceedings of the 1995 International Conference on Computer Processing of Oriental Languages (ICCPOL-95)*, pages 306–313, Hawaii.
Ma, Yan. 1996. The study and realization of an evaluation-based automatic segmentation system. In Changning Huang and Ying Xia, editors, *Essays in Language Information Processing*. Tsinghua University Press, Beijing, pages 2–36.
Nie, Jieyun, Jin Wanying, and M-L. Hannan. 1994. A hybrid approach to unknown word detection and segmentation of Chinese. In *Proceedings of the International Conference on Chinese Computing 1994 (ICCC-94)*, pages 326–335, Singapore.
Pachunke, T., O. Mertineit, K. Wothke, and R. Schmidt. 1992. Broad coverage automatic morphological segmentation of German words. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 1218–1222, Nantes, France.
Seo, J. and R. F. Simmons. 1989. Syntactic graphs: A representation for the union of all ambiguous parse trees. *Computational Linguistics* 15(1):19–32.
Sproat, Richard and Chilin Shih. 1990. A statistical method for finding word boundaries in Chinese text. *Computer Processing of Chinese and Oriental Languages* 4(4):336–349.
Sproat, Richard, Chilin Shih, William Gale, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics* 22(3):377–404.
Sun, Maosong and Changning Huang. 1996. Word segmentation and part-of-speech tagging for unrestricted Chinese texts. Tutorial given at the 1996 International Conference on Chinese Computing (*ICCC-96*), Singapore.

Sun, Maosong and Benjemin T'sou. 1995. Ambiguity resolution in Chinese word segmentation. In *Proceedings of the 10th Pacific Asia Conference on Language, Information and Computation (PACLIC-95)*, pages 121–126, Hong Kong.

Tung, C. H. and H. J. Lee. 1994. Identification of unknown words from corpus. *Computer Processing of Chinese and Oriental Languages* 8(Supplement):131–146.

Wang, Yongcheng, Haiju Su, and Yan Mo. 1990. Automatic processing Chinese word. *Journal of Chinese Information Processing* 4(4):1–11.

Wang, Xiaolong. 1989. *Word Separating and Mutual Translation of Syllable and Character Strings*. Ph.D. dissertation, Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China.

Wang, Xiaolong, Kaizhu Wang, and Xiaohua Bai. 1991. Separating syllables and characters into words in natural language understanding. *Journal of Chinese Information Processing* 5(3):48–58.

Webster, Jonathan J. and Chunyu Kit. 1992. Tokenization as the initial phase in NLP. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 1,106–1,110, Nantes, France.

Wong, K-F., H-H. Pan, B-T. Low, C-H. Cheng, V. Lum, and S-S. Lam. 1995. A tool for computer-assisted open response analysis. In *Proceedings of the 1995 International Conference on Computer Processing of Oriental Languages*, pages 191–198, Hawaii.

Wong, K. F., V. Y. Lum, C-Y. Leung, C-H. Leung, W-K. Kan, and L-C. Chan. 1994. A parallel approach for identifying word boundaries in Chinese text. IPOC Technical Report, /CHIRP/WP/SE/022, Department of Systems Engineering, Chinese University of Hong Kong.

Wu, Horng Jyh, Jin Guo, Ho Chung Lui, and Hwee Boon Low. 1994. Corpus-based speech and language research in the Institute of Systems Science. In *Proceedings of the International Symposium on Speech, Image Processing and Neural Networks (ISPIPNN'94)*, pages 142–145, Hong Kong.

Wu, Ming-Wen and Keh-Yih Su. 1993. Corpus-based automatic compound extraction with mutual information and relative frequency count. In *Proceedings of R.O.C. Computational Linguistics Conference (ROCLING) VI*, pages 207–216, Taiwan.

Yao, Tian-Shun, Gui-Ping Zhang, and Ying-Ming Wu. 1990. A rule-based Chinese automatic segmentation system. *Journal of Chinese Information Processing* 4(1):37–43.

Yeh, C-L. and H-J. Lee. 1991. Rule-based word identification for Mandarin Chinese sentences—A unification approach. *Computer Processing of Chinese and Oriental Languages* 5(2):97–118.

Yosiyuki, K., T. Takenobu, and T. Hozumi. 1992. Analysis of Japanese compound nouns using collocation information. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 865–869, Nantes, France.

Yun, B-H., H. Lee, and H-C. Rim. 1995. Analysis of Korean compound nouns using statistical information. In *Proceedings of the 1995 International Conference on Computer Processing of Oriental Languages (ICCPOL-95)*, pages 76–79, Honolulu, Hawaii.

Zhang, Jun-Sheng, Zhi-Da Chen, and Shun-De Chen. 1991. A method of word identification for Chinese by constraint satisfaction and statistical optimization techniques. In *Proceedings of R.O.C. Computational Linguistics Conference (ROCLING) IV*, pages 147–165, Taiwan.

Zhang, Jun-Sheng, Shun-De Chen, S. J. Ker, Y. Chan, and J. S. Liu. 1994. A multi-corpus approach to recognition of proper names in Chinese texts. *Computer Processing of Chinese and Oriental Languages* 8(1):73–86.