STRING COMPARISON BASED ON SUBSTRING EQUATIONS

Kyoji Umemura

Nippon Telegraph and Telephone, Basic Research Laboratories
Suite 4S222S, NTT, 3-1 Morinosato-Wakamiya, Atsugi, Kanagawa 243-01, Japan

Abstract

This paper describes a practical method to compute whether two strings are equivalent under certain equations. This method uses a procedure called Critical-Pair/Completion. that generates rewriting rules from equations. Unlike other Critical-Pair/Completion procedures, the procedure described here always stops for all equations because it treats strings of bounded length. This paper also explains the importance of the string equivalence problem if international data handling is required.

1. Introduction

The equality test is a fundamental operation. There are various kinds of equality functions. One function (i.e. STRING=) distinguishes lower case and upper case letter. Another function (i.e. STRING-EQUAL) ignores the case. Different kinds of equality test functions are necessary to handle various languages. For example, Japanese has Hiragana and Katanaka character sets. Although they may have different roles in written form, corresponding characters have identical pronunciations. This situation is very similar to lower and upper case letters in English. Therefore it is beneficial for Japanese people to have a function that ignores the Hiragana-Katakana difference. Futhermore, if the Japanese data is expressed in alphabet form, the equivalence is not simply between character and character, but between string and string. Other language may have same problem. This '*case-insensitive*' is not a simple problem for international data handling.

The case-insensitive comparison is the comparison under equations such as "A"="a", "B"="b", "C"="c", ... and "Z"="z". Since equations like "cha"="tya" is necessary in Japanese language, both sides of the equations should not be limited to one character. We can perform the case-insensitive comparison by replacing all upper case characters with the corresponding lowercase characters. In general, we compute equivalence by converting strings using some rules, replacing all of the substrings, obtaining canonical form and then comparing them exactly. These rules are usually generated by hand. It is a difficult task to generate these rules correctly for complex cases.

Suppose we have a equation set { "abc"="ab", "abc"="bc" }. The following strings are equivalent:"xabcabcy", "xababy", "xbcbcy", and "xabbcy". The rule set { "abc"->"ab", "abc"->"bc" }

is not correct since it is not confluent; For example "xabcy", may have two different results. The rule set {"ab" -> "abc", "bc" -> "abc"} is not usable since the conversion never stops. Though the rule {"abc"->"bc", "bc"->"ab"} looks adequate, the rule is not suitable for "xaaby" and "xbcy". The string "xaaby" is equivalent to "xbcy" because "xaaby" = "xaabcy" ="xabcy" = "xbcy". However, "xaaby" and "xbcy" are converted to "xaaby" and "xaby" using the rule set { "abc"->"ab", "bc"->"ab"}. The correct rule set that the procedure generates is {"aab"->"ab", "bc"->"ab"}.

In simple cases, like case-insensitive comparisons, the corresponding conversion rules are simple and apparent. However, the procedure that generates these rules from the equations are not simple. The procedure is one variation of the Critical-Pair / Completion procedure, which originally treats term-rewriting-systems. It is well known that Critical-Pair / Completion sometimes fails to stop. Even if we limit the domain to strings, the Critical-Pair / Completion procedure may not stop. For example, the equation {"aba"="ab"} will generate an infinite number of rules such as { "aba"->"ab", "abba" -> "abb", "abbba" -> "abbb" ... }. Since failure to stop is fatal for any application, we need a domain in which procedures always stops.

This paper formalizes the problem where strings are bounded in length. This problem is practical since the strings in computer system are bounded in length. This paper describes this variation of Critical-Pair/Completion procedure[Buchberger87]. This variation stops for all given equation. These characteristics are important for the actual application.

2. The Problem Definition

In this section, we define the problem to be solved.

N: N is a given number. N is the maximum length of any string.

C: C is a given finite set of characters. Any character has corresponding integer value.

String: String is a vector of the member of C.

N: SN is a set of all strings whose length is less than N. It includes the null string, which is the string whose length is 0.

G: G is a given set of equations among members of SN that defines the equivalence relation. An example of G is { "abc" = "bc", "abc" = "ab" }.

Substitution: Each of α, β, and γ is member of SN. Suppose we have a equation g in which $\alpha=\beta$ or $\beta=\alpha$, where α is a substring of γ. If there is a string δ that replaces the substring of γ with β and the length of δ is less than N, we call δ the substitution of γ with g.

Neighborhood:When α and β are the member of SN, If there is a equation $\exists$g that is member of G and β is a substitution of α with g, we call α as being in the neighborhood of β. We write $\alpha \leftrightarrow \beta$.

Equivalence:When α and β are the member of SN, $\exists\alpha_1, \exists\alpha_2, \ldots, \exists\alpha_n$ are members of of SN, and where $\alpha \leftrightarrow \alpha_1$, $\alpha_1 \leftrightarrow \alpha_2$, ..., $\alpha_{n-1} \leftrightarrow \alpha_n$, $\alpha_n \leftrightarrow \beta$, then we write $\alpha \equiv_N \beta$

Our problem is to compute whether or not $\alpha \equiv_N \beta$ for given N, G, C. The important limitation is that

search space is finite. If there is no limitation to the length, the problem is called word problem. The word problem is known to be undecidable. This means there is no algorithm that always stops. Our problem is decidable since it is a coloring problem of finite set. We might do as follows.

Suppose each of α and β is an element of SN,
1. Let S be { α }.
2. Do following while S grows.
   For all δ that are elements of S, get the neighborhood of δ. Then add to S.
3. If β is a member of S, then $\alpha \equiv_N \beta$, otherwise not $\alpha \equiv_N \beta$.

SN is determined by N and C. Since C is finite, SN is also finite and its size is $size(C)^N$.
Since S always grows and S is a subset of a finite set (SN), theoretically, this procedure always stops. It is, however, not usable in the practical sense because S may become too large to be stored in memory. Although the word problem on finite sets may not be theoretically interesting, it is still a challenge to compute it using actual computers.

3. The Order

The order of strings specifies Critical-Pair/Completion procedure. It is natural to convert the strings into simple form. For example, if we have equation set { "abc"="a" }, we will replace all of "abc" with "a", and not "a" with "abc". The order is used to compare strings, and thus determines the meaning of *simple*. We should carefully choose the order so that the conversion will always stop.

The order that we chose is as follows:
(1) If the two strings differ in length, the shorter string is simpler.
(2) If both strings are identical, neither is simpler.
(3) If the two strings are identical in length, compare the first character using the corresponding integer value. The smaller one is simpler.
(4) Otherwise, compare the strings that begin with the second character.

For the given rule w1->w2, where w2 is simpler than w1 by this definition, we call that *simplifying*. If all rules are simplifying, the transformation result will always be simpler than the original form. Because there is the simplest string (null string), the transformation will stop.

4. The Critical-Pair

Critical-Pair is a pair of strings generated from transformation rules. Both of the strings are equivalent under the original equations, but they have different transformation results. For example, if we have a rule set, {"x"->"b", "x"->"c"}, the pair ("b", "c") is a critical pair. Critical-Pair/Completion procedure generates the additional rule "c"->"b" so that the equivalent strings will always become the same in the final form.

There is a more complex case. For example, if we have the rule set {"ab"->"x", "bc"->"y"}, the pair ("xc", "ay") are equivalent because "xc" = "abc" = "ay". Since both "xc" and "ay" are different transformation results of "abc", the pair ("xc", "ay") is a critical pair of {"ab"->"x", "bc"->"y"}.

The following procedure generates the critical pair of length bounded strings:

Suppose we have rules r1 and r2, where r1 is written as w1->w2, and r2 is written as w3->w4.
(1) Find all of $w \in SN$,
where $w=\alpha\chi\beta$, $\chi\neq$"", and $(w1, w3) \in \{ (\alpha\chi, \chi\beta), (\alpha\chi\beta, \chi), (\chi\beta, \alpha\chi), (\chi, \alpha\chi\beta)\}$
(2) If such w does not exist, the result is empty.
(3) For each w, found in step 1,transform w with r1 and r2. Let the result be w5 and w6 respectively, add the pair (w5, w6) to the result.

Though this procedure generates all of the Critical-Pairs, some of them are unnecessary for the bounded length problem. Although we do not generates the pair if w is longer than N, there may be a chance where equivalence relation may have intermediate string that is longer than N. As the result, the generated rule may have unnecessary relations where intermediate string are longer than N. Nevertheless, it is important that w is a member of SN. This implies that the length of w is bounded. Without this limitation, the Critical-Pair / Completion procedure may not stop.

5. The Procedure

Although the order and the Critical-Pair may differ, the completion procedure is the same among various problems. The correctness of this procedure is described by Huet [Huet 81]. The following is a description of Critical-Pair/Completion procedure.

Let R be a set of rules, G be a set of equivalence equations.
Set G as initial equations, and R as an empty set.
Do the following:
(0) If G is empty, then R is the obtained rule sets.
(1) Select one element w1=w2 from G and remove it from G.
(2) If w1 and w2 are identical, go to (0).
(3) If w1 is simpler than w2, swap w1 and w2.
(4) Let r be w1->w2.
(5) For all elements w3->w4 of R, if w3 is transformable by r, remove it from R and add w3=w4 into G.
(6) For all elements w3->w4 of R, generate all the Critical-Pair's {(w5, w6)...} from w1->w2 and w3->w4. Add { w5=w6, ...} into G.
(7) Add w1->w2 into R.
(8) For all elements w3->w4 of R, transform w4 into w5 using R, and then replace the rule as w3->w5.

(9) For all elements w3=w4 of G, transform w3 and w4 into w5 and w6 using R, and then replace the equation as w5=w6

G:{"abc"="ab",	"abc"="bc"},	R:{}	; loop 0, step 0
G:{	"abc"="bc"},	R:{"abc"->"ab"}	; loop 0, step 8
G:{	"ab"="bc"} ,	R:{"abc"->"ab"}	; loop 1, step 0
G:{"abc"="ab"},		R:{"bc"->"ab"}	; loop 1, step 8
G:{"aab"="ab"},		R:{"bc"->"ab"}	; loop 2, step 0
G:{"aaab"="ab"},		R:{"bc"->"ab", "aab"->"ab"}	; loop 2, step 8
G:{"ab"="ab"},		R:{"bc"->"ab", "aab"->"ab"}	; loop 3, step 0
G:{}		R:{"bc"->"ab", "aab"->"ab"}	; loop 4, step 0

Fig. 1 : An execution trace.

Steps from (0) to (7) convert one equation to one rule, and add some equation so that the information may not be lost. Steps(8) and (9) simplify the R and G without changing equivalence relations. Fig 1 illustrates the process of procedure for {"abc"="ab", "abc"="bc"} and N > 3.

This procedure stops if steps(6)-(9) do not generate new relation. This means that the procedure either stops or generates new equations. Since the number of equation is finite, the number of all the possible Critical-Pairs is always finite. If this procedure generate enough Critical-Pairs, it will stop generating equations. Then the procedure will stop.

6. String Comparison

For the given equation set G and integer N, this procedure generates corresponding rules R. To compare two strings w1 and w2, first transform both of them using R, then compare the results. If they are not identical, w1 and w2 are not equivalent. If results are identical, w1 and w2 are equivalent for unbounded string domains. However, w1 and w2 may or may not be identical in bounded length.

Suppose, G is {"abc"="ab", "abc"="bc"},N is 3, w1is "aab" and w2 is "abc". This procedure will generate R:{"bc"->"ab", "aab"->"ab"}, then w1 and w2 are transformed into same string "ab". However "aab" and "abc" are not equivalent when N is 3. If N is greater than 3, "aab" and "abc" are equivalent since "aab" = "aabc" = "abc"; however if N is 3, "aabc" is not member is SN because its length is 4.

This means that even the translated results are identical, the original strings may not be equivalent in bounded length. Fortunately, this does not cause any problem in actual application because they are equivalent if the length is not limited.

7. Application

This comparison is applicable when the information is translated from other languages into alphabetical form. For example, if some data comes from Japanese, the name "Ito", "Itou" and "Itoo" may be the same name. Japanese language has a simpler phonetic system than English. As the result, {"chi"="ti", "ci"="si", "shi"="si", "fu"="hu", "fa"="fua", ...} are reasonable equations. Fig. 2 illustrates one example of the Japanese sound system and Fig. 3 shows the obtained rules.

It is interesting to note that original equations has two concepts -- case insensitive and Japanese sound system. The generated procedure satisfies both concepts. It is not always an easy task to mix two equivalence policies at the same time. Since equivalence policy is described in equations, it can be mixed easily in this framework.

```
{"cha"="tya", "chi"="ti", "texi"="ti", "chu"="tyu", "che"="tixe", "cho"="tyo",
"sha"="sya", "shi"="si", "suxi"="si", "shu"="syu", "she"="sixe", "sho"="syo",
"ja"="jya", "ji"="zi", "zuxi"="zi", "ju"="jyu", "je"="zixe", "jo"="jyo", "fa"="huxa",
"fi"="huxi", "fu"="hu", "fe"="huxe", "fo"="fuxo", "ky"="kiy", "sy"="siy", "ty"="tiy",
"ny"=niy", "hy"="hiy", "my"="miy", "ry"="riy", "gy"="giy", "dy"="diy", "by"="biy",
"py"="piy", "a-"="a", "i-"="i", "u-"="u", "e-"="e", "o"="ou", "xa"="a", "xi"="i",
"xu"="u", "xe"="e", "xo"="o", "nn"="n", "xn"="n", "A"="a", "B"="b", "C"="c",
"D"="d", "E"="e", "F"="f", "G"="g", "H"="h", "I"="i", "J"="j", "K"="k", "L"="l",
"M"="m", "N"="n", "O"="o", "P"="p", "Q"="q", "R"="r", "S"="s", "T"="t", "U"="u",
"V"="v", "W"="w", "X"="x", "Y"="y", "Z"="z" }
```

Fig. 2: equations for Japanese pronounciation

```
{"a"->"A", "b"->"B", "c"->"C", "d"->"D", "e"->"E", "f"->"F", "g"->"G", "h"->"H",
"i"->"I", "j"->"J", "k"->"K", "l"->"L", "m"->"M", "n"->"N", "o"->"O", "p"->"P", "q"->"Q",
"r"->"R", "s"->"S", "t"->"T", "u"->"U", "v"->"V", "w"->"W", "x"->"X", "y"->"Y",
"z"->"Z", "A-"->"A", "E-"->"E", "HU"->"FU", "I-"->"I", "NN"->"N", "O-"->"O",
"OU"->"O", "U-"->"U","XA"->"A", "XE"->"E", "XI"->"I", "XA"->"A", "XE"->"E",
"XI"->"I", "XN"->"N", "XO"->"O", "XU"->"U", "ZI"->"JI", "BIY"->"BY", "CHI"->TI",
"CHY"->"TI", "CHY"->"TY", "DIY"->"DY", "FUA"->"FA", "FUE"->"FE", "FUI"->"FI",
"FUO"->"FO", "GIY"->"GY", "HIY"->"HY", "JIE"->"JE", "JYA"->"JA", "JYO"->"JO",
"JYU"->"JU", "KIY"->"KY", "MIY"->"MY", "NIY"->"NY", "PIY"->"PY", "RIY"->"RY",
"SHI"->"SI", "SHY"->"SY", "SIE"->"SHE", "SIY"->"SY", "SUI"->"SI", "SYA"->"SHA",
"SYO"->"SHO", "SYU"->"SFU", "TEI"->"TI", "TIE"->"CHE", "TYA"->"CHA",
"TYO"->"CHO", "TYU"->"CFU", "JUI"->"JI"}
```

Fig. 3: transformation rules for Japanese pronounciation

8. The Importance

Small data variances cause the failure of information retrieval. This variance frequently happens when original data comes from non-English languages. The Japanese name is a clear example. If the lanaguge has non-alphabetical writing, the mapping between original character to alphabet form may not be one-to-one mapping. The word problem is a practical problem in this situation. Although superfluous data may be equivalent, we can reject this data after the selection. The failure of retrival is more problematic than superfluous output.

9. Related Works and Future Works

Knuth [Knuth 70] introduced this procedure and applied it to term-rewriting-system. Buchberger [Buchberger 87] surveyed the overall characteristics of Critical-Pair/Completion procedure. Book [Book 87] explained the relationship between string and term-rewriting-system. This paper is based on these works. Our contribution is to find the problem that is simple enough to be always solved, and that is powerful enough to be applied actual information systems.

We are applying these conversion rules to pick up telephone directory by names. People's name may have data specific equations. For example, "kamihayashi", and "kambayashi" may be the same person because the sound is changed due to the combination of "kami" and "hayashi". These strings are equation candidates in actual data retrieval. In this case, the number of equations is in the thousands.

In the future, it will be important to calculate computational complexity. Although we briefly explained that the procedure will stop, the worst case is still the exponential order of N. This would be no better than simple coloring procedures although our experience shows that our procedure actually ends in reasonable amounts of time.

10. Conclusion

This paper formulated Critical-Pair/Completion procedure for bounded length strings. This procedure generates rewriting rules from substring equations. We can test string equivalence under certain equations using these rules. This paper also explains that the string equivalence is an important problem for international data handling.

Acknowledgement

The author thanks Yoshihito Toyama(JAIST) and Hirofumi Katsuno (NTT) for his discussion and guidance about Critical-Pair / Completion procedure. The author also thanks Katsumi Takahashi (NTT) for his discussion about actual information system such as telephone directory services.

References

[Book 87] R. V. Book, "Thue Systems as Rewriting Systems," Journal of Symbolic Computation Vol.3, No.1, pp39-68, 1987

[Buchberger 87] B. Buchberger, "History and Basic Feature of the Critical-Pair/Completion Procedure," Journal of Symbolic Computation Vol.3, No.1, pp1-38, 1987

[Huet 80] G. Huet, "Confluent Reductions: Abstract Properties and Application to Term Rewriting Systems," Journal of the ACM, Voll.27, No.4, pp.797-821, 1980

[Huet81] G. Huet, "A complete proof of correctness of the Knuth-Bendix completion algorithm," Journal of Computer and System Science, Vol.23, No.1, pp.11-21, 1981

[Squire87] C. Squire and F. Otto "The Word Problem for finitely presented Monoids and Finite Cannonical Rewriting Systems.", Lecture Notes in Computer Science Vol.256, pp74-82, 1987