Report for Abbreviation Recognition Project

Xiaodong Wang

January 26, 2005

Example section is added to illustrate the method in this report. Implementation section is added to report the experiments. Some minor changes in the other sections.

1 Problem Definition

Given a set of technical papers in model checking area, this project aims to recognize abbreviations of names and identify their definition in these papers. Among these papers, one paper is used as main text which may refer to other papers for definition of abbreviations. This project will process the main text extracted from main paper and the refereed text if necessary and:

- Recognize abbreviations of names in the main text
- Identify definition candidates for each abbreviation
- Choose best match for abbreviation among definition candidates
- Output pairs of abbreviation and their best candidates

This proposal presents an improved method over an existing method in [1]. The improvements are based on the observations over the a bunch of papers listed in the bibliography.

2 Overview

The method given in [1] recognize abbreviations and their definitions in one text (refereed texts are not examined). The method iterates through each word sequentially in the text. For each word, the method check the word against a set of rules to tell if it is an abbreviation (section5). If it is an abbreviation, it is parsed into abbreviation pattern. Abbreviation pattern reveals the structure of the abbreviation. The pattern consists of a sequence of abbreviation elements which can a letter, non-alphanumeric character or a number. To look for definition, the method assume that definition appear around the abbreviation (within the search space defined in section 6). (It only search for definition in

this text; definition given in refereed text are neglected.) In the search space, definition candidates are identified based on a set of rules (section 6). This set of rules only requires that the sequence of word in the candidate form a proper definition. The similarity between the candidate and abbreviation has not yet be examined at this point. These definition candidates are then parsed into definition patterns to reveal the candidates' internal structures (section 6). In this process, prefixes are identified and represented in the pattern; number are recognized; stop-words such as "for", "in" which normally are represented in abbreviation are identified. Definition patterns are then matched against abbreviation patterns using a set of pattern-based rules. The pattern-based rules give the common ways of abbreviating from definition pattern (the structure of definition) into abbreviation pattern (the structure of abbreviation). If there exist one rule matching the definition pattern with the abbreviation pattern, the corresponding definition candidate will be reported as successful match. If multiple matched definitions exist for one abbreviation, the cue word and text marker which often occur between definition and abbreviation are used to choose the best candidate.

The major problems with the above method are:

- This method relies on the knowledge of the ways of abbreviating (the set of pattern-based rules). The accuracy of this method depends on the completeness of the set of pattern-based rules. Since the ways to abbreviate varies in different areas, this set of rules either contain too many rules or is not complete.

- It is very inefficient. Given a definition pattern and an abbreviation pattern, there can be many possible ways (rules) to abbreviate from the definition pattern to the abbreviation pattern. If all these ways are examined, it can be very inefficient. Otherwise, if only common ways are examined, it may lose accuracy.

- The method relies on well-formed definition pattern. That means the method works well only when all the prefixes, stop-words are properly identified. In some cases, there are area-specific prefixes which may not be known in advance and thus not recognized when it is parsed. The method will fail in this case.

- The definition in refereed text are neglected by this method.

The method presented in this proposal follows the same infrastructure. But it have several improvement over the above method. It utilizes a more accurate and efficient matching process based on sub-sequence matching. The propose method will search definition in refereed text. It also take the text context of definition into consideration. The text context can be further analyzed to include more information in future work.

- The author uses sub-sequence matching (section 7) instead of pattern-based rules. This method does not rely on the built-in knowledge of how

to abbreviate (the set of pattern-based rules) to identify definition. Instead, it try to test if there is sub-sequence relationship between definition pattern and abbreviation pattern. If there is no sub-sequence relationship between the patterns, we will remove those elements corresponding to stop-words or prefixes from definition pattern and test again. If there is still no sub-sequence relationship, the longest common subsequence between abbreviation pattern and definition pattern is computed. If the longest common subsequence is long enough (more than half of the definition pattern), the abbreviation pattern is matched with this subsequence of definition pattern. This method is more accurate since it does not rely on the completeness of set of pattern-based rules. It does not require all prefixes are recognized since it is looking for sub-sequence relationship. Missed prefixes can be accommodated by this sub-sequence relationship. This method is also efficient since it only need to test sub-sequence relationship for a few times instead of trying to matching according to each rule.

- Refereed text is examined for definition. The way to search for definition in refereed text is simple. Each time an abbreviation is followed by a pair of bracket, for example "abbreviation[1]", the characters in the bracket (here are "1") are taken as the file name of the refereed text. The abbreviation is added to the toBeMatched list of the refereed text. After we finish the search in the main text, we will go through each refereed text to look for the definition of those abbreviation in its toBeMatched list. The same process is followed for each refereed text except that we only look for definitions of those abbreviations in the toBeMatched list.

The author's method also take the text context of definition candidate into consideration when choosing the best definition candidate (section 8). The text context of the definition here is simply defined as the word following the definition. For name abbreviations, the definition sometimes have the same text context as the abbreviation. For example, in [2], abbreviation "VIS" is defined in the following text: "Verification Interface with Synthesis (VIS) tool". In this text, the word context of the definition candidate "Verification Interface with Synthesis" is "tool". When abbreviation "VIS" is used in the subsequent text, it is often used as "VIS tool". "VIS" here has the same text context as the definition. So if the matching process in section 7 can not determine the best definition candidate, we can choose the one whose word context best matches the context when we use the abbreviation. This can help us to choose the best definition candidate and improve the accuracy.

The disadvantage of the proposed method is that it can not handle the case where the order of words is permuted during abbreviation. For example, "third computer aided verification" can be abbreviated as "CAV3". Since the proposed method asks for sub-sequence relationship, this definition can not be matched with the abbreviation.

3 Example

We will use an example from [2] to illustrate the method described in this report. Here is a paragraph taken from [2].
"In this paper we present our experience on model checking of the input FIFO of RCMP-800 using the Verification Interacting with Synthesis (VIS) tool [1]. The RCMP-800 (Routing Control, Monitoring and Policing 800 Mbps [8]), a product from PMC-Sierra, Inc., is an integrated circuit that implements ATM (Asynchronous Transfer Mode) [2][3] 2 layer functions including fault and performance monitoring, header translation and cell rate policing). We wrote the RTL (Register Transfer Level) description of the input FIFO in Verilog. Since, the input FIFO has 128 x 16 bit memory which could contain 4 ATM cells, its verification could not be handled by VIS. Theore, we abstracted away the memory to concentrate on the functionality of the control circuitry, which is usually the critical part in the verification. We then established an environment for the input FIFO, which gives the inputs as random variables and defines registers as a default value."

The program will scan each word from this paragraph to look for abbreviations. According to rules in section 5, "FIFO" "RCM-800", "VIS", "PMC-Sierra", "ATM" and "RTL" in the paragraph are recognized as abbreviations. We take "RCM-800" as an example to explain the steps of the method. Other abbreviations follows the same process.

There are two occurrences of "RCM-800" in the paragraph. The method will look for definition for "RCM-800" around both occurrence. When the program scan to the first occurrence of "RCM-800" in the paragraph(*"input FIFO of* **RCMP-800** *using"***), it search for definition candidate in its search space defined in section 6. For this occurrence, the search space will be:** *"In this paper we present our experience on model checking of the input FIFO of RCMP-800 using the Verification Interacting with Synthesis (VIS) tool [1]. The"***. According to rules in section 6, the beginning word of the definition candidate should match the first character of "RCM-800". That means either the initial character of the beginning word or an internal capital letter matches the first character of the abbreviation. It is easy to see that there is no definition candidate in the search space of this "RCM-800" occurrence.**

When the program scan to second occurrence of "RCM-800" in the paragraph (*"tool [1]. The* **RCMP-800"), it looks for definition candidate in its search space:** *"of RCMP-800 using the Verification Interacting with Synthesis (VIS) tool [1]. The RCMP-800 (Routing Control, Monitoring and Policing 800 Mbps [8]), a product from PMC-Sierra, Inc., is an"***. According to the rules in section 6, the word sequence starting from** *"Routing"* **is a definition candidate. Since a definition candidate can not extended beyond text marker such as "(","[" and sentence boundary, the definition candidate are** *"Routing Control, Monitoring and Policing 800*

Mbps". Note that comma does not end a sentence. Words separated by comma are considered to be in one sentence.

After identify the definition candidate, we need to determine the similarity between the definition candidate and the abbreviation. First we parse the abbreviation into abbreviation pattern using function 5.2 to reveal its internal structure. The abbreviation pattern for "RCM-800" are "cccn" where "c" represents character such as "R", "C", "M" and "n" represents number "800". Because hyphen can be introduced by broken-line, it is not considered as a non-alphanumeric character that need to be identified in abbreviation pattern. The purpose of parsing the abbreviation is to identify all the elements and their relative order in the abbreviation. Different types of elements are matched differently during matching sub-step below. Character normally matches with initial character or internal capital character of a word. Number and non-alphanumeric characters which normally match exactly with a word or its replace-form in definition. Parsing can also get rid of some non-alphanumeric characters that are introduced during abbreviation such as "-" in this example.

The definition candidate is also parsed into definition pattern to reveal its internal structure using function 6.3. The definition pattern obtained ("wphwswnw") consists of elements such as "w"(word), "p"(prefix), "h"(head-word, portion of the word without prefix), "s"(stop-word) and "n"(number). Each definition pattern element has a corresponding element in the original definition candidate. In this example, "control" are represented by "p" and "h" since "con" is a prefix in the pre-defined prefix list. "and" are represented by "s" since it is one of the stop-words defined in the stop-word list. The prefix list and the stop-word list are in global configuration of the program which is pre-defined in the program. Definition pattern is then used to guide the product of possible abbreviations. Guided by definition pattern, we produce possible abbreviations by copying the non-alphanumeric character and number elements in the definition candidate and taking a number of characters (normally one, but can be several if the element has more than one capital letter in it) from each remaining definition candidate element. For example, one possible abbreviation for the above example can be "RCTMAP800". We may have multiple possible abbreviations if some words have replace-forms. Word having replace-form can be represented by itself or its replace-form in possible abbreviations. For example, "and" can represented by "&" or "A", "cross" can be represented by "X" or "C". So "RCTM&P800" is another possible abbreviation because "and" has a replace-form "&".

Now we try to match the possible abbreviations ("RCTMAP800" / "RCTM&P800") with the real abbreviation ("RCMP800") using the rules in section 7. It is easy to see that there is a sub-sequence relationship between the real abbreviation and the possible abbrevi-

ation. So the corresponding definition candidate is a good match. In case that the possible abbreviation is shorter than real abbreviation and the possible abbreviation is a sub-sequence of real abbreviation, we further require that every unmatched character in real abbreviation should have a credible source in the definition candidate. This can be done by searching for the unmatched character in the definition candidate.

In case that definition is given in refereed text, we recognized it by looking for "[" and "]" pair after the abbreviation. The similar process is followed in refereed text except that we only look for definition of abbreviations refereed in main text.

4 Implementation

The author has implemented all the major features of the method.

The application has been applied to a few set of papers. The first set consists of [2], [3] and [4] with [2] as the main text. The second set consists of [16] with [16] as main text. The third set consists of [17] with [17] as main text.The forth set consists of [18] with [18] as main text. The method works quite well on these papers. Almost all the abbreviations occurring in the [2], [16] ,[17] and [18] have been recognized. Their definitions are correctly identified with one exception. For those abbreviations whose definitions are given in refereed text, their definitions are also identified if the refereed text is available on line. There are also problems when this method is applied on the set of papers. Since [2] contains code with identifiers, these identifiers are recognized as abbreviations by the program. But the author believes that this is out of the scope of the method.

5 Recognize Abbreviation

, it search for definition candidate The first task is to recognize the abbreviations of names used in the main text. Generally speaking, abbreviations are shorten forms of words or phrases which are used instead of full forms after defined. Acronyms are special forms of abbreviations which are normally formed by taking first characters from multiple words in phrases.

5.1 Recognize abbreviation in the text

According to the observation, abbreviations follow the following set of rules:

- It has to be one word syntacticly.
- Its first character is either alphabetic or numeric.
- Its length is between 2 and 10 characters.

- It contains at least two capital letter or one capital letter and numbers. [1] only requires one capital letter and has no requirement on numbers. Since this project will use a dictionary as in [1], we ask for two capital letter to avoid the need for a dictionary.
- It is not a member of domain-well-known words whose definitions are generally not given. For example, definition of abbreviation "M/C" is generally not given in model checking paper since it is well understand in this area. The program will fail to find the definition for "M/C" since it is not given in the main text or any referred text.
- It is not the plural form of a recognized abbreviation. For example, in [2], "VPCs" appears as the plural form of "VPC". It satisfies that above rules. But "VPCs" should not be recognized as a different abbreviation from "VPC".

Data : word
Result: true/false
if *2 ≤ length(word) ≤ 10 and capLetterCount(word) ≥ 2* **then**
  if *word not in well-known-list* **then**
    word = single-form(word);
    if *word not in abbr-list* **then**
      return true;
    end
  end
end
return false;

Function: `isAbbreviation(`*word*`)`

5.2 Abbreviation Pattern for abbreviation

Abbreviation is processed to produce abbreviation pattern. Abbreviation pattern consists of a sequence of character "c" and character "n". The rules for generating abbreviation pattern can be listed as following:

- A alphabetic character is represented as "c". For example, abbreviation pattern for "VPC" will be "ccc"
- A continuous sequence of digits (including ".") are represented with single character "n". For example, abbreviation pattern for "SN1987A" will be "ccnc"
- Non-alphanumeric characters such "*" and "&" are copied in the same order into the pattern. The only exception is hyphen which can appear in abbreviation without correspondence in definition. For example, abbreviation pattern for "RCMP-800" will be "ccccn", abbreviation pattern for "OA&M" will be "cc&c"

```
Data  : abbreviation: abbr[N_left,N_right]
Result: abbreviation pattern: pattern
Initialize pattern to empty;
N_current ← N_left;
while N_current ≤ N_right do
    if abbr[N_current] is a digit then
        Add "n" to pattern;
        N_current ← N_current + 1;
    else
        Add "c" to pattern;
        N_current ← N_current + 1;
    end
end
return pattern;
```

Function: `abbrPattern(`*abbr*`)`

5.3 Contribution from author

The author has two contributions in this section. One contribution is to prevent the plural form of name abbreviations from being recognized as additional abbreviation besides its single form, which is the case in [1]. The other contribution is that non-alphabetic characters in abbreviations are copied in the abbreviation pattern. There are two reasons for that. According to the sample papers, non-alphabetic characters are normally copied from definition to abbreviation. We can improve accuracy by requiring that there be a exact match between definition candidate and abbreviation for non-alphabetic characters. Non-alphabetic characters can also be used to divide abbreviation and definition candidate into segment. We match the segments of abbreviation and definition one by one which is more efficient and accurate than matching the whole long definition and abbreviation.

6 Identify Definition

This section describes how to identify the definition candidates. After being identified, the definition candidate are processed to generate definition pattern which characterizes the candidate's structure. Text context of definition candidate is stored for future use.

6.1 Search space for abbreviation occurrence

Abbreviations are defined (either in the main text or the refereed text) and used in place of the definition in the subsequent text. According to [1], abbreviations are almost always defined near the place they are used in main text. Let N denotes the word location of a occurrence of the abbreviation. Let $|A|$ denotes

the length of the abbreviation in characters. We define the search space for this occurrence of abbreviation to be $(min\{|A|+5, |A|*2\}+5)$ words to the left and right of N. (According to [1], $min\{|A|+5, |A|*2\}$ is the maximum length of definition in words and "5" is the maximum offset between definition and abbreviation). In [1], the program will search the search spaces for all abbreviation occurrences for definition candidates.

There are two problems with this method. First, the definition of abbreviation may appear in referred text instead of main text. This can be indicated by a reference after the abbreviation occurrence. The search space defined for this abbreviation occurrence is incorrect in [1]. The second problem is that in technical papers, abbreviations are defined once in the main text and used subsequently. But [1] searches the search space for all abbreviation occurrence for definition candidate. According to the author's observation, abbreviations are mostly defined the first time they are used in the set of sample papers. In other cases, they are defined at the second or third time they occur if they appear in the abstract without definition. So it is not reasonable to search the search spaces for all abbreviation occurrences as in [1]. Search spaces for the first three occurrences will be searched to identify definition candidate. We will use other abbreviation occurrences to gather context information.

Data : Abbreviation: words[N]
Result: N_{left}: left boundary of search space, N_{right}: right boundary of search space
$N_{left} \leftarrow N - (min\{length(words[N]) + 5, length(words[N]) * 2\} + 5)$;
$N_{right} \leftarrow N + (min\{length(words[N]) + 5, length(words[N]) * 2\} + 5)$;
return words[N_{left},N_{right}];

Function: `searchSpace(`*words[N]*`)`

6.2 Definition Candidate

Within each searching space, a sequence of words are considered as a candidate definition if it meets the following requirements:

- The first word of definition candidate matches the first character in the abbreviation.
- All words in definition candidate are within the same sentence
- The first word of definition candidate is not preposition, be-verb, modal verb, conjunction or pronoun
- Special symbols such as "(", ")", "[", "]", "", "", "=" may not be inside definition candidate

This process can be illustrated with the following example. Here is a piece of text taken from [2]: "For the remaining cells, a subset of ATM header and

appended bits is used as a search key to find the VC (Virtual Channel) Table Record for the virtual translation. If .....". "VC" is a recognized abbreviation. Let's identify the definition candidates of "VC". Because $|"VC"| = 2$, the search space for this occurrence of "VC" will be $(min\{2+5, 2*2\}+5) = 9$ words to left and right of "VC", which will be text: "is used as a search key to find the VC (Virtual Channel) Table Record for the virtual translation. If". (According to [2], the maximum length of the definition is 4.) According to above rules, "Virtual Channel" and "virtual translation" are the definition candidates in this search space. Note that "Table" are not included in the definition candidate "Virtual Channel" because ")" should not be inside a definition. "If" is not included in the definition candidate "virtual translation" because it is in a different sentence.

Data : search space: words[N_{left}, N_{right}], abbreviation: words[N]
Result: a list of definition candidate
Initialize candidate-list to empty;
$N_{current} \leftarrow N_{left}$;
while $N_{current} \leq N_{right}$ **do**
  if *firstChar(words[*$N_{current}$*]) = firstChar(words[*N*])* **then**
    if *words[*$N_{current}$*] not in prep-list or be-verb-list or modal-verb-list or conj-list or pronoun-list* **then**
      $M_{left} \leftarrow N_{current}$;
      repeat
        $N_{current} \leftarrow N_{current} + 1$;
      until *words[*$N_{current}$*] in special-symb-list or words[*$N_{current}$*] = "."*;
      $M_{right} \leftarrow N_{current} - 1$;
      Add words[M_{left},M_{right}] into candidate-list;
    else
      $N_{current} \leftarrow N_{current} + 1$;
    end
  else
    $N_{current} \leftarrow N_{current} + 1$;
  end
end
return candidate-list;

Function: `defCandidate(`*searchSpace,abbreviation*`)`

6.3 Definition Pattern for a Definition Candidate

According to [1], each word in the candidate definitions is processed sequentially to generate a definition pattern which characterize the candidate. Each word is represented with one or two characters in the definition pattern. Unlike [1], non-alphabetic characters except "-" are also copied into the definition pattern

in author's method. Hyphens are not copied because it can be added in the definition candidate if a word is broken on line boundary.

During abbreviation, words such as "of", "for" are often neglected. They often do not have representation in abbreviations. They are called stop words which are given in a pre-defined list. Stop-words are represented with "s" in definition pattern. Syntactically, some words can be further decomposed into prefixes and headwords. For example "disable" can be further decomposed into "dis" (prefix) and "able" (headword). During abbreviation, prefix and headword can both have representation in abbreviations. For example, "disable" can be abbreviated as "DA" which takes a character from both prefix and headword. To accommodate these cases in matching process in section 7, such kind of words are represented as a "p" and a "h" in definition pattern. Prefixes are given in a pre-defined set. Numbers are represented with "n" in definition pattern. All the other words are represented with "w" in definition pattern.

For example, the definition pattern for definition candidate "Virtual Channel" is "ww"; the definition pattern for definition candidate "virtual translation" is "wph" since "trans" can be a prefix.

Data : definition candidate: words[N_{left}, N_{right}]
Result: definition pattern
Initialize pattern to empty;
$N_{current} \leftarrow N_{left}$;
while $N_{current} \leq N_{right}$ **do**
  if *words[$N_{current}$] in stop-words-list* **then**
    Add "s" to pattern;
    $N_{current} \leftarrow N_{current} + 1$;
  else
    if *words[$N_{current}$] is number* **then**
      Add "n" to pattern;
      $N_{current} \leftarrow N_{current} + 1$;
    else
      if *any prefix in prefix-list is the initial substring of words[$N_{current}$]* **then**
        Add "ph" to pattern;
        $N_{current} \leftarrow N_{current} + 1$;
      else
        Add "w" to pattern;
        $N_{current} \leftarrow N_{current} + 1$;
      end
    end
  end
end
return pattern;

Function: `defPattern(`*defCandidate*`)`

6.4 Context information of the Definition Candidate

Besides the definition pattern, some context information of the definition candidate is also stored with definition candidate. Two kinds of context information are stored. The first context information is the text marker or cue words as discussed in [1]. Generally, if there are text markers or cue words between definition candidate and abbreviation, the definition candidate has a better chance than other definition candidates. The following text markers and cue words are taken into consideration:

- "abbr (definition)" or "abbr [definition]"
- "abbr = definition" or "definition = abbr"
- "abbr, or definition" or "definition, or abbr"
- "abbr, stands/short/acronym...definition"
- "definition, for short abbr"

Suppose definition candidate is located between word N_{left} and N_{right} and abbreviation occurs at word N. We look in the sequence of words between N_{right} and N for substring such as "(", "[", "or", "stands", "short", "acronym" if abbreviation occurs to the right side of definition candidate. Otherwise we look in the sequence of words between N_{left} and N for substring such as ")", "]", "or", "stands", "short", "acronym". This information will be stored with the definition candidate.

Data : definition candidate:words[N_{left}, N_{right}], abbreviation:words[N]
Result: True/False
if $N \leq N_{left}$ **then**
  if *"("/"[" appears in words[N,N_{left}] and ")"/"]" does not appear in words[N,N_{left}]* **then**
    return true;
  end
  if *"stands"/"acronym" appears in words[N,N_{left}]* **then**
    return true;
  end
else
  if *"("/"[" appears in words[N_{right},N] and ")"/"]" does not appear in words[N_{right},N]* **then**
    return true;
  end
  if *"short"/"acronym" appears in words[N_{right},N]* **then**
    return true;
  end
end

Function: `markerOrCue(`*defCandidate, abbreviation*`)`

The other kind of context informations are the words context of definition candidate (abbreviation occurrence). We define the word context of a definition candidate (abbreviation occurrence) to be the word after the definition candidate (abbreviation occurrence). These words can not be punctuation, abbreviation itself or special symbols. According author observation, abbreviations often have the same word contexts when they are defined as when they are used. For example, in [2], the following text contains the definition of abbreviation "VIS": "Verification Interface with Synthesis (VIS) tool". In this text, the word context of the definition candidate "Verification Interface with Synthesis" is "tool". When abbreviation "VIS" is used in the subsequent text, it is often used as "VIS tool". "VIS" here has the same as the definition candidate. So if the matching process in section 7 can not determine the best definition candidate, we can choose the one whose word context best matches the context when we use the abbreviation. Details will be given in the section 8.

Data : definition candidate:words[N_{left}, N_{right}], abbreviation:words[N]
Result: word context of the definition candidate
if $N \leq N_{left}$ **then**
  $N_{current} \leftarrow N_{right} + 1$;
  repeat
    $N_{current} \leftarrow N_{current} + 1$;
  until *words[$N_{current}$] is not in punctuation-list or special-symb-list*;
  return words[$N_{current}$];
else
  $N_{current} \leftarrow N + 1$;
  repeat
    $N_{current} \leftarrow N_{current} + 1$;
  until *words[$N_{current}$] is not in punctuation-list or special-symb-list*;
  return words[$N_{current}$];
end

Function: `getWordContext(`*defCandidate, abbreviation*`)`

6.5 Contribution from Author

There are four contributions by the author in this section. The first contribution is that the author's method will search the referred text for definition candidates. The same process will be followed to search definition candidates in the refereed text. [1] only searches the main text for definition candidates which is not complete. The second contribution is that only the search spaces for the first three occurrences (instead of all occurrences) are searched for definition candidates. The other occurrence of abbreviations are used to gather context information. The author expects this will substantially speed up the processing without losing much accuracy if any. The third contribution is that to be consistent with section 5, non-alphabetic characters are copied into the definition

pattern. [1] does not represent non-alphabetic characters either in definition pattern or abbreviation pattern. Thus it can not utilize non-alphabetic information during matching. The forth contribution from the author is that word contexts of definition candidates are stored with definition candidates. This will help to choose the best definition candidate in the section 8.

7 Matching Definition Candidate and Abbreviation

This section describes how to determine if a definition candidate matches the abbreviation.

In [1], abbreviation rules are used to denote existing ways of abbreviation. Program uses this rules when matching abbreviation and definition. Refer to [1] for the definition of abbreviation rules.

The major problem with rule-based approach in [1] is that the Rule-Base contains too many rules. The rule-based approach has to enumerate all matching ways between a definition pattern and an abbreviation pattern. It can be very inefficient during to the large Rule-Base.

The author uses an algorithmic approach which is more efficient and accurate. The disadvantage of this approach is that it is not a adaptive approach while the rule-based approach is. The rule-based approach can add new rules to the initial Rule-Base if new abbreviation way is discovered. The author's method is described in the following subsections.

7.1 Segmenting

Since all non-alphabetic characters and numbers in abbreviation are required to have a exact match (or match their replace forms) in the definition candidate, we can use these non-alphabetic characters as segment separators to divide the abbreviation (pattern) and the definition (pattern) into segments. The matching is then carried out over the segments of abbreviation (pattern) and definition (pattern).

Data : abbreviation: abbr, abbreviation pattern: abbr-pattern, definition candidate: def, definition pattern: def-pattern

Result: segment list

Initialize segment-list to empty

$N_{abbr}, N_{abbr-pattern}, N_{def}, N_{def-pattern} \leftarrow 0$;

while $N_{abbr-pattern} \leq length(abbr - pattern)$ **do**

  abbr-seg-left $\leftarrow N_{abbr-pattern}$;

  while *abbr-pattern[*$N_{abbr-pattern}$*] is not non-alphabetic character or "n" and* $N_{abbr-pattern} \leq length(abbr - pattern)$ **do**

    $N_{abbr-pattern} \leftarrow N_{abbr-pattern} + 1$;

  end

  abbr-seg-right $\leftarrow N_{abbr-pattern}$;

  if *abbr-pattern[*$N_{abbr-pattern}$*] = "n"* **then**

    chr $\leftarrow$ abbr[$N_{abbr-pattern}$];

  else

    chr $\leftarrow$ abbr-pattern[$N_{abbr-pattern}$]

  end

  def-seg-left $\leftarrow N_{def-pattern}$;

  def-seg-right $\leftarrow$ firstOccurrence(chr,def-pattern[$N_{def-pattern}$,]);

  if *def-seg-right* $\neq$ *-1* **then**

    add (abbr[abbr-seg-left,abbr-seg-right],
    abbr-pattern[abbr-seg-left,abbr-seg-right],
    def[def-seg-left,def-seg-right], def-pattern[def-seg-left,def-seg-right])
    into segment-list;

    $N_{def-pattern} \leftarrow$ def-seg-right;

  end

end

Function: `segmenting(`*abbr,abbr-pattern,def,def-pattern*`)`

7.2 matching segment

Now we try to match the abbreviation (pattern) segments with the definition (pattern) segments obtained from subsection 7.1.

7.2.1 possible abbreviation for definition (pattern) segment

As defined in section 6, the definition pattern for a definition candidate is a sequence of characters. Each character represent a element (word for "s"/"w", prefix for "p" and headword for "h" and number for "n") in the definition candidate. So according to definition pattern, the definition candidate can be viewed as a sequence of elements.

We will process each element sequentially in the definition candidate to produce a possible abbreviation.

- If it contains a capital letter, it is represented by this capital letter in the possible abbreviation. Otherwise,
- If it has replacement form, it is represented by its replacement form in the possible abbreviation. Otherwise,
- It is represented by the first character of the element in the possible abbreviation.

Obviously, this possible abbreviation has the same length as the corresponding definition pattern.

Let's take definition candidate "Register Transfer Level" ("RTL") as an example. Its definition pattern is "phww" ("Re" is recognized as a prefix and "gister" as a headword). Accordingly, its possible abbreviation is "RgTL".

Data : Definition candidate segment: words[N_{left}, N_{right}], definition pattern segment: pattern[N_{left}, N_{right}]
Result: the possible abbreviation for the definition candidate segment
Initialize posAbbreviation to empty;
$N_{current} \leftarrow N_{left}$;
while $N_{current} \leq N_{right}$ **do**
    word $\leftarrow$ words[$N_{current}$];
    if *word contains capital letter* **then**
        add the capital letter to posAbbreviation;
    else
        if *word has replace form in replace-form-list* **then**
            add its replace form to posAbbreviation;
        else
            add initChar(word) to posAbbreviation;
        end
    end
end
return posAbbreviation;

Function: `posAbbreviation(`*definition,def-pattern*`)`

7.2.2 match the possible abbreviation and abbreviation

A sequence of character S_1 is contained in another sequence of character S_2 if S_2 can be reduced to S_1 by omitting some characters. Standard function LCS(S_1, S_2) gives the longest common subsequence of S_1 and S_2.

Let $|A|$ denote the length of abbreviation A and $|D|$ denote the length of the possible abbreviation D.

The matching process can be divided into four steps:

- Step 1:

- $|A| < |D|$ and A is contained in D. This is considered as a successful matching. For above example, $|RTL| < |RgTL|$ and "RTL" is contained in "RgTL". So this is a successful match.
- $|A| < |D|$ and A is not contained in D, goto step 2.
- $|A| > |D|$ and D is contained in A. That means some elements in D contribute more than one character in A. For example, "CONTOUR" (A) is the abbreviation of "Comet Nuclear Tour". The definition pattern of "Comet Nuclear Tour" is "www". The possible abbreviation of "Comet Nuclear Tour" "CNT" (D) is contained in "CONTOUR". In this case, we have to do additional checking over characters in A that does not appear in D before we report a successful matching. In the above example, we need to make sure there is a "O" between "C" and "N" in the definition (which is "omet ") and there are "O", "U", "R" appear in the same order after "T" in the definition (which is "our"). For this example, it is a successful matching. If the check fails, matching fails.
- $|A| > |D|$ and D is not contained in A, goto step 2.

- Step 2: Remove characters corresponding to the stop-word elements in D and obtained D'. The idea here is that stop-words often are not represented during abbreviation. By removing characters corresponding to stop-word elements, we have another try. The same matching process in step 1 is carried out between A and D'. If $|A| < |D'|$ and A is not contained in D' or $|A| > |D'|$ and D' is not contained in A, goto step 3.

- Step 3: Remove characters corresponding to the headword elements in D' and obtained D''. The idea here is that headwords often are not represented during abbreviation. By removing characters corresponding to headword elements, we have another try. The same matching process in step 1 is carried out between A and D''. If $|A| < |D''|$ and A is not contained in D'' or $|A| > |D''|$ and D'' is not contained in A, goto step 4.

- Step 4: Let $D''' \leftarrow LCS(A, D'')$. If $length(D''') \leq length(D'')/2$, matching fails. Otherwise, the same matching process in step 1 is carried out between A and D'''. If $|A| < |D'''|$ and A is not contained in D''' or $|A| > |D'''|$ and D''' is not contained in A, matching fails.

Data : abbreviation: A_{abbr}, abbreviation pattern: $A_{pattern}$, definition candidate: D_{def}, definition pattern: $D_{pattern}$, possible abbreviation: D_{string}

Result: True/False

if $|A_{abbr}| < |D_{string}|$ *and* A_{abbr} *is contained in* D_{string} **then**
  return true;
end
if $|A_{abbr}| > |D_{string}|$ *and* D_{string} *is contained in* A_{abbr} **then**
  charSeq ← subsequence of A_{abbr} which does not appear in D_{string};
  if *charSeq appear in* D_{def} **then**
    return true;
  else
    return false;
  end
end
$D'_{string} \leftarrow D_{string}-$ characters corresponding to stop-word elements;
matching A_{abbr} and D'_{string} in the same way as above;
$D''_{string} \leftarrow D_{string}-$ characters corresponding to headword elements;
matching A_{abbr} and D'_{string} in the same way as above;
$D'''_{string} \leftarrow \mathrm{LCS}(A_{abbr}, D''_{string})$;
if $length(D'''_{string}) \leq length(D''_{string})$ **then**
  return false;
else
  if $|A_{abbr}| < |D'''_{string}|$ *and* A_{abbr} *is contained in* D'''_{string} **then**
    return true;
  end
  if $|A_{abbr}| < |D'''_{string}|$ *and* A_{abbr} *is not contained in* D'''_{string} **then**
    return false;
  end
  if $|A_{abbr}| > |D'''_{string}|$ *and* D'''_{string} *is contained in* A_{abbr} **then**
    charSeq ← subsequence of A_{abbr} which does not appear in D'''_{string};
    if *charSeq appear in* D_{def} **then**
      return true;
    else
      return false;
    end
  end
  if $|A_{abbr}| > |D'''_{string}|$ *and* D'''_{string} *is not contained in* A_{abbr} **then**
    return false;
  end
end

Function: `matching(`*abbr,abbr-pattern, def, def-pattern, posAbbreviation*`)`

7.3 Contribution from the author

Instead of using rule-based method as in [1], the author use a algorithmic approach. The method is substantially different from the rule-based method. So it can be considered as the contribution of the author as a whole.

8 Choose the best definition candidates

If multiple definition candidates are found for a specific abbreviation after searching the search spaces for the first three occurrence, context informations stored with the definition candidate are used to resolve the ambiguity. Definition candidates' context informations are gathered when they are identified in section 6.

8.1 Text marker and cue word

The definition candidates with text marker and cue word in their context are favored against the others. Since this information has already stored with the definition candidates, it can be used straightforwardly.

8.2 Word context of definition candidate

If more than one definition candidates have text marker or cue word between the definition candidate and abbreviation occurrence, we will use the word context of these definition candidates (namely, the words after the definition candidates) to resolve the ambiguity. For each abbreviation occurrence after the third one, we get its word context. If this word context matches the word context of any definition candidate, the matching-count of that definition candidate is increased by one. After processing all the abbreviation occurrences, we can choose the definition candidate with the largest matching count.

8.3 Contribution from author

Besides the text marker and cue word context information in [1], the author also take the word context of definition candidate into consideration. According to the author's observation, this can be very useful.

References

[1] Yongia Park, Roy J. Byrd: *Hybrid text mining for finding abbreviations and their definitions*, 2001, IBM Thomas J. Watson Research Center

[2] Jiangping Lu and Sofiene Tahar: *Model checking of the RCMP-800 Input FIFO*, April 2002, Concordia University

[3] CHRISTOPH KERN and MARK R. GREENSTREET: *Formal Verification in Hardware Design: A Survey*, April 1999, University of British Columbia

[4] The VIS Group: *VIS: A System for Verification and Synthesis*, 1996, 8th International Conference on Computer Aided Verification

[5] Clarke, E. M., Filkorn, T., Jha, S.: *Exploiting Symmetry in Temporal Logic Model Checking.* CAV93, LNCS **697** Springer-Verlag, 1993.

[6] Browne, M., Clarke, E. M., Grumberg, O, *Reasoning about networks with many identical finite-state processes*, Inf. Comput., 1989, (Vol. 81), 13–31.

[7] Emerson, E. A.: *Temporal and modal logic.* In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier/MIT Press, Amsterdam/Cambridge, 1990.

[8] Emerson, E. A., Lei, C. L., *Modalities for Modelchecking: Branchin time Strikes Back*, In Proc. of 12th ACM Symposium on Principles of Programming Languages, 1985, pp 84-96.

[9] Emerson, E. A., Sistla, A. P.: *Symmetry and Model Checking.* CAV93, LNCS **697** Springer-Verlag, 1993; journal version appeared in Formal Methods in System Design, 9(1/2),1996, pp 105-130.

[10] Emerson, E. A., Sistla, A. P.: *Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach.* CAV95, LNCS **939** Springer-Verlag, 1995.

[11] Emerson E. A., Treffler R., *From Symmetry to Asymmetry: New techniques for Symmetry Reduction in Model-checking*, Proc. of CHARME 1999.

[12] Emerson E. A., Havlicek J. W., *Virtual Symmetry Reductions*, Proc. of LICS 2000.

[13] Gyuris, V., Sistla, A. P.: *On-the-Fly Model Checking under Fairness that Exploits Symmetry.* CAV97, LNCS **1254** Springer-Verlag, 1997; To appear in Formal Methods in System Design.

[14] Ip, C. N., Dill, D. L.: *Better Verification through Symmetry.* Formal Methods in System Design **9** 1/2, pp41–75, 1996.

[15] Sistla A. P., Godefroid P.,*Symmetry and Reduced Symmetry in Model Checking*,CAV01, LNCS **2102** Springer-Verlag, 2001.

[16] Dongwon Lee, Wesley W. Chu, *Conjunctive Point Predicate-based Semantic Caching for Wrappers in Web Database*

[17] CHRISTOPH KERN, MARK R. GREENSTREET*ormal Verification in Hardware Design: A Survey*

[18] S. Ramanathan, Martha Steenstrup *A Survey of Routing Techniques for Mobile Communications Networks*