A Preliminary Design of an Easy-to-Dictate Programming Language with Pronouns

Wennong Cai Soramichi Akiyama Shigeru Chiba

Programming-by-voice is a technique that enables programmers to code by partially or fully using voice-input. Many existing researches attempt to design an easy-to-dictate spoken programming syntax that can map into the syntax of an existing typing programming language. However, they are still thinking in a typing programming style, while dictating should have its own advantage over typing. This paper proposes the usage of context-dependent pronouns in the programming-by-voice design. The goal of using pronouns is to replace long repetitive information with a short-length pronoun. In typing programming, repetitive information can be typed quickly through the support of auto-complete, while the spoken programming can hardly have such support. This paper presents our prototype spoken programming language: NPLang, and shows the mechanism of using pronouns inside NPLang. In the end, this paper analyzes the limitation of current proposed design and concludes with future direction.

1 Introduction

Programming supports various essential parts in human's daily life, while the request of fixing code might occur anytime for a programmer. However, when people need to go for a business trip, it is inconvenient to carry a laptop only for preparing any suddenly appeared coding work. On the contrary, it is inefficient to complete coding work if they cannot exploit spare time during the trip. Therefore, being able to do programming on phones or tablets is a lightweight and efficient solution.

However, it is difficult to do programming on small devices because of the small screen size. For example, "+", "=" or "{ }" are frequently used punctuation marks in most programming languages, but typing either of them on an iPhone keyboard requires 3 times click: convert to numerical mode button, convert to punctuation mode button, and finally click on the target key.

Using voice-input as an assistance tool to support programming on small devices will alleviate such difficulty. Programming-by-voice is a research topic that attempts to combine programming and voice, though some researchers in this area are motivated from different aspects such as helping Repetitive Strain Injury programmers [2] [3] [4].

While existing programming-by-voice designs are still not good enough due to their long length, we show our idea of using pronouns in the programming to refer existing long-length identifiers or expressions in order to reduce length by pruning redundant information. We also present the prototype programming language NPLang we have implemented that makes use of our idea.

In this paper, we first introduce the background of programming-by-voice by presenting some existing programming-by-voice researches. Then we present our proposal of using context-dependent

---

* 代名詞を使った口述プログラミング言語の初期設計

蔡 文農, 穐山 空道, 千葉 滋, 東京大学大学院情報理工学系研究科, Graduate School of Information Science and Technology, The University of Tokyo.

pronoun and its usage examples inside our prototype programming language. At last, related work of using pronouns in programming language will be introduced before the conclusion.

2 Programming-by-Voice Background

There are several existing researches that attempts to design a programming-by-voice system. However, they are not easy-to-dictate enough, because they only focus on designing the syntax for programming language on a context-independent style, which means dictating each statement is treated independently.

For instance, VoiceGrip [3] is a system that translates a spoken programming language text into the C language code. It provides some pre-defined syntax structures so that users could dictate easier than reading the C code directly. A usage example is as follows:

" *if current record number less than max off set then*"

will generate the code:

```
1 if (currRecNum < maxOffSet) {
2     |   // cursor will automatically
          move to here
3 }
```

Code 1 VoiceGrip example

The VoiceGrip designs an abbreviation algorithm to combine multiple words into a camelCase variable name, which rescues users from dictating every character of a variable name that cannot be recognized by the speech recognition as a single word directly. Based on VoiceGrip, a further implementation VoiceCode [4] has designed more voice macros to support navigating cursors by dictating.

However, those systems are still designing their syntax in a typing programming thinking. For example, to dictate the python code in Code 2:

```
1 maxOffSet = 30
2 prevRecNum = 20
3 currRecNum = 10
4 database = [ maxOffSet, prevRecNum,
      currRecNum ]
```

Code 2 Variables occur repetitively

The algorithm used by VoiceGrip and VoiceCode could help users easily dictating the three camelCase variables in the first three lines, however, those variables appear again in the fourth line. Users cannot make use of the 3 sentences they have dictated previously to support the dictation of the fourth line, but have to speak "*max off set [pause] previous record number [pause] current record number*" again, which is an inefficient and unpleasant experience. Dictating longer also means harder for the speech recognition system to fully and accurately recognize user input.

We consider those repetitively appeared information as redundant information in the spoken programming context. Redundant information limits the experience of spoken programming, but does not cause a serious problem in typing programming because there are existing mature tool-level supports such as the auto-complete feature. Auto-completion helps users quickly typing long variable names appeared previously, while the spoken programming is currently lack for good-quality auto-complete support. One of the reasons that makes auto-complete not suitable for spoken programming is the possible latency of voice recognition system. For instance, to dictate the variable name "currRecNum" with the help of auto-complete, a likely workflow might require speaking "*current [wait speech recognition system to process, and editor to propose several completion candidates] first option*", which makes dictation more complicated than only speaking "*current record number*". Therefore, redundant phrase is a difficulty that makes dictation harder.

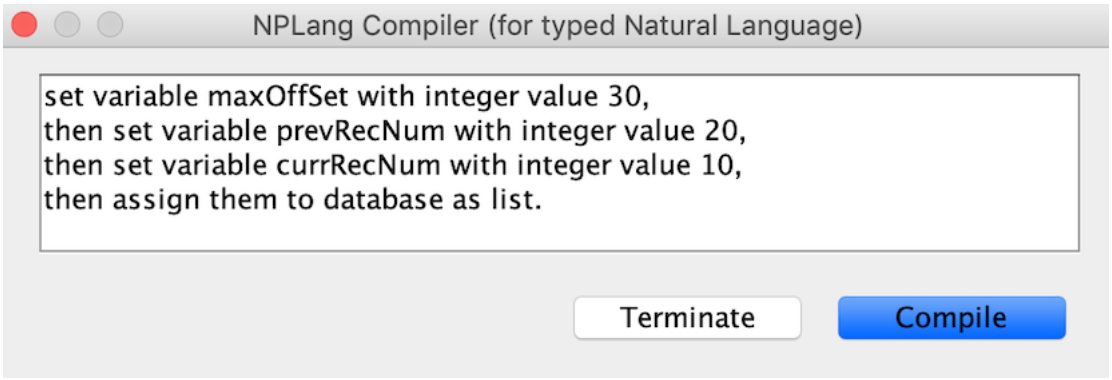

Figure 1 NPLang User Interface. The keyword 'them' refers to multiple variables: "maxOffSet", "prevRecNum", "currRecNum"

3 Proposal: Context-Dependent Pronoun

3.1 Overview

We propose the idea of Context-Dependent Pronoun, and present how it prunes the redundant information from the spoken programming code.

A context-dependent pronoun is a pre-defined keyword embedded in the spoken programming syntax, which refers to existing identifiers or expressions based on previous code context. Those pronouns are replaced by the phases they are referring during the compilation to form a valid code of an existing typing programming language.

For instance, in the previous Python example Code 2 mentioned in Section 2, users have to speak 9 words to dictate all elements in the list, and more words will need to be spoken if the length of list increases. In our system, as Figure 1 shows in the fourth line, users only use one keyword 'them' to refer all three variables. The python code compiled from it is the same as Code 2.

Since the meaning of the keyword 'them' changes depending on the previous code context, we name it the context-dependent pronoun.

The context-dependent pronouns prunes the redundant information when a pronoun is shorter than the phases it is referring to, which is the normal case in either natural language or our prototype language. If those identifier names or expressions are used repetitively and intensively, the sentence that needs to be spoken becomes verbose and contains many repetitive phases if we do not introduce context-dependent pronouns.

3.2 Prototype Programming Language: NPLang

The NPLang (*Natural Programming Language*) is a programming language we have designed for our pronouns experiment. Its syntax is designed to be similar to English so that it will be easy-to-dictate. The NPLang compiler we have implemented converts the NPLang code (as text form) into the equivalent Python code.

In NPLang, we focus on embedding the concept of context-dependent pronouns into the syntax, therefore we assume the input variable name is already well-recognised, instead of reproducing the algorithms for recognising abbreviated variable name from the work of VoiceGrip [3].

The BNF syntax is showed in Figure 2. The terminal symbol "TYPE" is used to distinguish string data from other types of data as well as the identifier, so that users do not have to dictate "open

value ::= TYPE “value” LITERAL

pronoun ::= **value** | IDENTIFIER | *“it” | “them”*

declaration ::= “set variable” IDENTIFIER [(“with” **value**)]

assignment ::= “assign” **pronoun** (“to” | “with”) **pronoun** [“as” TYPE]

calculation ::= OPERATOR **pronoun** [(“by” | “and”) **pronoun**]

statement ::= **declaration** | **assignment** | **calculation** | **invoke** | **function** | **returnExpr**

sentence ::= **statement** {“,” [EOL] “then” **statement**}

program ::= {**sentence** (“.” | EOL)}

invoke ::= “invoke” IDENTIFIER “with” **pronoun**

function ::= “define a function” IDENTIFIER [“taking” LITERAL “arguments”] (“.” | EOL) {**sentence** (“.” | EOL)} “end define”

returnExpr ::= “return” **pronoun**

Figure 2 BNF syntax of NPLang

double quotes” explicitly. Our current system does not support loop statement or conditional statement since we only focus on embedding the concept of pronouns into spoken programming at the current stage.

The core point of NPLang’s syntax is that the program consists of multiple sentences, and each sentence consists of multiple different kinds of statements concatenated by the keyword ‘then’. The context-dependent pronoun we propose works in statement-level, which means each sentence is considered as an independent context, and the meaning of each pronoun depends on previous statements within the same sentence.

3.3 Rules of Analyzing Pronouns

During the compilation, each context-dependent pronoun is replaced by the phrases it is referring based on its previous context. In current design, the meaning of each pronoun in NPLang is resolved through a switch statement, and match different cases based on the category of previous statements.

Specifically, the singular pronoun ‘it’ is interpreted as follows based on the category of the closest previous statement:

- **Declaration**: refer to the declared variable name.
- **Assignment**: refer to the assigned variable name.
- **Calculation**: refer to the assigned variable name if it involves calculation assignment operator ‘by’ (i.e. +=, -=, *=, /=), else refer to the whole expression.
- **Invoke**: refer to the whole expression.
- **Function**: refer to the declared function name.

For the plural pronoun 'them', all previous statements within the same sentence are examined separately through the above process, then separate results are concatenated with the comma separator.

3.4 Usage Examples

3.4.1 Refer based on Context

Figure 1 has already shown that using the context-dependent pronoun could refer to a single or multiple identifier name(s). As our proposal name "context-dependent" suggests, the entities being referred depend on previous programming context, as shown in Code 3. The first occurrence of keyword 'it' refers to 'num', while the second one refers to 'greeting'.

```
set variable num with integer value
    10,
then add it by integer value 5.
set variable greeting with string
    value hello,
then invoke print with it.
```

Compiled python result:

```
num = 10
num += 5
greeting = "hello"
print(greeting)
```

Code 3 Refer based on Context

3.4.2 Refer Expression

As described in the rules of analyzing pronouns, a pronoun sometimes refers to an expression rather than the identifier name in order to be more easy-to-dictate.

```
define a function addTwo taking 2
    arguments.
add p1 and p2,
then return it.
end define.
```

Compiled python result:

```
def addTwo(p1, p2) {
    return p1 + p2
}
```

Code 4 Refer Expression

In the current design, parameters of a function are automatically named "p" + the sequence number of parameters. The pronoun 'it' in this example refers to the expression "add p1 and p2".

3.4.3 Expression Omission

To make the generated Python code more concise, the NPLang compiler will omit an expression if it is being referred later and has no necessary side-effect such as assignment.

For instance, in the previous example Code 4, inside the definition of function "addTwo", there are two statements in the spoken text, which are a calculation, and a return statement with a pronoun 'it'. The compiler first translates them respectively into "p1+p2" and "return p1+p2", then the former statement is omitted since it has been referred by the latter statement and has no assignment effect.

Such design avoids the generated Python code becoming verbose or unnatural. The example Code 5 proves this more obviously, where all of the four statements are included in one single statement without changing program logic.

```
add integer value 1 and integer
    value 2,
then multiply integer value 2 and
    integer value 5,
then invoke addTwo with them,
then invoke print with it.
```

Compiled python result:

```
print(addTwo(1 + 2, 2 * 5))
```

Code 5 Expression Omission

3.5 Limitation

Currently we use pre-defined rules to resolve the meaning of each pronoun in the NPLang code. Also, as shown in previous sub-sections, the pronoun in our preliminary design has only two op-

tions, which is either ‘it’ for singular or ‘them’ for plural.

Such design is inefficient if we introduce new pronoun words apart from ‘it’ and ‘them’ in future. Because adding every new pronoun word requires more delicately designed rules to avoid causing grammar ambiguity.

Another limitation of the current design is that users have to remember those pre-defined rules instead of just using them intuitively in order to use context-dependent pronouns correctly, which increases the learning curve of such technique.

4 Related Work

Some pronoun words are already widely known as keywords in programming languages. For instance, the keyword “this” in well-known in some programming languages such as Java [5] or JavaScript [6]. It is usually used to refer the current object scope in order to fully exploit the design of object-oriented programming. For instance, without the help of the keyword ‘this’, the following JavaScript code becomes verbose.

```
1  var a = {
2      v: 1,
3      setV: function(n) {
4          this.v = n;
5      }
6  }
7  var b = {
8      v: 1,
9      setV: function(obj, n) {
10         obj.v = n;
11     }
12 }
13 a.setV(0);      // with 'this'
14 b.setV(b, 0);  // without 'this'
```

Code 6 Keyword ‘this’ in JavaScript

The primary goals of the pronoun keyword ‘this’ from JavaScript and the Context-Dependent Pronoun in NPLang are different. In OOP languages, keywords such as ‘this’ or ‘self’ are generally used to enhance the usage of object-oriented design, while in our research, the context-dependent pronoun serves the purpose of being an alternative but simpler way to express existing identifiers or expressions.

Another programming language example of using pronoun words is AppleScript [1]. In AppleScript, the keyword ‘me’ refers the current script and the keyword ‘it’ refers to the current target. As Code 7 (cited from [1]) shows, the word ‘its’ and ‘my’ are used to access properties of the objects referred to by ‘it’ and ‘me’.

Similar to previous example, AppleScript only provides pronouns for referring the current scope itself, rather than referring entities appeared in the previous context. Our system exploits pronoun words in a way that more similar to the usage in natural languages.

```
1  tell application "Finder"
2      version
3          --output: "10.5.1" (Finder
               version is the default
               in tell block)
4      its version
5          --output: "10.5.1" (
               specifically asks for
               Finder version)
6
7      version of me
8          --output: "2.0" (AppleScript
               version)
9      my version
10         --output: "2.0" (AppleScript
               version)
11     version of AppleScript
12         --output: "2.0" (AppleScript
               version)
13 end tell
```

Code 7 AppleScript Example

5 Conclusion

In this paper, we introduced the background of programming-by-voice and our proposal of embedding context-dependent pronouns to prune the redundant information in order to make the spoken programming syntax easier to be dictated. We

also presented our prototype programming language NPLang, and showed how our proposal perform in this prototype.

We have mentioned that the current design suffers from the limitation of flexibility to design more pronoun words, as well as the efficiency for users to learn how to use context-dependent pronouns. Therefore, as the future direction, we are considering to apply coreference resolution models from the area of Natural Language Processing into the NPLang in order to resolve the pronoun keywords automatically. The coreference resolution model is used to group phrases that are referring to the same entity. It is a evolving machine learning solution to automatically resolve the meaning of pronouns in natural languages. The usage of spoken programming language might be more flexible with help from such model, because the model might be able to infer the referred phrases from the semantic knowledge understanding of the pronouns.

参 考 文 献

[1] AppleScript Language Guide, Apple: *The it and me Keywords*. Retrieved from: https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/conceptual/ASLR_fundamentals.html.

[2] Arnold, S. C., M. L. and Goldthwaite, J.: Programming by voice, VocalProgramming, *In Proceedings of the fourth international ACM conference on Assistive technologies*, 2000, pp. 149–155.

[3] Desilets, A.: VoiceGrip: a tool for programming-by-voice, *International Journal of Speech Technology*, Vol. 4, No. 2(2001), pp. 103–116.

[4] Desilets, A., F. D. and Norton, S.: Voicecode: An innovative speech interface for programming-by-voice, *In CHI'06 Extended Abstracts on Human Factors in Computing Systems*, (2006 Apirl), pp. 239–242.

[5] Java Documentation, Oracle: *Using the this Keyword*. Retrieved from: https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html.

[6] JavaScript Reference, MDN Web Docs: *This*. Retrieved from: https://developer.mozilla.org/docs/Web/JavaScript/Reference/Operators/this.