CS4118 PROGRAMMING LANGUAGES AND TRANSLATORS

# Spoke

A Language for Spoken Dialogue Management & Natural Language Processing

Final Report

William Yang Wang, Chia-che Tsai, Xin Chen, Zhou Yu

**2010/12/23**

(yw2347, ct2459, xc2180, zy2147)@columbia.edu
Columbia University, New York, NY

## Table of Content

# 1. Introduction

## 1.1 Motivation

Spoken dialogue management remains one of the most challenging topics in natural language processing (NLP) and artificial intelligence (AI). There are various different spoken dialogue management theories and models (Cole et al., 1997; Pieraccine and Huerta, 2005), but there is no unified and light-weight programming language to describe them. Traditional programming languages, including Java, C, Perl and Lisp, are not designed to deal with natural language applications, and thus are slow, redundant, and ineffective when implementing spoken dialogue systems.

## 1.2 Overview

The ***Spoke*** programming language is a domain specific language, designing for implementing different spoken dialogue management strategies.  The Spoke users can set up their own spoken dialogue management schemas with very succinct syntax structure. In contrast to other general purpose languages, Spoke provides basic API support for ***dual programming language and natural language parsing***, as well as ***powerful syntax tree pattern matching methods***. Spoke language enables programmers to perform deep natural language analysis which can take English text input and give the parts of speech, and mark up the syntactic structure of sentences. It provides the foundational building blocks for higher level spoken dialog and natural language applications.

## 1.3 Language Features

### 1.3.1Basic Features

### 1.3.1.1 Basic Syntax

The Spoke language will be written in sequences, with each line representing one command in the source code. There is no need for using semicolon to terminate the command and the single expression argument containing multiple lines is currently not supported. The source code can be written in a single file, or be typed in by the user through the command line interactive interface.

### 1.3.1.2 Data Types, Operators, Expression

The four basic data types that the language supports will be Boolean, Integer, Float and String. A variable will be defined when it is first assigned. For each data type there will be several operators provided for processing. Also the conversion among data types is supported.

#### 1.3.1.2.1 Boolean

A variable of Boolean type can only be assigned by two values: TRUE and FALSE. Variable of other kind of data type can be converted to Boolean type by Boolean Expression. Operator supported for Boolean types include gt (greater than), lt (less than), eq (equal), ne (not equal), ge (not less than), le (not greater than) and logic gates like and, or and not.

#### 1.3.1.2.2 Integer and Float

A variable of Integer and Float can be assigned by a numeric value. For simplicity, direct assignment among different numeric types is not supported. Conversion between Integer and Float can be done by casting function Int() and Float(). Note that Int() will round the floating-point value. Operators supported for Integer and Float type include + (addition), - (subtraction), * (multiplication), / (division), % (modulus division).

#### 1.3.1.2.3 String

There will be no data type representing characters in this language. A variable of String type will be a sequence of character, and each character can be retrieved by specifying the position in the string. A constant of String type can be given by characters wrapped by quotation marks (“). Two quotation marks (“”) represent empty strings. Variable of other type can be converted into String type by the casting function Str(). One operator supported for String type will be + (concatenate). By the way, equality operator (=) is for the comparison of strings, word by word.

#### 1.3.1.2.4 Array

Arrays can be defined by assigning squared brackets (e.g., x = []). Also using squared brackets with a numeric value can specify the element in each array (e.g., x[0]) Addition operator (+) on arrays represents appending elements to the array.

### 1.3.1.3 Control Flows

#### 1.3.1.3.1 Conditional Structure

The only condition structure supported in this language will be if-else structure. The structure starts with a keyword “if” and finishes with another keyword “fi”. After the “if” keyword, a Boolean variable or a Boolean argument must be given to specify the condition to execute the code segment. Figure 2.3.1.1 shows an example for the if-then-else structure.

```
if x ne 0
      y = y / x
else
      y = 0
fi
```

**Figure 1.3.1.3.1** Sample code for if-else structure

#### 1.3.1.3.2 Loop Structure

Three loop structures will be supported in this language, while-loop, for-loop and do-while-loop. while-loop and do-while-loop will be given a condition by which the iteration of the code segment is determined. For for-loop, we will not provide a C style for-loop like for (i=0; i<10; i++). Instead, we adopt the common style provided in script languages like Bash or Python, in which for-loop iterates on an array. Figure 2.3.2.1 shows a sample for while-loop and do-while-loop, and Figure 2.3.2.2 features another sample for for-loop.

```
  while x ne 0          do                    for x in [0,
1, 2]
       y = y + x             y = y + x             y = y +
x
       x = x - 1           x = x - 1
  done                while x ne 0        done
```

**Figure 1.3.1.3.2.1** Sample code for while-loop **Figure 1.3.1.3.2.2**
and do-while-loop structure Sample code for for-loop structure

### 1.3.1.4 Input and Output

Our language provides input and output from standard devices and file systems. A “Print” command will print all the variables that follow the keyword (multiple variables have to be separated by comma). A “Rad” command and “readline” command will read a token or a whole line from the standard input and stored to the variable that follows. File I/O will be features in the style of python language. A Open() function will declare a file object and “print” or “read” command can be performed on it. Figure 2.4.1 shows an example of File I/O.

```
f1 = Open "test.txt" "w"
f2 = Open "input.txt" "r"
Readline from f2, buffer
Print to f1, buffer
```

**Figure 1.3.1.4** Sample code for File I/O

### 1.3.1.5 Functions and APIs

Functions in this language will be defined with a keyword “Func”. Parameters that follow the keyword consist of the name of the function, and a sequence of arguments. The argument will not be type-specific and can only be used as variable locally in the functions. Any other variable used outside the function will be global variable. At the end of the function, a “return” keyword will return all the parameters that follow. These return values are also not type-specific. Figure 2.5.1 shows an example of function.

```
Func concat str1 str2
     str = str1 + str2
     return str
End
```

**Figure 1.3.1.5**

```
str = concat "Hello" "world"
```

sample code for function

For the convenience of developers, we will implement a basic set of APIs in our library. These APIs include string processing, mathematical calculation, array manipulation and system commands. Table 1.3.1.5 lists all the APIs that we plan to implement.

| | | | | | |
|---|---|---|---|---|---|
| 1. | Len | Size of array or strings | 11. | Abs | Aboslute value |
| 2. | Left | Sub-string at the left of strings | 12. | Floor | Round up |
| 3. | Right | Sub-string at the right of strings | 13. | Ceil | Round down |
| 4. | Mid | Sub-string at the mid of strings | 14. | Pow | Raise to power |
| 5. | Find | Position of keyword in string | 15. | Sqrt | Square root |
| 6. | Sep | Seperate strings into tokens | 16. | Log | Logarithm |
| 7. | Rep | Replace keyword in strings | 17. | Exp | Exponential |
| 8. | Index | Index of element in arrays | 18. | System | Run command in shell |
| 9. | Sub | Substitution of arrays | | | |
| 10. | Map | Map elements to another type | 19. | Exit | Exit the program |

**Table 1.3.1.5** Build-in Functions and APIs

### 1.3.2 Advanced Features

#### 1.3.2.1 Spoken Dialog Management

We present a language by which developers can easily design their spoken dialog systems. We introduce a series of new data types, representing spoken dialogs in English, and provide adequate operators and APIs. Operations will be preformed on a concrete, well-formed syntax structure, which is produced by some NLP parser either provided by the standard libraries or designed by developers. The design of NLP parser must be flexible, and we will not spend time on implementing more than one NLP parser, since this work is not the primary goal of this project. Developers can always implement their own NLP parser using the basic features (mentioned in the previous chapter) we provide in this language. On the other hand, the NLP parser provided by the library should also be part of APIs instead of a feature of the language.

We adopt the syntax structure of the Penn Treebank designed by M.P. Marcus, B. Santorini, M.A. Marcinkiewicz at UPenn (Marcus et al., 1993). The Penn Treebank is a tag-based natural language repository, primarily but not necessarily designed for English, which can interpret a sentence into grammatical structures. They use a set of predefined tags to categorize each word, phrase or punctuation in a sentence, thus developers can easily perform whatever postprocessing on it. We assume that developers already know the format of tagged dialog and provide operation and expression interfaces.

In addition to tagging the sentence, the Penn Treebank uses a tree structure to describe the semantics of more complicated grammar. They use brackets to mark and separate each branch in the tree. Figure 3.1.1 shows an example of tagging and bracketing an English sentence.

Original sentence:

```
Battle-tested industrial managers here always buck up
nervous newcomers.
```

Tagged sentence:

```
Battle-tested/NNP industrial/JJ managers/NNS here/RB
always/RB buck/VB up/IN nervous/JJ newcomers/NNS ./.
```

Parsed sentence:

```
(S (NP Battle-tested/NNP industrial/JJ managers/NNS
here/RB)
   always/RB
   (VP buck/VB up/IN
      (NP nervous/JJ newcomers/NNS) ) .)
```

**Figure 1.3.2.1.1** Examples for tagging and bracketing sentences

Table 1.3.2.1.2 features the tags defined in our language. Tags will be all alphabetic, written in capital. During the development of the language, the definition of tags will be of 3 stages:

I. Adopt predefined tags of only basic representation like NN(Noun), VB(Verb), etc.

II. Adopt predefined tags of all grammatical representation in Penn TreeBank POS tag set.

III. Adopt user-defined tags

| | | | | | |
|---|---|---|---|---|---|
| 1. | CC | Coordinating conjunction | 19. | PRPS | Possessive pronoun |
| 2. | CD | Cardinal number | 20. | RB | Adverb |
| 3. | DT | Determiner | 21. | RBR | Adverb, comparative |
| 4. | EX | Existential *there* | 22. | RBS | Adverb, superlative |
| 5. | FW | Foreign word | 23. | RP | Particle |
| 6. | IN | Preposition/subord. conjunction | 24. | SYM | Symbol |
| | | | 25. | TO | *to* |
| 7. | JJ | Adjective | 26. | UH | Interjection |
| 8. | JJR | Adjective, comparative | 27. | VB | Verb, base form |
| 9. | JJS | Adjective, superlative | 28. | VBD | Verb, past tense |
| 10. | LS | List item marker | 29. | VBG | Verb, gerund or present participle |
| 11. | MD | Modal | | | |
| 12. | NN | Noun, singular or mass | 30. | VBN | Verb, past participle |
| 13. | NNS | Noun, plural | 31. | VBP | Verb, non-3rd person singular |
| 14. | NNP | Proper noun, singular | | | |
| 15. | NNPS | Proper noun, plural | 32. | VBZ | Verb, 3rd personA  singular |
| 16. | PDT | Predeterminer | 33. | WDT | Wh-determiner |
| 17. | POS | Possessive ending | 34. | WP | Wh-pronoun |
| 18. | PRP | Personal pronoun | 35. | WPS | Possessive wh-pronoun |
| | | | 36. | WRB | Wh-adverb |

Table **1.3.2.1.2** The Penn TreeBank Part-of-speech (POS) Tag Set

The data type we created for representing a semantic sentence is called Utterance, which means a sentence in the spoken dialog. An utterance must be well-tagged strings, generated by the default POS tagger, parser or customized parser. To distinguish utterances with normal strings, an utterance constant will be wrapped in apostrophe(`) instead of quotation mark (“) used for wrapping strings. The tags in the utterance will be wrapped in a pair of less-than mark and greater-than mark. (To avoid confusion with the less-than and greater-than operators, we use lt, gt instead as Boolean operators.) Figure 1.3.2.1 shows two sample codes for utterance manipulation.

```
# The first sample code, without I/O and conversion
Utterance = `<NN>I<VB>am<NN>Jerry`

if Utterance eq `<NN>I<VB>am<NN>*`
           Name = Utterance[2]
           print `<UH>Hello<NN>World` + Name
fi


# The second sample code, with I/O and conversion
```

```
      Use Parser default                        #   define   a
default parser

      readline Input                            # read a line
from stdin
      Utterance = Utter(Input)    # conversion

      if Utterance eq `<NN>I<VB>am<NN>*`
           Name = Utterance[2]
           Reply = str(`<UH>Hello<NN>World` + Name)
           print Reply + "\n"
      fi
```

**Figure 1.3.2.1** Sample codes for utterance manipulation.
The first sample code defines a utterance variable with hard-coded value. If the utter- ance contains a introduction of the speaker's name, the name is retrieved and a Hello World message is printed.
The second sample code works in a similar way, except the input is read from the key- board, converted into utterance using default parser.

The structure of bracketed utterance is actually a tree structure, we provide the same way as fetching elements from arrays to fetch branches from the semantic trees. In the first example of Figure 3.3, the name of the speaker is retrieved from the utterance by specifying the third branch (The ordering starts from 0.)  Note the branches being retrieved is still of the utterance data type, so the name of speaker here is in the form of another utterance `<NN>Jerry` instead of a string "Jerry". A branch can be appended to the tree structure by using the addition operator (+).

As the Penn Treebank, we also do bracketing on utterance. Parenthesis ( ( and ) ) can be used in the place of words in the utterance. Each sub-utterance wrapped in the parenthesis, tagged as well, represents a phrase in the sentence. From the structural point of view, bracketing utterance is actually creating branches in tree structure with additional descendants. We provides the same way as fetching elements from arrays to fetch descendants from utterances..

Conversion of utterance can be done by simply using casting functions. Utterances can only be casted into strings, by removing all the tags from the sentences. Conversion from strings to utterances is more complicated. Using keyword “Parser”, a NLP parser is declared, which is in fact a tagging program. This parser can be used to parse the semantic structure of a string and build up a utterance structure. We will provide a default parser which can be inefficient and only target on sentences in English.

Matching of utterance is done through Boolean operators. eq (equal) and ne (not equal) can match utterance variable with another utterance variable or constant. An asterisk (*) in the target means ambiguous matching. This feature helps developers design the logic of spoken dialog system. However we provide a even more powerful way to compare utterances. Using an operation called “Dictionary match”, a dictionary (e.g. WordNet) will be given by developers, and they can match utterances by a more flexible way. For example, the phrases “I am”, “My name is” and “Call me” has the same semantic meaning in English, and all indicate that a name follows. Developers can declare a dictionary file, with these three phrases linked together and use a few line of code to describe the whole logic.

Utterance is one of the most powerful features of this language. It combines natural language and programming language, and can be used in many domain of computer science. For example, the semantic structure of utterances can give advantage for development of machine translation software. Moreover, this feature can be used in domain which has the requirement of pattern matching or semantic parsing. (e.g. to represent genetic order in biologic information)

# 2. Language Tutorial

The Spoke Tutorial is a practical guide for programmers who want to use the Spoke programming language to create applications. It includes several complete, working examples, and dozens of language features.

## 2.1 What's New

The Spoke programming language is a domain specific language, designing for implementing different spoken dialogue management strategies. The Spoke users can set up their own spoken dialogue management schemas with very succinct syntax structure. In contrast to other general purpose languages, Spoke provides basic API support for dual programming language and natural language parsing, as well as powerful syntax tree pattern matching methods. Spoke language enables programmers to perform deep natural language analysis which can take English text input and give the parts of speech, and mark up the syntactic structure of sentences. It provides the foundational building blocks for higher level spoken dialog and natural language applications.

## 2.2 Trails Covering the Basics

Getting Started — An introduction to Spoke technology and lessons on installing Java development software and using it to create a simple program.

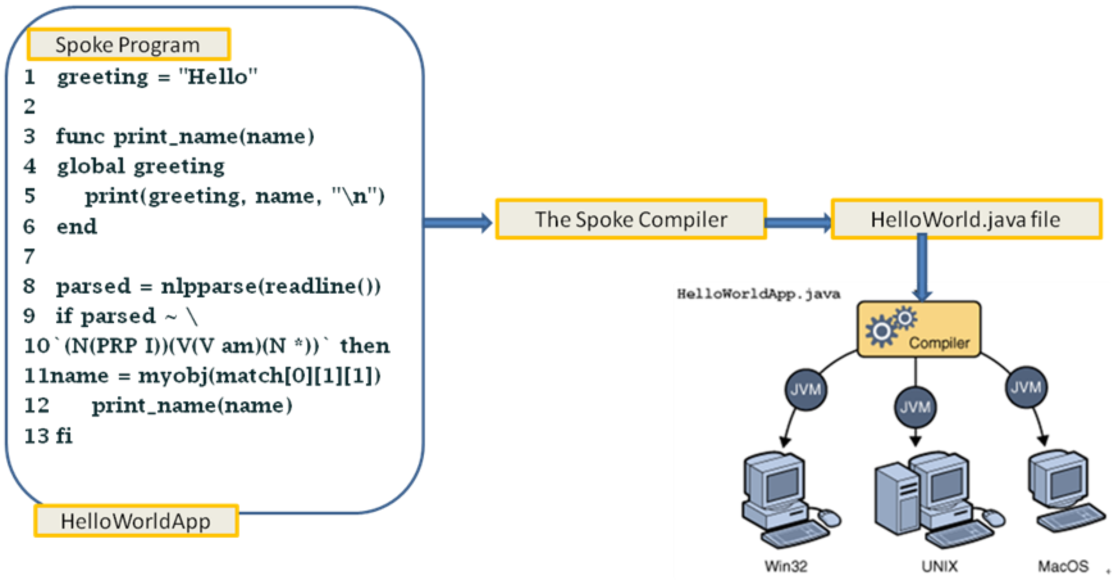
Learning the Spoke Language — Lessons describing the essential concepts and features of the Spoke Programming Language.

### 3.2.1 Getting Started

The Spoke programming language is a high-level language that can be characterized by all of the following buzzwords:

*Simple*: the syntax of The Spoke programming language is very similar with scripting language and the user of the language does not have to be a professional programmer. We assume that the users of The Spoke Language have some fundamental knowledge of Natural Language Process such as tagging and parsing, but they do not have to have intensive knowledge.

*Portable*: In the Spoke programming language, all source code is first written in plain text files without any extensions. Those source files are then translated into Java source file *.java* and then further compiled in .class files by the *javac* compiler. Since a .class file does not contain code that is native to the processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine1 (Java VM). The java launcher tool then runs the application with an instance of the Java Virtual Machine. All we need to compile and run the Spoke program are a Spoke compiler and Java VM.



```
greeting = "Hello"

func print_name(name)
global greeting
   print(greeting, name, "\n")
end

parsed = nlpparse(readline())
if parsed ~ \
`(N(PRP I))(V(V am)(N *))` then
name = myobj(match[0][1][1])
    print_name(name)
fi
```

*Figure 1 Spoke Language Architecture*

### 2.2.2 Learning the sample Spoke Programs

This section provides detailed instructions for compiling and running a simple "Hello World!" application.

**Hello World Program:**

```
greeting = "Hello"

func print_name(name)
    global greeting
    print(greeting, name, "\n")
end

parsed = nlpparse(readline())
if parsed ~ \`(N(PRP I))(V(V am)(N *))` then
     name = myobj(match[0][1][1])
  print_name(name)
fi
```

Hello, Jenny
I am Jenny

This HelloWorldApp Spoke program asks user to record a sentence, such as “I am Jenny”, and it captures the user’s name and reply “Hello, Jenny” to the user.

In line 1 we define a variable called “greeting” and assign "Hello" to it. The Spoke language does not have to declare the type of a variable, in this case, the variable greeting is in String type. Line 3~6 is a function, which prints the greeting message and the name. We can see that in line 4 we have a “global” keyword, which indicates that the scoping of greeting variable is a global variable and the value is “Hello” assigned in line 1. Line 8~10 does the Natural Language Processing operation. In line 8 we use the built-in API to read an input line and then parse it using NLP methods. It calls nlpparse() which parses the input string into a syntax tree, and assign this tree to the variable parsed, for example:

```
input = ‘How much did the Dow Jones drop today?’
parsed = nlpparse (Input)
# Parsed = `(ROOT
                 (SBARQ
                   (WHADJP (WRB how)
                                   (JJ much))
                   (SQ (VBD did)
                         (NP (NN dow)
                                   (NNS jones))
                         (VP (VB drop)
                               (NP (NN today))))))
```

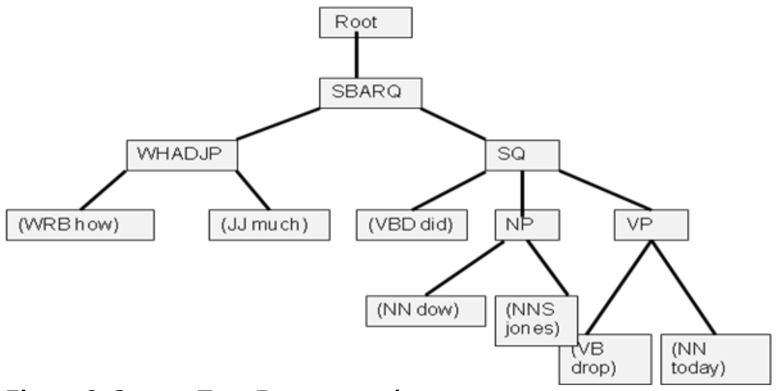
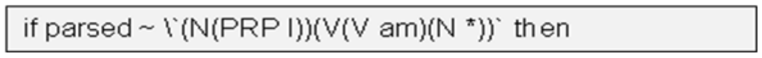


*Figure 2  Syntax Tree Representation*

Line 8 gives us an example of control flow, and it also demonstrate one of the key features of The Spoke Language — Partial Matching.



```
if parsed ~ \`(N(PRP I))(V(V am)(N *))` then
```

“~” is the symbol of partial matching, which indicates that whether or not the pattern `(N(PRP I))(V(V am)(N *))` is part (sub-tree) of the Syntax Tree, if yes the expression returns true, otherwise it returns false. The syntax tree of the pattern `(N(PRP I))(V(V am)(N *))` is shown in Figure 3.

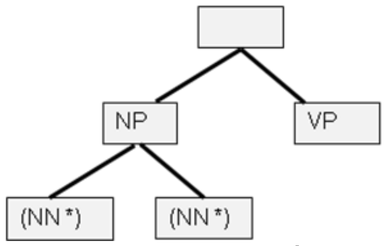


*Figure 3 Syntax tree of the pattern `(N(PRP I))(V(V am)(N *))`*

Let's go back to the previous example, in which the variable "parsed" represents the syntax tree of the sentence "How much did the Dow Jones drop today?" Does it match the pattern `(NP(NN *)(NN *))(VP *)`? The answer is yes. The pattern says that any sentence that contains the sub-syntax-tree in Figure 3 is matched to the pattern. See Figure 4.

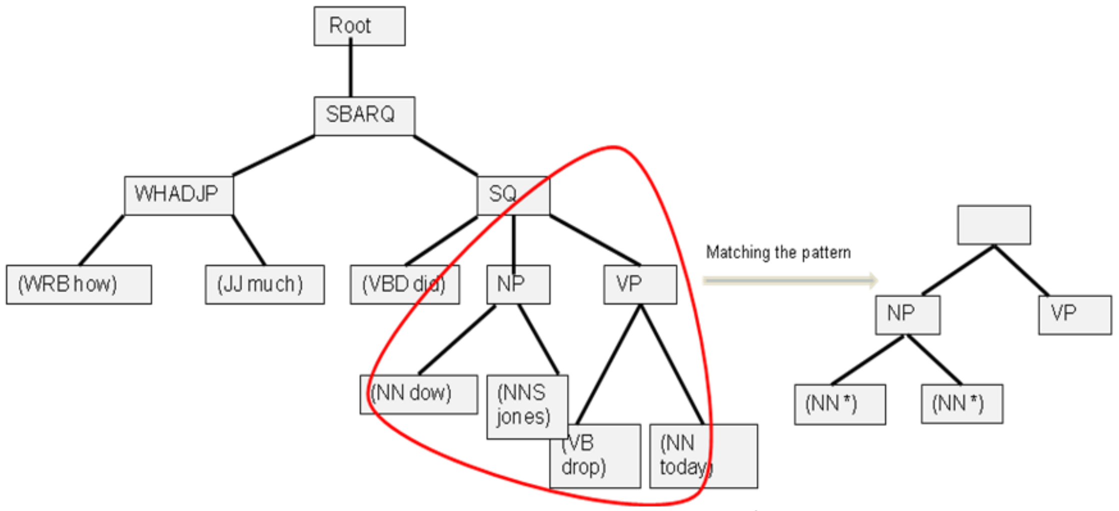


*Figure 4 The sentence "How much did the Dow Jones drop today?" matches the pattern `(NP(NN *)(NN *))(VP *)`*

Another feature that is indicated in line 9 is that we adopt a data structure in the form of "(Tag Word)". We tag each word in a sentence using NLP tagging convention and we use these tagged word in the syntax tree parsing later. We assume that the Spoke Language programmer knows some fundamental knowledge of NLP tagging and parsing. For example, "(NP(NN *)(NN *))(VP *)" indicates that we want a noun phrase contains 0 to 2 nouns and 0 to 1 adverb describes the previous noun phrase, such as "dow jones today".

### 2.2.3 Advanced Sample Spoken Dialogue Program

Assume we are using ***Spoke*** to build a financial market spoken dialogue system, where users can ask the following questions about stock market price:

```
      TEST_1 = 'DID DELTA AIR LINES DROP TODAY'
      TEST_2 = 'HOW MUCH DID DOW JONES RISE WEDNESDAY'
      TEST_3 = 'HOW WAS WALT DISNEY'
      TEST_4 = 'WHAT WAS THE PRICE OF USAIR GROUP ON FRIDAY'
      TEST_5 = 'WHICH STOCK CLIMBED ON MONDAY'
      TEST_6 = 'COULD YOU TELL ME THE PERFORMANCE OF STANDARD AND
POORS'
      TEST_7 = 'ARE YOU SURE IBM WENT UP'
      TEST_8 = 'HOW MANY STOCKS WENT DOWN TODAY'
      TEST_9 = 'MAY I KNOW IF WALT DISNEY ADVANCED THURSDAY'
      TEST_10 = 'WHO IS THE BIGGEST WINNER TODAY'
      TEST_11 = 'WHEN DID INDUSTRIALS AVERAGE SURGE'
      TEST_12 = 'WHERE DID DOW JONES CLOSE AT TODAY'
      TEST_13 = 'AT WHICH POINT DID DELTA AIR LINES OPEN AT THURSDAY'
```

However, one of the major challenges of spoken dialogue system is the front end automatic speech recognition component, which is always noisy. As a result, sometime the natural language understanding and spoken dialogue management component will have problem understand the correct content of the conversation. In this sample ***Spoke*** program, we use ***Spoke*** to parse the users' utterance, and try to match the question patterns in our database, of which we know the answer.

```
Input = 'How much did the Dow Jones drop today?'

Parsed = parse (Input)

print (Parsed)

# Parsed = `(ROOT (SBARQ (WHNP (WRB How) (JJ much)) (SQ (VP (VBD
did) (NP (DT the) (NNP Dow) (NNP Jones) (NN drop)) (NP (NN
today)))) (. ?)))`

if question == `(ROOT (SBARQ (WHNP (WRB How) (JJ much)) (SQ (VP
(VBD did) (NP * (NN drop)) (NP (NN *)))) (. ?)))` then

answer =  search_database($1, $2)

fi

print answer
```

Sample code:

```
func process(s)
    parsed = nlpparse(s)

    if parsed ~ `SBARQ (* *)(SQ *)` then
        ques = match[0][0]
        sent = match[0][1]
    else
        sent = parsed
    fi

    if   sent ~ `(VBD *)(NP *)(VP *)` then
        target = str(match[0][1])
        action = match[0][2]

        if   action ~ `(VB  *)(ADVP *)` then
            verb = str(match[0][0])
            advb = match[0][1]
        elif action ~ `VB *` then
            verb = str(match[0])
        fi
    elif sent ~ `(VBD *)(NP *)` then
        target = str(match[0][1])
    fi

    if ques then
        if ques ~ `WRB *` then
            type = str(match[0])
        fi
    fi

    day = "today"
    if advb then
        if advb ~ `RB *` then
            day = str(match[0])
        fi
    fi


    return [type, target, verb, day]

end

result = process(str(arg))

if result[0] == "how" then
print("it dropped 30.34")
elif result[1] == "dow jones" then
        print("30.34")
elif result[2] == "rise" then
        print("It rised 87.12")
elif result[1] == "walt" then
        print("It rised 87.12")
```

```
elif result[1] == "walt disney" then
        print("It rised 87.12")
elif result[2] == "rose" then
        print("It rised 87.12")
elif result[2] == "drop" then
        print("It dropped 23.01")
elif result[3] == "wednesday" then
        print("It rised 87.12")
elif result[3] == "today" then
        print("it dropped 30.34")
else
print('No information available.')
fi
```

## 2.3 Compile and Run

The ***Spoke*** language can be easily compiled by typing
**make**

In order to run the compiled code, just run:
**./spoke** < utterance

# 3 Language Manual

## 3.1 Lexical Conventions

The Spoke language will be written in sequences, with each line representing one command in the source code. There is no need for using semicolon to terminate the command or expression. The source code can be written in a single file, or be typed in by the user through the command line interactive interface.

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator-pairs. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

### 3.1.1 Comments

The character “#” introduces a comment, which comments out the entire line.

### 3.1.2 Identifiers (Names)

An identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore ‘‘_’’ counts as alphabetic. Upper and lower case letters are considered different. No more than the first eight characters are significant, and only the first seven for external identifiers.

### 3.1.3 Keywords

| | | | |
|---|---|---|---|
| IF | ELSE | ELIF | FI |
| FOR | IN | NEXT | WHILE |
| LOOP | BREAK | CONTINUE | FUNCTION |
| NATIVE | RETURN | END | IMPORT |
| EXTERN | GLOBAL | | |

### 3.1.4 Constants

There are several kinds of constants, as follows:

#### 3.1.4.1 Integer Constants

An integer constant is a sequence of digits.  All integers are decimal only.

#### 3.1.4.2 Floating Constants

A floating constant consists of an integer part, a decimal point, and a fraction part. The integer and fraction parts both consist of a sequence of digits.  Either the integer part or the fraction part (not both) may be missing.

### 3.1.5 Strings

There will be no data type representing characters in this language. A variable of String type will be a sequence of character, and each character can be retrieved by specifying the position in the string.

#### 3.1.5.1 Native Strings

Native strings will be wrapped by the single quotation mark “ ‘ ” symbol from the beginning to the end.

#### 3.1.5.2 Escape Strings

Escape strings will be wrapped by the double quotation mark “ “ ” symbol from the beginning to the end.

#### 3.1.5.3 Tagged Strings

Escape strings will be wrapped by the “ ` ” symbol from the beginning to the end.

## 3.2 Fundamental Types

Spoke supports three fundamental types of objects: strings, integers, and single-precision floating-point numbers.

Strings consist of characters (declared, and hereinafter called, char) chosen from the ASCII set; they occupy the right-most seven bits of an 8-bit byte. It is also possible to interpret chars as signed, 2’s complement 8-bit numbers.

Integers (int) are represented in 16-bit 2’s complement notation.

Single precision floating point (float) quantities have magnitude in the range approximately 10±38 or 0; their precision is 24 bits or about seven decimal digits.

## 3.3. Conversions

The ***spoke*** language itself does not support conversions at this moment. But it is possible to call API functions in Java to do conversions.

## 3.4. Expressions

### 3.4.1 Primary Expressions

#### 3.4.1.1 LPAREN/RPAREN Expression (Expression)

Description: A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. The presence of parentheses does not affect whether the expression is an lvalue. It does affect the order of operation.

Rules: If we have “(expression1) expression2”, expression1 will get computed first.

Example: (1 + 2) * 3 => 3*3 = 9

### 3.4.1.2 LBRACK/RBRACK Expression [Expression]

Description: A primary expression followed by an expression in square brackets is a primary expression. It is used to define an array.
Rules: we can have an array of size two with two constants as elements [constant, constant]; we can also append an array into another array [constant]+[constant : constant]. Also we can have expression as element in an array [expression, expression].

Example: array = [ ] + [1 , 2] => [1, 2]
array = [ ] + [1+2, 3] => [3, 3]
array[0] = 1 => [ 1 ]
array = array[0 : 2] => [0, 1, 2]

### 3.4.1.3 COMMA Expression , Expression

Description:  A pair of expressions separated by a comma is evaluated left-to-right and the value of the left expression is discarded.  The type and value of the result are the type and value of the right operand.  This operator groups left-to-right.

Rules: It should be avoided in situations where comma is given a special meaning, for example in actual arguments to function calls and lists of initializers.
Example:  lists of initializers : [1 , 2]

Function calls:    function foo
return 1, 2, 3

### 3.4.1.4 COLON Expression : Expression

Description: a : expression is used in array
Example: array[0 : 2] => [0, 1, 2]

### 3.4.2   Additive Operators

The additive operators +and − group left-to-right.

### 3.4.2.1 PLUS Expression + Expression

Description: The result is the sum of the expressions.  If both operands are int, the result is int.  If both are float, the result is float. If both are string, the result is string. We can also append an array to another array. No other type combinations are allowed.

Rules: int + int => int; float + float => float; string + string => string;
[ ] + [ ] => [ ]
Example:  3+4 => 7
[1, 2] + [3, 4] => [1, 2, 3, 4]
1.2 + 2.1 => 3.3
1.2 + 1 => error

### 3.4.2.2 MINUS Expression – Expression

Description: The result is the difference of the operands.  If both operands are int, the result is int.  If both are float, the result is float. We can't use "-" expression in string or array manipulation.
Rules: int - int => int; float - float => float;
Example: 7-3 => 4
1.2-3.2 => -2.0

### 3.4.3   Multiplicative Operators

The multiplicative operators *, /, and % group left-to-right.

### 3.4.3.1 TIMES Expression * Expression

Description: The binary * operator indicates multiplication. The result is the sum of the expressions.  If both operands are int, the result is int.  If both are float, the result is float. No other combinations are allowed.
Rules: int * int => int; float * float => float;

Example: 7*3 => 21
1.2*2.0=> 2.4

### 3.4.3.2 DIVIDE Expression / Expression

Description: The binary /operator indicates division.  The same type considerations as for multiplication apply.
Example: 7/3 => 2

1.2/2.0=> 0.6

#### 3.4.3.3 MODULUS Expression % Expression

Description: The binary %operator yields the remainder from the division of the first expression by the second.  Both operands must be int, and the result is int. In the current implementation, the remainder has the same sign as the dividend.
Rules: int % int => int
Example: 7%3 => 1

### 3.4.4 Equality Operator

The == (equal to) and the != (not equal to) operators are exactly analogous to the relational operators except for their lower precedence.  (Thus ‘‘a<b == c<d’’ is 1 whenever a<b and c<d have the same truth-value).

#### 3.4.4.1 EQ Expression == Expression

Description: equal to comparison
Rules:  int == int;  float == float
Example: a==b

#### 3.4.4.2 NEQ Expression != Expression

Description: not equal to comparison
Rules: int == int; float == float
Example: a!=b

### 3.4.5 Relational Operators

The relational operators group left-to-right, but this fact is not very useful; ‘‘a<b<c’’ does not mean what it seems to. The operators < (less than), > (greater than), <= (less than or equal to) and >= (greater than or equal to) all yield false if the specified relation is false and true if it is true.  Operand conversion is exactly the same as for the + operator.

#### 3.4.5.1 LT Expression < Expression

Description: less than comparison.
Rules: int < int ; float < float

Example: 1<2 => true;   1.0<2.0 => true

#### 3.4.5.2 GT Expression > Expression

Description: greater than comparison.
Rules: int > int ; float > float
Example: 2>1 => true;   2.0<1.0 => true

#### 3.4.5.3 LEQ Expression <= Expression

Description: less than or equal to.

#### 3.4.5.4 GEQ Expression >= Expression

Description: greater than or equal to

### 3.4.6 AND Expression && Expression

Description: The && operator returns true if both its operands are non-zero, false otherwise. && guarantees left-to-right evaluation; moreover the second operand is not evaluated if the first operand is false. The operands need have the same type, and each must have one of the fundamental types.
Example: (a&&b)

### 3.4.7 OR Expression || Expression

Description: The || operator returns true if either of its operands is non-zero, and false otherwise. || guarantees left-to-right evaluation; moreover, the second operand is not evaluated if the value of the first operand is non-zero. The operands need to have the same type, and each must have one of the fundamental types.
Example: (a||b)

### 3.4.8 Unary Operators

Expressions with unary operators group right-to-left.

#### 3.4.8.1 – Expression

The result is the negative of the expression, and has the same type. The type of the expression must be int or float.

#### 3.4.8.2 NOT Expression ! Expression

Description: The result of the logical negation operator ! is true if the value of the expression is false, false if the value of the expression is non-zero. The type of the result is Boolean. This operator is applicable only to int.
Example: (!a)

#### 3.4.9 ASSIGN Expression = Expression

Description: The assignment operator groups right-to-left. All require an lvalue as their left operand, and the type of an assignment expression is that of its left operand. The value is the value stored in the left operand after the assignment has taken place.
Rules: lvalue = expression
Example: a=3; a=2.0+3.0;

## 3.5 Declarations

There will be no declaration for variables and fundamental data types, which means variables and data types do not need to be declared before they are used.

## 3.6 Statements

Except as indicated, statements are executed in sequence.

### 3.6.1 Expression statement

Most statements are expression statements, which have the form expression; Usually expression statements are assignments or function calls.

### 3.6.2 Compound Statement

So that several statements can be used where one is expected, the compound statement is provided:

```
compound-statement:
( statement-list )
statement-list:
statement
statement statement-list
```

### 3.6.3 Conditional Statement

The three forms of the conditional statement are

```
1) IF ( expression ) THEN statement  FI
2) IF ( expression ) THEN statemtent ELSE statement  FI
3) IF ( expression ) THEN statement
     ELIF (expression) THEN statement
     ELSE statement
     FI
```

In all three cases the expression is evaluated and if it is non-zero, the first sub statement is executed.

### 3.6.4 While Statement

The while statement has the form

WHILE ( expression ) statement LOOP

The sub statement is executed repeatedly so long as the value of the expression remains non-zero. The test takes place before each execution of the statement.

### 3.6.5 For Statement

The for statement has the form:

```
FOR element in [number of iterations]
statement
NEXT
```

### 3.6.6 Return Statement

A function returns to its caller by means of the return statement, which has one of the forms return ( expression )

In this case, the value of the expression is returned to the caller of the function. If required, the expression is converted, as if by assignment, to the type of the function in which it appears. Flowing off the end of a function is equivalent to a return with no returned value.

## 3.7 Scope Rules

A complete ***Spoke*** program might not be compiled at the same time: the source text of the program may be kept in several files, and precompiled routines may be loaded from libraries. Communication among the functions of a program may be carried out both through explicit calls and through manipulation of external data.

Therefore, there are two kinds of scope to consider: first, what may be called the lexical scope of an identifier, which is essentially the region of a program during which it may be used without drawing ‘‘undefined identifier’’ diagnostics; and second, the scope associated with external identifiers, which is characterized by the rule that references to the same external identifier are references to the same object.

### 3.7.1 Lexical Scope

***Spoke*** is neither a block-structured language nor a parenthesis-based language. The lexical scope of names declared in external definitions extends from their definition through the end of the file in which they appear. It is an error to redeclare identifiers already declared in the current context, unless the new declaration specifies the same type and storage class as already possessed by the identifiers.

### 3.7.2 Scope of Externals

If a function declares an identifier to be extern, then somewhere among the files or libraries constituting the complete program there must be an external definition for the identifier. All functions in a given program which refer to the same external identifier refer to the same object, so care must be taken that the type and extent specified in the definition are compatible with those specified by each function which references the data.

# 4 Project Plan

In this section, we will first introduce the project planning processes that involve in this project. And then we will discuss how we specify the language designs, and carry out development and testing.

## 4.1 Project Processes

### 4.1.1 Planning

During the first planning meeting, we first concentrate on defining the functionality, domain, and the nature of our language. We agree that our language must be a light-weight, domain-specific, and user-friendly language that focuses on spoken dialogue management and natural language processing. We realize that with the current general-purposed languages, it is very difficult and inefficient in analyzing the deep syntactic structure of natural language. To date, in order to derive a syntax tree of natural language, programmers need to try various APIs, and have to deal with all kinds of problems (e.g. training a statistical model, configure the class-path, etc.). Most application developers and programmers do not have the background in natural language processing, so if we can provide programmer a full syntax parsing tree with only one line of ***Spoke***, how cool it is?

In this meeting, we elect Wang as the Project Manager, since he has many years of research experience in Natural Language Processing and Spoken Dialogue Systems. We also elect Tsai as the Chief Technical Officer of this project, as his research is mainly focused on system, security and programming language. Chen and Yu will be responsible for programming and testing, especially for backend JAVA functionality.

### 4.1.2 Specification

After we have a clear picture about our language, we focus on designing the components of our translator. During the first several meetings, we decide to implement a compiler, but not an interpreter. We feel that the interpreter structure is too easy, and might not be robust to handle complex natural language syntax and semantic representations.

Once we have made up our minds about pursuing the general compiler structure, we further discuss how we should specify the detail components of our compiler. During the first development cycle, we agree to use OCAML for building the scanner, parser, and JAVA source translator, because it is very efficient. We consider using JAVA as back-end, because we know there are existing JAVA libraries that have good performance on deep linguistic analysis.

When we actually involve in the compiler implementation, we feel that there is the need to include the IR code representation, syntax check, semantic check, and JAVA AST representation to our compiler. The rationale is that if we only do direct translation, there could be a lot of potential bugs in the JAVA source translation. For example, we might have problems handling scoping issues.

### 4.1.3 Development

In our initial responsibility assignment, we have Tsai on building the natural language components, Chen building the basic features, and Yu focus on machine learning techniques. However, we realize Tsai is more capable and efficient in building basic features, so Xin and Yu move to the JAVA backend development. Wang join Tsai in designing and implementing natural language features. As the target users of ***Spoke*** language are application developers who might not have machine learning knowledge, we decide to drop this emphasis.

The planned development will contain three main cycles. In the first cycle, we focus on the development of scanner, parser, and basic features. In parallel, we also start the development of backend JAVA Spoke data structures. In the second development cycle, we mainly work on developing advanced natural language processing feature, which involves support from scanner, parser, as well as backend special data structures. In the final cycle of development, we augment the compiler architecture with IR code representation and JAVA AST translation. We also start to combine and connect all components from the front end to the back end.

#### 4.1.4 Testing

We plan to have four main sections of testing: syntax test, semantic test, integration test, and application test. The purpose of syntax test is to check the robustness of our scanner, parser and syntax checker. We conduct the semantic test in order to check our semantic checkers (e.g. scoping, types and variables.) The purpose of compiler integration and application integration tests is to check the entire compiler and system robustness when processing advanced natural language syntax data structures.

In the testing, we plan to establish more than 50 test cases. For each experiment, we first predict the output of the test, and also predict the potential error types. Then, we run this experiment and see if it matches to our prediction. In our plan, we hope to handle errors and exceptions in front-end compilers, rather than the Java backend, as the purpose of this class is to develop our own compiler that handles errors and exceptions, rather than letting JAVAC to figure it out.

### 4.2 Programming Style Guideline

In the implementation of ***Spoke*** language, we try to follow the official OCAML guideline[1]. For the JAVA programming in our back-end, we use a well accepted JAVA programming style guideline[2]. In implementation, we try to follow the guidelines as precise as possible.

#### 4.2.1 Indentation

[1] http://caml.inria.fr/resources/doc/guides/guidelines.en.html
[2] http://geosoft.no/development/javastyle.html

In the implementation of ***Spoke*** language, we try to follow the official OCAML indentation law. We also have this law consistent through our implementation.

**Landin's pseudo law**: Treat the indentation of your programs as if it determines the meaning of your programs.

Below is a code snippet to represent the indentation style in our code:

```
targ:
    WORD WORD      { Tag(String($1),  String($2)) }
  | WORD STAR      { Tag(String($1),  Any) }
  | STAR WORD      { Tag(Any, String($2))  }
  | STAR STAR      { Tag(Any, Any) }
  | WORD targ_list { Tag(String($1),  Newlist($2)) }
  | STAR targ_list { Tag(Any, Newlist($2)) }
  | targ_list      { Tag(Any, Newlist($1)) }
```

In JAVA back end, we have the following sample code as programming style guideline.

```
package org.spoke;

public class SpokeTag extends SpokeObject {
  private SpokeObject myTag;
  private SpokeObject myObject;

  public SpokeTag(SpokeObject tag, SpokeObject object) {
          setTag(tag);
          setObject(object);
  }
}
```

### 4.2.2 Vertical Alignment

In OCAML, we carefully align the arrows and bars of a pattern matching conditional statement, though OCAML official guideline does not recommend this style. Here is an example:

```
let f = function
  | C1          -> 1
  | Long_name _ -> 2
  | _           -> 3;;
```

In Java, we follow the official guideline, and use left alignment for the programming style:

```
if      (a == lowValue)    compueSomething();
else if (a == mediumValue) computeSomethingElse();
else if (a == highValue)   computeSomethingElseYet();

value = (potential        * oilDensity)   / constant1 +
        (depth            * waterDensity) / constant2 +
        (zCoordinateValue * gasDensity)   / constant3;

minPosition     = computeDistance(min,     x, y, z);
averagePosition = computeDistance(average, x, y, z);

switch (phase) {
  case PHASE_OIL   : text = "Oil";   break;
  case PHASE_WATER : text = "Water"; break;
  case PHASE_GAS   : text = "Gas";   break;
}
```

### 4.2.3 Spaces

For the use of spaces, we are following the pseudo spaces law for both OCAML and JAVA implementation:

**Pseudo spaces law**: never hesitate to separate words of your programs with spaces; the space bar is the easiest key to find on the keyboard, press it as often as necessary!

### 4.2.4 Tabs

We avoid using tabs because it is not recommended by both OCAML and JAVA official programming style guideline.

## 4.3 Project Timeline

| # Phase | Date | Goal | Person-in-charge |
|---|---|---|---|
| 1 | 09/28/2010 | Proposal | PM |
| 2 | 10/10/2010 | Language Reference Manual<br>Scanner and Parser | CTO |

| | | | |
|---|---|---|---|
| 3 | 11/30/2010 | Basic Features | CTO |
| 4 | 12/10/2010 | Advanced NLP Features | PM |
| 5 | 12/15/2010 | Integration of Translator | CTO |
| 6 | 12/20/2010 | Final Language Implementation | CTO |
| 7 | 12/21/2010 | Testing and Debugging | CTO |
| 8 | 12/22/2010 | Demo and Presentation | PM |
| 9 | 12/22/2010 | Final Report | PM |

## 4.4 Roles and Responsibilities

| **Person** | **Role** | **Responsibilities** |
|---|---|---|
| Wang | Project Manager | 1. Create, maintain, and update schedules and progress, and coordinate with team members and the teaching staff.<br>2. Design and implement natural language parsing, tagging and advanced features throughout the front-end translator to backend implementation.<br>3. Design language scenarios, samples and integrate and test *spoke* language application.<br>4. Organize and document meetings.<br>5. Responsible for Proposal, LRM, Demo and final report. |
| Tsai | Chief Technical Officer | 1. Augment MicroC scanner and parser with basic features of *Spoke* language.<br>2. Responsible for the implementation and integration of front end translator, including Bytecode IR, Java AST translation.<br>3. Implement basic syntax, grammar and semantic checkers.<br>4. Coordinate back-end JAVA data structures and API implementation with front-end designs.<br>5. Collaborate with other members on advanced feature design and implementation. |
| Chen | Programmer | 1. Implementation of JAVA source code translation.<br>2. Responsible for JAVA back-end data structures. |

|  |  |  |
|---|---|---|
|  |  | 3. Collaborate with other members on back-end integration.<br>4. Responsible for syntax test. |
| Yu | Programmer | 1. Implementation of JAVA backend.<br>2. Collaborate with other members on back-end advanced features integration.<br>3. Responsible for back-end APIs.<br>4. Responsible for semantic test. |

### 4.5 Tools, languages, environments, and resources

We use Linux as official environment, since it has great command line support.
In our implementation, OCAML is the front end compiler implementation language, and JAVA is the back end implementation language. We use the following tools and resources in our implementation:

### 4.6 Project Log

#### 4.6.1 Selected Meeting Log

In this semester, the project leader has sent out ***163 emails*** to all team members. Here I post two selected meeting logs that represent different development cycles of our project:

(1) 12/02/2010
William Y. Wang <yw2347@columbia.edu> Thu, Dec 2, 2010 at 12:51 PM
To: Chia-che Tsai <ct2459@columbia.edu>, Wang Yang <yw2347@columbia.edu>, Xin Chen <xc2180@columbia.edu>, Zhou Yu <zy2147@columbia.edu>
Reply | Reply to all | Forward | Print | Delete | Show original
Hi guys,

Just a quick summary of today's meeting:

1. TA mentioned that there will be a final demo session 3-4 days before the final report due.
   The final demo will probably consist of three sections:
   1) Presentation
   2) Code Demo: part1. Compilation  part2. Demo of application
   3) Q&A part
2. We will need to form a concrete schedule and have new job assignment to wrap up the project.
3. We decide to finish the main compiler a week before the demo, so that we could have a week to integrate all the parts.
4. That will be a group meeting Friday (tomorrow) at 10am to check progress and assign new roles.

Thanks,
William

(2) 10/21/2010

William Y. Wang <yw2347@columbia.edu> Thu, Oct 21, 2010 at 1:28 PM
To: Chia-che Tsai <ct2459@columbia.edu>, Xin Chen <xc2180@columbia.edu>, zy2147@columbia.edu
Reply | Reply to all | Forward | Print | Delete | Show original
Hi all,

Here's a brief summary of Thursday meeting with TA:

1. Finalized our language architecture with our TA:
   For the architecture of our language, we decide to use the following one:

Our source code ----> Scanner + Parser in OCaml ----> AST ---> .Java file ---> Java complied classes ---> executable files

In this case, the intermediate code is java. And it's much easier for us to consider JAVA nlp APIs.

2. Identified the timeline and jobs of the LRM due on Nov. 3rd

Note that we have a LRM due on Nov. 3rd. So basically, we have three tasks:
1) The development of scanner, parser and a demo of sample hello world program (2 persons)
2) The documentation of LRM (1 person)
3) The design of NLP API. Namely, how we can incorporate the parser or the part-of-speech tagger in our language. (1 person)

Tsai, our CTO, will lead the development of scanner, parser and the hello world program. He needs one person to help him.
I will focus on documentation of the LRM and I can also help in developing NLP APIs into our language.

Xin and Yu, please let me know what sub-task you would like to work on or any suggestions you have.

Thanks,
William

### 4.6.2 SVN log

```
------------------------------------------------------------------------
r64 | chiache.tsai | 2010-12-22 05:41:57 -0500 (Wed, 22 Dec 2010) | 1 line
------------------------------------------------------------------------
r63 | chiache.tsai | 2010-12-22 05:41:09 -0500 (Wed, 22 Dec 2010) | 1 line
------------------------------------------------------------------------
r62 | chiache.tsai | 2010-12-22 05:40:30 -0500 (Wed, 22 Dec 2010) | 1 line
------------------------------------------------------------------------
r61 | chiache.tsai | 2010-12-22 05:39:34 -0500 (Wed, 22 Dec 2010) | 1 line
------------------------------------------------------------------------
r60 | chiache.tsai | 2010-12-22 00:42:01 -0500 (Wed, 22 Dec 2010) | 1 line
------------------------------------------------------------------------
r59 | chiache.tsai | 2010-12-21 23:27:57 -0500 (Tue, 21 Dec 2010) | 1 line
```

```
------------------------------------------------------------------------
r58 | chiache.tsai | 2010-12-21 19:57:36 -0500 (Tue, 21 Dec 2010) | 1 line
------------------------------------------------------------------------
r57 | chiache.tsai | 2010-12-21 18:33:18 -0500 (Tue, 21 Dec 2010) | 1 line
------------------------------------------------------------------------
r56 | chiache.tsai | 2010-12-21 17:06:15 -0500 (Tue, 21 Dec 2010) | 1 line
------------------------------------------------------------------------
r55 | chiache.tsai | 2010-12-21 08:56:46 -0500 (Tue, 21 Dec 2010) | 1 line
------------------------------------------------------------------------
r54 | wangwilliamyang | 2010-12-21 07:16:39 -0500 (Tue, 21 Dec 2010) | 2 lines
add the docs
------------------------------------------------------------------------
r53 | chiache.tsai | 2010-12-21 07:09:17 -0500 (Tue, 21 Dec 2010) | 2 lines
Working version
------------------------------------------------------------------------
r52 | chiache.tsai | 2010-12-21 03:58:14 -0500 (Tue, 21 Dec 2010) | 1 line
readme
------------------------------------------------------------------------
r51 | chiache.tsai | 2010-12-21 03:57:46 -0500 (Tue, 21 Dec 2010) | 2 lines
good not enough
------------------------------------------------------------------------
r50 | chenxinivy@gmail.com | 2010-12-20 21:44:26 -0500 (Mon, 20 Dec 2010) |
2 lines
add the test parser and scanner
------------------------------------------------------------------------
r49 | wangwilliamyang | 2010-12-20 21:34:18 -0500 (Mon, 20 Dec 2010) | 1 line
------------------------------------------------------------------------
r48 | chiache.tsai | 2010-12-20 20:52:33 -0500 (Mon, 20 Dec 2010) | 1 line
no isEmpty
------------------------------------------------------------------------
r47 | wangwilliamyang | 2010-12-20 20:29:51 -0500 (Mon, 20 Dec 2010) | 2
lines
add the NLP tagger
------------------------------------------------------------------------
r46 | wangwilliamyang | 2010-12-20 20:06:36 -0500 (Mon, 20 Dec 2010) | 2
lines
update nlp parser
------------------------------------------------------------------------
r45 | wangwilliamyang | 2010-12-20 19:50:43 -0500 (Mon, 20 Dec 2010) | 2
lines
```

```
NLPPARSER modified
------------------------------------------------------------------------
r44 | chiache.tsai | 2010-12-20 16:51:47 -0500 (Mon, 20 Dec 2010) | 1 line
good
------------------------------------------------------------------------
r43 | chiache.tsai | 2010-12-20 16:44:24 -0500 (Mon, 20 Dec 2010) | 1 line
good
------------------------------------------------------------------------
r42 | chiache.tsai | 2010-12-20 12:26:59 -0500 (Mon, 20 Dec 2010) | 1 line
new API
------------------------------------------------------------------------
r41 | chiache.tsai | 2010-12-20 06:12:41 -0500 (Mon, 20 Dec 2010) | 1 line
rm api
------------------------------------------------------------------------
r40 | chiache.tsai | 2010-12-20 05:57:54 -0500 (Mon, 20 Dec 2010) | 1 line
Good code
------------------------------------------------------------------------
r39 | wangwilliamyang | 2010-12-19 19:21:38 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r38 | wangwilliamyang | 2010-12-19 19:18:39 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r37 | wangwilliamyang | 2010-12-19 19:16:44 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r36 | wangwilliamyang | 2010-12-19 18:35:59 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r35 | wangwilliamyang | 2010-12-19 16:32:56 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r34 | chiache.tsai | 2010-12-19 16:24:04 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r33 | chiache.tsai | 2010-12-19 16:20:50 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r32 | wangwilliamyang | 2010-12-19 15:47:33 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r31 | chiache.tsai | 2010-12-19 15:30:01 -0500 (Sun, 19 Dec 2010) | 1 line
------------------------------------------------------------------------
r30 | chiache.tsai | 2010-12-19 15:09:41 -0500 (Sun, 19 Dec 2010) | 1 line
rm org
------------------------------------------------------------------------
r29 | wangwilliamyang | 2010-12-19 15:05:20 -0500 (Sun, 19 Dec 2010) | 2 lines
add changes
```

```
------------------------------------------------------------------------
r28 | chiache.tsai | 2010-12-17 06:51:30 -0500 (Fri, 17 Dec 2010) | 2 lines
good version
------------------------------------------------------------------------
r27 | wangwilliamyang | 2010-12-17 00:25:28 -0500 (Fri, 17 Dec 2010) | 1 line
------------------------------------------------------------------------
r26 | wangwilliamyang | 2010-12-16 23:01:03 -0500 (Thu, 16 Dec 2010) | 1 line
------------------------------------------------------------------------
r25 | chiache.tsai | 2010-12-16 22:14:49 -0500 (Thu, 16 Dec 2010) | 1 line
tokenbuf
------------------------------------------------------------------------
r24 | chiache.tsai | 2010-12-16 16:23:01 -0500 (Thu, 16 Dec 2010) | 1 line
good parser
------------------------------------------------------------------------
r23 | chiache.tsai | 2010-12-11 20:56:08 -0500 (Sat, 11 Dec 2010) | 1 line
javaast
------------------------------------------------------------------------
r22 | chiache.tsai | 2010-12-05 22:57:51 -0500 (Sun, 05 Dec 2010) | 1 line
------------------------------------------------------------------------
r21 | chiache.tsai | 2010-12-03 11:22:25 -0500 (Fri, 03 Dec 2010) | 3 lines
remove op = bop | uop
------------------------------------------------------------------------
r20 | chiache.tsai | 2010-12-03 11:19:58 -0500 (Fri, 03 Dec 2010) | 3 lines
all new codes
------------------------------------------------------------------------
r19 | chiache.tsai | 2010-11-09 13:21:13 -0500 (Tue, 09 Nov 2010) | 2 lines
new interface
------------------------------------------------------------------------
r18 | chiache.tsai | 2010-11-03 17:46:43 -0400 (Wed, 03 Nov 2010) | 2 lines
Good Java
------------------------------------------------------------------------
r17 | chiache.tsai | 2010-11-03 15:50:24 -0400 (Wed, 03 Nov 2010) | 1 line
Good Parser
------------------------------------------------------------------------
r16 | chiache.tsai | 2010-11-03 14:26:07 -0400 (Wed, 03 Nov 2010) | 1 line
reorganize java backend
------------------------------------------------------------------------
r15 | chiache.tsai | 2010-11-03 10:35:25 -0400 (Wed, 03 Nov 2010) | 2 lines
Working Parser
------------------------------------------------------------------------
```

```
r14 | chiache.tsai | 2010-11-02 23:27:34 -0400 (Tue, 02 Nov 2010) | 1 line
compilable version
------------------------------------------------------------------------
r13 | chiache.tsai | 2010-11-02 23:26:11 -0400 (Tue, 02 Nov 2010) | 1 line
------------------------------------------------------------------------
r12 | zhouyuyz | 2010-11-01 22:31:36 -0400 (Mon, 01 Nov 2010) | 1 line
------------------------------------------------------------------------
r11 | chiache.tsai | 2010-11-01 19:29:12 -0400 (Mon, 01 Nov 2010) | 1 line
test
------------------------------------------------------------------------
r10 | chiache.tsai | 2010-11-01 19:24:12 -0400 (Mon, 01 Nov 2010) | 1 line
test backend
------------------------------------------------------------------------
r9 | zhouyuyz | 2010-11-01 19:06:11 -0400 (Mon, 01 Nov 2010) | 1 line
------------------------------------------------------------------------
r8 | chiache.tsai | 2010-10-28 22:18:20 -0400 (Thu, 28 Oct 2010) | 1 line
*
------------------------------------------------------------------------
r7 | chiache.tsai | 2010-10-28 21:04:57 -0400 (Thu, 28 Oct 2010) | 1 line
*
------------------------------------------------------------------------
r6 | chiache.tsai | 2010-10-28 15:25:06 -0400 (Thu, 28 Oct 2010) | 1 line
------------------------------------------------------------------------
r5 | chiache.tsai | 2010-10-28 15:24:04 -0400 (Thu, 28 Oct 2010) | 2 lines
*
------------------------------------------------------------------------
r4 | chiache.tsai | 2010-10-28 15:21:54 -0400 (Thu, 28 Oct 2010) | 2 lines
initial scanner
------------------------------------------------------------------------
r3 | chiache.tsai | 2010-10-28 13:57:25 -0400 (Thu, 28 Oct 2010) | 2 lines
migrate from micro C
------------------------------------------------------------------------
r2 | chiache.tsai | 2010-09-26 21:39:06 -0400 (Sun, 26 Sep 2010) | 2 lines
Sample code
------------------------------------------------------------------------
r1 | (no author) | 2010-09-23 18:27:52 -0400 (Thu, 23 Sep 2010) | 1 line
Initial directory structure.
```

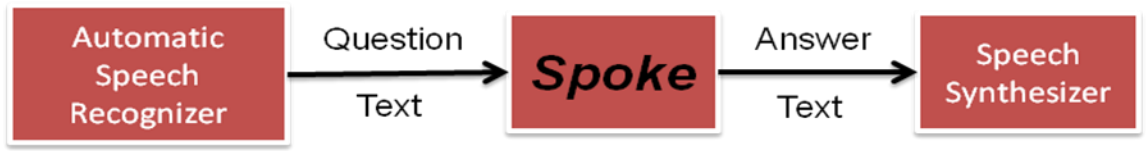
# 5 Architecture Design

## 5.1 Overview to Architecture

The architecture of Spoke compiler contains both sufficient implementation of general-purpose compilers and specific design on the purpose of natural language processing. We assume the users of Spoke language are non-professional developers in need of implementing spoken dialog systems in industries. Users of Spoke language may implement they own spoken dialog system without deep understanding to natural language processing. In some other cases, Spoke language can be used to develop general-purposed application, since we provides adequate data types, operators and built-in APIs.

The following figure shows the architecture of our expectation where the Spoke language can be used in common spoken dialogue system. In most spoken dialog solutions, the system is wrapped by two components of speech processing separately. The two components are Automatic Speech Recognizer and Speech Synthesizer. The task of Automatic Speech Recognizer is to use voice recognition technology to adaptively convert speeches into correspondent texts. Speech Synthesizer works in the opposite way, in which texts are converted into speeches that users may understand.

Working as an intermediate component, a Spoke program is capable of doing text-to-text processing on natural languages. Although Spoke is not designed as a language that implements applications on data sources others than text (for example, speech or vision), the generosity of Spoke language must make it applicable on processing or translation of other form of data.



Make an example of spoken dialog system in the real world. A financial company may be in need of a system that automatically responds to clients' requests for retrieval of financial information. Assuming clients may request information through phones, the questions or commands are submitted by clients in the form of speech. Suppose that the company already owns solutions of speech recognition and speech synthesizing, the developers working for the company can use Spoke language to write a responding system that waits for questions or commands in text and provides satisfactory answers or reactions.

Since Spoke owns a mature library of natural language processing, developers need not to preprocess the text by any other natural language analyzers. We provide natural language taggers and parsers as part of the built-in APIs.

The concentration on spoken dialog systems determines the principal of design of this language and its compilers. A spoke program is expected to run as a daemon or an application that serves on one or multiple inputs. The design of data types, operators and APIs will focus on string manipulation and tree structure construction.

## 5.2 Overview of Components

The implementation of Spoke compiler is divided into components that either does translation from one language to another or perform analysis on a specific language. Components are connected in sequence, and each component shares a form of languages with the one prior to it. Each component should not translate the incoming language as less concrete or more information, or it cannot be part of a good compiler.

Spoke uses Java language as the backend of the compiler. Instead of directly compiling Spoke source codes into machine codes, Spoke compiler translates source code into a well-structured, syntax-correct java source code. Later the Spoke compiler calls Java compiler as an external component to compile the generated Java source codes into Java byte codes, which can be loaded and executed in Java Virtual Machine.

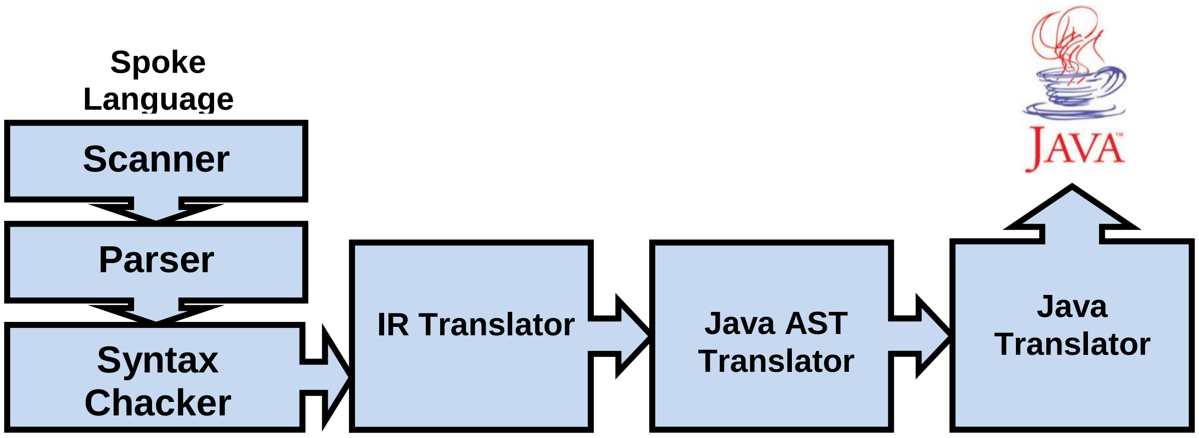
The translation of Java languages determines the principal of design of Spoke compiler. We followed the following principals in the process of implementation of Spoke compiler:

1. The language of Spoke language and its intermediate languages must be more preferable for translation to Java that other languages. In order word, the design of the language shared by component must bring advantages to Java translation.

2. The generated Java source codes must be robust enough to survive all the syntax checks of Java compilation. Any possible syntax error semantic error that can be caught or complained by Java compiler must be handled by one of the components of Spoke compiler.

3. Suppose the generated Java source codes can be explicitly compiled by Jav compiler. The Java byte codes must not cause the Java internal library to throw any exception in the runtime. For those errors that can only be dynamically detected during execution, they must only be caught and issued backend implementation of Spoke compiler.

The following figure shows the components the forms the compiler.



Each of the components in the Spoke compiler is given a specific task, based on processing of certain form of language:

1. **Scanner:**
   The task of scanner is to match the source character by character and determine whether the source satisfied the regular expression of the language. The result of scanning is a sequence of tokens, each of which represents a key word, a special character or a constant in the language. A error in the scanner indicate that the source contains key words that the language does not recognize, or data format that mismatches the rule.

2. **Parser:**
   The task of parser is to match tokens with grammar of the language. If the tokens failed to arrive at a terminated state of the deterministic finite

automata representing the grammar, the parser complains that parsing fails. In addition, the parser will perform initial analysis on the variable usage and provides information for memory allocation at the runtime. The result of the parser is a tree structure, Abstract Syntax Tree, that shows the syntax of the source codes.

3. **Syntax Checker:**
   The Syntax tree that the parser accepted might not be necessary correct. The parser is responsible for analysis the basic structures of the source codes, but not to check if every rule of the language. Some of rule is less representative, and more difficult to check. A syntax checker will focus on those rules one by one, either issue a warning or fix the source code. The detail of the syntax checks in Spoke compiler will be mentioned in a later section.

4. **Intermediate Representation (IR) Translator:**
   The syntax tree will be translated to a less structural, less readable language that is neutral to the source language and the target language of the whole compiler. The characteristic of the intermediate representation is that it is commonly sequential instead of hierarchical. All the blocks, statement and expressions will be break into a sequence of instructions. The identifiers and keywords in source codes will be replaced by numerical representations.

5. **Java Abstract Syntax Tree (AST) Translator:**
   Before translating intermediate code into Java source code, the compiler goes through a process of translating it into a former representation of Java classes. A Java class is a strictly constructed hierarchical structure, with definition of methods and fields. The Java AST defined the scopes of class members. For those statements in the methods, we designed a way of representation to define them. The Java AST we defined is similar enough to formal Java AST representation, except we only provide definition of statements to the least level.

6. **Java Translator:**
   The Java AST can be easily translated to Java source code, just using a line-by-line approach.

## 5.3 Intermediates of Components

### 5.3.1 Scanner to Parser - Tokens

Tokens represent individual units of words, constants or special characters. Some of the tokens represent an explicit unit (in other words, “a terminal”) while the others wrap up a value. The following table shows all the tokens recognizable by the Spoke compiler.

| | | | |
|---|---|---|---|
| LPAREN | Left Parenthesis | ELIF | Key word “elif” |
| RPAREN | Right Parenthesis | FI | Key word “fi” |
| COMMA | Comma | FOR | Key word “for” |
| LBRACK | Left Bracket | IN | Key word “in” |
| RBRACK | Right Bracket | NEXT | Key word “next” |
| COLON | Colon | WHILE | Key word “while” |
| PLUS | Plus | LOOP | Key word “loop” |
| MINUS | Minus | CONTINUE | Key word “continue” |
| TIMES | Time (*) | BREAK | Key word “break” |
| DIVIDE | Division (/) | FUNCTION | Key word “func” |
| MODULUS | Modulus Division (%) | END | Key word 0 |
| ASSIGN | Assignment (=) | RETURN | Key word “return” |
| EQ | Equal to (==) | GLOBAL | Key word “global” |
| NEQ | Not equal to (!=) | BOOL(b) | Boolean Value b |
| LT | Less than (<) | INTEGER(i) | Integer Value i |
| LEQ | Less than or equal (<=) | FLOAT(f) | Float Value f |
| GT | Greater than (>) | ID(s) | ID as string s |
| GEQ | Greater or equal (>=) | STRING(s) | String s |
| BELONG | Belongs to (~) | LTPAREN | Left Parenthesis |
| AND | And (&&) | RTPAREN | Right Parenthesis |
| OR | Or (\|\|) | WORD | Word as string s |
| NOT | Not (!) | NULL | Key word “null” |
| IF | Key word “if” | STAR | Wildcard character |
| THEN | Key word “then” | EOL | End of Line |
| ELSE | Key word “else” | EOF | End of File |

### 5.3.2 Parser / Syntax Checker to IR Translator – Abstract Syntax Tree

AST is constructed by nodes that represent each unit of statement or expression of the easiest form. A node of expression may be a leaf of the tree or be parent of nodes of expressions. A node of statement may be a leaf, or a parent of nodes that can be either of expressions or of statements.

### 5.3.2.1 Top Level Syntax

The top level of Abstract Syntax Tree is a main program and a multiple number of functions. The syntax of spoke language allows function definition insides statements. Therefore one of the task of Parser is to separate the body of each function from the main program. These functions are stored in a lists, in the order that they are declared.

The definition of the top level of AST is as follows.

```
Program := Main Program * List of Function
Main Program := { Names of Global Variables: List of String;
                  Names of Local Variables: List of String;
                  Code Body: List of Statement }
Function :=  { Name: String;
               Name of Arguments: List of String;
               Names of Global Variables: List of String;
               Names of Local Variables: List of String;
               Code Body: List of Statement }
```

### 5.3.2.2 Statement Syntax

Statements in a Spoke code are individual representation of instructions that are terminated by end-of-line. Some statements are non-recursive, which contains no statements in its descendents. The others, such as loops or branches, contain one or more blocks of statements.
The definition of statement syntax is as follows.

```
Statement :=    Expression: Expression
             |  Return: Expression
             |  Continue
             |  Break
             |  Branch: Expression * List of Statement * List of Statement
             |  For Loop: String * Expression * List of Statement
             |  While Loop: Expression * List of Statement
             |  No Expression
```

### 5.3.2.3 Expression Syntax

Expressions are the smallest units in the Spoke language. Unlike statements, each expression should have a value of specific type. Expressions with operators may wrap up other expressions as their descendent.

The definition of expression syntax is as follows.

```
   Expression :=    Boolean  |  Integer | Float  | String
                  | Variable: string
                  | New List: List of Expression
                  | Call Function:  String * List of Expression
                  | Binary Operation: Operator * Expression * Expression
                  | Tag: Expression * Expression
                  | Unary Operation: Operator * Expression
                  | Assign to Variable: String * Expression
                  | Assign to Element: Expression * Expression * Expression
                  | Value of Element: Expression * Expression
                  | Partition of List: Expression * Expression * Expression
                  | Any  | Null
```

### 5.3.3 IR Translator to Java AST Translator – Spoke Intermediate Code

In order to translate Abstract Syntax Tree to a form of language that is neutral to any language, we design the intermediate syntax, named as Spoke Intermediate Code. Spoke Intermediate Code looks similar to byte codes of several languages on virtual machines or Assembly as the representation of machine codes. However the syntax of Spoke Intermediate Code is different from those languages in some details, since it is designed to target Java Code Translation.

There are four characteristics of the syntax of Spoke Intermediate Code:

1. Spoke Intermediate Code is not strictly enumerated. In some cases, the syntax allows assignment of strings or Booleans. The reason that we didn't define it strictly enumerated is that Spoke Intermediated Code is not designed to be byte codes.

2. Spoke Intermediate Code does not restrict the number of attribute of each instruction. Some of them might have no attribute, while others have as much as four attributes. The reason of this design is the same as the previous one: Spoke Intermediate Code is not Byte Code.

3. Java does not have a convenience structure for stack manipulation. Unlike C language, in which programmers can manipulate pointers, Java language has no reference on address. Although Java does provide a Stack class in library, it is less reliable and efficient to use. In fact, Java is designed with scoping as stack-based. All the local variables of Java are located on stacks. What is more, Java has very good garbage collecting mechanism, so that we do not need to worry about memory leaks of local variables.

   As a result, we design operations in Spoke Intermediate Code using temporary local variables. Only instructions like “Load” or “Store” will touch real variables (no matter local or global). The others will performance operations on temporary variables. We keep recording the usage and temporary local variables, and to make sure wanted variables are not overwritten and unwanted variables are reused.

   In fact, this design is much more like byte code of LLVM (Low Level Virtual Machine). In fact, we can make an assumption that this design rarely exhausts stacks. The number of temporary variables required will be *O(n)* for *n*-level branches or loops.

4. Java has no Jump statement. It is not possible to translate branching or loops into conditional jumps as other translators do. As an alternative, branches and loops in Spoke Intermediate Code are represented by number of lines contained in the blocks instead of distance to jump. For example, an instruction Br(1,10,5) indicates that when temp1 is true the following ten instructions will be executed, otherwise, another five lines will be executed. This is also an advantage on translation into Java codes, since the compiler can easily translate this instruction into structures with blocks.

The following shows a list of the syntax of Spoke Intermediate Code:

| | | |
|---|---|---|
| Gl | Integer | (* Global variables *) |
| Lc | Integer | (* Local variables *) |
| Cb | Integer * bool | (* Constant bool *) |
| Ci | Integer * Integer | (* Constant Integereger *) |
| Cf | Integer * float | (* Constant float *) |
| Cs | Integer * String | (* Constant String *) |
| Ay | Integer | (* Any Object *) |
| Nl | Integer | (* Null Object *) |
| Wp | Integer * Integer | (* Wrap list *) |

```
Pt      Integer * Integer * Integer * Integer      (* Get partition *)
El      Integer * Integer * Integer                (* Get element *)
St      Integer * Integer * Integer                (* Set element *)
Ll      Integer * Integer                          (* Load local variable *)
Sl      Integer * Integer                          (* Store local variable *)
Lg      Integer * Integer                          (* Load global variable *)
Sg      Integer * Integer                          (* Store global variable *)
Bn      Integer * Op * Integer * Integer           (* Binary operation *)
Un      Integer * Op * Integer                     (* Unary operation *)
Fn      Integer                                    (* Function *)
Rt      Integer                                    (* Return *)
Cl      Integer * Integer * Integer                (* Call Function *)
Tg      Integer * Integer * Integer                (* Tag Object *)
Br      Integer * Integer * Integer                (* Branch *)
Lp      Integer                                    (* Loop *)
Bk                                                 (* Break *)
Ct                                                 (* Continue *)
```

### 5.3.4 Java AST Translator to Java Translator – Java AST

We define Java Abstract Syntax Tree as the formal definition of Java Class. Our definition of Java Class is a partial implementation of the official Java Abstract Syntax Tree. In fact, we do not intend to translate out intermediate code into Java Byte Code. Therefore we only define the least syntax necessary for the Spoke compiler.
The following is a formal definition of our Java AST.

```
JClass := { Package : JID;
            Accessor : JACCESS;
            Name : JID;
            Super Class : JID;
            Fields : List of JField;
            Methods : List of JMethod; }
JFiled := { Name : JID;
            Accessor : JID;
            Type: JID; }
JMethod := { Name : JID;
             Accessor : JID;
             Static : Boolean;
```

```
                    Argument : List of JID * JID;
                    Return Type: JID;
                    Body : List of JStatement }
      JID : = String
      JACCESS : = Public | Private | Protected | None
     JStatement :=   JDef : jtype * JID                          (* Type Declaration *)
                   | JNew : jtype * String list * JID            (* Allocation *)
                   | JInvoke : JID * String list * JID           (* Function Call *)
                   | JThrow : JID * String list                  (* Exception Throw *)
                   | JInstance: : JID * JID * JID                (* Instance Of *)
                   | JIf : String * jstmt list * jstmt list      (* Branch *)
                   | JFor : JID * String * jstmt list            (* For *)
                   | JWhile : String * jstmt list                (* Loop *)
                   | JBreak                                      (* Break *)
                   | JContinue                                   (* Continue *)
                   | JReturn : JID                               (* Return *)
```

## 5.4 Implementation of Components

In this section we will discuss the detail of implementation of each component.

### 5.4.1 Scanner

We keep Implementation of scanner as simple as possible. The only tricky thing is the scanning of tagged syntax tree for natural language processing. We make special rules to recognize the tagged syntax tree (wrapped by apostrophes) as early as in the scanner.

### 5.4.2 Parser

Two important tasks are performed by parsers. The first is to separate functions with main programs. The second is to recognize the declaration of local variables on first assignments. The parser will pass out the information of functions and local variables at the meanwhile of syntax parsing. The order of function names and variables will be kept as their declarations or first assignments.

### 5.4.3 Syntax Checker

The syntax checker will perform six checks on the incorrectness of syntaxes that parser may not detect. The six checks are as follows.

#### 5.4.3.1 Duplicated Definition of Built-in Variable

In Spoke language, we define three built-in variables, one is local variable and two is global variables. The local variable “arg” will be assigned as list of arguments passed at the beginning of function calls. “arg” also exists in main program as reference to command-line arguments. As well, global variables “match” and “star” is manipulated by built-in APIs. Those variables should not be declared as arguments of function or another global variable, or an error is issued in the compilation.

#### 5.4.3.2 Assignment to Built-in Variable

The three built-in variables should not be assigned variable in the user program. Any assignment to these variables will cause an error in the compilation.

#### 5.4.3.3 Return Statement outside Functions

A return statement should be written in a function. The checker goes through the statements of the main program and issues an error if find a return statement.

#### 5.4.3.4 Continue / Break Statement outside Loops

A continue statement or a break statement should not be written outside loops. The checker goes through the statements of the main program and function, holds their step when seeing a loop, and issues an error if find a continue statement or break statement.

#### 5.4.3.5 Functions not Closed by Return Statement

Java has a concrete definition of its language and performs sufficient syntax check on source codes. A function with non-void return type will be complained by Java compiler if the user forgets to put return statement at the end. Here syntax checker perform an easy analysis statically on functions, and detect the missing of return statements. This error will be automatically fixed without warning or exception. The checker fixes it by putting a return statement at the end of function.

#### 5.4.3.6 Statement after Return Statement

Java Syntax Checker also complains about the existence of statement after a return statement. This condition is warned as "Unreachable statements". In order to quit Java Syntax Checker, we perform checks on unreachable statements and warned it in the compilation time.

This checking is more like a static analysis. In fact, checking for unreturned function encounters the same problem. The issue is on branches. A branching with both blocks containing return statement will be considered returning in the function. The following example will be considered a returning function.

```
func foo(test)
    if test then
        return true
    else
        return false
    fi
end
```

**(Spoke)**

```
public  boolean foo(Boolean test)
{
    If ( test )  {
        return true;
    } else {
        return false
    }
}
```

**(Java)**

The analysis checker does on these condition is to pass flags in blocks of code to indicate if the block contains return statements. A branch will pass a flag of true if both of its blocks contain return statements.

### 5.4.4 IR Translator

The main task of compiler is on Intermediate representation Translator. IR Translator has to perform three most important tasks. The first is to handle scoping and naming. The second is to transform expression in trees as the simplest form of operation, with proper usage of temporary variable. The third is to translate statements into instructions that match the semantics.

#### 5.4.4.1 Scoping and Naming

The IR Translator maintains a String Map for the matching of numeric representation and string representation, for local variable global variables and functions. An access to a variable name not matched in either global variable list or local variable list will be considered not semantically corrected. The translator will issue an exception for undeclared variables.

Note that we have three built-in variables, two global ones and one local one, which should be put to the String Map for every function. These variables will be numbered statically. For example, the built-in local variable will always be numbered as 0.

We do the same static scoping on built-in APIs. We adopt the concept of system calls in operating systems. Each API is numbered as a fixed function number, so can be translated into enumerated form. Later the Java AST translator will translate it back to the real name of functions, which might be inconsistence at the first place where developers use them.

#### 5.4.4.2 Transformation of Expressions

Most of the operation of expressions needs usage of temporary variable. We have to keep a list of the used variables and recycle those variables not needed any more.

This problem is not actually difficult to solve. For every expression, it is for sure that exactly one value will be returned, stored in a temporary variable on the top of variable list. For retrieving this value, simply grab the first variable name on the list and use it in next operation.

#### 5.4.4.3 Translation of Statements

The translation of statements may be either simple or complex, depending on the semantic meaning of the statements. Take For loop in Spoke language as example. The For loops in Spoke Language is more like usages in scripting language like Bash or Python. The For loop works like an iterator on a list of objects. The translation of this statement may be very complex. The following codes shows how For loops are translated.

```
For a in list


next
```

**(Spoke)**

```
Load list to tmp1
Set tmp2 as -1
Begin to Loop
   Set tmp3 as 1
   Set tmp2 as tmp2 + tmp3
   Set tmp3 as len(list)
   Set tmp3 as tmp3 <= tmp2
   Branch on tmp3
        Break
   Set tmp3 as tmp1[tmp2]
   Store a as tmp3
```

**(Java)**

### 5.4.5 Java AST Translator

It appears to be straightforward to translate Spoke Intermediate Code into Java AST. The format of Spoke Intermediate Codes is designed specifically for Java Translation and theoretically can be matched into Java AST line to line.

However we want to put more works to make this translation robust. In a direct translation,  the Java source codes generated will suffered exception thrown by the backend. In order to prevent type mismatch on assignment, the generated code has to allocate new object for each assignment, which is not really economic. In fact, checks can be placed in the generated code an perform direct assignment of value instead of allocating a new object, in the case that the type is matched.

In fact, there are still several checks of these kinds placed in generated Java codes. For example, a non-enumerated variable being given to mathematical computation like subtraction, multiplication, divisions will throw a runtime exception rather than wait for backend to detect it. This reaction to the runtime failure is actually smarter than simply relies on the backend.

## 5.5 Contributions

William Yang Wang collaborated with all teammates and mainly contributed to the design and implementation of the natural language parsing and tagging features throughout the front end scanner, parser to JAVA back end. And he is also responsible for implementation and integration of the entire application system.

Chia-che Tsai implemented the front end scanner and parser based on MicroC language. He also implemented the Bytecode and Java AST parts of the translator. He designed, set up and improved the JAVA backend data structures according to the frontend. He collaborated with William on the design and implementation of advanced natural language features, and revised the implementation of JAVA backend.

Xin Chen is responsible for the implementation of JAVA source code translation. He also collaborated with Chia-che and Zhou on the implementation the JAVA backend data structures and APIs.

Zhou Yu is responsible for the JAVA backend implementation. He collaborated with all team members on the backend implementation of advanced natural language features.

# 6 Test Plan

This is our test plan for the Spoke language and our compiler. Our test plan consists of syntax test, semantic test (IR test), run time test and application test. The plan will have two levels of testing, First is unit testing to check whether our scanner, parser, java AST, semantic checking system and java backend works normally. The second level is to check whether our system works normally in dialog management and Natural language processing.

## 6.1 Spoke Test Program

### 6.1.1 Syntax Test

**(1) scanner**

| **Code ID** | **Test case** | **Expected result** | **Result** | **Problem** |
|---|---|---|---|---|
| **1.** # operators and operants | a 7 | | #pass | |
| | + - * / % | | #pass | |
| | = | | #pass | |
| | == | | #pass | |
| | != | | #pass | |
| | < | | #pass | |
| | <= | | #pass | |
| | >= | | #pass | |

| | | | | |
|---|---|---|---|---|
| | ~ | | #pass | |
| | && | | #pass | |
| | ! | | #pass | |
| | [] | | #pass | |
| | : | | #pass | |
| | (a,b) | | #pass | |
| **2.** # keywords | global true false if then ilif else fi for in next while loop break continue func retun end | | #pass | |
| **3.** #ID | a9 | | #ID(a9) EOL | |
| | a 9 | | #ID(a) INTEGER(9) EOL | |
| | a.a.a.a | | #compile error | |
| **4.** #Value | float 0.0.0 | | #compile error | |
| | 01.1 | | | |
| | float sss | | #ID(float) | |
| | float 1_1 | | #ID(float) INTEGER(1) ID(_1) | |
| **5.** #String | "" | | | |
| **6.** # estring: | '\" | | #compile error | |
| | " | | #compile error | |
| | ['\t' '\r'] | | | |
| **7.** # nstring: | '\" | | #compile error | |
| | " | | #compile error | |
| | _ | | #ID(_) EOL | |
| | '_\t' | | #STRING(_\\t) EOL | |
| | string SSS = "'string'" | | #ID(string) ID(SSS) EOL | |
| | string "" | | #ID(string) STRING() EOL | |
| | 'string' "\t\r\n" '\t\r\n' | | #STRING(string) STRING(\t\r\n) STRING(\\t\\r\\n) EOL | |
| | "1.2" | | #pass | |
| **8.** # tstring | `string` | | #pass | |
| | `*` | | #pass | |
| | `()` | | #pass | |
| | ("ssd" "") #LPAREN | | | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | STRING(ssd)<br>STRING()<br>RPAREN EOL |  |  |  |
|  | (sd sdf) #LPAREN<br>ID(sd) ID(sdf)<br>RPAREN EOL |  |  |  |
| **9.** #tag: | (NN I)` |  | #LTPAREN<br>TAG(NN)<br>WORD(I)<br>RTPAREN EOL |  |
|  | ("aa" a)` |  | #compile error |  |

## (2) parser

| **Code ID** | **Test case** | **Expected result** | **Result** | **Problem** |
|---|---|---|---|---|
| **1.** IF expression<br>IF expr THEN eol stmt_opt %prec NOELSE FI { If(fst $2, fst $5, []), snd $5 @ snd $2 }<br>#1) IF ( expression ) THEN statement<br>#2) IF ( expression ) THEN statemtent ELSE statement<br>#3) IF ( expression ) THEN statement<br>ELIF (expression) THEN statement<br>ELSE statement<br>FI | a = 1<br>if a == 1 then<br>a=2<br>fi |  | #pass |  |
|  | a = 1 |  | #compile error |  |
|  | if a == 1 then<br>a=2<br>elif a ==1.5 then<br>Return statement<br>a = 3<br>else a = 5<br>fi |  | #pass |  |
| **2.** # FOR element in [number of iterations]<br># statement<br># NEXT | a = 1<br>for a in [1,2,3]<br>a = 2<br>next |  | #pass |  |
|  | for a in [1,2,3] |  | #pass |  |

| | | | | |
|---|---|---|---|---|
| | next | | | |
| | for a in<br>[1,"","hello"]<br>a = 1<br>next | | #pass | |
| **3.**# WHILE<br>( expression )<br>statement LOOP<br># WHILE expr eol<br>stmt_opt LOOP<br>{ While(fst $2, fst<br>$4), snd $4 @ snd<br>$2 } | while (a<= "hello"<br>&& a>1 && a)<br>a = 3<br>a = a-1<br>loop | | #compile error | compile<br>bug:  (a<=<br>"hello" &&<br>a>1 && a) |
| | while (a<= "hello"<br>&& a>1 && a=1)<br>loop | | #compile error | |
| **4.** # Return<br>statement<br># RETURN<br>{ Return(Null), [] } | func foo(a , b)<br>return 1<br>end | | #pass | |
| # targ: | (a \`(a b)\`) | | # scanner :<br>LTPAREN<br>TAG(a) LPAREN<br>ID(a) ID(b)<br>RPAREN<br>RTPAREN EO | |
| | (a b)\` | | # scanner :<br>LTPAREN<br>TAG(a)<br>WORD(b)<br>RTPAREN EOL | |
| **5.** # Integretion test | a = [1,2,3]<br>for a in [2,3,4]<br>a=a+1<br>next | | #Expr(Assign(a<br>List(Int(1) Int(2)<br>Int(3)))) | |
| | if a == 1 then<br>func print(a)<br>end<br>fi | | # compile error,<br>bug might exist<br>here | |
| | while (a<= "hello"<br>&& a>1 && a)<br>func print(a)<br>end<br>loop | | # compile error | # have<br>compile<br>bug:  (a<=<br>"hello" &&<br>a>1 && a)<br>function<br>call can't<br>be nested<br>in while<br>loop |
| | func print(a)<br>end | | #pass | |
| | for i in range(0, 2) | | #pass | |

| | | | | |
|---|---|---|---|---|
| | print(i)<br>next | | | |

### 6.1.2 Semantic Test

| ID | Test case | Expected result | Result | Problem |
|---|---|---|---|---|
| **1.** # function definition | func print(a)<br>end<br>b= print (4) | | #pass | |
| | b=a(4) | | | #Fatal error: exception Failure("undefined function: a") |
| **2.** # variable definition | a=""<br>b= a | | #pass | |
| | b=a | | | #Fatal error: exception Failure("undefined variable: a") |
| **3.** # main function | if a == 1 then<br>return<br>else<br>return<br>fi | | #pass | |
| | if a == 1 then<br>return<br>else<br>return<br>fi<br>a=1 | | | #Fatal error: exception Failure("'return' in main function") |
| **4.** # return in function | func print(a)<br>return<br>end | | #pass | #<br>Gl 2<br>Fn 6<br>Lc 2<br>Ll 0 0<br>Ci 1 0<br>El 2 0 1<br>Sl 2 1<br>Rt -1<br>Lc 1 |
| | func print(a)<br>end | | #pass | #automatically add return in function (Rt is return in our intermediate code)<br>Gl 2<br>Fn 6<br>Lc 2<br>Ll 0 0<br>Ci 1 0<br>El 2 0 1 |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  |  | Sl 2 1<br>Rt -1<br>Lc 1 |
| **5.** # return in main function | a=1 |  |  |  |
|  | a=1<br>return |  | #pass | #Fatal error: exception Failure("'return' in main function") |
| **6.** # continue | a=1<br>while (a<= 10)<br>a=a+1<br>continue<br>loop |  |  |  |
|  | continue |  |  | #Fatal error: exception Failure("'continue' outside loops") |
| **7.** # break | a=1<br>while (a<= 10)<br>a=a+1<br>break<br>loop |  | #pass |  |
|  | break |  |  | #Fatal error: exception Failure("'continue' outside loops") |

### 6.1.3 Runtime Test

| **ID** | **Test case** | **Expected result** | **Result** | **Problem** |
|---|---|---|---|---|
| **1.** # type matching | a=1<br>b=2<br>c=a+b |  | #pass |  |
|  | a=1<br>b="1"<br>c=a+b |  |  | #Operation on incompatible types |
| **2.** # get list element | a=[1,2,3]<br>b=a[1] |  | #pass |  |
|  | a="a"<br>b=a[1] |  |  | #Get element on a non-list variable |
| **3.** # wet list element | a=[1,2,3]<br>a[1]=2 |  | #pass |  |
|  | a="a"<br>a[1]=2 |  |  | #Set element on a non-list variable |
| **4.** # equal type check | a=1<br>b=2<br>if a==b then<br>fi |  | #pass |  |

| | | | | |
|---|---|---|---|---|
| | a=5<br>b="c"<br>if a==b then<br>fi | | | #Operation on incompatible types |
| **5.** # equal type check | a=1<br>b=2<br>if a!=b then<br>fi | | #pass | |
| | a=6<br>b="t"<br>if a!=b then<br>fi | | | #Operation on incompatible types |

### 6.1.4 Application Test

| **ID** | **Test case** | **Expected result** | **Result** | **Problem** |
|---|---|---|---|---|
| **1.** # application sample 1 | Utterance = Utter(Input) # conversion<br><br>if Utterance eq \`<NN>I<VB>am<NN>*\`<br>Name = Utterance[2]<br>Reply = str(\`<UH>Hello<NN>World\` + Name)<br>print Reply + "\n"<br>fi | | #pass | |
| **2.** # application sample 2 | Utterance = \`<NN>I<VB>am<NN>Jerry\`<br>if Utterance eq \`<NN>I<VB>am<NN>*\`<br>Name = Utterance[2]<br>print \`<UH>Hello<NN>World\` + Name<br>fi | | #pass | |
| **3.** # application sample 3 | Input = ‘How much did the Dow Jones drop today?’<br><br>Parsed = parse (Input)<br><br>print (Parsed)<br><br># Parsed = \`(ROOT (SBARQ (WHNP (WRB How) (JJ much)) (SQ (VP (VBD did) (NP (DT the) (NNP Dow) (NNP Jones) (NN drop)) (NP (NN today)))) (. ?)))\`<br>if question == \`(ROOT (SBARQ (WHNP (WRB | | #pass<br>Print “”Dow Jones drop 30 points” | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | How) (JJ much)) (SQ (VP (VBD did) (NP * (NN drop)) (NP (NN *)))) (. ?)))` then answer = search_database($1, $2) fi<br><br>print answer |  |  |  |
| **4.** # application sample 4 dialog management | 'How much did the Dow Jones drop yesterday' (Dialog is input by person) |  | #pass “”Dow Jones drop 30 points” |  |
|  | 'How much is the Dow Jones today' (Dialog is input by person) |  | #pass “Dow Jones is 11559 points” |  |
|  | input = 'What is the Dow Jones today' (Dialog is input by person) |  | #pass “Dow Jones Industrial Average” |  |

## 6.2 Contributions

Out test plan contributes greatly to the reliability and stability of our system. We detected at least 25 bugs in our scanner, 12 bugs in our parser, 12 bugs about out Java AST and more than 30 bugs in our Java backend.

Since we do enough tests on semantic problems when using the ***Spoke***, we avoid semantic conflicts causing by semantic problems. We detect more than 10 bugs add about 10 types of semantic checking to avoid these bugs. We do carefully runtime checking. Our system based on compiler from Ocaml front end and Java backend. The interface between our front end and backend is a big problem when we developing our system. To handle this point, we press on runtime checking and greatly improve our java backend to solve all kinds of situations which might be caused by frontend input.

# 7 Lessons Learned

## 7.1 William Yang Wang

It really feels good that we finally manage to put all things together and have our own tiny language running within three months. I am also glad to learn my second functional programming language, OCAML, which certainly brings both of tears of pain and tears of joy to me. The learning curve of OCAML is flat, but worthy.

As the Project Manager, I believe three factors are essential to the success of a PLT project.

(1) ***People***. Choosing the right teammates is ****not**** half done. Understanding the *advantages of each team member* is the key to the success of this project. I am very fortunate to have Chia-che Tsai in charging of the front-end translator implementation details. As a system person, Tsai carried his high standards and requirements into this project. I am also fortunate to work with Xin Chen, who is a reliable and responsible teammate. He mainly works on the Java back-end and testing, and has made contribution to this project.

(2) ***Ideas and Design.*** In retrospect, I am glad we have a very clear blueprint of this project in the very beginning. The motivation of *spoke* language is well defined, and our goal of this project is clear: design and implement a light weight but powerful language that focuses on spoken dialog management. We also revise the detail features of our language to adapt the necessary changes in all different lifecycles of this project.

(3) ***Implementation and Execution.***
It is crucial to understand what can be done and what cannot be done within a limited time. We first implement the scanner and parser for the basic features, then, we start to translate the AST into Bytecode and Java AST. We also have two people working on the Java backend in parallel. After we have the basic features, then we immediately augment them with advanced features, for example, the joint parsing of natural language and programming language, and the natural language syntax tree matching.

### 7.2 Chia-che Tsai

The lessons I have learned in this project are two:

1. Why applying functional languages on system designs?
   During this project, I come to a deep understanding to the core of functional language. Functional language has a more strict definition of its functions, that each of them have to be make recursive. This actually makes the code less readable and easy to implement. However, when I began to master the languages, I found it provides a even stronger interpretion to algorithm. The reason why Functional Language is better on the implementation of algorithm is that it make developers thinks in recursive way, instead of the

traditional sequential way. A traditional language actually not only makes the code sequential but makes memory / object access sequential. The sequential access make the access vulnerable since it is hard to guarantee the memory contains data of consistent type. This problem is explicitly solved by functional languages.

2. The design of system from high-level to low-level, from front-end to back-end.
   I participated the whole implementation and design of this compiler from high level to low level, and from front-end to back-end. This makes an interesting fact: whenever we encounter some problem, we go back to the definition of languages and make the design a little be more concrete. Also we translate the language into Java source code. In order to fulfill this, we design our system, including intermediate code format, based on characteristic of Java.

## 7.3 Xin Chen

I feel excited that we have finally managed our programming language work out, and not only for a small Hello World application, but also for the more advanced feature of The Spoke Language. Also, this is my first time to learn a functional language— Ocaml. I didn't like Ocaml very much at the beginning, because I think a good programming language should be at least easy to read, and Ocaml always confused me, as a reader and a learner. Gradually, when I felt like that I had to learn it in order to do the project, and I began to know this language better. Right now, I have to acknowledge that Ocaml is really a great language in writing compiler.

Throughout the project, I think the most important thing that I have leant is that how to design a language. It is definitely not an easy question. From the very front end scanner to the backend Java source code, for each step we have to make a design decision, such as how many string types, how to deal with scoping and how to translate into Java code. I mainly worked on the middle layer, build JavaAST and translate the intermediate code into Java as well as backend implementation. I would like to thank Tsai, our CTO, who came up with the most design ideas and decisions, and I have learned a lot of things from him. Also, William is a very responsible project manager in the team, and he is mainly focusing on NLP part, if it was not him, we couldn't make the whole NLP dialog system work. Zhou has collaborated with me the backend implementation and testing, and I would thank him for his major contribution to the backend.

Additional credit: Before this project, I have no idea of how complex Natural Language Processing is. I would like to thank William who started this great project idea and shared the background NLP knowledge with me. I would take it as an additional credit because right now I am becoming professional in NLP.

### 7.4 Zhou Yu

The most valuable lesson I learned in PLT course is how to design and implement a compiler. Another lesson is that I learned Ocaml, which is a powerful functional language in implementing compiler.

(1) Ideas about how to design a compiler focusing on specific task.
Compiler is a complex system with several components. To design an efficient and effective compiler for specific task, we have to carefully design the component interfaces at the first place. Moreover, a good compiler is able to provide certain advanced feature to handle specific problem rather than trying to face all kinds of situations. In our system, it’s partial and wildcard matching between different NLP syntax trees.

(2) Detail about implementation of a compiler, specially the backend of a compiler.
After finishing this project, I have profound understanding about the implementation of a compiler. My work focuses on backend and java APIs. Because our backend depends on input of Ocaml frontend, it should be flexible enough to handle different situations and tolerate mistakes. We need to modify the design and code time by time to adapt to new changes so that we can realize a more flexible and stable backend

(3) knowledge and practice of Ocaml
Ocaml is succinct and strong functional language. It’s my first time using a functional language, so at first it’s a little difficult for me to understand Ocaml code. After finishing this project, I know not only how to use Ocaml but also how to think in functional language.

## Appendix:

### Front End:

### scanner.mll:

```
   { open Parser }

   let letter = ['a'-'z' 'A'-'Z']      (* english letters *)
   let digit  = ['0'-'9']              (* numerics *)
   let space  = [' '  '\t']            (* whitespace *)
   let break  = ['\r' '\n']            (* breaklines *)
   let punct  = ['.' ',' '?' '!' ';' ':']  (* punctuation *)

   rule token = parse
     space           { token lexbuf } (* split statement *)

   | '\\'space*break { token lexbuf } (* connect splitted statements *)
   | break       { [EOL] }

   | "#"         { comment lexbuf }   (* comments: start at pound and
end at EOL *)

   | '('         { [LPAREN] }           (* wrap expressions *)
   | ')'         { [RPAREN] }           (* EX: (1 + 2) * 3 *)
   | ','         { [COMMA] }

   | '+'         { [PLUS] }             (* arithmetic operators :
addition       (1 + 2) *)
   | '-'         { [MINUS] }            (*
substraction   (1 - 2) *)
   | '*'         { [TIMES] }            (*
mulitplication (1 * 2) *)
   | '/'         { [DIVIDE] }           (*
division       (1 / 2) *)
   | '%'         { [MODULUS] }          (*
modulus        (1 % 2) *)

   | '='         { [ASSIGN] }           (* arithmetic assignment : x =
y = 1 *)

   | "=="        { [EQ] }               (* comparation : equal
(a == b) *)
   | "!="        { [NEQ] }              (*             : not equal
(a != b) *)
   | '<'         { [LT] }               (*             : less than
(a <  b) *)
   | "<="        { [LEQ] }              (*             : less than or
equal    (a <= b) *)
   | ">"         { [GT] }               (*             : greater than
(a >  b) *)
   | ">="        { [GEQ] }              (*             : greater than
or equal (a >= b) *)
   | "~"         { [BELONG] }           (*             : Belongs to
(a ~  b) *)

   | "&&"        { [AND] }              (* boolean operators :
intersection  (a && b) *)
```

```
  | "||"        { [OR] }              (*                     : union
(a || b) *)
  | "!"         { [NOT] }             (*                     : negation
(!a)     *)

  | "["         { [LBRACK] }          (* array definition : [] [1,2]
*)
  | "]"         { [RBRACK] }          (*                      array[0]
array[0:1] *)
  | ":"         { [COLON] }

  | "global"    { [GLOBAL] }          (* local variable *)

  | "true"      { [BOOL(true)] }      (* boolean value true *)
  | "false"     { [BOOL(false)] }     (* boolean value false *)

  | "if"        { [IF] }              (* if-then-else: IF   a==0
THEN     *)
  | "then"      { [THEN] }            (*               ELIS a==-1
THEN     *)
  | "elif"      { [ELIF] }            (*               ELSE
*)
  | "else"      { [ELSE] }            (*               FI
*)
  | "fi"        { [FI] }
  | "for"       { [FOR] }             (* for-loop:   FOR a in [0, 1,
2]    *)
  | "in"        { [IN] }              (*             NEXT
*)
  | "next"      { [NEXT] }

  | "while"     { [WHILE] }           (* while-loop: WHILE a > 0
*)
  | "loop"      { [LOOP] }            (*                BREAK |
CONTINUE  *)
  | "break"     { [BREAK] }           (*             LOOP
*)
  | "continue"  { [CONTINUE] }

  | "func"      { [FUNCTION] }        (* function:   FUNCTION foo
*)
  | "return"    { [RETURN] }          (*                RETURN
*)
  | "end"       { [END] }             (*             END
*)

  | "null"      { [NULL] }

  | digit+ as lxm                   { [INTEGER(int_of_string lxm)] }
(* integer *)
  | digit+'.'digit+ as lxm          { [FLOAT(float_of_string lxm)] }
(* float   *)
```

```
   | (letter|'_')(letter|digit|'_')* as lxm { [ID(lxm)] }
(* ids     *)

   | '\''           { [STRING(nstring lexbuf)] } (* native strings:
'string' *)
   | '"'            { [STRING(estring lexbuf)] } (* escapable string:
"\t\r\n" *)
   | '`'            { [] @ tstring lexbuf }

   | eof         { [EOF] }

   | _ as char   { raise (Failure("illegal character " ^ Char.escaped
char)) }

   and nstring = parse
     '\''             { "" }
   | _ as char        { Char.escaped char ^ nstring lexbuf }

   and estring = parse
     '"'              { "" }
   | ['\t' '\r' '\n'] { " " ^ estring lexbuf }
   | '\\'_ as lxm     { lxm ^ estring lexbuf }
   | _ as char        { Char.escaped char ^ estring lexbuf }

   and tstring = parse
     '`'              { [] }
   | space            { tstring lexbuf }
   | punct as char    { [WORD(Char.escaped char)] @ tstring lexbuf }
   | letter+ as lxm   { [WORD(lxm)] @ tstring lexbuf }
   | '('              { [LTPAREN] @ tstring lexbuf }
   | ')'              { [RTPAREN] @ tstring lexbuf }
   | '*'              { [STAR] @ tstring lexbuf }

   and comment = parse
     break(space*break)*  { [EOL] }
   | _           { comment lexbuf }
```

**parser.mly:**

```
   ------------
   %{ open Ast;; %}

   %{ let unique e =
         let check_unique res e = if List.mem e res then res else
e::res
         in List.fold_left check_unique [] e;;

      let comple e l =
        let check_not_belong res e = if List.mem e l then res else
e::res
        in List.fold_left check_not_belong [] e;;
   %}

   %token LPAREN RPAREN
   %token COMMA
   %token LBRACK RBRACK
   %token COLON
   %token PLUS MINUS TIMES DIVIDE MODULUS
   %token ASSIGN
   %token EQ NEQ LT LEQ GT GEQ BELONG
   %token AND OR NOT
   %token IF THEN ELSE ELIF FI
   %token FOR IN NEXT
   %token WHILE LOOP
   %token CONTINUE BREAK
   %token FUNCTION END
   %token RETURN
   %token GLOBAL
   %token <bool> BOOL
   %token <int> INTEGER
   %token <float> FLOAT
   %token <string> ID
   %token <string> STRING
   %token LTPAREN RTPAREN
   %token <string> WORD
   %token NULL
   %token STAR
   %token EOL EOF

   %nonassoc NOELSE
   %nonassoc ELSE
   %nonassoc ELIF

   %right ASSIGN

   %left AND OR
   %left EQ NEQ LT GT LEQ GEQ
   %left PLUS MINUS
   %left TIMES DIVIDE MODULUS

   %start program
   %type <Ast.program> program

   %%

   program:
       program_rec EOF
       { (let g = unique (List.rev (List.concat (List.map (fun f ->
f.global) (snd $1))))
         and l = List.rev (unique (snd (fst $1)));
         in { gvar = g; lvar = comple l g; code = List.rev (fst (fst
$1)) }),
        List.rev (snd $1) }

   program_rec:
       /* nothing */    { ([], []), [] }
     | program_rec stmt { ((fst $2 :: fst (fst $1)), snd $2 @ snd (fst
$1)), snd $1 }
     | program_rec fdec { fst $1, ($2 :: snd $1) }

   eol:
       EOL {}

   stmt:
       stmt_unit eol { $1 }

   stmt_unit:
       /* empty */    { Noexpr, [] }
     | stmt_inline    { $1 }
     | IF expr THEN eol stmt_opt %prec NOELSE FI { If(fst $2, fst $5,
[]), snd $5 @ snd $2 }
     | IF expr THEN eol stmt_opt stmt_else FI    { If(fst $2, fst $5,
fst $6), snd $6 @ snd $5 @ snd $2 }
     | FOR ID IN expr eol stmt_opt NEXT     { For($2, fst $4, fst $6),
snd $6 @ snd $4 @ [$2] }
     | WHILE expr eol stmt_opt LOOP         { While(fst $2, fst $4),
snd $4 @ snd $2 }

   stmt_else:
       ELSE eol stmt_opt                          { $3 }
     | ELIF expr THEN eol stmt_opt %prec NOELSE  { [If(fst $2, fst $5,
[])], snd $5 @ snd $2 }
     | ELIF expr THEN eol stmt_opt stmt_else     { [If(fst $2, fst $5,
fst $6)], snd $6 @ snd $5 @ snd $2 }

   stmt_inline:
       expr_assign { Expr(fst $1), snd $1 }
     | expr_call   { Expr(fst $1), snd $1 }
     | RETURN      { Return(Null), [] }
     | RETURN expr { Return(fst $2), snd $2 }
     | CONTINUE    { Continue, [] }
     | BREAK       { Break, [] }
     | IF expr THEN stmt_inline %prec NOELSE     { If(fst $2, [fst $4],
[]), snd $4 @ snd $2 }
     | IF expr THEN stmt_inline ELSE stmt_inline { If(fst $2, [fst $4],
[fst $6]), snd $6 @ snd $4 @ snd $2 }

   stmt_opt:
       /* nothing */  { [], [] }
     | stmt_rec       { List.rev (fst $1), snd $1 }

   stmt_rec:
       stmt           { [fst $1], snd $1 }
     | stmt_rec stmt  { fst $2 :: fst $1, snd $2 @ snd $1 }

   expr:
       expr_assign      { $1 }
     | expr_com         { $1 }

   expr_assign:
       ID ASSIGN expr                            { Assign($1, fst $3),
 [$1] @ snd $3 }
     | var LBRACK expr RBRACK ASSIGN expr        { AssignElement(fst $1,
 fst $3, fst $6), snd $6 @ snd $3 @ snd $1 }

   expr_call:
       ID LPAREN arg_opt RPAREN    { Call($1, fst $3), snd $3 }

   arg_opt:
       /* nothing */    { [], [] }
     | arg_rec          { List.rev (fst $1), snd $1 }

   targ:
       WORD WORD      { Tag(String($1),  String($2)) }
     | WORD STAR      { Tag(String($1),  Any) }
     | STAR WORD      { Tag(Any, String($2))  }
     | STAR STAR      { Tag(Any, Any) }
     | WORD targ_list { Tag(String($1),  Newlist($2)) }
     | STAR targ_list { Tag(Any, Newlist($2)) }
     | targ_list      { Tag(Any, Newlist($1)) }

   targ_list:
       targ_rec { List.rev $1}

   targ_rec:
       LTPAREN targ RTPAREN           { [$2] }
     | targ_rec LTPAREN targ RTPAREN  { $3 :: $1 }

   arg_rec:
       expr               { [fst $1], snd $1 }
     | arg_rec COMMA expr { fst $3 :: fst $1, snd $3 @ snd $1 }

   expr_com:
       expr_prefix                { $1 }
     | expr_unit                  { $1 }
     | expr_com PLUS    expr_unit  { Binop(Add,   fst $1, fst $3), snd
 $3 @ snd $1 }
     | expr_com MINUS   expr_unit  { Binop(Sub,   fst $1, fst $3), snd
 $3 @ snd $1 }
     | expr_com TIMES   expr_unit  { Binop(Mult,  fst $1, fst $3), snd
 $3 @ snd $1 }
     | expr_com DIVIDE  expr_unit  { Binop(Div,   fst $1, fst $3), snd
 $3 @ snd $1 }
```

```
     | expr_com MODULUS expr_unit  { Binop(Mod,   fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com EQ      expr_unit  { Binop(Equal, fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com NEQ     expr_unit  { Binop(Neq,   fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com LT      expr_unit  { Binop(Less,  fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com LEQ     expr_unit  { Binop(Leq,   fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com GT      expr_unit  { Binop(Greater, fst $1, fst $3),
snd $3 @ snd $1 }
     | expr_com GEQ     expr_unit  { Binop(Geq,   fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com AND     expr_unit  { Binop(And,   fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com OR      expr_unit  { Binop(Or,    fst $1, fst $3), snd
$3 @ snd $1 }
     | expr_com BELONG  expr_unit  { Binop(Has,   fst $1, fst $3), snd
$3 @ snd $1 }

   expr_prefix:
       MINUS expr_unit   { Unaop(Minus, fst $2), snd $2 }
     | NOT   expr_unit   { Unaop(Not,   fst $2), snd $2 }

   expr_unit:
       BOOL                             { Bool($1), [] }
     | INTEGER                          { Int($1), [] }
     | FLOAT                            { Float($1), [] }
     | STRING                           { String($1), [] }
     | STAR                             { Any, [] }
     | NULL                             { Null, [] }
     | var                              { $1 }
     | targ                           { $1,[] }
     | expr_call                        { $1 }
     | LPAREN expr RPAREN               { $2 }
     | LBRACK arg_opt RBRACK            { Newlist(fst $2), snd $2 }

   var:
       ID                               { Var($1), [] }
     | var LBRACK expr RBRACK           { Element(fst $1, fst $3), snd
$3 @ snd $1 }
     | var LBRACK expr COLON expr RBRACK { Sublist(fst $1, fst $3, fst
$5), snd $5 @ snd $3 @ snd $1 }

   fdec:
       FUNCTION ID LPAREN id_opt RPAREN eol gdec stmt_opt END eol
       { let g = List.rev (unique $7) and l = List.rev (unique (snd $8))
         in { name = $2; args = $4; global = g; local = comple (comple
l g) $4;
             body = fst $8 } }
```

```
gdec:
    /* nothing */  { [] }
  | GLOBAL id_list eol { $2 }

id_opt:
    /* nothing */     { [] }
  | id_rec            { List.rev $1 }

id_list:
    id_rec            { List.rev $1 }

id_rec:
    ID                { [$1] }
  | id_rec COMMA ID   { $3 :: $1 }
```

**ast.ml:**

```
------------
type binop = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq
| Greater |
             Geq | And | Or | Has
type unaop = Minus | Not

type debug = {
    indent : string;
  }

type expr =
    Bool of bool | Int of int | Float of float
  | String of string | Var of string
  | Newlist of expr list
  | Call of string * expr list
  | Binop of binop * expr * expr
  | Tag of expr * expr
  | Unaop of unaop * expr
  | Assign of string * expr
  | AssignElement of expr * expr * expr
  | Element of expr * expr
  | Sublist of expr * expr * expr
  | Any
  | Null

type stmt =
    Expr of expr
  | Return of expr
  | Continue
  | Break
  | If of expr * stmt list * stmt list
  | For of string * expr * stmt list
```

```
     | While of expr * stmt list
     | Noexpr

   type main = {
       gvar : string list;
       lvar : string list;
       code : stmt list;
     }

   type func = {
       name : string;
       args : string list;
       global : string list;
       local : string list;
       body : stmt list;
     }

   type program = main * func list

   let rec string_of_expr = function
       Bool b   -> "Bool(" ^ string_of_bool b ^ ")"
     | Int i    -> "Int(" ^ string_of_int i ^ ")"
     | Float f  -> "Float(" ^ string_of_float f ^ ")"
     | Var s    -> "Var(" ^ s ^ ")"
     | Newlist el  -> "List(" ^ String.concat " " (List.map
string_of_expr el) ^ ")"
     | Call (s, el) ->
         "Call(" ^ s ^ " " ^ String.concat " " (List.map string_of_expr
el) ^ ")"
     | Element (a, i) ->
         "Element(" ^ string_of_expr a ^ " " ^ string_of_expr i ^ ")"
     | String s -> "String(\"" ^ s ^ "\")"
     | Sublist (l, e1, e2) ->
         "Sublist(" ^ string_of_expr l ^ " " ^ string_of_expr e1 ^ " "
^ string_of_expr e2 ^ ")"
     | Tag (e1, e2) ->
         "Tag(" ^ string_of_expr e1 ^ " " ^ string_of_expr e2 ^ ")"
     | Any -> "Any"
     | Null -> "Null"
     | Binop (o, e1, e2) ->
         (match o with
         Add -> "Add" | Sub -> "Sub" | Mult -> "Mult" | Div -> "Div" |
         Mod -> "Mod" | Equal -> "Equal" | Neq -> "Neq" | Less ->
"Less" |
         Leq -> "Leq" | Greater -> "Greater" | Geq -> "Geq" | And ->
"And" |
         Or -> "Or" | Has -> "Has") ^ "(" ^ string_of_expr e1 ^ " " ^
string_of_expr e2 ^ ")"
     | Unaop (o, e) ->
         (match o with Minus -> "Minus" | Not -> "Not") ^
         "(" ^ string_of_expr e ^ ")"
     | Assign (s, e) ->
```

```
        "Assign(" ^ s ^ " " ^ string_of_expr e ^ ")"
    | AssignElement (e1, e2, e3) ->
        "AssignElement(" ^ string_of_expr e1 ^ " " ^ string_of_expr e2
^ " " ^
        string_of_expr e3 ^ ")"

  let rec string_of_stmt debug = function
      Expr expr ->
        debug.indent ^ "Expr(" ^ string_of_expr expr ^ ")\n";
    | Return expr ->
        debug.indent ^ "Return(" ^ string_of_expr expr ^ ")\n";
    | If (e, s1, s2) ->
        debug.indent ^ "If(" ^ string_of_expr e ^ ",\n" ^
        (String.concat "" (List.map (string_of_stmt { indent =
debug.indent ^ "  " }) s1)) ^
        debug.indent ^ ",\n" ^
        (String.concat "" (List.map (string_of_stmt { indent =
debug.indent ^ "  " }) s2)) ^
        debug.indent ^ ")\n"
    | For (v, e, s) ->
        debug.indent ^ "For(" ^ v  ^ " " ^ string_of_expr e ^ "\n" ^
        (String.concat "" (List.map (string_of_stmt { indent =
debug.indent ^ "  " }) s)) ^
        debug.indent ^ ")\n"
    | While (e, s) ->
        debug.indent ^ "While(" ^ string_of_expr e ^ "\n" ^
        (String.concat "" (List.map (string_of_stmt { indent =
debug.indent ^ "  " }) s)) ^
        debug.indent ^ ")\n"
    | Break    -> debug.indent ^ "Break" ^ "\n"
    | Continue -> debug.indent ^ "Continue" ^ "\n"
    | Noexpr   -> ""

  let string_of_func f =
      f.name ^ ": args=[" ^ (String.concat " " f.args) ^ "]" ^
      " global=[" ^ (String.concat " " f.global) ^ "]" ^
      " local=[" ^ (String.concat " " f.local) ^ "]\n" ^
      (String.concat "" (List.map (string_of_stmt { indent = "  " })
f.body))

  let rec string_of_program (main, funcs) =
      (String.concat "\n" (List.map string_of_func funcs)) ^ "\n" ^
      "__global__: " ^ (String.concat " " main.gvar) ^ "\n" ^
      "__local__: " ^ (String.concat " " main.lvar) ^ "\n" ^
      "__main__:\n" ^
      (String.concat "" (List.map (string_of_stmt { indent = "  " })
main.code))
```

**checker.ml:**

```
------------
open Ast
open Backend

let rec check_stmt iter stmt =
  iter stmt; (match stmt with
     If (_, s1, s2) -> List.iter (check_stmt iter) (s1 @ s2)
   | For (_, _, s)  -> List.iter (check_stmt iter) s
   | While (_, s)   -> List.iter (check_stmt iter) s
   | _ -> ())

let rec check_stmt_flag iter flag stmt =
    let flag = iter flag stmt in (match stmt with
     If (_, s1, s2) -> List.iter (check_stmt_flag iter flag) (s1 @
s2)
   | For (_, _, s)  -> List.iter (check_stmt_flag iter flag) s
   | While (_, s)   -> List.iter (check_stmt_flag iter flag) s
   | _ -> ())

let rec check_stmt_fold iter flags stmt =
    [iter (match stmt with
     If (_, s1, s2) -> List.concat (List.map (List.fold_left
(check_stmt_fold iter) flags) [s1; s2])
   | For (_, _, s)  -> List.fold_left (check_stmt_fold iter) flags s
   | While (_, s)   -> List.fold_left (check_stmt_fold iter) flags s
   | _ -> flags) stmt]

let rec check_expr iter expr = iter expr;
  (match expr with
     Call (s, el) -> List.iter (check_expr iter) el
   | Element (a, i) -> List.iter (check_expr iter) [a; i]
   | Sublist (l, e1, e2) -> List.iter (check_expr iter) [l; e1; e2]
   | Tag (s, e) -> check_expr iter e
   | Binop (o, e1, e2) -> List.iter (check_expr iter) [e1; e2]
   | Unaop (o, e) -> check_expr iter e
   | Assign (s, e) -> check_expr iter e
   | AssignElement (e1, e2, e3) -> List.iter (check_expr iter) [e1;
e2; e3]
   | _ -> ())

let rec check_expr_in_stmt iter stmt =
  (match stmt with
     Expr e -> check_expr iter e
   | Return e -> check_expr iter e
   | If (e, s1, s2) -> check_expr iter e;
                       List.iter (check_expr_in_stmt iter) (s1 @ s2)
   | For (v, e, s) -> check_expr iter e; List.iter
(check_expr_in_stmt iter) s
   | While (e, s) -> check_expr iter e; List.iter
(check_expr_in_stmt iter) s
   | _ -> ())
   let iter_body iter funcs = List.iter (fun f -> iter f.body) funcs

   let check_syntax (main, funcs) =

     let result = (main, funcs)

     in let result =
       let check_var_has_builtin (main, funcs) target =
         let check_has_arg al = List.mem target al in
         let check_func_has_arg func =
           if check_has_arg func.args then
             raise (Failure ("\'" ^ target ^ "\' used in argument of
function " ^ func.name))
           else if check_has_arg func.args then
             raise (Failure ("\'" ^ target ^ "\' declared as global in
function " ^ func.name))
           else ()
         in List.iter check_func_has_arg funcs
       in List.iter (check_var_has_builtin result) (built_in_globals @
built_in_locals); result

     in let result =
       let check_has_assign_to_builtin (main, funcs) target =
         let check_is_assign_to_arg = function
           Assign (var, _) when var == target
             -> raise (Failure ("\'" ^ var ^ "\' is assigned")) | _ ->
()
         in List.iter (check_expr_in_stmt check_is_assign_to_arg)
main.code;
         iter_body (List.iter (check_expr_in_stmt
check_is_assign_to_arg)) funcs
       in List.iter (check_has_assign_to_builtin result)
(built_in_globals @ built_in_locals); result

     in let result =
       let check_return_in_main (main, funcs) =
         let check_is_return = function
           Return _ -> raise (Failure "\'return\' in main function") |
_ -> ()
         in List.iter (check_stmt check_is_return) main.code
       in check_return_in_main result; result

     in let result =
       let check_continue_break_in_loop (main, funcs) =
         let check_is_loop_and_set_flag flag = function
             For (_, _, _) -> false | While (_, _) -> false
           | Continue ->
               if flag then raise (Failure "\'continue\' outside loops")
else false
           | Break ->
               if flag then raise (Failure "\'continue\' outside loops")
else false
```

```
          | _ -> flag
        in List.iter (check_stmt_flag check_is_loop_and_set_flag true)
main.code;
           iter_body (List.iter (check_stmt_flag
check_is_loop_and_set_flag true)) funcs
      in check_continue_break_in_loop result; result

    in let result =
      let check_return_in_function (main, funcs) =
        let check_function_is_returned func =
          let check_is_returned flags = function
              Return (_) -> true
            | If (_, _, _) -> (List.nth flags 0) && (List.nth flags 1)
            | _ -> List.hd flags
          in List.hd (List.fold_left (check_stmt_fold
check_is_returned) [false] func.body)
        in main, List.map (fun f -> if check_function_is_returned f
then f
                            else { f with body = f.body @
[Return(Null)] }) funcs
      in check_return_in_function result

    in let result =
      let check_has_stmt_after_return (main, funcs) =
        let check_function_has_stmt_after_return func =
          let check_is_returned flags = function
              Return (_) -> true
            | If (_, _, _) -> (List.nth flags 0) && (List.nth flags 1)
            | For (_, _, _) -> false
            | While (_, _) -> false
            | _ -> if List.hd flags then raise (Failure "statement
after return") else false
          in List.hd (List.fold_left (check_stmt_fold
check_is_returned) [false] func.body)
        in List.iter (fun f -> ignore
(check_function_has_stmt_after_return f)) funcs
      in check_has_stmt_after_return result; result

    in result
```

## **compile.ml:**

```
------------
open Ast
open Intercode
open Backend

module StringMap = Map.Make(String)
```

```
   type env = {
       (* Function Reference *)
       fun_ref : int StringMap.t;
       (* Local Variable Reference *)
       loc_ref : int StringMap.t;
       (* Global Variable Reference *)
       glb_ref : int StringMap.t;
       (* Temporary Variable List *)
       tmp_lst : int list;
     }

   let available l =
     let rec find_available_index l e =
       if List.mem e l then find_available_index l (e+1) else e
     in find_available_index l 0

   let rec remove s = function
       [] -> [] | e :: el -> if e == s then remove e el else e ::
remove e el

   let unique e =
       let check_unique res e = if List.mem e res then res else res @
[e]
       in List.fold_left check_unique [] e

   let comple e l =
       let check_not_belong res e = if List.mem e l then res else res @
[e]
       in List.fold_left check_not_belong [] e

   let rec enum stride n = function
       [] -> []
     | hd::tl -> (n, hd) :: enum stride (n+stride) tl

   let string_map_pairs map pairs =
     List.fold_left (fun m (i, n) -> StringMap.add n i m) map pairs

   let intersect_map map el =
     let check_belong map new_map e =
       if StringMap.mem e map then StringMap.add e (StringMap.find e
map) new_map
                              else new_map
     in List.fold_left (check_belong map) StringMap.empty el

   let translate (main, funcs) =

     let rec intercode env stmts =

       let rec expr (code, env) = function
           Bool(b)   -> let t = available env.tmp_lst
                        in code @ [Cb(t,b)], { env with tmp_lst = t ::
env.tmp_lst }
```

```
        | Int(i)    -> let t = available env.tmp_lst
                       in code @ [Ci(t,i)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Float(f)  -> let t = available env.tmp_lst
                       in code @ [Cf(t,f)], { env with tmp_lst = t :: 
env.tmp_lst }
        | String(s) -> let t = available env.tmp_lst
                       in code @ [Cs(t,s)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Newlist(el) ->
            let t = available env.tmp_lst in
            let add_expr e =
               let s = expr ([],{ env with tmp_lst = t :: 
env.tmp_lst }) e in
               let i = List.hd (snd s).tmp_lst
               and t' = available (snd s).tmp_lst
               in fst s @ [Wp (t',i); Bn (t,Add,t,t')]
            in code @ [Wp (t,-1)] @ List.concat (List.map add_expr el),
               { env with tmp_lst = t :: env.tmp_lst }
        | Sublist(e, e1, e2) ->
            let s = expr (expr (expr (code, env) e) e1) e2 in
            let i1 = List.nth (snd s).tmp_lst 2
            and i2 = List.nth (snd s).tmp_lst 1
            and i3 = List.nth (snd s).tmp_lst 0
            and t = available (snd s).tmp_lst
            in fst s @ [Pt(t,i1,i2,i3)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Element(e, e1) ->
            let s = expr (expr (code, env) e) e1 in
            let i1 = List.nth (snd s).tmp_lst 1
            and i2 = List.nth (snd s).tmp_lst 0
            and t = available (snd s).tmp_lst
            in fst s @ [El(t,i1,i2)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Binop(And, e1, e2) ->
            let s1 = expr (code,  env) e1 in
            let s2 = expr ([], snd s1) e2 in
            let i1 = List.hd (snd s1).tmp_lst
            and i2 = List.hd (snd s2).tmp_lst
            and t = available (snd s2).tmp_lst
            in fst s1 @ [Br(i1,fst s2 @ 
[Br(i2,[Cb(t,true)],[Cb(t,false)])],[Cb(t,false)])],
                                        { env with tmp_lst = t :: 
env.tmp_lst }
        | Binop(Or, e1, e2) ->
            let s1 = expr (code,  env) e1 in
            let s2 = expr ([], snd s1) e2 in
            let i1 = List.hd (snd s1).tmp_lst
            and i2 = List.hd (snd s2).tmp_lst
            and t = available (snd s2).tmp_lst
            in fst s1 @ [Br(i1,[Cb(t,true)],fst s2 @ 
[Br(i2,[Cb(t,true)],[Cb(t,false)])])],
                                    { env with tmp_lst = t :: 
env.tmp_lst }
        | Binop(o, e1, e2) ->
            let s = expr (expr (code, env) e1) e2 in
            let i1 = List.nth (snd s).tmp_lst 1
            and i2 = List.nth (snd s).tmp_lst 0
            and t = available (snd s).tmp_lst
            in fst s @ [Bn(t,o,i1,i2)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Unaop(Not, e) ->
            let s = expr (code, env) e in
            let i = List.hd (snd s).tmp_lst
            and t = available (snd s).tmp_lst
            in fst s @ [Br(i,[Cb(t,false)],[Cb(t,true)])],
                                    { env with tmp_lst = t :: 
env.tmp_lst }
        | Unaop(o, e) ->
            let s = expr (code, env) e in
            let i = List.hd (snd s).tmp_lst
            and t = available (snd s).tmp_lst
            in fst s @ [Un(t,o,i)], { env with tmp_lst = t :: 
env.tmp_lst }
        | Assign(v, e) ->
            let s = expr (code, env) e in
            let i = List.hd (snd s).tmp_lst
            in fst s @
              (try [Sg(i,StringMap.find v env.glb_ref)] with Not_found 
->
               try [Sl(i,StringMap.find v env.loc_ref)] with Not_found 
->
               raise (Failure ("undeclared variable: " ^ v))),
               { env with tmp_lst = i :: env.tmp_lst }
        | AssignElement(e, e1, e2) ->
            let s = expr (expr (expr (code, env) e) e1) e2 in
            let i1 = List.nth (snd s).tmp_lst 2
            and i2 = List.nth (snd s).tmp_lst 1
            and i3 = List.nth (snd s).tmp_lst 0
            in fst s @ [St(i1,i2,i3)], { env with tmp_lst = i3 :: 
env.tmp_lst }
        | Var(v) ->
            let i = available env.tmp_lst
            in code @
               (try [Lg(i,StringMap.find v env.glb_ref)] with 
Not_found ->
                try [Ll(i,StringMap.find v env.loc_ref)] with 
Not_found ->
                raise (Failure ("undeclared variable: " ^ v))),
                { env with tmp_lst = i :: env.tmp_lst }
        | Call (f, el) ->
            let s = expr (code, env) (Newlist el) in
            let i = List.hd (snd s).tmp_lst
            and t = available (snd s).tmp_lst
            in fst s @
              (try [Cl (t, (StringMap.find f env.fun_ref), i)] with
Not_found ->
                raise (Failure ("undefined function: " ^ f))),
                { env with tmp_lst = t :: env.tmp_lst }
        | Tag (e1, e2) ->
            let s = expr (expr (code, env) e1) e2 in
            let i1 = List.nth (snd s).tmp_lst 1
            and i2 = List.nth (snd s).tmp_lst 0
            and t = available ((snd s).tmp_lst)
            in fst s @ [Tg (t, i1, i2)], { env with tmp_lst = t ::
env.tmp_lst }
        | Null ->
            let i = available env.tmp_lst
            in code @ [Nl i], { env with tmp_lst = i :: env.tmp_lst }
        | Any ->
            let i = available env.tmp_lst
            in code @ [Ay i], { env with tmp_lst = i :: env.tmp_lst }
      in

      let rec stmt (_, env) = function
          Noexpr -> [], env
        | Expr(e) -> let s = expr ([], env) e in fst s, env
        | If(e, s1, s2) ->
            let p = expr ([], env) e in
            let i = List.hd (snd p).tmp_lst in
            let p1 = List.concat(List.map fst (List.map (stmt ([],
env)) s1))
            and p2 = List.concat(List.map fst (List.map (stmt ([],
env)) s2))
            in fst p @ [Br (i, p1, p2)], env
        | For(v, e, s) ->
            let p = expr ([], env) e in
            let i = List.hd (snd p).tmp_lst
            and t1 = available (snd p).tmp_lst in
            let t2 = available ([t1] @ (snd p).tmp_lst) in
            let t' = available ([t2; t1] @ (snd p).tmp_lst) in
            let p' = List.concat(List.map fst (List.map (stmt ([],
                    { env with tmp_lst = ([t2; t1; i] @ (snd
p).tmp_lst) })) s))
            in fst p @
               [Wp (t',i); Cl (t1, (StringMap.find "len" env.fun_ref),
t'); Ci (t2,-1);
                Lp ([Ci (t',1); Bn(t2,Add,t2,t'); Bn (t',Leq,t1,t2);
                     Br (t',[Bk],[]); El (t',i,t2)] @
                    (try [Sg(t',StringMap.find v env.glb_ref)] with
Not_found ->
                     try [Sl(t',StringMap.find v env.loc_ref)] with
Not_found ->
                     raise (Failure ("undeclared variable: " ^ v))) @
p')], env
        | While(e, s) ->
```

```
             let p = expr ([], env) e in
             let i = List.hd (snd p).tmp_lst
             and p' = List.concat(List.map fst (List.map (stmt ([],
env)) s))
             in [Lp (fst p @ [Br (i, [], [Bk])] @ p')], env
         | Return(Null) -> [Rt (-1)], env
         | Return(e) ->
             let p = expr ([], env) e in
             let i = List.hd (snd p).tmp_lst
             in fst p @ [Rt i], env
         | Break -> [Bk], env
         | Continue -> [Ct], env

       in List.concat (List.map fst (List.map (stmt ([], env)) stmts))
in


     let glb_var = built_in_globals @ main.gvar in
     let glb_ref = string_map_pairs StringMap.empty (enum 1 0 glb_var)
in

     let loc_var = unique (built_in_locals @ main.lvar) in
     let loc_ref = string_map_pairs StringMap.empty (enum 1 0 loc_var)
in

     let plugin_fun_ref = string_map_pairs StringMap.empty
built_in_functions
     and fun_nam = List.map (fun f -> f.name) funcs in
     let fun_ref = string_map_pairs plugin_fun_ref (enum 1 0 fun_nam)
in
     let main_code = [Lc (List.length loc_var)] @
                     (intercode { fun_ref = fun_ref; loc_ref = loc_ref;
                                  glb_ref = glb_ref; tmp_lst = [] }
main.code) in

     let func_code =
       let translate_func func =
         let glb_ref = intersect_map glb_ref (built_in_globals @
func.global) in

         let loc_var = unique (built_in_locals @ func.args @ func.local)
in
         let loc_ref = string_map_pairs StringMap.empty (enum 1 0
loc_var) in

         let arg_ref = string_map_pairs StringMap.empty (enum 1 0
func.args) in
         let arg_lod = [Ll(0,0)] @ List.concat (List.map
                       (fun s -> [Ci (1,StringMap.find s arg_ref);
El(2,0,1);
                                  Sl (2,StringMap.find s loc_ref)])
func.args) in
```

```
      [Fn([Lc (List.length loc_var)] @ arg_lod @
          intercode { fun_ref = fun_ref; loc_ref = loc_ref;
                      glb_ref = glb_ref; tmp_lst = [] } func.body)]

    in List.concat (List.map translate_func funcs)

  in [Gl (List.length glb_var)] @ func_code @ main_code
```

**intercode.ml:**

```
------------
open Ast

type intercode =
    Gl of int                    (* Global variables *)
  | Lc of int                    (* Local variables *)
  | Cb of int * bool             (* Constant bool *)
  | Ci of int * int              (* Constant integer *)
  | Cf of int * float            (* Constant float *)
  | Cs of int * string           (* Constant string *)
  | Ay of int                    (* Any Object *)
  | Nl of int                    (* Null Object *)
  | Wp of int * int              (* Wrap list *)
  | Pt of int * int * int * int (* Get partition *)
  | El of int * int * int        (* Get element *)
  | St of int * int * int        (* Set element *)
  | Ll of int * int              (* Load local variable *)
  | Sl of int * int              (* Store local variable *)
  | Lg of int * int              (* Load global variable *)
  | Sg of int * int              (* Store global variable *)
  | Bn of int * Ast.binop * int * int   (* Binary operation *)
  | Un of int * Ast.unaop * int         (* Unary operation *)
  | Fn of intercode list         (* Function *)
  | Rt of int                    (* Return *)
  | Cl of int * int * int        (* Call Function *)
  | Tg of int * int * int        (* Tag Object *)
  | Br of int * intercode list * intercode list    (* Branch *)
  | Lp of intercode list         (* Loop *)
  | Bk                           (* Break *)
  | Ct                           (* Continue *)

let line_of_intercode codes =
  let add_line = fun f a b -> a + (f b) in
  let rec line_of_code = function
    Br(_, i1, i2) -> (List.fold_left (add_line line_of_code) 0 i1) +
                     (List.fold_left (add_line line_of_code) 0 i2)
  | Lp(i) -> List.fold_left (add_line line_of_code) 0 i
```

```
      | Fn(i) -> List.fold_left (add_line line_of_code) 0 i
      | _ -> 1
      in List.fold_left (add_line line_of_code) 0 codes



   let rec string_of_intercode = function
        Gl(i) -> ["Gl " ^ string_of_int i]
      | Lc(i) -> ["Lc " ^ string_of_int i]
      | Cb(i, true)  -> ["Cb " ^ string_of_int i ^ " 1"]
      | Cb(i, false) -> ["Cb " ^ string_of_int i ^ " 0"]
      | Ci(i, i1) -> ["Ci " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Cf(i, i1) -> ["Cf " ^ string_of_int i ^ " " ^ string_of_float i1]
      | Cs(i, i1) -> ["Cs " ^ string_of_int i ^ " \"" ^ i1 ^ "\""]
      | Ay(i) -> ["Ay " ^ string_of_int i]
      | Nl(i) -> ["Nl " ^ string_of_int i]
      | Wp(i,i1) -> ["Wp " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Pt(i, i1, i2, i3) -> ["Pt " ^ string_of_int i ^ " " ^
string_of_int i1 ^
                             " " ^ string_of_int i2 ^ " " ^
string_of_int i3]
      | El(i, i1, i2) -> ["El " ^ string_of_int i ^ " " ^ string_of_int
i1 ^
                         " " ^ string_of_int i2]
      | St(i, i1, i2) -> ["St " ^ string_of_int i ^ " " ^ string_of_int
i1 ^
                         " " ^ string_of_int i2]
      | Ll(i, i1) -> ["Ll " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Sl(i, i1) -> ["Sl " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Lg(i, i1) -> ["Lg " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Sg(i, i1) -> ["Sg " ^ string_of_int i ^ " " ^ string_of_int i1]
      | Bn(i, Ast.Add,     i1, i2) -> ["Ad " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Sub,     i1, i2) -> ["Sb " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Mult,    i1, i2) -> ["Ml " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Div,     i1, i2) -> ["Dv " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Mod,     i1, i2) -> ["Md " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Equal,   i1, i2) -> ["Eq " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Neq,     i1, i2) -> ["Ne " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Less,    i1, i2) -> ["Ls " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Leq,     i1, i2) -> ["Le " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
      | Bn(i, Ast.Greater, i1, i2) -> ["Gt " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
```

```
     | Bn(i, Ast.Geq,     i1, i2) -> ["Ge " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
     | Bn(i, Ast.And,     i1, i2) -> ["An " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
     | Bn(i, Ast.Or,      i1, i2) -> ["Or " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
     | Bn(i, Ast.Has,     i1, i2) -> ["Hs " ^ string_of_int i ^ " " ^
string_of_int i1 ^ " " ^ string_of_int i2]
     | Un(i, Ast.Minus,   i1) -> ["Mn " ^ string_of_int i ^ " " ^
string_of_int i1]
     | Un(i, Ast.Not,     i1) -> ["Nt " ^ string_of_int i ^ " " ^
string_of_int i1]
     | Rt(i)            -> ["Rt " ^ string_of_int i]
     | Cl(i, i1, i2)    -> ["Cl " ^ string_of_int i ^ " " ^
string_of_int i1 ^
                            " " ^ string_of_int i2]
     | Tg(i, i1, i2)    -> ["Tg " ^ string_of_int i ^ " " ^
string_of_int i1 ^
                            " " ^ string_of_int i2]
     | Br(i, i1, i2)    -> ["Br " ^ string_of_int i ^ " " ^
                            string_of_int (line_of_intercode i1) ^ " "
^
                            string_of_int (line_of_intercode i2)] @
                            List.concat (List.map string_of_intercode
i1) @
                            List.concat (List.map string_of_intercode
i2)
     | Lp(i) -> ["Lp " ^ string_of_int (line_of_intercode i)] @
                List.concat (List.map string_of_intercode i)
     | Fn(i) -> ["Fn " ^ string_of_int (line_of_intercode i)] @
                List.concat (List.map string_of_intercode i)
     | Bk    -> ["Bk"]
     | Ct    -> ["Ct"]

   let string_of_prog code =
     let translate = List.concat (List.map string_of_intercode code)
     in String.concat "\n" translate ^ "\n"
```

**javaast.ml:**

```
   ------------
   open Ast

   type jid = string

   type jvalue = string

   type jtype = string
```

```
type jname = string

type jaccess = Public | Private | Protected | None

type jarg = {
        a_type : jtype;
        a_name : jid;
}

type jstmt =
    Jdef of jtype * jid                        (* Type Declaration *)
  | Jnew of jtype * jvalue list * jid          (* Type Defination *)
  | Jinvoke of jid * jvalue list * jid         (* FunctionCall *)
  | Jthrow of jid * jvalue list                (* ExceptionThrow *)
  | Jinstanceof of jid * jid * jid             (* InstanceOf *)
  | Jif of jvalue * jstmt list * jstmt list (* Branch *)
  | Jfor of jid * jvalue * jstmt list          (* For *)
  | Jwhile of jvalue * jstmt list              (* Loop *)
  | Jbreak                                     (* Break *)
  | Jcontinue                                  (* Continue *)
  | Jreturn of jid                             (* Return *)

type jmethod = {
        m_name : jid;
        m_access : jaccess;
    m_static : bool;
        m_arg : jarg list;
        m_return : jtype;
        m_body : jstmt list;
}

type jfield ={
    f_access : jaccess;
    f_static : bool;
    f_type : jtype;
    f_name : jid;
}

type jclass = {
        c_package : jid;
        c_access : jaccess;
        c_name : jid;
    c_parent : jid;
        c_fields : jfield list;
        c_methods : jmethod list;
}

type debug = {
    indent : string;
}

let string_of_arg ag = ag.a_name ^ " As " ^ ag.a_type
```

```
   let rec string_of_stmt dbg = function
       Jdef(t, i) ->
         dbg.indent ^ String.concat " " (["Def"; i; "As"; t])
     | Jnew(t, al, i) ->
         dbg.indent ^ String.concat " " (["New"; i; "As"; t; "With"] @
al)
     | Jinvoke(i, al, "") ->
         dbg.indent ^ String.concat " " (["Invoke"; i; "With"] @ al)
     | Jinvoke(i, al, t) ->
         dbg.indent ^ String.concat " " (["Invode"; i; "With"] @ al @
["To"; t])
     | Jthrow(t, al) ->
         dbg.indent ^ String.concat " " (["Throw"; t; "With"] @ al)
     | Jinstanceof(a, t, i) ->
         dbg.indent ^ String.concat " " (["Check"; a; "Instance Of"; t;
"To"; t])
     | Jif(i, sl, []) ->
         dbg.indent ^ String.concat ("\n" ^ dbg.indent)
         (["If " ^ i] @ List.map (string_of_stmt { indent = "  " }) sl)
     | Jif(i, sl1, sl2) ->
         dbg.indent ^ String.concat ("\n" ^ dbg.indent)
         (["If " ^ i] @ List.map (string_of_stmt { indent = "  " }) sl1
@
          ["Else"]    @ List.map (string_of_stmt { indent = "  " }) sl2)
     | Jfor(i, a, sl) ->
         dbg.indent ^ String.concat ("\n" ^ dbg.indent)
         (["For " ^ i ^ " In " ^ a] @ List.map (string_of_stmt { indent
= "  " }) sl)
     | Jwhile(a, sl) ->
         dbg.indent ^ String.concat ("\n" ^ dbg.indent)
         (["While " ^ a] @ List.map (string_of_stmt { indent = "  " })
sl)
     | Jbreak ->
         dbg.indent ^ "Break"
     | Jcontinue ->
         dbg.indent ^ "Continue"
     | Jreturn("") ->
         dbg.indent ^ "Return"
     | Jreturn(i) ->
         dbg.indent ^ "Return " ^ i

   let string_of_access = function
       Public -> "Public"
     | Private -> "Private"
     | Protected -> "Protected"
     | None -> "None"

   let string_of_field dbg fd =
     dbg.indent ^ String.concat " "
     ((if fd.f_static then ["Static"] else []) @ ["Field"; fd.f_name;
"As"; fd.f_type;
```

```
                                     "Of"; string_of_access
fd.f_access])

   let string_of_method dbg mt =
     dbg.indent ^ String.concat ("\n" ^ dbg.indent)
     ([String.concat " "
     ((if mt.m_static then ["Static"] else []) @
      ["Method"; mt.m_name; "As"; mt.m_return; "Of"; string_of_access
mt.m_access;
       "With"] @ List.map string_of_arg mt.m_arg)] @
      List.map (string_of_stmt { indent = "  " }) mt.m_body)

   let string_of_class cl =
     (String.concat "\n"
     ((if String.length cl.c_package > 0 then ["Package " ^
cl.c_package] else []) @
      [String.concat " " (["Class"; cl.c_name] @
       (if String.length cl.c_parent > 0 then ["Extend"; cl.c_parent]
else []) @
       ["Of"; string_of_access cl.c_access])] @
      List.map (string_of_field  { indent = "  " }) cl.c_fields @
      List.map (string_of_method { indent = "  " }) cl.c_methods)) ^
"\n"
```

**javasrc.ml:**

```
   ------------
   open Javaast

   type built_in_type = B | I | F | S | L | O | T | N | Y | P

   type built_in_binop = Jadd | Jsub | Jmult | Jdiv | Jmod | Jequal |
Jneq
                       | Jless | Jleq | Jgreater | Jgeq | Jand | Jor |
Jhas

   type built_in_unaop = Jminus | Jnot

   let spokeexception = "org.spoke.SpokeRuntimeException"

   let spoketype = function
       B -> "org.spoke.SpokeBoolean"
     | I -> "org.spoke.SpokeInteger"
     | F -> "org.spoke.SpokeFloat"
     | S -> "org.spoke.SpokeString"
     | L -> "org.spoke.SpokeList"
     | O -> "org.spoke.SpokeObject"
     | T -> "org.spoke.SpokeTag"
     | N -> "org.spoke.SpokeNull"
```

```
    | Y -> "org.spoke.SpokeAny"
    | P -> "org.spoke.SpokeProgram"

 let mem i f = i ^ "." ^ f

 let array_of t = t ^ "[]"

 let string_of_binop = function
     Jadd     -> mem (spoketype O) "OpAdd"
   | Jsub     -> mem (spoketype O) "OpSub"
   | Jmult    -> mem (spoketype O) "OpMul"
   | Jdiv     -> mem (spoketype O) "OpDiv"
   | Jmod     -> mem (spoketype O) "OpMod"
   | Jequal   -> mem (spoketype O) "OpEq"
   | Jneq     -> mem (spoketype O) "OpNeq"
   | Jless    -> mem (spoketype O) "OpLt"
   | Jleq     -> mem (spoketype O) "OpLeq"
   | Jgreater -> mem (spoketype O) "OpGt"
   | Jgeq     -> mem (spoketype O) "OpGeq"
   | Jand     -> mem (spoketype O) "OpAnd"
   | Jor      -> mem (spoketype O) "OpOr"
   | Jhas     -> mem (spoketype O) "OpHas"

 let string_of_unaop = function
     Jminus -> mem (spoketype O) "OpNeg"
   | Jnot   -> mem (spoketype O) "OpNot"

 let translate java_class =

   let rec code_of_stmt debug = function
       Jdef(t, i) -> debug.indent ^ t ^ " " ^ i ^ ";"
     | Jnew(t, al, i) ->
         debug.indent ^ i ^ " = new " ^ t ^ "(" ^ String.concat ", "
al ^ ");"
     | Jinvoke(i, al, "") -> debug.indent ^ i ^ "(" ^ String.concat ",
" al ^ ");"
     | Jinvoke(i1, al, i) -> debug.indent ^ i ^ " = " ^ i1 ^ "(" ^
String.concat ", " al ^ ");"
     | Jthrow(t, al) ->
         debug.indent ^ "throw new " ^ t ^ "(" ^ String.concat ", "
al ^ ");"
     | Jinstanceof(a, t, i) ->
         debug.indent ^ i ^ " = " ^ a ^ " instanceof " ^ t ^ ";"
     | Jif(i, sl1, []) ->
         debug.indent ^ "if (" ^ i ^ ") {\n" ^
         String.concat "\n" (List.map (code_of_stmt { indent =
debug.indent ^ "  "} ) sl1) ^ "\n" ^
         debug.indent ^ "}"
     | Jif(i, sl1, sl2) ->
         debug.indent ^ "if (" ^ i ^ ") {\n" ^
         String.concat "\n" (List.map (code_of_stmt { indent =
debug.indent ^ "  "} ) sl1) ^ "\n" ^
```

```
           debug.indent ^ "} else {\n" ^
           String.concat "\n" (List.map (code_of_stmt { indent =
debug.indent ^ "  "} ) sl2) ^ "\n" ^
           debug.indent ^ "}"
       | Jwhile(a, sl) ->
           debug.indent ^ "while (" ^ a ^ ") {\n" ^
           String.concat "\n" (List.map (code_of_stmt { indent =
debug.indent ^ "  "} ) sl) ^ "\n" ^
           debug.indent ^ "}"
       | Jfor(i, a, sl) ->
           debug.indent ^ "For (" ^ i ^ " : " ^ a ^ ") {\n" ^
           String.concat "\n" (List.map (code_of_stmt { indent =
debug.indent ^ "  "} ) sl) ^ "\n" ^
           debug.indent ^ "}"
       | Jbreak -> debug.indent ^ "break;"
       | Jcontinue -> debug.indent ^ "continue;"
       | Jreturn("") -> debug.indent ^ "return new " ^ spoketype N ^
"();"
       | Jreturn(i) -> debug.indent ^ "return " ^ i ^ ";"

     in let code_of_arg ag = ag.a_type ^ " " ^ ag.a_name

     in let code_of_access = function
         Public -> "public "
       | Private -> "private "
       | Protected -> "protected "
       | None -> " "

     in let code_of_field fd =
       "  " ^ code_of_access fd.f_access ^ (if fd.f_static then "static
" else "") ^
       fd.f_type ^ " " ^ fd.f_name ^ ";"

     in let code_of_method mt =
       "  " ^ code_of_access mt.m_access ^ (if mt.m_static then "static
" else "") ^
       mt.m_return ^ " " ^ mt.m_name ^ " " ^
       "(" ^ (String.concat ", " (List.map code_of_arg mt.m_arg)) ^ ")
{\n" ^
       String.concat "\n" (List.map (code_of_stmt { indent = "    "})
mt.m_body) ^ "\n" ^
       "  }\n"

     in let code_of_class cl =
       (if (String.length cl.c_package > 0) then "package " ^
cl.c_package ^ ";\n\n" else "") ^
       code_of_access cl.c_access ^ "class " ^ cl.c_name ^
       (if (String.length cl.c_parent > 0) then " extends " ^
cl.c_parent else "") ^ " {\n" ^
       String.concat "\n" (List.map code_of_field cl.c_fields) ^ "\n\n"
^
```

```
      String.concat "\n" (List.map code_of_method cl.c_methods) ^ "\n"
^
      "}\n"

    in code_of_class java_class
```

**spoke.ml:**

```
  ------------
  type opt = Scan | Parse | Intercode | Javaast | Translate | Compile
| Execute

  let _ =

    let srcs =
      let getopt =
        if Array.length Sys.argv > 1 then
          (if Sys.argv.(1).[0] == '-' then [] else [Sys.argv.(1)])
          @ List.tl (List.tl (Array.to_list Sys.argv))
        else []

      in if List.length getopt == 0 then [(["a"], stdin)]
  (*
        let rand_path =
          let gen _ =
            let rand = (fun n -> int_of_char 'A' + n) (Random.int 26)
            in String.make 1 (char_of_int rand)
          in String.concat "" (Array.to_list (Array.init 8 gen))
        in [([rand_path], stdin)]
  *)
      else
        let rec split_char sep str =
          try
            let i = String.index str sep in
              String.sub str 0 i ::
              split_char sep (String.sub str (i+1) (String.length str
- i - 1))
          with Not_found -> [str]

        in let check_path path =
          let rev_path = List.rev path in
          let rm_ext fname =
            try String.sub fname 0 (String.rindex fname '.')
            with Not_found -> fname
          in List.map (fun t -> if String.contains t '.'
                    then raise (Failure ("path with dot: " ^ t)) else
t)
             (rm_ext (List.hd rev_path) :: List.tl rev_path)
        in List.map (fun f -> (check_path (split_char '/' f), open_in
f)) getopt in

    let action =
      if (Array.length Sys.argv) > 1 && Sys.argv.(1).[0] == '-' then
        List.assoc Sys.argv.(1) [("-s", Scan);
                                 ("-p", Parse);
                                 ("-i", Intercode);
                                 ("-j", Javaast);
                                 ("-t", Translate);
                                        ("-c", Compile);
                                 ("-x", Execute)]
      else Compile in

    let compile (path, instream) =
      let name = List.hd path
      and package = String.concat "." (List.rev (List.tl path)) in

      let lexbuf = Lexing.from_channel instream in

      let tokenbuf =
        let l = ref [] in
        fun lexbuf -> match !l with
            x::xs -> l := xs; x
          | [] -> match Scanner.token lexbuf with
                x::xs -> l := xs; x
              | [] -> failwith "oops" in

      if action == Scan then
        let tokenize = Token.tokenize tokenbuf lexbuf
        in print_string tokenize

      else let program = Parser.program tokenbuf lexbuf in
      let checked_program = Checker.check_syntax program in
      if action == Parse then
      let str_program = Ast.string_of_program checked_program
      in print_string str_program

      else let code = Compile.translate checked_program in
      if action == Intercode then
        let str_code = Intercode.string_of_prog code
        in if action == Intercode then print_string str_code
          else print_string str_code

      else let javaast = Translate.translate_java package name code in
      if action == Javaast then
        let str_javaast = Javaast.string_of_class javaast
        in print_string str_javaast

      else let javasrc = Javasrc.translate javaast in
      if action == Translate then print_string javasrc
```

```
        else
          let javafile = open_out (name ^ ".java")
          in output_string javafile javasrc; close_out javafile;
             ignore (Unix.system ("./compile.sh " ^ name));
             print_endline ("Compiling " ^ name ^ ".java done");
          if action == Execute then
             (print_endline ("Executing " ^ name ^ ".java");
              ignore (Unix.system ("./run.sh " ^ name)))
          else ()

      in List.map compile srcs
```

**token.ml:**

```
   ------------
   open Parser

   let tokenize (lexfun : Lexing.lexbuf -> token) (lexbuf :
Lexing.lexbuf) =
      let string_of_token token =
        match token with
          LPAREN      -> "LPAREN"
        | RPAREN      -> "RPAREN"
        | COMMA       -> "COMMA"
        | LBRACK      -> "LBRACK"
        | RBRACK      -> "RBRACK"
        | COLON       -> "COLON"
        | PLUS        -> "PLUS"
        | MINUS       -> "MINUS"
        | TIMES       -> "TIMES"
        | DIVIDE      -> "DIVIDE"
        | MODULUS     -> "MODULUS"
        | ASSIGN      -> "ASSIGN"
        | EQ          -> "EQ"
        | NEQ         -> "NEQ"
        | LT          -> "LT"
        | LEQ         -> "LEQ"
        | GT          -> "GT"
        | GEQ         -> "GEQ"
        | AND         -> "AND"
        | OR          -> "OR"
        | NOT         -> "NOT"
        | IF          -> "IF"
        | THEN        -> "THEN"
        | ELSE        -> "ELSE"
        | ELIF        -> "ELIF"
        | FI          -> "FI"
        | FOR         -> "FOR"
        | IN          -> "IN"
```

```
    | NEXT         -> "NEXT"
    | WHILE        -> "WHILE"
    | LOOP         -> "LOOP"
    | CONTINUE     -> "CONTINUE"
    | BREAK        -> "BREAK"
    | FUNCTION     -> "FUNCTION"
    | END          -> "END"
    | RETURN       -> "RETURN"
    | GLOBAL       -> "GLOBAL"
    | BELONG       -> "BELONG"
    | NULL         -> "NULL"
    | STAR         -> "STAR"
    | STRING s     -> "STRING(" ^ s ^ ")"
    | WORD s       -> "WORD(" ^ s ^ ")"
    | ID s         -> "ID(" ^ s ^ ")"
    | LTPAREN      -> "LTPAREN"
    | RTPAREN      -> "RTPAREN"
    | FLOAT f      -> "FLOAT(" ^ string_of_float f ^ ")"
    | INTEGER i    -> "INTEGER(" ^ string_of_int i ^ ")"
    | BOOL b       -> "BOOL(" ^ string_of_bool b ^ ")"
    | EOL          -> "EOL"
    | EOF          -> "EOF"
  in
    let rec trace s =
      match lexfun lexbuf with
          EOF      -> s ^ "EOF\n"
        | EOL      -> trace (s ^ "EOL\n")
        | t        -> trace (s ^ string_of_token t ^ " ")
    in trace ""
```

**translate.ml:**

```
------------
open Intercode
open Javaast
open Javasrc
open Backend

let rec range i j = if i < j then i :: (range (i+1) j) else []

let translate_java pname cname codes =

  let tmp i = "t_" ^ string_of_int i in
  let glb i = "g_" ^ string_of_int i in
  let loc i = "l_" ^ string_of_int i in
  let fnc i = "f_" ^ string_of_int i in

  let tmpb = "b" and tmpo = "o" in
```

```
    let throw_exception msg = Jthrow(spokeexception, [msg]) in

    let check_and_do var mt do1 do2=
       [ Jinvoke(mem var mt, [], tmpb); Jif(tmpb, do1, do2) ]
    in

    let compare_and_do var1 var2 mt do1 do2 =
       [ Jinvoke(mem var1 mt, [var2], tmpb); Jif(tmpb, do1, do2) ]
    in

    let rec java_of_intercode = function
      Lc(i) -> List.concat (List.map
               (fun i -> [Jdef(spoketype O, loc i); Jnew(spoketype N,
[], loc i)])
               (range (List.length built_in_locals) i))
    | Cb(i, i1) -> [Jnew(spoketype B, [string_of_bool i1], tmp i)]
    | Ci(i, i1) -> [Jnew(spoketype I, [string_of_int i1], tmp i)]
    | Cf(i, i1) -> [Jnew(spoketype F, [string_of_float i1], tmp i)]
    | Cs(i, i1) -> [Jnew(spoketype S, ["\"" ^ i1 ^ "\""], tmp i)]
    | Nl(i)     -> [Jnew(spoketype N, [], tmp i)]
    | Ay(i)     -> [Jnew(spoketype Y, [], tmp i)]
    | Wp(i,-1)      -> [Jinvoke(mem (spoketype L) "Wrap", ["null"], tmp
i)]
    | Wp(i, i1)     -> [Jinvoke(mem (spoketype L) "Wrap", [tmp i1], tmp
i)]
    | Tg(i, i1, i2) -> [Jnew(spoketype T, [tmp i1; tmp i2], tmp i)]
    | Pt(i, i1, i2, i3) ->
        check_and_do (tmp i1) "IsList"
          [Jinvoke(mem (tmp i1) "Partition", [tmp i2; tmp i3], tmp i)]
          [throw_exception "\"Partition on a non-list variable\""]
    | El(i, i1, i2) ->
        check_and_do (tmp i1) "IsList"
          [Jinvoke(mem (tmp i1) "GetElement", [tmp i2], tmp i)]
          [throw_exception "\"Get element on a non-list variable\""]
    | St(i, i1, i2) ->
        check_and_do (tmp i1) "IsList"
          [Jinvoke(mem (tmp i1) "SetElement", [tmp i2; tmp i], "")]
          [throw_exception "\"Set element on a non-list variable\""]
    | Ll(i, i1) ->
        compare_and_do (tmp i) (loc i1) "IsCompatible"
          [Jinvoke(mem (tmp i) "Assign", [loc i1], "")]
          [Jinvoke(mem (loc i1) "Clone", [], tmp i)]
    | Sl(i, i1) ->
        compare_and_do (loc i1) (tmp i) "IsCompatible"
          [Jinvoke(mem (loc i1) "Assign", [tmp i], "")]
          [Jinvoke(mem (tmp i) "Clone", [], loc i1)]
    | Lg(i, i1) ->
        compare_and_do (tmp i) (glb i1) "IsCompatible"
          [Jinvoke(mem (tmp i) "Assign", [glb i1], "")]
          [Jinvoke(mem (glb i1) "Clone", [], tmp i)]
    | Sg(i, i1) ->
        compare_and_do (glb i1) (tmp i) "IsCompatible"
```

```
           [Jinvoke(mem (glb i1) "Assign", [tmp i], "")]
           [Jinvoke(mem (tmp i) "Clone", [], glb i1)]
     | Bn(i, Ast.Add,     i1, i2) ->
         compare_and_do (tmp i1) (tmp i2) "IsCompatible"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jadd;
tmp i2], tmp i)]
           [throw_exception "\"Operation on incompatible types\""]
     | Bn(i, Ast.Sub,     i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jsub;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Mult,    i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jmult;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Div,     i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jdiv;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Mod,     i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jmod;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Equal,   i1, i2) ->
         compare_and_do (tmp i1) (tmp i2) "IsComparable"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jequal;
tmp i2], tmp i)]
           [throw_exception "\"Operation on incomparable types\""]
     | Bn(i, Ast.Neq,     i1, i2) ->
         compare_and_do (tmp i1) (tmp i2) "IsComparable"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jneq;
tmp i2], tmp i)]
           [throw_exception "\"Operation on incomparable types\""]
     | Bn(i, Ast.Less,    i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jless;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Leq,     i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jleq;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Greater, i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop
Jgreater; tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.Geq,     i1, i2) ->
         check_and_do (tmp i1) "IsEnumerated"
           (compare_and_do (tmp i1) (tmp i2) "IsCompatible"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jgeq;
tmp i2], tmp i)]
              [throw_exception "\"Operation on incompatible types\""])
           [throw_exception "\"Operation on non-enumerated types\""]
     | Bn(i, Ast.And,     i1, i2) ->
         check_and_do (tmp i1) "IsBoolean"
           (check_and_do (tmp i2) "IsBoolean"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jand;
tmp i2], tmp i)]
              [throw_exception "\"Operation on non-boolean types\""])
           [throw_exception "\"Operation on non-boolean types\""]
     | Bn(i, Ast.Or,      i1, i2) ->
         check_and_do (tmp i1) "IsBoolean"
           (check_and_do (tmp i2) "IsBoolean"
              [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jor;
tmp i2], tmp i)]
              [throw_exception "\"Operation on non-boolean types\""])
           [throw_exception "\"Operation on non-boolean types\""]
     | Bn(i, Ast.Has,     i1, i2) ->
         [Jinvoke(mem (spoketype P) "FlushMatch", [], "")] @
         compare_and_do (tmp i1) (tmp i2) "MayHave"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_binop Jhas;
tmp i2], tmp i)]
           [throw_exception "\"Operation on incompatible types\""]
     | Un(i, Ast.Minus, i1) ->
         check_and_do (tmp i1) "IsEnumerated"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_unaop Jminus;
"null"], tmp i)]
           [throw_exception "\"Operation on non-enumerated types\""]
     | Un(i, Ast.Not,   i1) ->
         check_and_do (tmp i1) "IsBoolean"
           [Jinvoke(mem (tmp i1) "Operation", [string_of_unaop Jnot;
"null"], tmp i)]
           [throw_exception "\"Operation on non-boolean types\""]
     | Rt(-1)           -> [Jreturn("")]
     | Rt(i)            -> [Jreturn(tmp i)]
     | Cl(i, i1, i2)    -> if i1 < built_in_base then [Jinvoke(fnc i1,
[tmp i2], tmp i)]
                           else [Jinvoke(List.nth
built_in_java_functions (i1-built_in_base),
                                 [tmp i2], tmp i)]
     | Br(i, i1, i2)    -> [Jinvoke(mem (tmp i) "IsTrue", [], tmpb);
                            Jif(tmpb, List.concat (List.map
java_of_intercode i1),
                            List.concat (List.map java_of_intercode
i2))]
     | Lp(i) -> [Jwhile("true", List.concat(List.map java_of_intercode
i))]
     | Bk    -> [Jbreak]
     | Ct    -> [Jcontinue]
     | Gl(_) -> []
     | Fn(_) -> []

     in let rec sum_gvar = function
         [] -> 0
       | Gl(i) :: cl -> i + sum_gvar cl
       | _ :: cl -> sum_gvar cl

     in let gvar =
       List.concat (List.map
        (fun i -> [{f_access = Private; f_type = spoketype O;
                    f_static = false; f_name = glb i}])
        (range (List.length built_in_globals) (sum_gvar codes)))

     in let gnew =
       List.concat (List.map
        (fun i -> [Jnew(spoketype N, [], glb i)])
        (range (List.length built_in_globals) (sum_gvar codes)))

     in let tdec codes =
       let rec add_tmp = function
         [] -> 0
       | e :: el -> let get_tmp = function
                     Cb(i, _)    -> i
                   | Ci(i, _)    -> i
                   | Cf(i, _)    -> i
                   | Cs(i, _)    -> i
                   | Ay(i)    -> i
                   | Nl(i)    -> i
                   | Wp(i,_)         -> i
                   | Pt(i, _, _, _) -> i
                   | Tg(i, _, _) -> i
                   | El(i, _, _) -> i
                   | Ll(i, _)    -> i
                   | Lg(i, _)    -> i
                   | Cl(i, _, _) -> i
                   | Bn(i, _, _, _) -> i
```

```
                    | Un(i, _, _) -> i
                    | Br(_, s1, s2) -> let t = add_tmp s1 and t' =
add_tmp s2 in
                                       if t > t' then t else t'
                    | Lp(s)      -> add_tmp s
                    | _           -> 0 in
                    let t = (get_tmp e) + 1 and t' = add_tmp el in
                    if t > t' then t else t'
      in List.concat (List.map (fun i -> [Jdef(spoketype O, tmp i);
                                           Jnew(spoketype N, [], tmp
i)])
         (range 0 (add_tmp codes)))

     in let fdec =
       let rec add_fdec i = function
         [] -> []
       | Fn(el) :: cl -> { m_name = fnc i; m_access = Public; m_static
= false;
                           m_arg = List.map (fun i -> { a_type =
spoketype O; a_name = loc i })
                                             (range 0 (List.length
built_in_locals));
                           m_return = spoketype O;
                           m_body = [Jdef("boolean", tmpb);
Jdef(spoketype O, tmpo)] @
                                   tdec el @ List.concat (List.map
java_of_intercode el) }
                         :: add_fdec (i+1) cl
       | _ :: cl -> add_fdec i cl
       in add_fdec 0 codes

     in let init = { m_name = "Init"; m_access = Public; m_static =
false;
                     m_arg = List.map (fun i -> { a_type = spoketype O;
a_name = loc i })
                                       (range 0 (List.length
built_in_locals));
                     m_return = "void";
                     m_body = gnew @ [Jdef("boolean", tmpb);
Jdef(spoketype O, tmpo)] @
                              tdec codes @ List.concat (List.map
java_of_intercode codes) }

     in let main = { m_name = "main"; m_access = Public; m_static =
true;
                     m_arg = [{ a_type = "String[]"; a_name = "args"}];
                     m_return = "void";
                     m_body = [Jdef(cname, "myMain"); Jnew(cname, [],
"myMain");
                               Jinvoke("Execute", ["myMain"; "args"],
"")] }
```

```
     in { c_package = ""; c_access = Public; c_name = cname;
          c_parent = spoketype P; c_fields = gvar;
          c_methods = main :: (init :: fdec); }
```

**run.sh**

```
   ------------
   #!/bin/bash

   java -classpath backend/src:backend/lib/stanford-
parser.jar:backend/lib/stanford-postagger.jar: $*
```

**compile.sh**

```
   ------------
   #!/bin/bash

   javac -classpath backend/src:backend/lib/stanford-
parser.jar:backend/lib/stanford-postagger.jar: $1.java
```

**Makefile:**

```
   ------------
   OBJS = ast.cmo parser.cmo token.cmo scanner.cmo intercode.cmo
backend.cmo \
          checker.cmo compile.cmo javaast.cmo javasrc.cmo translate.cmo
\
              spoke.cmo

   spoke : $(OBJS)
           ocamlc -g -o spoke unix.cma $(OBJS)

   scanner.ml : scanner.mll
           ocamllex scanner.mll

   parser.ml parser.mli : parser.mly
           ocamlyacc -v parser.mly

   %.cmo : %.ml
           ocamlc -g -c $<
   %.cmi : %.mli
           ocamlc -g -c $<

   .PHONY : clean
   clean :
           rm -f spoke parser.ml parser.mli scanner.ml *.cmo *.cmi
*.output *.diff *~

   # Generated by ocamldep *.ml *.mli
   ast.cmo:
   ast.cmx:
   token.cmo: parser.cmo
   token.cmx: parser.cmx
   parser.cmo: ast.cmo parser.cmi
   parser.cmx: ast.cmx parser.cmi
   parser.cmi: ast.cmo
   scanner.cmo: ast.cmo
   scanner.cmx: ast.cmx
   checker.cmo: ast.cmo backend.cmo
   checker.cmx: ast.cmx backend.cmx
   intercode.cmo:
   intercode.cmx:
   backend.cmo: backend.cmi
   backend.cmx: backend.cmi
   backend.cmi:
   compile.cmo: intercode.cmo backend.cmo ast.cmo
   compile.cmx: intercode.cmx backend.cmx ast.cmx
   javaast.cmo:
   javaast.cmx:
   javasrc.cmo: backend.cmo javaast.cmo
   javasrc.cmx: backend.cmx javaast.cmx
   translate.cmo: intercode.cmo backend.cmo javaast.cmo
   translate.cmx: intercode.cmx backend.cmx javaast.cmx
   spoke.cmo: scanner.cmo token.cmo parser.cmi ast.cmo compile.cmo
backend.cmo checker.cmo intercode.cmo javaast.cmo javasrc.cmo
   spoke.cmx: scanner.cmx token.cmx parser.cmx ast.cmx compile.cmx
backend.cmx checker.cmx intercode.cmx javaast.cmx havasrc.cmx
```

**Back End:**

**lib:**

```
   stanford-parser.jar:
   stanford-postagger.jar:
```

**model:**

```
englishPCFG.ser.gz:
left3words-wsj-0-18.tagger:
left3words-wsj-0-18.tagger.props:
```

**src/org/spoke:**

**SpokeBoolean.java**

```
------------
package org.spoke;

public class SpokeBoolean extends SpokeObject {
    private Boolean myObject;

    private void setObject(Object object) {
        if (object instanceof SpokeBoolean)
            setBoolean(((SpokeBoolean)object).getBoolean());

        else if (object instanceof Boolean)
            setBoolean(((Boolean)object).booleanValue());

        else
            throw new SpokeException("type does not match");
    }

    public SpokeBoolean(boolean value) {
        setBoolean(value);
    }

    public SpokeBoolean(Object object) {
        setObject(object);
    }

    public void setBoolean(boolean value) {
        this.myObject = new Boolean(value);
    }

    public boolean getBoolean() {
        return myObject.booleanValue();
    }

        @Override
        public boolean IsTrue() {
            return getBoolean();
        }
```

```
            @Override
            public boolean IsCompatible(SpokeObject object) {
                    return (object instanceof SpokeBoolean);
            }

        @Override
        public void Assign(SpokeObject object) {
            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny)
                throw new SpokeException("operation with any");

            setObject(object);
        }

        @Override
        public SpokeObject Clone() {
            return new SpokeBoolean(this);
        }

            @Override
            public boolean IsBoolean() {
                    return true;
            }

            @Override
            public boolean IsComparable(SpokeObject object) {
                    return object instanceof SpokeBoolean ||
                           object instanceof SpokeAny || object
instanceof SpokeNull;
            }

            @Override
            public boolean IsEnumerated() {
                    return false;
            }

            @Override
            public boolean MayHave(SpokeObject object) {
                    return object instanceof SpokeBoolean || object
instanceof SpokeAny;
            }

        @Override
        public SpokeObject Operation(int Op, SpokeObject object) {
            if (Op == SpokeObject.OpNot) {
                    return new SpokeBoolean(!myObject.booleanValue());
            }

            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
```

```
                return new SpokeBoolean(false);

        if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                return new SpokeBoolean(true);

        if (object == null || object instanceof SpokeNull)
            throw new SpokeException("operation with null");

        if (object instanceof SpokeAny && Op == SpokeObject.OpHas) {
                SpokeProgram.AddMatch(this);
            return new SpokeBoolean(true);
        }

        if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
            return new SpokeBoolean(true);

        if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
            return new SpokeBoolean(false);

        if (!(object instanceof SpokeBoolean))
            throw new SpokeException("type does not match");

        boolean myValue = myObject.booleanValue();
        boolean value = ((SpokeBoolean)object).getBoolean();

        switch(Op) {
        /* Comparison Operators */
        case SpokeObject.OpEq:
                return new SpokeBoolean(myValue == value);
        case SpokeObject.OpNeq:
            return new SpokeBoolean(myValue != value);
        case SpokeObject.OpHas:
                if (myValue == value) {
                        SpokeProgram.AddMatch(this);
                        return new SpokeBoolean(true);
                }
                else
                        return new SpokeBoolean(false);
        /* Boolean Operators */
        case SpokeObject.OpAnd:
            return new SpokeBoolean(myValue && value);
        case SpokeObject.OpOr:
            return new SpokeBoolean(myValue || value);
        /* Unsupported Operators */
        default:
            throw new SpokeException("operation is not supported");
        }
    }

        @Override
        public boolean IsList() {
                return false;
```

```
          }

      @Override
      public SpokeObject Partition(SpokeObject start, SpokeObject end)
{
          throw new SpokeException("operation is not supported");
      }

          @Override
          public void SetElement(SpokeObject index, SpokeObject object)
{
                  throw new SpokeException("operation is not
supported");
          }

          @Override
          public SpokeObject GetElement(SpokeObject index) {
                  throw new SpokeException("operation is not
supported");
          }

          @Override
          public String toString() {
                  return getBoolean() ? "(True)" : "(False)";
          }
   }
```

**SpokeException.java**

```
   ------------
   package org.spoke;

   public class SpokeException extends RuntimeException {
       private static final long serialVersionUID = -
6549270043287893286L;

       public SpokeException(String message) {
           super(message);
       }
   }
```

**SpokeFloat.java**

```
   ------------
   package org.spoke;

   public class SpokeFloat extends SpokeObject {
        private Double myObject;

        private void setObject(Object object) {
            if (object instanceof SpokeFloat)
                setFloat(((SpokeFloat)object).getFloat());

            else if (object instanceof Double)
                setFloat(((Double)object).doubleValue());

            else if (object instanceof SpokeInteger)
                setFloat(((SpokeInteger)object).getInteger());

            else if (object instanceof Integer)
                    setFloat(((SpokeInteger)object).getInteger());

            else
                throw new SpokeException("type does not match");
        }

        public SpokeFloat(Object object) {
            setObject(object);
        }

        public void setFloat(double value) {
            this.myObject = new Double(value);
        }

        public double getFloat() {
            return myObject.doubleValue();
        }

            @Override
            public boolean IsTrue() {
                    return getFloat() != 0.0;
            }

            @Override
            public boolean IsCompatible(SpokeObject object) {
                return (object instanceof SpokeFloat || object
instanceof SpokeInteger);
            }

        @Override
        public void Assign(SpokeObject object) {
            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny)
                throw new SpokeException("operation with any");

            setObject(object);
        }
        @Override
        public SpokeObject Clone() {
            return new SpokeFloat(this);
        }

            @Override
            public boolean IsBoolean() {
                return false;
            }

            @Override
            public boolean IsComparable(SpokeObject object) {
                    return object instanceof SpokeFloat ||
                            object instanceof SpokeAny || object
instanceof SpokeNull;
            }

            @Override
            public boolean IsEnumerated() {
                return false;
            }

            @Override
            public boolean MayHave(SpokeObject object) {
                    return object instanceof SpokeFloat || object
instanceof SpokeAny;
            }

        @Override
        public SpokeObject Operation(int Op, SpokeObject object) {
            if (Op == SpokeObject.OpNeg)
                return new SpokeFloat(-myObject.doubleValue());

            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(false);

            if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                    return new SpokeBoolean(true);

            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny && Op == SpokeObject.OpHas) {
                    SpokeProgram.AddMatch(this);
                    return new SpokeBoolean(true);
            }

            if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
                return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
```

```
                return new SpokeBoolean(false);

            if (!(object instanceof SpokeFloat) && !(object instanceof 
SpokeInteger))
                throw new SpokeException("type does not match");

            Double myValue = myObject.doubleValue();
            Double value = object instanceof SpokeFloat ?
                                    ((SpokeFloat)object).getFloat() :
                            ((SpokeInteger)object).getInteger();

            switch(Op) {
            /* Arithmetic Operators */
            case SpokeObject.OpAdd:
                return new SpokeFloat(myValue + value);
            case SpokeObject.OpSub:
                return new SpokeFloat(myValue - value);
            case SpokeObject.OpDiv:
                return new SpokeFloat(myValue * value);
            case SpokeObject.OpMul:
                return new SpokeFloat(myValue / value);
            case SpokeObject.OpMod:
                return new SpokeFloat(myValue % value);
            /* Comparison Operators */
            case SpokeObject.OpEq:
                return new SpokeBoolean(myValue == value);
            case SpokeObject.OpNeq:
                return new SpokeBoolean(myValue != value);
            case SpokeObject.OpGt:
                return new SpokeBoolean(myValue >  value);
            case SpokeObject.OpGeq:
                return new SpokeBoolean(myValue >= value);
            case SpokeObject.OpLt:
                return new SpokeBoolean(myValue <  value);
            case SpokeObject.OpLeq:
                return new SpokeBoolean(myValue <= value);
            case SpokeObject.OpHas:
                    if (myValue == value) {
                            SpokeProgram.AddMatch(this);
                            return new SpokeBoolean(true);
                    }
                    else
                            return new SpokeBoolean(false);
            /* Unsupported Operators */
            default:
                throw new SpokeException("operation is not supported");
            }
        }

        @Override
        public boolean IsList() {
            return false;
        }

        @Override
        public SpokeObject Partition(SpokeObject start, SpokeObject end)
{
            throw new SpokeException("this method is not supported");
        }

            @Override
            public void SetElement(SpokeObject index, SpokeObject object)
{
                    throw new SpokeException("operation is not
supported");
            }

            @Override
            public SpokeObject GetElement(SpokeObject index) {
                    throw new SpokeException("operation is not
supported");
            }

            @Override
            public String toString() {
                    return myObject.toString();
            }
    }
```

**SpokeInteger.java**

```
    ------------
    package org.spoke;

    public class SpokeInteger extends SpokeObject {
        private Integer myObject;

        private void setObject(Object object) {
            if (object instanceof SpokeInteger)
                setInteger(((SpokeInteger)object).getInteger());

            else if (object instanceof Integer)
                setInteger(((Integer)object).intValue());

            else if (object instanceof SpokeFloat)

setInteger((int)Math.round((((SpokeFloat)object).getFloat())));

            else if (object instanceof Double)
                setInteger(((Double)object).intValue());
            else
                throw new SpokeException("type does not match");
        }

        public SpokeInteger(Object object) {
            setObject(object);
        }

        public SpokeInteger(int value) {
            setInteger(value);
        }

        public void setInteger(int value) {
            myObject = new Integer(value);
        }

        public int getInteger() {
            return myObject.intValue();
        }

            @Override
            public boolean IsTrue() {
                    return getInteger() != 0;
            }

            @Override
            public boolean IsCompatible(SpokeObject object) {
            return (object instanceof SpokeInteger || object instanceof
SpokeFloat);
        }

        @Override
        public void Assign(SpokeObject object) {
            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny)
                throw new SpokeException("operation with any");

            setObject(object);
        }

        @Override
        public SpokeObject Clone() {
            return new SpokeInteger(getInteger());
        }

        @Override
        public boolean IsBoolean() {
            return false;
        }
```

```
            @Override
        public boolean IsComparable(SpokeObject object) {
            return object instanceof SpokeInteger ||
                   object instanceof SpokeAny || object instanceof
SpokeNull;
        }

            @Override
            public boolean IsEnumerated() {
            return true;
        }

            @Override
            public boolean MayHave(SpokeObject object) {
            return object instanceof SpokeInteger || object instanceof
SpokeAny;
            }

        @Override
        public SpokeObject Operation(int Op, SpokeObject object) {
            if (Op == SpokeObject.OpNeg)
                return new SpokeFloat(-myObject.intValue());

            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(false);

            if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                    return new SpokeBoolean(true);

            if (object == null || object instanceof SpokeNull)
                    throw new SpokeException("operation with null");

            if (object instanceof SpokeAny && Op == SpokeObject.OpHas) {
                    SpokeProgram.AddMatch(this);
                    return new SpokeBoolean(true);
            }

            if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
                return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
                return new SpokeBoolean(false);

            if (!(object instanceof SpokeFloat) && !(object instanceof
SpokeInteger))
                throw new SpokeException("type does not match");

            double myValue = myObject.floatValue();
            double value = object instanceof SpokeFloat ?
                                ((SpokeFloat)object).getFloat() :
                         ((SpokeInteger)object).getInteger();
```

```
               switch(Op) {
               /* Arithmetic Operators */
               case SpokeObject.OpAdd:
                   return new SpokeInteger(myValue + value);
                   case SpokeObject.OpSub:
                       return new SpokeInteger(myValue - value);
                   case SpokeObject.OpDiv:
                           return new SpokeInteger(myValue * value);
                   case SpokeObject.OpMul:
                           return new SpokeInteger(myValue / value);
                   case SpokeObject.OpMod:
                           return new SpokeInteger(myValue % value);
                   /* Comparison Operators */
                   case SpokeObject.OpEq:
                           return new SpokeBoolean(myValue == value);
                   case SpokeObject.OpNeq:
                           return new SpokeBoolean(myValue != value);
                   case SpokeObject.OpGt:
                           return new SpokeBoolean(myValue >  value);
                   case SpokeObject.OpGeq:
                           return new SpokeBoolean(myValue >= value);
                   case SpokeObject.OpLt:
                           return new SpokeBoolean(myValue <  value);
                   case SpokeObject.OpLeq:
                           return new SpokeBoolean(myValue <= value);
           case SpokeObject.OpHas:
                   if (myValue == value) {
                           SpokeProgram.AddMatch(this);
                           return new SpokeBoolean(true);
                   }
                   else
                           return new SpokeBoolean(false);
                   /* Unsupported Operators */
                   default:
                           throw new SpokeException("operation is not
supported");
               }
       }

       @Override
       public boolean IsList() {
           return false;
       }

       @Override
       public SpokeObject Partition(SpokeObject start, SpokeObject end)
{
           throw new SpokeException("This method is not supported");
       }

           @Override
```

```
            public void SetElement(SpokeObject index, SpokeObject object)
{
                    throw new SpokeException("operation is not
supported");
            }

            @Override
            public SpokeObject GetElement(SpokeObject index) {
                    throw new SpokeException("operation is not
supported");
            }

            @Override
            public String toString() {
                    return myObject.toString();
            }
   }
```

**SpokeAny.java**

```
   ------------
   package org.spoke;

   public class SpokeAny extends SpokeObject {

           @Override
           public boolean IsTrue() {
                   return true;
           }

           @Override
           public boolean IsCompatible(SpokeObject object) {
                   return object instanceof SpokeAny;
           }

       @Override
       public void Assign(SpokeObject object) {
           if (!(object instanceof SpokeAny))
               throw new SpokeException("operation is not supported");
       }

       @Override
       public SpokeObject Clone() {
           return this;
       }

           @Override
           public boolean IsBoolean() {
                return false;
            }

            @Override
            public boolean IsComparable(SpokeObject object) {
                return true;
            }

            public boolean IsEnumerated() {
                return false;
            }

            public boolean MayHave(SpokeObject object) {
                return object instanceof SpokeAny;
            }

        @Override
        public SpokeObject Operation(int Op, SpokeObject object) {
            switch(Op) {
            /* Comparison Operators */
            case SpokeObject.OpAdd:
                    if (!(object instanceof SpokeAny))
                        throw new SpokeException("operation is not
supported");
            /* Comparison Operators */
            case SpokeObject.OpEq:
                return new SpokeBoolean(true);
            case SpokeObject.OpNeq:
                return new SpokeBoolean(false);
            case SpokeObject.OpHas:
                    SpokeProgram.AddMatch(this);
                    return new SpokeBoolean(true);
            /* Unsupported Operators */
            default:
                throw new SpokeException("operation is not supported");
            }
        }

            @Override
            public boolean IsList() {
                    return false;
            }

            @Override
            public SpokeObject Partition(SpokeObject start, SpokeObject
end) {
                    throw new SpokeException("operation is not
supported");
            }

            @Override
```

```
		public void SetElement(SpokeObject index, SpokeObject object)
{
			throw new SpokeException("operation is not
supported");
		}

		@Override
		public SpokeObject GetElement(SpokeObject index) {
			throw new SpokeException("operation is not
supported");
		}

		@Override
		public String toString() {
			return "(Any)";
		}
   }
```

## SpokeFile.java

```
   ------------
   package org.spoke;

   public class SpokeFile extends SpokeObject {
		private java.io.FileDescriptor myFD;

		public void setObject(Object object) {
			if (object instanceof SpokeFile)
				setFD(((SpokeFile)object).getFD());

			else if (object instanceof java.io.FileDescriptor)
				setFD((java.io.FileDescriptor)object);

			else
				throw new SpokeException("Type does not
match");
		}

		public SpokeFile(Object object) {
			setObject(object);
		}

		public void setFD(java.io.FileDescriptor FD) {
			myFD = FD;
		}

		public java.io.FileDescriptor getFD() {
			return myFD;
		}
```

```
public boolean CanRead() {
        try {
                System.getSecurityManager().checkRead(myFD);
                return true;
        } catch (SecurityException e) {
                return false;
        }
}

public boolean CanWrite() {
        try {
                System.getSecurityManager().checkWrite(myFD);
                return true;
        } catch (SecurityException e) {
                return false;
        }
}

@Override
public boolean IsTrue() {
        return true;
}

@Override
public boolean IsCompatible(SpokeObject object) {
        return object instanceof SpokeFile;
}

@Override
public void Assign(SpokeObject object) {
if (object == null || object instanceof SpokeNull)
    throw new SpokeException("operation with null");

if (object instanceof SpokeAny)
    throw new SpokeException("operation with any");

setObject(object);
}

@Override
public SpokeObject Clone() {
        return new SpokeFile(myFD);
}

@Override
public boolean IsBoolean() {
        return false;
}

@Override
public boolean IsComparable(SpokeObject object) {
			return object instanceof SpokeNull;
		}

		@Override
		public boolean IsEnumerated() {
			return false;
		}

		@Override
		public boolean MayHave(SpokeObject object) {
			return false;
		}

		@Override
		public SpokeObject Operation(int Op, SpokeObject object) {
		if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
			return new SpokeBoolean(false);

		if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
			return new SpokeBoolean(true);

			throw new SpokeException("This method is not
supported");
		}

		@Override
		public boolean IsList() {
			return false;
		}

		@Override
		public SpokeObject Partition(SpokeObject start, SpokeObject
end) {
			throw new SpokeException("This method is not
supported");
		}

		@Override
		public void SetElement(SpokeObject index, SpokeObject object)
{
			throw new SpokeException("This method is not
supported");
		}

		@Override
		public SpokeObject GetElement(SpokeObject index) {
			throw new SpokeException("This method is not
supported");
		}

		@Override
		public String toString() {
```

```
                return "(File)";
            }
}
```

## SpokeList.java

```
------------
package org.spoke;


public class SpokeList extends SpokeObject {
    private SpokeObject[] myList;

    private void setObject(Object object) {
        if (object instanceof SpokeList) {
                setList(((SpokeList)object).getList());
        }
        else if (object instanceof SpokeObject[]) {
            setList((SpokeObject[])object);
        }
        else {
            throw new SpokeException("type does not match");
        }
    }

    public static SpokeList Wrap(SpokeObject object) {
        SpokeList newList = new SpokeList();
        if (object != null)
                newList.addObject(object);
        return newList;
    }

    public void addObject(SpokeObject object) {
        SpokeObject[] newList = new SpokeObject[myList.length + 1];
        for (int i = 0 ; i < myList.length ; i++)
                newList[i] = myList[i];
        newList[myList.length] = ((SpokeObject)object).Clone();
        myList = newList;
    }

    public SpokeList() {
        setObject(new SpokeObject[0]);
    }

    public SpokeList(Object object) {
        setObject(object);
    }

    public void setList(SpokeObject[] List) {
            myList = new SpokeObject[List.length];
            for (int i = 0 ; i < List.length ; i++)
                myList[i] = List[i];
        }

        public SpokeObject[] getList() {
            return myList;
        }

            @Override
            public boolean IsTrue() {
                    return myList.length > 0;
            }

            @Override
            public boolean IsCompatible(SpokeObject object) {
                    return object instanceof SpokeList;
            }

        @Override
        public void Assign(SpokeObject object) {
            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny)
                throw new SpokeException("operation with any");

            if (!(object instanceof SpokeList))
                    throw new SpokeException("type does not match");

            setObject(object);
        }

        @Override
        public SpokeObject Clone() {
            return new SpokeList(myList);
        }

            @Override
            public boolean IsBoolean() {
                    return false;
            }

            @Override
            public boolean IsComparable(SpokeObject object) {
                    return object instanceof SpokeList ||
                           object instanceof SpokeAny || object
instanceof SpokeNull;
            }

            @Override
            public boolean IsEnumerated() {
                    return false;
            }

        @Override
            public boolean MayHave(SpokeObject object) {
            return true;
            }

        @Override
        public SpokeObject Operation(int Op, SpokeObject object) {
            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(false);

            if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                    return new SpokeBoolean(true);

            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny && Op == SpokeObject.OpHas) {
                SpokeProgram.AddMatch(this);
                return new SpokeBoolean(true);
            }

            if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
                return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
                return new SpokeBoolean(false);

            if (!(object instanceof SpokeList) && Op ==
    SpokeObject.OpHas) {
                    boolean hasObject = false;
                    for (int i = 0 ; i < myList.length ; i++) {
                            if (myList[i].IsComparable(object) &&
                                myList[i].Operation(SpokeObject.OpHas,
    object).IsTrue())
                                    hasObject = true;
                    }
                    return new SpokeBoolean(hasObject);
            }

            if (!(object instanceof SpokeList))
                throw new SpokeException("type does not match");

            SpokeObject[] List = ((SpokeList)object).getList();

            switch(Op) {
            /* Arithmetic Operators */
            case SpokeObject.OpAdd:
                SpokeObject[] newList = new SpokeObject[myList.length +
    List.length];
```

```
                for (int i = 0 ; i < myList.length ; i++)
                    newList[i] = myList[i];
                for (int i = 0 ; i < List.length ; i++)
                    newList[myList.length + i] = List[i];
                return new SpokeList(newList);
            /* Comparison Operators */
            case SpokeObject.OpEq:
                    if (List.length != myList.length)
                            return new SpokeBoolean(false);
                    boolean eqObject = true;
                for (int i = 0 ; i < myList.length ; i++) {
                    if (myList[i].IsComparable(List[i]) &&
                        myList[i].Operation(OpEq, List[i]).IsTrue())
                            continue;
                    eqObject = false;
                }
                return new SpokeBoolean(eqObject);
            case SpokeObject.OpNeq:
                    if (List.length != myList.length)
                            return new SpokeBoolean(true);
                    boolean neqObject = false;
                for (int i = 0 ; i < myList.length ; i++) {
                    if (myList[i].IsComparable(List[i]) &&
                        myList[i].Operation(OpEq, List[i]).IsTrue())
                            continue;
                    neqObject = true;
                }
                return new SpokeBoolean(neqObject);
            case SpokeObject.OpHas:
                    boolean hasObject = false;
                for (int i = 0 ; i < myList.length - List.length + 1 ;
i++) {
                    for (int j = 0 ; j < List.length ; j++) {
                        if (myList[i + j].IsComparable(List[j]) &&
                            myList[i + j].Operation(SpokeObject.OpEq,
List[j]).IsTrue()) {
                            if (j == List.length - 1) {
                                    SpokeObject[] matchList = new
SpokeObject[List.length];
                                    for (int k = 0 ; k < List.length ;
k++)
                                            matchList[k] = myList[i + k];
                                    SpokeProgram.AddMatch(new
SpokeList(matchList));
                                    hasObject = true;
                            }
                            continue;
                        }
                        break;
                    }
                }
                for (int i = 0 ; i < myList.length ; i++) {
                    if (myList[i].MayHave(object) &&
                            myList[i].Operation(SpokeObject.OpHas,
object).IsTrue()) {
                            hasObject = true;
                    }
                }
                return new SpokeBoolean(hasObject);
                /* Unsupported Operators */
                default:
                    throw new SpokeException("operation is not
supported");
            }
        }

            @Override
            public boolean IsList() {
                    return true;
            }

        @Override
        public SpokeObject Partition(SpokeObject start, SpokeObject end)
{
            if (!(start instanceof SpokeInteger))
                throw new SpokeException("type does not match");

            if (!(end instanceof SpokeInteger))
                    throw new SpokeException("type does not match");

            int sindex = ((SpokeInteger)start).getInteger();
            int eindex = ((SpokeInteger)end).getInteger();

            if (sindex < 0) {
                sindex = myList.length + sindex;
            }
            if (eindex < 0) {
                eindex = myList.length + eindex;
            }
            if (sindex >= myList.length) {
                throw new SpokeException("List boundary is violated");
            }
            if (eindex >= myList.length) {
                throw new SpokeException("List boundary is violated");
            }
            if (sindex > eindex) {
                return new SpokeList(new SpokeObject[0]);
            }

            SpokeObject[] List = new SpokeObject[eindex - sindex + 1];
            for (int i = sindex ; i <= eindex ; i++) {
                List[i - sindex] = myList[i];
            }
            return new SpokeList(List);
        }

            @Override
            public void SetElement(SpokeObject index, SpokeObject object)
    {
            if (!(index instanceof SpokeInteger))
                throw new SpokeException("type does not match");

            int eindex = ((SpokeInteger)index).getInteger();

            if (eindex < 0) {
                eindex = myList.length + eindex;
            }
            if (eindex >= myList.length) {
                throw new SpokeException("List boundary is violated");
            }

            myList[eindex] = object.Clone();
            }

            @Override
            public SpokeObject GetElement(SpokeObject index) {
            if (!(index instanceof SpokeInteger))
                throw new SpokeException("type does not match");

            int eindex = ((SpokeInteger)index).getInteger();

            if (eindex < 0) {
                eindex = myList.length + eindex;
            }
            if (eindex >= myList.length) {
                throw new SpokeException("List boundary is violated");
            }

            return myList[eindex];
            }

            @Override
            public String toString() {
                    String result = "[";
                    for (int i = 0 ; i < myList.length ; i++) {
                result += myList[i].toString() + (i < myList.length - 1 ?
    ", " : "");
                    }
                    result += "]";
                    return result;
            }
      }
```

## SpokeObject.java

```
------------
package org.spoke;

public abstract class SpokeObject extends Object {
    /* 0-15: Arithmetic Operators   */
    public static final int OpAdd = 0;
    public static final int OpSub = 1;
    public static final int OpMul = 2;
    public static final int OpDiv = 3;
    public static final int OpMod = 4;
    public static final int OpNeg = 5;
    /* 16-31: Comparison Operators  */
    public static final int OpEq  = 16;
    public static final int OpNeq = 17;
    public static final int OpGt  = 18;
    public static final int OpGeq = 19;
    public static final int OpLt  = 20;
    public static final int OpLeq = 21;
    public static final int OpHas = 22;
    /* 32-40: Boolean gates         */
    public static final int OpAnd = 32;
    public static final int OpOr  = 33;
    public static final int OpNot = 34;

    /* Used in Branch or Loop to check if this object indicate True
*/
    abstract public boolean IsTrue();

    /* Used in Assignment to check if the type is compatible */
    abstract public boolean IsCompatible(SpokeObject object);

    /* Used as the easy Assignment when the type is compatible */
    abstract public void Assign(SpokeObject object);

    /* Used as the easy Assignment when the type is incompatible */
    abstract public SpokeObject Clone();

    /* Used in Operation to check if the type is Boolean */
    abstract public boolean IsBoolean();

    /* Used in Operation to check if the type can be compared */
    abstract public boolean IsComparable(SpokeObject object);

    /* Used in Operation to check if the type is enumerated */
    abstract public boolean IsEnumerated();

    /* Used in Operation to check if the type may have another */
    abstract public boolean MayHave(SpokeObject object);

    /* Used as the easy Operation on the object */
```

```
        abstract public SpokeObject Operation(int Op, SpokeObject
object);

        /* Used in Partition and Get Element to check if is List */
        abstract public boolean IsList();

        /* Used as the easy Operation to Partition the List */
        abstract public SpokeObject Partition(SpokeObject start,
SpokeObject end);

        /* Used as the easy Operation to Get Element in the list */
        abstract public void SetElement(SpokeObject index, SpokeObject
object);

        /* Used as the easy Operation to Get Element in the list */
        abstract public SpokeObject GetElement(SpokeObject index);
   }
```

## SpokeString.java

```
   ------------
   package org.spoke;

   public class SpokeString extends SpokeObject {
       private String myObject;

       private void setObject(Object object) {
           if (object instanceof SpokeString)
               setString(((SpokeString)object).getString());

           else if (object instanceof String)
               setString((String)object);

           else
               throw new SpokeException("type does not match");
       }

       public SpokeString(Object object) {
           setObject(object);
       }

       public void setString(String object) {
           myObject = new String(object);
       }

       public String getString() {
           return myObject;
       }
```

```
           @Override
           public boolean IsTrue() {
                   return myObject.length() > 0;
           }

           @Override
           public boolean IsCompatible(SpokeObject object) {
                   return object instanceof SpokeString;
           }

           @Override
           public void Assign(SpokeObject object) {
           if (object == null || object instanceof SpokeNull)
               throw new SpokeException("operation with null");

           if (object instanceof SpokeAny)
               throw new SpokeException("operation with any");

                   setObject(object);
           }

       @Override
       public SpokeObject Clone() {
           return new SpokeString(this);
       }

           @Override
           public boolean IsBoolean() {
                   return false;
           }

           @Override
       public boolean IsComparable(SpokeObject object) {
                   return object instanceof SpokeString ||
                          object instanceof SpokeAny || object
instanceof SpokeNull;
       }

           @Override
           public boolean IsEnumerated() {
                   return false;
           }

           @Override
           public boolean MayHave(SpokeObject object) {
                   return object instanceof SpokeString || object
instanceof SpokeAny;
           }

       @Override
       public SpokeObject Operation(int Op, SpokeObject object) {
            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(false);

            if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                    return new SpokeBoolean(true);

            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny && Op == SpokeObject.OpHas)
                    return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
                return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
                return new SpokeBoolean(false);

            if (!(object instanceof SpokeString))
                throw new SpokeException("type does not match");

            String string = ((SpokeString)object).getString();

            switch(Op) {
            /* Arithmetic Operators */
            case SpokeObject.OpAdd:
                    return new SpokeString(myObject.concat(string));
            /* Comparison Operators */
            case SpokeObject.OpEq:
                    return new SpokeBoolean(myObject.equals(string));
            case SpokeObject.OpNeq:
                    return new SpokeBoolean(!myObject.equals(string));
            case SpokeObject.OpHas:
                int index = myObject.lastIndexOf(string);
                if (index >= 0) {
                    SpokeProgram.AddMatch(object);
                    return new SpokeBoolean(true);
                }
                else
                    return new SpokeBoolean(false);
            /* Unsupported Operators */
            default:
                    throw new SpokeException("operation is not
supported");
            }
        }

            @Override
            public boolean IsList() {
                    return true;
            }
```

```
	@Override
	public SpokeObject Partition(SpokeObject start, SpokeObject end)
{

	    if (!(start instanceof SpokeInteger))
	        throw new SpokeException("type does not match");

	    if (!(end instanceof SpokeInteger))
	            throw new SpokeException("type does not match");

	    int sindex = ((SpokeInteger)start).getInteger();
	    int eindex = ((SpokeInteger)end).getInteger();

	    byte[] chars = myObject.getBytes();

	    if (sindex < 0) {
	            sindex = chars.length + sindex;
	    }
	    if (eindex < 0) {
	            eindex = chars.length + eindex;
	    }
	    if (sindex >= chars.length) {
	            throw new SpokeException("List boundary is
violated");
	    }
	    if (eindex >= chars.length) {
	            throw new SpokeException("List boundary is
violated");
	    }
	    if (sindex > eindex) {
	            return new SpokeString("");
	    }

	    return new SpokeString(new String(chars, sindex, eindex -
sindex + 1));
	}

	@Override
	public void SetElement(SpokeObject index, SpokeObject object) {
	    if (!(index instanceof SpokeInteger))
	        throw new SpokeException("type does not match");

	    if (!(object instanceof SpokeString))
	        throw new SpokeException("type does not match");

	    int eindex = ((SpokeInteger)index).getInteger();
	    String str = ((SpokeString)object).getString();

	    byte[] chars = myObject.getBytes();

	    if (eindex < 0) {
	            eindex = chars.length + eindex;
```

```
          }
          if (eindex >= chars.length) {
                throw new SpokeException("array boundary is
violated");
          }

          setString(new String(chars, 0, eindex - 1) + str +
                            new String(chars, eindex, chars.length -
eindex));
      }

      @Override
      public SpokeObject GetElement(SpokeObject index) {
          if (!(index instanceof SpokeInteger))
              throw new SpokeException("type does not match");

          int eindex = ((SpokeInteger)index).getInteger();

          byte[] chars = myObject.getBytes();

          if (eindex < 0) {
                eindex = chars.length + eindex;
          }
          if (eindex >= chars.length) {
                throw new SpokeException("array boundary is
violated");
          }

          return new SpokeString(new String(chars, eindex, 1));
      }

          @Override
          public String toString() {
                return "\"" + myObject + "\"";
          }
   }
```

**SpokeTag.java**

```
   ------------
   package org.spoke;

   public class SpokeTag extends SpokeObject {
          private SpokeObject myTag;
          private SpokeObject myObject;

          public SpokeTag(SpokeObject tag, SpokeObject object) {
                setTag(tag);
                setObject(object);
```

```
    }

public SpokeTag(Object object) {
    if (object instanceof SpokeTag) {
        setTag(((SpokeTag)object).getTag());
            setObject(((SpokeTag)object).getObject());
    }

    else
        throw new SpokeException("type does not match");
}

public void setTag(SpokeObject tag) {
    myTag = tag;
}

    public SpokeObject getTag() {
            return myTag;
    }

public void setObject(SpokeObject object) {
    myObject = object.Clone();
    }

public SpokeObject getObject() {
    return myObject;
    }

    @Override
    public boolean IsTrue() {
            return myTag.IsTrue() && myObject.IsTrue();
    }

    @Override
    public boolean IsCompatible(SpokeObject object) {
            return object instanceof SpokeTag;
    }

@Override
    public void Assign(SpokeObject object) {
    if (object == null || object instanceof SpokeNull)
        throw new SpokeException("operation with null");

    if (object instanceof SpokeAny)
        throw new SpokeException("operation with any");

    if (!(object instanceof SpokeTag))
        throw new SpokeException("type does not match");

    setTag(((SpokeTag)object).getTag());
            setObject(((SpokeTag)object).getObject());
    }
```

```
        @Override
        public SpokeObject Clone() {
            return new SpokeTag(this);
        }

            @Override
            public boolean IsBoolean() {
                    return false;
            }

            @Override
            public boolean IsComparable(SpokeObject object) {
                    return object instanceof SpokeTag ||
                           object instanceof SpokeAny || object
instanceof SpokeNull;
            }

            @Override
            public boolean IsEnumerated() {
                    return false;
            }

            @Override
            public boolean MayHave(SpokeObject object) {
                    return object instanceof SpokeTag || object
instanceof SpokeList ||
                           object instanceof SpokeString;
            }

        public SpokeObject Operation(int Op, SpokeObject object) {
            if (object instanceof SpokeNull && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(false);

            if (object instanceof SpokeNull && Op == SpokeObject.OpNeq)
                    return new SpokeBoolean(true);

            if (object == null || object instanceof SpokeNull)
                throw new SpokeException("operation with null");

            if (object instanceof SpokeAny && Op == SpokeObject.OpHas) {
                    SpokeProgram.AddMatch(this);
                    return new SpokeBoolean(true);
            }

            if (object instanceof SpokeAny && Op == SpokeObject.OpEq)
                    return new SpokeBoolean(true);

            if (object instanceof SpokeAny && Op == SpokeObject.OpNeq)
                return new SpokeBoolean(false);
            if (!(object instanceof SpokeTag) && Op == SpokeObject.OpHas)
{
                    if (myObject.MayHave(object))
                            return myObject.Operation(OpHas, object);
                    else
                            return new SpokeBoolean(false);
            }

            if (!(object instanceof SpokeTag))
                throw new SpokeException("type does not match");

            SpokeObject Tag = ((SpokeTag)object).getTag();
            SpokeObject Obj = ((SpokeTag)object).getObject();

            switch(Op) {
            /* Arithmetic Operators */
            case SpokeObject.OpAdd:
                    if (myObject instanceof SpokeList) {
                            SpokeObject[] List =
((SpokeList)myObject).getList();
                            SpokeObject[] newList = new
SpokeList[List.length + 1];
                            for (int i = 0 ; i < List.length ; i++)
                                    newList[i] = List[i];
                            newList[List.length] = object;
                            return new SpokeTag(myTag, new
SpokeList(newList));
                    }
                    else {
                            SpokeObject[] newList = new SpokeObject[2];
                            newList[0] = this;
                            newList[1] = object;
                            return new SpokeTag(new SpokeAny(), new
SpokeList(newList));
                    }
            /* Comparison Operators */
            case SpokeObject.OpEq:
                    if (myTag instanceof SpokeString && Tag instanceof
SpokeString) {
                            String myTagStr =
((SpokeString)myTag).getString();
                            String TagStr =
((SpokeString)Tag).getString();
                            return new
SpokeBoolean(myTagStr.startsWith(TagStr) &&

myObject.IsComparable(Obj) &&
                                                    myObject.Operation(OpEq,
Obj).IsTrue());
                    }
                    return new SpokeBoolean(myTag.IsComparable(Tag) &&
                                                        myTag.Operation(OpEq,
Tag).IsTrue() &&

myObject.IsComparable(Obj) &&
                                                   myObject.Operation(OpEq,
Obj).IsTrue());
            case SpokeObject.OpNeq:
                    return new SpokeBoolean(!(Operation(OpEq,
object).IsTrue()));
            case SpokeObject.OpHas:
                    boolean hasObject = false;

                    if (Operation(OpEq, object).IsTrue()) {
                            SpokeProgram.AddMatch(this);
                            hasObject = true;
                    }

                    if (myObject.MayHave(object) &&
                        myObject.Operation(OpHas, object).IsTrue())
                            hasObject = true;

                    return new SpokeBoolean(hasObject);
                /* Unsupported Operators */
                default:
                    throw new SpokeException("operation is not
supported");
            }
            }

            @Override
            public boolean IsList() {
            return myObject.IsList();
            }

            @Override
            public SpokeObject Partition(SpokeObject start, SpokeObject
end) {
                    return myObject.Partition(start, end);
            }

            @Override
            public void SetElement(SpokeObject index, SpokeObject object)
{
                    myObject.SetElement(index, object);
            }

            @Override
            public SpokeObject GetElement(SpokeObject index) {
                    return myObject.GetElement(index);
            }

            @Override
            public String toString() {
                    return myTag.toString() + "(" + myObject.toString()
+ ")";
            }
    }
```

**SpokeProgram.java**

```
    ------------
    package org.spoke;

    public abstract class SpokeProgram {

            /* global variable 'match' */
            protected static SpokeList g_0 = new SpokeList();
            /* global variable 'star'  */
            protected static SpokeList g_1 = new SpokeList();

            public static void AddMatch(SpokeObject object) {
                    g_0.addObject(object);
            }

            public static void FlushMatch() {
                    g_0 = new SpokeList();
            }

            public static void AddStar(SpokeObject object) {
                    g_1.addObject(object);
            }

            public static void FlushStar() {
                    g_1 = new SpokeList();
            }

        public static void Execute(SpokeProgram program, String[] args)
{
            SpokeObject[] args_list = new SpokeObject[args.length];
            for (int i = 0 ; i < args.length ; i++) {
                args_list[i] = new SpokeString(args[i]);
            }
            program.Init(new SpokeList(args_list));
        }

        abstract public void Init(SpokeObject args);
    }
```

**SpokeRuntimeException.java**

```
   ------------
   package org.spoke;

   public class SpokeRuntimeException extends RuntimeException {
           private static final long serialVersionUID = -
3077970640526088070L;

           public SpokeRuntimeException(String message) {
           super(message);
       }
   }
```

**Makefile**

```
   ------------
   JFLAGS = -g
   JC = javac

   default: backend

   BACKEND_SRC = SpokeAny.java SpokeApi.java SpokeBoolean.java
SpokeException.java SpokeFloat.java SpokeFile.java SpokeInteger.java
SpokeList.java SpokeNull.java SpokeObject.java SpokeProgram.java
SpokeRuntimeException.java SpokeString.java SpokeTag.java
NLPparser.java NLPtagger.java

   backend:
           $(JC) -classpath ../../../lib/stanford-
parser.jar:../../../lib/stanford-postagger.jar: $(BACKEND_SRC)

   clean:
           $(RM) *.class *~
```

**NLPtagger.java**

```
   ------------
   package org.spoke;

   import edu.stanford.nlp.ling.Sentence;
   import edu.stanford.nlp.tagger.maxent.MaxentTagger;

   public final class NLPtagger {

           static public SpokeObject nlpTag(SpokeObject obj) {
                   SpokeList args = (SpokeList)obj;
```

```
                    SpokeApi.check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    if (!(object instanceof SpokeString))
                            throw new SpokeException("Type does not
match");

                    if(((SpokeString)object).getString().equals(""))
                            throw new SpokeException("The String is
empty.");

                    SpokeObject[] tagged_list = null;
                    String raw = ((SpokeString)object).getString();

                    MaxentTagger tagger;
                    try {
                            tagger = new
MaxentTagger("backend/model/left3words-wsj-0-18.tagger");
                    } catch (Exception e) {
                            throw new SpokeException("MaxentTagger
initialization failed");
                    }

                    String tagged =
tagger.tagSentence(Sentence.toSentence(raw.split("\\s+"))).toString();
                    tagged = tagged.replaceAll("\\[", "");
                    tagged = tagged.replaceAll("\\]", "");
                    String [] tags = tagged.split(", ");
                    tagged_list = new SpokeObject[tags.length];
                    for (int i = 0; i < tags.length; i++) {
                            int split = tags[i].lastIndexOf("/");
                            String word = tags[i].substring(0, split);
                            String pos = tags[i].substring(split+1,
tags[i].length());
                            SpokeTag ST = new SpokeTag(new
SpokeString(pos), new SpokeString(word));
                            tagged_list[i] = ST;
                    }

                    return new SpokeList(tagged_list);
            }
   }
```

## NLPparser.java

```
   ------------
   package org.spoke;
   import java.util.*;

   import edu.stanford.nlp.trees.*;
   import edu.stanford.nlp.parser.lexparser.LexicalizedParser;

   public final class NLPparser {

           static public SpokeObject nlpParse(SpokeObject obj) {
           SpokeList args = (SpokeList)obj;

           SpokeApi.check_args_num(args, 1);

           SpokeObject object = args.getList()[0];

           if (!(object instanceof SpokeString))
               throw new SpokeException("Type does not match");

                   LexicalizedParser lp = new
LexicalizedParser("backend/model/englishPCFG.ser.gz");
               lp.setOptionFlags(new String[]{"-maxLength", "80", "-
retainTmpSubcategories"});

               if(((SpokeString)object).getString().equals(""))
                   throw new SpokeException("The String is empty.");

               String[] sent =
((SpokeString)object).getString().split(" ");
               Tree parse = (Tree)lp.apply(Arrays.asList(sent));

               Object curObj = null;
               edu.stanford.nlp.trees.LabeledScoredTreeNode curNode =
null;

               Iterator<Tree> iterator = parse.iterator();
               curObj = iterator.next();
               curNode =
(edu.stanford.nlp.trees.LabeledScoredTreeNode)curObj;

               return addNode(curNode);
           }

           static public SpokeTag addNode (Tree curNode){
                   SpokeObject Tag = null;
                   SpokeObject Obj = null;

                   if(curNode.value().equals(""))
                       throw new SpokeException("Error occurs during
parsing.");
                   else {
                           Tag = new SpokeString(curNode.value());
                   }
```

```
                        Tree[] children = curNode.children();

                        if (children.length == 0) {
                                throw new SpokeException("Wierd Tree");
                        }

                        if(children[0] instanceof
edu.stanford.nlp.trees.LabeledScoredTreeLeaf){
                                if (children.length > 1)
                                        throw new SpokeException("Wierd
Tree");

                                Obj = new SpokeString(children[0].value());
                        }
                        else{
                                SpokeObject[] siblings = new
SpokeObject[children.length];
                                for(int i = 0 ; i < children.length ; i++){
                                        if(children[i] instanceof
edu.stanford.nlp.trees.LabeledScoredTreeNode){
                                                siblings[i] = new
SpokeTag(addNode(children[i]));
                                        }
                                }
                                Obj = new SpokeList(siblings);
                        }

                        return new SpokeTag(Tag, Obj);
                }

                public static String remove_score (String raw){
                        String processed = raw.replaceAll("[.*0-9]","");
                        processed = processed.replaceAll("\\[", "");
                        processed = processed.replaceAll("\\]", "");
                        processed = processed.replaceAll("\\s+", " ");
                        return processed;
                }

                public static String remove_tag (String raw){
                        String processed = raw.replaceAll("\\)", "");
                        processed = processed.replaceAll("\\([A-Z!,.?:;' ]+",
"");
                        return processed;
                }

    }

    src/org/spoke/api:
```

## SpokeApi.java

```
   ------------
   package org.spoke;

   public final class SpokeApi {

           static public void check_args_num(SpokeList args, int num) {
                   if (args.getList().length < num)
                           throw new SpokeException("Too few
arguments");

                   if (args.getList().length > num)
                           throw new SpokeException("Too many
arguments");
           }

           static public int check_args_num(SpokeList args, int num1,
int num2) {
                   if (args.getList().length < num1)
                           throw new SpokeException("Too few
arguments");

                   if (num2 > 0 && args.getList().length > num2)
                           throw new SpokeException("Too many
arguments");

                   return args.getList().length;
           }

           static private float float_value(SpokeObject object) {
                   if (object instanceof SpokeInteger)
                           return ((SpokeInteger)object).getInteger();
                   else if (object instanceof SpokeFloat)
                           return (int)((SpokeFloat)object).getFloat();

                   throw new SpokeException("Type does not match");
           }

           static public SpokeObject SpokeCeil(SpokeObject obj) {
           SpokeList args = (SpokeList)obj;

           check_args_num(args, 1);

                   SpokeObject object = args.getList()[0];

                   return new
SpokeInteger(Math.ceil(float_value(object)));
           }


           static public SpokeObject SpokeFloor(SpokeObject obj) {
           SpokeList args = (SpokeList)obj;
```

```
            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    return new
SpokeInteger(Math.floor(float_value(object)));
            }

            static public SpokeObject SpokePow(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

                    check_args_num(args, 2);

                    SpokeObject obj0 = args.getList()[0];
                    SpokeObject obj1 = args.getList()[1];

                    return new
SpokeFloat(Math.pow(float_value(obj0),float_value(obj1)));
            }

            static public SpokeObject SpokeSqrt(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    return new
SpokeFloat(Math.sqrt(float_value(object)));
            }

            static public SpokeObject SpokeLog(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    return new SpokeFloat(Math.log(float_value(object)));
            }

            static public SpokeObject SpokeAbs(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    return new SpokeFloat(Math.abs(float_value(object)));
            }

            static public SpokeObject SpokeExp(SpokeObject obj) {
```

```
        SpokeList args = (SpokeList)obj;

        check_args_num(args, 1);

                SpokeObject object = args.getList()[0];

                return new SpokeFloat(Math.exp(float_value(object)));
        }

        static public SpokeObject SpokeRange(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

        int args_num = check_args_num(args, 1, 3);

        int sindex = 0;
        int eindex = 0;
        int inc = 1;

        switch (args_num) {
        case 1: {
                SpokeObject object = args.getList()[0];
                if (!(object instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");

                eindex = ((SpokeInteger)object).getInteger();
                break; }
        case 2: {
                SpokeObject obj0 = args.getList()[0];
                if (!(obj0 instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");

                SpokeObject obj1 = args.getList()[1];
                if (!(obj1 instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");

                sindex = ((SpokeInteger)obj0).getInteger();
                eindex = ((SpokeInteger)obj1).getInteger();
                break; }
        case 3: {
                SpokeObject obj0 = args.getList()[0];
                if (!(obj0 instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");

                SpokeObject obj1 = args.getList()[1];
                if (!(obj1 instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");
```

```
                SpokeObject obj2 = args.getList()[2];
                if (!(obj1 instanceof SpokeInteger))
                        throw new SpokeException("Type does not
match");

                sindex = ((SpokeInteger)obj0).getInteger();
                eindex = ((SpokeInteger)obj1).getInteger();
                inc = ((SpokeInteger)obj2).getInteger();
                break; }
        }

        if (eindex <= sindex) return new SpokeList();

        SpokeObject[] List = new SpokeObject[(eindex - sindex) /
inc];
        for (int i = sindex, j = 0 ; i < eindex ; i += inc, j++) {
                List[j] = new SpokeInteger(i);
        }
        return new SpokeList(List);
        }

        static public SpokeObject SpokeLen(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

        check_args_num(args, 1);

                SpokeObject object = args.getList()[0];

                if (object instanceof SpokeList)
                        return new
SpokeInteger(((SpokeList)object).getList().length);

        if (object instanceof SpokeString)
                        return new
SpokeInteger(((SpokeString)object).getString().length());

        if (object instanceof SpokeTag) {
                SpokeTag Tag = (SpokeTag)object;
                if (Tag.getObject() instanceof SpokeString)
                        return new SpokeInteger(0);
                if (Tag.getObject() instanceof SpokeTag)
                        return new SpokeInteger(1);
                if (Tag.getObject() instanceof SpokeList)
                        return new
SpokeInteger(((SpokeList)Tag.getObject()).getList().length);

              throw new SpokeException("Wierd Tree");
        }

                throw new SpokeException("Type does not match");
        }
        static private String str(SpokeObject obj) {
                if (obj instanceof SpokeString)
                        return ((SpokeString)obj).getString();
                if (obj instanceof SpokeInteger)
                        return
Integer.toString(((SpokeInteger)obj).getInteger());
                if (obj instanceof SpokeFloat)
                        return
Double.toString(((SpokeFloat)obj).getFloat());
                if (obj instanceof SpokeBoolean)
                        return ((SpokeBoolean)obj).getBoolean() ?
"True" : "False";
                if (obj instanceof SpokeAny)
                        return "Any";
                if (obj instanceof SpokeNull)
                        return "Null";
                if (obj instanceof SpokeList) {
                        SpokeObject[] List =
((SpokeList)obj).getList();
                        String myStr = "";
                        for (int i = 0 ; i < List.length ; i++) {
                            myStr += str(List[i]) + (i < List.length
- 1 ? " " : "");
                        }
                        return myStr;
                }
                if (obj instanceof SpokeTag)
                        return str(((SpokeTag)obj).getObject());

                throw new SpokeException("Type does not match");
        }

        static public SpokeObject SpokeStr(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

        check_args_num(args, 1);

                SpokeObject object = args.getList()[0];

                return new SpokeString(str(object));
        }

        static public SpokeObject SpokeDebug(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

        check_args_num(args, 1);

                SpokeObject object = args.getList()[0];

                return new SpokeString(object.toString());
        }
```

```
            static public SpokeObject SpokeTag(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

            if (object instanceof SpokeTag)
                return new SpokeString(str(((SpokeTag)object).getTag()));

            throw new SpokeException("Type does not match");
            }

            static public SpokeObject SpokeObj(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

            if (object instanceof SpokeTag)
                return ((SpokeTag)object).getObject().Clone();
            else
                    return object.Clone();
            }

            static public SpokeObject SpokeInt(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    if (object instanceof SpokeInteger || object
instanceof SpokeFloat)
                            return new SpokeInteger(object);
                    else if (object instanceof SpokeString)
                            return new
SpokeInteger(Integer.parseInt(((SpokeString)object).getString()));

                    throw new SpokeException("Type does not match");
            }

            static public SpokeObject SpokeFloat(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

                    SpokeObject object = args.getList()[0];

                    if (object instanceof SpokeInteger || object
instanceof SpokeFloat)
```

```
                        return new SpokeFloat(object);
                else if (object instanceof SpokeString)
                        return new
SpokeFloat(Float.parseFloat(((SpokeString)object).getString()));

                throw new SpokeException("Type does not match");
        }

        static public SpokeObject SpokePrint(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

                System.out.print(str(args));
                return new SpokeNull();
        }

      static public SpokeObject SpokeOpen(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

        check_args_num(args, 2);

        SpokeObject obj0 = args.getList()[0];
        SpokeObject obj1 = args.getList()[1];

        if (!(obj0 instanceof SpokeString) || !(obj1 instanceof
SpokeString))
              throw new SpokeException("Type does not match");

        String filename = ((SpokeString)obj0).getString();
        String filetype = ((SpokeString)obj1).getString();

        if (filetype.equals("r")) {
                try {
                        return new SpokeFile((new
java.io.FileInputStream(filename)).getFD());
                } catch (java.io.IOException e) {
                        return new SpokeNull();
                }
        }

        if (filetype.equals("w")) {
                try {
                        return new SpokeFile((new
java.io.FileOutputStream(filename, false)).getFD());
                } catch (java.io.IOException e) {
                        return new SpokeNull();
                }
        }

        if (filetype.equals("a")) {
                try {
                        return new SpokeFile((new
java.io.FileOutputStream(filename, true)).getFD());
```

```
                    } catch (java.io.IOException e) {
                            return new SpokeNull();
                    }
            }

            throw new SpokeException("Unknown file type");
        }

        static public SpokeObject SpokeClose(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

            check_args_num(args, 1);

            SpokeObject object = args.getList()[0];

            if (!(object instanceof SpokeFile))
                throw new SpokeException("Type does not match");

            SpokeFile file = (SpokeFile)object;

            if (file.CanRead()) {
                    try {
                            (new
java.io.FileInputStream(file.getFD())).close();
                    } catch (java.io.IOException e) {
                            throw new SpokeException("IO error");
                    }
                            return new SpokeNull();
            }

            if (file.CanWrite()) {
                    try {
                            (new
java.io.FileOutputStream(file.getFD())).close();
                    } catch (java.io.IOException e) {
                            throw new SpokeException("IO error");
                    }
                            return new SpokeNull();
            }

            throw new SpokeException("IO error");
        }

            static public SpokeObject SpokeRead(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

                    int args_num = check_args_num(args, 0, 1);

                    java.io.FileInputStream input;

                    if (args_num == 0) {
                            input = new
 java.io.FileInputStream(java.io.FileDescriptor.in);
                    }
                    else {
                    SpokeObject object = args.getList()[0];

                    if (!(object instanceof SpokeFile))
                        throw new SpokeException("Type does not match");

                    SpokeFile file = (SpokeFile)object;

                    if (!file.CanRead())
                            throw new SpokeException("File is not
 readable");

                    input = new java.io.FileInputStream(file.getFD());
                    }

                    try {
                            String readstr = "";

                            while (true) {
                                    char readchar = (char)input.read();
                                    if (readchar == ' ') break;

                                    readstr += readchar;
                            }
                            return new SpokeString(readstr);
                    } catch (java.io.IOException ex) {
                            throw new SpokeException("IOError");
                    }
            }

            static public SpokeObject SpokeReadline(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

                    int args_num = check_args_num(args, 0, 1);

                    java.io.FileInputStream input;

                    if (args_num == 0) {
                            input = new
 java.io.FileInputStream(java.io.FileDescriptor.in);
                    }
                    else {
                    SpokeObject object = args.getList()[0];

                    if (!(object instanceof SpokeFile))
                        throw new SpokeException("Type does not match");

                    SpokeFile file = (SpokeFile)object;
```

```
                    if (!file.CanRead())
                            throw new SpokeException("File is not
readable");

                    input = new java.io.FileInputStream(file.getFD());
                    }

                    try {
                            String readstr = "";

                            while (true) {
                                    char readchar = (char)input.read();
                                    if (readchar == '\n') break;

                                    readstr += readchar;
                            }
                            return new SpokeString(readstr);
                    } catch (java.io.IOException ex) {
                            throw new SpokeException("IO error");
                    }
            }

            static public SpokeObject SpokeWrite(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;

                    check_args_num(args, 1, 0);

            SpokeObject object = args.getList()[0];

                if (!(object instanceof SpokeFile))
                    throw new SpokeException("Type does not match");

                SpokeFile file = (SpokeFile)object;

                if (!file.CanWrite())
                    throw new SpokeException("File is not writeable");

                java.io.FileOutputStream output = new
java.io.FileOutputStream(file.getFD());

                    try {
                            output.write(str(args.Partition(new
SpokeInteger(1), new SpokeInteger(-1)))
                                                    .getBytes());
                            return new SpokeNull();
                    } catch (java.io.IOException ex) {
                            throw new SpokeException("IO error");
                    }
            }

            static public SpokeObject SpokeSystem(SpokeObject obj) {
            SpokeList args = (SpokeList)obj;
```

```
                check_args_num(args, 1);

        SpokeObject object = args.getList()[0];

            if (!(object instanceof SpokeString))
                throw new SpokeException("Type does not match");

            Process proc;
            try {
                proc =
Runtime.getRuntime().exec(((SpokeString)object).getString());
            } catch (java.io.IOException ex) {
                throw new SpokeException("IO error");
            }

            try {
                proc.wait();
            } catch (Exception e) {
            }

            try {
                return new SpokeInteger(proc.exitValue());
            } catch (Exception e) {
                return new SpokeInteger(255);
            }
        }

        static public SpokeObject SpokePipe(SpokeObject obj) {
        SpokeList args = (SpokeList)obj;

                check_args_num(args, 1);

        SpokeObject object = args.getList()[0];

            if (!(object instanceof SpokeString))
                throw new SpokeException("Type does not match");

            Process proc;
            try {
                proc =
Runtime.getRuntime().exec(((SpokeString)object).getString());
            } catch (java.io.IOException ex) {
                throw new SpokeException("IO error");
            }

            java.io.BufferedInputStream input =
                new
java.io.BufferedInputStream(proc.getInputStream());

            try {
                proc.wait();
            } catch (Exception e) {
            }

            try {
                byte[] buffer = new byte[input.available()];
                input.read(buffer);
                return new SpokeString(new String(buffer));
            } catch (java.io.IOException e) {
                throw new SpokeException("IO error");
            }
        }
}
```

## References

**R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, & V. Zue (eds.),** Discourse and Dialogueue. Grosz, Barbara; Scott, Donia; Kamp, Hans; Cohen, Phil; Giachin, Egidio. Chapter 6 of Survey of the State of the Art in Human Language Technology, Cambridge University Press, 1995.
**Pieraccini R. and Huerta, Juan**, "Where do we go from here? research and commercial spoken dialogue systems", In SIGdial-2005, 1-10.