Inside the Java Intelligent Tutoring System Prototype: Parsing Student Code Submissions with Intent Recognition

Edward R. Sykes
School of Applied Computing and Engineering Sciences, Sheridan College
1430 Trafalgar Road, Oakville, Ont., Canada, L6H 2L1
+1 (905) 845-9430 Ext. 2490
ed.sykes@sheridanc.on.ca

Franya Franek
Department of Computing and Software, Faculty of Science, McMaster University
1280 Main Street W., Hamilton, Ont., Canada, L8S 4L8
+1 (905) 525-9140 Ext. 23233
franek@mcmaster.ca

ABSTRACT

The "Java™ Intelligent Tutoring System" (JITS) research project involves the development of a programming tutor designed for students in their first programming course in Java™ at the College and University level. This paper describes recent progress on the work presented at the last WBE IASTED conference. The previous paper, entitled "A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java™", presented an overview of the architectural design including state-of-the-art web-based distributed architecture, the AI techniques used, and the programmer-optimized user interface. This paper delves further into the mechanism of the Java™ Tutor which is responsible for the syntax and semantic analysis of the code that the student submits for a programming problem. The ultimate goal of this inner-component of JITS is to understand the 'intent' of the student by carefully analyzing the student's code.

KEY WORDS

Web-Based Education, Programming Tutors, e-Learning, Intelligent Tutoring Systems.

1. Introduction

Based on Cognitive Science and Artificial Intelligence (AI), Intelligent Tutoring Systems have proven their worth in domains including Physics, Mathematics, Language Development, and many other disciplines [1, 2].

Currently, the demand for ITS is growing at an amazing rate [3]. ITS are gaining such strong acceptance and popularity due to the following reasons: i) higher student performance, ii) deepened cognitive development, and iii) reduced acquisition time for the student [1, 2, 4].

The current research goal is to bring together recent developments in the fields of Intelligent Tutoring Systems, Cognitive Science, and AI to construct an effective intelligent tutor to help students learn to program in Java™. In addition to contributing to the understanding of programming learning processes, it is hoped that this research will have a positive impact on professors teaching Java™. This research is significant since there are a growing number of students wishing to learn programming despite the fact that personalized instruction is decreasing [3, 5]. Additionally, since there are a growing number of institutions investing in e-learning, this research will play a significant role in providing appropriate methods of teaching this key subject to students learning remotely.

2. JITS Model Overview

This section presents the key components of intent recognition within the Java™ Intelligent Tutoring System. JITS is designed with two distinct mechanisms of functionality:

2.1. A-Type JITS Functionality

In many Intelligent Tutoring Systems, the process of authoring involves a professor to provide a set of problems, their specifications, and corresponding solutions. In JITS, this type of functionality is provided for very straight-forward programming problems.

The 'A'-type functionality is solved by a DPA edit-distance algorithm. See figure 1 to see a pictorial representation of how JITS performs pattern-matching to produce modified string of the student's code. This topic is discussed in detail in the extended version of *"A Prototype for an Intelligent Tutoring System for Students*

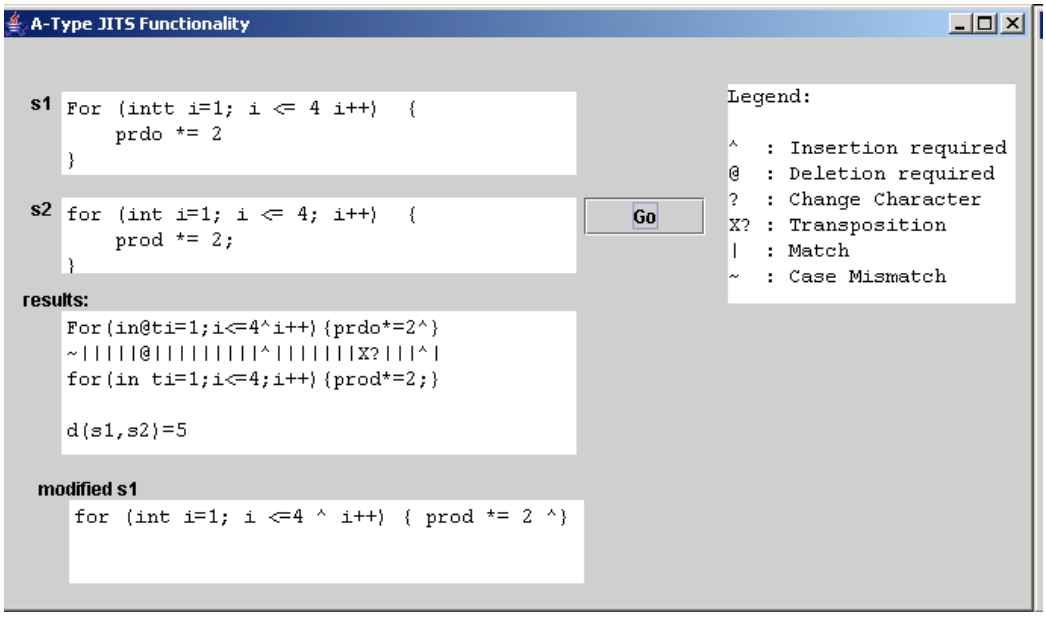

Figure 1. 'A'-Type JITS Functionality

Learning to Program in Java™" found in Special Issue of the IJCA Journal 2004.

2.2. B-Type JITS Functionality

The second mechanism of functionality that JITS provides is a consequence of the limitations from 'A'-type functionality described above. In many programming problems there are often many solutions. A professor may provide one solution to a problem but there may be many other solutions that are equally as suitable. As a result, the most reasonable approach is to request the professor to author only the problem, the problem specification, and the output (i.e., desired results) – JITS needs to determine the rest.

The B-type functionality requires much more rigor in terms of attempting to ascertain the 'intent' of the student by analyzing the code. The difficulty in these types of problems is that there is no coded solution from which JITS can use as a comparison. As a result, a specialized intent recognition scanner-parser algorithm prototype has been developed as a means of determining the intent behind the student's submission. This algorithm is described in greater detail in the following sections.

3. JITS Overview and Framework

This section describes JITS framework from a high-level perspective. Figure 2 presents a flowchart of how JITS processes the student's submission. Particular emphasis is placed on the Intent Recognition (IR) module. Due to the complexity involved with scanning and parsing

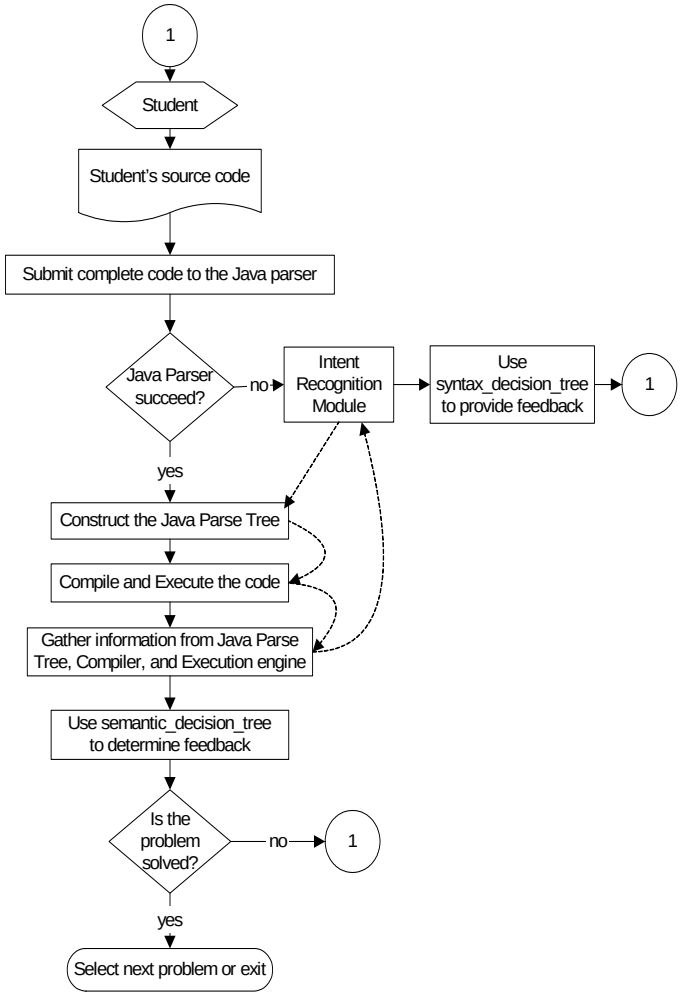

Figure 2. Flowchart of JITS AI Module

Java™ code, it is necessary to restrict JITS to tutor a small subset of the Java™ programming language. The current area of research focuses on basic Java™ constructs including variables, operators, and looping structures.

To better understand the issues associated with the IR module an example is presented in Table 1. Although the problem is trivial, it provides a suitable means of illustrating the intent recognition scanner-parser (i.e., B-Type JITS functionality).

Table 1. Java™ ITS Example Problem

Problem:
Write a program called "Exponentiation" which calculates 2^N, where N is a user specified number. For example, if N were assigned the value 4, then the result would be 2^4 = 2x2x2x2 = 16.

Program specifications:
This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

```
OUTPUT>Result: 16
```

Skeleton Program (located in Source Code area):

```
public class Exponentiation
 public static void main(String[]

    int prod = 1;

    /* student writes code here

    System.out.println("Result:" + prod);
}
```

Solution (one of many):

```
public class Exponentiation {
 public static void main(String[] args) {
    int prod = 1;
    for (int i = 1; i <= 4; i++) {
      prod *= 2;
    }
    System.out.println("Result: " + prod);
  }
}
```

There are limitless possibilities for student responses and the system cannot simply list incorrect responses coupled with feedback messages. JITS is designed to be pedagogically sound and focuses on the methodology by which a student attempts to solve a problem. Programming conventions, style, and professional coding techniques are modeled in JITS. In this fashion effective tutoring may take place.

In order for JITS to provide intelligent feedback to the student the Intent Recognition Module relies on a collection of information: the problem statement, the problem specification, student's code, the established student model, the expert model, the Java™ parser, the syntax decision tree, the semantic decision tree, the Java™ Parse Tree, the output from the Java™ compiler, and the result from the Java™ runtime engine [6]. Based on the context some of this information will not be available. However, the goal of this module is to carefully scrutinize all available information so that appropriate feedback may be generated for the student.

4. JITS Intent Recognition Module

The purpose of the Intent Recognition (IR) module is to ascertain the most probable submission of code the student intended. As identified in Figure 2, the IR is invoked when the standard Java™ parser fails. The IR's responsibility is to systematically employ a minimum distance error-correcting scanner-parser algorithm with the goal of remedying the student's code [7, 8]. The IR algorithm is explained in the following section.

4.1. Syntax Error Correction Strategy

Let L be a nonempty set of strings over the finite set of symbols used in the Java programming language (i.e., Σ). It is assumed that a string not in L may be derived from some sentence in L by a sequence of error-transformations. The IR module recognizes four types of syntax errors:

i) the replacement of a symbol by another symbol,
ii) the insertion of an extraneous symbol,
iii) the deletion of a symbol, and
iv) the transposition of two adjacent symbols.

These four errors can be represented by four transformations T_R , T_I , T_D , and T_S from Σ' to the subsets of Σ' defined as follows. For x and y in Σ':

i) xby is in $T_R(xay)$ for all $a \neq b$
ii) xay is in $T_I(xy)$ for all $a \in \Sigma$
iii) xy is in $T_D(xay)$ for all $a \in \Sigma$
iv) xy is in $T_S(yx)$

The goal of the IR is to select a sequence of intermediate strings and error transformations such that the result is a transformation sequence that produces an acceptable token for the parser.

For example, suppose $L=\{cde\}$. Given a string *ddfe*, the first '*d*' is a replacement error and the '*f*' is an insertion error because:

$$cde \xrightarrow[T_R]{} dde \xrightarrow[T_I]{} ddfe$$

Using Java™ for another example, consider the following declaration:

```
publik status flot TAX=5;
```

The IR would construct the following Transformation Sequence:

$publik \xrightarrow[T_R]{} public$

$status \xrightarrow[T_R]{} statis \xrightarrow[T_R]{} static$

$flot \xrightarrow[T_I]{} float$, resulting in the correct syntax:

```
public static float TAX=5;
```

4.2. IR Scanner-Parser Algorithm

This section describes the Intent Recognition Scanner-Parser Algorithm. The grammar that the scanner and parser operate under is the most current version of the J2SE – Sun Microsystem's Java 1.4.2_02 specification.

The algorithm is presented as follows.

i) The scanner examines the student's code and attempts to extract a token. Let *S* be the stream of characters to be validated as a token.

ii) The validation process ensues in which comparisons are done using the reserved words and keywords of Java (Table 2), and the symbol table (Table 3).

iii) If the scanner cannot ascertain an appropriate token then the transformations T_R , T_I , T_D , and T_S are employed in an attempt to convert *S* into a valid token (i.e., a reserved word, a keyword, or a new identifier)

iv) This Transformation Sequence (TS) is recorded by the scanner in a special table called the Transformation Sequence Table (TST).

v) After a sufficient number of transformations (i.e., k-error corrections), a token will be constructed.

vi) The token is submitted to the parser.

vii) The parser asks the question: "In the current context, has a reasonable token been accepted?"

    if (true) then
        parser 'locks onto' this token by adding it to the current parse tree.
    else
        reject the current form of the token and communicate this back to the scanner so that the scanner can make appropriate modifications to the transformation sequence, or construct an appropriate token based on the context. For example, ';', indicating the end of a statement may need to be created to meet the needs of the parser to complete the parse tree.

viii) Repeat i) through vii) until all input from the student's source code has been processed, and the parser has completed the construction of the parse tree.

Table 2. Java™ Reserved Words and Keywords

abstract	else	interface	super
boolean	extends	long	switch
break	false ***	native	synchronized
byte	final	new	this
case	finally	null ***	throw
catch	float	package	throws
char	for	private	transient
class	goto *	protected	true ***
const *	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp **	

Note:

* indicates a keyword that is not currently used

** indicates a keyword that was added for Java 2

*** `true`, `false`, and `null` are reserved words.

Table 3. Symbol table

Lexeme	**Token**	**Type** (identifier, method_name, reserved_word, or keyword)	**Attribute Values**
int	INT	keyword	
for	FOR	keyword	
foobar	IDENTIFIER	method_name	
prod	IDENTIFIER	identifier	value: 1
true	TRUE	reserved_word	
=	ASSIGNMENT		
…	…	…	…

4.2.1. IR Scanner-Parser Example 1 – Forward Processing

Based on the problem described in Table 1, the following example describes how the IR scanner-parser algorithm operates. Suppose JITS did not have available the solution as presented in Table 1. Also consider a student's code submission as follows:

```
For (intt i = 1; i <= 4; i++) {
    prod *= 2;
}
```

Based on this scenario, JITS would employ the IR scanner-parser algorithm. The first string of characters are extracted as 'For'. The search commences through the keywords, reserved words, and symbol table for an exact string match. This having failed, pattern matching ensues, by employing the transformation functions (T_R, T_I, T_D, and T_S). The string 'For' in the input would be converted to the keyword 'for', and in the scanner's Transformation Sequence Table would reside the details of T_R. This token would be passed to the parser which would 'lock onto' it. The scanner then reads the next symbol (i.e., '(' left-parenthesis) and passes it to the

parser, which in turn attaches it to the current parse tree. The string 'intt' is read next which undergoes the same treatment that the 'For' encountered. However, instead of a T_R transformation, a T_D would be recorded for the Transformation Sequence. Step by step, the scanner scrutinizes the input strings and attempts to classify each into a recognizable token for the parser. Figure 3 presents a pictorial view of the procedure.

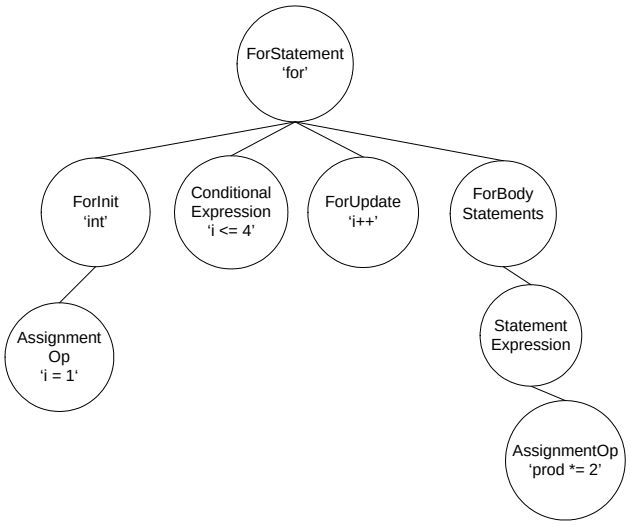

Figure 3. IR Scanner-Parser – parse tree construction

4.2.2. IR Scanner-Parser Example 2 – Forward and Backward Processing

In the previous example no situation arose where the parser rejected the token submitted by the scanner. Realistically, there will be times in which the parser cannot accept the token delivered by the scanner. For instance, consider the following:

```
For (intt i = 1; i <= 4 i++ {
    prdo *= 2
}
```

The first string of characters would be transformed in the same manner as in example 1. However, after the scanner successfully identifies token '4' as an integer, the scanner would happily submit the next token, identifier 'i'. It is here the parser would reject to continue. The last token, 'i', does not fit the grammar for the for-statement (i.e., for (initialization ; test ; increment)). The parser would ask the scanner to create, or revise the Transformation Sequence to satisfy the parser's needs to complete the sentence according to the grammar. In this example, several tokens representing symbols would be created by the scanner representing ';', ')', and ';' for the three remaining syntax errors respectively.

4.3. Tutoring Process

Before the parser 'locks-on' to a token from the scanner, the student is asked specific questions for intent clarification purposes. In this fashion, only modifications that the student knows upfront about will be made to the code.

So, using example 2 in the previous section, the following dialogue between JITS and the student would arise:

```
For (intt i = 1; i <= 4 i++ {
    prdo *= 2
}
```

JITS: "I think you meant the keyword 'for'. Is this correct?"
Student: Yes
JITS continues processing presenting the modified code below…

```
for (intt i = 1; i <= 4 i++ {
    prdo *= 2
}
```

JITS: "I see 'intt'. Do you mean the keyword 'int' ?"
Student: Yes
Resulting code:

```
for (int i = 1; i <= 4  i++ {
    prdo *= 2
}
```

JITS: "Did you mean to insert a ';' between the '4' and 'i++'?"
Student: Yes
Resulting code:

```
for (int i = 1; i <= 4; i++ {
    prdo *= 2
}
```

JITS: "There seems to be something missing. I think we are missing a ')' between 'i++' and '{'. Do you agree?"
Student: Yes
Resulting code:

```
for (int i = 1; i <= 4; i++) {
    prdo *= 2
}
```

JITS: "I found 'prdo' but there is a very similar identifier in this program named 'prod'. May I rename 'prdo' to 'prod'?"
Student: Yes
Resulting code:

```
for (int i = 1; i <= 4; i++) {
    prod *= 2
}
```

JITS: "I think we're missing a ';' at the end of this line. Do you agree?"
Student: Yes
Resulting code:

```
for (int i = 1; i <= 4; i++) {
    prod *= 2;
}
```

The tutoring process is dynamic. At any time the student is able to interject, disagree with JITS' suggestions, and modify the source code. This results in JITS having to reiterate the IR scanner-parser algorithm and the tutoring process begins again.

4.4. Logic Errors

The IR scanner-parser algorithm and tutoring process previously described does not address issues associated with logic errors. So, even though the IR algorithm and tutoring process will result in a source program that will compile, there is no guarantee that it will satisfy the program requirements.

Once the IR scanner-parser has completed the modification of the submitted code to one that parses, JITS uses information from the program specifications, and the Java™ run-time engine to extract more information regarding the correctness of the student's program. Please see the extended version of *"A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java™"* found in the Special Issue of the IJCA Journal 2004 for a discussion regarding this issue.

4.5. Efficiency Considerations

From an efficiency perspective traditional error-correction strategies in the topic of compiler construction require time proportional to the cube of the length of input [8]. That is, $O(N^3)$, where N is the number of characters in the source program. Clearly, this is not an efficient algorithm. However, JITS is not intended for programs of any size greater than 50 lines of code. As a result, considering such small values of N, the time cost would not be even noticeable to students. The purpose of JITS is to tutor beginning programming students at the College and University level and not to compile several hundred thousand lines of source code.

5. Conclusions

In summary, this research paper presented recent developments related to the Java™ Intelligent Tutoring System Prototype. The Intent Recognition scanner-parser algorithm is based on sound compiler construction theory and practices, pattern recognition techniques, and error-correction strategies. The ultimate goal of the Intent Recognition module in JITS is to understand the 'intent' of the student by carefully analyzing the student's code and to effectively tutor the student through programming problems.

This research is significant since it has the potential to be applied to many programming courses at the College and University level. This research is also quite timely considering the tremendous growth of web-based educational tools, and that Java™ has become an extremely popular programming language everywhere in the world.

6. References

[1] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons learned. *The Journal of the Learning Sciences, 4,* 167-207.

[2] Woolf, B., P., Beck, J., Eliot, C., & Stern, M. (2001). Growth and maturity of intelligent tutoring systems: A status report, In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 100-144). Cambridge, MA: MIT Press

[3] Koedinger, K. R., (2003). CIRCLE Summer School. *Lecture Series on Intelligent Tutoring Systems*, Carnegie-Mellon University, PA.

[4] Graesser A. C., Person, N. K., & Harter, D. (2001). Teaching tactics and dialog in autotutor. *International Journal of Artificial Intelligence in Education, 12,* 12-23.

[5] Koedinger, K. R. (2001). Cognitive tutors. In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 145-167). Cambridge, MA: MIT Press.

[6] Sykes, E. R., & Franek, F. (2003). A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java™, *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education, June 30-July 2, 2003, Rhodes, Greece*, 78-83.

[7] Aho, A. V., Sethi, R., & Ullman, J. D., (1988). *Compilers: principles, techniques, and tools.* Menlo Park, CA: Addison-Wesley Publishing.

[8] Aho, A. V., & Peterson, T. G., (1972). A Minimum Distance Error-Correction Parser for Context-Free Languages, *SIAM Journal of Computing, 1,* 305-312.