Schreier G, Hayn D, Ammenwerth E, editors.Tagungsband der eHealth2010. 6.-7.Mai 2010; Wien. OCG; 2010.

A FUZZY ARDEN SYNTAX COMPILER

Fehre K[1, 2], Mandl H[2], Adlassnig K-P[1, 2]

Abstract

The Arden Syntax for Medical Logic Systems is a standard for clinical knowledge representation, maintained by the Health Level Seven (HL7) organization and approved by the American National Standards Institute (ANSI). It offers a wide range of syntactical constructs (various forms of numerical, logical, temporal operators, conditions, ...), each of which crisply defines a specific unit of clinical knowledge (yes-no evaluations). As medical conditions and conclusions cannot always be formulated in a strict manner, methods of fuzzy set theory and logic are used to represent uncertainty, which is usually a part of practical clinical knowledge. Based on the extension of Arden Syntax to Fuzzy Arden Syntax by Vetterlein et al. (on the basis of Tiffe's earlier extension), we implemented a Fuzzy Arden Syntax compiler which is able to process a fully fuzzified version of Arden Syntax. We describe the compiler, its components (lexer, parser, and synthesis), and discuss its implementation.

Keywords – Clinical Decision Support, Clinical Knowledge Representation, Arden Syntax, Fuzzy Logic, Compiler Construction

1. Introduction

One of the main goals of health informatics, e-health, and medical computer sciences is to improve the quality of health care and avoid medical error. Clinical decision support systems (CDSSs) are tools that help to achieve these aims. The purpose of the systems is to provide diagnostic and therapeutic advice, alerts and reminders, or serve as disease and therapy management programs. Arden Syntax for Medical Logic Systems, maintained by the Health Level Seven (HL7) organization [1] and approved by the American National Standards Institute (ANSI), is a standardized representation form of clinical knowledge and an excellent technical point of departure to build CDSSs. An Arden Syntax knowledge base consists of a set of units known as medical logic modules (MLMs), each of which contains sufficient logic for a single medical decision. An MLM is partitioned into four categories: maintenance, library, resources, and knowledge. Maintenance, resources, and library include maintenance of the knowledge base, change control, localized strings, and explanatory information while the knowledge category includes implementable clinical information.

[1] Section for Medical Expert and Knowledge-Based Systems, Center for Medical Statistics, Informatics, and Intelligent Systems, Medical University of Vienna, Austria
[2] Medexter Healthcare GmbH, Vienna, Austria

A CDSS based on Arden Syntax checks the accessible computerized medical data with regard to specific conditions and reacts accordingly. However, medical data do not always permit an unequivocal statement as to whether a specific condition is fulfilled or not. Sometimes, preconditions of an implication are rather approximate and not entirely fulfilled. Fuzzy set theory and logic is an approach to solve this conflict.

Zadeh's article in 1965 [9] is seen as the starting point for the development of fuzzy set theory and logic. In fuzzy logic, a conclusion is permitted, provided that the result is weakened with regard to the content of its condition. Based on these considerations, Fuzzy Arden Syntax as a full extension (generalization) of Arden Syntax was first proposed by Tiffe in his Ph.D. thesis [6]. This proposal of Fuzzy Arden Syntax was taken as a basis, adapted to the interim advancements of Arden Syntax, and developed further by Vetterlein et al. [7]. For comparison we refer to [7, 8]. The Fuzzy Arden Syntax compiler we developed, programmed, and describe in this paper is based on the syntax proposed by Vetterlein et al. [7].

2. Methods

We implemented a Fuzzy Arden Syntax compiler which converts a Fuzzy Arden Syntax MLM into Java byte code runnable on an additionally implemented Fuzzy Arden Syntax engine as shown in *Figure 1*.

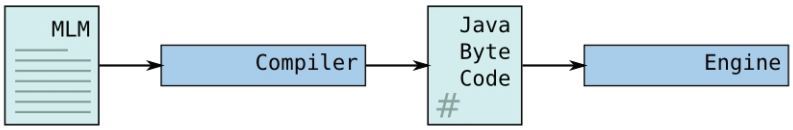

Figure 1: compiling and running an MLM

One of the main goals of the Arden Syntax is the ability to interchange MLMs or packages of MLMs (whole knowledge bases) between health care institutions. To fulfill this condition, the compiled MLMs generated by the compiler have to be as general as possible. Therefore, an additional Arden Syntax engine is needed to provide access to other MLMs, information about the surrounding environment, and additional control information.

Fuzzy Arden Syntax

The Fuzzy Arden Syntax proposed by Vetterlein et al. [7] is based on Arden Syntax, version 2.5. As the most recent version of Arden Syntax is 2.7 [1], the compiler described in this report includes several Arden Syntax extensions (actually from versions 2.1 to 2.7).

A compiler consists of several components: a lexical analyzer, a parser, and a synthesis component. As the Fuzzy Arden Syntax compiler also supports former versions of Arden Syntax, the compiler has to determine the Arden Syntax version of a provided MLM. For this purpose we developed a rudimentary lexer-parser pair which collects this information from a given MLM.

Lexer

A lexer, or lexical analyzer, converts a sequence of characters into a sequence of tokens. To this end the lexer partitions a given MLM into sequences of characters and assigns symbols to the strings according to the given language rules. Thus the lexer typifies sequences of characters. For example, the string "2 + 3" is converted into the token sequence *NUMBER PLUS NUMBER*. During the lexical analysis, the present MLM is also checked against some fundamental syntax constraints. For implementing such a lexer in Java, multiple generators are available. We selected

JFlex [4], because JFlex is a reliable Java tool and supports integration into our development process.

Parser

Following lexical analysis, the next step of the compilation process is parsing. The parser analyzes the tokens provided by the lexer and converts the linear sequence of these into a hierarchical structure, the so-called parse tree. For example, the expression "2 + 3" is represented by a node "+" with two leafs "2" and "3". During this conversion, the input (the MLM) is verified according to the syntactical and grammatical correctness as provided in the Fuzzy Arden Syntax specification. If there are any incorrect expressions, the process of compiling is aborted and an error message is sent. Similar to the lexer, parser generators are available. These are adjusted to automatically construct parsers from a given Backus–Naur form (BNF). However, the present Arden Syntax specification contains some shift-reduce problems (e.g., expressions like "sort time x") and therefore the use of a fully automated parser generator is not possible. As Arden Syntax is the foundation of Fuzzy Arden Syntax, this also holds for Fuzzy Arden Syntax. Therefore, we had to program the parser in a stepwise manner. In future implementations we will work closely with the HL7 Arden Syntax Special Interest Group to solve the issues for the upcoming Arden Syntax and Fuzzy Arden Syntax versions. Nevertheless, based on the most recent BNF for Arden Syntax we developed a BNF for Fuzzy Arden Syntax, which will be published outside of this paper.

Synthesis

The next phase of the compilation is synthesis. If the parse tree is successfully generated, the synthesis transfers the nodes of the parse tree into Java objects. This is done by creating an instance of a dedicated class for each occurrence of a language construct (e.g., an operator, value, or a variable) and by storing these in a data structure according to the parse tree. For example, a value of type number is represented by an instance of the class Arden Syntax number and subordinated to a "+" operator which may itself be subordinated to an assignment statement, a slot, and finally to an instance of the class MLM. Meanwhile, the compiler checks whether the respective MLM contains all necessary slots and resolves special expressions like the "*it*" keyword. Finally, the instance of the MLM class refers to all slots and statements in the given MLM. This instance is serialized, stored in a file, and constitutes a compiled MLM.

Engine

Contrary to other implementations of Arden Syntax [2, 3], we programmed an engine to read and execute the serialized MLMs. The engine is used to coordinate communications with the surrounding environment. Thus, for a running MLM the engine provides the ability to evaluate curly braces expressions, call events or other MLMs, and collect information about the system the engine is executed upon (e.g., local information, such as language and country). The compiled MLM is executed by calling the evaluation method of the MLM instance which itself evaluates all direct subordinated nodes. Furthermore, each node in the tree built by the compiler is evaluated by assessing its direct subordinated nodes and calculating and returning the corresponding result. If required, the engine is instructed to read data from a database or call other MLMs. Our implementation provides an interface for this engine, so that each user can adjust the engine's properties according to his/her needs.

Fuzzy Arden Syntax Handling

Compared to Arden Syntax, Fuzzy Arden Syntax contains several new language elements and extensions with respect to interpretation and evaluation of language expressions. Fuzzy linguistic

variables and a number of the data types (fuzzy number, fuzzy time, and fuzzy duration) are new in Fuzzy Arden Syntax.

Compiled linguistic variables and objects hold their values as a map containing the field names and the associated values. In contrast to Arden Syntax objects, the fields of linguistic variables may only store fuzzy data types. At present, fuzzy data types refer to piecewise linear fuzzy sets. A fuzzy set is a mapping, assigning each element of the set a given truth value, which is a real number from 0 to 1. *Figure 2* shows examples of fuzzy sets for an underweight, normal, and overweight body mass index (BMI).

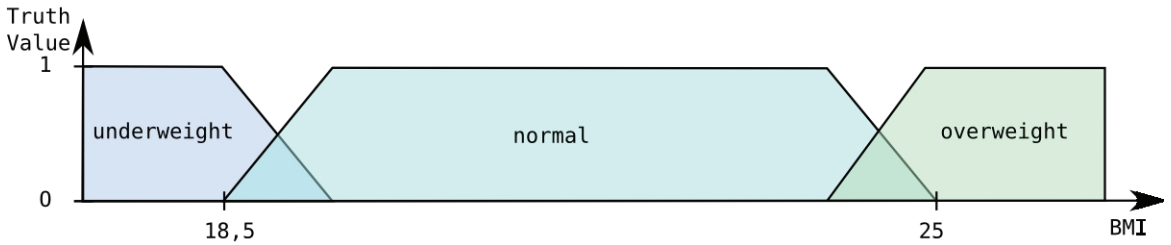

Figure 2: Three fuzzy sets

This type of fuzzy sets may be defined by the following statements:

underweight := Fuzzy Set (0,1),(18.5,1),(19.5,0); (1)
normal := Fuzzy Set (18.5,0),(19.5,1),(24,1), (25,0); (2)
overweight := Fuzzy Set (24,0),(25,1); (3)

The definition of fuzzy sets by lists of point-value pairs is also used for internal representation of these fuzzy sets. In other words, a compiled Fuzzy Arden Syntax data type has a list of point-value pairs as its value.

According to Vetterlein et al. [7], some operators must be able to handle such fuzzy data types as incoming parameters. These are, for example, the comparison operators in which one of its parameters refers to a fuzzy and the other to a crisp data type. To facilitate this process, each input parameter which refers to a fuzzy data type must provide a method, making it possible to determine the truth value at a certain point. To illustrate such a calculation of the truth value at a given point, we describe the algorithm in pseudo code below:

INPUT an arbitrary *point* whose truth value should be calculated
SET *lower-bound* to null
FOR each *fuzzy-point* in the list of *fuzzy-points*
  IF *fuzzy-point* < *point* **THEN**
    SET *lower-bound* to *fuzzy-point*
  ELSE IF *fuzzy-point* **EQUAL** *point* **THEN**
    RETURN truth-value at *fuzzy-point*
  ELSE IF *fuzzy-point* > *point* **AND** *lower-bound* is not set **THEN**
    RETURN truth-value at *fuzzy-point*
  ELSE IF *fuzzy-point* > *point* **AND** *lower-bound* is set **THEN**
    calculate *slope* of the truth value edge between *lower-bound* and *fuzzy-point*
    RETURN (*point* – *lower-bound*) * *slope* + truth value at *lower-bound*
  END IF
END FOR

Since the data type *boolean* is replaced by the data type *truth value* in Fuzzy Arden Syntax, the corresponding implementation will hold its value in a float variable. Furthermore, each data type implementation will store the applicability, a truth value expressing the degree to which the main value may be considered applicable.

A major challenge was the implementation of conditional statements such as *if*, *conclude*, or *switch*. If the condition is an expression of the *truth value* type then, unlike previous Arden Syntax versions, Fuzzy Arden Syntax has to execute each of the conditional blocks, each with the appropriate truth value. If the execution of a compiled MLM arrives at such a statement, the entire current variable heap is copied, and the applicabilities of the contained values are manipulated accordingly. If the conditional statement does not contain an aggregate expression, it returns a set of variable heaps to the calling node, which evaluates all following subordinated nodes with each of the resulting heaps. Thus, the evaluation of an MLM may provide several results that are interpreted according to the calling instance. The arithmetic operations for variables referring to fuzzy sets are defined and implemented according to Zadeh's extension principle [9], provided that the corresponding operation in the crisp case is defined as well.

3. Discussion and Conclusion

We developed a Fuzzy Arden Syntax compiler to compile the source code MLMs and a Fuzzy Arden Syntax engine to execute the compiled MLMs. The compiler is a complete extension of the existent Arden Syntax compiler which can parse all versions of Arden Syntax from 2.1 up to the most recent version 2.7. This Arden Syntax compiler together with its corresponding engine is embedded in an Arden Syntax server. The server is based on a service-oriented architecture protocol (SOAP)-based framework for remotely calling and running compiled MLMs. In addition, we developed a Fuzzy Arden Syntax integrated development environment (IDE) to be used by an MLM author to write, verify, compile, and test MLMs.

It should be noted that Arden Syntax is an imperative programming language which is procedural and modular. We suggest that the Arden Syntax (and thus Fuzzy Arden Syntax) is Turing-complete, but formal proof is yet to be obtained and will be a subject of future research. A quine [5], named after the philosopher and logician Willard van Orman Quine (1908–2000), is the simplest form of a self-replicating program. A quine prints its own code and exists for any programming language that is Turing complete. The construction of a quine for Arden Syntax is trivial. The presentation of a quine would exceed the scope of this report.

4. References

[1] HEALTH LEVEL 7, Arden Syntax for Medical Logic Systems, Version 2.7. Health Level 7, Ann Arbor, MI, 2008.

[2] HRIPCSAK, G., CIMINO, J.J., JOHNSON, S.B., CLAYTON, P.D., The Columbia-Presbyterian Medical Center Decision-Support System as a Model for Implementing the Arden Syntax, in: P. D. Clayton (Ed.) Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care, McGraw-Hill, New York, 248–252, 1992.

[3] JENDERS, R.A., HRIPCSAK, G., SIDELI, R.V., DUMOUCHEL, W., ZHANG, H., CIMINO, J.J., JOHNSON, S.B., SHERMAN, E.H., CLAYTON, P.D, Medical Decision Support: Experience with Implementing the Arden Syntax at the Columbia-Presbyterian Medical Center, in Proc AMIA Symp 1995, 169–173, 1995.

[4] JFlex 1.4.3. Available at http://www.jflex.de (last accessed: 10 January 2010).

[5] QUINE, W. V, On Decidability and Completeness. Synthése 7:441–446, 1949.

[6] TIFFE, S., Fuzzy Arden Syntax: Representation and Interpretation of Vague Medical Knowledge by Fuzzified Arden Syntax, Ph.D. Thesis, Technical University Vienna, Vienna 2003.

[7] VETTERLEIN, T., MANDL, H., ADLASSNIG, K.-P., Vorschläge zur Spezifikation der Programmiersprache Fuzzy Arden Syntax (Proposal of a specification of the programming language Fuzzy Arden Syntax – in German). Technical Report, Medical University of Vienna, Vienna, 2008. Available at http://www.meduniwien.ac.at/user/thomas.vetterlein/articles/FuzzyArdenSpezif.pdf (last accessed: 10 January 2010).

[8] VETTERLEIN, T., MANDL, H., ADLASSNIG, K.-P., Fuzzy Arden Syntax: A Fuzzy Programming Language for Medicine. Artificial Intelligence in Medicine, doi:10.1016/j.artmed.2010.01.003, 2010.

[9] ZADEH, L. A., Fuzzy Sets. Information and Control 8:338–353, 1965.

Corresponding Author
Karsten Fehre
Section for Medical Expert and Knowledge-Based Systems
Center for Medical Statistics, Informatics, and Intelligent Systems
Medical University of Vienna
Spitalgasse 23, BT 88.03, A-1090 Vienna
Email: karsten.fehre@meduniwien.ac.at