Arden Syntax – Advanced Topics

Karsten Fehre

Medexter Healthcare GmbH
Borschkegasse 7/5
A-1090 Vienna, Austria
www.medexter.com

Statements – Object Statements

- **Object:** Assigns an object declaration to a variable (objects are the only data types in Arden Syntax that are first declared and then “instantiated”)

```
MedicationDose := OBJECT [Medication, Dose, Status];
```

- **New:** Causes the creation of a new object (based on the used object declaration)

```
dose1 := NEW MedicationDose; //empty object
dose2 := NEW MedicationDose with "Ampicillin", "500mg", "Active";
```

- **Dot:** Selects an attribute from an object, based on the name following the dot. The dot operator is used to access the fields of an object

```
"John" := patient.Name.FirstName;
NameType := object [FirstName, LastName];
/* Assume namelist contains a list of 2 NameType objects */
("John", "Paul") := namelist.FirstName;
```

Statements – Object Statements – Example

```
// object declaration
bmiResult := object [bmi, classification];
result := new bmiResult; // create an empty result object
```

- The first statement creates an **object declaration** with two fields and assigns this declaration to the variable `bmiResult`
- The second statement creates an empty **instance** of the `bmiResult` object and assigns this to the variable `result`

```
write result.bmi || result.classification || ".”; // return result
```

- Concatenates the content of the field `bmi` and the content of the field `classification` of the object `result` to a string and sends this message to the default destination

```
result.bmi := result.bmi formatted with localized ‘msg’;
```

- The **dot operator (".")** is used to access the fields of an object
- The field `bmi` of the object `result` will be filled with the formatted text containing the calculated BMI

Statements – Advanced Object Statements

- **Attribute Assignment:** Allows the assignment to individual attributes of an object.

```
medication := new MedicationDose;
medication.Dose := "500mg";
medication.Status := "Active";
```

- **Enhanced Assignment:** Any expression that ends with a **dot** operation or **element** operation may be placed on the left hand side of an assignment.

```
my_obj.message_list[n].msg := "this is a replacement message";
```

- **Extract Attribute Names:** Returns a list containing the attribute names of the object argument.

```
"Medication", "Dose", "Status" := EXTRACT ATTRIBUTE NAMES medication;
```

Expressions – Curly Braces (Mapping Clauses)

- Are used in the data slot to signify institution-specific definitions such as database queries

- **Read statement:** Reads data from the host system

```
var1 := READ {select potassium from results where specimen = 'serum'};
```

- **Event statement:** Defines an event; an event can be used to call MLMs

```
event1 := EVENT {storage of serum potassium};
```

- **Message statement:** Text that is used by write statements

```
message1 := MESSAGE {increased body temperature};
```

- **Destination statement:** Target that is used by write statements

```
destination1 := DESTINATION {email: user@cuasdf.bitnet};
```

- **Interface statement:** function that is evaluated by host system

```
func_drugint := INTERFACE {char* API:Interaction (char*,char*) };
```

Expressions – Curly Braces (Mapping Clauses) – Example

```
// read all measured weights from the data base
Let weights be read {select measured_weights from DB where patID = patientID};
```

- This **assignment** statement assigns the result of the read statement (using mapping clause "SELECT measured_weight FROM DB WHERE patID = patientID") to the variable weights
- patientID is a variable that contains the patient ID currently in use and is substituted before execution of the mapping clause
- After evaluation of this statement, the variable weights refers to the result which is a list of all measured weights of the patient with the given patient ID
- The content of the **curly brace expressions** must be evaluated by the host system and its syntax is not part of the Arden Syntax

```
Let userEvent be event {getBMI};
```

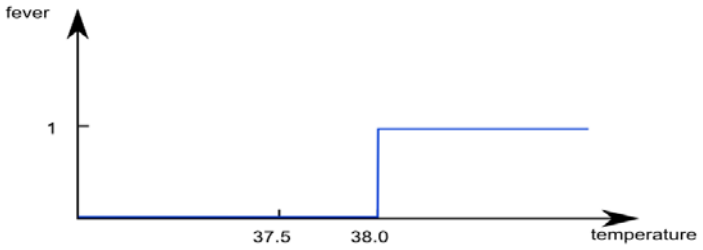
- **Assignment** statement that assigns the event getBMI to the variable userEvent

```
Evoke: userEvent;;
```

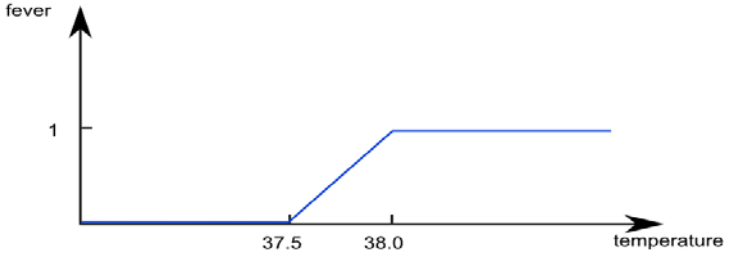
- If the event variable is used in the **evoke slot,** the MLM is always executed, when this event occurs

Fuzzy Sets – Background II

- **Crisp** boundary
  - Defines a **sharp** threshold
  - Checking if a given value is greater or less than the defined crisp threshold results in either true or false
  - Borderline cases are not detected



- **Fuzzified** boundary
  - Defines a **gradual** transition
  - Checking if a given value is greater or less than the defined fuzzified boundary results in a truth value between 0 and 1
  - Borderline cases are detected
  - Weighted results for borderline cases, all other are as usual



Fuzzy Sets – Example I

- **Classical** Arden Syntax

```
fever_limit := 38;
temperature := 37.9;

message := "patient has no fever";
IF temperature > fever_limit THEN
   message := "patient has fever";
END IF
```

  - Result message: “patient has no fever”
  - Borderline case is not detected

- **Fuzzy** Arden Syntax

```
fever_limit := FUZZY SET (37.5,0), (38,1);
temperature := 37.9;

message := "patient has no fever";
IF temperature > fever_limit THEN
   message := "patient has fever";
END IF
```

  - Result message: “patient has fever” (with applicability 0.8)

Fuzzy Sets – Example II

- **Classical** Arden Syntax

```
fever_border := 38;
sub_border := 37.5;
temperature := 37.9;

message := "patient has no fever";

IF temperature > fever_border THEN
    message := "patient has fever";
    app := 1;
ELSE IF temperature > sub_border THEN
    message := "patient has fever";
    app := (temperature-sub_border)/0.5;
END IF
```

  - Variable message contains the string “patient has fever”
  - Applicability (variable app) is the truth value 0.8

- **Fuzzy** Arden Syntax

```
fever_border := FUZZY SET (37.5,0), (38,1);

temperature := 37.9;

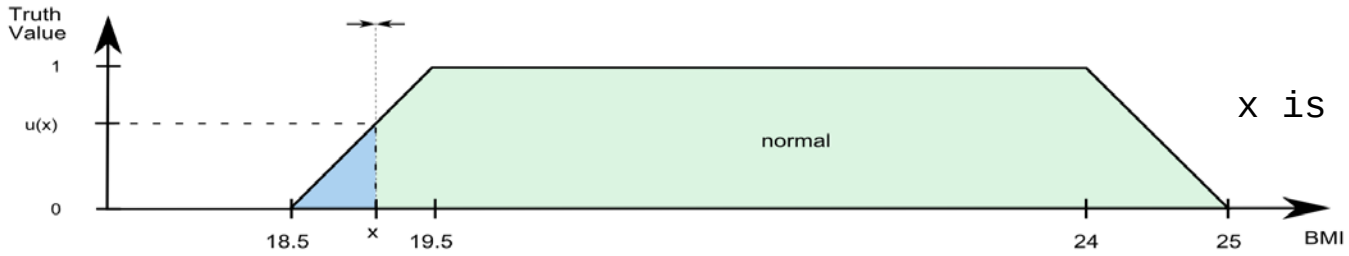
message := "patient has no fever";

IF temperature > fever_border THEN
    message := "patient has fever";
END IF

app := applicability of message;
```

  - Variable message contains the string “patient has fever”
  - Applicability (variable app) is the truth value 0.8

Data Types – Fuzzy Sets



x is in **BMI.normal**

- Definition of a fuzzy set

```
Fuzzyset_u := FUZZY SET (18.5,0), (19.5,1), (24,1), (25,0);
Fuzzyset_v := 7 fuzzified by 2;
```

- Fuzzy set based on other data types

```
Fuzzyset_duration := FUZZY SET (3 days,0), (10 days,1), (20 days,1), (25 days,0);
simple := 2009-10-10 fuzzified by 12 hours;
complex :=FUZZY SET (2009-10-10,0), (2009-10-11,1), (2009-11-10,1), (2009-11-11,0);
```

Applicability

- Arden Syntax contains two types of fuzziness:
  - Data types: for explicit calculations, e.g., truth value, fuzzy set
  - Degree of applicability: for weighting MLM evaluation and weighting of branches

- All simple data types are endowed with additional information concerning the degree of **applicability**

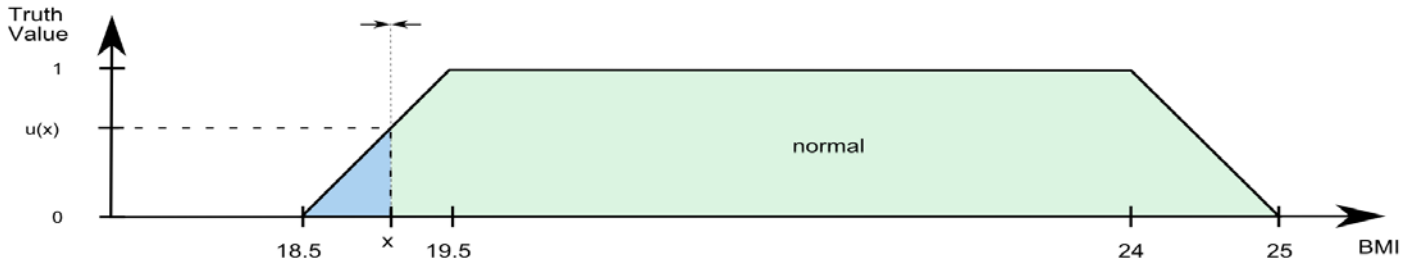
- Stores a truth value that refers to the degree to which it is reasonable to use the value of a variable

- Default degree of applicability is 1 and the degree of applicability is never null

- Can be accessed using the `applicability` operator

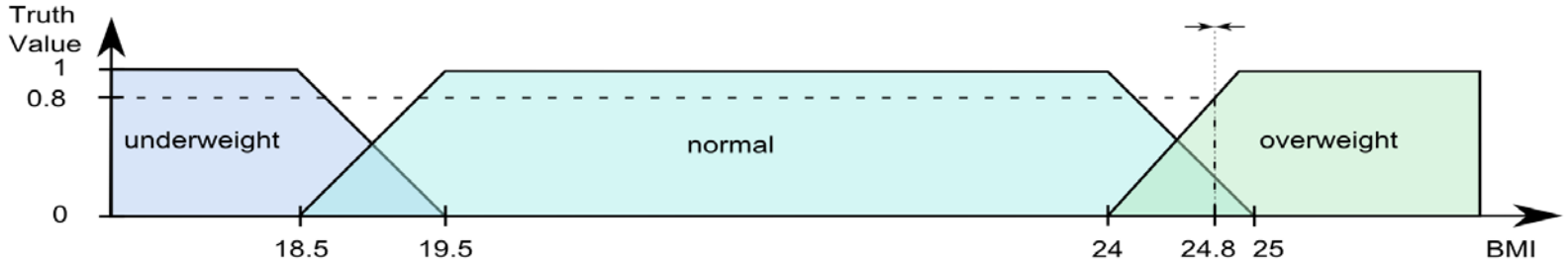
- If-then statements with a condition that evaluates to a truth value [0,1] result in a split of the MLM execution
  - Each branch will be executed under corresponding applicability
  - The applicability is implicit attached to each variable of the branch

Data Types – Truth Value



- Generalization of the Boolean data type
- Value between 0 and 1
- Boolean value true is equal to the truth value 1 and the Boolean value false is equal to the truth value 0
- May be the result of mapping a clinical value to a fuzzy set
- Can also be defined explicitly

Data Types – Linguistic Variable



- Construct to represent a linguistic concept and its sub-concepts
- Subsumes the sub-concepts of a concept under one term
- Definition of a **linguistic variable**
  - ```
    data:
    BMIDefinition := LINGUISTIC VARIABLE [underweight,normal,overweight];
    logic:
     BMI := new BMIDefinition;
     BMI.underweight := FUZZY SET (18.5,1), (19.5,0);
     BMI.normal      := FUZZY SET (18.5,0), (19.5,1), (24,1), (25,0);
     BMI.overweight  := FUZZY SET (24,0), (25,1);
    ```
    ```

Statements – If-Then-Else – Fuzzy Condition

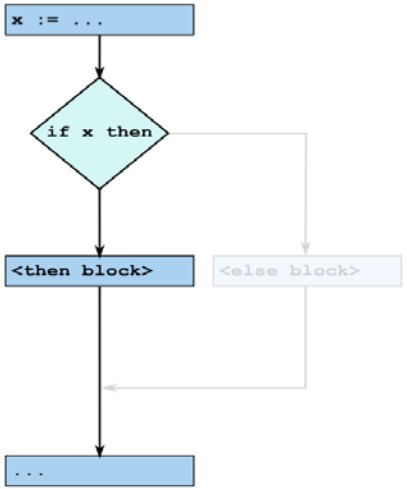
- If the used condition in an **If-Then-ElseIf** statement evaluates to a truth value between 0 and 1, both blocks are executed
- Each branch is provided with its own set of variables which are duplicated accordingly
- The **degree of applicability** of each variable in the if-block is multiplied with the truth value of the condition
- In the else-block, the degree of applicability of each variable is multiplied with 1 minus the truth value of the condition
- The general applicabilities of these blocks are called relative weights
- The weight of an MLM evaluation is 1, as long as it does not split
- The program may branch several times
- If the branches are not subsumed using the `aggregate` keyword, the branches are executed in parallel and the MLM will finish with 2 or more return values (with different applicabilities)

Statements – If-Then-ElseIf – Fuzzy Condition – Example

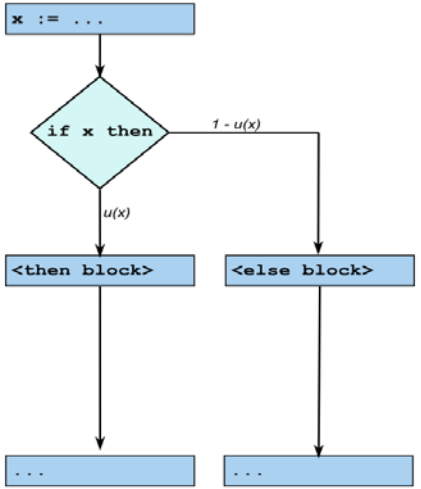
Source

```
maintenance: [...]
knowledge: [...]
logic:
 //define linguistic variable
 //BMI as above
 [...]
 myBMI := 24.8;
 x := myBMI <= BMI.overweight;
 if x then
   // this branch is executed
   // with applicability 0.8
   <then_block>
 else
   // this branch is executed
   // with applicability 0.2
   <else_block>
 endif;
[...]
end:
```

Classic Arden Syntax

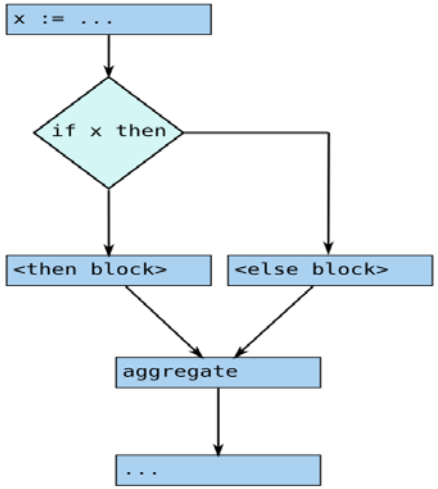


Fuzzy Arden Syntax



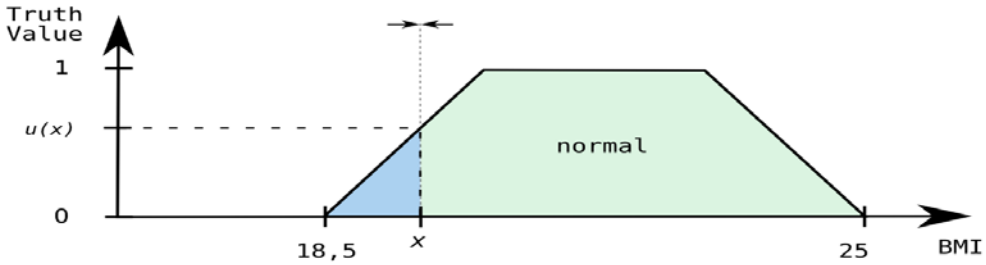
Statements – If-Then-Aggregate

```
if x then
   <then_block>
else
   <else_block>
endif AGGREGATE;
```



- Combination of the variable values in each execution branch according to their applicability
- Aggregations are common in fuzzy control

Operators – Comparison Operators – Fuzzy Comparison




```
x is in BMI.normal
x <= BMI.normal
x >= BMI.normal
```

- The behavior of the comparison operators is different to the standard case when a crisp value is compared to a **fuzzy set**
- `x is in BMI.normal`
  Returns the truth value (`u(x)`) to that the crisp value (`x`) is mapped by the fuzzy set
- `x <= BMI.normal`
  Returns the maximum of the mappings of all $y \geq x$ (green shape)
- `x >= BMI.normal`
  Returns the maximum of the mappings of all $y \leq x$ (blue shape)

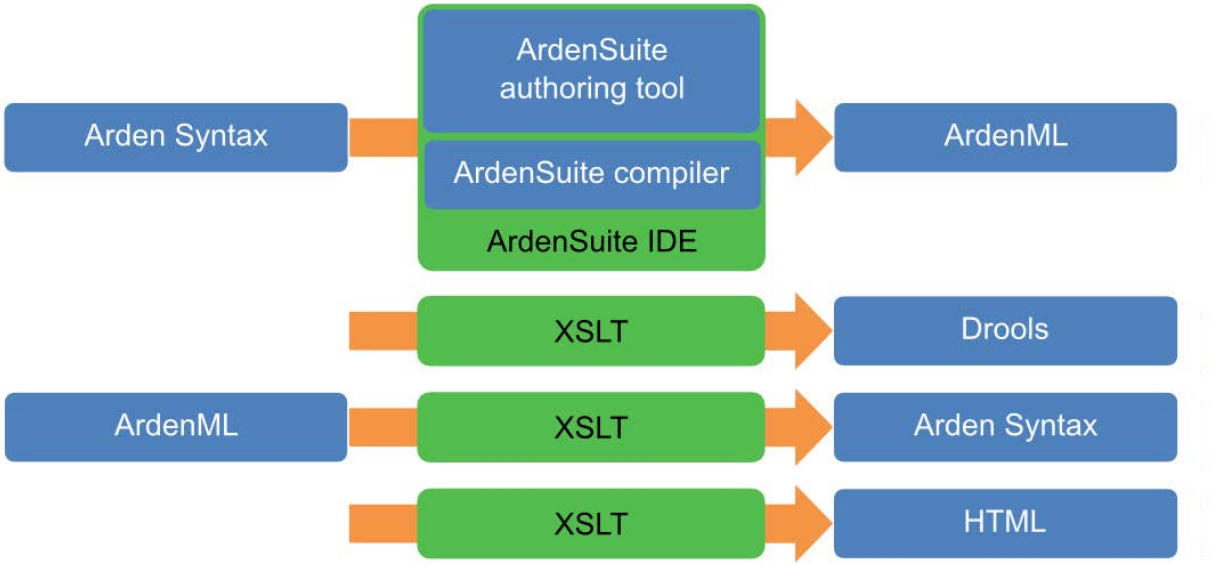
ArdenML: Objectives and applications

- Provide a complete XML schema for version 2.10 of the Arden Syntax to express MLMs in XML
- Thus, Arden Syntax is now compatible with all other HL7 standards based on XML (HL7 version 3, VmR, and others)
- Further benefit: To be able to use available XML tools

ArdenML: Example

```
<Library>
    <Purpose>Test</Purpose>
    <Explanation></Explanation>
    <Keywords></Keywords>
</Library>
<Knowledge>
    <Type>data_driven</Type>
    <Data></Data>
    <Evoke></Evoke>
    <Logic>
        <Assignment>
            <Identifier var='var1' />
            <Assigned>
                <Value otype='time'>1990-03-15T15:00:00</Value>
            </Assigned>
        </Assignment>
        <Assignment>
            <Identifier var='res1' />
            <Assigned>
                <ReplaceYearWith>
                    <Identifier var='var1' />
                    <Value otype='number'>2011</Value>
                </ReplaceYearWith>
            </Assigned>
        </Assignment>
        <Assignment>
            <Identifier var='res2' />
            <Assigned>
                <ReplaceYearWith>
                    <Identifier var='var1' />
                    <List>
                        <Value otype='number'>2011</Value>
                        <Value otype='number'>2010</Value>
                    </List>
                </ReplaceYearWith>
            </Assigned>
```

Cross compilation / transformation of Arden Syntax to / from ArdenML



Team effort by Intermountain Hospitals, Salt Lake City, Utah, U.S.A., and Medexter Healthcare, Vienna, Austria

THE END