Arden Syntax
What is Arden Syntax?

- ... a language used for representing and sharing medical knowledge.

- ... used for sharing of computerized health knowledge bases across personnel, information systems, and institutions.

- ... organized using modules, while each module, referred to as a Medical Logic Module (MLM), contains sufficient knowledge to make a single decision.

- ... an executable format which can be used by clinical decision support systems.
In Arden Syntax, medical knowledge is arranged within Medical Logic Modules (MLMs).

Each MLM represents sufficient knowledge to make a single clinical decision.

One or more MLMs are stored within a file that has the extension “.mlm”.

Each MLM is well organized and structured into categories and slots with specific content.

An MLM is composed of slots, grouped into the following four required categories: maintenance, library, knowledge, and resources.

Categories must appear in the correct order.

Within each category is a set of slots that must appear in the correct order, too.
MLMs are working in close contact with their host system. Ways of interaction are:

- **Input**: By calling an MLM, an input parameter can be committed

- **Curly Brace Expressions**: So called “curly brace expressions” implement a special kind of dynamic interaction between MLMs and host systems

- **Write Statements**: Texts can be written to destinations that are maintained by the host system

- **Output**: Analogous to the input parameter, data can be committed from the MLM to the host system after the execution of the MLM has finished

In order to start the execution of an MLM, an engine is needed that handles communication with the host system and can tell which of the MLMs are available

Ways to start running an MLM:

- **MLM call**: An MLM is directly called

- **Event call**: Any MLM that listens to a specific event is executed
Arden Syntax – Fundamentals III

- Data types

- Statements, expressions (assignments, loops, variables, constants, objects)

- Operators
  - List operators
  - Logical operators
  - Comparison operators
  - String operators
  - Arithmetic operators
  - Temporal operators
  - Aggregation operators
  - Time and object operators
Primary Time

- In addition to its value part each data value has a **primary time** part and an applicability
- Primary time represents the value part’s time of creation or measurement
- By default, primary time is null
- Can be accessed using the `time` operator
  
  \[
  2011-03-15T00:00:00 := 2 \text{ days AFTER} 2011-03-13T00:00:00
  \]
- Database query results should contain both, the value and the primary time
  - Might be the time when a blood test was drawn from the patient
  - Might be the time when a medication order was placed
  - Which time of a database entry is taken as primary time is left to the used Arden Syntax implementation
History

- A first draft of the standard was prepared at a meeting at the Arden Home-stead, New York, in 1989. Arden Syntax was previously adopted as a standard by the American Society for Testing and Materials (ASTM) as document E 1460, under subcommittee E31.15 Health Knowledge Representation.

- 1992: Arden Syntax version 1.0

- 1998: sponsorship moved to HL7 International (Arden Syntax Work Group)

- 1999: Arden Syntax version 2.0 adopted by HL7 and the American National Standards Institute (ANSI)

- 2014: Arden Syntax version 2.10
## History

<table>
<thead>
<tr>
<th>Version</th>
<th>Year</th>
<th>Important changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>2002</td>
<td>new string operators; reserved word “currenttime” returns the system time</td>
</tr>
<tr>
<td>2.5</td>
<td>2005</td>
<td>object capabilities: create and edit objects; XML representation of MLMs (except logic, action and data slot)</td>
</tr>
<tr>
<td>2.6</td>
<td>2007</td>
<td>UNICODE encoding; additional resources category to define text resources for specific languages; time-of-day and day-of-week data types; “localized” operator to access texts in specific languages</td>
</tr>
<tr>
<td>2.7</td>
<td>2008</td>
<td>enhanced assignment statement; extended “new” operator to allow easy and flexible object instantiation</td>
</tr>
<tr>
<td>2.8</td>
<td>2012</td>
<td>additional operators for list manipulation; operators to manipulate parts of given date and time values; switch statements; keyword “breakloop” for aborting a loop; number of editorial corrections</td>
</tr>
<tr>
<td>2.9</td>
<td>2013</td>
<td>fuzzification: fuzzy data types and fuzzy sets; adjustment of all available operators to be able to handle fuzzy data types</td>
</tr>
<tr>
<td>2.10</td>
<td>2014</td>
<td>XML representation of whole MLMs (including logic, action and data slot)</td>
</tr>
</tbody>
</table>
**Fuzzy Sets – Background I**

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

- **Fuzzified border**
  - Defines a **gradual** border
  - Checking if a given measurement is greater or less than the defined fuzzified border 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

• **Usual** Arden Syntax

\[
\begin{align*}
\text{fever}_\text{limit} & := 38; \\
\text{temperature} & := 37.9; \\
\text{message} & := \text{"patient has no fever"}; \\
\text{IF} \; \text{temperature} > \text{fever}_\text{limit} \; \text{THEN} \\
& \quad \text{message} := \text{"patient has fever"}; \\
\text{END IF}
\end{align*}
\]

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

• **Fuzzy** Arden Syntax

\[
\begin{align*}
\text{fever}_\text{limit} & := \text{FUZZY SET} \; (37.5, 0), \; (38, 1); \\
\text{temperature} & := 37.9; \\
\text{message} & := \text{"patient has no fever"}; \\
\text{IF} \; \text{temperature} > \text{fever}_\text{limit} \; \text{THEN} \\
& \quad \text{message} := \text{"patient has fever"}; \\
\text{END IF}
\end{align*}
\]

- Result message: “patient has fever” (with applicability 0.8)
How to execute MLMs

- **MLM calls**: When the MLM call statement is executed, the current MLM is interrupted, and the named MLM is called; parameters are passed to the named MLM

  ```plaintext
  /* Define find_allergies MLM */
  find_allergies := MLM 'find_allergies';
  (allergens, reactions):= call find_allergies;
  ```

- **Event calls**: When the event call statement is executed, the current MLM is interrupted, and all the MLMs whose evoke slots refer to the named event are executed; parameters are passed to the named MLMs

  ```plaintext
  allergy_found := EVENT {allergy found};
  reactions := call allergy_found;
  ```
How to get data into MLMs

- MLM and event calls
  - Provide data to an other MLM
  - Read data provided to the actual MLM
  - Return data to calling MLM or instance

- Curly brace expressions
  - Read data from external data sources
  - Write data to external data sources
  - Call external applications or interfaces
How to get data into MLMs – Curly Brace Expressions
How to execute MLMs – Web Service Usage