

Vorschläge zur Spezifikation der Programmiersprache FuzzyArden

Thomas Vetterlein

Institut für Medizinische Experten- und wissensbasierte Systeme
Besondere Einrichtung für medizinische Statistik und Informatik
Medizinische Universität Wien
Spitalgasse 23, BT 88.03, A - 1090 Wien, Österreich
`Thomas.Vetterlein@meduniwien.ac.at`

Harald Mandl

Medexter Healthcare GmbH
Borschkegasse 7/5, 1090 Wien, Österreich
`hm@medexter.com`

Klaus-Peter Adlassnig

Institut für Medizinische Experten- und wissensbasierte Systeme
Besondere Einrichtung für medizinische Statistik und Informatik
Medizinische Universität Wien
Spitalgasse 23, BT 88.03, A - 1090 Wien, Österreich
`Klaus-Peter.Adlassnig@meduniwien.ac.at`

Oktober 2009

Zusammenfassung

Wir geben eine Kurzdefinition der Programmiersprache Arden gemäß der offiziellen Vorgabe [HL7]. Im Bemühen um ein übersichtliches Programmiermodell blenden wir die technischen Aspekte dabei so weit wie möglich aus.

Weiter stellen wir eine Erweiterung der Sprache vor; Ziel dabei ist, Aussagen mit kontinuierlich abgestuften Zutreffensgraden auf konzeptioneller Ebene mit einzubeziehen. Grundlage bildet die in [Tif] entworfene Sprache FuzzyArden, deren modifizierte Version wir im Detail darlegen.

Inhaltsverzeichnis

1 Einleitung

4

2	Grundsätzliches über Arden	5
2.1	Das umgebende System	5
2.2	Die Grundstruktur eines Arden-Programmes	5
3	Datentypen in Arden	7
3.1	Variablen und Datentypen	7
3.2	Die einfachen internen Datentypen	8
3.3	Die Zeitkomponente	10
3.4	Die zusammengesetzten internen Datentypen	11
3.5	Die externen Datentypen	12
3.6	Abfrage des Typs einer Variablen	14
3.7	Vordefinierte Zeitvariablen	14
4	Operationen in Arden	15
4.1	Boolesche Werte	15
4.2	Zahlen	15
4.3	Zeit und Dauer	17
4.4	Manipulation von Zeichenketten	18
5	Bearbeitung von Listen in Arden	19
6	Die Auslösung von MLMs	21
6.1	Events	21
6.2	Direkter Aufruf eines MLMs	22
6.3	Ausführungsprioritäten	23
7	Datenein- und -ausgabe	23
7.1	Daten vom Host einlesen	23
7.2	Aufruf von Funktionen im Host	24
7.3	Datentransfer zwischen MLMs	24
8	Das Arden-Programm	25

8.1	Entscheidungspunkte	25
8.2	Entscheidung über Konsequenzen	26
9	Ergänzung: Änderung bei der Darstellung gegenüber [HL7]	26
10	Grundsätzliches über FuzzyArden	28
11	Änderungen bei den Datentypen in FuzzyArden	29
11.1	Der Datentyp Wahrheitswert	29
11.2	Die Datentypen Fuzzyzahl, Fuzzyzeit und Fuzzydauer	29
11.3	Die Wahrheitswertkomponente	31
12	Änderungen bei den Operationen in FuzzyArden	33
12.1	Wahrheitswerte	33
12.2	Zahlen	35
12.3	Zeit und Dauer	37
12.4	Abfrage des Typs einer Variablen	37
12.5	Defuzzifizierung	38
12.6	Linguistische Ausdrücke	38
13	Bearbeitung von Listen in FuzzyArden	40
14	Das FuzzyArden-Programm	41
14.1	Aufspaltung an Verzweigungspunkten	41
14.2	Die Zusammenführung von Programmzweigen	43
14.3	Die Bedingung für den Action-Slot	46
14.4	Ergänzende Anmerkungen zu Parallelausführung und Zusammenführung	46
14.5	Schleifen	48
15	Ergänzung: Unterschiede zu [Tif]	48

1 Einleitung

Der Sinn eines Arden-Systems ist es, aus den aus medizinischen Informations- und Kommunikationssystemen verfügbaren Daten eines Patienten Stellungnahmen abzuleiten, deren Kenntnis für das ärztliche Personal hilfreich sein könnte. In Echtzeit ist festzustellen, ob sich aus den zur Verfügung stehenden Daten Fakten ableiten lassen, die eine Konsequenz hinsichtlich Patientenbehandlung oder -führung nahelegen; in diesem Fall werden entsprechende Vorschläge angeboten und allenfalls weitere Funktionen des Systems aufgerufen. Ein derartiger Vorschlag kann in Form einer Warnung, eines Hinweises oder einer Empfehlung in Bezug auf Diagnose, Therapie, Prognose oder Administration erfolgen.

Im Rahmen eines typischen in der durch [HL7] spezifizierten Sprache Arden geschriebenen Programmes werden Bedingungen definiert, die aufgrund der Datenlage zutreffen oder nicht; im positiven Fall wird eine bestimmte Reaktion hervorgerufen, andernfalls erfolgt keine Reaktion. Die Erweiterung zur Programmiersprache FuzzyArden, wie sie in [Tif] vorgeschlagen worden ist, zielt darauf ab, weiche, graduelle Übergänge definieren zu können; berücksichtigt werden soll der Umstand, dass sich eine Sachlage, die eine bestimmte Empfehlung impliziert, im Allgemeinen nicht scharf ab- bzw. eingrenzen lässt. In FuzzyArden ist es in einfacher Weise möglich, Bedingungen auch mit Bezug auf Daten zu formulieren, denen Vagheit, Unschärfe, Unsicherheit o. Ä. anhaftet. Im Ergebnis können mehrere mit Gewichtung versehene Alternativen mitgeteilt oder aber kontinuierliche Übergänge zwischen den möglichen Schlussfolgerungen des Programms geschaffen werden.

2 Grundsätzliches über Arden

2.1 Das umgebende System

Ein Arden-System ist modular aufgebaut; es setzt sich aus einer Zahl einander gleichgeordneter Programmeinheiten zusammen, die sich *Medical Logic Modules* oder kurz *MLMs* nennen. Die zentrale Steuerung obliegt einem Host-System, welches die Ausführung der MLMs initiieren kann, von den MLMs angeforderte Daten bereitstellt und für die Umsetzung allfälliger Konsequenzen zu sorgen hat.

Ein einzelnes MLM wird unter einer der folgenden Bedingungen ausgeführt. Erstens kann im MLM selbst ein spezifisches Ereignis benannt sein, im folgenden *Event* genannt, das die Ausführung dieses MLMs nach sich zieht. Das Eintreten eines Events wird vom Host übermittelt, kann aber auch durch eines der übrigen MLMs signalisiert werden. Zweitens kann das MLM sowohl vom Host als auch von jedem anderen MLM direkt aufgerufen werden.

2.2 Die Grundstruktur eines Arden-Programmes

Ein MLM gliedert sich in die Abschnitte (engl. *categories*) *maintenance*, *library* und *knowledge*. Jeder dieser drei Programmteile beginnt mit einer Zeile, die die mit Doppelpunkt versehene Bezeichnung enthält, etwa `maintenance:.` Jeder Abschnitt ist wiederum in Unterabschnitte gegliedert, die sich *Slots* nennen. Ein Slot wird ebenfalls durch seine mit Doppelpunkt versehene Bezeichnung eingeleitet, und er schließt mit zwei aufeinanderfolgenden Semikola, d. h. mit `;;`.

Die ersten beiden der drei Abschnitte enthalten ausschließlich Metadaten. Die programmtechnisch relevanten Informationen hierin sind einzig der Name des MLMs, die Institution, die für das MLM verantwortlich zeichnet, sowie die Arden-Version. Im Maintenance-Abschnitt sind zu diesem Zweck Slots der folgenden Form enthalten:

```
mlmname: Prüfung_einer_Allergie_gegen_Wirkstoff_XY;;  
institution: Medizinische_Universität_Wien;;  
arden: version 2.5;;
```

(Hier wie im folgenden sind die ardeneigenen Schlüsselwörter durchwegs klein, Beispielnamen hingegen groß geschrieben. In Arden selbst wird zwischen großen und kleinen Buchstaben nicht unterschieden.)

Die eigentlichen Programmanweisungen folgen im Abschnitt *knowledge*. Dieser untergliedert sich in die Slots *type*, *data*, *priority*, *evoke*, *logic*, *action* und optional *urgency*, die in dieser Reihenfolge zu definieren sind.

Der Slot *type* ist funktionslos und hat immer die Form

```
type: data_driven;;
```

Der Evoke-Slot ist derjenige, in dem der Umstand definiert wird, dessen Eintreten den automatischen Aufruf des vorliegenden MLMs zur Folge hat. Haben mehrere MLMs auf ein Ereignis gleichzeitig zu reagieren, wird über den Priority-Slot die Reihenfolge der Aufrufe beeinflusst.

Der eigentliche Programmcode ist über den Data-, Logic- und Action-Slot verteilt und wird in dieser Reihenfolge abgearbeitet. Dadurch wird eine grobe Struktur vorgegeben, die allerdings nur in sehr beschränkter Hinsicht Vorschriften über die Programmgestaltung impliziert. Die Idee ist die folgende: Der Data-Slot ist für die Entgegennahme von Daten vorgesehen; für die etwaige Veranlassung von Reaktionen auf vom Programm Festgestelltes ist der Action-Slot gedacht; und die eigentliche Verarbeitung der Daten sollte im Logic-Slot ablaufen.

Im Urgency-Slot schließlich kann die Dringlichkeit sich allenfalls aus den Patientendaten ergebender, im Action-Slot definierter Konsequenzen festgelegt werden.

Gewisse Ausdrücke innerhalb von Programmzeilen dienen der Kommunikation mit dem Host, wie etwa die Abfrage von Werten aus der angeschlossenen Datenbank. Diese vom Host zu interpretierenden Ausdrücke sind in ihrer Form nicht durch Arden vorgegeben. Jeder solche Ausdruck wird zwecks Trennung vom übrigen Programmtext in geschweifte Klammern gesetzt und nennt sich sinnigerweise eine *Curly-brace-expression*. Die Curly-brace-expressions müssen sich im Data-Slot befinden.

Jede Befehlszeile wird mit einem Semikolon abgeschlossen. Zwischen */** und **/* Befindliches sowie innerhalb einer einzelnen Zeile auf *//* Folgendes gilt als Kommentar und wird übergangen.

Wir befassen uns im weiteren mit den Einzelheiten der Programmierung in Arden. Nicht alle zum Programmieren erforderliche Kenntnis kann hier vermittelt werden; für ergänzende Informationen konsultiere man [HL7]. Zu beachten ist zudem, dass wir von der Notation und der Sprechweise in [HL7] geringfügig abweichen, sofern es unseres Erachtens zur Klarheit der Darstellung beiträgt. Einige solcher Punkte sind in Abschnitt 9 aufgelistet.

3 Datentypen in Arden

3.1 Variablen und Datentypen

Eine Variable wird durch ein einzelnes Wort (d. h. eine aus Buchstaben, Zahlen und dem Unterstrich bestehende, mit einem Buchstaben beginnende Zeichenkette der Länge 1 bis 80) bezeichnet und speichert Daten bestimmten Typs. Es gibt die Datentypen *boolean*, *number*, *time*, *duration*, *string*, *list* und *object* sowie *destination*, *message*, *interface*, *mlm* und *event*.

Eine Variable wird in Arden i. d. R. nicht explizit als zu einem bestimmten Datentyp gehörig deklariert; dies geschieht vielmehr im Rahmen der ersten Zuweisung. Eine Variable kann ferner ihren Datentyp dynamisch wechseln; die Bindung an einen Datentyp gilt nur so lange, wie nichts anderes festgelegt worden ist. Wir empfehlen allerdings, diese Tatsache nur dann auszunutzen, wenn die Übersichtlichkeit des Programmtextes nicht darunter leidet, d. h. im Allgemeinen darauf zu verzichten.

Die Typen von Daten, die in einem Arden-Programm vorkommen können, lassen sich wie folgt gliedern.

Zunächst einmal gibt es die internen und die externen Datentypen. Die Inhalte der Variablen internen Typs können im MLM selbst verarbeitet werden; dies sind die grundlegenden Datentypen, und wenn von einem Datentyp ohne Zusatz die Rede ist, ist immer ein interner gemeint. Hierunter fallen insbesondere Zahlen- und boolesche Variablen. Variablen externen Datentyps hingegen speichern Informationen, die vom Host ausgewertet, im MLM selbst aber nicht bearbeitet werden. Dies sind beispielsweise Variablen zur Speicherung der Bezeichnung eines Events oder zur Speicherung vom Host interpretierbarer Nachrichtencodes.

Die internen Datentypen gliedern sich in die einfachen und zusammengesetzten. Im letzteren Fall handelt es sich um endliche Sequenzen von Daten einfachen Typs.

Einer Variable **Var** wird mittels des sogenannten Zuweisungsoperators **:=** ein Wert zugeordnet:

```
Var := /Ausdruck/;
```

oder alternativ

```
let Var be /Ausdruck/;
```

(Hier wie im folgenden stehen nicht explizierte Teile eines Programms kursiv zwischen Schrägstrichen.) Der einer Variablen zukommende Datentyp

wird wie folgt bestimmt. Handelt es sich um einen internen Datentyp, geschieht dies – mit einer Ausnahme – implizit im Rahmen einer Zuweisung wie der vorstehenden. Hierbei liefert */Ausdruck/* auf der rechten Seite einen Wert, dessen Datentyp spätestens zur Laufzeit klar ist, und zwar dadurch, dass es sich um einen konkreten Wert handelt oder eine Funktion, die auf den betreffenden Datentyp abbildet. Man beachte, dass die Bezeichnung des Datentyps hier gar nicht auftaucht.

Die Ausnahme bilden die Variablen des Typs *object*, dessen Struktur vorher festgelegt werden muss; die Details sind im nächsten Abschnitt enthalten.

Die externen Datentypen werden ebenfalls im Rahmen einer Zuweisung deklariert; der Datentyp wird dabei explizit angegeben, und zugewiesen wird stets eine Curly-brace-expression. Handelt es sich etwa um den Datentyp *Externer_Datentyp*, lautet der Befehl:

```
Var := Externer_Datentyp {Anweisung_an_Host};
```

Es folgen die Details zu den einzelnen Datentypen.

3.2 Die einfachen internen Datentypen

In den Abschnitten 3.2 bis 3.4 besprechen wir die internen Datentypen. Wir beginnen mit den einfachen: *boolean*, *number*, *time*, *duration* und *string*. Wie erwähnt erfolgt deren Definition in allen Fällen implizit.

Die einfachen Datentypen sind grundsätzlich zweigliedrig: Außer dem eigentlichen Wert wird ein Zeitpunkt gespeichert, d. h. ein Datum zusammen mit einer Uhrzeit. Zuweisung und Abruf der Hauptkomponente erfolgt über den Variablennamen. Die Details zur Zeitkomponente, die im übrigen in Zusammenhang mit der Verwendung der Hauptkomponente keine Rolle spielen, geben wir im nachfolgenden Abschnitt 3.3.

Der Datentyp *boolean* (boolesch) erlaubt drei Werte: die beiden klassischen Wahrheitswerte sowie einen zusätzlichen Wert, der eingesetzt wird, wenn kein Wahrheitswert zugeordnet werden kann. Einer booleschen Variable kann demgemäß der Wert *true* für „wahr“, *false* für „falsch“ oder *null* für „undefiniert“ zugewiesen sein. Die Zuweisung von „wahr“ oder „falsch“ erfolgt gemäß folgenden Anweisungen:

```
Var := true;
```

bzw.

```
Var := false;
```

Man beachte ein weiteres Mal, dass mittels dieser Zeile die Festlegung auf den Datentyp `boolean` vorgenommen wird. Die Variable `Var` kann vorher bereits verwendet und an einen anderen Datentyp gebunden worden sein; diese Bindung wird hiermit dann hinfällig. Und `Var` kann hier genauso gut zum ersten Mal auftauchen.

Es ist grundsätzlich auch zulässig, einer Variablen `Var` den Wert `null` explizit zuzuordnen. Zu beachten ist jedoch, dass `null` den Zustand der Undefiniertheit auch im Fall aller übrigen Datentypen bezeichnet und insofern dann keine Bindung an einen speziellen Datentyp erfolgt.

Der Datentyp *number* (Zahl) findet sowohl für natürliche als auch für ganze als auch für rationale Zahlen Verwendung; eine *number*-Variable vertritt eine in einem von der Implementation abhängigen Fließkommaformat darstellbare Zahl oder `null` für „undefiniert“. Die Zuweisung eines konkreten Wertes erfolgt gemäß den Mustern

```
Var := 2;  
oder  
Var := -3,463e-20;
```

für die natürliche Zahl 2 bzw. die rationale Zahl $-3,463 \cdot 10^{-20}$.

Der Datentyp *time* (Zeit) ist für die Darstellung von Zeitpunkten vorgesehen. Einer *time*-Variable ist ein Kalenderdatum, eine Uhrzeit und optional eine Zeitzone zugeordnet oder alternativ der Wert `null` für „undefiniert“. Muster für die Zuweisung einer Zeit in Bezug auf die lokale Zeitzone ist das folgende:

```
Var := 2007.11.15t10:43:02.347;
```

Ein an die Zeitangabe angehängtes `z` bedeutet den Bezug auf GMT, und ein angehängtes `+hh:mm` oder `-hh:mm` bedeutet den Bezug auf die Zeitzone, die gegenüber GMT um `hh` Stunden, `mm` Minuten nach vorne bzw. hinten verschoben ist.

Der Datentyp *duration* (Dauer) stellt eine zeitliche Dauer dar; eine solche Variable speichert alternativ eine Anzahl Monate oder eine Anzahl Sekunden oder auch `null` für „undefiniert“. Die Zuweisung konkreter Werte erfolgt nach dem Schema

```
Var :=  
2 years 3 months 4 weeks 4 days 13 hours 4 minutes 0.5 seconds;
```

Es sind für jede Einheit auch gebrochen rationale Werte zulässig; zudem sind auch negative Werte möglich. Wenn es sich um eine glatte Zahl Monate

handelt, wird als Einheit der Monat verwendet, ansonsten die Sekunde. Zu beachten ist dabei natürlich, dass Monate und Jahre nicht von einheitlicher zeitlicher Länge sind; gegebenenfalls werden Mittelwerte genommen.

Der Datentyp *string* (Zeichenkette) dient der Darstellung von Text; eine Zeichenkettenvariable speichert eine Textzeile variabler Länge. Das verwendete Alphabet ist eine der für die verwendete Sprache übliche ASCII-Variante, fürs Deutsche etwa Latin1; eine Maximallänge ist nicht vorgegeben. Die Zuweisung konkreten Textes erfolgt gemäß

```
Var := "Dies ist ein Probetext.";
```

Ein Anführungszeichen innerhalb zu speichernden Textes ist zu verdoppeln. Ein Zeilenumbruch wird als Leerzeichen, eine Leerzeile, die nur in diesem speziellen Fall innerhalb einer Befehlszeile zugelassen ist, als Zeilenumbruch interpretiert.

3.3 Die Zeitkomponente

Jede Variable einfachen internen Typs verfügt über eine Zeitkomponente, die sogenannte *primary time*. Diese wird typischerweise automatisch, kann über spezielle Funktionen aber auch explizit beschrieben werden.

Die *primary time* hat den folgenden Sinn. Werden Daten aus einer Patientendatenbank erhoben, wird standardmäßig automatisch zusammen mit jedem Wert dessen Entstehungszeit übergeben, d. h. etwa wann die zugehörige Messung stattgefunden hat oder einfach wann die entsprechende Eintragung in die Datenbank erfolgt ist. Dies ist der typische Fall dafür, dass die Zeitkomponente benutzt wird, ohne dass es hierzu einer besondere Anweisung im Programm bedürfte. Darüber hinaus steht die Komponente zur freien Verfügung.

Im einzelnen gilt folgendes. Ist **Var** eine Variable vom Typ *boolean*, *number*, *time*, *duration* oder *string*, so gilt für die Zeitkomponente von **Var** dasselbe wie für die Hauptkomponente einer Variable des Typs Zeit: Es handelt sich um ein Datum mit Uhrzeit und allenfalls einer Zeitzone oder alternativ den Wert **null**.

Beschrieben wird die Zeitkomponente zum einen zusammen mit der Hauptkomponente, d. h. im Rahmen eines Zuweisungsbefehls:

```
Var := /Ausdruck eines einfachen internen Typs/;
```

Mit welchem Wert die Zeitkomponente von **Var** beschrieben wird, hängt da-

von `ab`, wie der definierende Ausdruck spezifiziert ist. Handelt es sich etwa um einen extern abgerufenen Wert und ist bei Abruf ein Zeitpunkt mit übergeben worden, wird letzterer in der Zeitkomponente gespeichert; siehe den *read*-Befehl in Abschnitt 7.1. Handelt es sich um eine weitere Variable, wird von dieser auch die Zeitkomponente übernommen. Handelt es sich um eine Funktion anderer Werte, hängt es von dieser Funktion ab, inwieweit deren Zeitkomponenten die von `Var` bestimmen. Im Zweifel wird die Zeitkomponente auf `null` gesetzt; dies geschieht insbesondere, wenn es sich um einen konkreten Wert handelt. Wir setzen uns in dieser Hinsicht aber nicht mit den Details auseinander; bei Bedarf s. [HL7].

Zum anderen kann die Zeitkomponente auch explizit beschrieben werden:

```
time of Var := /Ausdruck des Typs time/;
```

Der Abruf der Zeitkomponente von `Var` erfolgt mittels des Ausdrucks

```
time of Var
```

vom Typ *time*. Dessen Haupt- wie Zeitkomponente enthält jeweils die Zeitkomponente der Variable `Var`. Das Wörtchen *of* kann grundsätzlich auch weggelassen werden.

3.4 Die zusammengesetzten internen Datentypen

In diese Kategorie fallen die sogenannten Listen sowie die Objektvariablen.

Listen. Der Datentyp *list* (Liste) stellt eine endliche Sequenz variabler Länge von Werten einfachen Typs dar. Die Datentypen der Komponenten müssen nicht einheitlich sein; sind sie es doch, heißt die Liste im weiteren homogen.

Die Zuordnung von Werten erfolgt gemäß folgenden Mustern:

```
Var := ();
```

für die Liste der Länge 0,

```
Var := ,Ausdruck1;
```

für eine Liste der Länge 1,

```
Var := Ausdruck1, Ausdruck2, ..., Ausdruckk;
```

für eine Liste der Länge $k \geq 2$.

Dass jeder `Ausdrucki` von einfachem Typ ist, heißt mit anderen Worten, dass Listen nicht geschachtelt werden können. Im Programmtext treten Aneinanderreihungen mehrerer Variablen des Typs *list* einzig im Rahmen von Ein-/Ausgabebefehlen auf; aber auch dann werden mehrere Listen nicht in

einer einzelnen Variable gespeichert.

Die Bezugnahme auf das i -te Listenelement erfolgt durch `Var[i]`.

Objektvariablen. Variablen des Typs *object* enthalten ein n -Tupel von Daten einfachen Typs, wobei n fix ist. Die Komponenten können wiederum beliebigen einfachen Typs sein. Dies ist der einzige Datentyp, der Vorbereitungen erfordert; für eine Variable dieses Typs muss im vorhinein festgelegt werden, um wieviele Komponenten es sich handelt und wie diese bezeichnet sind:

```
Objekttyp := object [Komponent1, ..., Komponenten];  
Var := new Objekttyp;
```

Im Ergebnis sind die n Untervariablen

```
Var.Komponent1, ..., Var.Komponenten;
```

geschaffen, deren jede so zu verwenden ist wie jede andere Variable einfachen Typs auch. Insbesondere erfolgt die Bindung an einen beliebigen einfachen Datentyp für jede Untervariable unabhängig, und zwar wie in Abschnitt 3.2 beschrieben mittels Zuweisung eines Wertes.

Es können mehrere Variablen vom selben Objekttyp deklariert werden. In dem Fall sollte der Typ der i -ten Komponente stets derselbe sein, worin $1 \leq i \leq n$ bei n Komponenten; verlangt ist dies allerdings nicht.

Die Zeitkomponente bei zusammengesetzten Variablen. Jede Komponente einer zusammengesetzten Variable ist einfachen Typs und enthält folglich eine eigene Zeitkomponente. Der `time`-Operator ist dennoch auch auf zusammengesetzte Variablen anwendbar: Sind die Zeitkomponenten aller Komponenten einer solchen Variable, etwa `Var`, gleich, liefert der Ausdruck `time of Var` ebendiesen Zeitpunkt, ansonsten `null`.

Wird umgekehrt `time of Var` beschrieben, werden die Zeitkomponenten aller Komponenten auf den gegebenen Wert gesetzt.

3.5 Die externen Datentypen

Es folgt die Besprechung weiterer vier Datentypen, die der Kommunikation des MLMs mit dem Host-System dienen. Variablen eines dieser Datentypen speichern einen Ausdruck, der vom Host interpretiert wird und dessen Form nicht durch Arden vorgegeben ist. Wir gehen hier nur auf die einfachen externen Datentypen ein; für zusammengesetzte siehe [HL7, 11.2.5.2, 11.2.6.1].

Es handelt sich dann um ein einzelnes Wort, welches, da vom Host auszuwerten, in geschweifte Klammern gesetzt wird. Die Zuweisungen all dieser Variablen muss im Data-Slot erfolgen.

Es handelt sich um die Typen *message*, *destination*, *interface*, *mlm* und *event*. Die Typdefinition erfolgt wiederum im Rahmen der Zuweisung, wobei im Unterschied zu den internen Datentypen der Typ explizit angegeben werden muss.

Eine Variable des Typs *message* enthält den Code einer an den Host weiterzugebenden Nachricht. Die Zuordnung hat die Form:

```
Var := message {Nachricht_XYZ012};
```

Die Variable **Var** dient im weiteren als Argument eines *write*-Befehls, dessen Wirkungsweise in Abschnitt 7.2 erklärt wird.

Eine Variable des Typs *destination* codiert den Empfänger an den Host gesendeter Nachrichten. Es handelt sich beispielsweise um eine E-Mail-Adresse. Die Zuweisung erfolgt gemäß

```
Var := destination {Name_einer_empfangenden_Einheit};
```

Zusammen mit einer *message*-Variable dient auch diese Variable als Argument eines *write*-Befehls.

Eine Variable des Typs *interface* speichert die Bezeichnung einer auf dem Host ausführbaren Routine:

```
Var := interface {Hostroutine};
```

Diese Variable dient im weiteren als Argument eines *call*-Befehls; siehe Abschnitt 7.2.

Eine Variable des Typs *mlm* speichert den Namen eines weiteren MLMs. Die Zuordnung erfolgt gemäß dem Muster

```
Var := mlm {Name_eines_anderen_MLMs};
```

Auch diese Variable kann als Argument des *call*-Befehls verwendet werden.

Variablen des Typs *event* speichern den Namen eines Events. Details und Beispiele zu diesem Datentyp folgen in Abschnitt 6.

3.6 Abfrage des Typs einer Variablen

Da der Typ einer Variable dynamisch wechseln kann, ist die Möglichkeit gegeben, sie auf ihren Typ hin abzufragen:

Var is [not] Datentyp.

Der Ausdruck **Var is Datentyp** liefert **true**, sofern **Var** an den bezeichneten Datentyp gebunden ist und dabei nicht **null** beinhaltet.

Weiter kann eine Variable auch daraufhin abgefragt werden, ob sie undefinierten Inhalts ist, d. h. ihr der Wert **null** zugeordnet ist:

Var is [not] null

oder, negiert,

Var is [not] present

3.7 Vordefinierte Zeitvariablen

Da ein MLM in Echtzeit auf Geschehnisse zu reagieren hat, muss im Arden-Programm eine Orientierung über den aktuellen zeitlichen Rahmen möglich sein. Hierfür sind die folgenden Ausdrücke vom Typ *time* gedacht.

eventtime - falls der MLM als Folge eines Events gestartet worden ist, der Zeitpunkt, zu dem der Host oder das betreffende andere MLM diesen Event signalisiert hat; ansonsten **null**.

triggertime - Zeitpunkt, zu dem die Ausführung des laufenden MLMs veranlasst worden ist. Ist das MLM durch einen Event ausgelöst und ist dabei keine Zeitverzögerung vereinbart worden (siehe Abschnitt 6 unten), stimmt dieser Wert mit **eventtime** überein; ist das MLM durch einen Event ausgelöst und dabei die Zeitverzögerung Δt vereinbart, die **triggertime** um Δt größer als **eventtime**.

now - der tatsächliche Beginn der Ausführung des laufenden MLMs. Nach Veranlassung der Ausführung des MLMs kann eine gewisse Zeit vergangen sein, bis das MLM tatsächlich gestartet wurde: dann nämlich, wenn mehrere MLMs gleichzeitig zu starten waren und andere Vorrang hatten. In diesem Fall ist **now** echt größer als **triggertime**, ansonsten sind die beiden Werte gleich. – Zu beachten ist, dass die Bezeichnung „now“ nicht ganz die passende ist; insbesondere ändert sich der Wert im Verlauf der Ausführung des MLMs nicht.

currenttime - die aktuelle Uhrzeit. Diese wird im Verlauf der Ausführung des MLMs ständig angepaßt.

`midnight` - 0 Uhr des auf den laufenden Tag folgenden Tages.

4 Operationen in Arden

Wir listen im folgenden die in Arden zur Verfügung stehenden Operationen mit Argumenten einfachen Datentyps auf.

Die Beschreibung gilt jeweils den Hauptkomponenten. Für die Zeitkomponenten gilt: Stimmen diese für alle Argumente überein, wird mit diesem Wert auch die Zeitkomponente des Ergebnisses beschrieben, ansonsten mit `null`.

Wird weiter eine Operation mit Argumenten des falschen Datentyps aufgerufen oder ist für die betreffenden Argumente nicht erklärt, wird auf `null` abgebildet. Eine Fehlermeldung wird nicht generiert.

4.1 Boolesche Werte

Operationen mit booleschen Werten. Für den Datentyp *boolean* stehen die folgenden Operationen zur Verfügung: Negation `not`, Konjunktion `and`, Disjunktion `or`, Äquivalenz `=` und dessen Negation `<>`. Eingeschränkt auf die beiden Wahrheitswerte sind dies die üblichen booleschen Operationen. Ist weiter bei einer binären Operation eines der Argumente `null`, hängt das Ergebnis der Operation von diesem aber nicht ab, wird dieses Ergebnis verwendet; andernfalls wird auf `null` abgebildet.

Für `not` ebenso wie für die im weiteren aufgelisteten unären Operationen gilt die Präfixschreibweise ohne Notwendigkeit, Klammern zu setzen. Für die binären Operationen gilt die Infixschreibweise.

Operationen mit Listen boolescher Werte. Ist `BoolescheListe` eine Liste boolescher Variablen, ist `all BoolescheListe` die Konjunktion, `any of BoolescheListe` die Disjunktion aller Werte, `no BoolescheListe` die Konjunktion der negierten Werte.

4.2 Zahlen

Operationen auf Zahlen. Die folgenden Operationen für den Datentyp *number* sind verfügbar: Addition `+`, Subtraktion `-`, Multiplikation `*`, Division `/`, Negation `-`, Potenzierung `**`. Die Syntax orientiert sich an in der

Mathematik Üblichen, auch was die Bindungsstärken anbelangt. Es können Klammern verwendet werden: (,).

Es stehen weiter die folgenden Funktionen zur Verfügung: Absolutwert `abs`, Quadratwurzel `sqrt`, Exponentialfunktion $x \mapsto e^x$ `exp`, Logarithmus zur Basis e `log` und zur Basis 10 `log10` sowie die trigonometrischen Funktionen `sine`, `cosine`, `tangent`, `arcsin`, `arccos`, `arctan`.

Weiter gibt es verschiedene Funktionen zur Konvertierung einer rationalen Zahl zu einer ganzen. Es sei `Zahl` eine *number*-Variable. `int Zahl` ergibt die nächstkleinere ganze Zahl, `ceiling Zahl` die nächstgrößere, `truncate Zahl` schneidet die Kommastellen ab, `round Zahl` rundet zur nächstgelegenen ganzen Zahl bzw. im Fall des Nachkommateils ,5 zur betragsmäßig (!) größeren ganzen Zahl.

Operationen mit Listen von Zahlen. Es sei `Zahlenliste` eine Liste von Zahlen. Dann ist `sum Zahlenliste` die Summe der Werte, `average Zahlenliste` der Durchschnitt, `median Zahlenliste` der Medianwert, `stddev Zahlenliste` die Standardabweichung und `variance Zahlenliste` die Streuung.

Vergleichsoperatoren. Funktionen, die Paare von Zahlen auf einen booleschen Wert abbilden, sind die Vergleichsoperatoren: Gleichheit `=`, Ungleichheit `<>`, kleiner `<=`, echt kleiner `<`, größer `>=`, echt größer `>`.

Sortieroperationen für Listen von Zahlen. Es sei `Zahlenliste` eine Liste von Zahlen. Dann ist `sort data Zahlenliste` oder `sort Zahlenliste` die Liste mit denselben Einträgen, die nun jedoch der Größe nach geordnet sind. Weiter ist `sort time Zahlenliste` die Liste mit zeitlich geordneten Einträgen, sofern alle Zeitkomponenten definiert sind; bei Überstimmung der zeitlichen Komponenten entscheidet die Hauptkomponente.¹

`sort time (Zahlenliste1, Zahlenliste2)`

ist abkürzbar zu

`Zahlenliste1 merge Zahlenliste2`.

Das Extremum in Bezug auf Haupt- oder, falls vorhanden, die Zeitkomponente liefert: `minimum Zahlenliste`, `maximum Zahlenliste`, `earliest Zahlenliste` bzw. `latest Zahlenliste`. Bei Uneindeutigkeit wird der weiter vorn in der Liste stehende Eintrag genommen. Den zugehörigen Index

¹Möchte man eine Liste allein der der Größe nach geordneten Zeitkomponenten der einzelnen Einträge erzeugen, muss man ausnahmsweise Klammern setzen: `sort (time [of] Zahlenliste)`.

dieser Werte erhält man durch Hinzufügen des Wortes `index`, etwa `index minimum Zahlenliste`.

Den Eintrag, dessen Zeitkomponente einem vorgegebenen Zeitpunkt am nächsten kommt, erhält man mit `nearest Zeitpunkt from Zahlenliste`, den zugehörigen Index wiederum durch Voranstellen von `index`.

Schließlich sei `Zahlzeitliste` eine Liste von *number*-Variablen, deren Zeitkomponenten alle definiert sind. Indem diese als Funktion von den betreffenden Zeitpunkten in die reellen Zahlen aufgefasst wird, lässt sich die lineare Regression bestimmen: `slope Zahlzeitliste`. Zu- und Abnahme des zweiten, dritten usw. Wertes im Vergleich zum vorhergehenden lässt sich absolut oder prozentual bestimmen: `increase Zahlzeitliste`, `decrease Zahlzeitliste`, `%increase Zahlzeitliste`, `%decrease Zahlzeitliste`; dies ergibt naturgemäß jeweils eine Liste mit einer um 1 verringerten Komponentenzahl. `interval Zahlzeitliste` ist ferner die Liste der Differenzen je zweier aufeinanderfolgender Zeitkomponenten.

4.3 Zeit und Dauer

Vergleichsoperatoren. Die Vergleichsoperatoren für Zahlen stehen auch für Paare von Zeitpunkten zur Verfügung mit der Bedeutung „früher“ bzw. „später“: Übereinstimmung `=`, Verschiedenheit `<>`, früher `<=` oder `is not after`, echt früher `<` oder `is before`, später `>=` oder `is not before`, echt später `>` oder `is after`.

Schließlich gibt es die Vergleichsoperatoren `=`, `<>`, `<=`, `<`, `>=`, `>` auch für Paare von Werten des Typs *Dauer* mit der Bedeutung „kürzer“ bzw. „länger“.

Weitere Vergleichsfunktionen fassen zwei der bislang aufgeführten zusammen und betreffen Beginn und Ende einer Zeitperiode; die folgenden Ausdrücke vom Datentyp *boolean* sind selbsterklärend:

```
Zeitpunkt1 is [not] within Zeitpunkt2 and Zeitpunkt3
Zeitpunkt1 is [not] within Dauer following Zeitpunkt2
Zeitpunkt1 is [not] within Dauer surrounding Zeitpunkt2
Zeitpunkt1 is [not] within past Dauer
Zeitpunkt1 is [not] within the same day as Zeitpunkt2
```

Ist hierin `Zeitpunkt1` Zeitkomponente einer *number*-Variable, etwa der Variable `Zahl`, sind letztere Ausdrücke abkürzbar; statt `time of Zahl is ...` kann geschrieben werden `Zahl occurred ...` oder `Zahl occurs ....`

Funktionen von Zeit und Dauer. Zu einer *time*-Variable kann eine Zeit-

dauer addiert werden:

Dauer from Zeitpunkt

oder

Dauer after Zeitpunkt

oder

Zeitpunkt + Dauer

ist der Zeitpunkt um Dauer später als Zeitpunkt;

Dauer before Zeitpunkt

oder

Zeitpunkt Dauer ago

oder

Zeitpunkt - Dauer

ist der um Dauer frühere Zeitpunkt.

Des weiteren können zwei Werte des Typs Dauer addiert und voneinander subtrahiert werden.

Bei allen Rechnungen, in denen Zeitlängen, d. h. Werte des Typs *duration*, vorkommen, ist eine gewisse Vorsicht dadurch geboten, dass die Einheiten Monat und Jahr nicht eindeutig sind. Das Problem wird pragmatisch gelöst; für die Details siehe [HL7, 8.5.2.3]. Im wesentlichen muss beachtet werden, dass, falls Monate in die Rechnung eingehen, das Ergebnis nur auf drei Tage genau ist.

4.4 Manipulation von Zeichenketten

Sind *Text1*, ..., *Textn* Zeichenketten, d. h. Ausdrücke des Typs *string*, ist

Text1 || ... || *Textn*

die durch Aneinanderhängen entstehende Zeichenkette. Weiter ist dieser Ausdruck auch für Argumente anderer Datentypen definiert. Ist etwa *Texti* nicht vom Typ *string*, wird *Texti* zunächst in eine Zeichenkette konvertiert. Details des Konvertierungsvorganges können durch Zugabe von **formatted with ...** gesteuert werden; siehe [HL7, 9.8.2].

Ist weiter *Textliste* die Liste (*Text1*, ..., *Textn*), liefert

string *Textliste*

dasselbe wie *Text1* || ... || *Textn*.

Ist *Text* eine Zeichenkette, ist **extract characters** *Text* die der Reihe

nach die einzelnen Buchstaben von `Text` enthaltende Liste.

Die Länge von `Text` ist `length Text`. Weiter ist

`substring n characters starting at m in Text`

diejenige Zeichenkette, die aus `Text` durch Abschneiden der ersten $m - 1$ Zeichen und anschließendes Abschneiden aller der Zeichen ab dem $n + 1$ -ten hervorgeht.

Vergleichsoperatoren. Für Zeichenketten stehen einerseits die Vergleichsoperatoren Gleichheit `=` und Ungleichheit `<>` zur Verfügung.

Weiter sind Vergleiche bezüglich der lexikographischen Ordnung möglich, wobei sich die Details an den Üblichkeiten (etwa dem Duden) orientieren: kleiner `<=`, echt kleiner `<`, größer `>=`, echt größer `>`.

Es ist möglich zu prüfen, ob eine Zeichenkette `Text` die Unterzeichenkette `Unterzeichenkette` enthält:

`Text matches pattern "%Unterzeichenkette%"`

ergibt den entsprechenden Wahrheitswert und

`find Unterzeichenkette in Text`

ergibt die Position von `Unterzeichenkette` in `Text` bzw. den Wert `null`.

5 Bearbeitung von Listen in Arden

Wie beschrieben stellt eine Liste ein n -Tupel von Variablen einfachen Typs dar. Im folgenden sind diejenigen Operationen mit Listen beschrieben, die sich auf deren Komponenten beziehen ohne Bezug auf die Inhalte.

Manipulation von Listen. Sind `Liste1` und `Liste2` zwei Ausdrücke des Typs `Liste`, ist

`NeueListe := Liste1, Liste2;`

die Liste, die durch Aneinanderhängen von `Liste1` und `Liste2` entsteht.

Weiter sei `Liste` die Liste (`Komponente1`, ..., `Komponenten`). Dann steht `first Liste` für `Komponente1` und `last Liste` für `Komponenten`. `reverse Liste` die Liste (`Komponenten`, ..., `Komponente1`). Weiter wird durch

`Liste := p seqto q;`

worin $1 \leq p \leq q \leq n$ ist, die Liste $p, p + 1, \dots, q$ definiert.

Es ist `count Var` die Länge der Liste, und `exists Liste` gibt Auskunft, ob

die Liste nicht leer ist.

Operationen mit Listenkomponenten. Es sei `Liste1` eine homogene Liste und `EinstOp` eine einstellige Funktion auf dem Datentyp der Listenkomponenten. Dann ist

`EinstOp Liste1`

diejenige Liste, die aus `Liste1` durch komponentenweise Anwendung von `EinstOp` hervorgeht. Ähnlich sei `Liste2` eine weitere homogene Liste derselben Länge wie `Liste1` und `ZweistOp` eine zweistellige Operation, dessen erstes Argument vom selben Typ wie in `Liste1` und dessen zweites Argument vom Typ wie in `Liste2` ist. Dann ist

`Liste1 ZweistOp Liste2`

die durch komponentenweise Anwendung von `ZweistOp` aus `Liste1` und `Liste2` hervorgehende Liste.

Des weiteren sei `Wert` von einfachem Datentyp, `Liste` eine n -komponentige homogene Liste und `ZweistOp` eine zweistellige Operation mit Argumenten vom Typ wie `Wert` und `Liste`. Dann ist `Wert ZweistOp Liste2` die Liste, die sich durch Anwendung von `ZweistOp` auf `Wert` und die einzelnen Listenelemente ergibt. Analoges gilt hinsichtlich eines Ausdrucks `Liste ZweistOp Wert`.

Der Selektionsoperator. Sind `Datenliste` und `BoolescheListe` Listen gleicher Länge und enthält `BoolescheListe` nur boolesche Variablen, dann stellt

`Datenliste where BoolescheListe`

die Liste dar, die aus `Datenliste` dadurch hervorgeht, dass jede Komponente `Datenliste[i]` gestrichen wird, falls `boolescheListe[i]` nicht gleich `true` ist.

Weiter sei `Datenliste` die Liste (`Komponente1`, ..., `Komponenten`) und `BoolescherAusdruck(Argument)`

ein boolescher Ausdruck, der ein Argument enthält. Dann wird durch

`Liste where BoolescherAusdruck(it)`

die folgende Liste definiert: Liefert für $i = 1, \dots, n$ der Ausdruck

`BoolescherAusdruck(Komponentei)`

nicht den Wert `true`, wird `Komponentei` aus `Liste` gestrichen, ansonsten übernommen.

Es ist zu beachten, dass die *where*-Funktion stärker bindet als die übrigen Listenfunktionen.

Zugehörigkeit zu Liste. Ist *Var* eine Variable und *Liste* eine Liste, ist der Ausdruck *Var is in Liste* wahr, falls für ein *i* *Liste[i]* vom Typ von *Var* ist und beide übereinstimmen.

6 Die Auslösung von MLMs

6.1 Events

Start durch Host mittels Events. Unter einem Event wird ein Ereignis auf Seiten des Hosts verstanden. Die Events sind hostseitig spezifiziert und jeweils als einzelnes Wort codiert.

Es bezeichne etwa *Bestellung_von_Medikament_XY* einen Event. Stellt nun der Host fest, dass der Event *Bestellung_von_Medikament_XY* eingetreten ist – wie etwa immer dann, wenn das *Medikament_XY* bestellt und die Datenbank dementsprechend modifiziert worden ist –, wird das entsprechende Signal ausgesendet. In der Folge werden alle MLMs gestartet, für die festgelegt ist, dass sie auf diesen Event zu reagieren haben.

Diese Festlegung erfolgt im MLM selbst. Im Data-Slot eines MLMs wird hierfür zunächst einmal eine Variable des Typs *event* definiert:

```
Auslösendes_Ereignis := event {Bestellung_von_Medikament_XY};
```

Sodann ist im Evoke-Slot aufzuführen, dass das MLM bei Eintreten des in *Auslösendes_Ereignis* gespeicherten Events die Ausführung beginnen soll:

```
evoke: Auslösendes_Ereignis;;
```

Es kann alternativ auch festgelegt werden, dass nach Eintreten des Events eine gewisse Zeitdauer, etwa *Dauer*, abgewartet wird:

```
evoke: Dauer after time of Auslösendes_Ereignis;;
```

Schließlich kann das MLM auch periodisch wiederholt ausgelöst werden; es gilt das Muster:

```
evoke: every Zeitperiode for Dauer
      starting time of Auslösendes_Ereignis;;
oder
evoke: every Zeitperiode
      starting time of Auslösendes_Ereignis
```

```
until Abbruchbedingung;;
```

Die Periode ist dann durch den Wert in **Zeitperiode** gegeben. Der Abbruch erfolgt im ersten Fall, wenn nach der ersten Auslösung die durch **Dauer** gegebene Zeitlänge verstrichen ist. Im zweiten Fall wird abgebrochen, sobald die Auswertung des booleschen Ausdrucks **Abbruchbedingung** vor einer etwaigen Auslösung des MLMs nicht mehr **true** ergibt.

Weiter kann die Ausführung des MLMs auch alternativ durch eines von mehreren Events ausgelöst werden. Werden etwa im Data-Slot die Event-Variablen **Auslösendes_Ereignis1**, ..., **Auslösendes_Ereignisk** definiert, bedeutet

```
evoke: Auslösendes_Ereignis1 or ... or Auslösendes_Ereignisk;;
```

dass das MLM startet, falls einer der betreffenden Events signalisiert worden ist. In diesem Fall ist eine verzögerte oder periodische Ausführung nicht möglich.

Start durch ein anderes MLM mittels Events. Das Eintreten eines Events kann nicht nur vom Host, sondern auch von einem anderen MLM behauptet werden. Dies geschieht durch folgende Befehlszeile in letzterem:

```
call Ereignis;
```

wobei **Ereignis** eine im Data-Slot definierte Event-Variable ist. Auf Seiten der betroffenen MLMs gilt das Vorstehende unverändert.

6.2 Direkter Aufruf eines MLMs

Der Host kann jederzeit ein spezifisches MLM unter Verwendung des im Maintenance-Slot festgelegten Namens direkt aufrufen.

Dasselbe kann jedes weitere MLM tun, und zwar mit dem Befehl

```
call 'Name_des_MLMs';
```

oder, wenn **Var** eine Variable des Typs *mlm* ist, die den MLM-Namen speichert:

```
call Var;
```

Wie in diesem Fall Daten übergeben und empfangen werden, wird im folgenden Abschnitt 7.3 erklärt.

6.3 Ausführungsprioritäten

Ergibt sich ein übereinstimmender Startzeitpunkt für mehrere MLMs, legt der Wert im Priority-Slot fest, welches MLM zuerst startet; der Slot hat die Form

```
priority:  $p$ ;;
```

worin p eine rationale Zahl zwischen 1 und 99 ist und ein höherer Wert eine höhere Priorität darstellt. Falls der Slot fehlt, wird vom Wert 50 ausgegangen. Sind die Prioritäten gleich, wird eine Pseudozufallswahl getroffen.

Die MLMs werden jeweils nur bis einschließlich zum Logic-Slot ausgeführt. Die Reihenfolge der Abarbeitung dann noch anstehender Action-Slots wird mittels des Parameters *urgency* bestimmt; siehe Abschnitt 8.2.

7 Datenein- und -ausgabe

7.1 Daten vom Host einlesen

Daten werden vom Host mit einem Befehl der folgenden Form angefordert:

```
Var := read [Listenfunktion] {/Anweisung an Host/}  
[where Bedingung(it)];
```

Hierin steht in geschweiften Klammern eine Anweisung zum Lesen gewisser Daten, z. B. aus einer Datenbank, wofür institutionsspezifische Syntax zur Anwendung kommt. Das Ergebnis ist auf Seiten des Arden-Programms vom Typ einer homogenen Liste. Diese Liste kann mittels der *where*-Bedingung und/oder des Operators *Listenfunktion* verändert werden; es ergibt sich ein Einzelwert oder eine Liste, welche an *Var* übergeben wird.

Die eingelesene Liste darf auch aus Werten vom Typ *object* bestehen. Dieser Typ muss dann bereits definiert sein, etwa als *Objekttyp*, und das Wort *read* ist durch den Ausdruck *read as Objekttyp* zu ersetzen.

Weiter ist die Möglichkeit zugelassen, dass mit einem einzelnen *read*-Befehl mehrere Listen eingelesen werden. Die institutionsspezifische Anweisung an die Datenbank besagt dann, dass n Listen zu senden sind, worin $n \geq 2$ fix ist. Die linke Seite des *read*-Befehls ist in diesem Fall durch einen Ausdruck (*Var1*, ..., *Var n*) zu ersetzen.

7.2 Aufruf von Funktionen im Host

Nachrichten. An den Host kann eine Textnachricht gesendet werden. Der Befehl lautet

```
write Text;
```

worin *Text* vom Typ *string* ist, oder

```
write Nachricht;
```

worin *Nachricht* eine *message*-Variable ist.

In diesen beiden Beispielen geht die Nachricht an die Standardausgabeeinheit. Optional kann in jedem solchen Befehl aber auch eine empfangende Einheit mit angegeben werden. Es sei *Zieladresse* eine *destination*-Variable; dann kann der *write*-Befehl durch *at* erweitert werden:

```
write Text at Zieladresse;
```

bzw.

```
write Nachricht at Zieladresse;
```

Host-Routinen. Eine Host-Routine, deren Bezeichnung in einer Variable des Typs *interface*, etwa *Var*, gespeichert ist, wird mithilfe des *call*-Befehls aufgerufen:

```
call Var;
```

Es können dabei Daten übergeben werden:

```
call Var with Wert_1, ..., Wert_n;
```

worin die Variablen *Wert_i* von beliebigem internen Typen sind; die Typen sind von der entsprechenden Host-Routine vorzugeben.

7.3 Datentransfer zwischen MLMs

Ein MLM kann mit einem *call*-Befehl ein anderes ausführen lassen. Dabei können, wenn gewünscht, Daten übergeben und empfangen werden; zu diesem Zweck wird der *call*-Befehl nach dem folgenden Muster verallgemeinert:

```
ZuErhaltenderWert :=
```

```
    call MLMName with ZuÜbergebenderWert;
```

oder

```
(ZuErhaltenderWert1, ..., ZuErhaltenderWertm) :=
```

```
    call MLMName
```

```
        with ZuÜbergebenderWert1, ..., ZuÜbergebenderWertn;
```

Hierin ist `MLMName` eine *mlm*-Variable und die übrigen Variablen von je beliebigem internen Typ, auch Listen sind zugelassen.

Im aufgerufenen MLM ist im Data-Slot die folgende Zeile zwecks Entgegennahme der Daten einzufügen:

```
ErhaltenerWert := argument
```

bzw.

```
(ErhaltenerWert1, ..., ErhaltenerWertn) := argument
```

Die Rückgabe von Werten an das aufrufende MLM erfolgt mit dem *return*-Befehl:

```
return ZurückzugebenderWert
```

bzw.

```
return (ZurückzugebenderWert1, ..., ZurückzugebenderWertm)
```

Dass die passende Zahl von Werten des richtigen Typs übergeben und zurückgenommen werden, muss der Programmierer selbst sicherstellen; eine Kontrolle durch den Compiler erfolgt nicht. Zu Laufzeiten wird zu viel Übergebenes ignoriert und Fehlendes oder Daten vom falschen Typ als `null` behandelt.

8 Das Arden-Programm

8.1 Entscheidungspunkte

Verzweigungspunkte eines Arden-Programmes sind von der selbsterklärenden Form

```
if Bedingung then /Programmblock/ endif;
```

oder

```
if Bedingung then /Programmblock_1/
```

```
else /Programmblock_2/ endif;
```

oder, für jedes $n \geq 1$,

```
if Bedingung1 then /Programmblock_1/
```

```
elseif Bedingung2 then /Programmblock_2/
```

```
elseif . . .
```

```
else /Programmblock_n+1/ endif;
```

Weiter sind Programmschleifen möglich:

```
for Eintrag in Parameterliste do ... enddo;  
while Bedingung do ... enddo;
```

Im ersten Fall wird der Variable `Eintrag` der Reihe nach die Einträge der Liste `Parameterliste` zugewiesen. Im zweiten Fall wird der Programmblock wiederholt abgearbeitet, solange `Bedingung` den Wert `true` liefert, andernfalls übersprungen.

8.2 Entscheidung über Konsequenzen

Ein Arden-Programm zielt dabei darauf ab, zu evaluieren, ob der Action-Slot abgearbeitet werden soll oder nicht. Dies wird beim ersten Antreffen einer Zeile der folgenden Form entschieden:

```
conclude Bedingung;
```

Hat dann `Bedingung` nicht den Wahrheitswert `true`, wird die Ausführung des MLMs beendet. Andernfalls wird umgehend ein Sprung zum Beginn des Action-Slots durchgeführt.

Ist neben dem laufenden MLM der Start noch weiterer MLMs zum selben Zeitpunkt veranlasst worden, werden alle bis zu genau diesem Punkt ausgeführt. Diejenigen, die nicht abbrechen, sondern noch den Action-Slot abzuarbeiten haben, werden sodann hinsichtlich ihrer Priorität neu sortiert. Hierzu dient die reservierte Variable `urgency`. Deren Wert kann im Verlauf des Data- oder Logic-Slots mit einem die Dringlichkeit beschreibenden Wert zwischen 1 und 99 beschrieben werden; ein höherer Wert steht für größere Dringlichkeit.

9 Ergänzung: Änderung bei der Darstellung gegenüber [HL7]

In mindestens den folgenden Punkten weicht unsere Darstellung von der offiziellen Spezifikation [HL7] ab. Wir betonen aber, dass es sich nur um eine andere Sprechweise handelt und inhaltlich keine Unterschiede beabsichtigt sind.

- Gemäß [HL7] gibt es die Unterscheidung von internen und externen Datentypen nicht; letztere gelten nicht als Datentypen.
- In [HL7] ist `null` ein eigener Datentyp.

Des weiteren sind einige Feinheiten bei der Beschreibung weggelassen; entsprechende Hinweise sind im Text erfolgt.

10 Grundsätzliches über FuzzyArden

FuzzyArden [Tif] stellt eine konservative Erweiterung von Arden dar; in Arden geschriebene Programme sollen auch unter FuzzyArden lauffähig sein und eine unveränderte Funktionsweise haben.

Ziel von FuzzyArden ist es, die Bearbeitung von Informationen zu ermöglichen, mit denen sich Unbestimmtheit verbindet. Die Ursache kann Vagheit - englisch: *fuzziness* - sein, wie sie typisch ist für natürlichsprachlich ausgedrückte Eigenschaften eines Gegenstandes. Es kann sich ebenso gut auch um Unsicherheit hinsichtlich eines Faktums handeln, insbesondere um die Unschärfe einer Größe, wie sie sich infolge einer ungenauen Messung ergibt.

Kern der Erweiterung ist die Verallgemeinerung des booleschen Datentyps; statt der beiden Werte **true** und **false** werden alle (darstellbaren) Zahlen zwischen 0 und 1 zugelassen. Es bedeutet 1 das klare Zutreffen, 0 das Nichtzutreffen einer Eigenschaft, und falls eine Eigenschaft nicht klar zutrifft oder nicht zutrifft, wird eine Tendenz durch Werte im offenen Einheitsintervall ausgedrückt.

Erreicht werden soll auf diese Weise, bei den von einem MLM generierten Empfehlungen sprunghaftes Verhalten zu vermeiden. Im Fall eines typischen Arden-Programms verursacht eine kleine Änderung in der Eingabe möglicherweise ein völlig anderes Ergebnis – ganz einfach deswegen, weil Grenzen gewöhnlich scharf gezogen werden. Sinn der Erweiterung von FuzzyArden ist, ohne nennenswerten programmiertechnischen Mehraufwand weiche, d. h. kontinuierlich verlaufende und nicht abrupt auftretende Grenzen spezifizieren zu können und eine stetige Abhängigkeit des Ergebnisses von der Eingabe zu erreichen.

Das sich dem Anwender anbietende Ergebnis kann dabei auf zwei verschiedene Weisen ausgestaltet werden. Entweder werden gewichtete Konsequenzen angeboten; aus einer Vielzahl möglicher Konsequenzen anhand der zugeordneten Gewichtungen die konkrete Reaktion abzuleiten bleibt in diesem Fall dem Anwender überlassen. Oder aber es wird eine einzelne, mehrere mögliche Konsequenzen je nach deren Gewichtung zusammenfassende Reaktion generiert.

Wir fügen an, dass über die Natur der Wahrheitswerte von Seiten FuzzyArdens keine eigentlichen Vorgaben gemacht werden. Ein in einem MLM verwendeter Wahrheitswert kann sich auf einen graduellen Übergang welcher Art auch immer beziehen; sich darüber im klaren zu sein, welcher Aspekt relevant ist und wie die betreffenden Begriffe genau zu verstehen sind, ist

Sache des Programmierers.

Wir beschreiben, was sich in FuzzyArden gegenüber Arden im einzelnen ändert. Wir lehnen uns so weit wie möglich an die Vorgabe [Tif] an, bringen jedoch auch unsere eigenen Vorschläge ein; einige Abweichungen sind im abschließenden Abschnitt 15 aufgelistet.

11 Änderungen bei den Datentypen in FuzzyArden

Der boolesche Datentyp wird verallgemeinert; und für die Datentypen *number*, *time* und *duration* werden fuzzifizierte Analoga neu eingeführt.

Weiter erhalten in FuzzyArden alle einfachen internen sowie die externen Datentypen eine Zusatzkomponente, die als Grad der Verwendbarkeit des betreffenden Wertes zu interpretieren ist.

11.1 Der Datentyp Wahrheitswert

Der Datentyp *boolean* erlaubt nunmehr kontinuierliche Wahrheitswerte; zugelassen sind alle Zahlen zwischen 0 und 1 sowie `null`. Den Werten `false` und `true` entspricht 0 bzw. 1. Da die Bezeichnung boolesch nicht mehr notwendigerweise die passende ist, heißt dieser Datentyp alternativ auch *truth value* (Wahrheitswert). Dieser Datentyp wird weiterhin implizit deklariert. Bei der Zuweisung eines konkreten Wertes wird zur Unterscheidung der Wahrheitswerte von Zahlen das Schlüsselwort `truth value` angefügt:

```
Var := truth value 0;
oder alternativ
Var := false;
Var := truth value 0.667;
Var := truth value 1;
oder alternativ
Var := true;
```

11.2 Die Datentypen Fuzzyzahl, Fuzzyzeit und Fuzzydauer

Weiter wird als Pendant zu den Typen *number*, *time* und *duration* je ein Datentyp neu eingeführt, der Fuzzymengen über dem jeweiligen Grundbereich

repräsentiert: *fuzzy number* (Fuzzyzahl), *fuzzy time* (Fuzzyzeit) bzw. *fuzzy duration* (Fuzzydauer). Gegenüber dem ursprünglichen Typ wird jeweils die Hauptkomponente verändert: Anstelle eines scharfen Wertes wird eine Fuzzymenge gespeichert. Diese muss stückweise linear, an jedem Punkt entweder links- oder rechtsseitig stetig und außerhalb eines endlichen Intervalles konstant sein.

Bei Bedarf können die ursprünglichen Datentypen gegen die neuen durch Zugabe des Wortes *crisp* abgegrenzt werden. Mit anderen Worten heißt der Datentyp *number* alternativ auch **crisp number**, der Datentyp *time* auch **crisp time** und der Datentyp *duration* auch **crisp duration**.

Die Deklaration der neuen Typen erfolgt im Rahmen der ersten Zuweisung unter Zugabe des Schlüsselwortes **fuzzy set**. Es werden diejenigen Funktionswerte notiert, zwischen denen linear zu interpolieren ist. Ist $u : \mathbb{R} \rightarrow [0, 1]$ stetig, geht die Definition wie folgt. Es sei $a_1 < a_2 < \dots < a_n$, $u(x) = t_1$ für $x \leq a_1$, u auf allen Intervallen $[a_1, a_2], \dots, [a_{n-1}, a_n]$ linear, $u(a_2) = t_2, \dots, u(a_{n-1}) = t_{n-1}$, und $u(x) = t_n$ für $x \geq a_n$. Dann lautet die Programmzeile zur Definition dieser Fuzzymenge:

Fuzzymenge_u := fuzzy set $(a_1, t_1), (a_2, t_2), \dots, (a_n, t_n);$

Im allgemeinen sind Fuzzymengen $u : \mathbb{R} \rightarrow [0, 1]$ folgenden Typs zugelassen. Für gewisse $a_1 < a_2 < \dots < a_n$ sei u auf $(-\infty, a_1)$ konstant l_1 , auf (a_n, ∞) konstant r_n , auf den offenen Intervallen $(a_1, a_2), \dots, (a_{n-1}, a_n)$ linear und am Punkt a_i , $i = 1, \dots, n$, wahlweise gleich dem linksseitigen Grenzwert l_i oder gleich dem rechtsseitigen Grenzwert r_i von u .

Die Definition erfolgt in FuzzyArden wie vorstehend mittels einer Liste, jedoch mit den notwendigen Modifikationen. Ist u bei einem der Punkte a_i stetig, d. h. $u(a_i) = l_i = r_i$, ändert sich nichts:

Fuzzymenge_u := fuzzy set $\dots, (a_i, l_i), \dots;$

Ist u bei a_i linksstetig, d. h. $u(a_i) = l_i$, wird erst der linksseitige Grenzwert – der Funktionswert – und dann der rechtsseitige Grenzwert angegeben:

Fuzzymenge_u := fuzzy set $\dots, (a_i, l_i), (a_i, r_i), \dots;$

Ist schließlich u bei a_i rechtsstetig, wird dasjenige Paar, das den Funktionswert angibt, verdoppelt:

Fuzzymenge_u := fuzzy set $\dots, (a_i, l_i), (a_i, r_i), (a_i, r_i), \dots;$

Ein simples Beispiel möge diese Syntax verdeutlichen.

Fuzzymenge := fuzzy set $(0, 0), (0, 0.5), (0, 0.5), (1, 1), (1, 0);$

ist die Fuzzyzahl $u: \mathbb{R} \rightarrow [0, 1]$, die gegeben ist durch $u(x) = 0$ für $x < 0$ und $x > 1$ sowie $u(x) = \frac{x+1}{2}$ für $0 \leq x \leq 1$.

Handelt es sich um eine symmetrische triangulare normale Fuzzymenge, kann statt des Ausdrucks **fuzzy set** $(a-b, 0), (a, 1), (a+b, 0)$ verkürzt **a fuzzified by b**

geschrieben werden.

Für die Typen *fuzzy time* und *fuzzy duration* gelten mutatis mutandis all diese Vorgaben ebenfalls. Die Abgrenzung gegen den Typ *fuzzy number* erfolgt klarerweise anhand der ersten Komponente jedes Paares durch das entsprechende Format bzw. die Zugabe von Zeiteinheiten.

Typische Beispiele wären hier Ausdrücke der folgenden Art:

three days ago fuzzified by 12 hours
sowie
fuzzy set (10 years, 0), (13 years, 1),
(19 years, 1), (19 years, 0)

Im ersten Fall handelt es sich um eine triangulare Fuzzymenge, die ihr Maximum bei demjenigen Zeitpunkt hat, der sich aus **now** durch Subtraktion von drei Tagen ergibt. Im zweiten Fall handelt es sich um die Fuzzydauer, die etwa ein Alter zwischen 13 und 19 Jahren modelliert, wobei die Untergrenze unscharf gehalten ist, jedoch eine strikte Obergrenze gilt.

11.3 Die Wahrheitswertkomponente

Allen einfachen internen Datentypen, einschließlich der neu hinzugekommenen Fuzzyversionen, sowie den externen Datentypen wird in FuzzyArden eine weitere Komponente zugegeben: Zu den bisherigen Komponenten – dem eigentlichen, allenfalls unscharfen, Wert und ggf. dem diesem zugeordneten Zeitpunkt – kommt eine zusätzliche hinzu, in der ein Wahrheitswert gespeichert wird.

Diese Wahrheitswertekomponente drückt allenfalls die eingeschränkte Verwendbarkeit des in der Hauptkomponente vorliegenden Wertes aus. Ist etwa W dieser Wert und der assoziierte Wahrheitswert $t \in [0, 1]$, so gilt: Der Wert W wurde unter einer bestimmten Bedingung gewonnen oder wird unter einer bestimmten Bedingung betrachtet, und diese Bedingung trifft zum Grade t zu. Dementsprechend heißt im weiteren t der **Verwendbarkeitsgrad** bzw. *degree of applicability* von W .

Ein Verwendbarkeitsgrad von 1 drückt aus, dass für die Verwendung des Wertes keine Einschränkung bekannt ist, und wird daher als Standardwert gesetzt. Das bedeutet: Wird der Verwendbarkeitsgrad bei der Definition der Variablen nicht spezifiziert oder ist eine Zuweisung ungültig, wird er auf 1 gesetzt. Ein Verwendbarkeitsgrad von 0 bedeutet, dass mit dem Wert der Hauptkomponente überhaupt nichts anzufangen ist; dies ist insbesondere der Fall, wenn die Hauptkomponente undefiniert ist, d. h. `null` enthält. Im übrigen ist der Verwendbarkeitsgrad stets ein Wert zwischen 0 und 1; `null` ist nicht zugelassen.

Die folgenden Fälle sind die prototypischen; zugehörige Details folgen an gegebener Stelle weiter unten.

- Es soll eine Aussage über den Verlauf einer Messgröße innerhalb der vergangenen 24 Stunden gewonnen werden, ausgehend von einer Liste mit Werten aus den vergangenen Tagen. Die Grenze von 24 Stunden ist willkürlich und insofern nicht scharf; man möchte daher wie folgt vorgehen. Man lässt alle Werte, die älter als 28 Stunden sind, außer acht. Den späteren Werten ordnet man hingegen einen sukzessive größer werdenden Verwendbarkeitsgrad zu. Letzterer kann in der folgenden Rechnung dazu verwendet werden, den betreffenden Messwert entsprechend zu gewichten.
- Das FuzzyArden-Programm gelangt an einen Punkt, an dem die weitere Ausführung von einer Bedingung abhängt; die Bedingung trifft jedoch weder eindeutig zu noch nicht zu, sondern habe vielmehr den Wahrheitswert $b \in (0, 1)$. In diesem Fall werden beide Zweige ausgeführt; die Verwendbarkeitsgrade der verarbeiteten Variablen werden jedoch entsprechend erniedrigt, sprich mit b bzw. $1 - b$ multipliziert, und im Fall der Aggregation wieder aufaddiert.
- Die Wahrheitswertkomponente steht auch zur freien Verfügung des Programmierers. Insbesondere ist es möglich, bei der Initialisierung einer Variablen diese von vornherein mit einem von 1 verschiedenen Verwendbarkeitsgrad zu versehen. Es ist an den Fall gedacht, dass zusammen mit Werten aus einer Patientendatenbasis nicht nur ein Zeitpunkt, sondern auch ein Wahrheitswert mit übergeben werden kann, der dessen Zuverlässigkeit widerspiegelt.

Der Wahrheitswertkomponente lässt sich explizit ein Wert zuweisen:

`applicability of Var := /Ausdruck des Typs Wahrheitswert/;`

Abgerufen wird der Wahrheitswert über denselben Ausdruck:

`applicability of Var`

Dieser ist vom Typ *truth value*; die Hauptkomponente enthält den Wahrheitswert von `Var`, die Zeitkomponente wird unverändert übernommen, und die eigene Wahrheitswertkomponente ist 1.

Die zusammengesetzten Datentypen *list* und *object* ändern sich in FuzzyArden nicht, von der Tatsache abgesehen, dass in ihnen nun die modifizierten sowie die neuen einfachen Datentypen erscheinen.

Ist `Var` von zusammengesetztem Typ, lässt sich mittels `applicability of Var := ...` die Wahrheitswertkomponente aller Komponenten von `Var` mit demselben Wert beschreiben. Umgekehrt liefert `applicability of Var` das Infimum der Wahrheitswerte der Komponenten von `Var`.

12 Änderungen bei den Operationen in FuzzyArden

Zur Spezifikation gewisser Optionen im Zusammenhang mit dem Verrechnen von Fuzzymengen wird ein weiterer Abschnitt im MLM eingeführt; der Abschnitt *fuzzy options* wird zwischen *library* und *knowledge* eingeschoben. Hier kann in jeweils eigenen Slots die der Konjunktion von Wahrheitswerten dienende Funktion sowie die Defuzzifizierungs- und die Aggregationsmethode festgelegt werden.

Ansonsten gelten für die Behandlung der Zeitkomponente sowie für das Verhalten im Fehlerfall die gleichen Regeln wie in Arden.

12.1 Wahrheitswerte

Operationen mit booleschen Werten. Die Rolle der Konjunktion übernimmt eine t-Norm. Es soll die Möglichkeit bestehen, die Wahl der t-Norm selbst zu treffen; dies geschieht im Fuzzy-options-Abschnitt in einem Slot mit der Bezeichnung `connectives`, in dem eine Zeile folgender Form stehen muss:

`conjunction by TNorm;`

Hierbei ist `TNorm` entweder durch die Bezeichnung einer fest implementierten t-Norm zu ersetzen: durch `Lukasiewicz`, `Product` oder `Goedel`. Oder `TNorm`

ist von der Form `mlm 'EigeneTNorm'`, worin `EigeneTNorm` der Name eines MLMs ist. Mit diesem MLM, das zwei Argumente des Typs Wahrheitswert annehmen und ein solches zurückgeben muss, kann der Programmierer die t-Norm selbst programmieren. – Wird die Konjunktion nicht spezifiziert, wird die gödelsche t-Norm verwendet.

Weiter ist im Zweifelsfall die Disjunktion die zur t-Norm gehörige t-Konorm. Es ist aber auch eine abweichende eigene Definition möglich; eine Zeile

```
disjunction by mlm 'EigeneTKonorm';
```

muss dann im Connectives-Slot stehen, worin wiederum `EigeneTKonorm` der Name eines MLMs ist. Schließlich ist die Negation, wenn nichts anderes vereinbart ist, die Subtraktion von 1. Alternativ kann ein eigenes MLM für diese Operation verfasst werden, in welchem Fall im Connectives-Slot ein Eintrag

```
negation by mlm 'EigeneNegation';
```

auftauchen muss.

Folgende Regelung gilt im Fall, dass eines der Argumente den Wert `null` hat. Es seien `Var1` und `Var2` Variablen des Typs Wahrheitswert, und etwa `Var1` sei undefiniert, d. h. beinhalte `null`:

- `Var1 and Var2` ergibt 0, falls `Var2` den Wert 0 hat, ansonsten `null`.
- `not Var1` ergibt `null`.
- `Var1 or Var2` ergibt `null`, falls `Var2` 0 oder `null` beinhaltet, und ansonsten `Var2`.

Im Einklang mit der Vorgabe von Seiten Ardens gilt demgemäß für die Konjunktion und Negation: Hängt das Ergebnis einer logischen Operation von einem der Argumente nicht ab, wird dieses Ergebnis verwendet, und zwar unabhängig davon, ob das keine Rolle spielende Argument `null` ist oder nicht.

Im Fall der Disjunktion ist hingegen folgende Besonderheit zu beachten: Wird ein Wahrheitswert $t \in (0, 1)$ mit dem Wert `null` oder-verknüpft, ist das Resultat t . Der Grund für diese Festlegung ist die folgende. Wir gehen davon aus, dass im Regelfall die Disjunktion der Kumulation mehrerer Wahrheitswerte dient und dabei undefinierte Werte das Ergebnis nicht aufheben, sondern stattdessen unberücksichtigt bleiben sollen. Dies ist eine Annahme, die sich für das Expertensystem Cadiag-2 [AdKo, AKSEG, AKSG] bewährt

hat und selbstverständlich nicht für alle Anwendungen sinnvoll ist. Im Bedarfsfall kann aber, wie vorstehend angegeben, die Disjunktion ohne großen Aufwand undefiniert werden; von einer vordefinierten Konjunktion kann dabei Gebrauch gemacht werden.

Die Operatoren mindestens, höchstens. Neu eingeführt werden die Operatoren **at least** und **at most**. Ist **Anzahl** eine die natürliche Zahl n enthaltende *number*-Variable und sind **Var1**, ..., **Var k** Wahrheitswertvariablen, wobei $k \geq n$ ist, wird der Ausdruck

at least Anzahl of (Var1, ..., Var k)

als die Disjunktion aller Konjunktion über n der k Werte **Var1**, ..., **Var k** interpretiert. Ist als Konjunktion das Minimum, d. h. die gödelsche t -Norm, festgelegt, ist dies der n -tgrößte Wert unter den **Var i** . Zu interpretieren ist dieser Wert als Verallgemeinerung der Aussage „Mindestens n der k Bedingungen sind richtig“. – Ist die Bedingung $n \leq k$ nicht erfüllt, ist das Resultat immer **false**.

Analog ist

at most Anzahl of (Var1, ..., Var k)

erklärt. Der Verwendbarkeitsgrad wird auf das Minimum der Verwendbarkeitsgrade von **Var1**, ..., **Var k** gesetzt.

Für die beiden Operationen sind allerdings auch alternative Interpretationen denkbar. Enthält etwa **Anzahl** den Wert n und die Variablen **Var1**, ..., **Var k** die Werte $v_1, \dots, v_k$, so kann man den Ausdruck **at least Anzahl of Var1, ..., Var k** beispielsweise auch mittels

$$\frac{(v_1 + \dots + v_k) \wedge n}{n}$$

auswerten. Für eigene Definitionen ist im Connectives-Slot die Zeile

at least by mlm ‘Eigenes_atleast’;

bzw.

at most by mlm ‘Eigenes_atmost’;

anzugeben. Die MLMs **Eigenes_atleast** bzw. **Eigenes_atmost** nehmen hierfür eine *number*-Variable und eine Liste von Variablen des Typs Wahrheitswert an und geben einen Wert des Typs Wahrheitswert zurück.

12.2 Zahlen

Operationen auf Zahlen. Rechnungen mit (scharfen) Zahlen, sei es einzelnen, Paaren oder ganzen Listen, sind definiert wie bisher. Für Fuzzyszahlen werden die Addition und die Subtraktion gemäß Zadehschem Erweiterungsprinzip [1] neu eingeführt, ebenso die Multiplikation mit einem und die Division durch einen scharfen Wert.

Als Verwendbarkeitsgrad wird der kleinste der einzelnen Argumente genommen.

Die übrigen Operationen, die für Zahlenwerte erklärt sind, werden nicht auf Fuzzyszahlen verallgemeinert, schlicht deshalb, weil die Gesamtheit der zur Verfügung stehenden Fuzzymengen unter den kanonisch verallgemeinerten Operationen nicht abgeschlossen ist.

Vergleichsoperatoren. Der Vergleich zweier Zahlen hinsichtlich der natürlichen Ordnung erfolgt wie bislang, und zwar unter Ignorierung des Verwendbarkeitsgrades.

Neu eingeführt wird der Vergleich einer scharfen mit einer triangularen Fuzzymenge:

`Var is FuzzyVar, Var <= FuzzyVar, Var >= FuzzyVar`

Hier ist `Var` vom Typ *number*; und `FuzzyVar` ist vom Typ *fuzzy number*, hat bei expliziter Spezifizierung also etwa die Form `fuzzy set (a1, t1), ..., (an, tn)` oder `a fuzzified by b`. Es sei u die in `FuzzyVar` gespeicherte Fuzzymenge.

Der Ausdruck `Var is FuzzyVar` liefert denjenigen Wahrheitswert, auf den u den in `Var` gespeicherten Wert abbildet.

Weiter definieren wir für eine beliebige Fuzzymenge $v: \mathbb{R} \rightarrow [0, 1]$

$$v^{\rightarrow}: \mathbb{R} \rightarrow [0, 1], \quad x \mapsto \bigvee \{v(y) : y \leq x\},$$

$$v^{\leftarrow}: \mathbb{R} \rightarrow [0, 1], \quad x \mapsto \bigvee \{v(y) : y \geq x\}.$$

Der Ausdruck `Var >= FuzzyVar` ist dann der Wahrheitswert, auf den $u^{\rightarrow}$ `Var` abbildet, und `Var <= FuzzyVar` ist derjenige, auf den $u^{\leftarrow}$ abbildet.

Im Fall der Relation `is not` gilt: Es wird der Wahrheitswert, der sich im Fall `is` ergibt, negiert, d. h. im Standardfall von 1 abgezogen. Mit anderen Worten wird ein Ausdruck `Var is not FuzzyVar`, worin `FuzzyVar` vom Typ *fuzzy number* sei, wie

`not(Var is FuzzyVar)`

interpretiert. Schließlich wird ein Ausdruck `Var < FuzzyVar` wie

`(Var <= FuzzyVar) and (Var is not FuzzyVar)`

evaluiert; und Entsprechendes wird für die Relation `>` erklärt.

Auch der Wahrheitswert, der sich durch den Vergleich zweier scharfer oder einer scharfen mit einer Fuzzyzahl ergibt, erhält einen Verwendbarkeitsgrad, und zwar einfach den kleineren der beiden verglichenen Ausdrücke.

Die Sortieroperatoren für Listen von Zahlen werden wie gehabt definiert. Der Verwendbarkeitsgrad bleibt für jede Komponente unverändert und geht in die Bestimmung der neuen Reihenfolge nicht ein.

12.3 Zeit und Dauer

Operationen sowie Vergleiche von Werten der Typen *time*, *duration* und ihrer Fuzzyanaloga werden in analoger Weise erklärt wie für den Typ *number*. Die arithmetischen Operationen sind dabei natürlich nur dann erklärt, wenn sie für die entsprechenden scharfen Datentypen definiert sind.

12.4 Abfrage des Typs einer Variablen

Wie in Arden kann eine Variable auf ihren Typ hin abgefragt werden.

Var is Datentyp

ist so wie bisher definiert, wobei **Datentyp** für jeden der bisherigen oder neuen Datentypen stehen kann. Man beachte, dass **boolean** und **truth value** äquivalente Ausdrücke sind; ob eine Wahrheitswertvariable einen der Werte 0 oder 1 enthält, fragt man mit `(Var = true)` or `(Var = false)` ab.

Var is fuzzy

liefert den Wert **true**, falls **Var** vom Typ **fuzzy number**, **fuzzy time**, **fuzzy duration** oder **truth value** (bzw. **boolean**) ist und sonst **false**. Analog liefert

Var is crisp

den Wert **true**, falls **Var** vom Typ **crisp number** oder **crisp time** oder **crisp duration** ist und sonst **false**.

Alle gerade erklärten Ausdrücke können durch den Einschub von **not** negiert werden.

12.5 Defuzzifizierung

Eine Fuzzymenge kann in einen scharfen Wert umgewandelt werden. Es sei **Fuzzymenge** eine Variable des Typs *fuzzy number*, *fuzzy time* oder *fuzzy duration*; dann liefert der Ausdruck

defuzzified Fuzzymenge

einen Wert vom Typ *number*, *time* bzw. *duration*.

Die Methode der Defuzzifizierung kann im Fuzzy-options-Abschnitt festgelegt werden. Hierzu muß ein Slot *defuzzification* angelegt werden und die folgende Zeile enthalten:

defuzzify by Defuzzifizierungsmethode;

Hier ist **Defuzzifizierungsmethode** entweder durch die Ausdrücke **mean of maximum** oder **centre of gravity** zu ersetzen, die die implementierten Methoden bezeichnen. Oder aber es wird der Name eines MLMs angegeben, der ein Argument eines der Fuzzydatentypen annimmt und einen Wert des entsprechenden scharfen Datentyps zurückgibt.

Im Fall der *Mean-of-maximum*-Methode wird der Durchschnitt der Mittelpunkte der Intervalle berechnet, die auf das Supremum des Bildes der Fuzzymenge abbilden. Handelt es sich nicht ausschließlich um endliche Intervalle, wird **null** genommen. Das *Centre-of-Gravity*-Verfahren berechnet den Schwerpunkt der Fläche zwischen Abszisse und Graph der Fuzzymenge. Ist hierbei der Träger der Fuzzymenge nicht endlich oder ist leer, wird **null** genommen.

Wird kein Verfahren angegeben, kommt die Standardmethode zum Einsatz: *Centre of Gravity*.

Die Zeit- und die Wahrheitswertekomponente übernimmt der Operator **defuzzified** vom Argument.

12.6 Linguistische Ausdrücke

Die fuzzifizierten Versionen der Typen *number*, *time* und *duration* finden ihre primäre Verwendung in Vergleichen. Enthält etwa **ScharferWert** die Zahl x und enthält **Fuzzymenge** die Fuzzymenge u , ergibt gemäß Abschnitt 12.2

ScharferWert is Fuzzymenge,
ScharferWert >= Fuzzymenge,

ScharferWert $\leftarrow$ Fuzzymenge

denjenigen Wahrheitswert, auf den x von u bzw. $u^{\rightarrow}$ bzw. $u^{\leftarrow}$ abgebildet wird.

Die klare Empfehlung an den Programmierer bei der Verwendung eines der Fuzzydatentypen geht nun dahin, Fuzzymengen nie einzeln, sondern stets im Verbund mit weiteren zu verwenden, zusammen mit denen die Aufteilung eines Wertebereiches definiert wird.

Vorgesehen ist das folgende Vorgehen. Gegeben sei ein Parameter, gespeichert im FuzzyArden-Programm in der Variable **Parameter**, der Werte aus einem abgeschlossenen (möglicherweise unbeschränkten) Intervall $W \subseteq \mathbb{R}$ annehmen kann; weiter seien u_1, u_2, u_3 Fuzzymengen über W , die die Wertebereiche „niedrig“, „mittel“, „hoch“ in diesem Intervall repräsentieren. Dabei sei (u_1, u_2, u_3) eine Teilung der 1 auf W , d. h. es mögen sich u_1, u_2 und u_3 auf W punktweise zu 1 aufaddieren; außerhalb von W sind die Fuzzymengen z. B. auf 0 zu setzen. In solchem Fall gilt es, diese drei Fuzzymengen gemeinsam in einer einzelnen Variable des Typs *object* zu speichern, deren Komponenten nach den Wertebereichen benannt sind, etwa:

```
Wertebereiche := object [Niedrig, Mittel, Hoch];  
Wert := new Wertebereiche;  
Wert.Niedrig := /Definition der Fuzzymenge  $u_1$ /;  
Wert.Mittel := /Definition der Fuzzymenge  $u_2$ /;  
Wert.Hoch := /Definition der Fuzzymenge  $u_3$ /;
```

Ob dann **Parameter**, der eine Zahl aus W enthalte, einen niedrigen, mittleren oder hohen Wert hat, kann in der Folge mittels der Bedingungen

```
Parameter is Wert.Niedrig  
bzw.  
Parameter is Wert.Mittel  
bzw.  
Parameter is Wert.Hoch
```

bestimmt werden, die drei Wahrheitswerte liefern, deren Summe 1 ist. Diese können im Logic-Slot zur unscharfen Fallunterscheidung verwendet werden; die Details folgen unten.

Um die Bedeutung von FuzzyArden-Programmteilen dieses Typs suggestiver zu machen, wird vereinbart, dass eine Objektvariable, deren Komponenten alle vom Typ *fuzzy number* oder *fuzzy time* oder *fuzzy duration* sind, statt durch das Schlüsselwort **object** alternativ auch durch **linguistic variable** gekennzeichnet werden darf. Ein Befehl der Form

Wertebereich :=

```
linguistic variable [niedrig, hoch];
```

definiert demgemäß die beiden Variablen `Wertebereich.niedrig` und `Wertebereich.hoch` und impliziert, dass diese von einem der Fuzzydatentypen sind.

Weiter sei daran erinnert, dass bei Vergleichen eines scharfen Wertes mit einer Fuzzymenge das Schlüsselwort `is` verwendet wird.

Wir fügen ein konkretes Beispiel an. Es sei `AntonsAlter` eine Variable des Typs `Dauer`, und es sei u_1, u_2, u_3 die Fuzzymenge, die den Altersbereich „jung“ bzw. „mittleren Alters“ bzw. „alt“ repräsentiert. Es muss $u_1(A) + u_2(A) + u_3(A) = 1$ für jede Dauer A gelten, die ein Alter repräsentieren kann. Dann ist der passende Programmabschnitt für den Data-Slot der folgende:

Altersbereich :=

```
linguistic variable [Jung, MittlerenAlters, Alt];
```

```
Alter := new Altersbereich;
```

```
Alter.Jung := fuzzy set
```

```
(0 years, 1), (25 year, 1), (35 years, 0);
```

```
Alter.MittlerenAlters := fuzzy set
```

```
(25 years, 0), (35 years, 1), (65 years, 1), (75 years, 0);
```

```
Alter.Alt := fuzzy set
```

```
(65 years, 0), (75 years, 1);
```

Ist nun die Variable `AntonsAlter` ihrer Bezeichnung gemäß zu interpretieren, liefert

```
AntonsAlter is Alter.Jung.
```

einen Wahrheitswert, der angibt, zu welchem Grad die Aussage gerechtfertigt ist, dass Anton jung sei; enthält `AntonsAlter` etwa den Wert 31, ergibt sich, dass Anton zum Grad 0,4 der Alterklasse junger Menschen zuzurechnen ist.

13 Bearbeitung von Listen in FuzzyArden

Ein weiteres Element von FuzzyArden ist der verallgemeinerte Selektionsoperator `where`. Es seien die folgenden Listen gegeben:

```
Liste := Wert1, ..., Wertn;
```

```
Bedingungen := Wahrheitswert1, ..., Wahrheitswertn;
```

Hierin sind die **Wert i** Variablen beliebigen einfachen Typs und die Variablen **Wahrheitswert i** vom Typ *truth value*, d. h. eine Zahl aus dem reellen Einheitsintervall oder **null** enthaltend. Dann ist

Liste where Bedingung

die folgende Liste. Für jedes $i \in \{1, \dots, n\}$ wird der Verwendbarkeitsgrad des Eintrages **Wert i** mit **Wahrheitswert i** und-verknüpft; für die Undverknüpfung gilt wie üblich die Deklaration im Connectives-Slot des Fuzzyoptions-Abschnitts. Der Eintrag wird gestrichen, falls das Ergebnis 0 oder **null** ist, andernfalls mit dem Ergebnis als neuem Verwendbarkeitsgrad versehen.

Zugehörigkeit zu Liste. Ist **Var** eine Variable und **Liste** eine Liste, die den in **Var** gespeicherten Wert enthält und zudem mit einem Wahrheitswert t versehen ist, liefert der Ausdruck **Var is in Liste** diesen Wert t und ansonsten 0.

14 Das FuzzyArden-Programm

14.1 Aufspaltung an Verzweigungspunkten

Die Art des Programmablaufes ändert sich in FuzzyArden gegenüber Arden wesentlich. Dies ergibt sich aus dem Verhalten bei Programmverzweigungen, die von einer unscharfen Bedingung abhängen. Der Programmlauf ist nunmehr nicht mehr notwendig linear; es sind Aufspaltungen in mehrere parallele Zweige möglich.

Die Syntax des *If-then*-Befehls bleibt die gleiche, in der allgemeineren Form etwa

```
if Bedingung then /Programmblock_1/
else /Programmblock_2/ endif;
```

In FuzzyArden ist **Bedingung** vom Typ *truth value* und kann dementsprechend jeden echt zwischen 0 und 1 liegenden Wert enthalten. Ist dieser 0 oder 1, wird vorgegangen wie bisher; und der Wert **null** wird stets mit 0 identifiziert.

Enthält **Bedingung** hingegen einen Wert $b \in (0, 1)$, geschieht folgendes. Es werden alle bislang deklarierten Variablen dupliziert und mit je einem der beiden Variablensätze parallel sowohl */Programmblock_1/* als auch */Programmblock_2/* ausgeführt. Wir sprechen im weiteren auch von zwei **Pro-**

grammzweigen, die gleichzeitig abgearbeitet werden. Die Verwendbarkeitsgrade aller Variablen von */Programmblock_1/* werden vor der Ausführung mit b und diejenigen von */Programmblock_2/* mit $1 - b$ multipliziert.

Ferner wird Sorge getragen, dass der Verwendbarkeitsgrad jeder im Verlauf von */Programmblock_1/* veränderten Variable kleiner oder gleich der gerade geltenden Gewichtung ist. Eine Gewichtung $g \in [0, 1]$ gilt bei der Abarbeitung jeder Programmzeile eines Arden-Programms gemäß folgendem Konzept. Das Programm startet mit $g = 1$. Trifft das Programm auf einen Verzweigungsbefehl, während die Gewichtung g gilt, und wird die Bedingung zu b berechnet, setzt der erste Programmzweig mit der Gewichtung $g \cdot b$ und der zweite mit der Gewichtung $g \cdot (1 - b)$ fort.

Wird nun, während die Gewichtung g gilt, eine Variable verändert und ihrer Wahrheitswertkomponente t zugewiesen, so wird dieser Wert, wenn nötig, auf $g \wedge t$ abgesenkt. Handelt es sich insbesondere um eine neu eingeführte Variable, die mit einem expliziten Wert beschrieben wird, ergibt sich ein Verwendbarkeitsgrad von g .

Des weiteren ist die Anzahl der Programmzweige, in die sich ein Fuzzy Arden-Programm aufteilen kann, nicht auf zwei begrenzt. Eine Befehlszeile der Form

```
if Bedingung1 then /Programmblock_1/
elseif Bedingung2 then /Programmblock_2/
. . .
elseif Bedingungn /Programmblock_n/
else /Programmblock_n + 1/ endif;
```

wird nämlich nicht als Verschachtelung von *If-then*-Bedingungen interpretiert, sondern als Aufspaltung in $n + 1$ gleichberechtigte Programmzweige. Es gilt alles bisher für den Fall $n = 2$ Erklärte mutatis mutandis auch für den Fall $n > 2$. D.h., die Variablen werden ver- $(n + 1)$ -facht und alle $n + 1$ Programmblöcke parallel bearbeitet. Enthält dann **Bedingung i** den Wert $b_i \in (0, 1]$, werden im i -ten Zweig die Verwendbarkeitsgrade und die Gewichtung mit b_i multipliziert; im Fall $b_i = 0$ wird der Block nicht abgearbeitet. Ist weiter die Summe aller b_i echt kleiner 1, geschieht das gleiche mithilfe des Wertes $b_{n+1} = 1 - b_1 - \dots - b_n$ im letzten Block, der im Fall $b_{n+1} \leq 0$ ausgelassen wird. Wiederum wird im Fall, dass eine Variable **Bedingung i** null ist, von $b_i = 0$ ausgegangen.

Falls die Bedingungen alle von der Form **Var = Wert i** bzw. **Var is Wert i** sind, worin **Var** eine einzelne Variable und **Wert i** Zahlen bzw. Fuzzymengen

sind, kann man alternativ wie folgt schreiben:

```
switch Var
  case Wert1 /Programmblock_1/
    . . .
  case Wertn /Programmblock_n/
  default /Programmblock_n + 1/
endswitch;
```

Der Sinn liegt allein darin, größere Übersichtlichkeit zu schaffen.

14.2 Die Zusammenführung von Programmzweigen

Nach Abschluss der parallelen Abarbeitung mehrerer Programmblöcke als Folge einer Verzweigung gemäß uneindeutiger Bedingung stehen dem Programmierer zwei Möglichkeiten offen: Entweder bleibt das Programm bis zum Schluss aufgespalten, oder die Programmzweige werden wieder zusammengeführt.

Für einen gegebenen *If-then*-Konstrukt gilt die erstgenannte Möglichkeit im Zweifel; eine Kennzeichnung dieser Option ist nicht notwendig. Es wird dann das gesamte verbleibende Arden-Programm parallel mehrfach ausgeführt. Die Kommunikation mit dem Host erfolgt von jedem Programmzweig aus unabhängig; das Host-System muss dann zwingend so eingerichtet sein, dass die Verwendbarkeitsgrade der empfangenen Daten mit ausgewertet werden.

Sollen hingegen die Programmzweige eines *If-then*-Konstruktes wieder zusammengeführt werden, muss dem *endif*- bzw. *endswitch*-Befehl das Schlüsselwort **aggregate** beigegeben werden, etwa:

```
if Bedingung then /Programmblock_1/
else /Programmblock_2/ endif aggregate;
```

Die Programmgewichtung wird dann auf die Summe der Gewichtungen der einzelnen Zweige gesetzt, d. h. auf den Wert, der vor der Aufspaltung gegolten hat.

Weiter werden die in den einzelnen Programmzweigen im Allgemeinen verschiedenen Inhalte einer jeden Variable aggregiert; das Vorgehen ist das folgende.

Es sei **Var** eine in mindestens einem der Zweige existente Variable. Zunächst im Hinblick auf die Hauptkomponente wird wie folgt verfahren. Hat **Var** in allen Zweigen übereinstimmenden Inhalt, wird dieser übernommen. Ist an-

dernfalls **Var** in allen Zweigen vom selben einfachen Datentyp, nicht jedoch vom Typ *string* und zudem in keinem Zweig gleich **null**, wird ein Aggregationsoperator aufgerufen, und **Var** erhält dessen Ergebnis.

Ist **Var** in allen Zweigen eine Liste, wird so der Reihe nach mit der ersten, zweiten, ... Komponente verfahren; ist dann die *i*-te Komponente nicht mehr in allen Zweigen vorhanden, werden die Komponenten ab der *i*-ten verworfen. Ist **Var** in allen Zweigen von ein und demselben *object*-Typ, wird mit den einzelnen Komponenten so verfahren wie beschrieben.

In den verbleibenden Fällen wird **Var** auf **null** gesetzt.

Als zu verwendende Aggregationsoperator kann zum einen ein im ganzen MLM gültiger Standard definiert werden, und zwar für die scharfen und die Fuzzy-Datentypen getrennt. Im Data-Slot muss in Bezug auf die scharfen Daten eine Zeile der folgenden Form stehen:

crisp aggregation by Aggregationsmethode;

Für die Aggregation von Fuzzymengen lautet die Zeile:

fuzzy aggregation by Aggregationsmethode;

Zum anderen kann mit jedem Schlüsselwort **aggregate** die Aggregationsmethode individuell festgelegt werden, dann allerdings für scharfe und Fuzzydaten gemeinsam. Das Muster lautet

```
if Bedingung then /Programmblock_1/
else /Programmblock_2/ endif aggregate by Aggregationsmethode;
```

Aggregationsmethode steht entweder für eines der zur Verfügung stehenden Verfahren. Für scharfe Daten wird nur eine Methode implementiert, und zwar **weighted mean**. Im Fall der Fuzzyanalogie steht zur Auswahl **weighted mean** sowie **supremum**. Alternativ können hier die Namen von MLMs angegeben werden; diese müssen eine Liste der relevanten Datentypen annehmen und einen scharfen Wert des jeweils selben Typs zurückgeben.

Ist **weighted mean** gesetzt, werden Zahlen oder Fuzzymengen $r_1, \dots, r_n$ mit dem Verwendbarkeitsgrad $t_1, \dots, t_n$ aggregiert zu

$$\frac{t_1 r_1 + \dots + t_n r_n}{t_1 + \dots + t_n}.$$

Gilt für die Fuzzymengen-Datentypen die Option **supremum**, wird der punktweise zu verstehende Ausdruck

$$(u_1 \wedge \bar{t}_1) \vee \dots \vee (u_n \wedge \bar{t}_n)$$

gebildet, worin $u_1, \dots, u_n$ die Fuzzymengen, $t_1, \dots, t_n$ die zugehörigen Gewichtungen und $\bar{t}_1, \dots, \bar{t}_n$ konstant auf $t_1, \dots, t_n$ abbildenden Fuzzymengen sind.

Als Standard gilt das Verfahren **weighted mean**.

Ist die Zeitkomponente in allen Zweigen gleich, wird sie übernommen und andernfalls auf **null** gesetzt.

Der Verwendbarkeitsgrad des aggregierten Wertes wird auf die Summe der Verwendbarkeitsgrade der Komponenten gesetzt. Man beachte, dass die Summe 1 nicht übersteigen kann, da ein Verwendbarkeitsgrad stets höchstens so groß sind wie die Programmzweiggewichtung. Weiter ist klar, dass der Verwendbarkeitsgrad, falls er in jedem der Zweige unangetastet bleibt, wieder denselben Wert hat wie vor der Aufspaltung.

Wir geben ein einfaches Beispiel. Es gelten die n Deklarationen für die *linguistic variable* **Alter**. Ähnliches gelte für eine Variable **Dosis**; es seien Fuzzymengen **Dosis.Niedrig**, **Dosis.Mittel**, **Dosis.Hoch** erklärt.

```
switch AntonsAlter
  case Alter.Jung
    EinzunehmendeMenge := Dosis.Niedrig;
  case Alter.MittlerenAlters
    EinzunehmendeMenge := Dosis.Mittel;
  default
    EinzunehmendeMenge := Dosis.Hoch;
endswitch aggregate;
Abzumessen := defuzzfied EinzunehmendeMenge;
```

Ist beispielsweise das Alter 27, wird der erste Zweig mit einer Gewichtung von 0,8, der zweite mit einer Gewichtung von 0,2 ausgeführt. Dies heißt, dass im ersten Zweig der Variable **EinzunehmendeMenge**, sofern diese vorher nicht schon in Verwendung war, die Fuzzyzahl **Dosis.Niedrig** mit Verwendbarkeitsgrad 0,8 zugeordnet wird, im zweiten Zweig **Dosis.Mittel** zum Grad 0,2. Der dritte Zweig wird nicht ausgeführt; somit sind die Wert für den ersten und zweiten zu aggregieren. Je nach gewählter Option wird von **Dosis.Niedrig** und **Dosis.Mittel** das gewichtete Mittel berechnet oder das Supremum der abgeschnittenen Fuzzymengen gebildet. Mittels des abschließenden Befehls im Beispiel wird schließlich das Ergebnis defuzzifiziert; ist die Methode hierfür die Standardmethode, wird nach dem *Centre-of-gravity*-Verfahren vorgegangen. – Man beachte, dass hier das Verhalten eines Mamdani-Fuzzycontrollers auf einfache Weise imitiert werden kann.

Alternativ ist natürlich auch denkbar, dass **Dosis.Niedrig**, **Dosis.Mittel**,

`Dosis.Hoch` scharfe Werte sind. Dann ist das Vorgehen ganz genauso, entfällt lediglich die letzte Programmzeile.

14.3 Die Bedingung für den Action-Slot

Ob der Action-Slot ausgeführt wird oder nicht, hängt in Arden davon ab, ob im Logic-Slot der `conclude true` ausgeführt wird oder nicht. In FuzzyArden ist die Möglichkeit gegeben, die Ausführung des Action-Slots graduell zu beschränken.

Konkret wird wie folgt vorgegangen. Es gilt eine virtuelle Variable des Typs Wahrheitswert als deklariert, die ebenfalls mit *conclude* bezeichnet wird. Diese Variable kann auf einen beliebigen Wert gesetzt werden, und zwar mittels des Befehls

```
conclude Wahrheitswert;
```

In Arden hat diese Zeile die Bedeutung „Hat `Wahrheitswert` den Wert `true`, führe den Action-Slot aus, andernfalls beende das Programm“. In FuzzyArden wird der oben beschriebenen Modifikation des *If-then*-Befehls entsprechend verfahren. Das bedeutet folgendes. Enthält `Wahrheitswert` einen von 0 und `null` verschiedenen Wert, wird ein Sprung zum Beginn des Action-Slots durchgeführt und werden die Verwendbarkeitsgrade der Variablen sowie die Programmgewichtung mit `Wahrheitswert` multipliziert. Unabhängig von dem Wert von `Wahrheitswert` endet die Ausführung des Programmzweiges.

Im Action-Slot steht der Wert von *conclude* unter diesem Namen zur freien Verfügung. Das Verhalten kann etwa in Abhängigkeit einzelner Wertebereiche festgelegt werden, etwa nach dem Schema

```
if conclude <= 0.1 then
...
elseif conclude <= 0.2
. . .
endif;
```

14.4 Ergänzende Anmerkungen zu Parallelausführung und Zusammenführung

Wir geben im Folgenden einige praktische Hinweise mit dem Ziel, etwaige Schwierigkeiten mit der in FuzzyArden neu eingeführten Möglichkeit zu

vermeiden, das Programm in Abhängigkeit von unscharfen Bedingungen zweigen zu lassen.

Zur Aufsplittung in mehr als zwei Programmzweige. Wir merken warnend an, dass die Unterscheidung von n Bedingungen außer über die wiederholte Verwendung des Schlüsselwortes *elseif* im Prinzip auch durch n ineinander geschachtelte *if*-Bedingung möglich ist; die folgende Konstruktion ist nicht verboten:

```
if Bedingung1 then /Programmblock_1/
else if Bedingung2 then /Programmblock_2/
. . .
else if Bedingungn /Programmblock_n/
else /Programmblock_n + 1/ endif ... endif;
```

Hier steht das Schlüsselwort **endif** am Ende n -mal. Eine Gewichtung von 1 am Anfang vorausgesetzt und mit den vorhergehenden Bezeichnungen, ergibt sich dann für den ersten Programmblock eine Gewichtung von t_1 , für den zweiten von $(1 - t_1) \cdot t_2$ usw., für den n -ten von $(1 - t_1) \cdot \dots \cdot (1 - t_{n-1}) \cdot t_n$ und für den letzten von $(1 - t_1) \cdot \dots \cdot (1 - t_n)$. Eine Anwendung hierfür ist wohl eher selten zu finden; wir raten ab.

Die Wahrheitswertekomponente in einzelnen Programmzweigen.

Ein weiterer Hinweis betrifft die Verwendung der Wahrheitswertekomponente. Diese kann frei ausgelesen und beschrieben werden; dem Programmierer muss aber bewusst sein, wo diese ohne sein Zutun geändert wird.

So kann der Verwendbarkeitsgrad jederzeit beschrieben werden; es gilt jedoch die Programmzweigengewichtung als maximaler Wert. Empfohlen wird, sich auf eine etwaige explizite Initialisierung des Verwendbarkeitsgrades ganz am Anfang des Programms zu beschränken.

Weiter empfiehlt es sich, Verwendbarkeitsgrade nicht in die Bedingung eines *If-then*-Konstruktes eingehen zu lassen. Man bedenke folgenden Programabschnitt:

```
if applicability of Var then /Programmblock/ endif;
```

Ist der Verwendbarkeitsgrad der Variable **Var** anfangs 0,6, so ist er im */Programmblock/* selbst jedoch gleich $0,6^2 = 0,36$ und nicht etwa 1, wie man vielleicht meinen könnte. Es habe nun */Programmblock/* seinerseits die Form:

```
/.../
```

```
if applicability of Var then /Unterprogrammblock/ endif; /.../
```

Dann wird */Unterprogrammblock/* nicht zum relativen Gewicht 1, sondern zu 0,36 ausgeführt. Die Bedingung in *If-then*-Anweisungen sollte daher nur aus Hauptkomponente von Wahrheitswertvariablen bestimmt werden.

Zur Aggregation von Daten. Wir merken wiederholend an, dass gewisse grundsätzliche Einschränkungen für *If-then*-Programmblöcke gelten sollten, die von möglicherweise nicht scharfen Bedingungen abhängen und die zusammengeführt werden. Ausschließlich Variablen der internen Datentypen sollten neu eingeführt oder verändert werden; nur diese sind aggregierbar. In jedem dieser Blöcke sollten weiter die identischen Variablen neu eingeführt werden, d. h. sie müssen gleich bezeichnet sein, vom selben Datentyp sein und im Falle von Listen übereinstimmende Länge haben. Eine Zeichenkettenvariable muss außerdem in jedem Zweig den identischen Text beinhalten. Nur so wird anschließend die geordnete Zusammenführung der Programmzweige möglich.

Andernfalls wird eine Variable auf den Wert `null` gesetzt; denn es erscheint uns sinnvoller, festzulegen, dass ein Wert im Zweifel undefiniert ist, als dass er zufälligen oder sinnlosen Inhaltes ist.

Zum *conclude*-Befehl. Man beachte, dass, falls sich *conclude*-Anweisungen in mehreren Blöcken einer *If-then*-Bedingung befinden, es zu Parallelausführung mehrerer Instanzen des Action-Slots auch dann kommen kann, wenn diese Blöcke mit einer Anweisung zu Aggregation schließen. Variablen eines Programmzweiges, der eine *conclude*-Anweisung enthält, werden nie aggregiert, weil vorher mit dem *conclude*-Befehl entweder abgebrochen oder unmittelbar mit dem Action-Slot weitergemacht wird.

14.5 Schleifen

Die Funktionen *while* und *for* werden in FuzzyArden nicht modifiziert. Im Fall von *while* wird hierfür jeder von 1 verschiedene Wahrheitswert als 0 interpretiert; im Fall von *for* wird der Verwendbarkeitsgrad der Listeneinträge ignoriert.

15 Ergänzung: Unterschiede zu [Tif]

Die Änderungen unserer Spezifikation von FuzzyArden gegenüber [Tif] sind letztendlich gering ausgefallen.

- Statt einen neuen Datentyp *fuzzy* einzuführen, wird der bisherig *boolean* verallgemeinert. Dies entspricht wohl dem Geist von Arden, zu speichern, was auch immer übergeben wird. Dieser Vorschlag bedeutet keine Änderung in der Substanz, erleichtert allenfalls die Spezifikation.
- Die Konzepte *degree of presence* und *degree of applicability* stehen in klarerem Verhältnis. Beide Konzepte explizit parallel zu verwenden macht die Programmierung etwas unübersichtlich. Es wäre möglich und steht weiterhin zur Debatte, die *degree of presence* zu streichen und stattdessen nur Listen mit Teilzugehörigkeiten zu definieren. Derzeit ist jedoch umgekehrt vorgesehen, dass eine *degree of presence* jeder Variablen zukommt. Die *degree of applicability* ist hingegen explizit nicht definiert, vielmehr nur ein Mechanismus, durch den Programmzweiggewichtungen die *degree of presence* ändern. Dies vermeidet ein Nebeneinander zweier Begriffe, läuft aber nichtsdestoweniger in der Praxis aufs selbe hinaus, wie im ursprünglichen Konzept vorgesehen war.
- Wir lassen weiterhin zu, dass Zeichenketten und andere diskrete Werte mit einer *degree of presence* versehen werden. Jedoch wird bei der Defuzzifizierung keine scharfe Auswahl getroffen; verschiedene Zeichenketten können sinnvoll nicht automatisch defuzzifiziert werden und werden daher auf `null` zurückgesetzt.

Vielmehr muss sich der Programmierer im Fall von Texten um eine sinnvolle Kombination der Resultate selbst kümmern; das Argument ist das folgende. Es sei davon ausgegangen, dass eine Zeichenkettenvariable in einer Reihe von Fuzzy-If-then-Blöcken Warntexte enthält, die nach draußen zu geben sind. Es könnte sich nun etwa unter der einen Bedingung eines If-then-Abschnittes die Aussage „Bitte rasch das Mittel XY geben“ und unter der anderen die gegenteilige Aussage „Das Mittel XY zu geben ist nicht angezeigt“. Hiervon per höherem *degree of applicability* einfach eine Aussage auszusuchen widerspräche dem Grundgedanken von FuzzyArden und der Fuzzylogik überhaupt. Stattdessen werden entweder beide Nachrichten zusammen mit ihrer Gewichtung an den Host gegeben. Oder aber es wird im MLM aus dem Paar einer Botschaft „Gabe des Mittels XY“ und eines Wahrheitswertes die eigentliche Nachricht erst im Action-Slot generiert. So können auch im Fall, dass es sich um eine Zweieralternative handelt, mehr als zwei Möglichkeiten für den Nachrichtentext zum Zuge kommen.

- while-Schleifen werden nicht fuzzifiziert. Eine Warnung vor der Ver-

wendung der while-Schleife ist ja bereits in [Tif] enthalten.

Ebenso scheint es problematisch, ein sinnvolles Konzept für for-Schleifen zu definieren, wenn die Zugehörigkeitsgrade der Listenelemente mit einbezogen werden sollen.

- Die Möglichkeit, dass einer Wahrheitswertvariable der Wert `null` zugewiesen wird, wird generell zugelassen; `null` wird nicht mit dem Wahrheitswert 0 identifiziert. Einen beliebigen Wahrheitswert anzugeben ist etwas anderes als zu sagen, man wisse nichts.
- Den Umgang mit vordefinierten Fuzzymengen, wie sie bei Verwendung linguistischer Variablen benötigt wird, ist stark vereinfacht. Das Prinzip, nicht mit Fuzzymengen als solchen Berechnungen durchzuführen (etwa mit den Fuzzymengen über dem Wertebereich von „Körpertemperatur“), sondern nur mit den Wahrheitswerten, die sprachlichen Attributen zugeordnet sind (etwa „Es liegt kein“, „leichtes“, „hohes Fieber vor“), ist umgesetzt.

Weiter ist die Auslagerung von Fuzzymengendefinitionen in andere MLMs klarerweise wünschenswert. Hierfür jedoch einen eigenen Typ MLM zu kreieren halten wir für zu weitgehend; es sollten die vorhandenen Möglichkeiten zum Einsatz kommen.

Literatur

- [AdKo] K.-P. Adlassnig, G. Kolarz, Cadiag-2: Computer-Assisted Medical Diagnosis Using Fuzzy Subsets, in: M. M. Gupta, E. Sanchez (eds.), „Approximate Reasoning in Decision Analysis“, North-Holland Publ. Comp., Amsterdam 1982; 219 - 247.
- [AKSEG] K.-P. Adlassnig, G. Kolarz, W. Scheithauer, H. Effenberger, G. Grabner, Cadiag: Approaches to computer-assisted medical diagnosis, *Comput. Biol. Med.* **15** (1985), 315 - 335.
- [AKSG] K.-P. Adlassnig, G. Kolarz, W. Scheithauer, G. Grabner, Approach to a hospital-based application of a medical expert system, *Med. Inform.* **11** (1986), 205 - 223.
- [HL7] „Arden Syntax for Medical Logic Systems“, Version 2.5, Health Level Seven, 2005.

- [Tif] S. Tiffe, „FuzzyArden: Representation and Interpretation of Vague Medical Knowledge by Fuzzified Arden Syntax“, Dissertations-schrift, Technische Universität Wien, Wien 2003.
- [1] L. A. Zadeh, The concept of linguistic variable and its application to approximate reasoning I, II, III, *Inform. Sci.* **8** (1975), 199 - 249; **8** (1975), 310 - 357; **9** (1975), 43 - 80.