Wissensrepräsentation
Vorlesung

Sommersemester 2008

4. Sitzung

Dozent

Nino Simunic M.A.

Computerlinguistik, Campus DU

UNIVERSITÄT DUISBURG ESSEN

Wissensrepräsentation: Einführung

- **Wissensbasierte Systeme**
- KI-Programmiersprachen

UNIVERSITÄT DUISBURG ESSEN

Wissens**b**asierte **S**ysteme versus Konventionelle Programme

- **Konventionelle Programme**: (Domänen-)**Wissen** ist **mit der** Software, welche die **Anwendung bzw. Verarbeitung des Wissens** kontrolliert, **verdrahtet**.
- **Wissensbasierte Systeme**: **Trennung der beiden Rollen**
- Im einfachsten Fall: Zwei Module
  - Die **Wissensbasis** (kurz: WB) enthält nur das Wissen
  - Die **Inferenz-Maschine** nutzt die WB, um Wissen kontrolliert anzuwenden

UNIVERSITÄT DUISBURG ESSEN

Szenario: Web-Agent (Software) zur Analyse von Nachrichten

```
for ( int i=0; i<Politik.size(); i++ ) { ...

if (article.getTopic() instanceof Innenpolitik) {
        ...
} else if (article.getTopic() instanceof Außenpolitik)) {
        ...
} else if ... {
        ...
} ...
else { ... } ...
```

Innenpolitik extends → Außenpolitik extends → Politik

```
for ( int i=0; i<HotTopics.size(); i++ ) { ...

if (article.getTopic() instanceof Inland) {
        ...
} else if (article.getTopic() instanceof Ausland)) {
        ...
} else if ... {
        ...
} ...
else { ... }
```

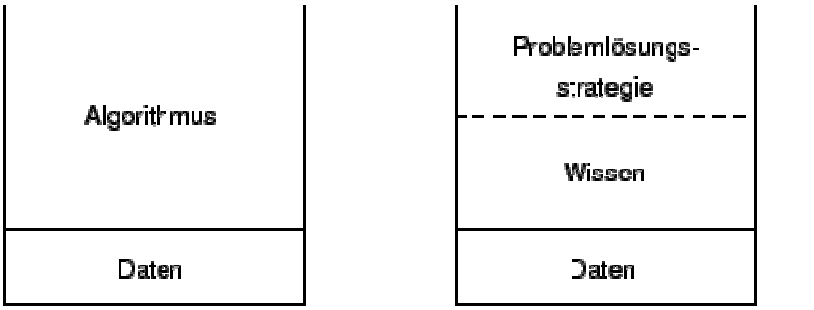
Innenpolitik extends → Inland extends

Außenpolitik extends → Ausland

Ausland → HotTopics

UNIVERSITÄT DUISBURG ESSEN

Wissensbasiertes Systeme versus Konventionelle Programme

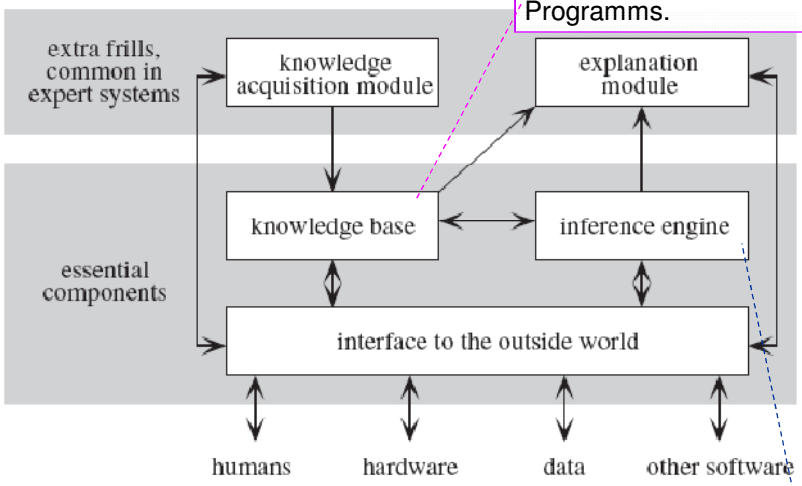


Konventionelle Programme | **Wissensbasierte Systeme**

- Konventionelle Programme (implizite Wissensspeicherung) erschweren die (nacträgliche) Änderung von Wissen. Ferner sind dazu i.d.R. (fundierte) Programmierkenntnise nötig.
- Bei wissensbasierten Systemen: Klare Schnittstelle zwischen Wissen und der allgemeinen Problemlösungsstrategie.
- Vorteil: Problemlösungsstrategie kann für mehrere Anwendungsgebiete verwenden kann. Es muss nur die entsprechende Wissensbasis ausgetauscht werden.

Allgemeiner Aufbau wissensbasierter Systeme

Wissen ist **explizit** in der Wissensbasis gespeichert – und nicht implizit in der Struktur eines Programms.



Die Inferenz-Maschine (Stück Software) verwendet die Wissensbasis: Ähnelt konventionellen Programmen, die zu verarbeitende Daten aus Dateien einlesen (ohne Wissen vorher »hart zu verdrahten«).

Wissensbasis, Fakten, Regeln: Beispiele

- Allgemein: Wissensbasis enthält **Regeln** und **Fakten**.
- Beispiel: KBS für die Gehaltsabrechnung des ACME (*American Company that Manufactures Everything*) Unternehmens.
- Ein Fakt und eine Regel der KB könnten sein:

```
/* Fact 1.1 */
Joe Bloggs works for ACME

/* Rule 1.1 */
IF ?x works for ACME THEN ?x earns a large salary
```

`x` ist eine Variable, welche mit Werten, wie bspw. Joe Bloggs oder Mary Doe, belegt werden kann.

Fakten: **Statisch**, transient, defaults

- (I.d.R.) Permanente Fakten. Z.B.:
  »ACME hat seinen Hauptsitz in Disneyland«
- Können/Sollten bei der Entwicklung in die KB geschrieben werden.
- Müssen nicht zwingend permanent sein. Unregelmäßige Änderungen führen jedoch i.d.R. zum Update der gesamten KB.

Fakten: Statisch, **transient**, **defaults**

- Nur gültig für spezifische Instanzen bzw. während eines System-Durchlaufs.
  - »Der Benutzer des Systems ist Jack«,
  - »Öldruck liegt bei 3000 Pa«, …
- ***Defaults*** *(Standard(-s))* in Abwesenheit von transienten Fakten:
  - »Der Benutzer des Systems ist GAST«,
  - »Öldruck liegt bei 0 Pa«, …

Attribute, Relationen

- Attribute sind Eigenschaften von Objekt-**Instanzen** oder Objekt-**Klassen**
  - **Klasse**: Auto / **Klasse:** Maus
  - **Instanz:** Mein Golf / **Instanz:** Jerry
- Relationen existieren zwischen Instanzen aber auch zwischen Klassen und setzen diese (semantisch) in Beziehung zueinander.
- Darstellbar als **Assoziatives** bzw. **Semantisches Netzwerk**

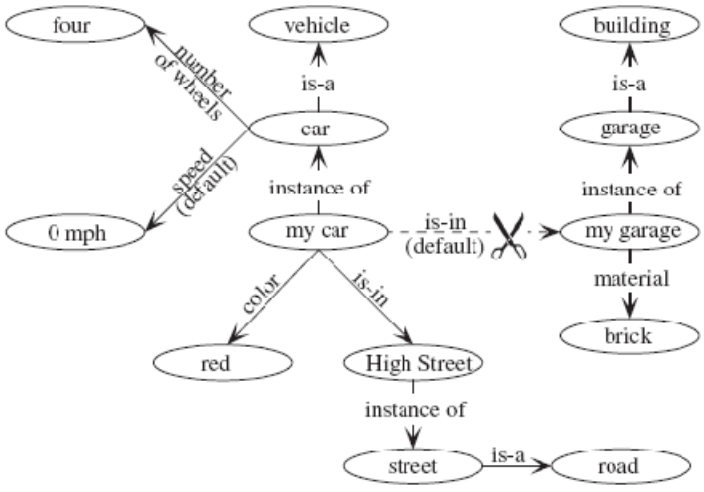
Beispiele: Statische, transiente, *default* Fakten; Attribute, Relationen

- My car is a car (**static relationship**)
- A car is a vehicle (**static relationship**)
- A car has four wheels (**static attribute**)
- A car's speed is 0mph (**default attribute**)
- My car is red (**static attribute**)
- My car is in my garage (**default relationship**)
- My garage is a garage (**static relationship**)
- A garage is a building (**static relationship**)
- My garage is made from brick (**static attribute**)
- My car is in the High Street (**transient relationship**)
- The High Street is a street (**static relationship**)
- A street is a road (**static relationship**)

-11-

UNIVERSITÄT DUISBURG ESSEN

Semantisches Netzwerk mit überschriebenem *default*



-12-

UNIVERSITÄT DUISBURG ESSEN

Gegebene Fakten (*given facts*)

- Bisherige Fakten:
  (a) Werden der KB anfänglich zur Verfügung gestellt (statische Fakten).
  (b) Werden der KB während der Laufzeit zur Verfügung gestellt.
- In beiden Fällen können die Fakten als gegebene Fakten (*engl.* given facts) beschrieben werden.

Abgeleitete Fakten (engl. *derived facts*)

- Ein oder mehrere gegebene Fakten könnten die Bedingung einer Regel erfüllen. Resultiert in der Generierung neuer Fakten, bekannt als *abgeleitete Fakten.*
- Z.B., aus der Anwendung von Rule 1.1 auf Fact 1.1

```
/* Fact 1.1 */
Joe Bloggs works for ACME

/* Rule 1.1 */
IF ?x works for ACME THEN ?x earns a large salary
```

können wir ableiten:

```
/* Fact 1.2 */
Joe Bloggs earns a large salary
```

Kettenreaktion

- Abgeleitete Fakten können weitere (Regel-)Bedingungen erfüllen, z.B.

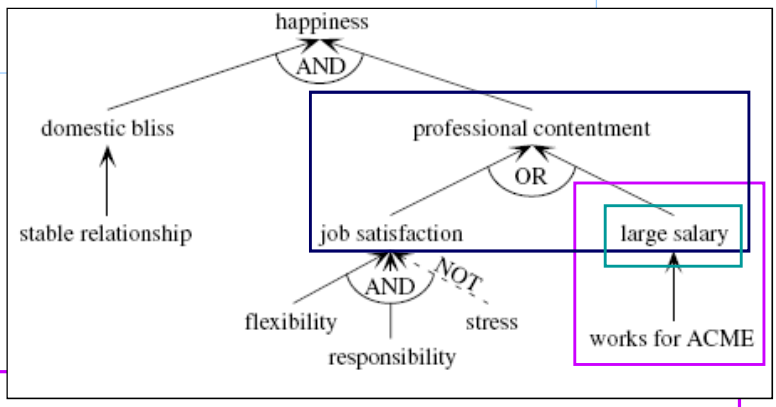
```
/* Rule 1.2 */
IF ?x earns a large salary OR ?x has job satisfaction
THEN ?x is professionally content
```

- Dies wiederum kann zur Generierung neuer abgeleiteter Fakten führen.
- Rule 1.1 and 1.2 sind voneinander abhängig / verflochten:
  - Die Schlussfolgerung der einen R. kann die Bedingung einer anderen R. erfüllen
- Diese **Interdependenzen** von Regeln lassen sich auch als Infererenz-Netzwerk darstellen.

-15-

UNIVERSITÄT DUISBURG ESSEN

Inferenz-Netzwerk




```
/* Fact 1.1 */
Joe Bloggs works for ACME
/* Rule 1.1 */
IF ?x works for ACME THEN ?x earns a large salary
```

```
/* Fact 1.2 */
Joe Bloggs earns a large salary
```

```
/* Rule 1.2 */
IF ?x earns a large salary OR ?x has job satisfaction
THEN ?x is professionally content
```

-16-

UNIVERSITÄT DUISBURG ESSEN

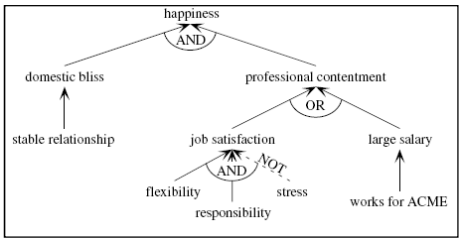
Deduktion

Die Regeln des Netzwerks, bzw. das Netzwerk per se, werden verwendet, um Ursache (*engl.* cause) und Wirkung (engl. *effect*) zu verbinden:

```
IF <cause> THEN <effect>
```

Ursache: Joe Bloggs arbeitet für ACME, und er ist in einer stabilen Beziehung.

Wirkung: → Joe Bloggs ist glücklich



Dieser Prozess wird als **Deduktion** bezeichnet.

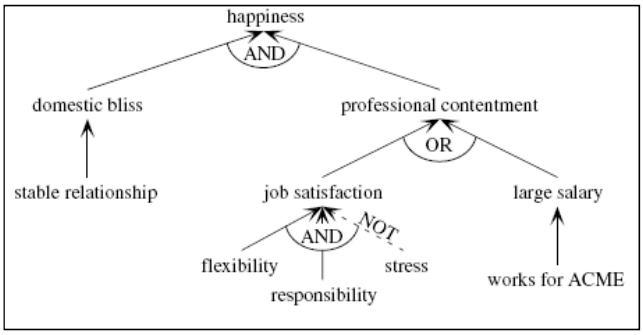
-17-

Abduktion

- Viele Probleme, z.B. Diagnostik, involvieren Schlussfolgerungen in die umgekehrte Richtung:
- **Bestimmung der Ursache bei gegebenen Fakten: Abduktion**. Z.B.

  Beobachtung: Joe Bloggs ist glücklich.

  Ableitung: Joe Bloggs genießt Hausfrieden und berufliche Zufriedenheit.



-18-

Closed-world assumption

- Die letzte Schlussfolgerung ist real nur gültig, wenn das Netzwerk *alle* Wege zeigt, die eine Person glücklich machen könnten.
- Das gezeigte Netzwerk beruht jedoch auf einer »geschlossenen« Welt (closed world), von der nichts außerhalb der Modellgrenzen bekannt sind.
  - Der Rest ist in solche spezifischen Fällen auch schlichtweg irrelevant.

Das »Frame Problem«

- Die Änderung eines Aspekts des Netzwerks kann prinzipiell eine Kettenreaktion auslösen und viele weitere Knoten im Netz beeinflussen.
- Festzustellen, was sonst noch im Welt-Modell geändert wird (weitere Knoten), wird als das *Frame Problem* bezeichnet.

Joe und das **Frame Problem**

- In Joe Bloggs Welt: Frame Problem äquivalent zur Bestimmung des Umfangs der Beziehungen zwischen Knoten.
- Bsp.: **Wenn Joe Bloggs einen neuen Job erhält**, ist die **einzige direkte Auswirkung sein Gehalt**, welches wiederum **seine berufliche Zufriedenheit und Happiness** beeinflussen.
- In komplexeren Modellen könnten viel mehr Knoten beeinfluss werden.

Induktion

- If we **have many examples of cause and effect**, we can **infer the rule** (**or inference network**) that links them.
- For instance, **if every employee of ACME** that we have met **earns a large salary**, then we might **infer Rule 1.1**:

```
/* Rule 1.1 */
IF ?x works for ACME THEN ?x earns a large salary
```

- Inferring a rule from a set of example cases of cause and effect is termed **induction**.

Zusammenfassung: Induktion, Deduktion, Abduktion

- **Deduktion***:* cause + rule → effect
- **Abduktion***:* effect + rule → cause
- **Induktion***:* cause + effect → rule

Die I**nferenz-Maschine** (*engl.* inference engine)

- Zwei wichtige Typen von Inferenz-Maschine:
  - ***Forward-chaining***
    - ***Data driven* bzw. *bottom-up***
    - ***Daten-gesteuert***
  - ***Backward-chaining***.
    - ***Goal-driven* bzw. *top-down***
    - ***Ziel-gesteuert***

Vorwärtsverkettende Inferenzmaschine

- Nimmt die verfügbaren Informationen (gegebene Fakten) und generiert so viele abgeleitete Fakten wie möglich.
- Regeln werden dabei vorwärts ausgeführt (Antezedens, dann Konsequenz).
- Der Output ist unvorhersagbar.
  - Vorteile: Neue Erkenntnisse, innovative Lösungen
  - Nachteile: Zeitverschwendung, wenn irrelevante Informationen generiert werden.
- Vorwärtsverkettung geht transitiv vor
  - Aus einem Fakt wird gegeben einer Regel und einem Schlussfolgerungsmechansismus (oft: *modus ponens*) geschlussfolgert.
  - Das abgeleitete Faktum kann wiederum, die wiederum erfüllte Bedinung einer weiteren Regel für eine weitere Schlussfolgerung verwendet werden, …usw. → Vorwärtskette

Rückwärtsverkettende Inferenzmaschine

- Regeln werden rückwärts ausgeführt (Konsequenz, Antezedenz).
- Es wird vom Ziel(-objekt) ausgegangen
  - Überprüfung nur derjenigen Regeln, die das Ziel in der Schlussfolgerung haben.
  - Ist der Wert des Objekts in der Bedingung einer Regel unbekannt, wird die Herleitung aus anderen Regeln versucht.
  - Bleibt das erfolgslos, wird der Wert (vom User) erfragt.

Deklarative vs. **prozedurale Programmierung**

- Distinkes Merkmal von WBS: Wissen und Schlussfolgerung existieren getrennt voneinander.
- Die WB enthält deklaratives Wissen (Fakten, Regeln, Beziehungen,…) ohne Details des *Wie und Wann* die Informationen verarbeitet werden.
- Vorteil: Eine Inferenz-Maschine, mehrere Wissensbasen (versch.) Wissensdomänen.

Deklarative Programmierung

- Beispiele für deklarative Programmierung:

```
/* Rule 1.3 */
IF pressure is above threshold THEN close valve
/* Fact 1.3 */
valve A is shut /* a simple fact */
/* Fact 1.4 */
valve B is connected to tank 3 /* a relation */
```

- Die Inferenz-Maschine ist i.d.R. prozedural programmiert, um die Wissensbasis zu verwerten.

Metaknowledge

- Ist Wissen über Wissen. Nützlich z.B. für Prioritäten:

```
/* Metarule 1.4 */
Examine rules about valves
before rules about pipes
```


Prozedurale Programmierung: Beispiel

```
/* A program in C to read 10 integers from a file and */
/* print them out */
#include <stdio.h>
FILE *openfile;
main()
{ int j, mynumber;
   openfile = fopen("myfile.dat", "r");
   if (openfile == NULL)
        printf("error opening file");
   else
   {
        for (j=1; j<=10; j=j+1)
        {
                fscanf(openfile,
                printf("Number %
        }
        fclose(openfile);
   }
}
```

The data file, on the other hand, contains no instructions for the computer at all, just information in the form of a set of integers.

```
43
21
523
545
332
...
```

Explicit step-by-step instructions telling the computer to perform the following actions:

(i) **open** a data file;
(ii) **print** a message if it cannot open the file, otherwise perform the remaining steps;
(iii) **set** the value of j to 1;
(iv) **read** an integer from the file and store it in the variable mynumber;
(v) **print** out the value of mynumber;
(vi) **add** 1 to j;
(vii) if j<=10 **repeat steps** (iv)–(vi), otherwise move on to step (viii);
(viii) **close** the data file.

Daten vs. Programm (am letzten Beispiel)

- **Software**: Prozedurale Anweisungen
  - Verarbeitungsanweisungen der Zahlen
- **Daten-Datei**: **Deklarative** Anweisungen

- **Daten-Datei** verhält sich **analog zu** einer trivialen **Wissensbasis**
- **Software** verhält sich **analog** zur **Inferenzmaschine**

Expertensysteme

- Typ von WBS um **Expertise bzw. Experten** in einer **spezifischen Wissensdomäne** zu verkörpern
- Beispiel-Domänen: Computer-Netzwerke, Diagnostik im Allgemeinen (Medizin, Fehleranalyse, …), …

Typisches Szenario

- Benutzer beschreibt sein Problem (bspw. die Symptome eines Fehlers im Netzwerk)

→Das Expertensystem gibt nur Vorschläge, bzw. Ratschläge zur/als Lösung des Problems aus.

- Oft: Der Dialog mit dem System wird vom System geleitet: Der Benutzer beantwortet eine Folge von Fragen oder Informationen (Formular)
- Alternativ: Das System erlaubt dem Benutzer die Übernahme der Initiative. Der Benutzer gibt in diesem Fall Informationen ein, nach denen er nicht explizit vom System gefragt wurde.

Erklärungsbedarf / -komponente

- Wünschenswert und oft benötigt: Expertensystem soll (analog zum menschl. Experten) seine Antwort begründen können.
  - **Explanation module (Erklärungskomponente/-modul)**
- Allgemeine Umsetzung: Trace (*Spur*) der verwendeten Regeln und Fakten des Systems, welche zur Lösung beitrugen.
- Äquivalent zu bspw. *stack trace* moderner Programmiersprachen wie Java (Ausführungspfad bis zum Fehler bspw.)

Expert system shell

- Ein *expert system shell (dt.* Hülle*)* (ESS) ist ein Expertensystem mit einer leeren Wissensbasis.
- Prinzipiell: Kauf eines mit Eigenentwicklung der Wissensbasis zur Erstellung eines einsatzbereiten Expertensystems für die vorliegende Wissensdomäne.
- Warum ESS? Alle Domänen sind unterschiedlich – es würd keinen Sinn machen, wenn der ESS-Entwickler auch diese implementieren würde.
- Ansprüche: Felixibilität, Fähigkeit zur generischen Nutzung von Wissensbasis/verschiedenen Domänen, sowie die Möglichkeit zur Schlussfolgerung.

Knowledge Engineering, Knowledge Acquisition (**Wissensaquisition**)

- Repräsentiertes Wissens in einer KB kann nur genutzt werden, wenn Wissen bekannt/vorhanden ist.
- Drei populäre Umsetzungstrategien zur **Aquisition von relevantem Wissen** in einer Wissensdomäne:
  - Experteninterviews (Entwickler ➔ Experte)
  - Der Entwickler der KB *ist* ein Domänenexperte
  - Das System lernt automatisch anhand von Beispielen
- **Erster Ansatz (häufig eingesetzt)**
  - Die Person, welche das Wissen aus einem Experten extrahiert und in der KB enkodiert, wird als Wissensingenieur (engl. ***knowledge engineer***) bezeichnet.
  - Der Wissensingenieur interviewt einen oder mehere Experten, die versuchen, die Expertise so zu explizieren, dass es der Knowledge Engineer versteht.
  - Problem ist insbesondere die Vermittlung der Expertise an den Ingenieur (welcher kein Experte ist). Evtl. Missverständnisse, falsche Abbildung von Wissen, Kosten (hin-her),…

- Wissensbasierte Systeme
- **KI-Programmiersprachen**

UNIVERSITÄT DUISBURG ESSEN

PROLOG

- PROLOG (PROgramming in LOGic) is a language **designed to enable programmers to build a database of facts and rules**, and then to have the system answer questions by a process of logical deduction using the facts and rules in the database.
- PROLOG provides a way for programmers to manipulate **data** in the form of rules and facts **without needing to select algorithms** or methodologies for handling those data.

UNIVERSITÄT DUISBURG ESSEN

PROLOG: **Fakten**

- **Facts** entered into a PROLOG database might look as follows:

```
tasty (cheese).
made_from (cheese, milk).
contains (milk, calcium).
```

- These facts can be expressed as the following English statements:

Cheese is tasty.

Cheese is made from milk.

Milk contains calcium.

PROLOG: **Regeln**

- We can also specify rules in a similar way, which express relationships between objects and also provide the instructions that the PROLOG theorem prover will use to answer queries.
- The following is an example of a rule in PROLOG:

```
contains (X, Y) :-
made_from (X, Z), contains (Z, Y).
```

PROLOG: **Regeln**

- The rule thus takes the form:

```
B :- A
```

  - "if A is true, then B is true," or "A implies B."
- Hence, the rule

```
contains (X, Y) :-
made_from (X, Z), contains (Z, Y).
```

  can be translated as

  "If X is made from Z and Z contains Y then X contains Y."

Abfrage von Fakten

- Having entered the three facts and one rule given above, the user might want to ask the system a question:

```
?- contains (cheese, calcium).
```

- Using a process known as **resolution** the PROLOG system is able to use the rule and the facts to determine that because cheese is made from milk, and because milk contains calcium, therefore cheese does contain calcium.
- It thus responds:

```
yes
```

Abfrage von Fakten

- It would also be possible to ask the system to name everything that contains calcium:

```
?- contains (X, calcium)
```

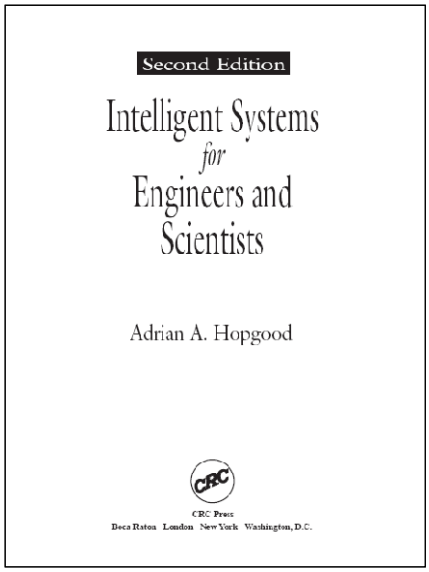
- The system will use the same rules and facts to deduce that milk and cheese both contain calcium, and so will respond:

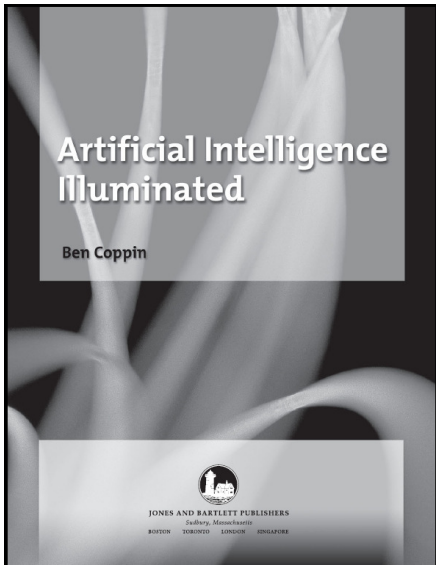
```
X = milk.
X = cheese.
```


LISP

- LISP (LISt Programming) is a l**anguage that more closely resembles the imperative programming languages such as C++** and Pascal than does PROLOG.
- As its name suggests, LISP is **based around handling of lists** of data.
  - A list in LISP is contained within brackets, such as: `[A B C]`
  - This is a list of three items. LISP uses lists to represent data, but also to represent programs. Hence, a program in LISP can be treated as data.
  - This introduces the possibility of writing **self-modifying programs** in LISP […]
- Komplexere Syntax als PROLOG
- Konventionelles (Verzweigung, Schleifen, …)
- …

Referenzen





-45-

UNIVERSITÄT DUISBURG ESSEN