



# Jess

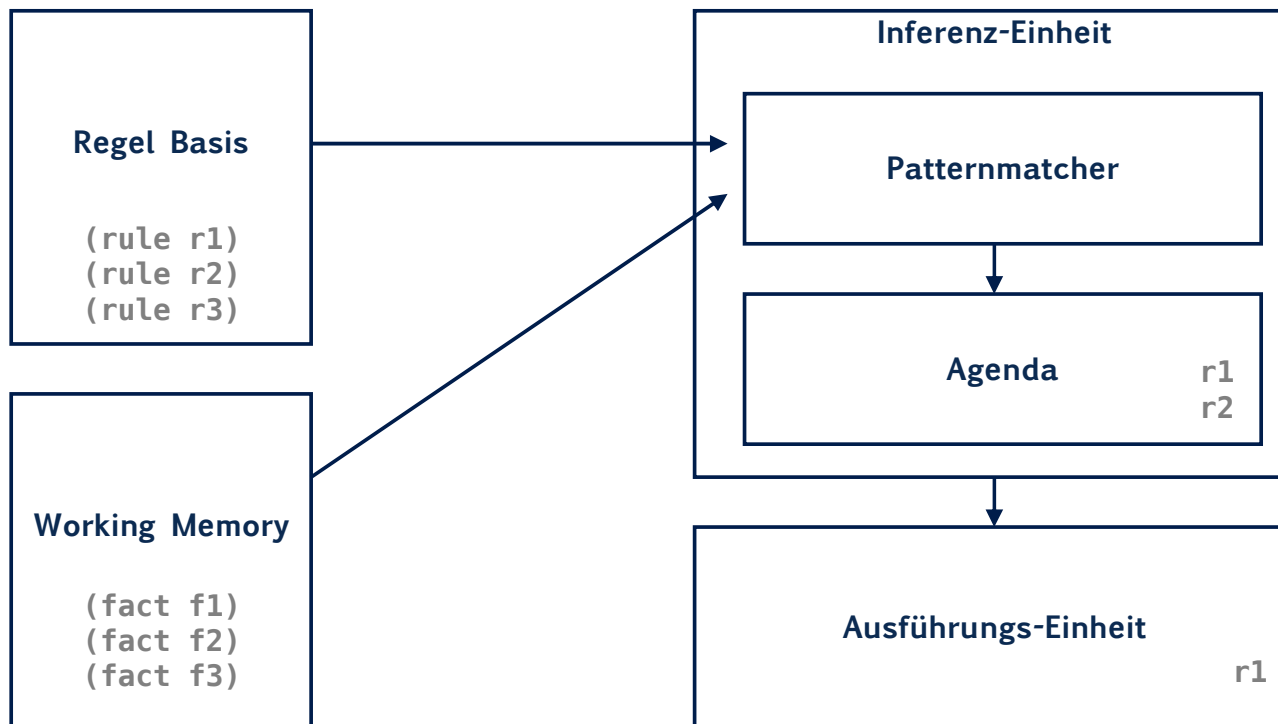
Teil 2

"Wenn die Fakten nicht zur Theorie passen, ändere die Fakten!"  
Albert Einstein

Jens Hauptert

02. November 2006

# Rückblick



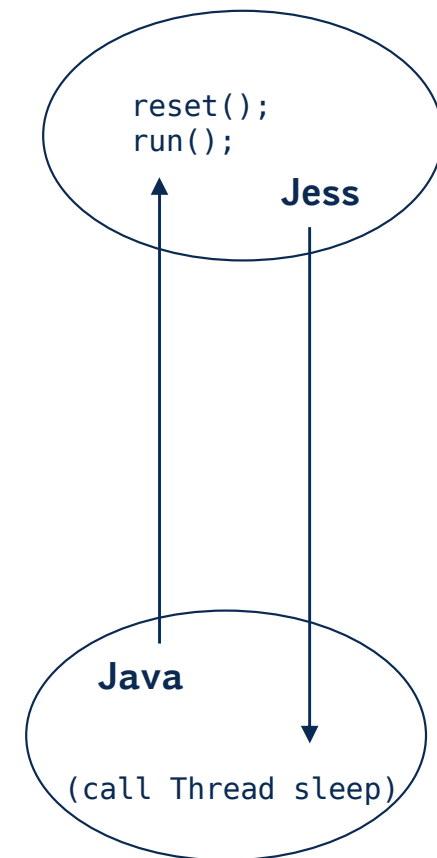
## Inhalt

- **Jess & Java**
  - Java skripten / Jess als Java-Objekt
  - Datentypenkonvertierung
- **Unter der Oberfläche / Wie Jess arbeitet**
  - Regelverarbeitung
  - RETE
- **PC-Reperatur-Assistent**
  - Vom Flussdiagramm zur Regel
  - Rückwärtsverkettung
  - Erstellung der Regeln
  - Demo: Assistent in Aktion

## Jess & Java

### Überblick

- **Von Jess aus Java-Elemente nutzen:**
  - Vollständiger Zugriff auf alle Java Funktionen und Konzepte
  - Jess bildet Skriptsprache für Java
  - Jess als Java-Interpreter nutzbar
- **Java-Anwendungen mit Jess-Funktionalität:**
  - Vollständiger Zugriff auf alle Jess Funktionen
  - Jess-Engine steht als eigene Klasse zur Verfügung
  - Einbetten von Jess in alle Arten von Java-Anwendungen



## Jess & Java

### Codebeispiele

#### Java mit Jess skripten

```
Jess> (bind ?prices (new java.util.HashMap))  
<External-Address:java.util.HashMap>  
  
Jess> (call ?prices put bread 0.99)  
Jess> (call ?prices put peas 1.49)  
Jess> (call ?prices put peas)
```

#### Jess als Java-Objekt

```
Import jess.*;  
...  
Rete engine = new Rete();  
engine.executeCommand("(assert fact (a 10) (b 20))");  
engine.run();  
Value val = engine.fetch("fact");
```

## Jess & Java

### Datentypenkonvertierung

Jess type	Possible Java types
<code>RU.EXTERNAL_ADDRESS</code>	The wrapped object
<code>nil</code>	<code>null</code>
<code>TRUE, FALSE</code>	<code>String, boolean</code>
<code>RU.INTEGER</code>	<code>long, short, byte, ...</code>
<code>RU.LIST</code>	Java array



## Inhalt

- **Jess & Java**
  - Java skripten / Jess als Java-Objekt
  - Datentypenkonvertierung
- **Unter der Oberfläche / Wie Jess arbeitet**
  - Regelverarbeitung
  - RETE
- **PC-Reperatur-Assistent**
  - Vom Flussdiagramm zur Regel
  - Rückwärtsverkettung
  - Erstellung der Regeln
  - Demo: Assistent in Aktion

## Unter der Oberfläche

- **Jess Vorgehensweise nach Aufruf von (run):**
  - 1: Finde alle Regeln, die zu den Fakten des Working Memory (WM) passen
  - 2: Falls keine passende Regel existiert => Abbruch
  - 3: Erstelle Agenda aus allen passenden Kombinationen
  - 4: Wähle ein Element aus der Agenda und führe es aus
  - 5: GOTO 1

## Unter der Oberfläche

*"4: Wähle ein Element aus der Agenda und führe es aus"*

### **Konfliktauflösung**

- "depth"-Strategie:
  - Jede Regel hat einen "salience"-Wert
  - Je später die Regel aktiviert wurde desto höher der Wert
  - Die Regel mit dem höchsten "salience"-Wert feuert
  - (`declare (salience +/- <Wert>)`) verändert Wert manuell
- "breadth"-Strategie:
  - Regeln feuern in der Reihenfolge ihrer Aktivierung
- Eigene Strategie festlegen:
  - Implementierung von "jess.Strategy" erstellen
  - Via (`set-strategy <classname>`) in Jess laden

## Unter der Oberfläche

- **Primitiver Algorithmus:**
  - Liste aller Regeln vorhalten
  - Prämissen aller Regeln mit WM vergleichen
- **Worst-Case-Laufzeit:**
  - $P$  = Anzahl der Vergleiche ("if's")
  - $F$  = Anzahl der Fakten im WM

$$O(F^P)$$

## Unter der Oberfläche

- **Nachteile:**
  - Mehrzahl der Tests haben in jeder Iteration gleiches Ergebnis
  - Viele Tests werden mehrfach pro Zyklus durchgeführt  
→ sehr ineffizient
- **Informationen über regelbasierte Systeme:**
  - Anzahl der Regeln bleibt üblicherweise konstant
  - Working Memory ändert sich pro Zyklus
    - Allerdings: Nur wenige Elemente werden geändert!

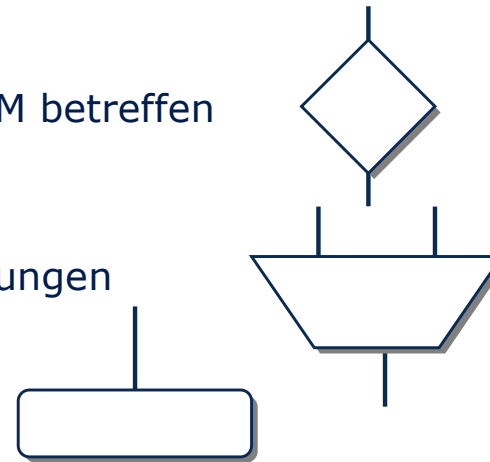
## Unter der Oberfläche

- **Idee:**
  - Bereits bekannte Testergebnisse wieder verwenden
  - Erneuter Test nur bei sich tatsächlich geänderten Fakten
- **Der RETE-Algorithmus:**
  - Vom lateinischen "rete" für "Netz"
  - Entwickelt von Charles Forgy (Carnegie Mellon University, 1979)
  - Sehr effiziente Regelverarbeitung
  - Weit verbreitet, verwendet in
    - OPS5 und CLIPS
    - ART bzw. Jess

## Unter der Oberfläche

RETE-Algorithmus » Funktionsweise

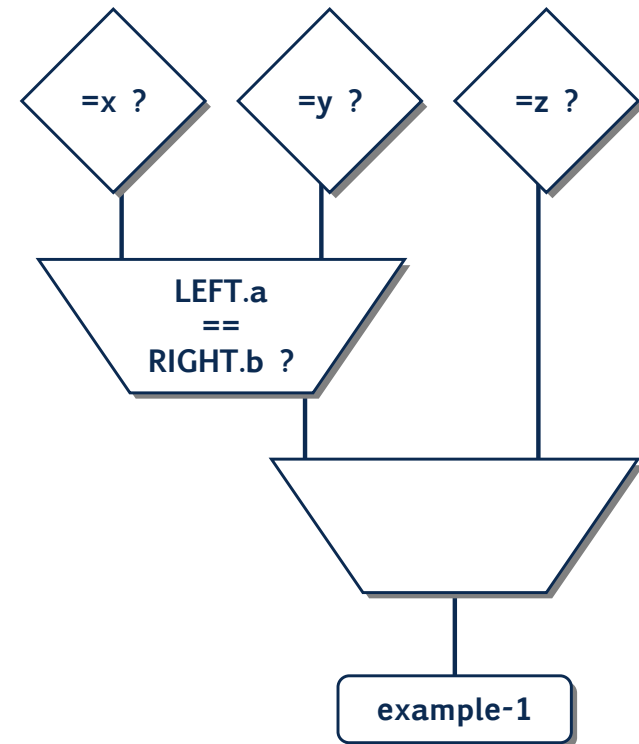
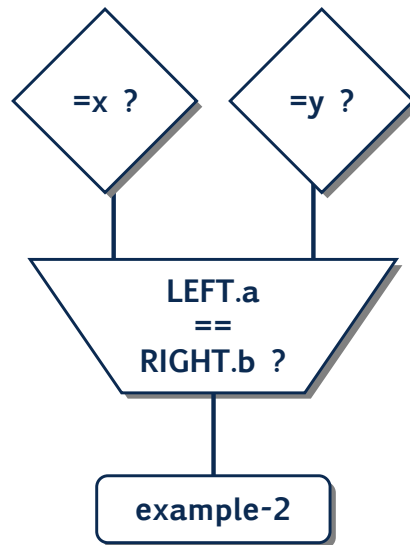
- **Netz von untereinander verbundene Knoten**
- **Knoten:**
  - Repräsentieren einen oder mehrere Tests einer Regel
  - Haben einen oder zwei Eingänge
  - Besitzen einen Ausgang, der beliebig verzweigt werden kann
  - **Einerknoten:**
    - Bedingungen die nur ein Element des WM betreffen
  - **Zweierknoten:**
    - Beziehungen zwischen den Einzelbedingungen
  - **Terminalknoten:**
    - Schließt eine Regel ab



# Unter der Oberfläche

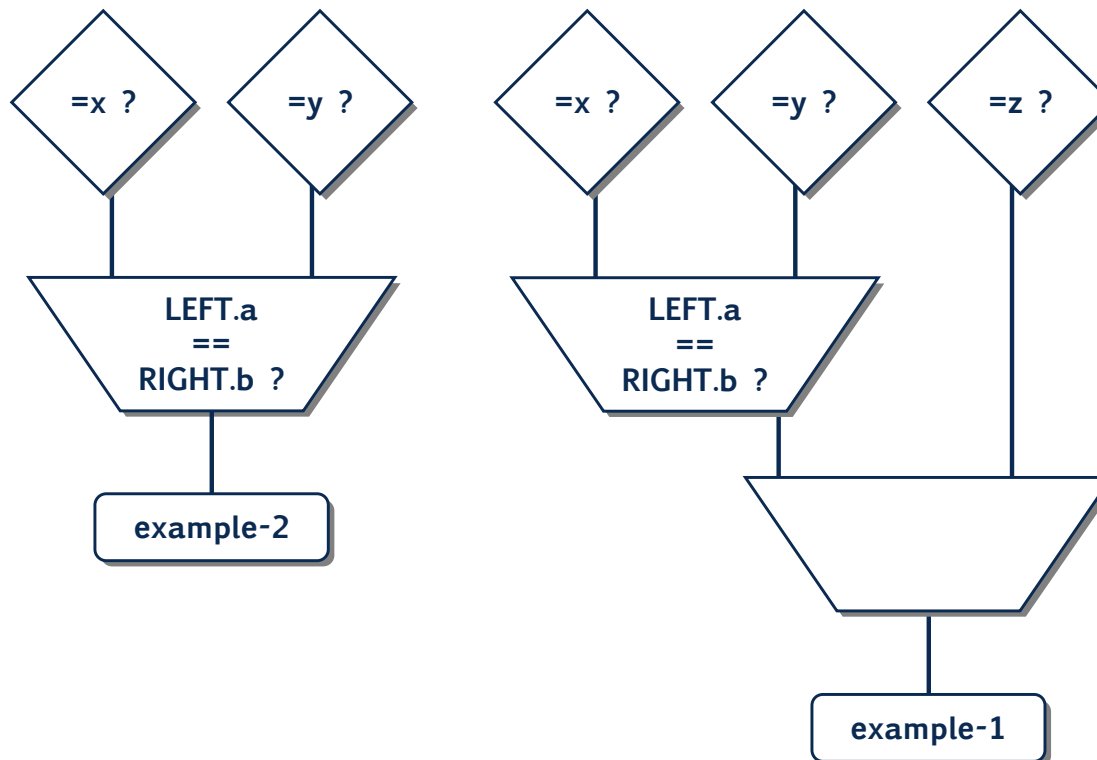
RETE-Algorithmus » Funktionsweise

```
Fact: (x (slot a))  
Fact: (y (slot b))  
Fact: (z (slot c))  
Regel: ((x (a ?v1))  
        (y (b ?v1))  
        =>  
        )  
Regel: ((x (a ?v2))  
        (y (b ?v2))  
        (z)  
        =>  
        )
```



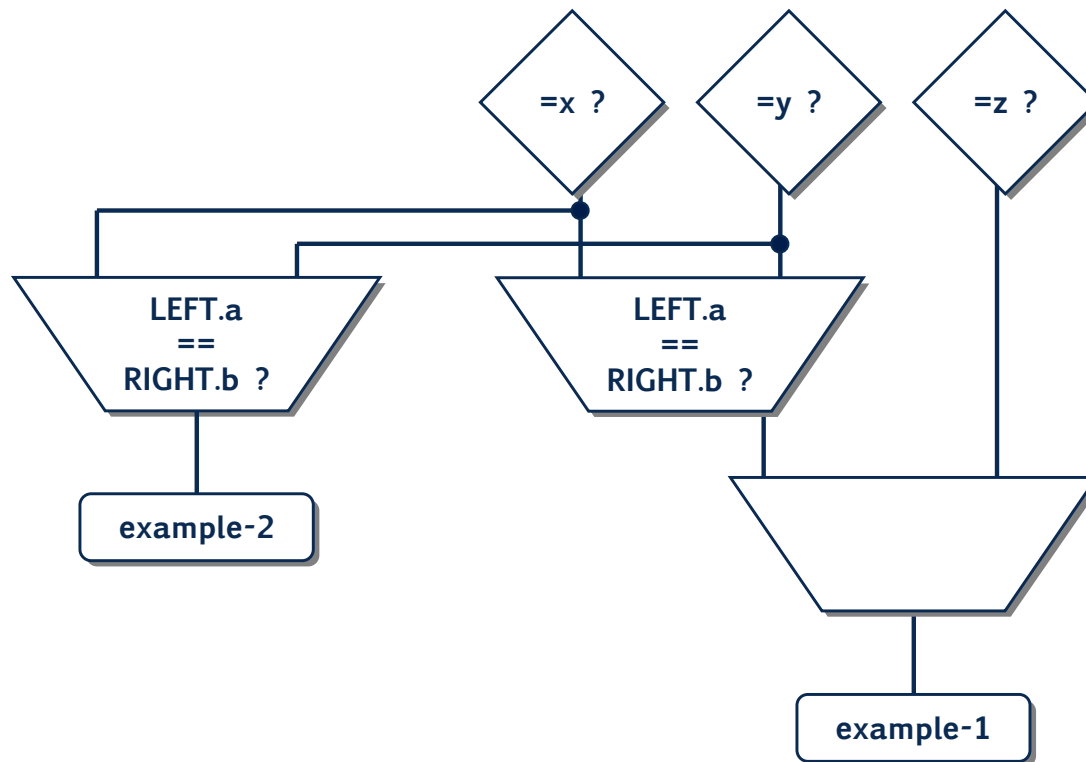
# Unter der Oberfläche

RETE-Algorithmus » Optimierungen



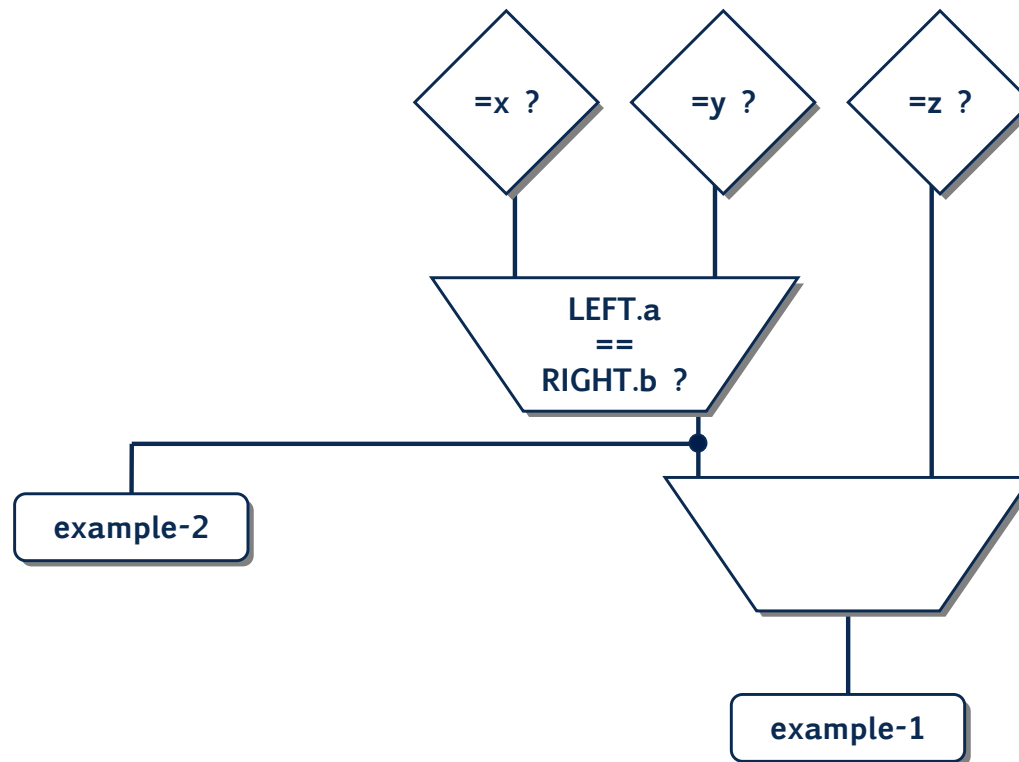
# Unter der Oberfläche

RETE-Algorithmus » Optimierungen



# Unter der Oberfläche

RETE-Algorithmus » Optimierungen



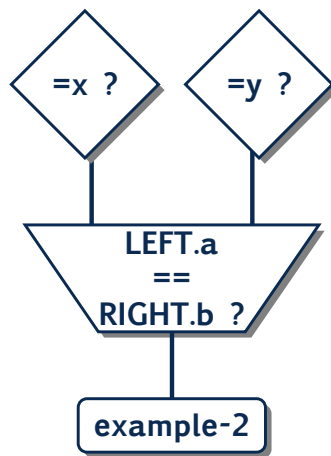
## Unter der Oberfläche

RETE-Algorithmus » Rückmeldung

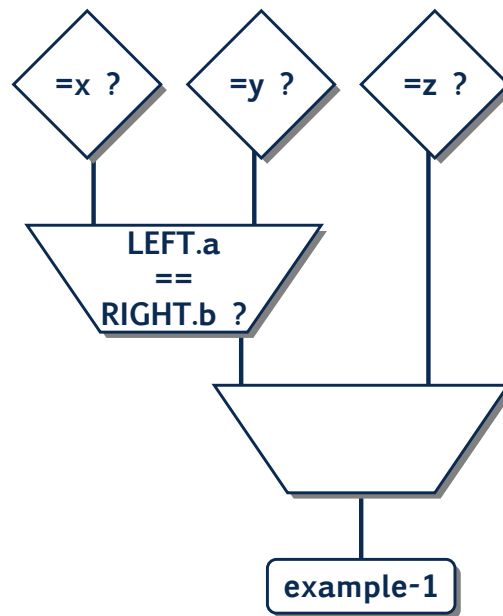
- **Jess Ausgabe bei Regeleingabe:**
  - Beispiel: "+1+1+2+t"
  - "+1" neuer Einerknoten
  - "+2" neuer Zweierknoten
  - "+t" neuer Terminalknoten
  - "=1" Einerknoten konnte übernommen werden
  - "=2" Zweierknoten konnte übernommen werden

## Unter der Oberfläche

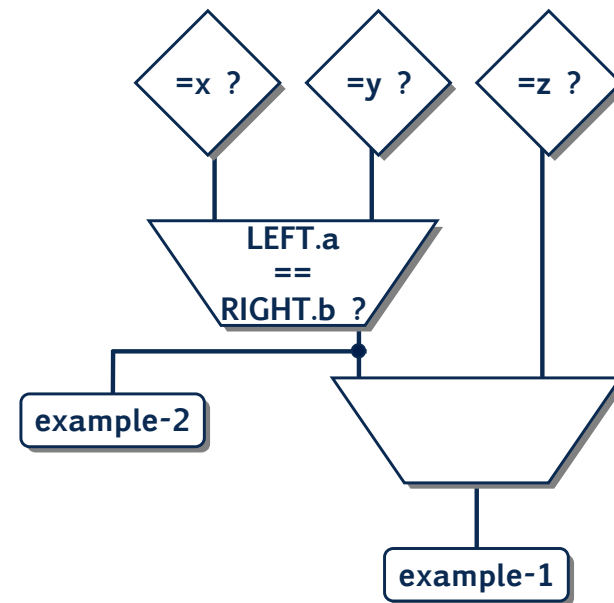
RETE-Algorithmus » Rückmeldung



example-2: +1+1+1+2+t



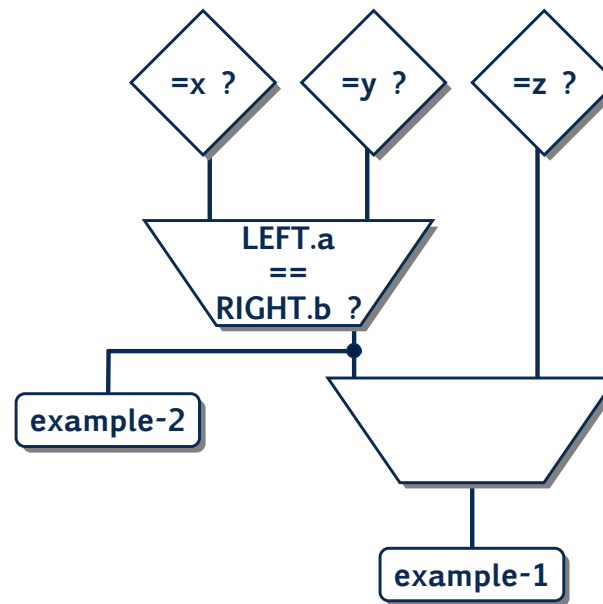
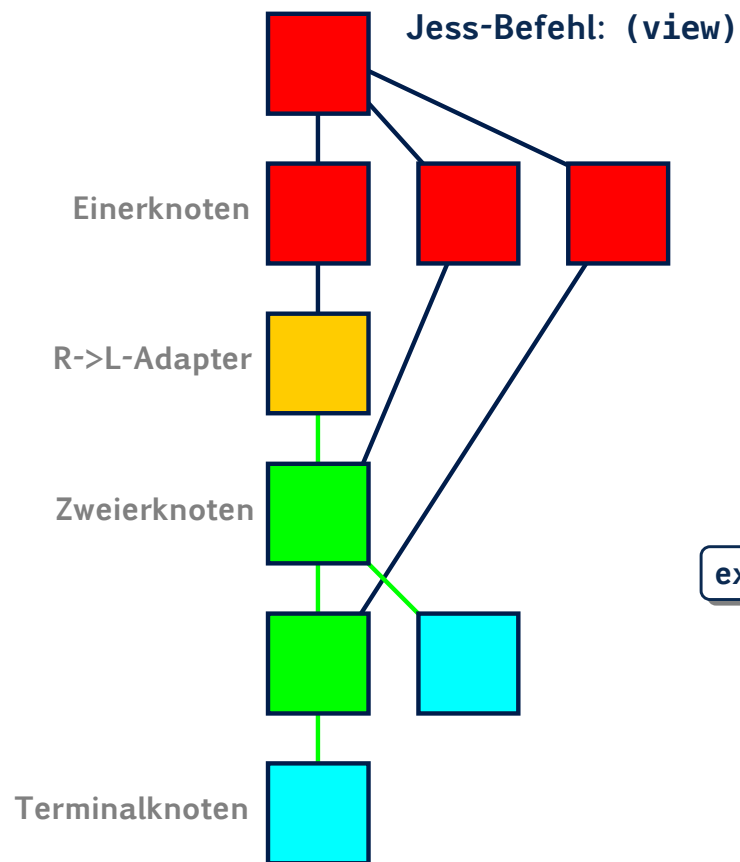
example-1: +1+1+1+2+1+2+t



example-2: =1=1=1=2+t

## Unter der Oberfläche

RETE-Algorithmus » Implementierung



example-1: +1+1+1+2+1+2+t

example-2: =1=1=1=2+t

## Unter der Oberfläche

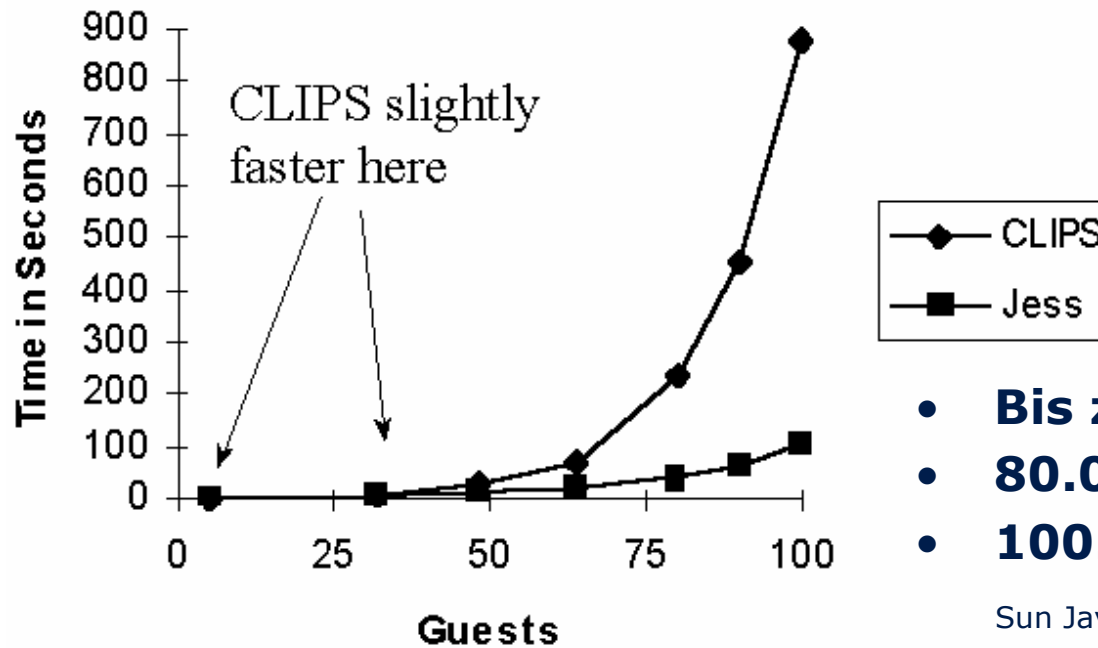
Geschwindigkeitsvergleich

	<b>Jess</b>	<b>CLIPS</b>
<b>Sprache</b>	<b>Java</b>	<b>C</b>
<b>Erscheinungsdatum</b>	<b>1995</b>	<b>1985</b>
<b>Autor</b>	<b>E. Friedmann-Hill Sandia National Labs</b>	<b>NASA-Team Johnson Space Center</b>
<b>Zweck</b>	<b>Verknüpfung von regelbasierten Systemen mit Java</b>	<b>Ersatz für kommerzielle Werkzeuge</b>

# Unter der Oberfläche

Geschwindigkeitsvergleich

## Manners Benchmark



- **Bis zu 20 mal schneller als CLIPS**
- **80.000 Regel/Sekunde**
- **100.000 Fakten/Sekunde**

Sun Java VM auf Pentium III 800 MHz



## Inhalt

- **Jess & Java**
  - Java skripten / Jess als Java-Objekt
  - Datentypenkonvertierung
- **Unter der Oberfläche / Wie Jess arbeitet**
  - Regelverarbeitung
  - RETE
- **PC-Reperatur-Assistent**
  - Vom Flussdiagramm zur Regel
  - Rückwärtsverkettung
  - Erstellung der Regeln
  - Demo: Assistent in Aktion

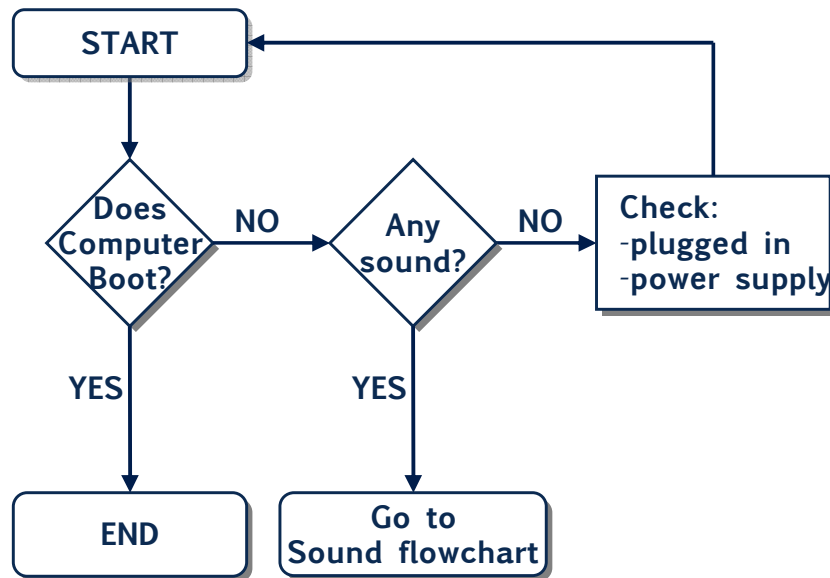
## PC-Reperatur-Assistent

### Übersicht

- **Idee:**
  - Regelbasiertes Expertensystem zur Fehlerdiagnose
- **Umsetzung:**
  - Erstellung von Flussdiagrammen zur Modellierung des Problems
  - Übersetzung von Flussdiagrammen in Jess-Regeln
  - Interaktive Benutzerführung
  - Automatische Problemeingrenzung

# PC-Reperatur-Assistent

Eingrenzung des Problembereichs



```
(defrule not-plugged-in
  (answer (ident sound) (text no))
  (answer (ident plugged-in) (text no))
  =>
  (recommend-action "plug in the
                    computer")
  (halt)
)

(defrule power-supply-broken
  (answer (ident sound) (text no))
  (answer (ident plugged-in) (text yes))
  =>
  (recommend-action "repair or replace
                    power supply")
  (halt)
)
```

## PC-Reperatur-Assistent

Problemlösung durch Rückwärtsverkettung

- **Vorwärtsverkettung:**
  - Bisher verwendet
  - Algorithmus:
    - Suche eine Regel deren Prämisse zum WM passt
- **Rückwärtsverkettung:**
  - Bestimmte Fakten werden als Ziel ("goal") definiert
  - Algorithmus:
    - Suche eine Regel deren Konklusion das Ziel generiert

## PC-Reperatur-Assistent

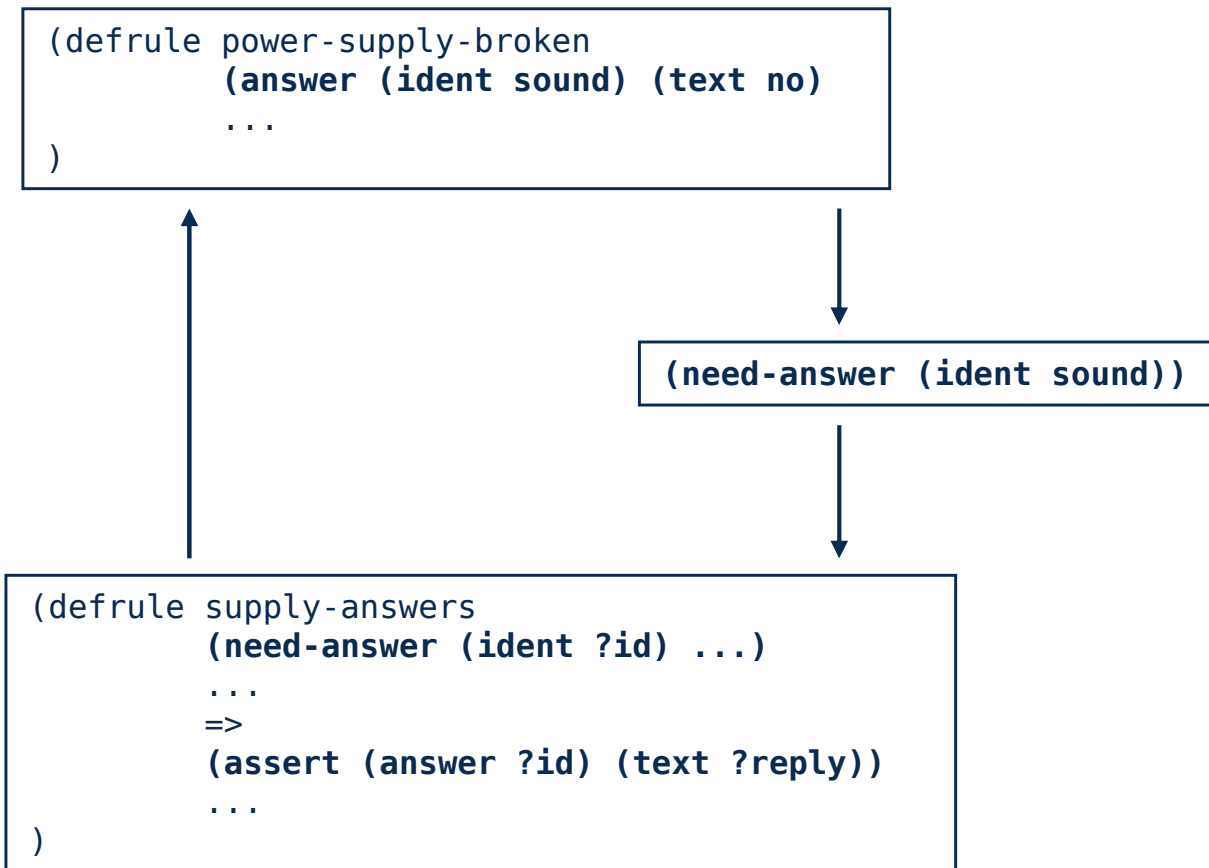
Problemlösung durch Rückwärtsverkettung

- **Rückwärtsverkettung in Jess:**
  - Keine echte Rückwärtsverkettung
  - Interne Simulation über Vorwärtsverkettung
  - (do-backward-chaining <fact>)
  - Automatisches Einfügen von (need-<fact>) ins WM
  - Voraussetzung ist eine Regel der Form:

```
(defrule trigger123
  (need-<fact>)
  =>
  (assert <fact>)
)
```

# PC-Reperatur-Assistent

Problemlösung durch Rückwärtsverkettung



## PC-Reperatur-Assistent

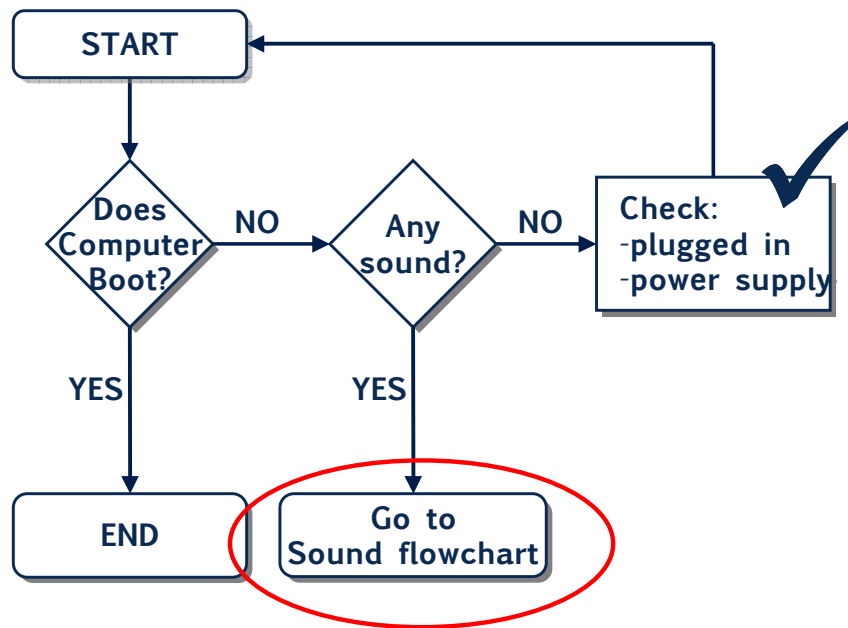
Problemlösung durch Rückwärtsverkettung

```
(do-backward-chaining answer)

(defrule supply-answers
  (need-answer (ident ?id))
  (not answer (ident ?id))
  =>
  (bind ?answer (ask-user ?id))
  (assert (answer (ident ?id) (text ?answer)))
  (return)
)
```

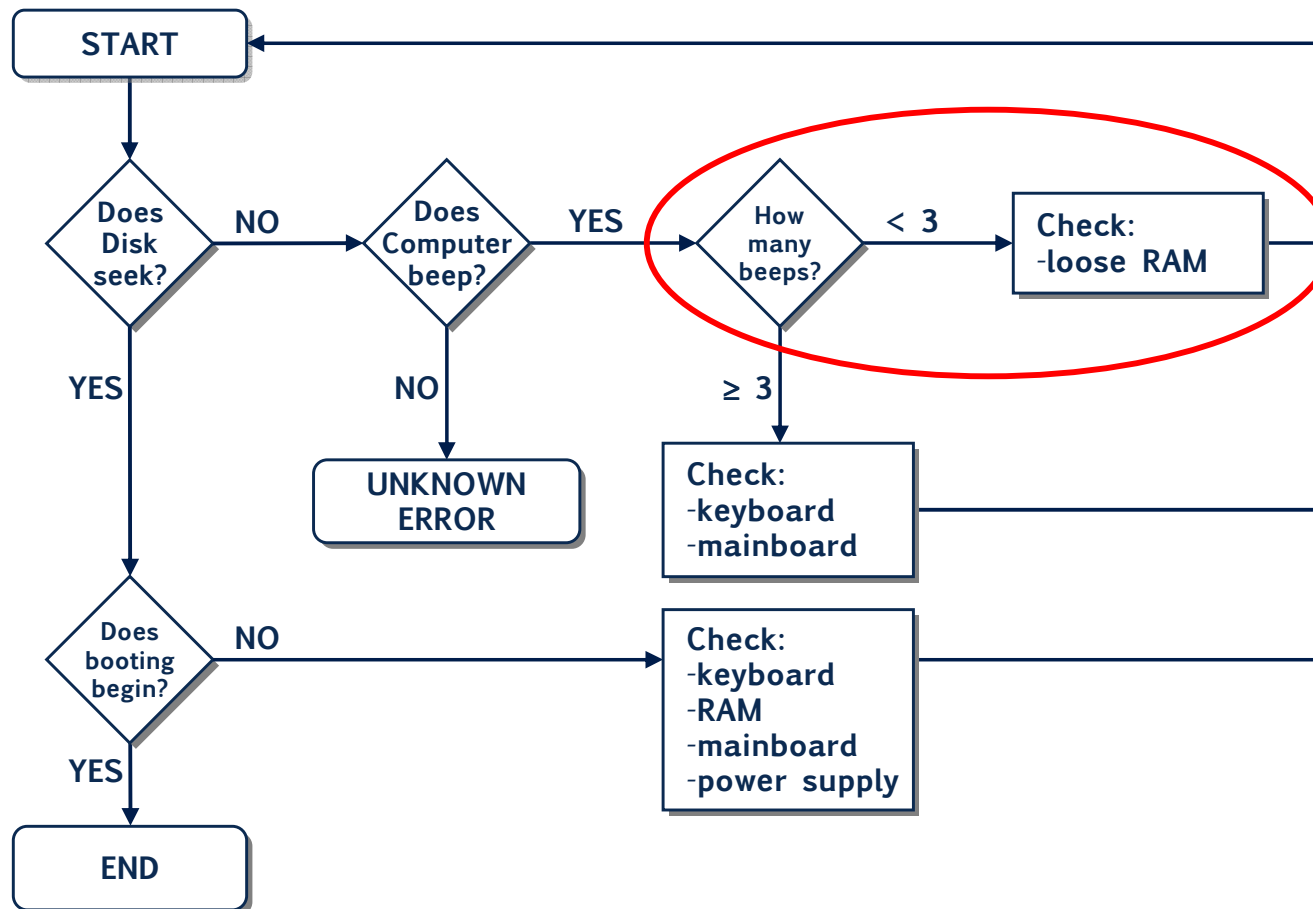
# PC-Reperatur-Assistent

Restliche Regeln » bisheriger Stand



# PC-Reperatur-Assistent

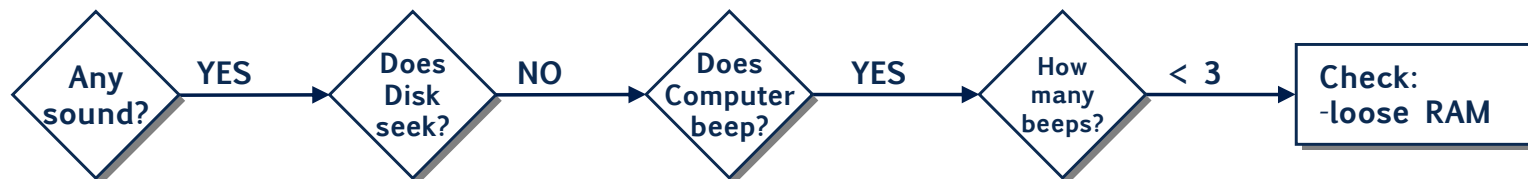
Restliche Regeln » "Sound flowchart"



## PC-Reperatur-Assistent

### Restliche Regeln

```
(defrule check-ram
  (answer (ident sound) (text yes))
  (answer (ident seek) (text no))
  (answer (ident does-beep) (text yes))
  (answer (ident how-many-beeps) (text ?t))
  (test (< (integer ?t) 3))
  =>
  (recommend-action "check for loose RAM")
)
```



## PC-Reperatur-Assistent

Fragen definieren

```
(deffacts question-data
  (question (ident hardware)
    (type multi) (valid x86 Mac other)
    (text "What kind of hardware is it?"))
  (question
    (ident sound) (type multi) (valid yes no)
    (text "Does the computer make any sound?"))
  (question
    (ident plugged-in) (type multi) (valid yes no)
    (text "Is the computer plugged in?"))
  ...
  (question
    (ident how-many-beeps) (type number) (valid)
    (text "How many times does it beep?"))
)
```



## PC-Reperatur-Assistent

# DEMO

## Zusammenfassung

- **Verknüpfung**
  - Fließender Übergang von Jess zu Java erlaubt:
    - Jess-Systeme mit Java-Objekten (z.B. GUI)
    - Java-Anwendungen mit Jess-Produktionensystem
- **Implementierung**
  - Einfacher Prämissenvergleich bietet nur unzureichende Performance
  - RETE-Algorithmus erlaubt eine sehr effiziente Regelverarbeitung
- **Anwendung**
  - Rückwärtsverkettung ermöglicht die automatisierte Ausführung von Aktionen bei Bedarf, z.B. in Form eines Interview
  - Konzentration auf die notwendigen Regeln
  - Bereits durch wenige Regeln lassen sich einfache Expertensysteme erstellen

## Fazit

“Best of Both Worlds”

- **Regelbasierte Systeme:**
  - Aus Flussdiagrammen lassen sich leicht Regeln ableiten
  - Kontrollfluss muss nicht vom Programmierer festgelegt werden
  - Als Expertensystem weit verbreitet
- **Jess:**
  - Gesamte Java-Welt in regelbasierten Systemen nutzbar und umgekehrt
  - Sehr effizient durch optimierten RETE-Algorithmus
  - Synthese von prozeduralem und deklarativem Programmieren
  - Keine objektorientierten Konzepte



# FRAGEN?



## Literaturangaben

- **„Jess in Action“, Ernest Friedman-Hill, MANNING**
- **„Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem“, Charles L. Forgy (1982)**
- **JESS Homepage: <http://herzberg.ca.sandia.gov>**