Der RETE-Algorithmus

Der *Rete-Algorithmus* wurde 1979 von Charles Forgy an der Carnegie Mellon University im Rahmen der Entwicklung von OPS5 (Expertensystem-Shell) entwickelt.

> Forgy, Charles, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, Artificial Intelligence, 19, pp. 17-37, 1982.

☞ Der Rete-Algorithmus bildet die Basis für viele neue Entwicklungen im Bereich von Expertensystemen (Jess, Drools, ILog JRules, IBM CommonRules)

Ziel des Rete-Algorithmus:

- Für eine Menge von Mustern und sich eine ändernde Objektmenge auf effiziente Weise die Muster zu bestimmen, die durch die aktuelle Objektmenge erfüllt werden.

In der Anwendung für Regelsysteme sind die Muster die Produktionsregeln.

Der Rete-Algorithmus erreicht die Effizienzsteigerung durch:

- ☞ Einschränkung der zu überprüfenden Bedingungen
  Regelprämissen enthalten teilweise identische Teile. Die mit diesen Teilen verbundenen Überprüfungen werden nur einmal durchgeführt.
- ☞ Einschränkung der zu überprüfenden Datenbasiseinträge
  Im nächsten Zyklus werden nur die Änderungen an der Faktenbasis daraufhin überprüft, ob sich eine Änderung an der Konfliktmenge ergibt.

Rete-Netzwerk

- “rete” heißt im Lateinischen Netz.
- Der Rete-Algorithmus erzeugt aus einer Menge von Regelprämissen ein Entscheidungsnetzwerk in Form eines Datenflussgraphen.
- Der Datenflussgraph besteht im wesentlichen aus zwei Typen von Knoten, die die eigentlichen Operationen repräsentieren.

Bezeichnungen im Umfeld von Rete:

- Die Regelprämissen werden auch als *LHS (left hand side)* bezeichnet.
- Die Konklusionen bzw. Aktionen einer Regel heißen *RHS (right hand side)*.
- Die Faktenbasis bezeichnet man als *working memory*.

Typen von Knoten im Rete-Netzwerk

α-Knoten: repräsentieren Selektionsbedingungen, die sich auf einzelne Objekte (Elemente, Tupel) des working memory beziehen.

β-Knoten: repräsentieren Bedingungen, die über Junktoren (und Variablenbindungen) miteinander verknüpft sind.

Beispiel 4.7.
Wenn X Elternteil von Y ist und X weiblich ist
Dann ist X Mutter von Y.

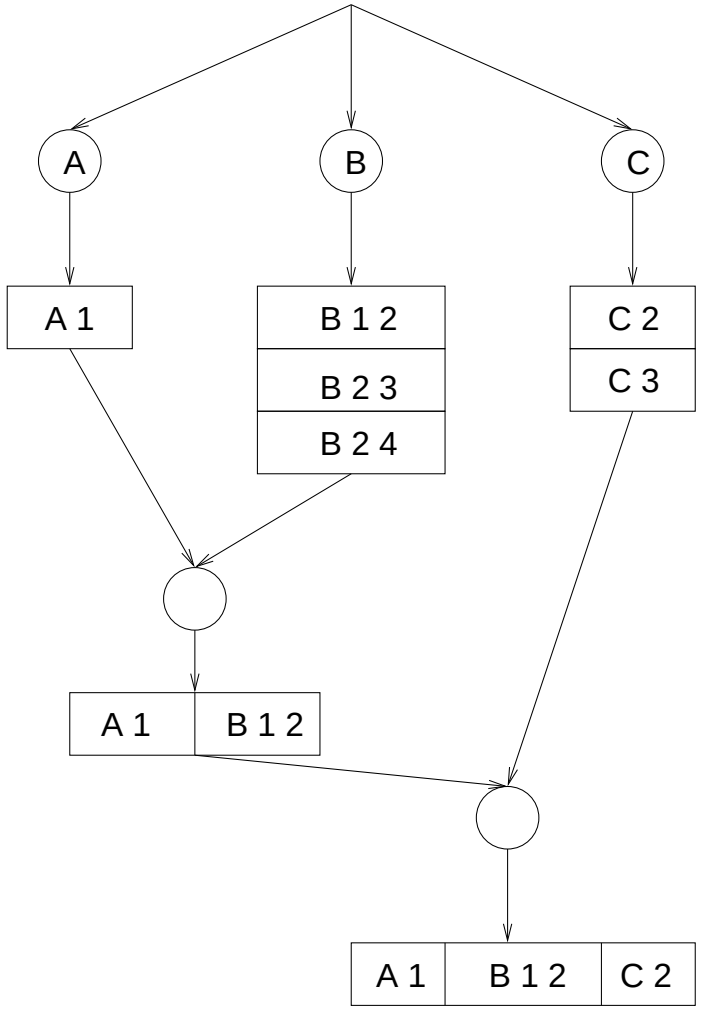
α-Knoten: X ist weiblich
α-Knoten: X ist Elternteil von Y
β-Knoten: Das “und” in Verbindung mit der gebundenen Variablen X

Beispiel 4.8. Wir betrachten die Regel:

$$A(X) \wedge B(X,Y) \wedge C(Y) \rightarrow \mathrm{Aktion}$$

α-Knoten: drei Stück für die Einzelbedingungen $A(X), B(X,Y), C(Y)$
β-Knoten: zwei Stück für die Verbindung von $A(X)$ mit $B(X,Y)$ und $B(X,Y)$ mit $C(Y)$.

Working Memory:
(A 1)
(B 1 2)
(B 2 3)
(B 2 4)
(C 2)
(C 3)
(D 2)
(D 4)



Selektion
α −Knoten

Verbund
β −Knoten

β −Knoten
Verbund

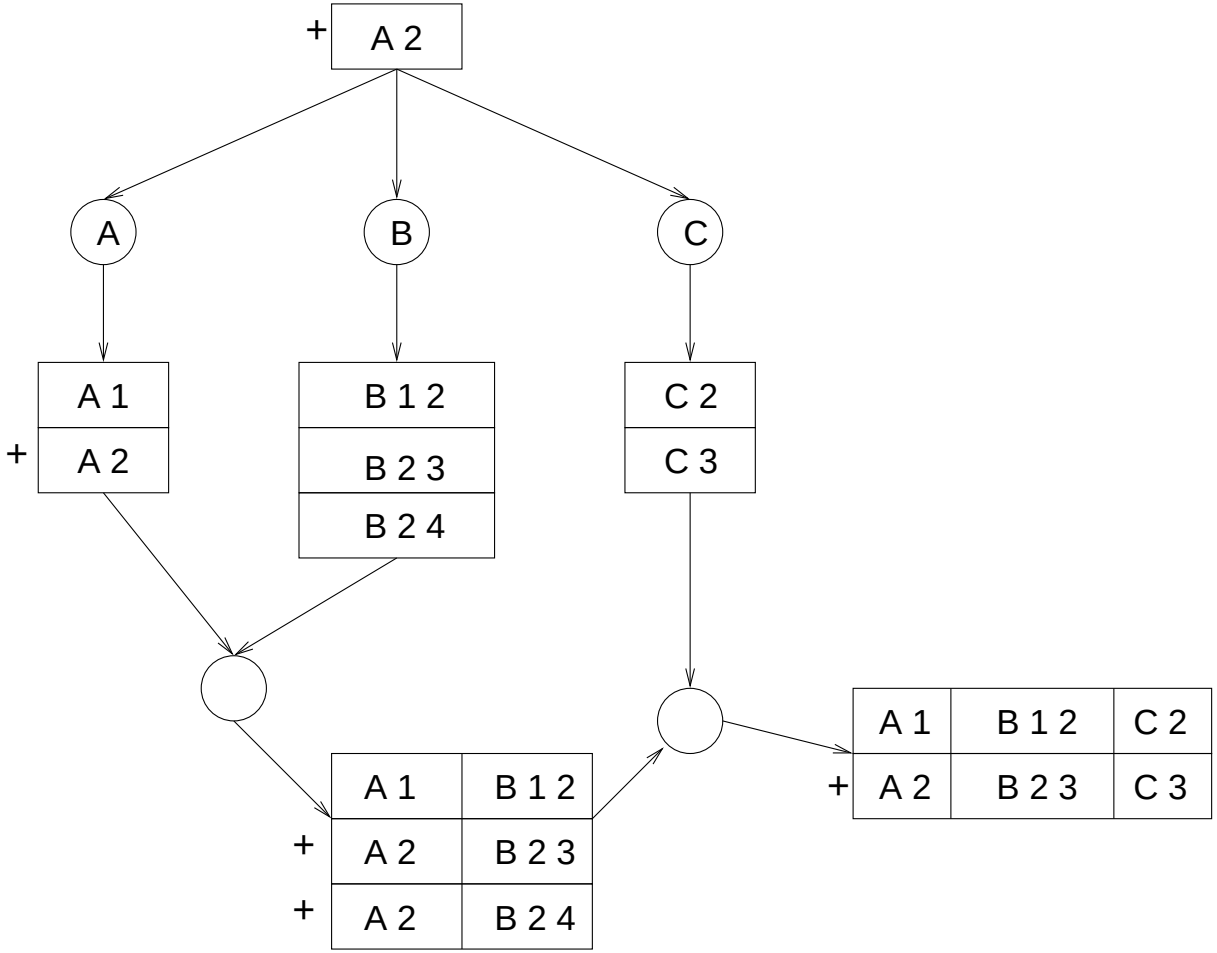
Bemerkungen:

- Zu den einzelnen Knoten werden die Ergebnisse abgespeichert, d.h.:
  - zu einem α-Knoten die Objekte/Tupel, die die Selektionsbedingung erfüllen und
  - zu einem β-Knoten die Tupelkombinationen, die die Verbundbedingung erfüllen.
- Wenn die beiden “Eingänge” eines β-Knotens nicht über eine gemeinsame Variablen verbunden sind, liefert der Knoten das Kreuzprodukt der Eingangstupelmengen als Ergebnis.
- Das Ergebnis des letzten Knotens einer Regel stellt die Menge der Regelinstanzen dar, die diese Regel erfüllen.

Inkrementelle Änderungen in einem Rete-Netzwerk

- Änderungen am Working Memory werden durch das Netzwerk propagiert.
- Wird in das Working Memory ein Tupel eingefügt, wandert dieses als Plus-Tupel (+) durch das Rete-Netz, wobei die Knotenspeicher aktualisiert werden.
- Analoges gilt für Tupel, die aus dem Working Memory gelöscht werden (–).
- Änderungen an der Ergebnismenge des letzen Knotens stellen Änderungen an der Konfliktmenge dar.

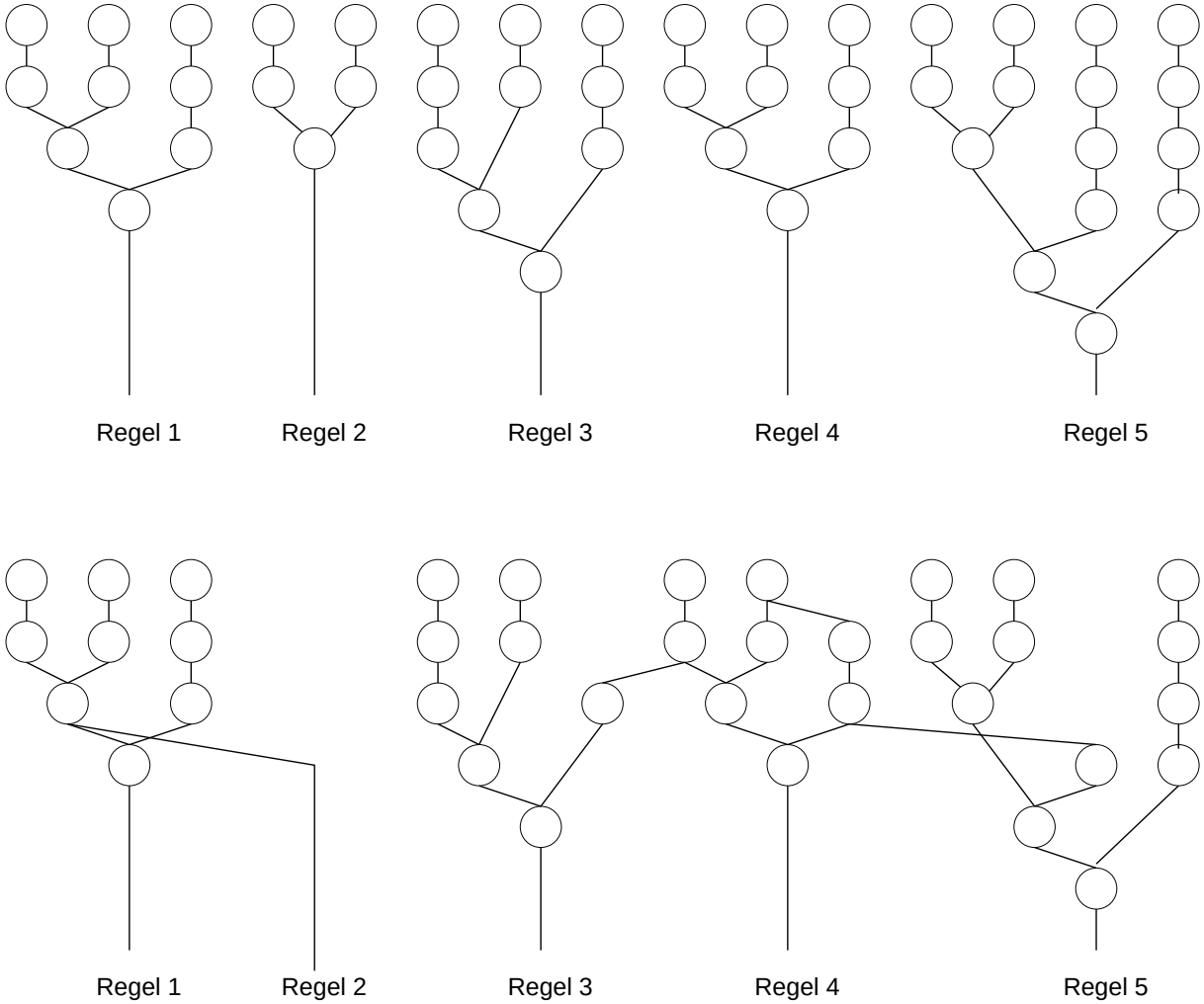
Beispiel 4.9. Das Tupel (A 2) wird zusätzlich in das Working Memory aufgenommen:



Auswertung mehrerer Regeln

- Wenn man die Rete-Netze naiv auf Regelmengen erweitert, würde jede Regel in ein separates Netz übersetzt.
- Das Gesamtnetz für die Regelbasis würde dann aus einer Menge von Einzelnetzen bestehen.
- Der Rete-Algorithmus ist intelligenter: Er verschmilzt gleiche Teile der Einzelnetze.
- Genauer: Er verschmilzt gleiche Anfangsstücke von den Einzelnetzen.

Prinzip:

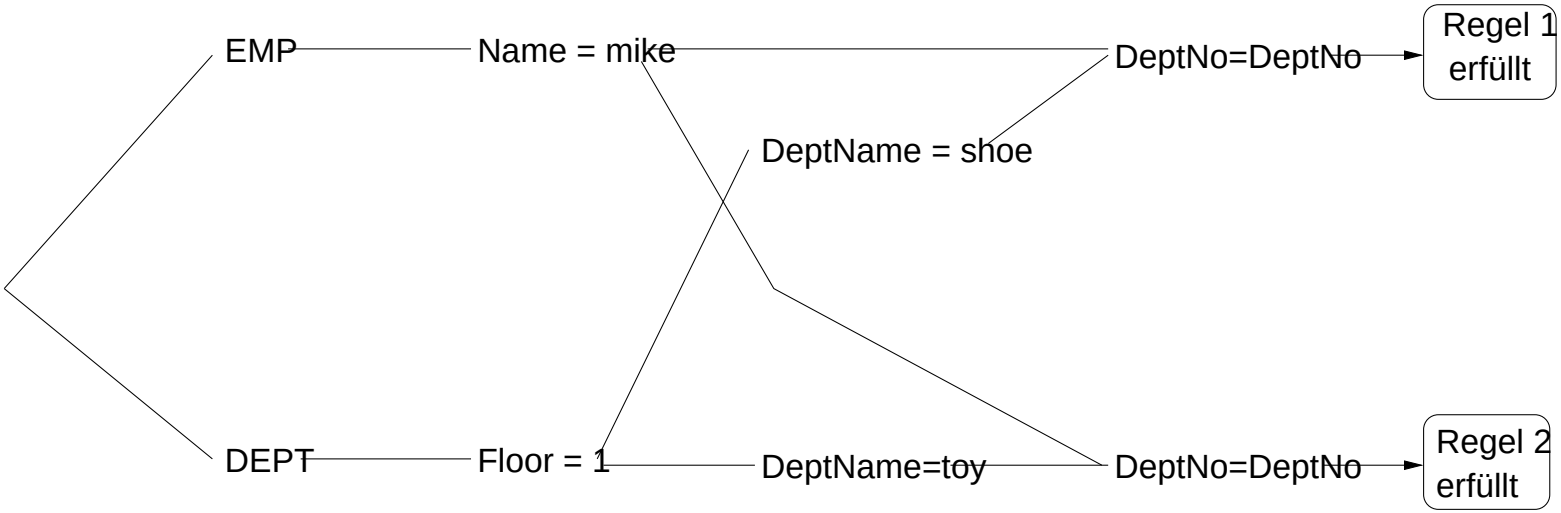

Regel 1
Regel 2
Regel 3
Regel 4
Regel 5
Regel 1
Regel 2
Regel 3
Regel 4
Regel 5

Beispiel 4.10. Gegeben seien die Relationen:
$emp(EmpName, Salary, DeptNo)$,
$dept(DeptNo, DeptName, Floor, Mgr)$ sowie die Regeln:

$$R1 : emp(mike, Salary, DeptNo) \wedge dept(DeptNo, shoe, 1, Mgr) \rightarrow Aktion1$$

$$R2 : emp(mike, Salary, DeptNo) \wedge dept(DeptNo, toy, 1, Mgr) \rightarrow Aktion2$$

Rete-Netzwerk:



Beispiel 4.11.

Wenn der Status der Wettervorhersage *aktuell* ist und
die Quelle der Wettervorhersage die *Rundfunknachrichten* sind und
Regen vorhergesagt wird und
dunkle Wolken beobachtet werden,
dann gebe eine Regenwarnung aus.

Wenn der Status der Wettervorhersage *aktuell* ist und
die Quelle der Wettervorhersage eine *Bauernregel* ist und
Sonnenschein vorhergesagt wird und
der heutige Wochentag ein *Freitag* ist und
der heutige Monat *Juni, Juli, August oder September* ist,
dann gebe eine Ausflugsempfehlung aus.

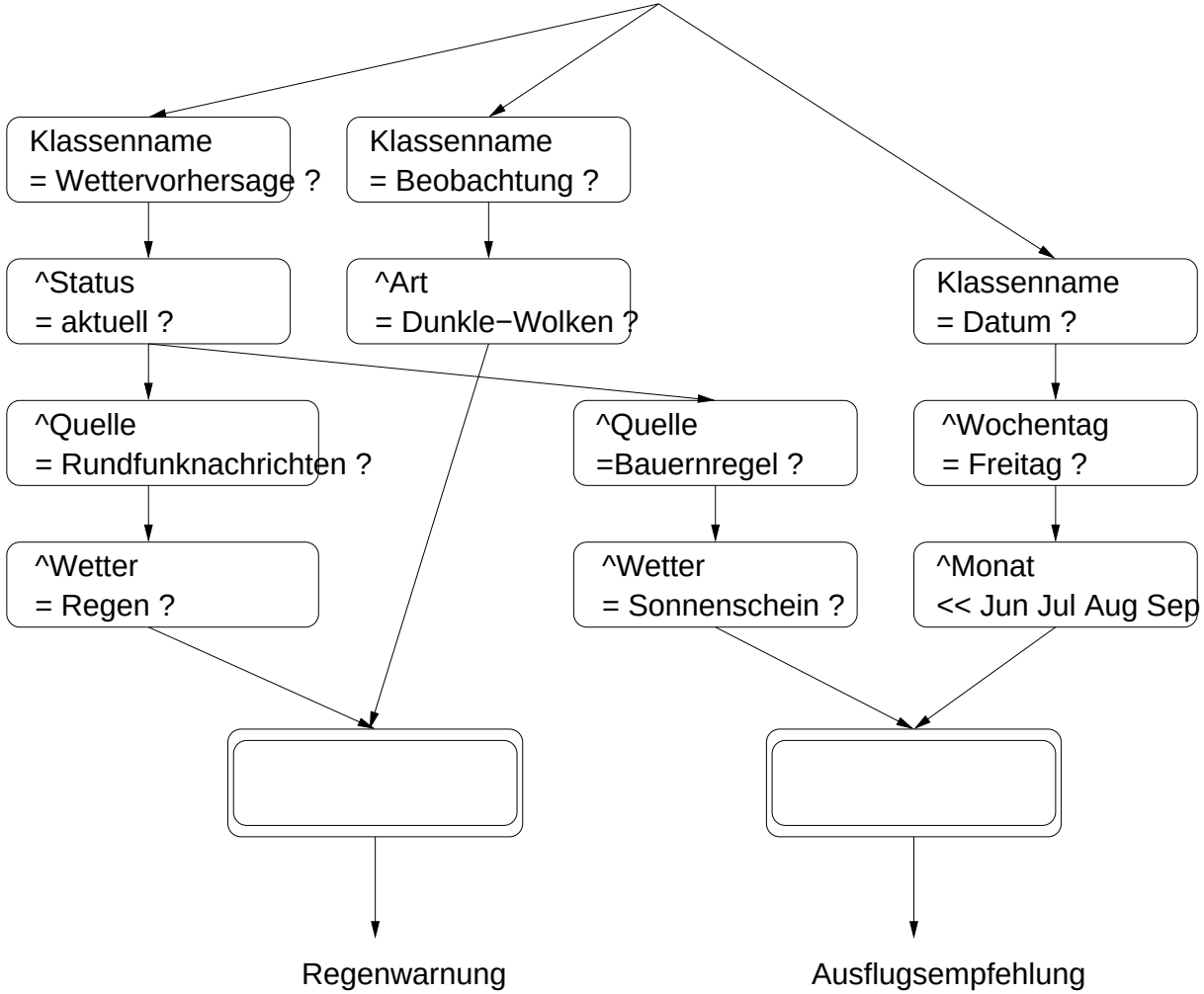
Strukturierung in Klassen (Relationenschema) und Attribute:

Klasse	**Attribute**
Wettervorhersage	Status, Quelle, Wetter
Beobachtung	Art
Datum	Wochentag, Monat

Darstellung der Regeln in OPS5 Syntax:

```
(P Regenwarnung
   (Wettervorhersage
      ^Status aktuell
      ^Quelle Rundfunknachrichten
      ^Wetter Regen)
   (Beobachtung
      ^Art    dunkle-Wolken)
--> (WRITE |Es wird Regen geben.| (CRLF)
           |Ich empfehle, einen Schirm mitzunehmen.|)
)
```

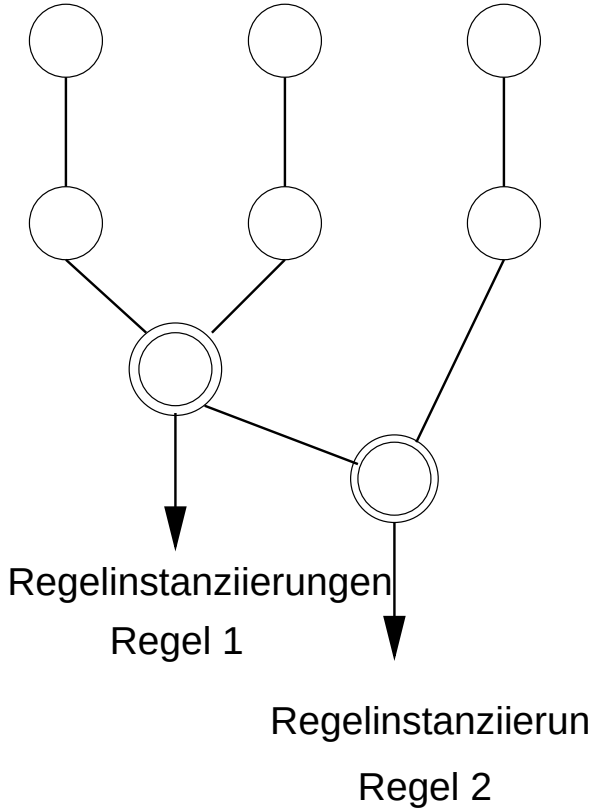
```
(P Ausflugsempfehlung
   (Wettervorhersage
      ^Status aktuell
      ^Quelle Bauernregel
      ^Wetter Sonnenschein)
   (Datum
      ^Wochentag Freitag
      ^Monat << Juni Juli August September >>)
--> (WRITE |Das Wetter wird prima.| (CRLF)
           |Ich empfehle, dieses Wochenende ans Meer zu fahren.|)
)
```


Klassenname
= Wettervorhersage ?
Klassenname
= Beobachtung ?
^Status
= aktuell ?
^Art
= Dunkle−Wolken ?
Klassenname
= Datum ?
^Quelle
= Rundfunknachrichten ?
^Quelle
=Bauernregel ?
^Wochentag
= Freitag ?
^Wetter
= Regen ?
^Wetter
= Sonnenschein ?
^Monat
<< Jun Jul Aug Sep
Regenwarnung
Ausflugsempfehlung

Es kann vorkommen, daß die Regelinstanziierungen am Ausgang eines Knotens gleichzeitig Input für eine anderen Knoten sind.

```
(Regel1
   (Bedingung1 ... )
   (Bedingung2 ... )
--> (Aktionsteil ... )
)

(Regel2
   (Bedingung1 ... )
   (Bedingung2 ... )
   (Bedingung3 ... )
--> (Aktionsteil ... )
)
```



Optimierungen für RETE

Zur Verkleinerung der Zwischenergebnisse an Knoten sollte man die Regeln so spezifisch wie möglich formulieren.

Beispiel 4.12. Regel, die zu Personen den Ort des Arbeitsplatzes ausgibt:

```
(P Ort_des Arbeitsplatzes
   (Person
      ^Name             <Name>
      ^beschaeftigt_bei <Arbeitgeber>)
   (Firma
      ^Name             <Arbeitgeber>
      ^Ort              <Ort>)
--> (WRITE <Name> |arbeitet in| <Ort>))
```

Wenn wir zusätzlich wissen, daß Personen unter 16 und über 65 sowie Schüler und Studenten keinen Arbeitgeber haben, so können wir durch eine spezifischere Regel die Eingabe für den β-Knoten verkleinern:

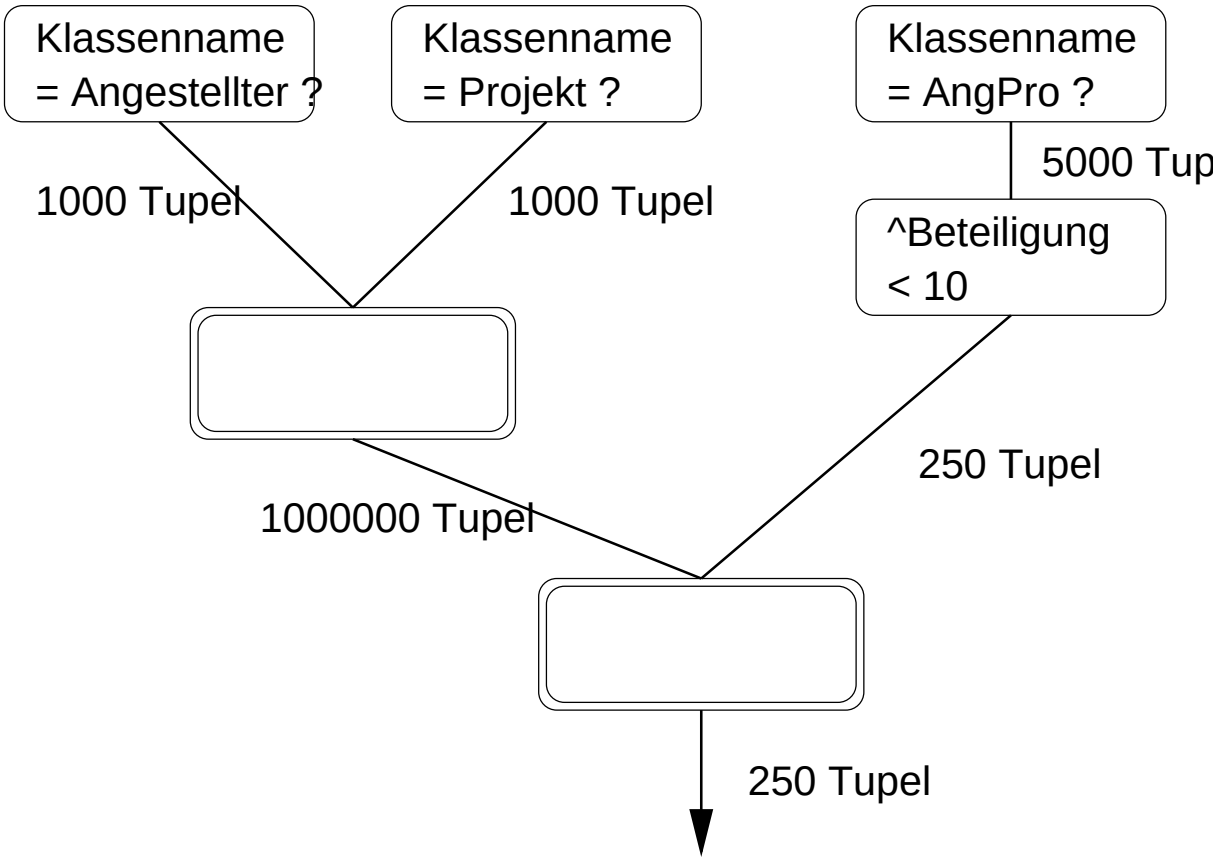
```
(P Ort_des Arbeitsplatzes
   (Person
      ^Name             <Name>
      ^Alter            { >= 16 <= 65 }
      ^Beruf            { <> Schueler <> Student }
      ^beschaeftigt_bei <Arbeitgeber>)
   (Firma
      ^Name             <Arbeitgeber>
      ^Ort              <Ort>)
--> (WRITE <Name> |arbeitet in| <Ort)
)
```

Zwischenergebnisse können sehr groß werden, wenn Bedingungen, die nicht über eine Variable verbunden sind, an einem β-Knoten zusammengeführt werden.

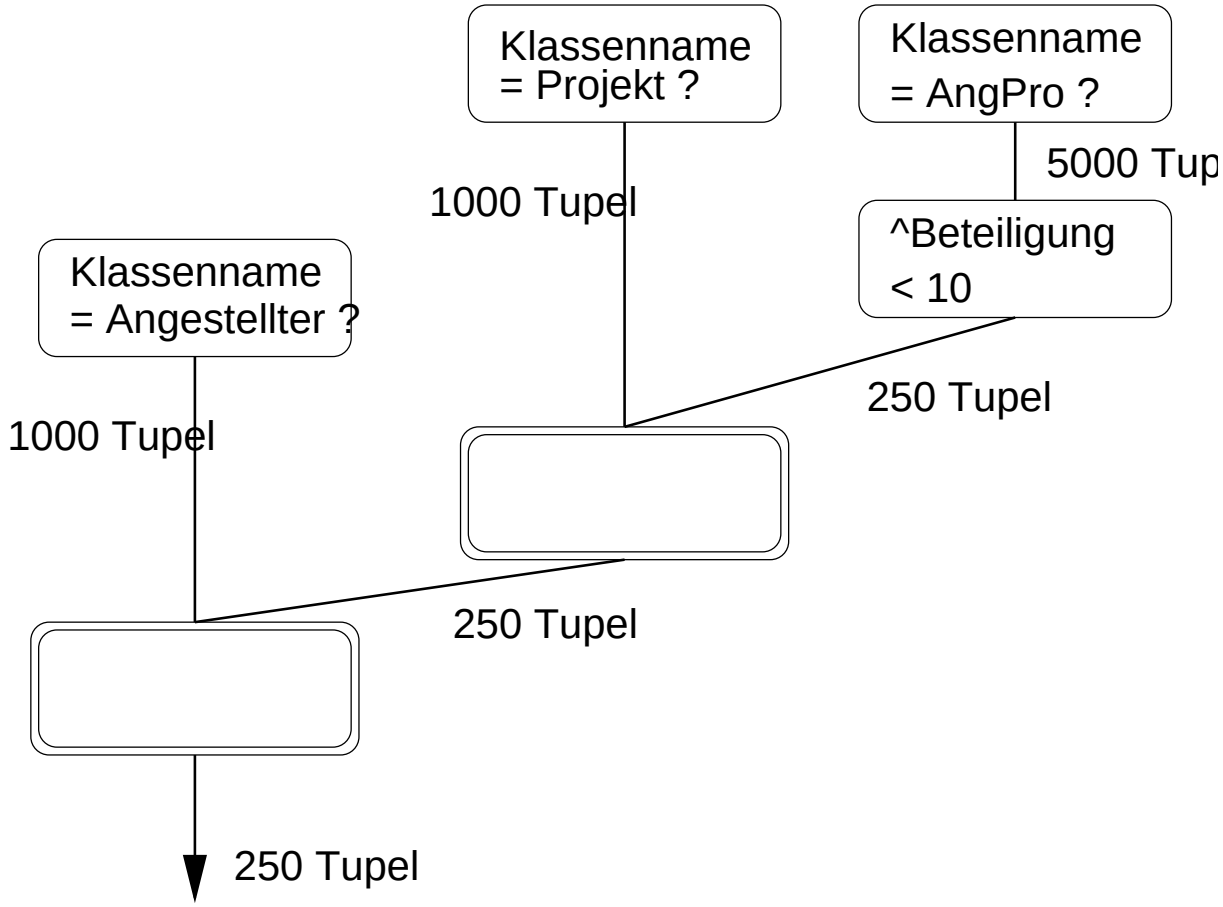
Beispiel 4.13. Die Namen und Projektbezeichnungen zu Angestellten, die weniger als 10% ihrer Arbeitszeit an einem Projekt beteiligt sind, sollen ermittelt werden:

```
(P EliminiereKleineProjektbeteiligung
   (Angestellter
      ^PersNr       <persNr>
      ^Name         <name>)
   (Projekt
      ^ProjNr       <projNr>
      ^Bezeichnung  <bezeichnung>)
   (AngPro
      ^PersNr       <persNr>
      ^ProjNr       <projNr>
      ^Beteiligung  < 10)
--> (WRITE |Angestellter| <name> (CRLF)
           |sollte aus Projekt| (CRLF)
           <bezeichnung> (CRLF)
           |abgezogen werden.|)
)
```

Ein ungünstiges RETE-Netzwerk für diese Regel:



Besser ist das folgende RETE-Netzwerk:



Integration von Regeln in Anwendungen

Bei ILOG JRules besteht eine Regel aus den Komponenten:

- Name
- Priorität
- Menge von Bedingugnen
- Menge von Aktionen

Syntax der Regelsprache:

```
rule ruleName { [priority = value; ]
   [ packet = packetName; ]
   when { conditions ... }
   then { [actions ... ] }
};
```

Beispiel 4.14.

```
rule OccupancyStatus {
   packet = mortage;
   when {
      ?loanApp: Loan( firstHome == true; lienType == FIRSTMORTGAGE );
      Property( occupancyStatus != PRINCIPALRESIDENCE );
   }
   then {
      execute ?loanApp.GenerateMessage(
```

```
        "Occupancy status must be principal residence." );
   }
};
```

- `Loan` und `Property` sind Java-Klassen.
- `firstHome`, `lienType` und `occupancyStatus` sind Instanz-Attribute dieser Java-Klassen.
- Durch `?loanApp` wird eine Variablenbindung eingeführt. Die Variable ist mit dem entsprechenden Java-Objekt verbunden.
- Durch `execute` wird auf dem qualifizierenden Java-Objekt die Methode `GenerateMessage` aufgerufen.

Bedingungen in Regeln:

- Zugriff auf Attribute von Java- oder C++-Objekten
- Java- oder C++-Methodenaufrufe
- boolsche Ausdrücke

- Existenzquantor
- Existentielle Negation: Bedingungung ist war, wenn kein Objekt existiert, das den positiven Teil erfüllt (impliziter Allquantor)
- Anzahlen von Objektinstanzen die eine Bedingung erfüllen

Aktionen:

- Update von Objekt-Attributen
- Anlegen oder Löschen von Objekten im bzw. aus dem Working Memory
- Aktivierung oder Deaktivierung von Regelpaketen
- Lokale Variable deklarieren und initialisieren
- Ausführung von Java- oder C++-Anweisungen

Entwicklungszyklus: