

micro UNIXPERT.2. Ein wissens-
basiertes System zur Online-Fehler-
diagnose in UNIX-Druckaufträgen

Ralph Bergmann

SEKI Working Paper

SWP-89-02

micro UNIXPERT.2. Ein wissens-
basiertes System zur Online-Fehler-
diagnose in UNIX-Druckaufträgen

Ralph Bergmann
SEKI Working Paper SWP-89-02

micro - UNIXPERT.2

Ein wissensbasiertes System zur Online - Fehlerdiagnose

in UNIX-Druckaufträgen

Ralph Bergmann

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1
W.Germany

Mai 1989

SEKI Working Paper 89-02

ABSTRACT

Micro-UNIXPERT.2 is an interactive, online, knowledge-based system, designed for diagnosis of faults in UNIX[^] periphery devices. For micro-UNIXPERT.2 the domain of diagnosis was restricted to faults in the printer periphery and the UNIX spooling system. The diagnostic system has the principal advantage of running in the same environment as the object of analysis. This means, the expert system has direct access to some UNIX-specific system parameters such as the printer environment, the *history* of die operating system, printer queues and some more. These parameters enter the expert system as symptoms, without involving the user.

Micro-UNIXPERT.2 was developed on the basis of the earlier micro-UNIXPERT; drawing on the experience in this special domain of diagnosis. A further goal of micro-UNIXPERT.2 was to represent more explicitly the strategy of "how" questions to the UNIX system itself and to the user were asked and to be more flexible in dealing with uncertain or missing knowledge.

[^] UNIX is a registered trade-mark of Bell Laboratories

Gliederung

1. Einleitung

- 1.1 Die Diagnosedomäne
- 1.2 Diagnostizierbare Fehler
 - 1.2.1 Der Druckbefehl
 - 1.2.2 Die Datei
 - 1.2.3 Der Drucker
 - 1.2.4 Der Spooler
- 1.3 Grober Diagnoseablauf
- 1.4 Der Systemaufbau

2. Die Ein-/Ausgabeschnittstelle

- 2.1 Die System- Ein-/Ausgabe
- 2.2 Die Benutzer Ein-/Ausgabe
 - 2.2.1 Bearbeitung von Fragen an den Benutzer
 - 2.2.2 Analyse der Starkeingabe
 - 2.2.3 Bearbeitung von Fragen an die Erklärungskomponente
 - 2.2.4 Die Analyse von Eingaben

3. Die Vordiagnose

- 3.1 Aufgabenstellung
- 3.2 Der Regelinterpretier
 - 3.2.1 Datenbasis
 - 3.2.2 Regelbasis
 - 3.2.3 Die Regelselektion und Inferenzstrategie
- 3.3 Die Regeln
 - 3.3.1 Die Regelmenge 1
 - 3.3.2 Die Regelmenge 2

4. Die Wissensbasis für die Hauptdiagnose

- 4.1 Allgemeine Übersicht
- 4.2 Regelsyntax
- 4.3 Probleme

5. Der Backward-Regelinterpreter

5.1 Darstellung der Regeln

5.2 Regeln mit Prioritäten

5.3 Modifizierte Logik

5.4 Der Abarbeitungsbaum

6. Die Erklärungskomponente

6.1 Einleitung

6.2 Erklärungskomponente für die Vordiagnose

6.3 Erklärungskomponente für die Hauptdiagnose

7. Implementierung

7.1 Aufteilung in Files

7.2 Anpassen an verschiedene Druckerkonfigurationen

7.3 Aufruf des Systemes

8. Schlußbemerkung

Literatur:

Anhang A: Beispieldialog

Anhang B: Programmlistings

1. Einleitung

Wichtig für die Konstruktion eines Expertensystems ist ein abgegrenzter Problembereich, so daß möglichst wenig Wechselwirkungen nach außen bestehen. Dies ist bei der Diagnose von UNIX-Druckaufträgen der Fall. Dennoch können eine Fülle von Fehlern auftreten und ein unerfahrener Benutzer ist bei der Suche nach Fehlerursachen schnell überfordert. Ein Experte geht nach einer bestimmten Problemlösungsstrategie vor. Er benutzt die vorhandenen Informationen, um die Menge der möglichen Diagnosen einzuschränken und fragt, falls nötig, gezielt nach weiteren Informationen. Micro-Unixpert.2 wurde aufbauend auf micro-UNIXPERT [Lessel 86] entwickelt, so daß bereits auf Erfahrungen mit der Domäne zurückgegriffen werden konnte. Ziel von micro-UNIXPERT.2 war es die Diagnosestrategie des Experten explizit zu modellieren und das System flexibler im Umgang mit unsicherem und fehlendem Wissen zu gestalten, als es in micro-UNIXPERT der Fall war.

1.1 Die Diagnosedomäne.

Micro-Unixpert.2 ist ein Expertensystem, daß für die Diagnose von Fehlern an Peripheriegeräten von UNIX-Maschinen ausgelegt ist. Hier ist die Diagnosedomäne jedoch noch weiter auf die Diagnose von Fehler bei Druckaufträgen unter UNIX eingeschränkt worden. Dabei hat das Diagnosesystem den besonderen Vorteil, daß es auf der zu diagnostizierenden Maschine abläuft und so direkten Zugriff auf bestimmte Systemparameter besitzt, die als Symptome dem Expertensystem so ohne Benutzereingriff zur Verfügung stehen (siehe Kapitel 2.1).

1.2 Diagnostizierbare Fehler

Die von micro-UNIXPERT.2 diagnostizierbaren Fehler lassen sich grob in vier Bereiche einteilen.

1.2.1 Der Druckbefehl

Der UNIX-Druckbefehl hat die Syntax: `lpr [-Option] [Dateiname]`.

Man kann im `lpr`-Befehl Optionen angeben, um die Ausgabe auf dem Drucker zu beeinflussen. Die vollständige Liste der möglichen Optionen ist im UNIX-Manual beschrieben. micro-UNIXPERT.2 beschränkt sich auf die Diagnose der `-P` Option, mit

deren Hilfe es möglich, ist den Ausdruck auf einen bestimmten Drucker zu leiten. Ferner kann in der Datei ".login" ein bestimmter Zieldrucker spezifiziert werden.

Außerdem kennt das System einen Defaultprinter, der angesteuert wird, wenn keine -P Option angegeben wurde und der Datei ".login" kein Drucker spezifiziert wurde. Die -P Option hat Vorrang vor der Voreinstellung in der Datei ".login", die wiederum Vorrang vor dem eingestellten Defaultprinter hat. Fehler können dadurch auftreten, daß der Benutzer die Großschreibung der -P Option übersieht, so daß die Optionsangabe vom lpr-Befehl falsch interpretiert wird. Zur Analyse dieser Fehler ist es notwendig, den eingegebenen Druckbefehl selbst zu analysieren.

1.2.2 Die Datei

Auch in der auszudruckenden Datei können viele Fehler ihre Ursache haben. Wird kein Dateiname angegeben, erwartet das Betriebssystem eine Eingabe vom Terminal. Es erscheint dann kein Prompt.

Das Ausdrucken einer Objektdatei bewirkt ein völlig sinnloses Druckbild. Fehlt am Ende der Datei ein <Return>, so kann es passieren, daß die letzte Zeile der Datei nicht gedruckt wird. Um Dateifehler zu erkennen, kann, nach Kenntnis des konkreten Druckbefehles, die angegebene Datei auf evtl. vorhandene Fehler hin untersucht werden.

1.2.3 Der Drucker

Beim Drucker liegen die meisten Fehlermöglichkeiten. Die Diagnose von Drucker-Hardwarefehlern (Farbband schwach, Papier alle, Drucker auf OFFLINE, etc.) soll dabei möglichst abstrahiert von Druckerdetails gestellt werden, damit micro-UNIXPERT.2 unabhängig vom verwendeten Druckertyp benutzt werden kann. Auf Druckerfehler kann meist nur durch Benutzerangaben und aus dem dynamischen Verhalten des Spooling-Systems (Druckerwarteschlange) geschlossen werden, da der Systemdurchgriff auf die hardwaremäßigen Druckerstatusleitungen nicht möglich ist.

1.2.4 Der Spooler

Um Dateien auszudrucken, benutzt UNIX einen Spoolerdämon. Muß ein Druckjob warten bis ein anderer fertig ist, wird er in eine Warteschlange eingereiht. Tritt in diesem Bereich ein Softwarefehler auf, so ist er meist nur durch einen neuen Systemstart zu

beheben. Spoolerfehler sind schwer zu diagnostizieren, da keine direkten Evidenzen für Spoolerfehler aus dem Spoolingsystem gewonnen werden können und so nur durch die Falsifikation nahezu aller anderen Fehlerursachen auf Spoolerfehler geschlossen werden kann.

1.3 Grober Diagnoseablauf

Der Programmablauf beginnt mit der Aufforderung an den Benutzer sein Problem kurz zu beschreiben. Dies geschieht in einer quasi-natürlichen Sprache (z.B. "Der Ausdruck ist nicht gekommen"). Die Vordiagnose versucht nun aus diesen Angaben einen Diagnosebereich auszuwählen. Falls zu wenig Informationen eingegeben wurden um einen Diagnosebereich auszuwählen, stellt die Vordiagnose weitere Fragen (siehe Kapitel 2). Hat die Vordiagnose ausreichend Informationen um die Diagnosebereichsauswahl erfolgreich zu beenden, so übergibt sie der Hauptdiagnose eine Liste, in der die Diagnosebereiche aufgeführt sind. Zu jedem Bereich wird ein Certainty-Faktor übergeben, der ein Maß für die Wahrscheinlichkeit ist, mit der der Fehler in dieser Klasse liegt. Am Ende der Vordiagnose kann die Erklärungskomponente aufgerufen werden und beantworten, wie es zur Auswahl eines bestimmten Bereiches kam. Die Hauptdiagnose nimmt nun die Fehlerklasse mit dem höchsten Certainty-Faktor und versucht die Diagnose für einen Fehler zu stellen. Gelingt dies, so wird der Benutzer gefragt, ob er mit der Diagnose einverstanden ist. Wenn nicht, versucht das System durch Backtracking eine andere Diagnose zu stellen. Während der Hauptdiagnose kann nach jeder Ausgabe des Systems die Erklärungskomponente aufgerufen werden. Sie ermöglicht einen Trace durch die abgearbeiteten Regeln und gibt Aufschluß darüber, wie eine bestimmte Diagnose zustandekam. Ferner beantwortet sie Fragen nach den ausgeschlossenen Diagnosen und beantwortet "warum-nicht"-Fragen.

Kann micro-UNIXPERT.2 zu dem geschilderten Problem keine Diagnose stellen, so wird der Benutzer an einen lokalen, menschlichen Experten weiterverwiesen.

1.4 Der Systemaufbau

Im Folgenden sei kurz die Modulstruktur von micro-UNIXPERT.2 erläutert, die in Abb.1 in Form eines datenflußähnlichen Diagrammes dargestellt ist. Die Zweiteilung des Diagnoseprozesses in Vor- und Hauptdiagnose spiegelt sich in der Softwarestruktur wieder. Vor- und Hauptdiagnose arbeiten mit einer unterschiedlichen Wissensrepräsentation, Inferenzkomponente und Faktenbasis. Aufgrund der verschiedenen Inferenzmechanismen werden auch unterschiedliche

Erklärungskomponenten benötigt, die dem Benutzer die Beweisabläufe verdeutlichen. Konsultationsspezifische Fakten, die durch die Vordiagnose akquiriert werden und für die Hauptdiagnose weiter relevant sind, werden durch eine Selektionsfunktion ausgewählt und der Faktenbasis der Hauptdiagnose hinzugefügt.

Gemeinsamer Zugriff von Vor- und Hauptdiagnose erfolgt nur auf die Benutzerschnittstelle (Kommunikation : Expertensystem - Mensch), sowie die Systemschnittstelle (Kommunikation: Expertensystem - Spoolingsystem).

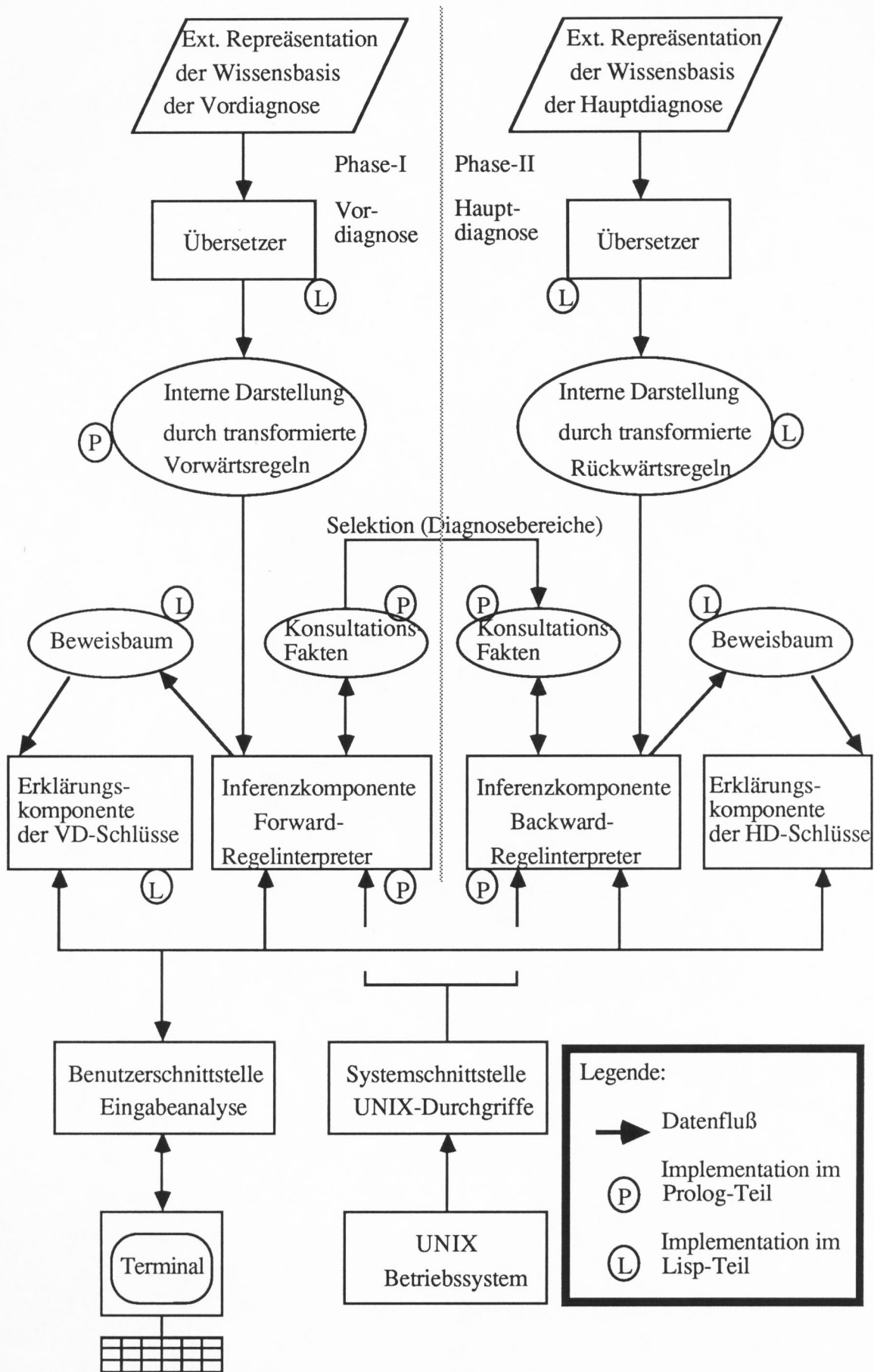


Abb. 1 Modulstruktur von micro-UNIXPERT.2

2. Die Ein-/Ausgabe-Schnittstelle

Die Ein-/Ausgabe ist unterteilt in System- und Benutzer-Ein-/Ausgabe. Aufgabe der System-Ein-/Ausgabe ist es, micro-UNIXPERT.2 mit Informationen über Dateien, Druckerwarteschlangen oder z.B. den letzten Druckbefehl, den der Benutzer abgeschickt hat, zu versorgen. Die Absicht dabei ist es, möglichst viele Informationen zu erhalten ohne den Benutzer befragen zu müssen. Die Benutzer-Ein-/Ausgabe dient dazu, dem Anwender Fragen zu stellen, dessen Antworten auszuwerten und die Resultate den Inferenzkomponenten in Form von Einträgen in die Faktenbasen zur Verfügung zu stellen.

2.1. Die System- Ein-/Ausgabe

Folgende Funktionen werden micro-UNIXPERT.2 durch diesen Programmteil zur Verfügung gestellt:

Funktionsname	Eingabeparameter	Ausgabe
letzter-druckbefehl	-	Der letzte abgeschickte Druckbefehl.
printed-file	-	Der Dateiname im letzten Druckbefehl, falls dieser ordnungsgemäß war. NIL sonst.
printable	-	T, falls printed-file eine druckbare Datei liefert. NIL sonst (z.b. ein Objectfile).
P-option	-	Die Druckerspezifikation im letzten Druckbefehl. NIL, falls keine solche gegeben wurde.
p-option	-	Die p-Option im letzten Druckbefehl. NIL, falls keine solche gegeben wurde.

targetprinter	-	Die Druckerspezifikation des Druckers auf dem der Job wirklich lief.
exists	<filename>	T, falls <filename> existiert. NIL sonst.
filesize	<filename>	Die Grösse von <filename>, falls die Datei existiert. NIL sonst.
default-printer	-	Den durch das Spoolingsystem eingestellten Defaultprinter.
buffersize	-	Ungefäre Größe des Druckerpuffers.
maxsize	-	Maximale Dateigröße.
l-printer	-	Die Druckerspezifikation des im Loginfiles angegeben Druckers. NIL, falls keine solche existiert.
pqueue	<drucker>	Die Druckerwarteschlange des spezifizierten Druckers in Form einer Liste.
pq-change	<drucker-option> <delay-time>	T, falls sich die Druckerwarteschlange des spezifizierten Druckers innerhalb der letzten <delay-time> sec verändert hat. NIL sonst.
file-in-queue	-	T, falls die im letzten Druckbefehl angegebene Datei noch in der Druckerwarteschlange von 'target-printer' steht. NIL sonst.

CRvergessen	-	T, falls in der letzten Zeile der zu druckenden Datei das CR vergessen wurde. NIL sonst.
initEA	-	Initialisiert die System-Ein-/Ausgabe.
cleanup	-	Löscht die von der System-Ein-/Ausgabe angelegten Dateien.

2.2. Die Benutzer-Ein-/Ausgabe

2.2.1. Bearbeitung von Fragen an der Benutzer.

Die Vorgehensweise hierbei ist die folgende:

Durch einen Aufruf von "stelle-frage" mit einem Frageschlüssel wird eine Frage generiert, an den Benutzer ausgegeben, dessen Antwort analysiert und an micro-UNIXPERT.2 in einer internen Darstellung zurückgegeben.

Dabei unterscheidet man zwei Arten von Fragen: ja-nein-Fragen, die auch z.B. mit "Das weiß ich nicht" beantwortet werden können und Fragen, die sich auf einen Drucker beziehen (z.B.: "Auf welchem Drucker sollte der Job laufen?"). Liefert die Analyse keine befriedigenden Ergebnisse, so wird die HELP-Funktion aufgerufen und danach die Frage erneut gestellt. Auf die Vorgehensweise der Analyse wird später noch separat eingegangen.

Ferner ist es möglich, selbständig die HELP-Funktion aufzurufen und WARUM-Fragen (z.B.: "Warum stellst Du diese Frage?") an die Erklärungskomponente zu stellen. Weiterhin kann man noch erfragen, welche Diagnosen bereits ausgeschlossen sind (z.B.: "Welche Diagnosen sind bereits ausgeschlossen?").

2.2.2. Analyse der Starteingabe

Durch den Aufruf der Funktion "PROBLEM" wird der Benutzer nach dem aufgetretenen Fehler gefragt, dessen Antwort analysiert und das Ergebnis der Analyse der Vordiagnose weitergegeben. Dabei kann natürlich auch die HELP-Funktion zu Hilfe genommen werden. Die interne Darstellung dabei ist eine Liste von Paaren <Prädikatsname Certainty-Factor>. Der Certainty-Factor dient dabei zur Unterscheidung, ob das Prädikat TRUE

oder FALSE gesetzt werden muß, z.B.: (druckgekommen -1) bedeutet, daß der Ausdruck nicht angekommen ist.

Außerdem ist es möglich, mehrere Informationen in einem Satz zusammenzufassen, z.B. "Der Ausdruck ist zwar angekommen war aber unleserlich" wird weitergereicht als ((druckgekommen 1) (druckunsauber 1)). Kommata darf man dabei nicht verwenden!

Kann die Analyse keine befriedigenden Informationen extrahieren, wird der Benutzer erneut nach seinen Problemen gefragt.

Außerdem werden dem Anwender nach einer erfolgreichen Eingabe die daraus gewonnenen Informationen nochmals dargestellt und nach einer Quittung gefragt. Fällt diese negativ aus, so beginnt die Startabfrage von Neuem.

2.2.3. Bearbeitung von Fragen an die Erklärungskomponente

Hierbei handelt es sich um Fragen, die unmittelbar nach Ende der Vordiagnose an die Erklärungskomponente gestellt werden. Die HELP-Funktion liefert auch hier die nötigen Erklärungen. So kann man z.B. nach den Wahrscheinlichkeiten für die einzelnen Diagnosebereiche fragen ("Welche waren die 3 wahrscheinlichsten Diagnosebereiche?", "Welche sind die wahrscheinlichen Diagnosebereiche?") oder auch nach einzelnen Diagnosebereichen kann gefragt werden ("Wie groß ist die Wahrscheinlichkeit, daß der Drucker defekt ist?"). Ferner kann man noch nach einzelnen Diagnosen fragen ("Warum ist es unwahrscheinlich, daß die Option falsch war?").

2.2.4. Die Analyse von Eingaben

Die Analyse erfolgt nach folgendem Schema:

Der Eingabesatz wird Wort für Wort verarbeitet. Dabei wird in mehreren Normwörtertabellen nachgesehen, ob das aktuelle Wort in einer der Tabellen enthalten ist. Ist dies der Fall, so wird es durch ein Normwort ersetzt. So werden z.B. "Drucker", "printer", "pRiNter" alle zu "drucker" umgewandelt.

Der daraus entstandene Satz wird dann auf seinen Inhalt geprüft. Bei ja-nein-Fragen wird z.B. nach den Schlüsselwörtern "ja" oder "nein" gesucht, bei Fragen nach einem Drucker nach "B14" (dies ist das Normwort für "bau14" "14" "lpb14").

Etwas umfangreicher ist die Analyse der Starteingabe. Hier sind die Schlüsselwörter "drucker", "unsauber", "druck", "lückenhaft"... .

Analog dazu funktioniert die Analyse der Fragen an die Erklärungskomponente. Der Unterschied liegt hauptsächlich in den anderen Normwörtertabellen.

3. Die Vordiagnose

3.1 Aufgabenstellung

An das micro-UNIXPERT.2-System werden folgende Anforderungen gestellt, die eine eigene Vordiagnose mit Inferenzkomponente notwendig machen :

- Die aus einer allgemeinen Problembeschreibung gewonnenen Informationen sollen genutzt werden, um einen Diagnosebereich zu identifizieren, der den vorliegenden Fehler mit hoher Wahrscheinlichkeit enthält.

Um dies zu erreichen, wird jedem der vier Diagnosebereiche ein Certainty-Factor im Bereich von -1 bis 1 zugeordnet, der die Wahrscheinlichkeit für einen Fehler im Diagnosebereich repräsentiert.

- Reichen die Informationen die aus der allgemeinen Problembeschreibung akquiriert werden nicht aus, so soll durch eine geeignete Nachfragestrategie genügend Wissen erworben werden, um einen Diagnosebereich wahrscheinlich zu machen. Dabei sollte die Fragestrategie möglichst explizit modelliert werden.

- Eine Erklärungskomponente soll am Ende der Vordiagnose die klassische WIE-Frage beantworten, um so dem Benutzer zu erklären, wie ein Diagnosebereich entstanden ist. Aus den Anforderungen ergeben sich folgende Probleme, die durch einen geeigneten Wissensrepräsentationsmechanismus gelöst werden müssen:

- Aufgrund von unvollständiger Information sind viele Schlüsse nur mit einer bestimmten Wahrscheinlichkeit zu ziehen (Certainty-Factors für Hypothesen).

- Die Abwesenheit eines Faktums darf nicht dessen logische Negation implizieren, d.h. "Negation as Failure" und die "Closed World Assumption" darf i.a. nicht zur Modellierung der Negation herangezogen werden.

3.2 Der Regelinterpreter

Die Anforderungen an die Vordiagnose haben zu einer regelbasierten Inferenzkomponente geführt, die nach dem Prinzip des forward-chaining arbeitet. Die Inferenzkomponente hat dabei folgenden Aufbau :

3.2.1 Datenbasis

In der Datenbasis ist der Wissenszustand jedes Faktums gespeichert. Der Zustand wird dabei dadurch repräsentiert, daß jedem Faktum, das in der Datenbasis eingetragen ist, ein Certainty-Factor (CF) im Bereich von -1 bis 1 zugeordnet wird. Je größer dieser CF ist,

desto wahrscheinlicher ist das Faktum. Ein Faktum, das mit einem CF = 0 eingetragen ist, bedeutet, daß keine Information über das Faktum vorliegt, bzw. erschlossen werden kann. Ist ein Faktum noch nicht in der Datenbasis eingetragen, so wurde noch nicht versucht, Information zu gewinnen. Dies ist aber durch Anfragen an den Benutzer (oder durch Systemaufrufe) eventuell noch möglich.

3.2.2 Regelbasis

Die Regeln in der Regelbasis dienen zur Manipulation der Datenbasiselemente und zum Steuern von Aktionen, die z.B. aus Anfragen an den Benutzer oder Systemaufrufen bestehen. Die Regeln werden hierzu in mehrere (hier konkret zwei) Regelmengen unterteilt, die verschiedene Sorten von Regeln enthalten können. Dies sind Regeln, die aus vorhandener Information einen Diagnosebereich identifizieren und Regeln, die bei Bedarf zusätzliches Wissen akquirieren, also die Fragestrategie steuern.

Die Regelbasis hat nun folgende Syntax :

```

<REGELBASIS> ::= { <REGELMENGE> }*

<REGELMENGE> ::= ( ruleset <NAME> )
                  { <REGEL> }*

<REGEL> ::= ( regel <NAME>
              <PRÄMISSEN> <FEUERBEREICH>
              <KONKLUSSION> )

<PRÄMISSEN> ::= ( { <KOND-ELEMENT> }* )

<KOND-ELEMENT> ::= ( factum <PRÄD> ) |
                   ( negativ <PRÄD> ) |
                   ( unknown <PRÄD> ) |
                   ( <LISPLOG-PRÄD> )

<FEUERBEREICH> ::= ( <LOW-LIMIT> <HIGH-LIMIT> )

<KONKLUSSION> ::= ( { <HYPOTHESE> | <AKTION> }* )

<HYPOTHESE> ::= ( hypo <PRÄD> <CF-HYPO> )

```


<AKTION> ::= (frage <PRÄD>) |
 (systemfrage <PRÄD>)

<PRÄD> ::= <0-STELL-PRÄD> | <n-STELL-PRÄD>

<0-STELL-PRÄD> ::= <PRÄDIKATS_NAME>

<n-STELL-PRÄD> ::= (<PRÄDIKATS_NAME> { <TERM> } *n)

<TERM> ::= <LISPLOG-VAR> | <KONSTANTE>

<CF-HYPO>,
 <LOW-LIMIT>,
 <HIGH-LIMIT> : reele Zahlen im Bereich -1 .. 1

Zur Erklärung der Semantik der Regeln :

Eine Regel besteht aus einem Regelnamen (oder Regelnummer), den Prämissen, einem Feuerbereich und den Konklusionen. Dabei bestimmen die Prämissen und der Feuerbereich die Anwendbarkeit der Regel. Eine Regel kann genau dann angewendet werden wenn :

1.) alle Prämissen beweisbar sind
und

2.) die berechnete Gesamtsicherheit der Prämissenliste im Intervall <LOW-LIMIT> .. <HIGH-LIMIT> liegt. Zur Berechnung der Gesamtsicherheit der Prämissenliste wird jeder Prämisse P_i beim Beweis ein Certainty-Factor CF_i zugeordnet. Das Minimum aller CF_i berechnet die Gesamtsicherheit.

Eine Prämisse P_i mit CF_i ist genau dann beweisbar wenn gilt :

- $P_i == (\text{factum } \langle \text{PRÄD} \rangle)$ und $\langle \text{PRÄD} \rangle$ ist in der Datenbasis mit CF_i eingetragen.
- $P_i == (\text{negativ } \langle \text{PRÄD} \rangle)$ und $\langle \text{PRÄD} \rangle$ ist in der Datenbasis mit $-CF_i$ eingetragen.

(Dies ermöglicht die explizite Negation)

- $P_i == (\text{unknown } \langle \text{PRÄD} \rangle)$ und $\langle \text{PRÄD} \rangle$ ist nicht in der Datenbasis eingetragen. Dabei wird $CF_i := 1$ gesetzt.
- $P_i == \langle \text{LISPLOG-PRÄD} \rangle$. $\langle \text{LISPLOG-PRÄD} \rangle$ wird mittels Lisplog bewiesen. Dabei wird $CF_i := 1$ gesetzt.

Ist eine Regel anwendbar, so feuert sie und alle Konklusionen werden bestimmt. Aktionen werden ausgeführt, d.h. die angegebenen Prädikate werden erfragt und in die

Datenbasis eingetragen. Für Hypothesen werden neue Wahrscheinlichkeiten nach folgender Funktion berechnet :

$$CF_{\text{new}} = CF_{\text{old}} + (CF * (1 - CF_{\text{old}})) \text{ falls } CF, CF_{\text{old}} \geq 0$$

$$CF_{\text{new}} = CF_{\text{old}} + (CF * (1 + CF_{\text{old}})) \text{ falls } CF, CF_{\text{old}} < 0$$

$$CF_{\text{new}} = (CF_{\text{old}} + CF) / (1 - \min \{ |CF|, |CF_{\text{old}}| \}) \text{ sonst}$$

Dabei ist CF_{old} der Certainty-Factor mit dem <PRÄD> bereits in der Datenbasis eingetragen ist (sonst $CF_{\text{old}} = 0$) und

$$CF = \min \{ CF_i \mid \text{für alle Prämissen} \} * \langle CF\text{-HYPO} \rangle.$$

3.2.3 Regelselektion und Inferenzstrategie

Die Regeln die auf Anwendbarkeit getestet werden, werden jeweils nur aus einer Regelmenge ausgesucht . Ist eine Regel anwendbar, so feuert sie. Jede Regel feuert höchstens einmal, wodurch die Terminierung des Regelsystems garantiert wird. Die Regelmenge aus der die Regeln ausgesucht werden, wird in folgendem Algorithmus bestimmt, der die gesamte Vordiagnose steuert:

1. Ermittle die essentielle Information aus der Problembeschreibung und trage sie in die Datenbasis ein.
2. Suche solange eine Regel in der Regelmenge R_1 aus und feuere sie, bis in R_1 keine Regel mehr feuern kann.
3. Wende solange wie möglich Regeln aus R_2 an . Falls keine Regel anwendbar war, so terminiere . Sonst weiter bei 4.
4. Wende solange wie möglich Regeln aus R_1 an . Falls keine Regel anwendbar war, so terminiere . Sonst weiter bei 3.

Dabei ist R_1 die Regelmenge die den Diagnosebereich bestimmt und R_2 die Regelmenge die bei Bedarf zusätzliche Prädikate erfragt.

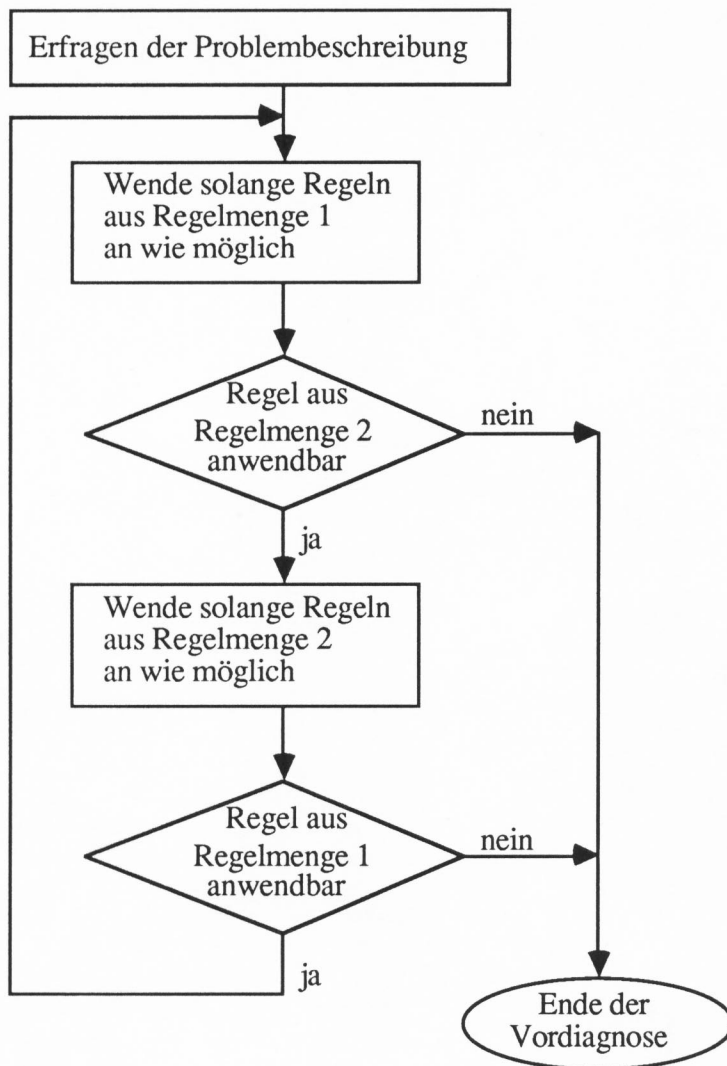


Abb. 2 Abarbeiten der Regelmengen der Vordiagnose

3.3 Die Regeln

Die Regelbasis der Vordiagnose besteht insgesamt aus 23 Regeln. Davon dienen 19 Regeln zur Identifikation eines Diagnosebereiches und 4 Regeln erfragen nach Bedarf zusätzliches Wissen.

3.3.1 Die Regelmenge 1

Die ersten drei Regeln testen, ob der Drucker auf dem der Druck tatsächlich ausgedruckt wurde mit dem Drucker identisch ist, auf dem der Benutzer seinen Druck erwartete. Dabei werden alle Möglichkeiten der Zieldruckerspezifikation getestet (durch eine Option, durch eine Vereinbarung im ".login-file", durch Voreinstellung eines Defaultprinters).

Die nächsten drei Regeln testen den tatsächlichen Zieldrucker und den gewünschten Zieldrucker auf Ungleichheit. Dies ist nicht identisch mit einem Fail der ersten drei Regeln, da keine "Closed World Assumption" gilt. Man benötigt getrennte Regeln um zu zeigen, ob H+ oder H- gilt.

Regel sieben schließt, daß der richtige Drucker angewählt wurde, falls der Benutzer mitteilt, daß der Druck gekommen ist. Die Regeln acht bis zwölf überprüfen, ob ein korrekter Druckbefehl eingegeben wurde. Wenn die Option des Druckbefehls in der Menge der erlaubten Optionen enthalten ist und die auszudruckende Datei existiert, dann ist der Druckbefehl in Ordnung.

In den Regeln 13 bis 19 werden, aufgrund der bis jetzt erschlossenen Fakten die Diagnosebereiche ausgewählt bzw. mit den entsprechenden Certainty-Faktoren belegt. Ist z.B. der gewünschte Zieldrucker ungleich dem tatsächlichen und der Benutzer hat eingegeben, daß der Druck nicht gekommen ist, so wird der Diagnosebereich "fehler-falscher-drucker" mit Wahrscheinlichkeit eins belegt. War der Druck sinnlos, so wird der Bereich "fehler-file" ausgewählt (vermutlich Ausdruck eines Objektfiles). Liegt der Fehler nicht in den beiden genannten Bereichen und der Druckbefehl ist ebenfalls in Ordnung, so wird der Bereich "funktionsstörung-drucker" ausgewählt.

An die Hauptdiagnose wird eine Liste mit den vier Diagnosebereichen und den zugehörigen Certainty-Faktoren übergeben:

((funktionsstoerung-drucker cf1)

(fehler-file cf2)

(fehler-druckbefehl cf3)

(fehler-falscher-drucker cf4))

3.3.2 Die Regelmenge 2

Falls in Regelmenge 1 aufgrund fehlender Information keine Regel gefeuert hat und somit auch kein Diagnosebereich ausgewählt wurde, hat die Regelmenge 2 die Aufgabe, diese Information zu beschaffen. Regel 101 fragt nach, ob der Druck gekommen ist, falls dies unbekannt ist. Regel 102 fragt, wohin der Druck sollte und veranlaßt die Systemfragen zur Überprüfung von Option, login-file und Druckbefehl, falls der Druck nicht gekommen ist. Mit diesen Informationen kann in Regelmenge 1 eine der ersten sechs Regeln feuern und testen, ob der richtige Drucker angewählt wurde. Die Regel 103 veranlaßt eine Systemfrage, ob die auszudruckende Datei vorhanden ist, falls der Druck nicht gekommen ist und der Dateiname ungleich nil war. Eine Überprüfung der Druckerwarteschlange wird von Regel 104 initiiert, wenn der Druck nicht gekommen ist und der Druckbefehl fehlerfrei war. Diese Information wird z.B. in Regel 19 benötigt, um zu testen, ob eine Funktionsstörung des Druckers vorliegt (bei OFFLINE bleibt File in

Warteschlange) oder, ob der Druck auf einem falschen Drucker gelandet ist (Warteschlange dann leer). Am Ende der Regelbasis werden die gültigen Optionen und der Default-Printer als Fakten in die Datenbasis geschrieben. Diese Fakten sind das Vorwissen des Systems.

4. Die Wissensbasis für die Hauptdiagnose

4.1 Allgemeine Übersicht

Die Wissensbasis für die Enddiagnose enthält 84 Regeln. Diese Regeln werden in 4 Diagnosebereiche eingeteilt. Die Prioritäten dieser Bereiche werden von der Vordiagnose geliefert. Insgesamt existieren 20 verschiedene mögliche Enddiagnosen. Zu jeder Enddiagnose enthält das System zum Teil mehrere Abhilfen. Diese geben dem Benutzer Anweisungen, wie er den aufgetretenen Fehler beheben kann. Hierbei wird versucht eine Abhilfe vorzuschlagen, die nicht nur die eigentliche Enddiagnose berücksichtigt, sondern auch die Gesamtsituation beachtet.

4.2 Regelsyntax

Die Regeln in der Regelbasis haben folgendes Format :

```
( <Name_der_Regel> <Priorität>
  <Hypothese>
  <---
  { <Prämisse> }* )
```

In der Regelbasis gibt es verschiedene Typen von Regeln. Zunächst müssen Regeln vorhanden sein, die angeben, welche Enddiagnose in einem Diagnosebereich welche Priorität hat. Zu jeder Enddiagnose existiert also eine Regel folgendem Aussehens :

```
( <Name_der_Regel> <Priorität_der_Enddiagnose>
  ( <Diagnosebereich> (<Enddiagnose> <Parameter>*))
  <---
  (<Enddiagnose> <Parameter>*) )
```


Ferner enthält die Regelbasis (natürlich) die Regeln, die versuchen, eine Enddiagnose zu beweisen. Hierbei gibt es für eine bestimmte Enddiagnose natürlich i.a. mehrere Möglichkeiten. Die Prioritäten an den Regeln, die diese verschiedenen Möglichkeiten repräsentieren, übernehmen dann die Kontrolle, welche Regel bevorzugt feuern soll. Desweiteren müssen Regeln formuliert werden, die angeben, wann eine Enddiagnose definitiv falsch ist. Die Hypothese dieser Regeln hat dann die Gestalt :

`<Hypothese> = (false (<Enddiagnose> <Parameter>*))`

Die Prämissen werden einfach aufgezählt und enthalten einfache Lisplog Ausdrücke. Hierbei gibt es einige Besonderheiten zu beachten. Systemaufrufe werden von der Funktion 'callfun' umschlossen, welche den Wert des Systemaufrufs speichert. Damit werden unnötige und zeitaufwendige Systemaufrufe vermieden. Hierbei muß der eigentliche Systemaufruf 'gequotet' werden,

z.B. `(callfun '(printed-file)).`

Eine ähnliche Aufgabe hat die Einbettung von Fragen in die Funktion 'ask'. Auch hier wird, falls nötig, dem Benutzer die entsprechende Frage gestellt und dann in die Datenbasis eingetragen. Diese Funktion liefert aber IMMER 'true' als Wert zurück, die explizite Abfrage auf das gewünschte Element folgt daher unmittelbar nach der "ask" Funktion,

z.B. `(ask (druckgekommen))`
`(false (druckgekommen)).`

Ferner ist die Auswertung der 'ask' Funktion noch abhängig vom Zustand der TRUE-FRAGE-Unknown-Logik. Bei der TRUE-Logik wird nicht nachgefragt; die entsprechende Regel 'failed' daher sofort. Erst bei der FRAGE- und UNKNOWN-Logik wird die natürlichsprachliche Ein-/Ausgabe bemüht, um den Benutzer zu fragen.

Ansonsten werden für die Prämissen im wesentlichen nur einfache Lisplog Ausdrücke verwendet, die jedem Benutzer verständlich sein sollten, wie etwa 'not', 'is', 'equal', '>' oder '<' .

4.3 Probleme

Ein wesentliches Problem der Erstellung der Regelbasis war natürlich das relativ beschränkte Wissen über die möglichen Fehler. Nachdem die möglichen Enddiagnosen einigermaßen klar waren, ergab sich das schwerere Problem der Verteilung von geeigneten Prioritäten. Da hier jegliche Erfahrungswerte fehlten, wurden diese Prioritäten nach bestem Wissen und Gewissen verteilt.

Als zum Teil unschön erwies sich der Tatbestand, daß die Regeln keine Äquivalenzen sind. Deshalb mußten die Regeln zum Teil einfach umgekehrt noch einmal hingeschrieben werden. Die zusätzliche Einführung der Äquivalenz wäre daher durchaus sinnvoll.

Ein kritischer Punkt bei der Erstellung der Regelbasis war auch die nötige Abstimmung mit der Vordiagnose. Es mußte klargestellt werden, welche Prädikate in den jeweiligen Diagnosebereichen gelten bzw. nicht gelten, um die Regeln nicht unnötig aufzublähen. Dieses ergab insbesondere auch Probleme bei der Untersuchung von mehrfachen bzw. nicht eindeutigen Fehlern.

5. Der Backward-Regelinterpreter

Für die Hauptdiagnose von micro-UNIXPERT.2 wurde ein Backward-Regelinterpreter implementiert. Dieser arbeitet im wesentlichen so wie ein üblicher PROLOG-Regelinterpreter (SLD-Resolution); in folgenden Punkten wurden jedoch vom Prolog-Standard abweichende Forderungen gestellt:

- Steuerung der Abarbeitungsreihenfolge verschiedener Regeln durch Verteilung von Prioritäten an die einzelnen Regeln.

- modifizierte Logik (um Unsicherheit zu berücksichtigen).

- Explizitmachung des Abarbeitungsbaumes für die Erklärungskomponente.

5.1 Darstellung der Regeln

Für die Darstellung der Regeln wurde ein externes Format (bessere Lesbarkeit für den Benutzer) und ein dazu äquivalentes internes Format (adäquate Repräsentation in Lisp) gewählt (zum externen Regelformat siehe Kapitel 4.). Der Regelinterpreter muß die Möglichkeit haben, auf Regeln in der Datenbasis direkt zugreifen zu können. Hierzu gibt es in den meisten PROLOG-Versionen das Prädikat `CLAUSE` : dieses Prädikat liefert dem Benutzer per Backtracking nacheinander alle Regeln der Datenbasis, deren Konklusion mit einem eingegebenen Prädikat unifizierbar ist. Das Prädikat `CLAUSE` gibt es in Lisplog jedoch leider nicht.

Diesem Problem wurde folgendermaßen begegnet:

Alle internen Regeln der micro-UNIXPERT.2-Datenbasis werden in dem Prädikat `RULE` als Facts gespeichert. Das Prädikat `RULE` hat folgende Gestalt:

```
(rule <name> <konklusion> <prämissen>*)
```

Will man also etwa das Prädikat `(father john _x)` beweisen, so muß man `(rule ID (father john _x) . _präm)` aufrufen und erhält in der Variablen `_präm` die Liste aller Prämissen die nötig sind, um obiges Goal zu beweisen.

5.2 Regeln mit Prioritäten

Die Abarbeitungsreihenfolge der Regeln sollte durch die Vergabe von Prioritäten an die einzelnen Regeln gesteuert werden. Sind zwei oder mehr Regeln anwendbar um ein Goal zu beweisen, so wird zunächst die Regel mit der höchsten Priorität ausgewählt, dann (falls failure) die mit der zweithöchsten usw. Dieses Problem wurde so gelöst, daß die Regeln im externen Format zunächst alle eingelesen werden. Danach werden sie absteigend nach ihren Prioritäten geordnet und in dieser Reihenfolge im Prädikat `RULE` in der Datenbasis (nun ohne Prioritäten) abgespeichert.

Beispiel:

Die Regeln (externe Darstellung)

(R1 23

(schlau _x) <--- (intelligent _x))

R2 4711

(schlau _x) <--- (klug _x))

(R3 15

(schlau _x) <--- (helle _x))

werden intern abgelegt als:

(rule R2 (schlau _x) (klug _x))

(rule R1 (schlau _x) (intelligent _x))

(rule R3 (schlau _x) (helle _x))

Die Abarbeitungsstrategie von PROLOG (Regeln werden von "oben" nach "unten" abgearbeitet) sichert nun, daß die wahrscheinlicheren Regeln zuerst angewendet werden.

5.3 Modifizierte Logik

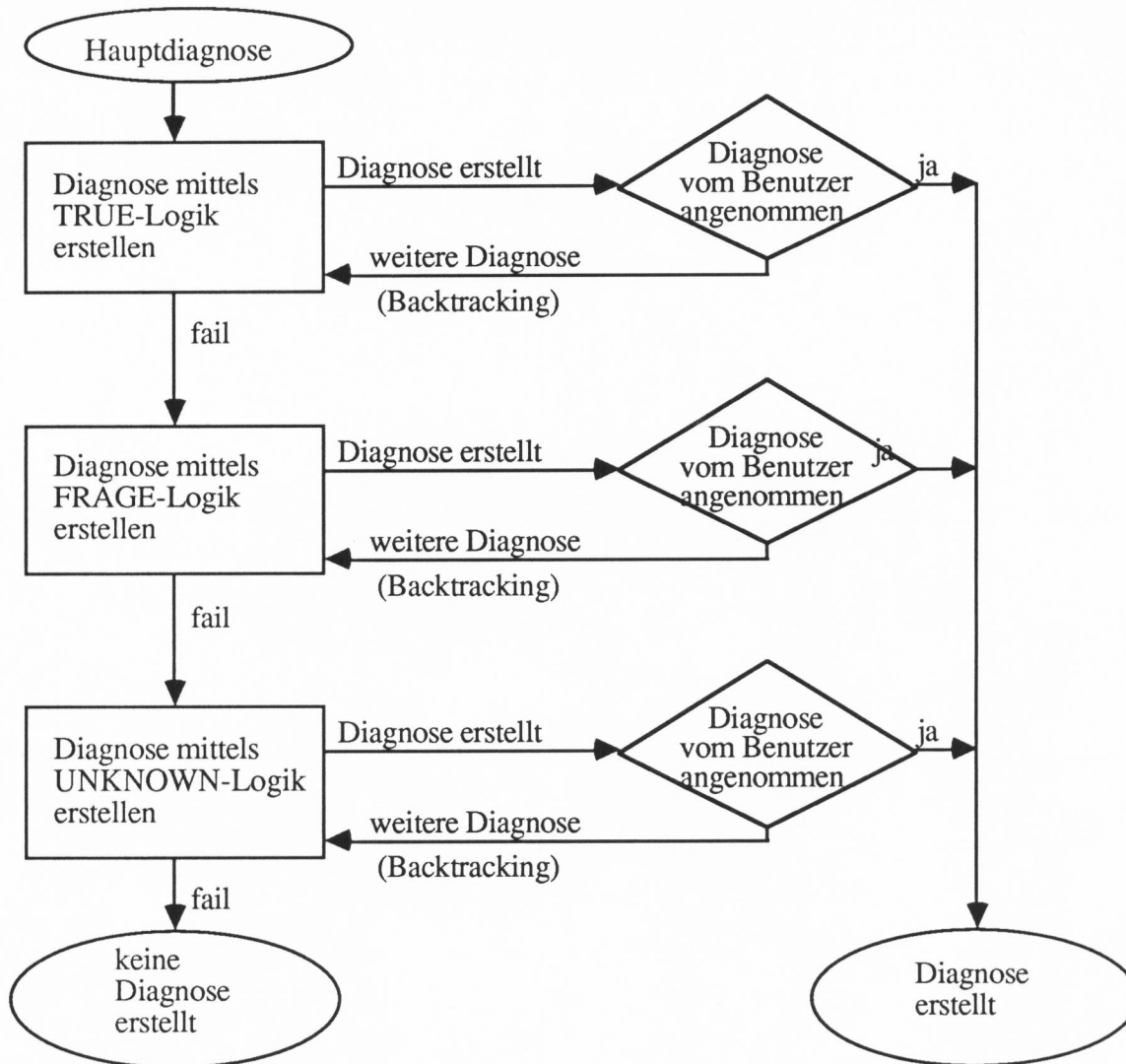
Um mit dem Problem der Unsicherheit fertig zu werden, wurden Prädiakte zur TRUE-, FRAGE- und UNKNOWN-Logik als Bestandteile des Regelinterpreters implementiert. Sie geben jeweils an, wann ein bestimmtes Prädikat SUCCESS liefert:

- a. TRUE(P) gdw P ist Fact in der Datenbasis
 oder Es gibt eine Regel
 P :- A1,...,An mit TRUE(Ai) für alle i

- b. FRAGE(P) gdw P ist Fact in der Datenbasis
 oder P ist nachfragbar beim Benutzer
 und Benutzer sagt,daß P gilt
 oder Es gibt eine Regel
 P :- A1,...,An mit FRAGE(Ai) für alle i

- c. UNKNOWN(P) gdw P ist Fact in der Datenbasis
oder der Benutzer hat gesagt, daß
er nicht weiß, ob P gilt
oder es gibt eine Regel
P :- A1,...,An mit UNKNOWN(Ai) für alle i

Mit diesen Prädikaten kann nun die Toplevelfunktion für die Hauptdiagnose arbeiten. Für die einzelnen Fehlerklassen kann nun nacheinander versucht werden, ob es einen Fehler gibt, der mit TRUE-, FRAGE- oder UNKNOWN-Logik beweisbar ist. Die folgende Abbildung verdeutlicht dabei die Vorgehensweise bei der Beweisfindung.



True-Logik: Hypothese \leftarrow Präm-1,..., Präm-n
 Die Hypothese ist dann bewiesen, wenn alle Prämissen ohne Nachfragen an den Benutzer bewiesen werden können.

Frage-Logik: Hypothese \leftarrow Präm-1,..., Präm-n
 Die Hypothese ist dann bewiesen, wenn alle Prämissen evtl. mittels Nachfragen bewiesen werden können. Stellt sich also eines der Prädikate in den Prämissen als nachfragbar heraus und gibt der Benutzer an, daß das Prädikat gilt, so gilt die Prämisse als bewiesen.

Unknown-Logik: Hypothese \leftarrow Präm-1,..., Präm-n
 Die Hypothese gilt als bewiesen, wenn für alle Prämissen gilt:
 - die Prämisse läßt sich beweisen
 oder
 - die Prämisse ist nachfragbar und der Benutzer gibt an, daß er die Antwort auf die Frage nach dem Prädikat nicht kennt.

Abb.3 Beweisstrategie der Hauptdiagnose

Zunächst wird also versucht, einen Beweis für eine Diagnose nur mittels der TRUE-Logik zu finden. Es werden nur Regeln angewendet, in denen Prämissen vorkommen die bereits als wahr bewiesen wurden (mittels TRUE-Logik) oder sich durch Systemfragen als gültig herausstellen. Fragen an den Benutzer werden dabei nicht gestellt. Eine Diagnose die mittels dieser Logik bewiesen wurde, kann also als sichere Diagnose bezeichnet werden, da Fehlentscheidungen aufgrund von falschen Benutzerangaben ausgeschlossen werden können.

Kann keine durch den Benutzer angenommene Diagnose mittels TRUE-Logik erstellt werden, so wird die FRAGE-Logik angewendet. Es wird also erneut versucht, einen Beweis für einen Fehler zu finden. Trifft jetzt der Resolutionsbeweiser auf eine Regel in der sich eine Hypothese befindet die als nachfragbar angesehen wird, so wird die entsprechende Benutzerfrage gestellt und evtl. die Regel angewendet. Auf diese Weise werden mehr Fehler beweisbar, als mittels der TRUE-Logik beweisbar waren, da jeder Fehler der mittels der TRUE-Logik beweisbar war auch mittels der FRAGE-Logik zu beweisen sein muß.

Wird nun auch durch die FRAGE-Logik kein Beweis gefunden, so findet die UNKNOWN-Logik Anwendung. Es wird nun zusätzlich auch dann eine Regel anwendbar, wenn eine Hypothese nachfragbar ist und der Benutzer die Antwort auf die Frage nicht kennt. In diesem Fall wird das zur Hypothese zugehörige Prädikat als UNKNOWN behandelt und Regeln in denen das Prädikat vorkommt, sind trotzdem anwendbar. So können also zusätzliche Beweise gefunden werden, die aber nicht mehr als sicher angenommen werden können, da fehlende Information quasi defaultmäßig immer so angewendet wird, daß Beweise möglich werden.

5.4 Der Abarbeitungs-Baum

Der Regelinterpreter muß ständig mitprotokollieren, was er bisher getan hat, denn zu jedem Zeitpunkt könnte ja die Erklärungskomponente aufgerufen werden und dann muß verfügbar sein, was sich bisher getan hat. Als geeignete Datenstruktur für die Darstellung des Zustandes des Systems bot sich der abstrakte Datentyp "Baum" an. Ein Knoten des Baumes ist dabei ein Goal P. Die Söhne dieses Goals sind die Prämissen der Regel, die ausgewählt wurde um P zu beweisen.

Beispiel:

Gegeben seien folgende Klauseln:

a :- b , c .
b :- d .
c .
d .

Der zugehörige Baum würde dann wie folgt aussehen:

Knoten : a

1. Sohn : Knoten b

1. Sohn : Knoten d

1. Sohn : nil

2. Sohn : Knoten c

1. Sohn : nil

Dieser Baum wird vom Regelinterpreter dynamisch erzeugt und ständig modifiziert (z.B. werden bei failure-Zweigen die entsprechenden Knoten wieder aus dem Baum gelöscht).

6. Die Erklärungskomponente

6.1 Einleitung

Aufgabe der Erklärungskomponente (EK) ist zunächst, dem Benutzer einen Einblick in die Schlußweisen des Systems zu geben. Außerdem dient sie zur Überprüfung des Deduktionsmechanismus in der Testphase. Auf micro-UNIXPERT.2 bezogen, bedeutet dies, daß die EK erklären können sollte, wie eine bestimmte Diagnose zustande kam oder warum eine bestimmte Frage gestellt wird. Eine komfortablere Eigenschaft ist schon die Beantwortung von sogenannten "Warum nicht ?"-Fragen. Um dies leisten zu können, muß der EK von der Deduktionskomponente eine Art Ablaufprotokoll zur Verfügung gestellt werden, das sie dann durchlaufen kann. Da dabei vorwiegend prozedurale Programmierung erforderlich ist, wurde als Programmiersprache für die EK die LISP-Komponente von LISLOG benutzt.

Wie oben bereits erwähnt, gibt es in micro-UNIXPERT.2 zwei Deduktionsmechanismen:

1) Forward-Chaining für die Vordiagnose (Verdachtsgenerierung)

2) Backward-Chaining für die Hauptdiagnose (Verdachtsbestätigung)

Dies bewirkt das sich die EK auch in zwei (weitgehend unabhängige) Komponenten gliedert :

1) EK für Vordiagnose

2) EK für Hauptdiagnose

Aus Gründen der Zusammengehörigkeit von Vor- und Hauptdiagnose, sowie der Einheitlichkeit der Schnittstelle, gibt es nach außen (zur Ablaufkontrolle) nur eine Toplevel-Funktion "EK", die dann die entsprechenden Unterfunktionen zur Vor- und Hauptdiagnose aufruft.

6.2 Erklärungskomponente für die Vordiagnose

Die Vordiagnose soll aus einer vom Benutzer vorgegebenen Problembeschreibung einen Verdacht generieren, d.h. die Wahrscheinlichkeiten für die einzelnen Fehlerklassen erzeugen. Dies geschieht durch Forward-Chaining, wobei nur dann nachgefragt wird, wenn keine Regel mehr feuern kann (Regelanwendung aus Regelmenge 2), was in der Praxis nur dann vorkommt, wenn vom Anwender fast keine Information gegeben wurde. Deshalb hielten wir es für wenig sinnvoll an diesen Fragestellen eine Schnittstelle zur EK einzurichten (Warum-Frage); es gibt ja fast nichts zu erklären (ein statischer Antwortsatz "zu wenig Information" würde genügen). Der Benutzer erhält die Möglichkeit nach Beendigung der Vordiagnose die EK anzusprechen (siehe Unterfunktion "ende-der-vordiagnose"), wobei die Ein-/Ausgabe die Kontrolle übernimmt (stellt fest, wann der Benutzer mit Hauptdiagnose weitermachen will).

Es gibt dann folgende Fragemöglichkeiten:

a) Fragen zu den Diagnosebereichen:

- die n wahrscheinlichsten Diagnosebereiche
- die Diagnosebereiche mit positivem CF
- die Diagnosebereiche mit negativem CF
- sämtliche Diagnosebereiche mit CF

b) Wie wurde Diagnosebereich D erschlossen ?

(Funktion "ursache-diag")

c) Wie wurde Konditionelement (Prämisse) KE erschlossen ?

(Funktion "ursache-kondelem")

6.3 Erklärungskomponente für die Hauptdiagnose

Die Hauptdiagnose soll nun Diagnosen aus den wahrscheinlichsten Fehlerklassen finden und diese mit Hilfe der TRUE/FALSE/UNKNOWN-Logik bestätigen (siehe Kapitel 5). Dabei können auch, falls mit TRUE-Logik nicht beweisbar, Fragen an den Benutzer gestellt werden. Hier und nach der Diagnoseerstellung soll der Benutzer die Möglichkeit haben die EK anzusprechen.

Insgesamt gibt es 3 Modi innerhalb der EK:

1) WARUM-Modus

(Funktion "In-Diagnose")

Es ist noch keine Diagnose gestellt und an den Benutzer wird vom System eine Frage gerichtet. Mit "warum" kann dieser nun in den WARUM-Modus der EK kommen. Die Steuerung des EA-Dialogs wird dann (durch Menüs) solange von der EK übernommen, bis der Benutzer den WARUM-Modus verlassen will. Im Modus kann er WARUM- und WIE-Fragen stellen, wobei das Menü angibt, wo man sich gerade im Abarbeitungsbaum (Kapitel 5) befindet. Mit WARUM kommt man eine Stufe höher und mit WIE in den entsprechenden Zweig darunter. Außerdem kann man auch mit WIE <prädikat> im Baum umherspringen.

2) WIE-Diagnose-Modus

(Funktion "Diagnose-gestellt")

Die Diagnose ist gerade gestellt worden. Mit "wie" kann der Benutzer nun in den WIE-Diagnose-Modus kommen, um zu erfahren wie die Diagnose zustande kam. Die EA-Steuerung wird dann wie im WARUM-Modus durch Menüs geregelt. Der Anwender kann allerdings nur noch mit "wie" weiterfragen.

3) WIE-AUSGESCHLOSSEN-Modus

(Funktion "wie-ausgeschlossen")

Nach gestellter Diagnose kann man sich mit "ausgeschlossen" die bisher ausgeschlossenen Diagnosen numeriert ausgeben lassen. Durch Eingabe der entsprechenden Zahl kann sich der Benutzer nun informieren, wie ein bestimmter Fehler ausgeschlossen wurde. Er befindet sich also in einer Art WIE-Modus mit einem Abarbeitungsbaum, in dem false(<Diagnose>) statt <Diagnose> bewiesen wurde.

7. Implementierung

Wie bereits in Kapitel 1.4 beschrieben wurde, ist mirco-UNIXPERT.2 in der Sprache LISPLOG einer funktional-logischen Integration von LISP und PROLOG implementiert worden. LISPLOG unterstützt ein Modulkonzept, welches es gestattet die PROLOG-Datenbasis (Fakten und Regeln) in mehrere Einheiten zu splitten um so Zusammenhänge durch Gruppierung besser darzustellen und den Gültigkeitsbereich von Prädikatsnamen einzugrenzen. Im micro-UNIXPERT.2 wurde je ein Modul für die Vordiagnose und eines für die Hauptdiagnose angelegt. Auch wurden die einzelnen Module nochmals in verschiedene Files aufgeteilt. Dabei ist folgende Filestruktur entstanden:

7.1 Aufteilung in Files

1.) Vordiagnose:

rule.inter: Die Regeln der oben beschriebenen Form werden beim Einladen in das System in eine Lisplog-Klauselform übersetzt. Diese Datei enthält den Übersetzer und einige Hilfsfunktion, die bei der Interpretation der übersetzten Regeln benutzt werden.

regelbasis: Diese Datei enthält die eigentliche Regelbasis und einige Lisplog-Prädikate die zur Modellierung der Regeln benutzt werden.

rule.schnitt: Diese Datei enthält Schnittstellenfunktionen zum Backward-Regelinterpreter, zur Erklärungskomponente und zur System-Ein-/Ausgabe.

2.) Hauptdiagnose

bew: Dies ist der Backward-Regelinterpreter mit Prädikaten bew-true (TRUE-Logik), bew-frage (FRAGE-Logik) bew-unknown (UNKNOWN-Logik).

baum: Implementierung des Abstrakten Datentyps "Baum" mit Zugriffsoperationen.

consult: Zugriffsfunktionen zur internen Datenbasis und Transformationsfunktion für Regeln von der externen in die interne Darstellung.

dbase: Datenbasis in externer Darstellung.

dbase-intern: Datenbasis in interner Darstellung.

toplevel: Funktionen zur Ablaufsteuerung der Hauptdiagnose.

ausgabe: Ausgabe-Funktionen für die Hauptdiagnose.

enddiagnose: Ausgabe-Funktionen zur Erstellung der Enddiagnose.

abhilfe: Maßnahmen zur Fehlerbeseitigung.

3.) Benutzer- und System-Ein/-Ausgabe

execfile: Hier sind die Funktionen zur Analyse der Starkeingabe, die Help-Funktion und die Systemschnittstelle implementiert. In dieser Datei werden auch die dem UNIX-System zugängliche Drucker definiert.

4.) Erklärungskomponente

ek: Toplevelfunktionen zur Erklärungskomponente.

ek-ausgabe: Ausgabefunktionen zur Erklärungskomponente.

5.) Hauptsteuerung

unixpert.lader: Diese Datei steuert das Laden der Files für die verschiedenen Systemkomponenten und startet das Expertensystem.

.lisprc.unixpert2: Für ein automatisches Laden von micro-UNIXPERT.2 im LISPLOG-System erforderliche Initialisierungsdatei, die nach dem Starten des LISP-Systemes die LISPLOG-Spracherweiterung einbindet und mit unixpert2.lader startet.

unixpert2b: UNIX-Stapeldatei die das LISP-System startet.

7.2 Anpassen an verschiedene Druckerkonfigurationen

Um micro-UNIXPERT.2 an verschiedene Drucker anpassen zu können, müssen in der Datei für die System-Ein-/Ausgabe "execfile" alle Drucker mit ihren Optionen und Klartextbezeichnungen eingetragen werden. Für die Konfiguration an der Universität Kaiserslautern muß diese Parametrisierung folgendermaßen aussehen:

```
(setq Available-Printer
  '(
    (lp B48 Bau48 48)
    (lpb14 B14 Bau14 14)
    (lp48211 B48211 Bau48-Raum211 48211)
  ))
```

Dies bedeutet:

Es gibt insgesamt drei Drucker, die direkt (nicht über das Ethernet) angesprochen werden können. Dies sind :

Drucker mit Option lp, den der Anwender von micro-UNIXPERT.2 auch synonym als B48, Bau48 oder 48 bezeichnen kann.

Drucker mit Option lpb14, den der Anwender auch als B14, Bau14 oder 14 bezeichnen kann.

Drucker mit Option lp48211, den der Anwender auch als B48211, Bau48-Raum211 oder 48211 bezeichnen kann.

Drucker, die nur über das Ethernet ansprechbar sind, können von micro-UNIXPERT.2 nicht behandelt werden.

7.3 Aufruf des Systems

micro-UNIXPERT.2 kann nach Ausführen des Druckbefehles der fehlerhaft war, direkt mit dem Aufruf : 'UNIXPERT2X' auf UNIX-Kommandoebene gestartet werden. Darauf hin wird, nach Kopieren der aktuellen Systemhistory, das Expertensystem aufgerufen und der letzte Druckbefehl analysiert. Dazu ist es allerdings nötig, daß im ".login" - File des aufrufenden Benutzers ein entsprechender ALIAS gesetzt ist. Dieser lautet:

```
alias unixpert2x '(histroy -r lfgrep lpr > ~/history.file ;/usr/users/unixpert2b/unixpert2b )'
```

8. Schlußbemerkung

Diese Arbeit ist aus einem Expertensystem-Praktikum bei Prof. M.M.Richter im SS 1987 hervorgegangen, an dem neben dem Autor auch Peter Clausen, Christoph Decker, Thomas Ernst, Georg Seul und Ralf Scheidhauer beteidigt waren.

Das beschriebene micro-Unixpert.2 hat bereits teilweise Eingang in micro-Unixpert.M gefunden, das in einer Diplomarbeit von Michael Lessel [Lessel 1988] entwickelt wurde. Mehrere mögliche Weiterentwicklungen der UNIXPERT-Systeme sind zur Zeit in der Diskussion.

Literatur

[Abramson 1984] H. Abramson: A Prological Definition of HASL: A Purely Functional Language with unification-based conditional Binding Expressions. New Generation Computing, Vol. 2, No.1, pp.3-35, 1984. Also in: D. DeGroot & G. Lindstrom and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Allen et al. 1983] J. F. Allen, M. Giuliano, A. M. Frisch: The HORNE Reasoning System. University of Rochester, Computer Science Department, Rochester, NY 14627, TR 126, Dec. 1983

[Bailey 1985] D. Bailey: The University of Salford Lisp/Prolog System. Software - Practice and Experience, 15(6), Juni 1985, pp. 595-609

[Barbuti, Bellia, Levi 1986] R. Barbuti, M. Bellia, G. Levi: LEAF: A Language which integrates Logic, Equations and Functions. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Bellia, Levi 1986] M. Bellia, G. Levi: The Relation between Logic and Functional Languages: A Survey. Journal of Logic Programming, Vol. 3, No. 3, pp. 217-236, 1986.

[Bentley 1975] J. L. Bentley: Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9), Sep. 1975, pp. 509-517

[Bernardi 1986] A. Bernardi: Ein Indexierungskonzept fuer LISPLOG-Datenbasen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP 86-10, Dezember 1986

[Bernardi 1987] A. Bernardi: LISPLOG: LISP/PROLOG-Vereinheitlichung in COMMON LISP auf IBM 6150. Vorfuehrung 63. IBM Hochschulkongress '87, Berlin, Juli 1987.

[Bernardi et al. 1987] A. Bernardi, M. Dahmen, M. Meyer: LISPLOG Benutzerhandbuch. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP 87-01, Januar 1987

[Bernardi & Kammermeier 1987] A. Bernardi, F. Kammermeier: LISPLOG.2: Implementation einer LISP/PROLOG-Vereinheitlichung. Vorstellung des erweiterten Systems in COMMON LISP. 4. Workshop Alternative Konzepte fuer Sprachen und Rechner, Bad Honnef, Maerz/April 1987. Universitaet Muenster, Angewandte Mathematik und Informatik, 2/87-1.

[Bobrow 1984] D. G. Bobrow: If PROLOG is the answer, what is the question? Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 138-145

[Boley 1982/83] H. Boley: Artificial Intelligence Languages and Machines. Universitaet Hamburg, FB Informatik, IFI-HH-8-94/82, Dec. 1982. Also in: Technology and Science of Informatics, 2(3), Mai-Juni 1983, pp. 137-158

[Boley 1983] H. Boley: FIT - PROLOG: A Functional/Relational Language Comparison. Universitaet Kaiserslautern, FB Informatik, interner Bericht 95/83, MEMO SEKI-83-14, Dec. 1983

[Boley 1984] H. Boley: LISP - eine funktionale Einführung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-84-05, September 1984

[Boley 1986a] H. Boley: RELFUN: A Relational/Functional Integration with Valued Clauses. Universitaet Kaiserslautern, FB Informatik, SEKI Report SR-86-04, May 1986. Also in: SIGPLAN Notices 21(12), Dec. 1986, pp. 87-98

[Boley 1986b] H. Boley (Ed.): A Bird's-Eye View of LISP-LOG: The LISP/PROLOG Integration with Initial-Cut Tools. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-08, Dec. 1986

[Boley 1987a] H. Boley: LISP/PROLOG-Kombinatorik. mod-ki message 0043, June 1987.

[Boley 1987b] H. Boley: Fone and Fall: Forward-with-Backward Chaining in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-03, June 1987

[Boley 1987c] H. Boley: Goal: Backward-with-Forward Chaining in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-04, June 1987

[Boley 1987d] H. Boley: FIT: Declarative programming as transformer and adapter fitting. Universitaet Hamburg, FB Informatik, August 1987

[Boley 1987e] H. Boley: Frame and Heir: Clausal Frames and Multiple Inheritance in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-09, November 1987

[Boley & Kammermeier et al. 1985] H. Boley, F. Kammermeier u. die LISPLOG-Gruppe: LISPLOG: Momentaufnahmen einer LISP/PROLOG-Vereinheitlichung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-85-03, August 1985. Short version in: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 36-53

[Boley & Meyer 1986] H. Boley, M. Meyer: Implementation einer LISP/PROLOG-Vereinheitlichung und ihrer Interaktionsumgebung. In: F. Simon (Ed.): Implementierung von funktionalen und logischen Programmiersprachen, Bericht Nr. 8603, Institut fuer Informatik und Praktische Mathematik, Christian-Albrechts-Universitaet Kiel, Mai 1986.

[Boyer & Moore 1972] R. S. Boyer, J. S. Moore: The Sharing of Structure in Theorem Proving Programs. Machine Intelligence 7, Edinburgh 1972, pp. 101-116

[Bourgault et al. 1985] S. Bourgault, M. Dincbas, J. P. Le Pape: THE LISLOG SYSTEM. Centre National d'Etudes des Telecommunications, Note technique NT/LAA/SLC/186, Mai 1985

[Bruynooghe 1982] M. Bruynooghe: The Memory Management of PROLOG Implementations. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 83-98

[Campbell & Hardy 1984] J. A. Campbell, S. Hardy: Should PROLOG be List or Record oriented. In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Carlsson 1981] M. Carlsson: (Re)implementing PROLOG in LISP or YAQ - Yet Another QLOG. UPMAIL Technical Report 5, Uppsala University, Oktober 1981

[Carlsson 1985] M. Carlsson: A Microcoded Unifier for Lisp Machine Prolog. 1985 Symposium on Logic Programming, Juli 1984, Boston, Massachusetts, IEEE Computer Society Press, pp. 162-171

[Charniak & McDermott 1985] E. Charniak, D. McDermott: Introduction to Artificial Intelligence. Addison Wesley, 1985.

[Chester 1979] D. L. Chester: Using HCPRVR. Internal Report, Department of Computer Science, University of Texas at Austin, August 1979

[Chester 1980] D. Chester: HCPRVR: An Interpreter for Logic Programs. 1st NCAI-80, Stanford University, August 1980

[Clocksin & Mellish 1981/84] W. Clocksin & C. Mellish: Programming in PROLOG. Springer Verlag, Berlin Heidelberg New York, 1981. Second Edition 1984

[Cohen 1986] S. Cohen: The APPLLOG Language. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Cohen & Feigenbaum 1982] P. R. Cohen, E. A. Feigenbaum: The Handbook of Artificial Intelligence, Volume III. Pitman, 1982

[Combuechen 1985] M. Combuechen: Symbolics to announce PROLOG Support for the 3600 Family of LISP Machines. SYMBO 10/84 E, European Symbolics Users Newsletter, 1(4), Januar 1985

[Dahmen 1985] M. Dahmen: Ein Translator von CPROLOG nach LISPLLOG. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Dahmen 1986a] M. Dahmen: Iterativer LISPLLOG Interpreter Implementierung, Dokumentation und Evaluation. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-03, Juni 1986

[Dahmen 1986b] M. Dahmen: LAZY-LISPLLOG. Unveroeffentlichtes Manuskript, Universitaet Kaiserslautern, FB Informatik, 1986

[Dahmen, Herr, Hinkelmann, Morgenstern 1985] M. Dahmen, J. Herr, K. Hinkelmann, H. Morgenstern: LISPLLOG: Beitrage zur LISP/PROLOG-Vereinheitlichung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-85-10, November 1985

[Darlington, Field, Pull 1986] J. Darlington, A. J. Field, H. Pull: The Unification of Functional and Logic Languages. In: D. DeGroot & G. Lindstrom (Eds.): Logic Pro-

gramming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Foderaro et al. 1983] J. K. Foderaro, K. L. Sklower, K. Laver: The FRANZ LISP Manual. University of California, Juni 1983

[Fogelholm 1984] R. Fogelholm: Exeter Prolog - some Thoughts on Prolog Design by a LISP User. In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Fribourg 1984] L. Fribourg: Oriented Equational Clauses as a Programming Language. J. Logic Programming, 1984:2, pp. 165-177

[Frisch et al. 1983] A. M. Frisch, J. F. Allen, M. Giuliano: An Overview of the HORNE Logic Programming System. SIGART Newsletter, No. 84, April 1983, pp. 27-19

[Ghallab 1981] M. Ghallab: Decision Trees for Optimizing Pattern-Matching Algorithms in Production Systems. Proc. 7th IJCAI-81, Vancouver, Aug. 1981, pp. 310-312

[Goguen & Mesequer 1984] J. Goguen, J. Mesequer: Equality, Types, Modules, and Generics for Logic Programming. Stanford University, Center for the Study of Language and Information, Report No. CSLI-84-5, March 1984. Also in: J. Logic Programming, 1984:2, pp. 179-210 Also in: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Gray 1984] P.M.D. Gray: Logic, Algebra and Databases. Ellis Horwood Ltd., 1984

[Greussay 1983] P. Greussay: LOVLISP: Une extension de VLISP vers PROLOG. Documentation en ligne. Universite Paris-8 & L.I.T.P., August 1983. Also in: M. Dincbas (Ed.): Programmation en Logique. Actes du Seminaire 1983, Perros-Guirec, Maerz 1983

[Gross 1985] E. Gross: BABYLON-Prolog. In: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 30-35

[Hansson, Haridi, Taernlund 1982] A. Hansson, S. Haridi, S. A. Taernlund: Properties of a Logic Programming Language. In: K. Clark, S. A. Taernlund (Eds.): Logic Programming. Academic Press, London, 1982

[Herr 1985] J. Herr: Breitensuche und Klauselcompilation fuer LISPLUG. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Herr 1986] J. Herr: Ansatz fuer einen LISPLUG Compiler mit LISP als Zielsprache. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-06, Dezember 1986

[Herr 1988] J. Herr: SPADE: A Program Analysing the Stability of Finite Difference Methods for the Numerical Solution of Linear Partial Differential Equations With Constant

Coefficients. Universitaet Kaiserslautern, Fachbereich Informatik, Diplomarbeit, erscheint

[Hinkelmann 1986] K. Hinkelmann: Uebersetzung von LISPLOG-Programmen nach CPROLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-05, September 1986. Erweiterte Fassung April 1987

[Hinkelmann 1988] K. Hinkelmann: SASLOG: Eine funktional-logische Sprachintegration mit Lazy Evaluation und semantischer Unifikation. Universitaet Kaiserslautern, FB Informatik, Diplomarbeit, erscheint

[Hinkelmann & Morgenstern 1985] K. Hinkelmann, H. Morgenstern: Ein Verfahren zur Transformation von LISP-Funktionen in PROLOG-Relationen. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Hinkelmann, Noekel, Reibold 1988] K. Hinkelmann, K. Noekel, R. Reibold: SASLOG: Lazy Evaluation Meets Backtracking. Universitaet Kaiserslautern, FB Informatik, erscheint

[Hoelldobler 1984] S. Hoelldobler: Functional and Logic Programming. Universitaet der Bundeswehr, FB Informatik, Muenchen, Bericht Nr. 8408, Juni 1984

[Hoelldobler, Furbach, Laussermair 1985] S. Hoelldobler, U. Furbach, T. Laussermair: Extended Unification and its Implementation. Universitaet der Bundeswehr, FB Informatik, Muenchen, Bericht Nr. 8504, Juni 1985

[JIPDEC 1980] Interim report on study and research on fifth-generation computers (outline). Japan Information Processing Development Center, 1980

[Kahn 1981] K. M. Kahn: UNIFORM - A Language based upon Unification which unifies (much of) LISP, PROLOG, and ACT 1. Proc. 7th IJCAI-81, Vancouver, Aug. 1981, pp. 933-939

[Kahn 1982] K. M. Kahn: Unique Features of Lisp Machine Prolog. In: Procedures of the Workshop on PROLOG Programming Environments, Linkoeping, March 1982

[Kahn 1983] K. M. Kahn: Unique Features of Lisp Machine Prolog. UPMAIL Technical Report No. 15, Uppsala University, 14. Februar 1983

[Kahn 1983/84] K. M. Kahn: Pure PROLOG in Pure LISP. Logic Programming Newsletter 5, Winter 83/84, pp.3-4

[Kahn 1986] K. M. Kahn: UNIFORM - A Language based upon Unification which unifies (much of) LISP, PROLOG and ACT 1. (Revised version) In: Doug DeGroot & Gary Lindstrom (Eds.): Logic Programming - Functions, Relations and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 411-438

[Kahn & Carlsson 1983] K. M. Kahn, M. Carlsson: LM-Prolog User Manual. UPMAIL, Department of Computing Science, Uppsala University, October 1983

[Kahn & Carlsson 1984] K. M. Kahn, M. Carlsson: How to implement Prolog on a LISP Machine. In: J. A. Campbell

- (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984
- [Kammermeier 1984] F. Kammermeier: Franz-Lisp Mini Handbuch. Universitaet Kaiserslautern, FB Informatik, Juni 1984
- [Kammermeier 1985] F. Kammermeier: Dokumentation der Prolog-Implementation in Franz-Lisp. Universitaet Kaiserslautern, FB Informatik, April 1985
- [Kammermeier 1986] F. Kammermeier: LISPLOG im Kontext anderer LISP/PROLOG-Vereinheitlichungen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-09, Dezember 1986
- [Kluzniak & Szpakowicz 1985] F. Kluzniak, S. Szpakowicz, J. S. Bien (Contr.): Prolog for Programmers. Academic Press, London, 1985
- [Knoepfler & Hotop 1985] S. Knoepfler, S. Hotop: ConProlog - Eine Prolog-Implementation in Interlisp-D. In: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 54-59
- [Komorowski 1982] H. Komorowski: QLOG - The Programming Environment for PROLOG in LISP. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 315-322
- [Kornfeld 1983] W. Kornfeld: Equality for Prolog. Proc. 8th IJCAI-83, Karlsruhe, Aug. 1983, pp. 514-519
- [Kowalski 1983] R. Kowalski: Logic Programming. In: Proc. IFIP, pp. 133-145, Amsterdam, North-Holland
- [Kowalski 1985] R. Kowalski: The Relation between Logic Programming and Logic Specification. In: C.A.R. Hoare, J.L. Shepherdson: Mathematical Logic and Programming Languages, Prentice/Hall International, 1985
- [Lee 1986] N. S. Lee: Programming with P-Shell. IEEE Expert 1(2), Summer 1986
- [Lessel 1986] M. Lessel: micro-UNIXPERT: Ein wissensbasiertes System zur Behandlung von Problemen bei UNIX-Druckauftraegen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-04, Mai 1986
- [Lessel & Boley 1987] M. Lessel, H. Boley: micro-UNIXPERT: Diagnosis of Printer Problems. Universitaet Kaiserslautern, FB Informatik, SEKI Report SR-87-09, August 1987
- [Lessel 1988] M. Lessel: Modellbasierte Diagnose von Benutzer- und Systemsoftware-Fehlern, dargestellt am Beispiel des UNIX-Spoolingsystems. Universitaet Kaiserslautern, FB Informatik, Diplomarbeit Michael Lessel, 1988.
- [Mayer & Richter 1987] O. Mayer, M. M. Richter: Funktional/logische Expertensystemtools. Allgemeine Angaben zum Teilprojekt FLEX. SFB 314, Universitaet Kaiserslautern, FB Informatik, Januar 1987

- [McDermott 1975] D. McDermott: Very Large PLANNER-Type Data Bases. MIT, AI Memo 339, Sep. 1975
- [McDermott 1980] D. McDermott: The PROLOG Phenomenon. SIGART Newsletter, No. 72, Juli 1980, pp. 16-20
- [Mellish 1982] C. S. Mellish: An Alternative to Structure Sharing in the Implementation of a PROLOG Interpreter. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 99-106
- [Meyer 1987] M. Meyer: Entwurf und Implementierung einer Interaktionsumgebung fuer LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-02, Januar 1987
- [Moffat & Gray 1986] D. S. Moffat, P. M. D. Gray: Interfacing Prolog to a persistent Data Store. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 577-584.
- [Nilsson 1984] M. Nilsson: The World's Shortest Prolog Interpreter? In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984
- [Noekel 1985] K. Noekel: SASL: Implementierung eines Reduktionsalgorithmus und Steuerung und Organisation eines Beweissystems fuer eine Logik. Diplomarbeit, RWTH Aachen, Dezember 1985
- [Noekel & Reibold 1986] K. Noekel, R. Reibold: SASL: Implementierung einer rein funktionalen Sprache mit Lazy Evaluation. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-07, November 1986
- [Okuno et al. 1984] H. G. Okuno, I. Takeuchi, N. Osato, Y. Hibino, K. Watanabe: TAO: A Fast Interpreter-Centered System on Lisp Machine ELIS. Conference Record of the 1984 ACM Symposium on LISP and Functional Programming, Austin, Texas, August 1984, pp. 140-149
- [Pereira ohne Datum] F. Pereira (Ed.): CProlog User's Manual. University of Edinburgh, Dept. of Architecture
- [Ramamohanaro & Shepherd 1986] K. Ramamohanaro, J. Shepherd: A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 569-576.
- [Read & Dyer 1985] W. Read, M. G. Dyer: TLOG - Yet another Logic System in Lisp? Tech. Rep. UCLA-AI-85-1, 1985
- [Reddy 1986] U. S. Reddy: On the Relationship between Logic and Functional Languages. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986
- [Reibold 1985] R. Reibold: SASL: Implementierung eines Abstraktionsalgorithmus und Beweisalgorithmen fuer eine Logik. Diplomarbeit, RWTH Aachen, Dezember 1985

- [Robinson 1986] J. Robinson: A Prolog Processor Based on a Pattern Matching Memory Device. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 172-178.
- [Robinson & Sibert 1981] J. Robinson, E. Sibert: The LOGLISP User's Manual. School of Computer and Information Science, Syracuse University, December 1981
- [Robinson & Sibert 1982a] J. Robinson, E. Sibert: LOGLISP: Motivation, Design and Implementation. In: K. Clark, S. A. Taerlund (Eds.): Logic Programming. Academic Press, London, 1982, pp. 299-313
- [Robinson & Sibert 1982b] J. Robinson, E. Sibert: LOGLISP: an alternative to PROLOG. Machine Intelligence 10, Chichester, 1982, pp. 399-419
- [Rulifson et al. 1972] J. F. Rulifson, J. A. Derksen, R. J. Waldinger: QA4: A Procedural Calculus for Intuitive Reasoning. Menlo Park: SRI Technical Note 73, 1972
- [SALFORD 1984] The University of Salford LISP/PROLOG Reference Manual. University of Salford, Second Edition, March 1984
- [Sansonnet 1986] J. P. Sansonnet: The Machine for Artificial Intelligence Applications: MAIA. Proc. 7th ECAI-86, Volume I, Brighton, Juli 1986
- [Sato & Sakurai 1983] M. Sato, T. Sakurai: Qute: A Prolog/Lisp Type Language for Logic Programming. Proc. 8th IJCAI-83, Karlsruhe, Aug. 1983, pp. 507-513
- [Sato & Sakurai 1984] M. Sato, T. Sakurai: QUTE: A Functional Language based on Unification. Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 157-165
- [Schrag 1984] Robert C. Schrag: Compilation and Environment Optimisations for LOGLISP. Rome Air Development Center, In-House Report RADC-TM-84-14, July 1984
- [Subrahmanyam 1984] P. A. Subrahmanyam, J.-H. You: Conceptual Basis and Evaluation Strategies for Integrating Functional and Logic Programming. 1984 International Symposium on Logic Programming, Februar 1984, Atlantic City, New Jersey, IEEE Computer Society Press, pp. 144-153
- [Subrahmanyam, You 1986] P. A. Subrahmanyam, J.-H. You: FUNLOG: A Computational Model integrating Logic Programming and Functional Programming. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986
- [SYMBOLICS 1985] User's Guide to Symbolics Prolog. Symbolics Inc., Cambridge, Massachusetts, June 1985
- [Takeuchi et al. 1983] I. Takeuchi, H. Okuno, N. Ohsato: TAO - A harmonic mean of Lisp, Prolog and Smalltalk. SIG-PLAN Notices, V18 #7, Juli 1983, pp. 65-74
- [VanEmden 1980] M. VanEmden: McDermott on Prolog: A

Rejoinder. SIGART Newsletter, No. 73, Oktober 1980, pp. 19-20

[Voda & Yu 1984] P. J. Voda, B. Yu: RF-Maple: A Logic Programming Language with Functions, Types, and Concurrency. Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 341-347

[Wallace 1983] R. S. Wallace: An Easy Implementation of PiL. SIGART Newsletter, No. 85, July 1983, pp. 29-32

Anhang A:

Beispieldialog-1

--- UNIXPERT2 wird gestartet -----
BITTE WARTEN

Franz Lisp, Opus 38.89 mit Cmu-, Ktu- und Flavors-Erweiterungen 08.02.1985

Patch Nr. 38.3 geladenNr. 38.4 geladenNr. 38.5 geladenSystem geladen.

Version umfasst :

- LISPLOG Interpreter Version 2
- Box Modell Tracer
- Modul System
- schaltbare Quotierungsautomatik
- Streams
- Indexierungssystem

LISPLOG mit: (lisplog)

Loading UNIXPERT2 Version 2.0

consulting vordiagnoseBackward - InterpreterSystem-Ein-AusgabeErklaerungs-
Komponente

```
*****  
*                                     *  
*      UNIXPERT-2                    *  
*                                     *  
*                                     *  
* Diagnose-System zur Fehlersuche    *  
* in UNIX-Druckauftraegen.          *  
*                                     *  
*                                     *  
*****
```

Sie hatten Probleme beim AbsetzenUNIX-Druckauftrages?
beschreiben Sie Ihr Problem: *der druck ist nicht gekommen*

Druck ist nicht angekommen

bestaetigen Sie die Analyse Ihres Problem (ja/nein)

> *ja*

Beginn der Vordiagnose

welchem Drucker sollte der Druck laufen ? help

Drucker sind:

Bau48
Bau14
Bau48-Raum211
Bau34
IRZ

welchem Drucker sollte der Druck laufen ? Bau48-Raum211

Vordiagnose ist beendet ! Noch Fragen ?

> *help*

Die Vordiagnose ist nun beendet: Jeder der folgenden vier Diagnosebereiche mit einem Certainty-Factor zwischen 1 und -1 belegt:

funktionsstoerung-drucker
fehler-file
fehler-druckbefehl
falscher-drucker

koennen Sie sich

, die wahrscheinlichen (positiven) die unwahrscheinlichen (negativen) eine bestimmte Anzahl von Diagnosebereichen (1 - 4) lassen

> *alle*

Die Bereiche sind:

fehler-falscher-drucker 0.855
fehler-file 0
funktionsstoerung-drucker 0
fehler-druckbefehl -0.855
keine weiteren Bereiche

> *fehler-falscher-drucker*

Diagnosebereich wurde gewaehlt, weil folgende Praemissen gelten:

in-warteschlange -1

richtiger-drucker -1

fehler-druckbefehl -0.855

druckgekommen -1

Es gibt keine weiteren Praemissen

> *richtiger-drucker*
(richtiger-drucker)

Regel Nr. 6 hat gefeuert

Es gelten folgende Praemissen:

(ungleich B48 B48211) 1
(wohindruck B48211) 1
(defaultprinter B48) 1
(login nil) 1
(option nil) 1

keine weiteren Praemissen

Aufgrund dieser Praemissen wurden folgende Hypothesen erschlossen:
richtiger-drucker -1
keine weiteren Hypothesen

> *fehler-druckbefehl*

Diagnosebereich wurde gewaehlt, weil folgende Praemissen gelten:

datei-korrekt 0.9

option-korrekt 1

Es gibt keine weiteren Praemissen

> *datei-korrekt*
(datei-korrekt)

Regel Nr. 11 hat gefeuert
Es gelten folgende Praemissen:
(datei-vorhanden xps) 1
(dateiname xps) 1
keine weiteren Praemissen

Aufgrund dieser Praemissen wurden folgende Hypothesen erschlossen:
datei-korrekt 0.9
keine weiteren Hypothesen

> *weiter*

Beginn der Hauptdiagnose

Aufruf Systemfunktion : (default-printer)

Aufruf Systemfunktion : (targetprinter)

=====
Enddiagnose :

Ich vermute folgenden FEHLER:

Der Druck ist zum Defaultprinter gegangen

Als ABHILFE schlage ich vor :

Der richtige Druckbefehl lautet :
lpr -Plp48211 xps
Der Druck liegt jetzt beim default-printer in B48

Ist der Fehler behoben ?

----> *ja*

=====
Ich freue mich,dass ich Ihnen helfen konnte.

=====
Auf Wiedersehen !!!

> exit
exit

script done on Wed Apr 19 13:53:09 1989

Beispieldialog-2

Script started on Wed Apr 19 13:54:04 1989

> *lpr -Plp48211 xps*

> *unixpert2x*

----- UNIXPERT2 wird gestartet -----

BITTE WARTEN

Franz Lisp, Opus 38.89 mit Cmu-, Ktu- und Flavors-Erweiterungen 08.02.1985

Patch Nr. 38.3 geladen

Patch Nr. 38.4 geladen

Patch Nr. 38.5 geladen

LISPLOG System geladen.

Diese Version umfasst :

- LISPLOG Interpreter Version 2
- Box Modell Tracer
- Modul System
- schaltbare Quotierungsautomatik
- Streams
- Indexierungssystem

Starte LISPLOG mit: (lisplog)

Loading UNIXPERT2 Version 2.0

consulting vordiagnose

consulting Backward - Interpreter

consulting System-Ein-Ausgabe

consulting Erklaerungs-Komponente

* * *

* UNIXPERT-2 * *

* * *

* * *

* Diagnose-System zur Fehlersuche * *

* in UNIX-Druckauftraegen. * *

* * *

* * *

Sie hatten Probleme beim Absetzen
eines UNIX-Druckauftrages?

Bitte beschreiben Sie Ihr Problem: *der druck ist nicht gekommen*

Der Druck ist nicht angekommen

Bitte bestaetigen Sie die Analyse Ihres Problemes (ja/nein)

> *ja*

Beginn der Vordiagnose

Auf welchem Drucker sollte der Druck laufen ? 48211

Die Vordiagnose ist beendet ! Noch Fragen ?

> *help*

Die Vordiagnose ist nun beendet: Jeder der folgenden vier Diagnosebereiche wurde mit einem Certainty-Factor zwischen 1 und -1 belegt:

funktionsstoerung-drucker
fehler-file
fehler-druckbefehl
falscher-drucker

Sie koennen Sie sich

alle,
nur die wahrscheinlichen (positiven)
nur die unwahrscheinlichen (negativen)
oder eine bestimmte Anzahl von Diagnosebereichen (1 - 4)

ausgeben lassen

> *alle*

Die Bereiche sind:
funktionsstoerung-drucker 0.684
fehler-file 0
fehler-druckbefehl -0.855
fehler-falscher-drucker -1
keine weiteren Bereiche

> *fehler-druckbefehl*

Diagnosebereich wurde gewaehlt, weil folgende Praemissen gelten:

datei-korrekt 0.9

option-korrekt 1

Es gibt keine weiteren Praemissen

> *funktionsstoerung-drucker*

Diagnosebereich wurde gewaehlt, weil folgende Praemissen gelten:

richtiger-drucker 1

fehler-druckbefehl -0.855

Es gibt keine weiteren Praemissen

> *weiter*

Beginn der Hauptdiagnose

Aufruf Systemfunktion : (targetprinter)

=====

Enddiagnose :

Ich vermute folgenden FEHLER:

Der Drucker B48211 ist ausgeschaltet .

Als ABHILFE schlage ich vor :

Schalten Sie den Drucker ein !
Der Zieldrucker ist B48211
Schicken Sie den Druck dann noch mal ab.

Ist der Fehler behoben ?

----> help

=====

Sie haben die Moeglichkeit folgendes einzugeben:

- ja
- nein (dann versuche ich,einen anderen Fehler zu diagnostizieren)
- aus(geschlossen) (dann erklare ich Ihnen,welche moeglichen Fehler ich bereits ausgeschlossen habe)
- wie (dann erklare ich Ihnen,wie ich auf die Loesung gekommen bin)

----> *wie*

=====

Mit Strategie 1 habe ich bewiesen:

ausgeschaltet(_drucker)

Praemissen :

- (1) B48211 is #(targetprinter)
- (2) not(#(file-in-queue))
- (3) ask(druckgekommen)
- (4) false(druckgekommen)

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit

> 2

***** Systemaufruf *****

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit

> z

Mit Strategie 1 habe ich bewiesen:

ausgeschaltet(_drucker)

Praemissen :

- (1) B48211 is #(targetprinter)
- (2) not(#(file-in-queue))
- (3) ask(druckgekommen)
- (4) false(druckgekommen)

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit

> 4

***** Vordiagnose *****

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit

> z

Mit Strategie 1 habe ich bewiesen:

ausgeschaltet(_drucker)

Praemissen :

- (1) B48211 is #(targetprinter)
- (2) not(#(file-in-queue))
- (3) ask(druckgekommen)
- (4) false(druckgekommen)

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit

> q

=====

Enddiagnose :

Ich vermute folgenden FEHLER:

Der Drucker B48211 ist ausgeschaltet .

Als ABHILFE schlage ich vor :

Schalten Sie den Drucker ein !
Der Zieldrucker ist B48211
Schicken Sie den Druck dann noch mal ab.

Ist der Fehler behoben ?

----> ja

=====

Ich freue mich,dass ich Ihnen helfen konnte.

=====

Auf Wiedersehen !!!

```
exit  
> exit  
exit
```

script done on Wed Apr 19 14:04:27 1989

Anhang B: Programmlisting

```
-----
--- UNIXPERT.2
--- Filename : ABHILFE
-----

; Diese Datei enthaelt zu jedem Fehler, der auftreten
; kann die noetigen Abhilfsmassnahmen

(destroy abhilfe-fehler)
(ass (abhilfe-fehler (traktor-defekt _drucker))
  (not (tab 15)) (princ "Reparieren Sie den Traktor !")
  (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker)
  (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (farband-leer _drucker))
  (not (tab 15)) (princ "Wechseln Sie das Farband aus !")
  (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker)
  (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (papier-alle _drucker))
  (not (tab 15)) (princ "Legen Sie neues Papier ein !")
  (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker)
  (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (papier-falsch-eingelegt _drucker))
  (not (tab 15)) (princ "Legen Sie das Papier richtig ein !")
  (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker)
  (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (auf-OFFLINE _drucker))
  (not (tab 15)) (princ "Schalten Sie den Drucker auf ONLINE !") (not (terpri))
  (not (tab 15)) (princ "Warren Sie, ob der Druck noch kommt. ") (not (terpri))
  (not (tab 15)) (princ "Falls nicht, ist er im Buffer verlorengegangen.") (not (terpri))
  (not (tab 15)) (princ "Sie muessen den Druck also nochmal abschicken."))

(ass (abhilfe-fehler (stecker-draussen _drucker))
  (not (tab 15)) (princ "Stecken Sie den Stecker rein !") (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker) (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (ausgeschaltet _drucker))
  (not (tab 15)) (princ "Schalten Sie den Drucker ein !") (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker) (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))
```

```
(ass (abhilfe-fehler (papierstau _drucker))
  (not (tab 15)) (princ "Beheben Sie den Papierstau !")
  (not (terpri))
  (not (tab 15)) (princ "Der Zieldrucker ist ") (princ _drucker) (not (terpri))
  (not (tab 15)) (princ "Schicken Sie den Druck dann noch mal ab."))

(ass (abhilfe-fehler (druck-nach-default-printer _printer))
  (not (equal (callfun (p-option)) nil))
  (not (tab 15)) (princ "Sie muessen die P-Option angeben ") (not (terpri))
  (not (tab 15)) (princ " und das P gross schreiben !!! ") (not (terpri))
  (call ((wohindruck _ziel)))
  (is_help (inv-P-option _ziel))
  (is_file (callfun (printed-file))))
  (not (tab 15)) (princ "Der richtige Druckbefehl lautet.:" (not (terpri))
  (not (tab 15)) (princ "lpr -P") (princ _help) (princ " ") (princ _file)
  (not (terpri))
  (not (tab 15)) (princ "Der Druck liegt jetzt beim default-printer in ")
  (princ _printer))

(ass (abhilfe-fehler (druck-nach-default-printer _printer))
  (call ((wohindruck _ziel)))
  (is_help (inv-P-option _ziel))
  (is_file (callfun (printed-file))))
  (not (tab 15)) (princ "Der richtige Druckbefehl lautet.:" (not (terpri))
  (not (tab 15)) (princ "lpr -P") (princ _help) (princ " ") (princ _file)
  (not (terpri))
  (not (tab 15)) (princ "Der Druck liegt jetzt beim default-printer in ")
  (princ _printer))

(ass (abhilfe-fehler (druck-nach-login-printer _printer))
  (not (equal (callfun (p-option)) nil))
  (not (tab 15)) (princ "Sie muessen die P-Option angeben !!!") (not (terpri))
  (call ((wohindruck _ziel)))
  (is_help (inv-P-option _ziel))
  (is_file (callfun (printed-file))))
  (not (tab 15)) (princ "Der richtige Druckbefehl lautet.:" (not (terpri))
  (not (tab 15)) (princ "lpr -P") (princ _help) (princ " ") (princ _file)
  (not (terpri))
  (not (tab 15)) (princ "Der Druck liegt jetzt beim login-printer in ")
  (princ _printer))

(ass (abhilfe-fehler (druck-nach-login-printer _printer))
  (call ((wohindruck _ziel)))
  (is_help (inv-P-option _ziel))
  (is_file (callfun (printed-file))))
  (not (tab 15)) (princ "Der richtige Druckbefehl lautet.:" (not (terpri))
  (not (tab 15)) (princ "lpr -P") (princ _help) (princ " ") (princ _file)
  (not (terpri))
  (not (tab 15)) (princ "Der Druck liegt jetzt beim login-printer in ")
  (princ _printer))
```



```

(not (terpri))
(not (tab 15)) (princ "Schicken Sie dann die Datei ") (princ _datei)
(not (tab 15)) (princ " noch mal ab !")

(ass (abhilfe-fehler (datei-zu-gross _datei))
      (not (tab 15)) (princ _datei)
      (princ " in kleinere Teile zerlegen, und einzeln ausdrucken"))

(ass (abhilfe-fehler (datei-nicht-druckbar ID))
      (not (tab 15)) (princ "Keine Abhilfe moeglich ,da")
      (not (terpri))
      (not (tab 15)) (princ "die gewuenschte Datei NICHT druckbar ist !!!"))

(ass (abhilfe-fehler (CR-vergessen _datei))
      (not (tab 15)) (princ "Fuegen Sie in der letzten Zeile der datei ")
      (princ _datei)
      (not (terpri))
      (not (tab 15)) (princ "ein CR ein !"))

(ass (abhilfe-fehler (spooler-defekt))
      (not (tab 15)) (princ "Informieren Sie bitte den Operator !"))

(ass (abhilfe-fehler (falsches-keyword))
      (not (tab 15)) (princ "Das Druck-Keywrd lautet 'lpr' !")
      (not (terpri))
      (not (tab 15)) (princ "Beispiel : lpr -P1pb14 testdatei"))

(ass (abhilfe-fehler (dateiname-fehlt))
      (not (tab 15)) (princ "Bitte Dateinamen spezifizieren !"))

-----
--- UNIXPERT-2
--- Filename : AUSGABE
-----

(defun praed-to-string (p)
  ; transformiert ein beliebiges Lisp-Objekt p
  ; in eine lesbarere Form (Praefix- oder Infix-Schreibweise)
  ; in Form eines Strings

  ; Beispiele : Eingabe: (f a b (g x y))
  ;             Ausgabe: "f(a,b,g(x,y))"
  ;
  ; Eingabe: (= (f a) (f b))
  ; Ausgabe: "f(a)=f(b)"
  (praed-to-string1 (loesche-quote p)))

(defun praed-to-string1 (p)
  (cond ((atom p) (if (eql p 'callfun)
                     "#"
                     p))
        ((member (car p) '(is > < eq equal))
         (concat (praed-to-string1 (nth 1 p))
                 (konvertiere-in-infix (car p))
                 )))

```

```

(ass (abhilfe-fehler (druck-nach-login-printer _printer))
      (call ((wohindruck _ziel))
            (not (tab 15)) (princ "Aendern Sie den Drucker im login-File auf ")
            (princ _ziel) (not (terpri))
            (not (tab 15)) (princ "Der Druck liegt jetzt beim eingestellten login-printer in ")
            (princ _printer)))

(ass (abhilfe-fehler (druck-nach-login-printer _printer))
      (is _default (callfun (default-printer)))
      (call ((wohindruck _ziel))
            (equal _default _ziel)
            (not (tab 15)) (princ "Loesche Drucker im login-File, Druck dann nach ")
            (princ _ziel) (not (terpri))
            (not (tab 15)) (princ "Der Druck liegt jetzt beim login-printer in ")
            (princ _printer)))

(ass (abhilfe-fehler (druck-nach-P-printer _printer))
      (call ((wohindruck _ziel))
            (is _login (callfun (login-printer)))
            (equal _ziel _login)
            (not (tab 15)) (princ "Lassen Sie die -P Option weg.") (not (terpri))
            (not (tab 15)) (princ "Druck dann nach ") (princ _ziel) (not (terpri))
            (not (tab 15)) (princ "Der Druck liegt jetzt (durch -P spezifiziert) in ")
            (princ _printer)))

(ass (abhilfe-fehler (druck-nach-P-printer _printer))
      (call ((wohindruck _ziel))
            (is _default (callfun (default-printer)))
            (equal _ziel _default)
            (not (tab 15)) (princ "Lassen Sie die -P Option weg.") (not (terpri))
            (not (tab 15)) (princ "Druck dann nach ") (princ _ziel) (not (terpri))
            (not (tab 15)) (princ "Der Druck liegt jetzt (durch -P spezifiziert) in ")
            (princ _printer)))

(ass (abhilfe-fehler (druck-nach-P-printer _printer))
      (call ((wohindruck _ziel))
            (is _help (inv-P-option _ziel))
            (is _file (callfun (printed-file)))
            (not (tab 15)) (princ "Der richtige Druckbefehl lautet: ")
            (not (tab 15)) (princ 'lpr -P') (princ _help) (princ " ") (princ _file)
            (not (terpri))
            (not (tab 15)) (princ "Der Druck liegt jetzt (durch -P spezifiziert) in ")
            (princ _printer)))

(ass (abhilfe-fehler (druck-in-queue))
      (not (tab 15)) (princ "WARTEN !!!!") (not (terpri))
      (not (tab 15)) (princ "Der Druck befindet sich noch in der Warteschlange. ")
      (ass (abhilfe-fehler (datei-existiert-nicht ID))
            (not (tab 15)) (princ "Geben Sie den Druckbefehl erneut mit dem richtig ")
            (not (terpri))
            (not (tab 15)) (princ " Dateinamen an !"))

(ass (abhilfe-fehler (datei-zu-klein _datei))
      (not (tab 15)) (princ "Stellen Sie zunaechst den Drucker auf ONLINE, ")
      (not (terpri))
      (not (tab 15)) (princ "falls er nicht schon auf ONLINE steht !"))

```



```

(praed-to-string 1 (nth 2 p))))
((eq? (cdr p) (praed-to-string 1 (car p)))
 (concat "-" (cdr p)))
t
(concat (praed-to-string 1 (car p))
        "-")
(concat
 (cdr (mapcan #(lambda (x) (list " " (praed-to-string 1 x)))
              (cdr p))))
        ""))
)
(defun konvertiere-in-infix (symbol)
  (caseq symbol
    (> " > ")
    (< " < ")
    (is " is ")
    (eq " = ")
    (equal " = "")))
)
(defun loesche-quote (l)
  ; l ist eine verschachtelte Listen-Struktur
  ; Ausgabe ist l nach Loeschen des Atomes quote
  ; Beispiel: Ist l = (a c (quote c d))
  ;           Ausgabe : (a c c d)
  (if (atom l)
      l
      (if (eq (car l) 'quote)
          (loesche-quote (cdr l))
          (mapcar #'loesche-quote l))))
)

```

```

(destroy ausgabe)
; Dieses Praedikat uebernimmt die Ausgabe wenn ein Fehler
; diagnostiziert wurde
(ass (ausgabe_fehler)
     (enddiagnose-fehler_fehler_strings)
     (drucke-enddiagnose_strings)
     (princ ""))

```

```

Als ABHILFE schlage ich vor :
""
(not (tab 15))
(hilfe-ab_fehler)
(princ ""))

```

Ist der Fehler behoben ?

```

----> ""
(is_eingabe (read))
(readline)

```

```

(princ ""
=====)
)
(normiere_eingabe_normiert_fehler)
(cond ((eq _normiert n)
      (nil)
      ((eq _normiert j) t)
      ((eq _normiert aus)
       (setq EK-Baum (vererbe (gt)))
       (wie-aus)
       (n-solutions (ausgabe_fehler) 1))
      ((eq _normiert wie)
       (setq EK-Baum (vererbe (gt)))
       (EK '(Diag-gestellt neues-wie))
       (n-solutions (ausgabe_fehler) 1))))
)

```

```

(destroy helfe-ab)
(ass (helfe-ab _f)
     (n-solutions (abhilfe-fehler _f) 1))
(ass (helfe-ab _f)
     (not (falscher-fehler _f)
          (ass-top (rule user (false _f))
                   (ass (falscher-fehler _f)
                        nil))))
)
(defun readline()
  ; ueberliest alle Zeichen in der Eingabe bis zum naechsten CR
  (if (eq (readc)(ascii 10))
      t
      (readline)))
)

```

```

(defun wie-aus()
  ; ruft die Erklaerungskomponente auf
  ; fuer ausgeschlossene Fehler
  (af
   (princ " Nummer (0 = zurueck) ----> ")
   (let ((h (help (read)))
         (readline)
         (if (neq hilf 0)
             then
             (EK (list 'wie-ausgeschlossen hilf))
             else
             t)))
     (defun gt() *gt*)
     (defun drucke-enddiagnose (strings)
       (princ ""
=====)
)
)

```

```

""
(princ-breit "Enddiagnose : ")(terpri)

```



```

(princ "
Ich vermute folgenden FEHLER:
")
(mapc #(lambda (x) (tab 15) (princ x)(terpri)
strings))

(destroy normiere)
(ass! (normiere y j ID))
(ass! (normiere yes j ID))
(ass! (normiere ja j ID))
(ass! (normiere j j ID))
(ass! (normiere n n ID))
(ass! (normiere no n ID))
(ass! (normiere nein n ID))
(ass! (normiere aus aus ID))
(ass! (normiere wie wie ID))
(ass! (normiere wie-aus aus ID))
(ass! (normiere wa aus ID))
(ass! (normiere ausgeschlossen aus ID))
(ass! (normiere a aus ID))
(ass (normiere _wort _x _f)
(if (not (member _wort (h help hilfe ?)))
then
(princ "
Ihre Eingabe ")
(princ _wort)
(princ " verstehe ich nicht.")
else
t)
(princ "

Sie haben die Moeglichkeit folgendes einzugeben:
- ja (dann versuche ich,einen anderen Fehler zu
diagnostizieren)
- aus(geschlossen) (dann erkläre ich Ihnen,welche moeglichen
Fehler ich bereits ausgeschlossen habe)
- wie (dann erkläre ich Ihnen,wie ich auf die
Loesung gekommen bin)

Bitte wiederholen Sie die Eingabe.
:")
(readline)
(n-solutions (ausgabe _f 1))

```

```

(defun ausgabe-ausgeschlossen-fehler (fehlerliste)
(princ "
Von den folgenden Fehlern weiss ich bereits,
dass sie nicht gelten:
")
(let ((hilf 0))
(mapc #(lambda (x) (princ (concat " ("

```

```

(setq hilf (1 + hilf))
")
(prac-to-string x)
(ascii 10)))

fehlerliste))
(princ "
t)

(defun af()
(ausgabe-ausgeschlossen-fehler (ausgeschlossene-fehler)))
-----
--- UNIXPERT.2
--- Filename : BAUM
-----

```

```

; Diese Datei enthaelt Funktionen zur Konstruktion und zum Zugriff
; auf Baeume.Ein Baum wird vom Regelerpreter (Datei 'bew') kon-
; struiert: mit diesem Baum arbeitet dann die Erklaerungskomponente
; weiter.
; Die Knoten des Baumes enthalten jeweils einen Regelnamen und Praedikate.
; Die Soehne eines Knotens geben an,wie der Vater mit dieser
; Regel hergeleitet wurde.
; Beispiel: folgende Regeln seien gegeben:

; Regel1 : A <--- B , C .
; Regel2 : B <---
; Regel3 : C <---

; Der zu A gehoerige Baum haette dann
; als Wurzel den Knoten (Regel1 A) und
; als linken Sohn (Regel2 B) und
; als rechten Sohn (Regel3 C) .

; Baeume werden dargestellt als 3-elementige Liste:
; 1.Element ist ein Zeiger auf die Liste der Soehne
; 2.Element ist ein Zeiger auf den Vater des Knotens
; (es ist also moeiglich im Baum noch unten UNd nach
; oben zu laufen)
; 3.Element ist ein Zeiger auf den Inhalt (Liste aus
; 2 Elementen der Form (Regelname Praedikat) )

```

```

(defun make-tree (inhalt soehne)
; Eingabe : INHALT ist ein beliebiges Lisp-Objekt
; : SOEHNE ist eine Liste mit bel. Lisp-Objekten
; Wert : es wird ein Baum/Knoten mit Inhalt INHALT und
; : den Soehnen SOEHNE erzeugt
; (let ((hilf (list nil nil inhalt)))
; (mapc #(lambda (sohn)

```



```

(insert-son hilf sohn)
  soehne
  hilf)
(defun mache-baum (inhalt soehne)
  (let (hlf (list soehne nil inhalt))
    (mapc #'(lambda (sohn)
              (setf (cadr sohn) hilf))
          soehne)
    hilf))

```

```

(defun inhalt (knoten)
  ; Eingabe: KNOTEN ist ein Knoten
  ; Wert : Inhalt des Knotens KNOTEN
  (caddr knoten))

```

```

(defun father (knoten)
  ; Eingabe: KNOTEN ist ein Knoten
  ; Wert : Vater des Knotens KNOTEN
  (cadr knoten))

```

```

(defun sons (knoten)
  ; Eingabe: KNOTEN ist ein Knoten
  ; Wert : Liste der Soehne des Knotens KNOTEN
  (car knoten))

```

```

(defun nth-son (n knoten)
  ; Eingabe: KNOTEN ist ein Knoten
  ; N ist eine numerische Zahl
  ; Wert : n-ter Sohn des Knotens KNOTEN
  (nth (- n 1) (sons knoten)))

```

```

(defun ausgabe-baum (baum)
  ; Eingabe: BAUM ist ein Knoten
  ; Wert : der Baum BAUM wird in eine externe, nicht zyklische Darstellung
  ; transformiert: extern wird ein baum als Liste dargestellt,
  ; wobei das erste Element der Liste der Inhalt der Wurzel ist
  ; und die folgenden Elemente sind dann wieder die Soehne in
  ; externer Darstellung.
  (cons (inhalt baum)
        (mapcar #'ausgabe-baum
                (sons baum))))

```

```

(defun insert-son (knoten sohn)
  ; Eingabe: ein Knoten KNOTEN und ein beliebiges Lisp-Objekt SOHN
  ; Wert : als letzter Sohn von KNOTEN wird ein neuer Knoten mit Inhalt
  ; SOHN eingefuegt
  (rplaca knoten
           (nconc (car knoten) (list (list nil knoten sohn))))))

```

```

(defun delete-last-son (knoten)
  ; Eingabe: ein Knoten KNOTEN
  ; Wert : der letzte Sohn von KNOTEN wird destruktiv geloescht
  (rplaca knoten (delete-last (sons knoten))))
(defun delete-last (liste)

```

```

; Eingabe: eine Liste LISTE
; Wert : die Liste LISTE ohne ihr letztes Element
; (vgl. 'butlast' in Common-Lisp)
; (if (null (cdr liste))
    nil
    (cons (car liste)
          (delete-last (cdr liste))))))

```

```

(defun root (knoten)
  (if (null (father knoten))
      knoten
      (root (father knoten))))

```

```

(defun vererbe (x) x)

```

```

--- UNIXPERT.2
--- Filenname : BEW

```

```

; Diese Datei enthaelt einen Regelinterpreter.
; es wurden 3 Praedikate implementiert, die angeben, wann ein
; Praedikat P SUCCESS liefert:

```

```

; 1. bew-true:
; P gilt gdw 1) P ist factum
; oder 2) Es gibt eine Regel P <--- A1, ..., An
; wobei alle Ai als true beweisbar.

```

```

; 2. bew-frage:
; P gilt gdw 1) P ist factum
; oder 2) P ist nachfragbar und der Benutzer sagt,
; dass P gilt.
; oder 3) Es gibt eine Regel P <--- A1, ..., An
; wobei alle Ai mit bew-frage beweisbar.

```

```

; 3. bew-unknown:
; P gilt gdw 1) P ist factum
; oder 2) P ist nachfragbar und der Benutzer hat gesagt,
; dass P gilt oder unknown ist.
; oder 3) Es gibt eine Regel P <--- A1, ..., An
; wobei alle Ai mit bew-unknown beweisbar.

```

```

; Der Regelinterpreter speichert seinen aktuellen Zustand in einem Baum.
; Dabei ist die globale Variable *akt-node* ein Zeiger auf den Knoten, der
; gerade bearbeitet wird. Dies gewährleistet, dass die Erklarungskompo-
; nente zu jedem Zeitpunkt rekonstruieren kann, wie der Regelinterpreter
; in seinem momentanen Zustand gekommen ist. (Nacheres zum Zugriff auf
; Baume siehe Datei 'baum')

```

```

(defun gt(*gt*) ; *gt* ist ein Zeiger auf den globalen Baum
  (defun akt-node(*akt-node*)

```



```

(question_p _frage-form)
(bew-frage _rest))

(destroy askable)
; (askable p) gdw P ist nachfragbar
; (ass (askable (druckgekommen)))
; (ass (askable (wohindruck ID)))
; (ass (askable (Online)))
; (ass (askable (druckunsauber)))
; (ass (askable (drucksinnlos)))
; (ass (askable (drucklueckenhaft)))

(defun ask (p)
; ask testet, ob das Eingabe-Praedikat P schon nachgefragt wurde
; und tut dies, falls man sich in der Toplevel-Schleife im
; Beweiszustand 'Beweisen durch Nachfragen befindet'
; (if (n-solutions ` (schon-gefragt .p) 1)
t
  (if (= 1 (waschgang (gt)))
      nil
      (question p))))

(destroy schon-gefragt)
; schon-gefragt testet, ob das Eingabe-Praedikat P
; schon nachgefragt wurde
; (ass (schon-gefragt _p)
  (call (_p)))
; (ass (schon-gefragt _p)
  (call ((false _p))))
; (ass (schon-gefragt _p)
  (call ((unknown _p))))

(defun question (praed)
; question fragt das Praedikat praed nach und traegt
; die Antwort des Benutzers in die Datenbasis ein.
; (let ((antwort (caar (stelle-frage (car praed)))))
  (cond ((= antwort 'warum)
        (setq EK-Baum (vererbe (gt)))
        (EK '(In-Diag neues-warum))
        (question praed))
        ((= antwort 'ausgeschlossen)
        (setq EK-Baum (vererbe (gt)))
        (wie-aus)
        (question praed))
        ((= (car praed) 'wohindruck)
        (if (eq antwort 'unknown)
            (assert 'user '(wohindruck antwort 1)))
            (assert 'user '(wohindruck antwort 1)))
        (t (cond ((= antwort 'ja) (assert 'user praed 1))
                  ((= antwort 'nein) (assert 'user praed -1))
                  ((= antwort 'unknown) (assert 'user praed 0) )))))
t)

```

```

(destroy beweise)
; dieses Praedikat wird fuer die Erklaerungskomponente
; benoetigt: wird eine wie-Frage gestellt, so wird das nachgefragte
; Praedikat noch einmal mittels 'beweise' bewiesen.
; (ass (beweise _f)
  (dbereich_diag-bereich _f)
  (set-wg (1 _diag-bereich))
  (bew-true (_f)))
; (ass (beweise _f)
  (dbereich_diag-bereich _f)
  (set-wg (2 _diag-bereich))
  (bew-frage (_f)))
; (ass (beweise _f)
  (dbereich_diag-bereich _f)
  (set-wg (3 _diag-bereich))
  (bew-unknown (_f)))

(destroy bew-true)
; (ass (bew-true nil))
; (ass ! (bew-true ((ask _p) . _rest))
  (mache-knoten (general-database (ask _p)))
  (ask _p)
  (bew-true _rest))
; (ass (bew-true (_p . _rest))
  _p
  (mache-knoten (general-database _p))
  (bew-true _rest))
; (ass (bew-true (_p . _rest))
  (rule _name _p . _rest)
  (mache-knoten (_name _p))
  (akt-node-down)
  (bew-true _rest))
; (akt-node-up)
; (bew-true _rest))

(destroy bew-frage)
; (ass (bew-frage nil))
; (ass (bew-frage (_p . _rest))
  (mache-knoten (general-database _p))
  _p
  (bew-frage _rest))
; (ass (bew-frage (_p . _rest))
  (rule _name _p . _rest)
  (mache-knoten (_name _p))
  (akt-node-down)
  (bew-frage _rest))
; (akt-node-up)
; (bew-frage _rest))

; (ass (bew-frage (_p . _rest))
  (askable _p)
  (not (call ((unknown _p))))
  (not (call ((false _p))))
  (not (call (_p)))
  (mache-knoten (user _p))
  (setq EK-Baum (vererbe (gt)))

```



```

(destroy bew-unknown)
(ass (bew-unknown nil))
(ass (bew-unknown (p . _rest))
  (make-knoten (general-database _p)
    _p
    (bew-unknown _rest))
  (ass (bew-unknown (p . _rest))
    (rule _name _p . _praemissen)
    (make-knoten (_name _p))
    (akt-node-down)
    (bew-unknown _praemissen)
    (akt-node-up)
    (bew-unknown _rest))
  (ass ! (bew-unknown ((false _p) . _rest))
    (rule _name (unknown _p))
    (make-knoten (_name (user-unknown _p)))
    (bew-unknown _rest))
  (ass (bew-unknown (p . _rest))
    (rule _name (unknown _p))
    (make-knoten (_name (user-unknown _p)))
    (bew-unknown _rest))
(destroy mache-factum)
; mache-factum hat 2 Eingaben: ein Praedikat P und eines der Schlüsselworte
; 'j', 'n' oder 'u'. Je nach Schlüsselwort wird P als true,false oder
; unknown in die Datenbasis (im Praedikat 'rule') eingefuegt.
(ass (mache-factum _praedikat j)
  (ass (rule user _praedikat))
  (ass (mache-factum _praedikat n)
    (ass (rule user (false _praedikat))
      nil)
    (ass (mache-factum _praedikat u)
      (ass (rule user (unknown _praedikat))
        nil)
      (destroy call)
      ; call hat als Eingabe eine Liste L von Praedikaten.
      ; call liefert success genau dann wenn alle Praedikate in L mit
      ; den Regeln in der Datenbasis oder im Praedikat 'rule' beweisbar sind.
      (ass (call nil))
      (ass ! (call ((false (false _p)) . _rest))
        (call (_p . _rest)))
      (ass (call (_p . _rest))
        (rule ID _p . _praemissen)
        (call _praemissen)
        (call _rest))
      (ass (call (_p . _rest))
        _p
        (call _rest))
      (destroy mache-knoten)
      ; mache-knoten fuegt im aktuellen Knoten *akt-node* einen neuen Sohn mit
      ; inhalt _inhalt an und loescht diesen wieder, falls ueber mache-knoten
      ; backgetrakt wird.
      (ass (mache-knoten _inhalt)
        (insert-son (akt-node) _inhalt))
        (ass (mache-knoten ID)
          (delete-last-son (akt-node))
          nil)
        (destroy akt-node-up)
        ; akt-node-up setzt den aktuellen Knoten *akt-node* auf seinen Vater
        ; und setzt ihn wieder zurueck ,falls ueber akt-node-up backgetrakt wird.
        (ass (akt-node-up)
          (setq *akt-node* (father (akt-node))))
        (ass (akt-node-down)
          (setq *akt-node* (car (last (sons (akt-node))))))
          nil)
        (defun aktuell() *akt-node*)
        (defun hole-regel (name)
          ; liefert eine Liste mit:
          ; 1 Element der Liste ist die Konklusion der Regel mit Namen 'name'.
          ; i-tes Element der Liste ist die (i-1)-te Praemisse der Regel
          ; mit Namen 'name'.
          (car (prove `(rule ,name . _rest)) 1 _rest)))
        ; --- UNIXPERT.2
        ; --- Filename : CONSULT
        ; Diese Datei enthaelt eine Funktion zum Transformieren von Klauseln
        ; von der NUK-NUK-Darstellung in eine interne Darstellung: 'transform'
        ; Eine Klausel in NUK-NUK-Darstellung hat die Form:
        ; ( <name> <prioritaet> <konklusion> <--- <praemisse>* )
        ; Dabei ist <name> ein beliebiges Lisp-Objekt
        ; <prioritaet> ist eine bel. Zahl
        ; <konklusion> ist ein bel. Praedikat
        ; <praemisse> ist ein bel. Praedikat
        ; Eine Klausel in interner Darstellung hat die Form:

```

```

(ass (mache-knoten _inhalt)
  (insert-son (akt-node) _inhalt))
  (ass (mache-knoten ID)
    (delete-last-son (akt-node))
    nil)
  (destroy akt-node-up)
  ; akt-node-up setzt den aktuellen Knoten *akt-node* auf seinen Vater
  ; und setzt ihn wieder zurueck ,falls ueber akt-node-up backgetrakt wird.
  (ass (akt-node-up)
    (setq *akt-node* (father (akt-node))))
  (ass (akt-node-down)
    (setq *akt-node* (car (last (sons (akt-node))))))
    nil)
  (defun aktuell() *akt-node*)
  (defun hole-regel (name)
    ; liefert eine Liste mit:
    ; 1 Element der Liste ist die Konklusion der Regel mit Namen 'name'.
    ; i-tes Element der Liste ist die (i-1)-te Praemisse der Regel
    ; mit Namen 'name'.
    (car (prove `(rule ,name . _rest)) 1 _rest)))
  ; --- UNIXPERT.2
  ; --- Filename : CONSULT
  ; Diese Datei enthaelt eine Funktion zum Transformieren von Klauseln
  ; von der NUK-NUK-Darstellung in eine interne Darstellung: 'transform'
  ; Eine Klausel in NUK-NUK-Darstellung hat die Form:
  ; ( <name> <prioritaet> <konklusion> <--- <praemisse>* )
  ; Dabei ist <name> ein beliebiges Lisp-Objekt
  ; <prioritaet> ist eine bel. Zahl
  ; <konklusion> ist ein bel. Praedikat
  ; <praemisse> ist ein bel. Praedikat
  ; Eine Klausel in interner Darstellung hat die Form:

```



```

; (rule <name> <konklusion> <praemisse>*)
;
; Dabei ist <name> ein beliebiges Lisp-Objekt
; <konklusion> ist ein bel. Praedikat
; <praemisse> ist ein bel. Praedikat

(destroy append)
(ass (append nil _liste _liste))
(ass (append (_kopf. _rest) _liste (_kopf. _ergebnis))
      (append _rest _liste _ergebnis)))

; 'transform' laed von einer Eingabe-Datei Klauseln in NUK-NUK-Darstellung,
; transformiert diese in die interne Darstellung und speichert sie
; absteigend nach ihrer Prioritaet in dem Praedikat 'rule' in einer
; Ausgabe-Datei.
; In der internen Darstellung wird die Prioritaet weggelassen.
; Beispiel:
; In der Datei XYZ soll folgendes stehen:
;
; (R1 3 (a) <--- (b)(c))
; (R2 2 (b) <--- )
; (R3 4711 (c) <--- )
;
; Nach Ausfuehrung des Aufrufs (transform 'XYZ' 'ABC') steht dann in
; der Datei ABC das folgende:
;
; (destroy rule)
; (ass (rule R3 (c)))
; (ass (rule R1 (a) (b) (c)))
; (ass (rule R2 (b)))

(defun transform(indatei outdatei)
  (let ((inport (infile indatei))
        (outport (outfile outdatei)))
    (print '(index 2 rule) outport)
    (terpri outport)
    (print '(destroy rule) outport)
    (speichere-regeln (sort (einlesen-regeln inport) #'mysort)
                      outport)))

(defun einlesen-regeln (port)
  (let ((hilf (delete '<--- (read port '(eof)))))
    (if (equal hilf '(eof))
        then
        (close port)
        nil
        else
        (cons hilf
              (einlesen-regeln port))))))

(defun speichere-regeln (regelmenge port)
  (if (null regelmenge)

```

```

then
(terpri port)
(close port)
nil
else
(pp-form `(ass ,(append 'rule)
                    (list (nth 0 (car regelmenge))
                          (nth 2 (car regelmenge))
                          (nthcdr 3 (car regelmenge))))
          port)
)

(speichere-regeln (cdr regelmenge) port)))

```

```

(defun mysort (x y)
; mysort dient zum Sortieren der von 'einlesen-regeln' eingelesenen
; Klauseln
(> (cadr x) (cadr y)))

```

```

; Nachfolgend noch einige Funktionen zum Ein- und Ausfuegen von Praedikaten
; in die interne Datenbasis.

```

```

(destroy retract)
; retract erhaelt als Eingabe ein Praedikat P und loescht in der Datenbasis
; die erste Regel, deren Konklusion mit P unifizierbar ist.
(ass (retract _p)
      (rule _name _p. _rest)
      (rex (rule _name _p. _rest))))

```

```

(destroy retract-name)
; retract-name erhaelt als Eingabe einen Namen und loescht in der Datenbasis
; die erste Regel, mit diesem Namen.
(ass (retract-name _name)
      (rule _name _p. _rest)
      (rex (rule _name _p. _rest)))

```

```

(defun assert (name fact &optional (CF 1))
; assert erhaelt als Eingabe einen Namen NAME ein Praedikat FACT und eine
; beliebige Zahl CF. Abhaengig von CF wird FACT als true (CF=1),
; false (CF=-1) oder als unknown in die interne Datenbasis eingefuegt.
(switch-to-modul 'backward)
(n-solutions `(assert-hilf ,name ,fact ,CF) 1)
t)

```

```

(destroy assert-vd)
; dieses Praedikat dient zum Einfuegen von Klauseln durch die Vordiagnose
(ass (assert-vd _name _fact _cf)
      (assert _name _fact _cf))

```

```

(destroy assert-hilf)
(ass ! (assert-hilf _name _fact 1)
      (ass (rule _name _fact)))
(ass ! (assert-hilf _name _fact -1)
      (ass (rule _name (false _fact))))
(ass (assert-hilf _name _fact ID)
      (ass (rule _name (unknown _fact))))

```



```

-----
:--- UNIXPERT.2
:--- Filename : DBASE
-----

(funk-sto-dru-tra-def 0.2
(funktionsstoerung-drucker (traktor-defekt_drucker))
<---
(traktor-defekt_drucker))

(tra-def 0.02
(traktor-defekt_drucker)
<---
(is_drucker (callfun 'targetprinter)))
(not (callfun 'file-in-queue)))
(druckgekommen)
(ask (druckunsauber))
(druckunsauber))

(tra-def 0.01
(false (traktor-defekt_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(false (druckunsauber)))

(tra-def 0.01
(false (traktor-defekt_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(callfun 'file-in-queue)))

(funk-sto-dru-ausgesch 0.6
(funktionsstoerung-drucker (ausgeschaltet_drucker))
<---
(ausgeschaltet_drucker))

(ausge 0.02
(ausgeschaltet_drucker)
<---
(is_drucker (callfun 'targetprinter)))
(not (callfun 'file-in-queue)))
(ask (druckgekommen))
(false (druckgekommen)))

(ausge 0.01
(false (ausgeschaltet_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(callfun 'file-in-queue)))

(ausge 0.01
(false (ausgeschaltet_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(druckgekommen))

(funk-sto-dru-farbba-leer 0.4
(funktionsstoerung-drucker (farbba-leer_drucker))
<---
(farbba-leer_drucker))

(farbba-leer 0.02
(farbba-leer_drucker)
<---
(is_drucker (callfun 'targetprinter)))
(druckgekommen)
(ask (druckunsauber))
(druckunsauber)
(not (callfun 'file-in-queue))))

(farbba-leer 0.01
(false (farbba-leer_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(false (druckunsauber)))

(farbba-leer 0.01
(false (farbba-leer_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(false (druckgekommen)))

(farbba-leer 0.01
(false (farbba-leer_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(callfun 'file-in-queue)))

(funk-sto-dru-pap-alle 0.85
(funktionsstoerung-drucker (papier-alle_drucker))
<---
(papier-alle_drucker))

(pap-alle 0.02
(papier-alle_drucker)
<---
(is_drucker (callfun 'targetprinter)))
(ask (druckgekommen))
(false (druckgekommen))
(not (callfun 'file-in-queue))))

(pap-alle 0.01
(false (papier-alle_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(callfun 'file-in-queue)))

(pap-alle 0.01
(false (papier-alle_drucker))
<---
(is_drucker (callfun 'targetprinter)))
(false (druckgekommen)))

```



```

(funk-sto-dru-pap-fa-ein 0.5
(funktionsstoerung-drucker (papier-falsch-eingelegt _drucker))
<---
(papier-falsch-eingelegt _drucker))
(pap-fa-ein 0.02
(papier-falsch-eingelegt _drucker)
<---
(is_drucker (callfun '(targetprinter)))
(druckgekommen)
(ask (druckunsauber))
(not (callfun '(file-in-queue))))
(pap-fa-ein 0.01
(false (papier-falsch-eingelegt _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(false (druckunsauber)))
(pap-fa-ein 0.01
(false (papier-falsch-eingelegt _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(false (druckgekommen)))
(pap-fa-ein 0.01
(false (papier-falsch-eingelegt _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(false (druckgekommen)))
(funk-sto-dru-OFFLINE 0.9
(funktionsstoerung-drucker (auf-OFFLINE _drucker))
<---
(auf-OFFLINE _drucker))
(OFFLINE 0.02
(auf-OFFLINE _drucker)
<---
(is_drucker (callfun '(targetprinter)))
(ask (druckgekommen))
(false (druckgekommen))
(ask (Online))
(false (Online)))
(OFFLINE 0.01
(false (auf-OFFLINE _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(Online))
(OFFLINE 0.01
(false (auf-OFFLINE _drucker))
<---
(is_drucker (callfun '(targetprinter))))
(funk-sto-dru-pap-stau 0.8
(funktionsstoerung-drucker (papierstau _drucker))
<---
(papierstau _drucker))
(pap-stau 0.02
(papierstau _drucker)
<---
(is_drucker (callfun '(targetprinter)))
(not (callfun '(file-in-queue)))
(druckgekommen)
(ask (druckunsauber))
(druckunsauber))
(pap-stau 0.02
(false (papierstau _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(false (druckunsauber)))
(pap-stau 0.01
(false (papierstau _drucker))
<---
(is_drucker (callfun '(targetprinter)))
(callfun '(file-in-queue)))
(funk-sto-dru-queue 0.1
(funktionsstoerung-drucker (druck-in-queue))
<---
(druck-in-queue))

```



```

(queue 0.02
(druck-in-queue)
<---
(callfun 'file-in-queue)
(callfun '(pq-change))))
(queue 0.01
(false (druck-in-queue))
<---
(not (callfun 'file-in-queue))))
(funk-sto-dru-spo-defekt 0.2
(funktionsstoerung-drucker (spooler-defekt))
<---
(spooler-defekt))
(defekt 0.02
(spooler-defekt)
<---
(druckgekommen)
(ask (drucklueckenhaft))
(drucklueckenhaft))
(defekt 0.02
(spooler-defekt)
<---
(ask (druckgekommen))
(false (druckgekommen))
(ask (Online))
(Online)
(not (callfun '(pq-change))))))
(defekt 0.01
(false (spooler-defekt))
<---
(druckgekommen)
(false (drucklueckenhaft)))
(defekt 0.01
(false (spooler-defekt))
<---
(false (druckgekommen))
(callfun '(pq-change)))
(feh-fa-dru-def 0.6
(fehler-falscher-drucker (druck-nach-default-printer _drucker))
<---
(druck-nach-default-printer _drucker))
(def 0.02
(druck-nach-default-printer _drucker)
<---
(is _drucker (callfun '(targetprinter)))
(not (equal _drucker (callfun '(targetprinter))))))
(que 0.01
(false (druck-nach-default-printer _drucker))
<---
(is _drucker (callfun '(default-printer)))
(not (equal _drucker (callfun '(targetprinter))))))
(feh-fa-dru-login 0.5
(fehler-falscher-drucker (druck-nach-login-printer _drucker))
<---
(druck-nach-login-printer _drucker))
(login 0.02
(druck-nach-login-printer _drucker)
<---
(is _drucker (callfun '(targetprinter)))
(equal _drucker (callfun '(login-printer)))
(ask (wohindruck _printer))
(not (equal _printer _drucker)))
(login 0.01
(false (druck-nach-login-printer _drucker))
<---
(is _drucker (callfun '(login-printer)))
(not (equal _drucker (callfun '(targetprinter))))))
(feh-fa-dru-P 0.1
(fehler-falscher-drucker (druck-nach-P-printer _drucker))
<---
(druck-nach-P-printer _drucker))
(P 0.02
(druck-nach-P-printer _drucker)
<---
(is _drucker (callfun '(targetprinter)))
(equal _drucker (callfun '(P-option)))
(ask (wohindruck _printer))
(not (equal _printer _drucker)))
(P 0.01
(false (druck-nach-P-printer _drucker))
<---
(is _drucker (callfun '(P-option)))
(not (equal _drucker (callfun '(targetprinter))))))
(feh-fi-klein 0.3
(fehler-file (datei-zu-klein _datei))
<---
(datei-zu-klein _datei))
(small 0.02

```



```

(datei-zu-klein _datei)
<---
(is _datei (callfun 'printed-file))
(ask (druckgekommen))
(false (druckgekommen))
(callfun 'exists _datei))
(is _dateisize (callfun 'filesize _datei)))
(< _dateisize (callfun 'buffersize)))

(small 0.01
 (false (datei-zu-klein _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (callfun 'exists _datei))
 (is _dateisize (callfun 'filesize _datei)))
 (not (< _dateisize (callfun 'buffersize))))))

(small 0.01
 (false (datei-zu-klein _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (druckgekommen))

(small 0.01
 (false (datei-zu-klein _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (is _datei (callfun 'printed-file)))
 (not (callfun 'exists _datei))))

(feh-fi-gross 0.3
 (fehler-file (datei-zu-gross _datei))
 <---
 (datei-zu-gross _datei))

(gross 0.02
 (datei-zu-gross _datei)
 <---
 (is _datei (callfun 'printed-file)))
 (ask (druckgekommen))
 (false (druckgekommen))
 (callfun 'exists _datei))
 (is _dateisize (callfun 'filesize _datei)))
 (> _dateisize (callfun 'maxsize))))

(gross 0.01
 (false (datei-zu-gross _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (callfun 'exists _datei))
 (is _dateisize (callfun 'filesize _datei)))
 (not (> _dateisize (callfun 'maxsize))))))

(gross 0.01
 (false (datei-zu-gross _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (druckgekommen))

```

```

(gross 0.01
 (false (datei-zu-gross _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (not (callfun 'exists _datei))))

(feh-fi-ni-dru 0.5
 (fehler-file (datei-nicht-druckbar _datei))
 <---
 (datei-nicht-druckbar _datei))

(ni-dru 0.02
 (datei-nicht-druckbar _datei)
 <---
 (is _datei (callfun 'printed-file)))
 (ask (druckgekommen))
 (false (druckgekommen))
 (callfun 'exists _datei))
 (not (callfun 'printable))))

(ni-dru 0.015
 (datei-nicht-druckbar _datei)
 <---
 (is _datei (callfun 'printed-file)))
 (druckgekommen)
 (ask (drucksinnlos))
 (drucksinnlos))

(ni-dru 0.01
 (false (datei-nicht-druckbar _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (callfun 'exists _datei))
 (callfun 'printable)))

(ni-dru 0.01
 (false (datei-nicht-druckbar _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (druckgekommen))

(ni-dru 0.01
 (false (datei-nicht-druckbar _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (false (drucksinnlos)))

(ni-dru 0.01
 (false (datei-nicht-druckbar _datei))
 <---
 (is _datei (callfun 'printed-file)))
 (false (datei-nicht-druckbar _datei))
 (not (callfun 'exists _datei))))

(feh-fi-CR 0.1
 (fehler-file (CR-vergessen _datei))
 <---
 (CR-vergessen _datei))

```



```

(false (datei-existiert-nicht _datei))
<---
(is _datei (callfun 'printed-file)))
(druckgekommen))
(feh-dru-bef-key 0.9
(fehler-druckbefehl (falsches-keyword))
<---
(falsches-keyword))
(key 0.02
(falsches-keyword)
<---
(is _keyword (car (callfun 'last-lpr))))
(not (equal _keyword 'lpr)))
(key 0.01
(false (falsches-keyword))
<---
(is _keyword (car (callfun 'last-lpr))))
(equal _keyword 'lpr))
(feh-dr-bef-nofile 0.6
(fehler-druckbefehl (dateiname-fehlt))
<---
(dateiname-fehlt))
(nofile 0.02
(dateiname-fehlt)
<---
(equal (callfun 'printed-file) nil))
(nofile 0.01
(false (dateiname-fehlt))
<---
(not (equal (callfun 'printed-file) nil))))

```

```

-----
:--- UNIXPERT.2
:--- Filename : EK
:---

```

```

: *****
: *
: * ERKLÄRUNGSKOMPONENTE *
: *
: *****

```

```

: Diese Datei enthaelt die Erklarungskomponente. Sie wurde vollstaendig in
: LISP geschrieben und gliedert sich in :
:
: (1) Toplevel-Funktion "EK"
: (2) Erklarungsfunktionen zur Vordiagnose ("ende-vordiagnose")
: (3) Erklarungsfunktionen zur Hauptdiagnose
:

```

```

(CR 0.02
(CR-vergessen _datei)
<---
(is _datei (callfun 'printed-file)))
(ask (druckgekommen))
(druckgekommen)
(callfun '(CRvergessen _datei)))
(CR 0.012
(CR-vergessen _datei)
<---
(is _datei (callfun 'printed-file)))
(druckgekommen)
(ask (drucklueckenhaft))
(drucklueckenhaft))
(CR 0.01
(false (CR-vergessen _datei))
<---
(is _datei (callfun 'printed-file)))
(not (callfun '(CRvergessen _datei))))
(CR 0.01
(false (CR-vergessen _datei))
<---
(is _datei (callfun 'printed-file)))
(false (druckgekommen)))
(CR 0.01
(false (CR-vergessen _datei))
<---
(is _datei (callfun 'printed-file)))
(false (drucklueckenhaft)))

```

```

(feh-dru-bef-ex-ni 0.9
(fehler-druckbefehl (datei-existiert-nicht _datei))
<---
(datei-existiert-nicht _datei))

```

```

(ex-ni 0.02
(datei-existiert-nicht _datei)
<---
(is _datei (callfun 'printed-file)))
(ask (druckgekommen))
(false (druckgekommen))
(not (callfun '(exists _datei))))

```

```

(ex-ni 0.01
(false (datei-existiert-nicht _datei))
<---
(is _datei (callfun 'printed-file)))
(callfun '(exists _datei)))
(ex-ni 0.01

```



```
; ("In-Diagnose" "Diagnose-gestellt" "wie-ausgeschlossen")
```

```
; *****  
; * * * * *  
; * * * * *  
; * * * * *  
; * * * * *  
; * * * * *  
; * * * * *
```

```
; Die Toplevel-Funktion "EK" ist der einzige Ansprechpartner der Kontroll-  
; funktionen von Vor- und Hauptdiagnose. Abhängig vom Wert des Schlüssels  
; (= erstes Element der Eingabeliste ea), ruft sie die zuständige Unter-  
; funktion (z.B. "ende-vordiagnose" bei Schlüssel "VDE") auf.  
; EK-Antwort macht aus dem Wert dieser Unterfunktion einen passenden  
; Ausgabesatz.
```

```
(defun EK (ea)  
  (caseq (nth 0 ea)  
    (VDE (EK-Antwort (ende-vordiagnose ea)))  
    (In-Diag (print (In-Diagnose (nth 1 ea))))  
    (Diag-gestellt (Diagnose-gestellt (nth 1 ea)))  
    ('wie-ausgeschlossen (wie-ausgeschlossen (nth 1 ea))))))
```

```
; *****  
; * * * * *  
; * * * * *  
; * * * * *  
; * * * * *
```

```
; "ende-vordiagnose" ist die Toplevel-Funktion fuer die Vordiagnose.  
; Abhängig vom Wert des zweiten Schlüssels der Eingabeliste ruft sie die  
; entsprechende Unterfunktion auf. Dies kann sein:
```

- (1) "diagnobereiche": liefert Information ueber die Menge der Diagnose-
bereiche (etwa nur die Diagnosebereiche mit posi-
tiven Certainty-faktor)
- (2) "ursache-diag": liefert Erklarung, warum bestimmter Diagnosebereich
gewaehlt wurde.
- (3) "ursache-konsolelem": liefert Erklarung, warum ein bestimmtes Kondi-
tion-Element erschlossen wurde.

```
; Dabei werden vom Forward-Regelinterpreter folgende globale Variable zur  
; Verbuegung gestellt:
```

- (1) "d-bereiche": enthaelt eine Liste der Form {(Dber.1 CF1)(Dber.2 CF2)..}
die nach CF's absteigend geordnet ist.
- (2) "uxp-stack": enthaelt eine Liste der gefeuerten Regeln, wobei eine
Regel die Form {Regeln ((Kondel.1 CF1)..) ((Hyp1 CF1)..)}
hat. Zuerst steht die zuletzt gefeuerte Regel, dann folgt
die davor gefeuerte Regel usw.

```
(defun ende-vordiagnose (ea)  
  (let ((schluessel (nth 1 ea))  
        (wert (nth 2 ea)))  
    (caseq schluessel  
      ('Diagnobereiche  
       (cons wert (diagbereiche d-bereiche wert)))  
      ('Diagnobereich  
       (cons DB (ursache-diag uxp-stack wert nil)))  
      ('Konsolelem  
       (list KE (ursache-konsolelem wert)))  
      ('(princ "Fehler Ein/Ausgabe"))))
```

```
(defun diagbereiche (dbl wert)
```

```
; *** Eingabe : dbl Liste von Paaren (Diagber. Cert.fakt.) absteigend  
; nach CF geordnet. (erhaelt seinen Wert von "d-bereiche")  
; wert Schluesel, der angibt, welche der Bereiche aus dbl  
; geliefert werden sollen.  
; *** Ausgabe : abhaengig von "wert" die positiven (nach CF), negativen,  
; ersten n, oder alle Diagnosebereiche aus "dbl".
```

```
(caseq wert  
  ('gesamt dbl)  
  ('+ (poscf dbl))  
  ('- (negcf dbl))  
  ('(firstn dbl wert))  
  )
```

```
(defun firstn (l n)
```

```
; *** liefert die ersten "n" Elemente der Liste "l".
```

```
(if (= n 0)  
  nil  
  (cons  
    (car l)  
    (firstn (cdr l) (- n 1))  
  )  
)
```

```
(defun poscf (dblste)
```

```
; *** liefert die Bereiche mit positivem CF aus "dblste".  
(if (or (null dblste) (<= (car (last (cdr dblste))) 0))  
  nil  
  (cons (car dblste) (poscf (cdr dblste))))  
)
```



```

(t (simple-check schluesel (cdr hypliste)))
)
)

(defun ursache-kondelem (ke)
; *** Eingabe : ke Konditionelement, nach dessen Ausloeser gesucht wird.
; *** Ausgabe : zuletzt angewandte Regel (von "element" aus gesehen), bei
; der "ke" in der Hypothesenmenge vorkommt. Ihr Kondition-
; Teil wird noch um "ke" mit dem bei der Regelanwendung
; aktuellen CF erweitert.
; Falls es keine solche Regel gibt, wird entsprechend
; 'systemfrage oder 'sie-sagten-so geliefert.
; *** Vorgehensweise : Normalerweise braucht nur der von "element" einge-
; schraenkte Suchraum bearbeitet zu werden, falls
; der Benutzer jedoch ein neues Konditionelement "ke"
; angibt, wird im ganzen Diagnosebereich ("bereich")
; gesucht. Falls auch dann kein Erfolg vorliegt, muss
; es sich um "Vorwissen" handeln.
(or (suche element ke)
    (suche bereich ke)
    'Vorwissen))

(defun suche (liste ke)
; *** durchsucht die Liste "liste" sequentiell mit Hilfe der Funktion
; "check" nach "ke" und liefert beim ersten Erfolg die durch "letztes
; -vorkommen" angereicherte Regel (siehe "ursache-kondelem"), oder
; : eine andere Begrueendung (z.B. 'systemfrage).
(if (null liste)
    nil
    (caseq (check ke (nth 2 (car liste)))
           (regel (setf element (cdr liste)))
           (let ((regel (copy (car liste))))
             (setf (nth 1 regel)
                   (append2 (nth 1 regel)
                           (letztes-vorkommen
                            ke
                            (cdr liste))))
             (system-frage 'system-frage)
             (frage 'sie-sagten-so)
             (t (suche (cdr liste) ke))
             )
           )
    )
)

(defun append2 (l e)
  (if (null e)
      l

```

```

(defun negcf (dbliste)
; *** liefert die Bereiche mit negativem CF aus "dbliste".
(cond ((null dbliste) nil)
      ((minusp (car (last (car dbliste)))) dbliste)
      (t (negcf (cdr dbliste))))
)

(defun ursache-diag (grl db modus)
; *** Eingabe : grl Liste der gefueuerten Regeln (wird beim ersten Aufruf
; mit "uxp-stack" aufgerufen).
; db Name des Diagnosebereiches, dessen "Ausloeser" gesucht
; werden.
; modus gibt an, ob im "uxp-stack" die Hypothese "db" schon
; einmal gefunden wurde.
; *** Zur Einschraenkung der Suche im "uxp-stack" gibt es globale Zeiger
; "bereich" und "element", die auf den Teil im "uxp-stack" zeigen, der
; den gerade aktuellen Diagnosebereich bzw. das gerade aktuelle Kondition-
; enthaelt.
; *** Ausgabe : Liste der Konditionelemente (jeweils mit CF) aus allen gefue-
; erten Regeln, bei denen "db" in der Hypothesenmenge vorkam.
; (dies kann mit der Funktion "simple-check" getestet werden)
(cond ((null grl) nil)
      ((and (simple-check db (nth 2 (car grl)))
            (modus)
            (append (nth 1 (car grl))
                    (ursache-diag (cdr grl) db modus)))
       ((simple-check db (nth 2 (car grl)))
        (setf bereich (cdr grl))
        (setf element (cdr grl))
        (append (nth 1 (car grl))
                (ursache-diag (cdr grl) db 'gef))
        (t (ursache-diag (cdr grl) db modus)))
       )
)

(defun simple-check (schluesel hypliste)
; *** liefert 'gefunden ,falls "schluesel" irgendwo in "hypliste" vorkommt.
(cond ((null hypliste) nil)
      ((or (equal (caar hypliste) schluesel)
           (and (listp (caar hypliste))
                (member-all schluesel (caar hypliste))))
       'gefunden)
)

```



```
(append1 l e)))
```

```
(defun letztes-vorkommen (schluessel regelliste)
```

```
; *** liefert "schluessel" mit juengstem CF in "regelliste".
```

```
(if (null regelliste)
```

```
nil
```

```
(or (vorhanden schluessel (nth 2 (car regelliste)))
```

```
(letztes-vorkommen schluessel (cdr regelliste))))))
```

```
(defun vorhanden (schluessel hl)
```

```
; *** durchsucht Hypothesenliste "hl" nach "schluessel" und liefert
```

```
; ggf. "schluessel" mit CF.
```

```
(if (null hl)
```

```
nil
```

```
(let ((hyp (caar hl)))
```

```
(if (or (schluessel hyp)
```

```
(and (listp hyp)
```

```
(member-all schluessel hyp))))
```

```
(list schluessel
```

```
(nth 1 (car hl)))
```

```
(vorhanden schluessel (cdr hl))))))
```

```
(defun check (schluessel hypliste)
```

```
; *** durchsucht sequentiell "hypliste" nach "schluessel" ;
```

```
; Falls eine Hypothese in "hypliste" gleich "schluessel" ist muss
```

```
; es sich um eine "normale Regelanwendung" handeln (d.h. keine
```

```
; Metaregel), falls das Prädikat "systemfrage" "schluessel" als
```

```
; Argument hat, muss eine Metaregel die Nachfrage nach "schluessel"
```

```
; veranlasst haben analog bei Benutzernachfrage ("frage"). Falls
```

```
; "schluessel" ueberhaupt nicht in "hypliste" vorkommt wird nil
```

```
; geliefert.
```

```
(if (null hypliste)
```

```
nil
```

```
(let ((hilf (caar hypliste)))
```

```
(cond
```

```
((equal hilf schluessel) regel)
```

```
((and (listp hilf)
```

```
(member-all schluessel hilf)
```

```
(equal (car hilf) 'system-frage))
```

```
'system-frage)
```

```
((and (listp hilf)
```

```
(member-all schluessel hilf)
```

```
(equal (car hilf) 'frage))
```

```
'frage)
```

```
(t (check schluessel (cdr hypliste))))
```

```
)
```

```
)
```

```
)
```

```
; *****
```

```
; * *
```

```
; * H A U P T D I A G N O S E *
```

```
; * *
```

```
; *****
```

```
; In der Hauptdiagnose gibt es drei verschiedene Modi:
```

```
; (1) WARUM-Modus ,d.h. es ist noch keine Diagnose gestellt ,an den Benutzer
```

```
; wird vom System eine Frage gestellt ,von der dieser wissen
```

```
; moechte , warum sie gestellt wurde. Anschliessend kann er
```

```
; dann mit WIE und WARUM weiterfragen.
```

```
; Die dafuer zustaendige Toplevel-Funktion ist
```

```
; " In-Diagnose " .
```

```
; (2) WIE-Diagnose-Modus ,d.h. die Diagnose ist gerade gestellt worden und
```

```
; der Benutzer moechte gem wissen WIE sie zustande
```

```
; kam. Anschliessend kann er dann mit WIE nach den
```

```
; einzelnen Praemissen weiterfragen.
```

```
; Die dafuer zustaendige Toplevel-Funktion ist
```

```
; " Diagnose-gestellt " .
```

```
; (3) WIE-AUSGESCHLOSSEN-Modus ,d.h. nach gestellter Diagnose kann der Be-
```

```
; nutzer nachfragen. WIE einzelne Diagnosen
```

```
; AUSGESCHLOSSEN wurden und dabei mit WIE
```

```
; ggf. weiterfragen .
```

```
; Die zustaendige Toplevel-Funktion dafuer ist
```

```
; " wie ausgeschlossen " .
```

```
; Alle drei Toplevel-Funktionen benutzen dieselbe "wie"-Funktion im WARUM-Modus
```

```
; gibt es eine entsprechende "warum"-Funktion.
```

```
; Um WIE- und WARUM-Fragen beantworten zu koennen wird vom Backward-Interpre-
```

```
; folgende globale Variable zur Verfuegung gestellt :
```

```
; (1) EK-Baum enthaelt den Erklarungsbaum fuer die gestellte Diagnose ; muss
```

```
; bei WARUM-NICHT (d.h. wie ausgeschlossen ?) Fragen umgesetzt
```

```
; werden . Die Knoten des Baumes sind mit (Regelname Praedikat)
```

```
; gelabelt und enthalten als Soehne die Praemissen der Regel
```

```
; vollstaendig , falls Diagnose gestellt , ansonsten nur alle
```

```
; bisher bewiesenen Praemissen . Die Wurzel enthaelt die aktuelle
```

```
; Fehlerklasse und den benutzten Waschgang .
```

```
; Es stehen die ueblichen Bearbeitungsfunktionen zur Verfuegung
```

```
; ("Father" "nth-son" usw.) .
```

```
; Ausserdem gibt es noch zwei globale Zeiger :
```

```
; (2) ek-warum zeigt zunaechst auf den Vater der vom System an den Benutzer
```

```
; gestellten Frage und wird nach jeder WARUM-Frage um eine Stufe
```



```

; im Baum hochgesetzt bzw. bei einer WIE-Frage auf den Vater von
; "ek-wie".
;
; (3) ek-wie zeigt zunächst auf die Wurzel (d.h. den Vater der gestellten
; Diagnose. Bei einer WIE-Frage wird er dann auf das gefragte
; Element (das ein Sohn des bisherigen "ek-wie" sein sollte) um-
; gesetzt.
;
; Der Benutzer kann anhand eines Menues also nach dem Vater des gerade aktu-
; ellen Elements fragen (WARUM ? Antwort = "ek-warum"), oder nach seinen Soehnen
; (WIE ? Antwort = Soehne von "ek-wie"). Ausserdem sind auch WIE-Fragen nach
; beliebigen Prädikaten (im Baum) möglich und es wird die Möglichkeit gege-
; ben sich im Baum auf- und abwaerts zu bewegen.

```

```

; *****
; *
; * TOPELVEL-FUNKTIONEN DER HAUPTDIAGNOSE *
; *
; *****

```

```

; ***** WARUM-MODUS *****

```

```

(defun In-Diagnose (selektor)

```

```

; *** Eingabe : selektor wird beim Aufruf durch die Backward-Kontrollfunktion
; mit 'neues-warum' belegt. Anschliessend fuehrt "In-
; Diagnose" den Dialog mit dem Benutzer selbst (durch
; Menue) bis dieser den WARUM-Modus verlassen will (mit
; 'q fuer "Quit".

```

```

; *** Ausgabe : Antwort auf die erste WARUM-Frage (durch Aufrufen der "warum"-
; Funktion wobei "ek-warum" auf den Vater des Frageknotens im
; "EK-Baum" gesetzt wird: "finde-frage") und anschliessend den
; Benutzerantworten auf das Menue entsprechenden Reaktionen :

```

```

; q Verlassen der Erklarungskomponente
; z zurueck zur letzten Frage (d.h. eine Stufe runter im "EK-
; Baum", wobei "ek-warum" und "ek-wie" entsprechend umgesetzt
; werden muessen (anschliessend wieder Menue-Ausgabe)
; 0 spaetere WARUM-Frage, d.h. die "warum"-Funktion wird aufge-
; rufen
; <Praed> WIE wurde <Praed> bewiesen, d.h. Aufruf der "wie"-
; Funktion
; n WIE wurde n-te Praemisse (des letzten "wie"- oder
; "warum"-Aufrufs) bewiesen. ("wie" und "warum" liefern
; dem Benutzer jeweils eine nummerierte Liste der
; aktuellen Praemissen ; die n-te Praemisse entspricht
; gerade dem n-ten Sohn von "ek-wie")

```

```

; Bemerkung: "wie" und "warum" erzeugen, nachdem sie eine Antwort
; geben haben, natuerlich wieder das o.g. Menue.

```

```

(caseq selektor
  ('neues-warum (setq ek-warum (father (finde-frage EK-Baum))))

```

```

  (warum))
  ('z (setq ek-warum ek-wie)
  (warum))
  ('t (wie selektor)))
  ('let ((eingabe (read)))
    (if (eq eingabe 'q)
        t
        (In-Diagnose eingabe))))

```

```

; ***** WIE-Diagnose-Modus *****

```

```

(defun Diagnose-gestellt (selektor)

```

```

; *** Eingabe : selektor bei Aufruf von Backward-Kontrollfunktion 'neues-wie
; sonst wieder entsprechend Menue (s.o.)
;

```

```

; *** Ausgabe : Beim ersten Aufruf mit 'neues-wie wird "wie" aufgerufen wobei
; "ek-wie" initial auf die Wurzel des Erklarungsbaums gesetzt
; wird (Vater von der gestellten Diagnose) ; wie bei "In-Diag-
; nose" wird der Dialog dann von "Diagnose-gestellt" uebernommen
; wobei dasselbe Menue verwendet wird. Ein Unterschied besteht
; lediglich in der Bedeutung von 'z fuer "zurueck zur letzten
; Frage" ; da bei WIE-Fragen im Baum runter gegangen wird
; bedeutet "zurueck" nun "eine Stufe hoeher im EK-Baum".

```

```

(caseq selektor
  ('neues-wie (setq ek-wie EK-Baum)
  (wie 1))
  ('t (wie selektor)))
  ('let ((eingabe (read)))
    (caseq eingabe
      ('z (setq ek-wie (father ek-wie))
        (Diagnose-gestellt 'neues-wie))
      ('q t)
      ('t (Diagnose-gestellt eingabe))))))

```

```

; ***** WIE-AUSGESCHLOSSEN-Modus *****

```

```

(defun wie-ausgeschlossen (nr)

```

```

; *** Eingabe : nr Position des ausgeschlossenen Fehlers in der Liste der
; ausgeschlossenen Fehler, die durch die Funktion (ausge-
; schlossene-fehler" (siehe Backward-Komponente) zur Ver-
; fuegung gestellt wird.

```

```

; *** Ausgabe : Der "EK-Baum" wird umgesetzt auf den Baum der den ausge-
; schlossenen Fehler enthaelt; anschliessend kann dieser dann
; wie im WIE-Diagnose-Modus abgearbeitet werden.

```

```

(setq EK-Baum (erzeuge-baum (list 'false
  (nth (1- nr)

```



```
(Diagnose-gestellt 'neues-wie')
  (ausgeschlossene-fehler))))))
```

```
*****
*
*   FRAGE- UND HILFS- *
*   *
*   FUNKTIONEN *
*   *
*****
```

```
(defun warum ()
```

```
;; ** Vom Zeiger "ek-warum" ausgehend wird festgestellt welche Soehne
;; schon bewiesen wurden (alle bis zur Frage) um den Inhalt von "ek-
;; warum" zu beweisen und welche noch bewiesen werden muessen (dies
;; wird durch Betrachien der angewandten Regel ("hole-regel"), die
;; auch im Label von "ek-warum" steht, erreicht) .
;; Das Ganze wird zusammen mit dem Waschung von "antwort-warum" ver-
;; arbeitet und ausgegeben.
;; Anschliessend werden "ek-wie" und "ek-warum" eine Stufe im Baum erhoeht
;; um bei der naechsten Frage wieder auf das aktuellen Element zu zeigen.
;; Falls "ek-warum" auf die Wurzel von "EK-Baum" zeigt wird "weil ich
;; dann fertig bin" geliefert.
```

```
(if (father ek-warum)
  (let ((bewiesen (bis-zur-frage (sons ek-warum)))
        (regel (cdr (hole-regel (car (inhalt ek-warum))))))
    (antwort-warum (list (waschung ek-warum)
                        (nth 1 (inhalt ek-warum))
                        bewiesen)
      (if (= (length regel) (1+ (length bewiesen)))
          (list p)
          (nthcdr (length bewiesen) regel))))))
  (setq ek-wie ek-warum)
  (setq ek-warum (father ek-warum)))
  (progn (terpri)
         (princ "***** Weil ich dann fertig bin *****")
         (menuue)
         )
  )
```

```
(defun finde-frage (knoten)
```

```
;; ** geht im Baum ("knoten") soweit wie moeglich nach rechts unten
;; und liefert dann diesen Knoten
(let ((anzahl-soehne (length (sons knoten))))
  (if (zerop anzahl-soehne)
      knoten
      (finde-frage (nth-son anzahl-soehne knoten))))))
```

```
(defun waschung (knoten)
```

```
;; ** liefert den Waschung, der in der Wurzel stehen muss
;;
(car (inhalt (root knoten))))
```

```
(defun bis-zur-frage (knoten-liste)
```

```
;; ** liefert von einer "knoten-liste" alle Knoten bis zur ersten Frage
;; (da bei einem Baum im WARUM-Modus die Frage aber zuletzt gestellt
;; wurde, muss sie also im letzten Knoten sein)
```

```
(if (null (cdr knoten-liste))
```

```
nil
  (cons (nth 1 (inhalt (car knoten-liste)))
        (bis-zur-frage (cdr knoten-liste))))))
```

```
(defun wie (p)
```

```
;; ** Eingabe : p Entweder Zahl n ("wie wurde n-ter Sohn von "ek-wie"
;; bewiesen ?") oder Praedikat (in diesem Fall muss es
;; noch mit "suche-back" im "EK-Baum" gefunden werden
;;
;; ** Ausgabe : zunaechst wird "ek-wie" auf das entsprechende Praedikat p
;; gesetzt ; die Soehne von "ek-wie" sind dann gerade die
;; Praemissen die zu dem Praedikat gefuehrt haben. "antwort-
;; wie" bereitet das Ganze dann fuer die Terminalausgabe auf.
;; Falls keine Soehne existieren wird durch Untersuchen des
;; Inhalts von "ek-wie" festgestellt, wie das Praedikat er-
;; schlossen wurde (z.B. "Vordiagnose", falls Regel 'vordiag-
;; nose ).
```

```
(setq ek-wie (if (and (numberp p) (<= p (length (sons ek-wie))))
  (nth-son p ek-wie)
  (or (suche-back p EK-Baum) ek-wie)))
```

```
(setq ek-warum (father ek-wie))
```

```
(cond
  ((null ek-wie)
   (terpri))
  (terpri)
  (princ "***** Praedikat nicht im Baum *****")
  (menuue))
  ((sons ek-wie)
   (antwort-wie (list (waschung ek-wie)
                     (nth 1 (inhalt ek-wie))
                     (mapcar #'(lambda (x) (nth 1 (inhalt x)))
                             (sons ek-wie))))
    ((eq (car (inhalt ek-wie)) 'user)
     (terpri))
    (princ "***** Sie sagten so *****")
    (menuue))
    ((eq (car (inhalt ek-wie)) 'user-unknown)
     (terpri))
```



```

(terpri)
(terpri)
(let ((zaehler 0))
  (mapc #' (lambda (p) (princ " ("
    (princ (setq zaehler (1+ zaehler)))
    (princ ")")
    (princ (praed-to-string p))
    (terpri))
    (nth 2 antwort)))
  (princ "
Ich muss noch zeigen: ")
  (terpri)
  (terpri)
  (let ((zaehler 0))
    (mapc #' (lambda (p) (princ " ("
      (princ (setq zaehler (1+ zaehler)))
      (princ ")")
      (princ (praed-to-string p))
      (terpri))
      (nth 3 antwort)))
    (princ "

```

Tippen Sie : 0 fuer 'Warum ?'
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit
> ")

```

(defun menue ()
; *** druckt das Menue auf dem Bildschirm aus
(princ "

```

Tippen Sie : 0 fuer 'warum ?' (nur sinnvoll waehrend Diagnose !!)
n fuer 'Wie wurde n-te Praemisse bewiesen ?'
<Praed> fuer 'Wie wurde <Praed> bewiesen ?'
z fuer 'zurueck zur letzten Frage'
q fuer quit
> ")

```

:--- UNIXPERT.2
:--- Filename : EK-AUSGABE
:---

```

```

(defun positiv (f)
  (cond
    ((null f) (princ "keine weiteren positiven Bereiche."))
    ((greaterp (car(cdr(car f))) 0) (print (car(car f)))
     (terpri)
     (positiv (cdr f))))

```

```

(t (positiv (cdr f))))))
(defun negativ (f)
  (cond
    ((null f) (princ "keine weiteren negativen Bereiche."))
    ((lessp (car(cdr(car f))) 0) (print (car(car f)))
     (terpri)
     (negativ (cdr f))))
    (t (negativ (cdr f))))))
(defun gesamt (f n)
  (cond
    ((null f) (cond
      ((equal n 'b)(princ "keine weiteren Bereiche")
       ((equal n 'p)(princ "keine weiteren Praemissen")
        ((equal n 'h)(princ "keine weiteren Hypothesen")
         (terpri)))
      (t (print (car(car f)))
         (princ " ")
         (print (car(cdr(car f))))
         (terpri)
         (gesamt (cdr f) n))))))
    (defun ausgabeek (f num)
      (cond
        ((zerop num) (princ "Das waren die gewünschten Bereiche."))
        (t (print (car(car f)))
           (terpri)
           (ausgabeek (cdr f)(sub1 num))))))

```

```

(defun bereiche (f)
  (terpri)
  (cond
    ((null f) (terpri)
     (princ "Es gibt keine weiteren Praemissen")
     (terpri))
    (t
     (print (car(car f)))
     (princ " ")
     (print (car(cdr(car f))))
     (terpri)
     (bereiche (cdr f))))))

```

```

(defun kondelem (f)
  (caseq f
    (sie-sagen-so (princ "Sie sagten so.")
     (terpri))
    (system-frage (princ "Resultat einer Systemfrage")
     (terpri))
    (Vorwissen (princ "Vorwissen")
     (terpri))
    (nil (princ "Vorwissen")
     (terpri))
    (t (princ "Regel Nr. ") (print (car f))
     (princ " hat gefeuert")
     (setq f (cdr f))))

```



```

(terpri)
(princ "Es gelten folgende Praemissen: ")
(terpri)
(gesamt (car f) p)
(terpri)
(terpri)
(princ "Aufgrund dieser Praemissen wurden folgende ")
(princ "Hypothesen erschlossen:")
(terpri)
(gesamt (car (cdr f)) 'h)
(terpri))

(defun EK-Antwort (e)
(terpri)
(caseq (car e)
(+ (princ "Die positiven Bereiche sind: ")
(terpri)
(positiv (cdr e)))
(- (princ "Die negativen Bereiche sind: ")
(terpri)
(negativ (cdr e)))
(gesamt (princ "Die Bereiche sind: ")
(terpri)
(gesamt (cdr e) 'b)))
(DB (princ "Diagnosebereich wurde gewaehlt, ")
(princ "weil folgende Praemissen gelten: ")
(terpri)
(bereiche (cdr e))
(terpri)
(KE (kondelem (car (cdr e)))
(terpri))
(4 (princ "Die ersten ") (print (car e)) (princ " Bereiche sind: ")
(terpri)
(ausgabeek (cdr e) (car e)))
(3 (princ "Die ersten ") (print (car e)) (princ " Bereiche sind: ")
(terpri)
(ausgabeek (cdr e) (car e)))
(2 (princ "Die ersten ") (print (car e)) (princ " Bereiche sind: ")
(terpri)
(ausgabeek (cdr e) (car e)))
(princ "Die ersten ") (print (car e)) (princ " Bereiche sind: ")
(terpri)
(ausgabeek (cdr e) (car e))))
-----
:--- UNIXPERT.2
:--- Filename : ENDDIAGNOSE
:---
: Diese Datei enthaelt die Klartexte, die ausgegeben werden
: wenn die Enddiagnose gestellt ist.
(destroy enddiagnose-fehler )

(ass ! (enddiagnose-fehler (traktor-defekt _d ) _erg)
(is _erg (list (concat (Der Traktor von Drucker l _d l ist defekt. l )))))

```

```

(ass ! (enddiagnose-fehler (farbband-leer _d ) _erg)
(is _erg (list (concat (Das Farbband von Drucker l _d l ist leer. l )))))

(ass ! (enddiagnose-fehler (papier-alle _d ) _erg)
(is _erg (list (concat (Das Papier von Drucker l _d l ist alle. l )))))

(ass ! (enddiagnose-fehler (papier-falsch-eingelegt _d ) _erg)
(is _erg (list (concat (Das Papier von Drucker l _d l ist falsch eingelegt. l )))))

(ass ! (enddiagnose-fehler (auf-OFFLINE _d ) _erg)
(is _erg (list (concat (Der Drucker l _d l steht auf OFFLINE. l )))))

(ass ! (enddiagnose-fehler (stecker-draussen _d ) _erg)
(is _erg (list (concat (Der Stecker von Drucker l _d l ist draussen. l )))))

(ass ! (enddiagnose-fehler (ausgeschaltet _d ) _erg)
(is _erg (list (concat (Der Drucker l _d l ist ausgeschaltet. l )))))

(ass ! (enddiagnose-fehler (papierstau _d ) _erg)
(is _erg (list (concat (Am Drucker l _d l ist) kein Papierstau aufgetreten. l ))))

(ass ! (enddiagnose-fehler (druck-nach-default-printer _d ) _erg)
(is _erg (list (concat (Der Druck ist zum Defaultprinter gegangen))))

(ass ! (enddiagnose-fehler (druck-nach-login-printer _d ) _erg)
(is _erg (list (concat (Der Druck ging zu dem im login-filial
|spezifizierten Drucker. l )))))

(ass ! (enddiagnose-fehler (druck-nach-P-printer _d ) _erg)
(is _erg (list (concat (Der Druck ging zu dem durch die
|P-Option spezifizierten Drucker. l )))))

(ass ! (enddiagnose-fehler (druck-in-queue ) _erg)
(is _erg (list (concat (Der Druck ist noch in der Warteschlange. l )))))

(ass ! (enddiagnose-fehler (datei-existiert-nicht _d ) _erg)
(is _erg (list (concat (Die Datei l _d l existiert nicht. l )))))

(ass ! (enddiagnose-fehler (datei-zu-klein _d ) _erg)
(is _erg (list (concat (Die Datei l _d l ist zu klein. l )
"Daher passte sie komplett in den Buffer."
"Wahrscheinlich hat jemand den Drucker aus-"
"geschaltet als die Datei schon im Buffer war. l ))))

(ass ! (enddiagnose-fehler (datei-zu-gross _d ) _erg)
(is _erg (list (concat (Die Datei l _d l ist zu gross. l )))))

(ass ! (enddiagnose-fehler (datei-nicht-druckbar _d ) _erg)
(is _erg (list (concat (Die Datei l _d l ist nicht druckbar. l )))))

(ass ! (enddiagnose-fehler (CR-vergessen _d ) _erg)
(is _erg (list (concat
|Sie haben in der Datei l _d
| in der letzten Zeile das CR vergessen. l ))))

(ass ! (enddiagnose-fehler (spooler-defekt ) _erg)
(is _erg (list (concat (Der Spooler ist defekt. l )))))

```



```
(setq letzter-druckbefehl (lookup-for-lpr (make-list '~/'history.file)))
else
letzter-druckbefehl]

(defun lookup-for-lpr (history-list)
(cond ((null history-list)
      nil)
      ((equal 'lpr (norm-lpr(klein(car history-list))))
       (lpr-command history-list))
      (t
       (lookup-for-lpr (cdr history-list))
       )
))
```

```
(defun norm-lpr (command)
(if (my-member command '(lpr print prf plot p d druck pr)
              'lpr
          command)
    (defun lpr-command (history-list)
      (cond
        ((null history-list) nil)
        ((numberp (cadr history-list))
         (list (car history-list)))
        (t (cons (car history-list)
                  (lpr-command (cdr history-list))
                  )))
      ))
))
```

```
(defun printed-file ()
(let ((lastlpr (last-lpr)))
  (if (equal 'lpr (car lastlpr))
      (fname (cdr lastlpr))
      nil)
))
```

```
(defun fname (lpr-list)
(if (nequal (car (explode (car lpr-list))) '-')
    then
    (car lpr-list)
    else
    (fname (cdr lpr-list))
))
```

```
(defun printable ()
(let ((fname (printed-file)))
  (if (exists fname)
      then
      (equal
        (boole 1
          (+ 1 8 64)
          (filestat:mode (filestat fname))))
      0)
))
```

```
(ass ! (endiagnose-fehler (falsches-keyword ) _erg)
      (is_erg (list
              !Sie haben im Druckbefehl ein | falsches Keyword angegeben.) )))
(ass ! (endiagnose-fehler (dateiname-fehlt ) _erg)
      (is_erg (list
              !Sie haben im Druckbefehl vergessen | den Dateinamen anzugeben.) )))
:-----
:--- UNIXPERT 2
:--- Filename : EXECFILE
:-----
```

```
: LISP - Funktionen fuer die System - Ein/Ausgabe
:-----
:*****
:*****
:*****
:*****
:*****
: Hier muss die Systemabhaengige Liste der vorhandenen Drucker
: angegeben werden.
: Diese Liste hat folgendes Format :
: ((<druckoption> <normbezeichnung> <Synonyme|> ... <Synonym_n>))
: Der erste in diese Liste eingetragene Drucker MUSS der Default-Printer
: sein
:-----
```

```
(setq Available-Printer
  '(
    (lp B48 Bau48 48 bau34)
    (lpb14 B14 Bau14 14 bau14)
    (lp48211 B48211 Bau48-Raum211 B48211 48-211 bau48211 211)
    (lpb34 B34 Bau34 34 bau34)
    (lpirz BIRKZ IRZ irz Rechenzentrum)
  ))
```

```
:*****
: Funktionen zur Informationsgewinnung ueber den letzten Druckbefehl:
:*****
:*****
:*****
:*****
(setq letzter-druckbefehl nil)
(defun last-lpr ()
  (if (null letzter-druckbefehl)
      then
      0
      ))
```



```
else  
nil]
```

```
(defun P-option ()  
  (look-for-Pop (last-lpr]  
(defun look-for-Pop (list  
  (cond ((null list)  
        nil)  
        ((or (nequal (car(explode(car list)))) '-)  
              (nequal (cadr(explode(car list))) 'P))  
          (look-for-Pop (cdr list))  
        (t  
         (printer (implode (caddr (explode (car list)
```

```
(defun p-option ()  
  (look-for-pop (last-lpr]  
(defun look-for-pop (list  
  (cond ((null list)  
        nil)  
        ((or (nequal (car(explode(car list)))) '-)  
              (nequal (cadr(explode(car list))) 'p))  
          (look-for-pop (cdr list))  
        (t  
         (implode (caddr (explode (car list)
```

```
(setq t-printer nil)  
(defun targetprinter ()  
  (if (null t-printer)  
      then  
      (setq t-printer  
            (let ((lastlpr (last-lpr]))  
              (if (or (null lastlpr)  
                    (nequal 'lpr (car lastlpr)))  
                  nil  
                  (cond ((nequal (P-option)  
                                nil)  
                        (P-option))  
                        ((nequal (login-printer)  
                                nil)  
                        (login-printer))  
                        (t  
                         (default-printer)))))))  
      else  
      t-printer))
```

```
; *****  
; Funktionen zur Informationsgewinnung ueber Dateien  
; *****
```

```
(defun exists (filename)  
  (probed filename)
```

```
(defun filesize (filename)  
  (if (exists filename)  
      then  
      (filestat:size (filestat filename]
```

```
; *****  
; Funktionen zur Informationsgewinnung ueber Drucker  
; *****
```

```
; Zugriff auf VAX-Druckerkonfigurationsliste  
; Liste Available-Printer hat folgende Form :  
; ((Option Normbezeichnung Synonym1 .. Synonym-n) .. )
```

```
; Normdrucker liefert eine Liste alle Normbezeichnungen der Drucker  
(defun Normdrucker ()  
  (Normdrucker-1 Available-Printer))  
(defun Normdrucker-1 (l)  
  (cond  
    ((null l) nil)  
    (t  
     (cons (caddr l)  
           (Normdrucker-1 (cdr l))))  
  )
```



```

(defun printer (option)
  (printer-1 option Available-Printer))

(defun printer-1 (option list)
  (cond
   ((null list) 'illegal)
   ((equal (caar list) option) (cadr list))
   (t (printer-1 option (cdr list)))
  )
)

(defun inv-P-option (Druckername)
  (inv-P-1 Druckername Available-Printer))

(defun inv-P-1 (name list)
  (cond
   ((null list) 'illegal)
   ((equal (cadr list) name) (caar list))
   (t (inv-P-1 name (cdr list))))
)

(defun default-printer 0
  (cadr Available-Printer))

(defun buffersize 0
  '13)

(defun maxsize 0
  '1000)

(setq l-printer 'null)
(defun login-printer 0
  (if (equal l-printer 'null)
      then
      (setq l-printer (lookup-for-printer (make-list '-/.login)))
      else
      l-printer))

(defun lookup-for-printer (filelist)
  (cond ((lessp (length filelist) 3)
         nil)
        ((and (equal (car filelist) 'setenv)
              (equal (car(cdr filelist)) 'PRINTER))
         (printer (caddr filelist)))
        (t
         (lookup-for-printer (cdr filelist)

```

```

; *****
; Funktionen zur Informationsgewinnung ueber die Druckerwarteschlangen
; *****
(defun pqueue (&optional (pq-option 'lp))
  (setq pq-command (concat 'lpq 'l'-P pq-option 'l'>'l' 'pqfile))
  (pqueue2 pq-option))

(defmacro pqueue2 (pq-option)
  'illegal)
  then
  (exec .pq-command)
  (make-linelist 'pqfile)
  else
  (princ "illegal option:")
  (msg B B)
  (print pq-option)
  (terpri)

(defun pq-change (&optional (delay 30))
  (let* ((pqc-option (if (null (P-option))
                        'lp
                        (inv-P-option (P-option))))
         (old-pq (pqueue pqc-option))
         (sleep delay)
         (not (equal old-pq
                     (pqueue pqc-option))))))

(defun old-pq-change (pqc-option delay)
  (if (nequal (printer pqc-option)
              'illegal)
      then
      (let ((pq-old (pqueue pqc-option)))
        (sleep delay)
        (not (equal (car pq-old)
                    (car (pqueue pqc-option))))))
      else
      (princ "illegal option")
      (terpri)

(defun file-in-queue ()
  (look-for-frame (printed-file)
                  (pqueue (inv-P-option (P-option))

(defun look-for-frame (fname queue)
  (cond ((null queue)

```



```

nil)
((my-member fname (car queue))
 t)
(t
 (look-for-fname fname (cdr queue)

(defun CRvergessen (&optional (filename nil))
  (let ((fname (if (null filename)
                  (printed-file
                    filename)))
        (if (exists fname)
            then
              (let ((port (infile fname)))
                (fseek port -2 2)
                (prog1 (not (and (equal (readc port 'eof)
                                     (ascii 10))
                               (equal (readc port 'eof)
                                     (ascii 10))))))
                  (close port)))
            else
              nil)
    (defun initEA ()
      (setq leztier-druckbefehl nil)
      (setq l-printer nil)
      (setq l-printer nil)
      t)
    ; *****
    ; Funktionen zur Behandlung von Fragen aus dem XPS heraus
    ; *****
    (defun make-list (filename)
      (let ((port (infile filename)))
        (prog1 (make-list2 port)
          (close port)

(defun make-list2 (port)
  (let ((next (ratom port 'eof)))
    (if (equal next 'eof)
        nil
        (cons next
              (make-list2 port)

(defun cleanup ()
  (exec rm -f lsfile pqfile ~/history.file)
  (terpri)
  (terpri)
  (terpri)
  (princ "
OWN FILES REMOVED")
  (terpri)
  (terpri)
; *****
; Funktionen zur Behandlung von Fragen aus dem XPS heraus
; *****
(defun stelle-frage (art)
  (terpr)
  (terpr)
  (princ (frage art))
  (msg B B)
  (antwort art)

(defun frage (art)
  (caseq art
    ('druckgekommen
     (concat "Ist der Druck bei Drucker " (targetprinter) " angekommen ? "))
    ('wohindruck
     "Auf welchem Drucker sollte der Druck laufen ? ")
    ('traktor-defekt
     (concat "Ist der Traktor des Druckers " (targetprinter) " defekt ? "))
    ('farbband-leer
     (concat "Ist das Farbband des Druckers " (targetprinter) " leer ? "))
    ('papier-alle
     (concat "Ist das Papier des Druckers " (targetprinter) " alle ? "))

```

```

; *****
; Hilfsfunktionen
; *****
(defun make-linelist (filename)
  (let ((port (infile filename)))
    (prog1 (make-linelist2 port)
      (close port)

(defun make-linelist2 (port)
  (let ((first (ratom port 'eof)))
    (if (equal first 'eof)
        nil
        (cons (cons first
                    (lineread port))
              (make-linelist2 port)

```



```

(papier-falsch-eingelegt
(concat "Ist das Papier im Drucker " (targetprinter) " falsch eingelegt ? "))
(offline
(concat "Steht der Drucker " (targetprinter) " auf OFFLINE ? "))
(online
(concat "Steht der Drucker " (targetprinter) " auf ONLINE ? "))
(stecker-draussen
(concat "Ist der Netzstecker des Druckers " (targetprinter) " eingesteckt ? "))
(drucker-stau
(papierstau
(concat "Ist im Drucker " (targetprinter) " ein Papierstau aufgetreten ? "))
(druckunsauber
"Ist der Druck schief, unleserlich, unsauber, schraeg oder schwach ? ")
(drucksinnlos
"Ist der Druck sinnlos ? ")
(drucklueckenhaft
"Ist der Druck lueckenhaft ? Fehlt z.B. die letzte Zeile ? ")
(t
(print art)
"Frage unbekannt")

(defun Scope (art)
(cond ((my-member art '(druckgekommen
traktor-defekt
farbband-leer
papier-alle
papier-falsch-eingelegt
offline
Online
stecker-draussen
drucker-aus
papierstau
druckunsauber
drucksinnlos
drucklueckenhaft))
janein)
(my-member art '(wohindruck))
'drucker]

(defun antwort (art)
(analyse (normsatz (setq benutzerantwort (lineread))
normwoerterliste1)
(Scope art)
art])

(defun analyse (information scope frageart)
(cond
((my-member 'help information)
(help scope)
(stelle-frage frageart))

```

```

((and (my-member 'warum information)
(nequal scope 'drucker))
'((warum)))
((and (my-member 'ausgeschlossen information)
(nequal scope 'drucker))
'((ausgeschlossen)))
(equal 'janein scope)
(analyse-janein information frageart))
(equal 'drucker scope)
(analyse-drucker information))

(defun analyse-janein (information frageart)
(cond ((and (my-member 'ja information)
(not(my-member 'nein information)))
'(ja))
((and (my-member 'nein information)
(not(my-member 'ja information)))
'(nein))
((member 'unknown information)
'((unknown)))
(t
(help 'janein)
(stelle-frage frageart)

(defun analyse-drucker (info)
(analyse-dr-1 info Available-Printer))

(defun analyse-dr-1 (info printerlist)
(cond
((null printerlist)
(princ "Dieser Drucker ist UNIXPERT nicht bekannt ")
(terpri)
(help 'drucker)
(stelle-frage 'wohindruck))
((my-member (caddr printerlist) info)
(list(list(caddr printerlist))))
(t
(analyse-dr-1 info (cdr printerlist))
)
)


```



```

; *****
; *****

```

```

; Funktionen zur Analyse von Fragen an die Erklärungskomponente

```

```

; *****
; *****

```

```

(defun Ende-der-Vordiagnose ()
  (terpri)
  (terpri)
  (princ "Die Vordiagnose ist beendet ! ")
  (princ "Noch Fragen ? ")
  (analyse-EK-fragen nil
    'Ende-der-VD
    nil))

(defun analyse-EK-fragen (satz aufruf-von frageart)
  (if (null satz)
      then
      (terpri)
      (terpri)
      (princ "> "))
    (let ((info (if (null satz)
                    then
                    (filter?-aus-satz (lineread))
                    else
                    (satz))))
      (if (not (my-member 'nein (normsatz info normnein)))
          then
          (analyse-EK-frage info
            aufruf-von
            frageart)
          (if (null satz)
              (analyse-EK-fragen nil
                aufruf-von
                nil))
          (let ((info (normsatz satz normwoerterliste3)))
            (cond ((not (null info))
                   (EK (list 'VDE 'Kondelem (car info))))
                  ((not (null frageart))
                   (EK (list 'VDE 'Diagnosebereich frageart)))
                  (t
                   (princ " unzulässige Eingabe ! ")
                   (terpri)
                   (terpri)
                   (if (equal aufruf-von 'Ende-der-VD)
                       then
                       (princ " Wenn Du keine Fragen mehr hast gib 'weiter' ein ! ")
                       (help 'VDE)
                       (terpri))
                   (defun zahl-in-liste (liste)
                     (cond ((null liste)
                            nil)
                           ((my-numberp (car liste))
                            (my-numberp (car liste)))
                           (t
                            (princ " zahl-in-liste (liste)
                                (cond ((null liste)
                                       nil)
                                       ((my-numberp (car liste))
                                        (my-numberp (car liste)))
                                       (t
                                        (setq ready nil)
                                        )))
                            (EK 'VDE Diagnosebereich fehler-druckbefehl)))
                   ((my-member 'fehler-falscher-drucker info)
                    (and (my-member 'falsch.info)
                         (my-member 'drucker info)))
                   (EK 'VDE Diagnosebereich fehler-falscher-drucker)))
            ((my-member 'funktionsstoerung-drucker info)
             (EK 'VDE Diagnosebereich funktionsstoerung-drucker))
            (or
             (my-member 'fehler-falscher-drucker info)
             (and (my-member 'falsch.info)
                  (my-member 'drucker info)))
             (EK 'VDE Diagnosebereich fehler-falscher-drucker)))
            ((my-member 'fehler-file info)
             (EK 'VDE Diagnosebereich fehler-file)))
            (my-member 'funktionsstoerung-drucker info)
            (EK 'VDE Diagnosebereich funktionsstoerung-drucker info)
            (EK 'VDE Diagnosebereich funktionsstoerung-drucker))
            (not (null (zahl-in-liste info)))
            (EK (list 'VDE 'Diagnosebereiche (zahl-in-liste info))))
            (my-member 'pos info)
            (EK 'VDE Diagnosebereiche +)))
            (my-member 'neg info)
            (EK 'VDE Diagnosebereiche -)))
            (my-member 'alle info)
            (EK 'VDE Diagnosebereiche gesamt)))
            (not (null (zahl-in-liste info)))
            (EK (list 'VDE 'Diagnosebereiche (zahl-in-liste info))))))

```

```

(if (null ready)
    then
    (print satz)
    (terpri))

(let ((info (normsatz satz normwoerterliste3)))
  (cond ((not (null info))
         (EK (list 'VDE 'Kondelem (car info))))
        ((not (null frageart))
         (EK (list 'VDE 'Diagnosebereich frageart)))
        (t
         (terpri)
         (terpri)
         (princ " unzulässige Eingabe ! ")
         (terpri)
         (terpri)
         (if (equal aufruf-von 'Ende-der-VD)
             then
             (princ " Wenn Du keine Fragen mehr hast gib 'weiter' ein ! ")
             (help 'VDE)
             (terpri))
         (defun zahl-in-liste (liste)
           (cond ((null liste)
                  nil)
                 ((my-numberp (car liste))
                  (my-numberp (car liste)))
                 (t
                  (princ " zahl-in-liste (liste)
                      (cond ((null liste)
                             nil)
                          ((my-numberp (car liste))
                           (my-numberp (car liste)))
                          (t
                           (setq ready nil)
                           )))
                  (EK 'VDE Diagnosebereich fehler-druckbefehl)))
           ((my-member 'fehler-falscher-drucker info)
            (and (my-member 'falsch.info)
                 (my-member 'drucker info)))
           (EK 'VDE Diagnosebereich fehler-falscher-drucker)))
           ((my-member 'funktionsstoerung-drucker info)
            (EK 'VDE Diagnosebereich funktionsstoerung-drucker))
           (or
            (my-member 'fehler-falscher-drucker info)
            (and (my-member 'falsch.info)
                 (my-member 'drucker info)))
            (EK 'VDE Diagnosebereich fehler-falscher-drucker)))
           ((my-member 'fehler-file info)
            (EK 'VDE Diagnosebereich fehler-file)))
           (my-member 'funktionsstoerung-drucker info)
           (EK 'VDE Diagnosebereich funktionsstoerung-drucker info)
           (EK 'VDE Diagnosebereich funktionsstoerung-drucker))
           (not (null (zahl-in-liste info)))
           (EK (list 'VDE 'Diagnosebereiche (zahl-in-liste info))))
           (my-member 'pos info)
           (EK 'VDE Diagnosebereiche +)))
           (my-member 'neg info)
           (EK 'VDE Diagnosebereiche -)))
           (my-member 'alle info)
           (EK 'VDE Diagnosebereiche gesamt)))
           (not (null (zahl-in-liste info)))
           (EK (list 'VDE 'Diagnosebereiche (zahl-in-liste info))))))

```



```
(car liste))
(t
(zahl-in-liste (cdr liste)
```

```
(defun EK-Antwort (liste)
(terpri)
(print liste)
(terpri) )
```

```
: *****
: *****
: *****
: *****
```

```
: Funktionen zur Analyse der Starteingabe des Benutzers
```

```
: *****
: *****
: *****
: *****
```

```
(setf last-context nil)
```

```
(defun problem (&optional (nr 1))
(terpri)
(cond ((= nr 1) (princ "Sie hatten Probleme beim Absetzen
eines UNIX-Druckauftrages?
```

```
Bitte beschreiben Sie Ihr Problem: ")
```

```
(msg B B))
(princ " Bitte formulieren Sie Ihr Problem wie angegeben. ")
(terpri)
(msg B B)))
```

```
(let ((info (analyse-start (normsatz (lineread) normwoerterliste))))
(if (null(car info))
then
(terpri)
(princ "Diese Eingabe kann nicht analysiert werden ! ")
(help 'start)
(terpri)
(problem (add1 nr))
elseif
(my-member 'help (car info))
then
(help 'start)
(problem 1)
```

```
else
(check-info info nr]
```

```
(defun check-info (info nr)
(terpri)
(gib-aus info)
(terpri)
(terpri)
(princ "Bitte bestaetigen Sie die Analyse Ihres Problems (ja/nein)")
(terpri)
(if (member 'ja (normsatz (lineread) normwoerterliste1))
then
(terpri)
info
else
(problem (add1 nr)
```

```
(defun gib-aus (info)
(if (null info)
then
(terpri)
else
(setq nicht (if (member -1 (car info))
'nicht
'1))
(cond ((member 'druckgekommen (car info))
(princ (concat "Der Druck ist " nicht " angekommen"))
((member 'druckunsauber (car info))
(princ (concat "Der Druck ist " nicht " unsauber"))
((member 'drucksinnlos (car info))
(princ (concat "Der Druck ist " nicht " sinnlos"))
((member 'drucklueckenhaft (car info))
(princ (concat "Der Druck ist " nicht " lueckenhaft"))))
(terpri)
(gib-aus (cdr info))))
```

```
(defun analyse-start (eingabe)
(if (my-member 'help eingabe)
then
'((help))
else
(if (or (null eingabe)
(null (teilsatz2 eingabe)))
(terpri)
(list (analyse-teilsatz (teilsatz1 eingabe)))
(cons (analyse-teilsatz (teilsatz1 eingabe))
(analyse-start (teilsatz2 eingabe))
```

```
(defun analyse-teilsatz (eingabe)
(cond
((my-member 'druck eingabe)
(Analyse-druck eingabe))
(t
(Analyse-context eingabe)
```



```
; *****  
; weitere Hilfsfunktionen  
; *****
```

```
(defun Analyse-druck (eingabe)  
  (setq last-context 'druck)  
  (let ((certainty-factor (if (my-member 'nicht eingabe)  
                              -1  
                              1))))  
    (cond ((my-member 'kam eingabe)  
          (list 'druckgekommen certainty-factor))  
          (my-member 'unsauber eingabe)  
          (list 'druckunsauber certainty-factor))  
          (my-member 'sinnlos eingabe)  
          (list 'drucksinnlos certainty-factor))  
          (my-member 'lueckenhaft eingabe)  
          (list 'drucklueckenhaft certainty-factor]))  
  
  (defun Analyse-context (eingabe)  
    (let ((certainty-factor (if (my-member 'nicht eingabe)  
                                -1  
                                1))))  
      (cond ((equal last-context 'druck)  
            (cond ((my-member 'kam eingabe)  
                  (list 'druckgekommen certainty-factor))  
                  ((my-member 'unsauber eingabe)  
                   (list 'druckunsauber certainty-factor))  
                  ((my-member 'sinnlos eingabe)  
                   (list 'drucksinnlos certainty-factor))  
                  ((my-member 'lueckenhaft eingabe)  
                   (list 'drucklueckenhaft certainty-factor))))  
            (t (print "unknown context")  
               nil))))
```

```
; *****  
; Hilfsfunktionen zum Unterteilen der Eingabesatze  
; *****
```

```
(defun teilsatz1 (satz)  
  (if (or (null satz)  
         (equal 'und (car satz))))  
      nil  
      (cons (car satz)  
            (teilsatz1 (cdr satz))))  
  
(defun teilsatz2 (satz)  
  (cond ((null satz)  
        nil)  
        ((equal (car satz) 'und)  
         (cdr satz))  
        (t  
         (teilsatz2 (cdr satz))
```

```
(defun my-member (elem list)  
  (not(null(member elem list)))  
  
(defmacro my-numberp (form)  
  `(numberp ,form))  
  
(defun help (fuer)  
  (terpri)  
(terpri)  
(help-txt fuer)  
(terpri))  
  
(defun help-txt (fuer)  
  (caseq fuer  
    ('janein  
     'princ " Sie koennen hier folgendes eingeben:  
     ja  
     nein  
     weiss nicht  
     warum (dann erklare ich Ihnen, warum ich diese Frage stelle)  
     ")  
    (terpri))
```

```
(drucker  
  (princ "moegliche Drucker sind: ")  
  (terpri)  
(help-list-printers Available-Printer)  
(terpri))  
  
(start  
  (princ " Sie muessen beschreiben was falsch gelaufen ist.")  
  (terpri)  
  (princ " Die folgenden Eingaben sind beispielsweise zulassig : ")  
  (terpri)  
  (princ " 'Der Ausdruck ist nicht angekommen' ")  
  (terpri)  
  (princ " 'Der Druck ist unsauber' ")  
  (terpri)  
  (princ " 'Der Druck ist lueckenhaft' ")  
  (terpri)  
  (princ " 'Der Ausdruck ist sinnlos' ")  
  (terpri)  
  (princ " Die Eingabe von Sonderzeichen kann zu Fehler fuehren !! ")  
  (terpri))  
(VDE
```



```
(princ "
Die Vordiagnose ist nun beendet. Jeder der folgenden vier Diagnosebereiche
wurde mit einem Certainty-Factor zwischen 1 und -1 belegt:

funktionsstoerung-drucker
fehler-file
fehler-druckbefehl
falscher-drucker
```

Sie koennen Sie sich

```
alle,
nur die wahrscheinlichen (positiven)
nur die unwahrscheinlichen (negativen)
oder eine bestimmte Anzahl von Diagnosebereichen (1 - 4)
ausgeben lassen"]
```

; Alle definierten Drucker ausgeben

```
(defun help-list-printers (plist)
  (cond
    ((null plist) nil)
    (t
     (princ (concat " " (caddr plist))))
     (help-list-printers (cdr plist))))
  )
)
```

```
; *****
```

; Normwertertabellen

```
; *****
```

```
(defun make-printer-dict (Plist)
  (cond
    ((null Plist) nil)
    (t (cons (append (list (caddr Plist))
                      (list (caar Plist)))
              (caddr Plist)))
          (make-printer-dict (cdr Plist))))
  )
)

(setq normwoerterliste1
  (append
   (make-printer-dict Available-Printer)
   (drucker printer plotter)
   (druck listing ausdruck ausgabe datei)
   (nicht mit garnicht)
   (unsauber schief schraeg undeutlich unleserlich unbrauchbar schwach)
   (lueckenhaft zeichenaussetzer unvollstaendig)
   (sinnlos sinn unsinnig)
   (kam angekommen ausgedruckt ausgegeben gedruckt gekommen kumme)
```

```
(ja jo joh j y yes)
(nein nae nach noi nee ne n no)
(unknown ahnung keine nicht weis weiss wissen)
(und aber doch jedoch)
(help)
(warum wieso weshalb w wa)
(ausgeschlossen aus unmoeglich moeglich)
)))
```

; Hier koennen beliebige Erweiterungen gemaess dem eigenen Wortschatz
; vorgenommen werden

```
(setq normwoerterliste2
```

```
(
  (help h)
  (falsch falscher falschen)
  (drucker printer plotter)
  (fehler-file file filefehler file-fehler)
  (fehler-druckbefehl druckbefehl befehl)
  (fehler-falscher-drucker falscher-drucker)
  (funktionsstoerung-drucker funktionsstoerung defekt kaputt stoerung gestoert)
  (pos positiven positiv wahrscheinlich wahrscheinlich +)
  (neg negativen negativ unwahrscheinlich unwahrscheinlichen -)
  (alle gesamt gesamte)
  (1)
  (2)
  (3)
  (4)
  (ja j y yes jo)))
```

```
(setq normwoerterliste3
```

```
(
  (druck gekommen gekommen kam)
  (richtiger-drucker drucker drucker plotter printer)
  (fehler-druckbefehl befehl druckbefehl)
  (option-korrekt)
  (datei-korrekt datei file)
  (datei-vorhanden da vorhanden existient existiert)
  (in-warteschlange warteschlange)
  (druckunsauber unsauber schief)
  (druck sinnlos sinnlos unsinnig sinn)
  (druck lueckenhaft luecken lueckenhaft aussetzer fehlen fehlt)
  (ungleich gleich genau so)
  (option)
  (login)
  (wohindruck wohin)
  (gueltige-option)
  (nein fertig weiter go ende)
  ))
```



```
(setq normnein '(nein fertig weiter go ende end)))
```

```
; *****
```

```
; Funktionen zum Erzeugen eines Normsatzes
```

```
; *****
```

```
(defun normsatz (satz normwoerterliste)
  (cond ((null satz)
         nil)
        (t (if (null (normwort(car satz) normwoerterliste))
                then
                (normsatz (cdr satz) normwoerterliste)
                else
                (cons (normwort (car satz)
                               normwoerterliste)
                      (normsatz (cdr satz)
                               normwoerterliste)))))
```

```
(defun normwort (wort normwoerter)
  (cond ((null (car normwoerter))
         nil)
```

```
        ((my-member wort
                     (car normwoerter))
         (car normwoerter))
```

```
        ((my-member (klein wort)
                     (car normwoerter))
         (car normwoerter))
```

```
        (t (normwort wort
                      (cdr normwoerter))
```

```
(defun klein (wort)
  (if (my-numberp wort)
      then
      wort
      else
      (implode (gross-to-klein (exploden wort))
```

```
(defun gross-to-klein (wortliste)
  (cond ((null wortliste)
         nil)
        ((null (cdr wortliste))
         (list (change (car wortliste))))))
```

```
(t (cons(change(car wortliste))
         (gross-to-klein (cdr wortliste))
      )
  (defun change (chr-nr)
    (cond ((and (< chr-nr 91)
                (> chr-nr 64))
           (+ chr-nr 32))
          (t chr-nr)
```

```
(defun filter-?-aus-satz (satz)
  (cond ((null satz)
         nil)
        ((my-numberp (car satz))
         (cons (car satz)
               (filter-?-aus-satz (cdr satz))))
        (t (cons (implode (filter-?-aus-wortliste (explode (car satz))))
                  (filter-?-aus-satz (cdr satz)))))
```

```
(defun filter-?-aus-wortliste (wortliste)
  (cond ((null wortliste)
         nil)
        ((equal (car wortliste)
                 '?)
         (filter-?-aus-wortliste (cdr wortliste)))
        (t (cons (car wortliste)
                  (filter-?-aus-wortliste (cdr wortliste)))))
```

```
-----
; --- UNIXPERT.2
; --- Filename : ABHILFE
; -----
```

```
(destroy rule)
(destroy rules)
(destroy ungleich init-fakten-basis)
```

```
(ruleset 1)
(regel 1 ((option _p)
         (wohindruck _p))
         ()
         ((hypo richtiger-drucker 1)))
```

```
(regel 2 ((option nil)
         (login _p)
         (wohindruck _p))
         ()
         ((hypo richtiger-drucker 1)))
```



```

(regel 3 ((option nil)
(login nil)
(defaultprinter _p)
(wohindruck _p)))
()
((hypo richtiger-drucker 1)))

(regel 4 ((option _p1)
(wohindruck _p2)
(ungleich _p1 _p2)))
()
((hypo richtiger-drucker -1)))

(regel 5 ((option nil)
(login _p1)
(wohindruck _p2)
(ungleich _p1 _p2)))
()
((hypo richtiger-drucker -1)))

(regel 6 ((option nil)
(login nil)
(defaultprinter _p1)
(wohindruck _p2)
(ungleich _p1 _p2)))
()
((hypo richtiger-drucker -1)))

(regel 7 ((factum druckgekommen))
(0 1)
((hypo richtiger-drucker 1)))

(regel 8 ((option _p)
(factum (gueltige-option _p))))
(-1 1)
((hypo option-korrekt 1)))

(regel 10 ((dateiname nil)
(0 1)
(hypo datei-korrekt -1)))

(regel 11 ((dateiname _d)
(factum (datei-vorhanden _d))))
(-1 1)
(hypo datei-korrekt 0.9)))

(regel 12 ((factum option-korrekt)
(factum datei-korrekt))
(-1 1)
(hypo fehler-druckbefehl -0.95)))

(regel 13 ((negativ fehler-druckbefehl)
(factum richtiger-drucker))
(0 1)
(hypo funktionsstoerung-drucker 0.8)))

(regel 14 ((negativ druckgekommen)
(negativ fehler-druckbefehl)
(negativ richtiger-drucker 1)))
(0 1)
(hypo funktionsstoerung-drucker 1)))

(negativ richtiger-drucker
(negativ in-warteschlange))
(-1 1)
((hypo fehler-falscher-drucker 1)))

(regel 15 ((factum druckunsauber))
(0 1)
(hypo druckgekommen 1)
(hypo funktionsstoerung-drucker 1)))

(regel 16 ((factum drucksinnlos))
(0 1)
(hypo druckgekommen 1)
(hypo fehler-file 0.85)))

(regel 17 ((factum drucklueckenhaft))
(0 1)
(hypo druckgekommen 1)
(hypo fehler-file 0.8)
(hypo funktionsstoerung-drucker 0.5)))

(regel 18 ((factum druckgekommen))
(0 1)
(hypo fehler-file 0.1)
(hypo funktionsstoerung-drucker 0.2)))

(regel 19 ((negativ druckgekommen)
(negativ fehler-druckbefehl)
(negativ richtiger-drucker)
(factum in-warteschlange))
(0 1)
((hypo funktionsstoerung-drucker 1)))

(ruleset 2)

(regel 101 ((unknown druckgekommen)
)
(0 1)
((frage druckgekommen)))

(regel 102 ((negativ druckgekommen)
(not (wohindruck ID))
)
(0 1)
((frage wohindruck)
(system-frage login)
(system-frage option)
(system-frage dateiname)))

(regel 103 ((negativ druckgekommen)
(not (dateiname nil))
(dateiname _d)
(unknown (datei-vorhanden _d)))
(0 1)
((system-frage (datei-vorhanden _d))))

```



```

(regel 104 ((negativ druckgekomen)
  (negativ fehler-druckbefehl)
  (dateiname d)
  (unknown in-warteschlange))
  (0 1)
  ((system-frage in-warteschlange)))

(ass (ungleich _p1 _p2) (not (null _p1)) (not (null _p2)) (not (equal _p1 _p2)))

; grundzustand der fakten-basis herstellen
(ass (init-fakten-basis)
  (destroy defaultprinter wohindruck option dateiname login factum fired)
  (get-quellige-optionen)
  (is _def (default-printer))
  (ass (defaultprinter _def)
    )
  )
-----
;--- UNIXPERT.2
;--- Filename : ABHILFE
-----

```

```

;--- regel-interpreter/compiler und laufzeitfunktionen

```

```

(destroy uxp-cond hypo rule-set l interpret interpret1)

```

```

;---- allgemeine funktionen -----

```

```

;-- liefer das n-te Element der Liste l

```

```

(def n-th-elem
  (lambda (n l)
    (cond
      ((eq 1 n) (car l))
      (t (n-th-elem (sub1 n) (cdr l))))
    )
  )
)

```

```

;-- testet ob min <= cf <= max

```

```

(def test-cf
  (lambda (min max cf)
    (cond
      ((and (<= min cf) (<= cf max)) t)
      (t nil)
    )
  )
)

```

```

;-- berechnet den neuen cf fuer eine hypothese aufgrund der alten-cf
; dem cf der bedingung und dem cf der hypothese aufgrund der bedingung

```

```

(def new-cf
  (lambda (old-cf cond-cf rule-cf)
    (let ((cf (times cond-cf rule-cf)))
      (cond
        ((and (< 0 cf) (< 0 old-cf))
         (plus old-cf (times cf (plus 1 (minus old-cf)))))
        ((and (> 0 cf) (> 0 old-cf))
         (plus old-cf (times cf (plus 1 old-cf))))
        (t (quotient (plus old-cf cf) (plus 1 (minus (min (abs cf) (abs old-cf)))))))
      )
    )
  )
)

```

```

;-- stack-funktionen fuer schrittweise erklarungskomponente

```

```

;-- der uxp-stack ist eine globale variable zur uebergabe der Liste der

```

```

;-- instanziierten gefuerten Regeln und hat folgenden Aufbau

```

```

; uxp-stack=((<regelname1> (<Praemisse1.1> <CF1>) .. (<Praem1.n> <CFn>))
;          ((<Konklusion1.1> <CF1>) .. (<Konklusion1.n> <CFn>)))

```

```

;          (<regelname2> ....)

```

```

;          )

```

```

;-- (uxp-cond _co) pusht _co auf zum beweis _co auf den uxp-c-stack

```

```

;-- und popl _co beim backtracken wieder runter

```

```

(ass (uxp-cond _co) (uxp-push-c _co))
(ass (uxp-cond ID) (uxp-pop-c) (fail))

```

```

;-- Lisp-stack-operationen auf uxp-c-stack

```

```

;-- loescht den uxp-c-stack

```

```

(def uxp-init-c
  (lambda ()
    (setq uxp-c-stack nil)
    )
  )
)

```

```

;-- pusht l auf den uxp-c-stack

```

```

(def uxp-push-c
  (lambda (l)
    (setq uxp-c-stack (cons (remove-free l) uxp-c-stack))
    )
  )
)

```

```

;-- popl beim backtrack com uxp-c-stack

```

```

(def uxp-pop-c
  (lambda ()
    (setq uxp-c-stack (cdr uxp-c-stack))
    )
  )
)

```



```

)
t
)
)

;- ersetze freie variablen in liste durch das atom 'beliebig
(def remove-free
  (lambda (l)
    (cond
      ((null l) nil)
      ((eq 1 'ID) 'beliebig)
      ((atom l) l)
      ((eq (car l) '?)
       'beliebig)
      (t (mapcar 'remove-free l)
        )
    )
  )
)

;- eintrag eines Praedikates in Datenbasis und Berechnung der Certainty
;- und eintragen in uxp-stack

;- praedikat_h ist schon vorhanden
(ass (hypo_h_cf-cond_cf-rule)
  (factum_h_h_cf-old)
  (rex (factum_h_h_cf-old))
  (is_newcf (new-cf_cf-old_cf-cond_cf-rule))
  (ass (factum_h_h_newcf)
    (uxp-action (_h_newcf)))
)

;- praedikat_h ist noch nicht vorhanden
(ass (hypo_h_cf-cond_cf-rule)
  (not (factum_h_h_cf-old))
  (is_newcf (new-cf_0_cf-cond_cf-rule))
  (ass (factum_h_h_newcf)
    (uxp-action (_h_newcf)))
)

;----- DER REGEL-COMPILER -----

;- variablen-symbole
(setq var-cf '(cf1_cf2_cf3_cf4_cf5_cf6_cf7_cf8_cf9_cf10))
(setq var-cf1 '(cf1a_cf2a_cf3a_cf4a_cf5a_cf6a_cf7a_cf8a_cf9a_cf10a))

;- liefert die ersten n-elemente der liste l
(def n-first
  (lambda (n l)
    (cond
      ((= 0 n) nil)
      (t (cons (car l) (n-first (sub1 n) (cdr l))))
    )
  )
)

;- macro zum festlegen der regelmenge fuer die folgenden regeln
(defmacro ruleset (set-name)
  (list 'setq 'rule-set-name set-name)
)

;- macro zum analysieren der regel
;- r-name ist der name (nummer) der regel
;- cond-l ist die Praemissenliste
;- action-l ist die Konklussionsliste
(defmacro regel (r-name cond-l fire-r action-l)
  (eval (list 'ass (list 'rules rule-set-name)
    (list 'not (list 'fired r-name))
    (list 'rule r-name)
  ))
)

```

```

;- Der uxp-stack ist der haupt-stack fuer abgearbeiteten regeln
;- uxp-stack loeschen
(def uxp-init
  (lambda ()
    (setq uxp-stack nil)
  )
)

;- uxp-c-stack ist stack der gefuehrten Praemissen
;- anhaengen des uxp-c-stack und der regel-nummer an den uxp-stack
;- am ende des aufgebauten cond-stack ,alle condition-elemente sind erfuehlt
(def uxp-trans-c-stack
  (lambda (rule)
    (setq uxp-stack (cons
      (list rule uxp-c-stack nil)
      uxp-stack)
    )
  )
)

;- anhaengen einer aktion (hypothese) an das regel-element des uxp-stack
;- Die Aktion wird der letzten eingetragenen Praemissenliste zugeordnet
;- wird vor jeder aktion aufgerufen
(def uxp-action
  (lambda (act)
    (setq uxp-stack
      (cons (list (caar uxp-stack)
        (cadr uxp-stack)
        (cons act
          (caddr uxp-stack)
        )
      )
      uxp-stack)
    )
  )
)

```



```
(cons act
  (cons-action (cdr l))))
)
)
)
)
)
)
```

```
;ausfuehren der regelmenge rule-set(name erfolg) , erfolg=t wenn eine regel feuerte
;sonst nil
```

```
(ass (rule-set _set-name _success)
  (is _succ (prove (rules _set-name)) 1 nil))
(rule-set1 _set-name _succ _success))

(ass (rule-set1 ID nil nil))
(ass (rule-set1 _set-name ID t)
  (rule-set _set-name ID))

;ausfuehren des interpreters : regelmenge 1 2 1 ...
(ass (interpret (rule-set 1 ID) (interpret 1 2 t))
  (ass (interpret 2 t) (rule-set 2 _res) (interpret 1 1 _res))
  (ass (interpret 1 t) (rule-set 1 _res) (interpret 1 2 _res))
  (ass (interpret 1 ID nil)))
```

```
-----
;--- UNIXPERT.2
;--- Filename : RULE.SCHNITT
;---
```

```
(destroy frage system-frage trage-ein trage-ein-P)
(destroy make-h-l make-hypos construct-hypos)
(destroy problem-eintragen problem-lesen)

(ass (import$ backward assert-vd))

;- schnitstelle zur system und benutzer ein-ausgabe
;---- Eintragen der gueltigen Normbezeichnungen fuer Drucker in
;---- Faktenbasis der Vordidgnose

(ass (get-gueltige-optionen)
  (is _alle (Normdrucker))
  (gueltige-optionen-eintragen _alle))

(ass ( (gueltige-optionen-eintragen nil)
  (ass (factum (gueltige-option nil) 1))
  (ass (factum (gueltige-option ID) -1))))

(ass (gueltige-optionen-eintragen [_head _tail]) )
```

```
(ass ( (factum (gueltige-option _head) 1))
  (gueltige-optionen-eintragen _tail)
  )
```

```
;- eintragen einer erfragten antwort in Faktenbasis der Hauptdiagnose
```

```
;Aktionen zum Stellen einer Frage ueber System-Ein-Ausgabe .
;Die Antwort auf die Frage wird dabei in die Datenbasis eingetragen
```

```
(ass (frage druckgekommen)
  (is _a (stelle-frage druck-gekommen))
  (trage-ein druckgekommen _a))

(ass (frage wohindruck)
  (is _a (stelle-frage wohindruck))
  (trage-ein-P wohindruck _a))

(ass (trage-ein _h ((ja))) ;Antwort ist ja , Trage ein mit CF=1
  (ass (factum _h) 1))

(ass (trage-ein _h ((unknown))) ;Antwort ist weiss nicht , Trage ein mit CF=0
  (ass (factum _h) 0))

(ass (trage-ein _h ((nein))) ;Antwort ist nein , Trage ein mit CF=-1
  (ass (factum _h) -1))

(ass (trage-ein-P _p (_a)) ;Antwort ist ein Argment fuer ein Praedikat
  (ass (_p _a)) ;z.B B48 , und eingetragen wohindruck(B48))
```

```
;Stellen von Systemfragen und eintragen in die Datenbasis
```

```
(ass (system-frage option)
  (is _a (P-option))
  (ass (option _a)))

(ass (system-frage dateiname)
  (is _a (printed-file))
  (ass (dateiname _a)))

(ass (system-frage login)
  (is _a (login-printer))
  (ass (login _a)))

(ass (system-frage (datei-vorhanden _d))
  (exists _d)
  (ass (factum (datei-vorhanden _d) 1)))

(ass (system-frage (datei-vorhanden _d))
  (ass (factum (datei-vorhanden _d) -1)))

(ass (system-frage in-warteschlange)
  (file-in-queue)
  (ass (factum in-warteschlange 1)))
```



```

(ass (system-frage in-warteschlange)
  (ass (factum in-warteschlange -1)))
;Eintragen der Information , die aus der Problembeschreibung gewonnen
;wurde : Eingabe : Liste von (<Praed> <CF>)
(ass ! (problem-eintragen nil))
(ass (problem-eintragen [(p _cf) . _rest])
  (ass (factum _p _cf)
    (uxp-action ((frage _p) _cf) ;Eintragen in der Stack fuer die EK
      (problem-eintragen _rest)))
;Einlesen der Problembeschreibung aufrufen und Eintragen in die Datenbasis
;steuern
(ass (problem-lesen)
  (is _antwort (problem))
  (uxp-init-c) ;EK-Stack initialisieren
  (uxp-trans-c-stack 0)
  (problem-eintragen _antwort))
;-- schnitstelle zur Hauptdiagnose --
;Auslesen der Diagnosebereiche mit Cf aus der Datenbasis
;1.Arg : Liste der Name der Diagnosebereiche
;2.Arg : Ausgabe : Liste aus Elementen (<Diag-bereich> , <CF>)
(ass ! (make-h-1 nil nil))
(ass (make-h-1 [_hypo . _rest] [(_hypo _cf) . _tail])
  (factum _hypo _cf)
  (make-h-1 _rest _tail))
(ass (make-h-1 [_hypo . _rest] [(_hypo 0) . _tail])
  (make-h-1 _rest _tail))
;Erzeugen einer globalen Liste die die Diagnosebereiche sortiert enthaelt
(ass (make-hypos)
  (make-h-1 (funktionsstoerung-drucker fehler-file fehler-falscher-drucker fehler-
druckbefehl) _list)
  (construct-hypos _list))
;Liste l nach steigendem CF sortieren und an 'd-bereiche binden
(def construct-hypos
  (lambda (l)
    (setq d-bereiche (sort l #'cf-order))
    t
  )
)
;Ordnung fuer sortierung sortiere nach hoechstem cf
(def cf-order

```

```

(lambda (l l2)
  (> (cadr l1) (cadr l2))
)
;-- Uebergabe von aquirierten Praedikaten an Backward-Diagnose
;zur Vermeidung von mehrfach Fragen.
;Aufruf : (make-ass)
(ass (make-ass) ;wohindruck eintragen
  (wohindruck _d)
  (assert-vd vordidiagnose (wohindruck _d) 1) ;Eintragen in Backward-DB
  (fail))
(ass (make-ass) ;druckgekommen eintragen
  (factum druckgekommen _cf)
  (assert-vd vordidiagnose (druckgekommen) _cf)
  (fail))
(ass (make-ass) ;druckunsauber eintragen
  (factum druckunsauber _cf)
  (assert-vd vordidiagnose (druckunsauber) _cf)
  (fail))
(ass (make-ass) ;drucksinnlos eintragen
  (factum drucksinnlos _cf)
  (assert-vd vordidiagnose (drucksinnlos) _cf)
  (fail))
(ass (make-ass) ;drucklueckenhaft eintragen
  (factum drucklueckenhaft _cf)
  (assert-vd vordidiagnose (drucklueckenhaft) _cf)
  (fail))
(ass (make-ass) ;datei in warteschlange eintragen
  (factum in-warteschlange 1)
  (assert-vd system ((file-in-queue) (t)) 1)
  (fail))
(ass (make-ass)
  (factum in-warteschlange -1)
  (assert-vd system ((file-in-queue) (nil)) 1)
  (fail))
(ass ! (make-ass)) ;alles eingetragen , success
;--- UNIXPERT.2
;--- Filename : TOPAUSGABE
;---
(defun princ-breit (string)
  (patom (concat (ascii 27) "#6" string))
  t)

```



```
(defun princ-hoch (string)
  (patom (concat (ascii 27) "#3" string))
  (terpri)
  (patom (concat (ascii 27) "#4" string))
  1)
```

```
(princ-hoch "*****")(terpri)
(princ-hoch "*"
  (princ (concat (ascii 27) "#3" "*"
    (ascii 27) "7m" (ascii 27) "#3" " UNIXPERT-2
    (ascii 27) "[0m" (ascii 27) "#3" "*"
    (ascii 27) "7m" (ascii 27) "#4" " UNIXPERT-2
    (ascii 27) "[0m" (ascii 27) "#4" "*"
    *)
    (terpri)
  (princ-hoch "*"
    (princ-hoch "*" Diagnose-System zur Fehlersuche *)
    (princ-hoch "*" in UNIX-Druckauftraegen. *)
    (princ-hoch "*"
      (princ-hoch "*****")(terpri)(terpri)(terpri)
```

```
-----
: --- UNIXPERT.2
: --- Filename : TOPLEVEL
: -----
```

```
: Diese Datei enthaelt die Toplevel-Funktionen
: fuer die Ablaufsteuerung der Hauptdiagnose
```

```
(destroy kill-dbase)
: dient zur Initialisierung fuer einen eventuellen Neustart.
: es werden alle vom Benutzer gemachten Eingaben in der Datenbasis
: geloescht.
(ass (kill-dbase)
  (destroy falscher-fehler)
  (member _z (system user vordidiagnose))
  (retract-name _z)
  nil)
(ass (kill-dbase))

(destroy back)
: Dies ist das Praedikat, das die Ablaufsteuerung
: fuer die Hauptdiagnose uebernimmt
(ass (back)
  (is_db (d-ber))
  (not (terpri))
  (princ-hoch " Beginn der Hauptdiagnose "
    (not (terpri)) (not (terpri))
    (toplevel_db)
    (frage-neuer-durchlauf))
  (defun frage-neuer-durchlauf()
    (princ
```

Ich freue mich,dass ich Ihnen helfen konnte.

 Moechten Sie eine weitere Anfrage an

```
"
(princ-breit "UNIXPERT2")
(princ "
stellen?
--> "
(let ((hlf (read))))
  (if (member hlf '(j ja y yes))
    then
    (n-solutions '(unixpert) 1)
  else
  (if (member hlf '(n no nein))
    then
    (terpri)
  (terpri)
  (princ-hoch " Auf Wiedersehen! ")
  (terpri)
  (terpri)
  (exit)
  else
  (princ "
+++++++
Sie koennen hier nur JA oder NEIN eingeben.
Please acknowledge: "
(readline)
(frage-neuer-durchlauf))))))
```

```
(defun d-ber() d-bereiche)
; d-bereiche ist die globale Variable,die die Vordidiagnose liefert.
; Sie enthaelt die Diagnosebereiche mit den von der Vordidiagnose
; berechneten CF's.
```

```
(destroy toplevel)
; toplevel waehlt einen Diagnose-bereich aus und versucht
; darin nacheinander mit den verschiedenen Strategien zu beweisen
(ass (toplevel _diagnose-liste)
  (is _diagnose-liste-sortiert
    (partitioniere _diagnose-liste))
  (member _diag-bereich-liste _diagnose-liste-sortiert)
  (initialize-tree))
```



```

; erzeugt den zum Praedikat p gehoerigen Beweisbaum
; und liefert einen Zeiger darauf zumeck. Der globale Baum
; wird vorher gesichert, so dass er durch diese Funktion nicht
; geaendert wird
(let ((altgt (gt)
      (akt-node *akt-node*)
      (initialize-tree)
      (n-solutions `(beweise ,p) 1)
      (prog1 (vererbe (gt))
              (setq *gt* altgt)
              (setq *akt-node* akt-node))))))

```

```

(defun set-wg(n)
  ; setzt den Waschgang auf n (der Waschgang ist der Inhalt
  ; der Wurzel des Globalen Baumes)
  (setf (nth 2 *gt*) n))

```

```

(destroy hilf-call)
(ass (hilf-call _f)
     (call ((false _f))
           (ass (falscher-fehler _f))))

```

```

(defun teste-ob-false(f)
  ; testet ob Fehler f als false beweisbar
  (if (n-solutions `(falscher-fehler ,f) 1)
      nil
      (not (n-solutions `(hilf-call ,f) 1))))

```

```

(defun ausgeschlossene-fehler()
  ; liefert die Liste aller bisher als false
  ; bewiesener Fehler
  (prove '((falscher-fehler _x) 100 ' _x))

```

```

(destroy member)
(ass (member _x (_x . ID)))
(ass (member _x (ID . _rest))
     (member _x _rest))

```

```

(defun callfun (auf)
  ; Startet einen System-Aufruf 'auf
  ; und speichert das Ergebnis
  (let* ((hilf (car (prove `(rule system (auf _wert)) 1 ' _wert))))
    then
    (terpri)
    (princ "Aufruf Systemfunktion : ")
    (princ auf) (terpri)
    (setq hilf (list (eval auf)))
    (assert system `(auf ,hilf))
    (car hilf)))

```

```

(defun partitioniere(m)
  (let ((hilf1 (sort m #'mysort))
        (hilf2 nil)(hilf3 nil))
    (mapc #(lambda (x) (if (> (cadr x) 0.2)

```

```

(bew _diag-bereich-liste _f)
  (ausgabe _f)
  (ass (toplevel ID)
      (princ "
*****
*
* Es tut mir wirklich sehr leid, aber ich kann *
* Ihnen leider nicht weiterhelfen. *")
      (princ "
*****
*
*****"))

```

```

""
(exit))

(defun initialize-tree()
  ; initialisiert den globalen Baum
  (setq *akt-node* (list nil nil '(ok wurzel)))
  (setq *gt* *akt-node*)
  t)

```

```

(destroy bew)
; versucht einen Fehler im Diagnose-Bereich zu beweisen
(ass (bew _diag-bereich-liste _f)
     (member _diag-bereich _diag-bereich-liste)
     (rule ID (_diag-bereich _f) . ID)
     (set-wg (1 _diag-bereich))
     (teste-ob-false _f)
     (bew-true (_f)))

```

```

(ass (bew _diag-bereich-liste _f)
     (member _diag-bereich _diag-bereich-liste)
     (rule ID (_diag-bereich _f) . ID)
     (teste-ob-false _f)
     (set-wg (2 _diag-bereich))
     (bew-true (_f)))
(ass (bew _diag-bereich-liste _f)
     (member _diag-bereich _diag-bereich-liste)
     (rule ID (_diag-bereich _f) . ID)
     (set-wg (1 _diag-bereich))
     (teste-ob-false _f)
     (set-wg (3 _diag-bereich))
     (bew-unknown (_f)))

```

```

(destroy dbereich)
; liefert den Diagnose-Bereich zu dem ein bestimmter Fehler gehoert
(ass ! (dbereich _db (false _f))
     (rule ID (_db _f) . ID))
(ass (dbereich _db _f)
     (rule ID (_db _f) . ID))
(ass (dbereich _db unknown))

(defun erzeuge-baum (p)

```



```

(setq hilf2 (append1 hilf2 (car x)))
(if (> (cadr x) -0.5)
  (setq hilf3 (append1 hilf3 (car x))))))
  hilf1)
(list hilf2 hilf3)))
(defun my-sort (x y)
  (> (cadr x) (cadr y)))

```

```

-----
--- UNIXPERT.2
--- Filename : UNIXPERT.LADER
-----

```

```

-----
: Hauptlade-file fuer UNIXPERT2
-----

```

```

: -laed saemtliche file fuer die Vordidiagnose und
: ; definiert die Top-Level-Steuerung der Vordidiagnose
(destroy vordidiagnose) ; Praedikat vordidiagnose loeschen
(destroy vordidiagnose) ; Modul vordidiagnose loeschen
(create vordidiagnose)
(switch-to-modul vordidiagnose)

```

```

(princ "Loading UNIXPERT2 Version 2.0 ") (terpri)

```

```

:----- files laden -----
(princ "consulting vordidiagnose") (terpri)
:-- regelcompiler/interpreter laden
(consult /usr/users/unixpert2b/rule-inter)

:-- regelbasis laden
(consult /usr/users/unixpert2b/regelbasis)

:-- Schnittstellenfunktionen laden
(consult /usr/users/unixpert2b/rule.schnitt)

(ass (export$ vordidiagnose) :-- Top-Level Steuerung exportieren
(ass (import$ backward kill-dbase))
:-(egal _p) versucht _p zu beweisen , succeeded aber immer , auch wenn _p faillt
(ass (egal _p) (_p))
(ass (egal ID))

```

```

(ass (vordidiagnose)
  (init-fakten-basis) ; consultationsspezifischen Teil der DB loeschen
  (uxp-init) ; arbeits-stack (EK) loeschen
  (problem-lesen) ; Eingabe der Problembeschreibung und eintagen

```

```

(princ-hoch " Beginn der Vordidiagnose ")
(not (terpri))
(interpret) ; Regelinterpreter starten
(make-hypos) ; Hypothesenliste generieren
(kill-dbase) ; Datenbasis der backward-diagnose loeschen
(make-ass) ; erfragt wissen in backward-diagnose uebergeben
(egal Ende-der-Vordidiagnose) ; Frage-Komponente zur Vordidiagnose starten
)

```

```

; Anweisungen, die noetig sind,
; um die Backwardinterpreter-Komponente von
; UNIXPERT2 zu laden.

```

```

(destroy backward)
(create backward)
(switch-to-modul backward)
(ass (import$ kill-dbase back toplevel assert-vd))
(ass (import$ unixpert unixpert))

```

```

(princ "consulting Backward - Interpreter") (terpri)

```

```

(consult /usr/users/unixpert2b/baum)
(consult /usr/users/unixpert2b/ausgabe)
(consult /usr/users/unixpert2b/bew)
(consult /usr/users/unixpert2b/consult)
(consult /usr/users/unixpert2b/toplevel)
; consulting Unixpert-Database

```

```

(consult /usr/users/unixpert2b/dbase-intern)
(consult /usr/users/unixpert2b/enddiagnose)
; consulting Abhilfsmassnahmen
(consult /usr/users/unixpert2b/abhilfe)

```

```

(princ "consulting System-Ein-Ausgabe") (terpri)
(consult /usr/users/unixpert2b/execfile)

```

```

(princ "consulting Erklaerungs-Komponente") (terpri)
(consult /usr/users/unixpert2b/ek)
(consult /usr/users/unixpert2b/ek-ausgabe)

```

```

(consult /usr/users/unixpert2b/topausgabe)

```

```

(destroy unixpert)
(destroy unixpert)
(create unixpert)
(switch-to-modul unixpert)
(ass (export$ unixpert))

```

```

(ass (import$ vordidiagnose vordidiagnose))
(ass (import$ backward back))
(ass (unixpert)
  (initEA)
  (vordidiagnose)
  (back)
)

```



```
(n-solutions '(unixpert) 1)
```

```
-----  
:--- UNIXPERT.2  
:--- Filename : UNIXPERT2B  
-----
```

```
echo " ----- UNIXPERT2 wird gestartet ----- "  
echo "          BITTE WARTEN "  
if [ -s .lisprc.l ]  
then cp .lisprc.l .oldlisprc  
fi  
cp /usr/users/unixpert/unixpert2b/.lisprc.unixpert2b .lisprc.l  
Lisp unixpert2b  
rm .lisprc.l  
if [ -s .oldlisprc ]  
then mv .oldlisprc .lisprc.l  
fi
```