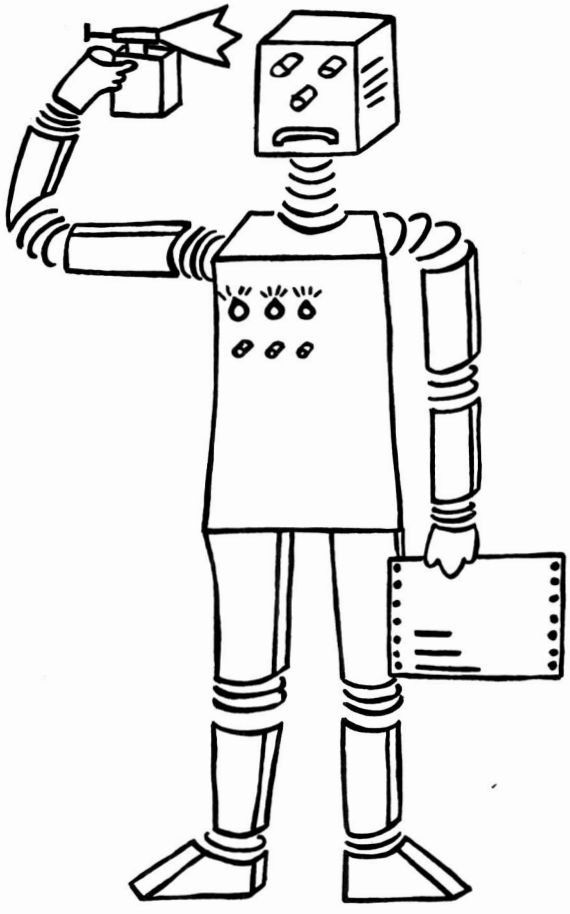
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

SEKI-Working Paper



HIQUAL

Implementierung eines Systems zur Darstellung von Zeit und Kausalität

Hans-Werner Eiden
Marc Linster

Juli 1986 SWP-86-02

HIQUAL
Implementierung eines Systems zur Darstellung von Zeit und Kausalität

Abstract

Hiqual is a language for representation and analysis of modularly constructed hierarchical physical or technical systems.
Models are the basic elements. They allow the cause-effect description of elementary systems on both the causal and temporal level.
Simple hierarchical systems are described in aggregates. Aggregates combine models on two levels of abstraction. More complex hierarchies are described by means of combined aggregates.
The analysis of a model description generates a cyclic graph containing all the possible behavioural sequences of the model.
Aggregates are checked to make sure that the different levels of abstraction describe the same, but more or less abstract behaviour. As every model may have an infinite number of behavioural sequences, these tests are only executed for user selected sequences.
The analysis is based upon Allen's temporal relations and his propagation algorithm. Hiqual allows to deduce causal relations among events belonging to different models and to find dead-locks caused by inconsistencies.
The present paper describes the implementation and mentions those problems that did not find a satisfactory solution, i.e. the representation of inequalities and discontinuous values, and the computational complexity of the analysis.

Authors address :

Hans-Werner Eiden
Kurt Schumacher Str. 22
6750 Kaiserslautern

Marc Linster
Stangensohl 25
6750 Kaiserslautern

0. Inhaltsverzeichnis

I. Einführung in die Hiqual-Implementierung

Dieses Kapitel gibt einen Überblick über die existierende Hiqual-Implementierung und stellt damit gleichzeitig die verwendete Terminologie vor.
Zur Motivation und Einführung in Hiqual wird an dieser Stelle die Lektüre von [1] vorausgesetzt.
Hiqual versteht sich als Repräsentations- und Analysesystem für qualitatives Wissen in (technischen) Systemen.
Qualitatives Wisssen bedeutet in Hiqual

- Wissen über die hierarchische Struktur von Systemen und
- Wissen über zeitlich kausale Zusammenhänge zwischen Ereignissen, dargestellt als qualitative zeitliche Relationen.

Systeme sind in Hiqual in drei verschiedenen Abstraktionsstufen modellierbar; als Modell, Aggregat oder komplexes Aggregat, wobei Modelle die Komponenten von Aggregaten und Aggregate die Komponenten von komplexen Aggregaten darstellen. Aus diesem Zusammenhang läßt sich bereits vermuten, daß es neben den horizontalen Beziehungen innerhalb einer Stufe auch vertikale Verbindungen geben muß.

I.1 Modelle

Ein Modell ist die höchste Abstraktionsebene einer Systembeschreibung. Es repräsentiert die kleinste unabhängige Einheit in Hiqual und kann somit auch unabhängig von seiner Umwelt analysiert werden.
Abb. I.1 zeigt ein Beispiel für die Modellierung eines RS-Flip-Flops. Die einzelnen Elemente der Modellbeschreibung sind nachfolgend beschrieben.

I.1.1 Strukturkomponenten

Beschreibungs- bzw. Strukturkomponenten von Modellen sind:

- In- und Outports,
- Variablen und Attribute,
- Regeln und
- Facetten.

I.1.1.1 In- und Outports

Ein Modell kann nur über seine In- und Outports mit der Außenwelt (andere Modelle) kommunizieren. Ports repräsentieren i.a. physikalische Größen, die während der Lebenszeit eines Modells Werte aus einem bekannten Wertebereich annehmen können. Inports ändern ihre Werte ausschließlich durch äußeren Einfluß, d. h. das Modell kann die entsprechenden Werte der Inports nur lesen. Umgekehrt gilt für die Outports, daß das Modell sie nur schreibend verwalten kann.

I.1.1.2 Variablen und Attribute

Variablen und Attribute sind lokal innerhalb eines Modells definierbar. Variablen sind als Hilfsspeicher gedacht, um sich bestimmte innere Zustände zu merken.
Attribute sind Konstanten, die gewisse Eigenschaften eines Modells festlegen und auf die somit nur ein lesender Zugriff erlaubt ist.

I.1.1.3 Regeln

Mit Regeln wird das Verhalten eines Modells definiert.
Sie setzen sich aus drei Komponenten zusammen:
- Precondition-Teil,
- Action-Teil,
- Relations-Teil.

Im Precondition-Teil einer Regel ist eine Liste von Conditions aufgeführt, die implizit durch logisch UND verknüpft sind. In Conditions aus dem Precondition-Teil können Inports, Variablen und Attribute auf konkrete Wertebelegungen überprüft werden. Ist der Precondition-Teil erfüllt, kann die Regel feuern, d.h. der Action-Teil ist ausführbar.
Der Action-Teil erlaubt zwei unterschiedliche Arten von Aktionen:

```
TYPE RS_FF IS MODEL

   TYPES BINARY IS (low;high);

   PORTS IN R, S ARE BINARY;
         OUT Q, NQ ARE BINARY;

   OKFACET high IS
         R1 IS PRECONDS S=low,R=low;
               ACTIONS  Q=high,NQ=low;

         R2 IS PRECONDS S=high,R=low;
               ACTIONS  Q=high,NQ=low;

         R3 IS PRECONDS S=low,R=high;
               ACTIONS  ENTER low,
                        Q=low,NQ=high;

   OKFACET low IS
         R1 IS PRECONDS S=low,R=low;
               ACTIONS  Q=low,NQ=high;

         R2 IS PRECONDS S=high,R=high;
               ACTIONS  Q=low,NQ=high;

         R3 IS PRECONS  S=high,R=low;
               ACTIONS  ENTER high,
                        Q=high,NQ=low;

INITIALLY low;

END RS_FF
```

Abb. I.1: Modellspezifikation eines RS-Flip-Flops.

R	S	Q	NQ
0	0	Q	NQ
0	1	1	0
1	0	0	1
1	1	nicht erlaubt	

Abb. I.2: Wahrheitstabelle für ein RS_FF.

- Wechsel der aktuellen Facette (siehe unten),
- Definition von Conditions, die jedoch im Gegensatz zu Preconditions eine andere Interpretation verlangen. Conditions aus dem Action-Teil sind als Assertions zu verstehen, d.h. sie legen Wertebelegungen für Outports und gegebenenfalls auch für Variablen fest. Inports und Attribute sind hier nicht zugelassen, da sie innerhalb eines Modells nicht manipulierbar sind. Conditions in diesem Teil einer Regel werden auch als NON-ENTER-Actions bezeichnet.

Auf den Relations-Teil wird später eingegangen.

I.1.1.4 Facetten

Jedes Modell hat mindestens eine Facette. Facetten lassen sich als Zustände interpretieren, die die Modellierung von differenzierten Verhaltensweisen für die gleiche Eingabekonstellation erlaubt. Sie ermöglichen ebenfalls eine Strukturierung der erforderlichen Regelmenge. Regeln sind somit immer einer Facette zugeordnent. Nur wenn sich das Modell in dem Zustand der entsprechenden Facette befindet, können die Regeln innerhalb der Facette bei gültiger Precondition feuern. Ein Modell befindet sich immer genau in einer Facette; ein Facettenwechsel ist nur über eine Enter-Action einer anwendbaren Regel möglich.
Die Initialfacette wird in der Definition eines Modells festgelegt.

I.1.2 Ereignisse und zeitliche Relationen

Ereignisse sind zunächst ausschließlich durch Regeln definiert. Alle Conditions im Precondition- und Action-Teil einer Regel sowie das Betreten einer neuen Facette sind als Ereignisse festgelegt.
Zwischen diesen Ereignissen gelten gewisse zeitliche Relationen, die auf Allen's Zeitmodell [2] basierenden, komplexen Relationen. Sie werden in [1] als Allen-Relationen referiert.
Grob gesagt beschreiben sie die Ursache-Wirkungskausalität bezogen auf die zeitliche Abfolge von Ereignissen. Im besonderen heißt das:
Es gelten Relationen zwischen dem Precondition- und dem Action-Teil einer Regel.
Es gibt vier Kategorien von Beziehungen, die durch sogenannte ontologische Typen benannt sind; siehe auch Abb. I.3.:

- Eine Facette, die durch eine Enter-Action betreten wird, ist von den Precondition-Ereignissen verursacht worden.

Ontologischer Typ: **is-defined-by**, kurz **id**.

- Non-Enter-Actions sind ebenfalls von den Precondition-Ereignissen verursacht.
  Ontologischer Typ: **is-caused-by**, kurz **ic**.
- Precondition-Ereignisse werden innerhalb der sie umschliessenden Facette beobachtet, falls sie keinen Facettenwechsel verursachen.
  Ontologischer Typ: **is-registrated-in**, kurz **ir**.
- Non-Enter-Actions werden erst in einer bestimmten Facette wirksam.
  Ontologischer Typ: **is-effected-in**, kurz **ie**.

Für jeden ontologischen Typ ist eine Menge von maximal zulässigen Relationen definiert.

- is-defined-by und
- is-caused-by (ic) steht für die Not(starts-before) - Relationenmenge {=, mi, oi, f, d, s, si}; die aussagt, daß die Wirkung nicht vor der Ursache sein kann;
- is-registrated-in (ir) steht für die occurs-in - Relationenmenge not{<, >, m, mi};
- is-effected-in (ie) steht für die starts-in - Relationenmenge {=, d, oi, s, si}.

Sind wie im Beispiel oben für das RS_FF keine expliziten Relationen aufgeführt, dann gelten die oben aufgelisteten Default-Relationenmengen zwischen den jeweiligen Ereignissen.

Beispiel einer Relationeneinschränkung:
Soll die Schaltverzögerung eines NOR-Gatters berücksichtigt werden, dann könnte eine entsprechende Regel so aussehen:

```
R1 is PRECONDS (Eing1=low, Eing2=low label1);
      ACTIONS (Ausg=high label2);
      RELATIONS label2 is mi,oi,f,d WITH label1;
```

Die im RELATIONS-Teil explizit angegebene Relationenmenge zwischen Out-Action (Belegung eines Outports) und den Preconditions sagt aus, daß der Effekt (Ausg=high) nicht vor und nicht gleichzeitig mit der Ursache (Eing1=low, Eing2=low) beginnen kann.
Die Definition von zeitlichen Relationen zwischen Ereignissen hat zur Folge, daß Ereignisse selbst eine zeitliche Ausdehnung besitzen müssen. Sie werden deshalb als qualitative zeitliche Intervalle interpretiert. Qualitative zeitliche Intervalle haben immer eine Ausdehung größer als Null, auch wenn zwischen **instantaneous** (in) und **non-instantaneous** (ni) Ereignissen unterschieden wird. Instantaneous Ereignisse stehen für Zeitpunkte; Zeitpunkte sind jedoch zeitliche Intervalle der Ausdehnung Null. Hiqual umgeht diesen Konflikt, indem es diese Intervalle auch als qualitative zeitliche Intervalle mit der Ausdehnung > Null interpretiert, jedoch ist die Ausdehnung von instantaneous Intervallen immer kleiner als die Dauer jedes non-instantaneous Intervalls. D.h. ein NI- und ein IN-Intervall kann niemals gleich sein bzw. ein IN-Intervall kann nie ein NI-Intervall enthalten.

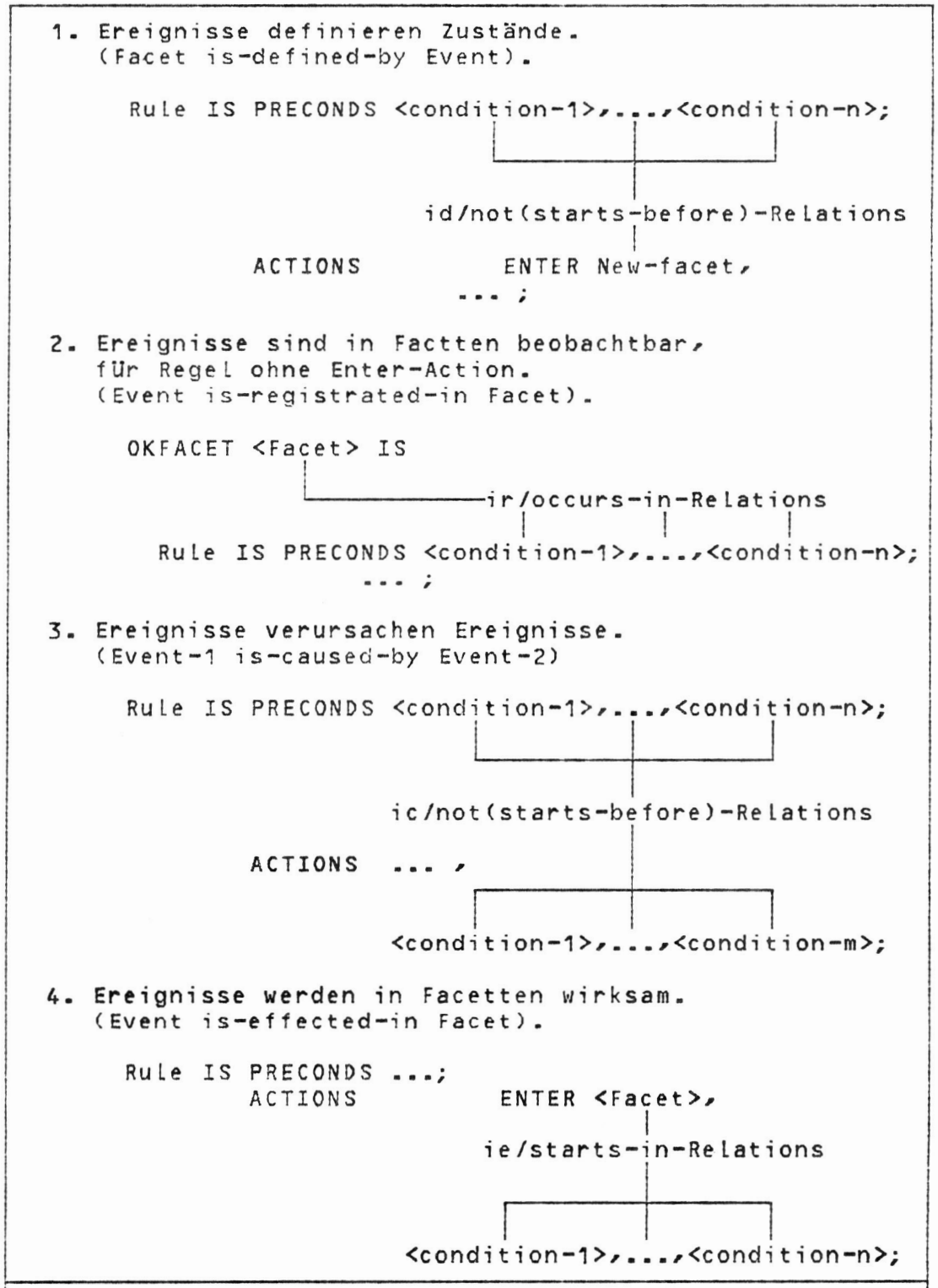


Abb. I.3: Darstellung der zeitlichen Kausalitäten zwischen definierbaren Ereignissen.

Dieser Bedingung ist in der Menge der definierbaren Relationen zwischen diesen Ereignissen Rechnung getragen. Ist in einer Relation ein IN-Ereignis beteiligt, dann ist nur eine Teilmenge der oben genannten Relationenmengen zugelassen; siehe in [1].

I.1.3 Modellanalyse

Eine konkrete Verhaltensweise eines Modells ist durch eine Folge von wechselnden Werten an den Eingangsgrößen und den daraus resultierenden Ausgangswerten charakterisiert. Die entsprechenden Zusammenhänge zwischen Eingangswerten, innerem Zustand und Ausgangswerten sind in der Modellspezifikation (MSP) innerhalb von Regeln manifestiert. Mit innerem Zustand ist die Belegung von Variablen und die aktuelle Facette assoziiert.
Es stellt sich die Frage, ob die lokal beschriebenen Kausalitäten entlang der zeitlichen Abfolge bestimmter Eingangsverhalten immer noch konsistent zueinander sind, d.h. ist das resultierende Constraint-Netz (CN) von zeitlichen Relationen als ganzes noch konsistent? Dies kann von einer Constraint-Propagierung gemäß [2] ermittelt werden.
Wie aus einer MSP ein CN entsteht, wird im folgenden skizziert:

1) Parsen der MSP und Umwandlung in eine interne Repräsentation, das Modellspezifikationsnetz (MSN);
2) Erstellen eines Modellgraphen (MG) auf der Basis des MSN;
3) Auswählen einer konkreten Verhaltensweise aus dem MG und Darstellung in einer Zwischenrepräsentation;
4) Erstellen des CN aus dieser Zwischenrepräsentation durch Generierung sogenannter Eventstrukturen.

I.1.3.1 Parsen

Eine Modellspezifikation ist in Hiqual als Modelltyp definiert. Ein Modelltyp beschreibt die Menge aller Modelle, die als Instanzen aus dem Modelltyp entstehen können. Modellinstanzen können sich z.B. durch unterschiedliche Attributbelegungen voneinander unterscheiden. Die weitere Modellanalyse bezieht sich ausschließlich auf Modellinstanzen. Die Instanziierung eines Modells geschieht mittels Parsen einer MSP, wobei der Name einer Modellinstanz (MI) vom Benutzer festgelegt werden muß. Eine MI wird intern als MSN gespeichert.
Die Knoten eines MSN sind Ereignisse (Condtions & Facetten), die durch Regeln definiert worden sind. Bestehende Relationen

zwischen diesen Knoten bilden die Kanten.

I.1.3.2 Der Modellgraph

Auf der Basis des MSN soll für eine MI deren mögliche Verhaltensweisen abgeleitet werden.
Was heißt in diesem Sinne Verhalten?
Das Verhalten eines physikalisch-technischen Systems ist im allg. von zwei Faktoren abhängig:

- Von den auf das System einwirkenden äußeren physikalischen Größen und
- vom inneren Zustand.

Sichtbar wird dieses Verhalten durch die Reaktion des Systems auf die beiden genannten Faktoren. In der Spezifikation von Hiqual-Modellen können von außen einwirkende Größen über Inports auf ein System einwirken. Facetten und lokale Variablen repräsentieren den inneren Zustand eines Systems. Eine nach außen sichtbare Reaktion ist nur über die Outports möglich.
Die interessanten Fragen für eine Modellanalyse sind demnach zunächst:

- Welche Werte müssen die definierten äußeren Größen annehmen, um eine neue (innere und äußere) Reaktion auszulösen, in Abhängigkeit vom inneren Zustand und in
- welcher sequentiellen Folge können sie auftreten?

Diese beiden Fragen sollen als Verhaltensanalyse in Form eines Modellgraphen beantwortet werden.
Ausgehend von der Initialfacette und den dort definierten Regeln wird beim Erstellen des Graphen untersucht, welche Werte in welcher Reihenfolge an den Inports anliegen können und welche Aktionen sie aufgrund der existierenden Regeln auslösen.
Die dabei beobachteten Ereignisse werden im MG als syntaktische Objekte notiert, analog zu der in [1] vorgestellten Darstellung von ES-Expressions in der Form:

<caf>!<inst-name>!<rule-name>!<type>!<id>!<index> ,

wobei diese Objekte innerhalb der Implementierung als **Cf-Intervalle** benannt sind. Diese Bezeichnung wird auch hier verwendet, um sie von den später eingeführten Ereignisarten zu unterscheiden. Mit Ausnahme von <id> sind alle Komponenten des Cf-Intervall-Symbols aus [1] bekannt bzw. werden im III. Kapitel näher erläutert.

I.1.3.3 Cf-Instance-Pfade

Der MG ist eine Kompaktdarstellung aller möglichen Verhalten eines Modells. Mit Hilfe des **Inspectors** [4] ist es möglich,

sich eine konkrete Verhaltensweise mit den Daten aus dem MG zusammenzustellen. Insbesondere lassen sich zykliche Verhaltensweisen mehrfach in Kombination mit anderen Sequenzen aufrollen, um beispielsweise deren Konsistenz verifizieren zu können.
Das Ergebnis der Auswahl ist ein sequentialisierter Verhaltenspfad in einer neuen Zwischenrepräsentation. Ereignisse sind jetzt als sogenannte **Cf-Instance-Intervalle** dargestellt. Jedem cf-instance-Intervall ist eindeutig ein Cf-Intervall aus dem MG zugeordnet, aber einem selektierten Cf-Intervall können mehrere Cf-Instance-Intervalle entsprechen. Der entstandene Pfad wird als **Cf-Instance-Pfad** bezeichnet. Die syntaktische Struktur der Cf-Instance-Intervalle ist analog zu den Cf-Intervallen.
Bis zu diesem Stadium der Modellanalyse kamen noch keine zeitlichen Relationen zum Tragen. Sie werden jetzt berücksichtigt und ergänzt.
Alle in der MSP bekannten Relationen sind auch dem MG bekannt und sind deshalb in den Cf-Instance-Pfad übertragbar. Aus der sequentiellen Anordnung der Ereignisse lassen sich noch weitere Relationen ableiten.
Innerhalb der MSP sind Relationen zwischen den Ereignissen im Precondition- und Action-Teil einer Regel definierbar. Aus einem Cf-Instance-Pfad lassen sich zwei weitere Relationenarten bestimmen:

- Die Meets-Relationen zwischen aufeinanderfolgenden Ereignissen (Cf-Instance-Intervallen) für das gleiche PVA (Port, Variable, Attribut) bzw. zwischen aufeinanderfolgenden Facetten;
- Relationen zwischen Ereignissen aus dem Precondition-Teil von Regeln.
  Z.B. steht die Relationenmenge {=,s,si} für den gleichzeitigen Anfang zweier Ereignisse usw.

Das Ergebnis bis zu diesem Teil der Analyse ist die Vorstufe des gewünschten Constraint-Netzes.

I.1.3.4 Eventstrukturen

In diesem Stadium ist es notwendig, den Ereignisbegriff neu zu definieren. Bis jetzt waren Ereignisse Conditions aus Regeln oder Facettenbezeichner.
Aber wie sind syntaktisch gleiche Conditions in verschiedenen konsekutiven Regeln zu interpretieren? Wie sind die entsprechenden Relationen zu interpretieren, wenn mehrere Regeln nacheinander feuern, in denen die gleichen Conditions vorkommen?
Um in diesem Fall eine eindeutige Interpretation aufrecht zu erhalten, wurden die Ereignis- bzw. Eventstrukturen eingeführt.

Streng genommen ist ein Ereignis das Eintreten eines Wertes oder Zustandes und die Beobachtung seiner Dauer. Dem soll mit Hilfe der Eventstrukturen Rechnung getragen werden. Treten in einem Cf-Instance-Pfad nacheinander Cf-Instance-Intervalle auf, die ein syntaktisch gleiches Ereignis (den <caf>-Teil des Intervallsymbols) in verschiedenen konsekutiven Regeln beschreiben, so werden diese Vorkommen in einer Eventstruktur zusammengefaßt. Die Relationen aus den Cf-Instance-Intervallen sind nicht direkt in die Eventstrukturen übertragbar. Die neuen Relationen entstehen durch lokale Propagation der Relationen der Cf-Instance-Intervalle, die das Event repräsentieren. Auf diese Weise entsteht ein vollkommen neues CN auf der Basis von Eventstrukturen mit neuen Zeitrelationen, die sich auf Events stützen.

Zur Betrachtung des neu entstandenen CN steht der **Observer** zur Verfügung. Für diese Ausgabe müssen die Eventstrukturen und die etablierten zeitlichen Relationen graphisch aufbereitet werden. Eventstrukturen sind wieder als qualitative zeitliche Intervalle zu interpretieren, wobei deren absolute Dauer nicht von Bedeutung ist. Bestehende Relationen zwischen zwei Eventstrukturen sagen lediglich aus, wie sich die beiden Ereignisse zeitlich zueinander verhalten, z.B. E1 ist vor (<) E2 oder E1 überlappt (o) E2. Die bekannten Relationen werden dazu benutzt, um die Intervalle relativ zueinander zu orientieren und entsprechend auszugeben, d.h. nicht die Relationen selbst, sondern die daraus resultierende zeitliche Orientierung der Intervalle zueinander steht als visuelle Information zur Verfügung. Zusätzlich ist es natürlich möglich, sich die explizit gültigen Relationen für bestimmte Ereignisse zeigen zu lassen.

I.2 Aggregate

Als kleinste unabhängige Einheit zur Beschreibung von Systemen wurden Modelle vorgestellt. Es ist wünschenswert, ein komplexes System, das zunächst als ein Modell spezifiziert wurde, in mehrere Module aufzuspalten. Jedes Modul soll als unabhängiges Objekt modellierbar und später über vordefinierte Schnittstellen mit anderen Modulen koppelbar sein.
Solche Module sind wiederum als Hiqual-Modelle definierbar, so daß das gleiche System sich als ein einzelnes Modell oder als eine Zusammenschaltung von mehreren Modellen beschreiben läßt, wobei jede dieser Darstellungen unabhängig voneinander ist.
Ein Hiqual-Aggregat erlaubt es, zwei Beschreibungsebenen als eine Hierarchie darzustellen.
Innerhalb eines Aggregats lassen sich sowohl vertikale als auch horizontale Beziehungen beschreiben.
Horizontale Beziehungen sind Verbindungen zwischen den Modellen einer Hierarchieebene. Strukturelle Zusammenhänge zwischen

den Ebenen sind vertikale Beziehungen .
Die Wurzel eines Aggregats, die oberste Hierarchiestufe, ist ein einzelnes Modell. Es umfaßt bereits die komplette Systemspezifikation. Als Beispiel sei das oben definierte RS_FF genannt. Ein RS_FF kann auch als Zusammenschaltung zweier NOR-Gatter beschrieben werden, die die zweite Hierarchieebene darstellen. Die vertikalen Beziehungen legen fest, daß der R- und S-Eingang des RS_FF jeweils einem Eingang des NOR-Gatters entspricht; das Gleiche gilt für die Ausgänge.
Die Beschreibung eines NOR-Gatters kann verfeinert werden durch die Zusammenschaltung zweier Transistoren. Der Wert Low auf der Stufe der Gatter entspricht einem Spannungsbereich von 0 bis 2.5 Volt; High wird auf einen Spannungsbereich von 2.6 bis 5 Volt auf der Ebene der Transistoren abgebildet.
Das Aggregatprinzip kann rekursiv angewendet werden, es entstehen dann komplexe Aggregate.
Im Beispiel heißt das:
Das RS_FF kann als Zusammenschaltung eines NOR-Gatters und zweier Transistoren (die das zweite NOR-Gatter im Detail repräsentieren) beschrieben werden.

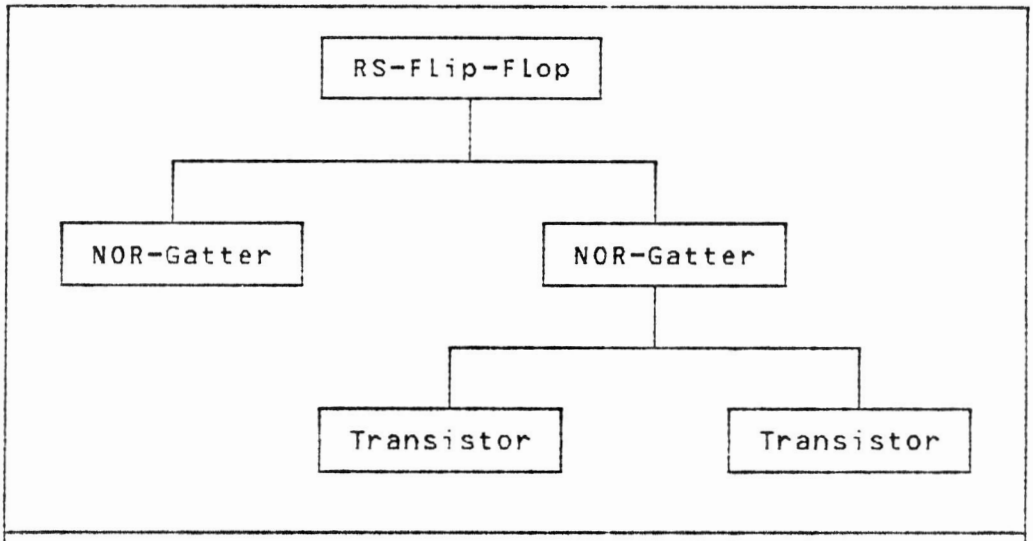


Abb. I.4: Struktur eines Aggregates für ein RS-FF, das aus einer Zusammenschaltung von zwei NOR-Gattern und zweier Transistoren hervorgeht.

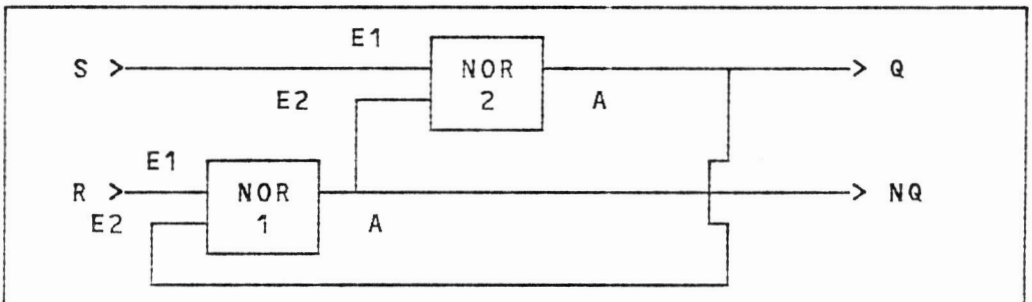


Abb. I.5: Schaltplan eines RS_FF bestehend aus zwei NOR-Gattern.

```
TYPE R_S_AGG IS AGGREGATION (R_S_F_F, NOR_1, NOR_2)

ROOT R_S_F_F;

MODELS R_S_F_F IS RS_FF,
       NOR_1, NOR_2 ARE NOR_GATE;

CONNECTIONS NOR_1@A WITH NOR_2@E2,
            NOR_2@A WITH NOR_1@E2;

REPRESENTATIONS Rep1 IS DOMAIN BINARY;
       DOMAIN IS BINARY;
       RANGE NOR_GATE@BINARY, NOR_GATE@BINARY;
       GRAPH high IS (high,high),
             low IS (low,low);

STRUCTURE_MAP R IS NOR_1@E1,
              S IS NOR_2@E1,
              Q Rep1 (NOR_1@A,NOR_2@A),
              NQ Rep1 (NOR_2@A,NOR_1@A);
```

Abb. I.6: Aggregatdefinition in Hiqual für das in Abb. I.5 dargestellte RS_FF. Die REPRESENTATIONS- und STRUCTURE_MAP-Klauseln sind die vertikale Verbindung, die CONNNECTIONS-Klausel beschreibt die horizontale Verbindung innerhalb der Ebene der NOR-Gatter.

I.2.1 Aggregatanalyse

Existiert die Beschreibung eines Systems in Form eines Aggregats, dann stellt sich die Frage, ob die einzelnen Stufen zueinander konsistent sind bzw. ob sich die Wurzel des Aggregats genauso verhält (in einer abstrakteren Form) wie die Kopplung der darunter liegenden Modelle.
Die Aggregatanalyse soll über die festgelegten horizontalen und vertikalen Beziehungen diese Bedingung unter Berücksichtigung der zeitlichen Relationen überprüfen.
Nach dem gleichen Prinzip der Modellinstanziierung entsteht mittels Parsen der Aggregattypdefinition eine Aggregatinstanz, in der die formalen Parameter durch die aktuellen Parameter, also Modellinstanzen, ersetzt worden sind. Die Aggregatinstanz steht anschließend der Aggregatanalyse zur Verfügung.
Die Aggregatanalyse muß versuchen, das Verhalten des Systems, das durch den MG der Wurzel des Aggregates festgelegt ist, in

den MG der anderen Modelle wiederzufinden. Jede Verhaltensweise der Wurzel muß ihre Enstprechung in den Verhaltensweisen der anderen Modelle finden.
Aus Komplexitätsgründen kann dies nicht für die kompletten MG aller beteiligten Modelle geschehen, sondern es muß aus dem MG der Aggregatwurzel eine konkrete Verhaltensweise des Systems zusammengestellt und nach dem oben beschriebenen Kriterium analysiert werden.
Wenn aus den MG der von der Wurzel gesteuerten Modelle analoge Cf-Intervall-Pfade identifizierbar sind, werden diese Pfade in ihrer Cf-Instance-Darstellung zurückgeliefert; sie bilden das Ergebnis einer erfolgreichen Aggregatanalyse.
An eine erfolgreiche Analyse muß die Eventanalyse anschließen, um die zeitlichen Relationen innerhalb des Aggregates auf ihre Konsistenz zu überprüfen. Interessant sind dabei vor allem die modellübergreifenden Auswirkungen der definierten Vertikal- und Horizontalbeziehungen.
Bei konsistenter Relationenmenge läßt sich das resultierende CN wieder mit dem Observer ausgeben und inspizieren.

I.3 Komplexe Aggregate

Aggregate bieten die Möglichkeit, ein durch ein einzelnes Modell beschriebenes System in zwei Hierarchieebenen gleichzeitig zu repräsentieren.
Die nächste logische Abstraktion basiert auf der Idee, daß jede MI eines Aggregats (ungleich der Wurzel) selbst wieder die Wurzel eines eigenständigen Aggregats sein kann. Das Resultat ist nicht mehr ein Aggregat aus MI, sondern ein Aggregat aus Aggregatinstanzen. Diese neue Abstraktionsebene einer Systemspezifikation in Hiqual sind die komplexen Aggregate .
Komplexe Aggregate bieten die Möglichkeit, ein durch ein einzelnes Modell beschriebenes System in beliebig vielen Hierarchieebenen gleichzeitig zu repräsentieren und zu analysieren.
Es ist klar, daß die Analyse von komplexen Aggregaten analog zu der Analyse von Aggregaten abläuft.

Damit sind alle Elemente der Hiqual-Implementierung vorgestellt. Die nachfolgenden Kapitel enthalten detailiertere Information.

II Die Parser in Hiqual

II.0 Einführung

Modell-, Aggregat-, Funktions- oder Type-Spezifikationen werden in der Sprache Hiqual formuliert. Sie werden zum Teil mittels des Standardeditors in die Hiqualdatenbasis geschrieben, von dort gelesen, geparst, in eine interne Struktur übersetzt, die dann der Analyse übergeben wird (Modelle und Aggregate) oder sie werden als Funktionsparameter eingegeben, zerteilt und in Dateien in der Hiqualdatenbasis abgelegt, um dann später von dort in Modellspezifikationen importiert zu werden (Funktionen und Typen).

II.1 Modell-Parser

II.1.1 Syntax für Hiqualmodelle

II.1.1.1 Verwendete Notation

Alle Syntaxspezifikationen werden in einer erweiterten Backus-Naur-Form angegeben.
Zur Standardform wurden die Produktionen {...}* und {...}+ dazugenommen, um die beliebige und die nichtleere Wiederholung zu beschreiben.
Die Bedeutung der verwendeten Metasymbole:

```
::=   ersetze linke Seite durch rechte Seite,
|     Auswahl zwischen zwei Möglichkeiten,
[...] Optionalklammern,
(...) Prioritätsklammern,
{...}* Wiederholungsanweisung,
{...}+ nichtleere Wiederholung.
```

Nichtterminale Symbole werden klein geschrieben, Wortsymbole werden groß geschrieben, Zeichensymbole werden in Anführungs-

zeichen eingefaßt, Namen und Zahlen werden als <NAME> resp. <NUMBER> repräsentiert.
Diese Notation gilt für alle nachfolgenden Syntaxspezifikationen.

II.1.1.2 Grammatik

```
model ::=
        TYPE <NAME> IS MODEL
        [ types ] [ ports ] [ vars ] [ attributes ]
        [ functions ] [ okfacets ] [ exfacets ]
        init-clause
        END <NAME> ";"

types ::=
        TYPES { type ";" } *

type ::= <NAME> { "," <NAME> } * ( IS | ARE )
        ( record ** | simple-type | [ <NAME> ] IMPORTED )

simple-type ::=
        ("{"( <NAME> | <NUMBER> ) [ IS_INST ]
        {"," ( <NAME> | <NUMBER> ) [ IS_INST ] } * "}" |
        ("(" ( <NAME> | <NUMBER>) [ IS_INST ]
        { ";" (<NAME> | <NUMBER>) [ IS_INST ] } * ")"

record ::=
        RECORD component { "," component } * END

component ::
        <NAME> IS simple-type

vars ::= VARIABLES { var-def ";" } +

var-def ::=
        var-id { "," var-id } * ( IS | ARE ) <NAME>

var-id ::=
        <NAME> [ BOUNDARIES ** ]
        [ INITIALLY (<NAME> | <NUMBER>) |
        "(" <NAME> | <NUMBER>
        { ";" <NAME> | <NUMBER> } * ")" ]

boundaries ::=
        FROM index TO index

index ::=
        <NAME> | <NUMBER>

ports ::=
```

```
        PORTS [ in-ports ] [ out-ports ]

in-ports ::=
        IN { port ";" } *

out-ports ::=
        OUT { port ";" } *

port ::= <NAME> [ boundaries ** ]
        { "," <NAME> [ boundaries ** ] } *
        IS | ARE <NAME>

attributes ::=
        ATTRIBUTES { attribute-def ";" } +
        [ RESTRICTED_BY comparison { "," comparison } * ";" ]

attribute-def ::=
        attribute-id { "," attribute-id } *
        IS | ARE <NAME>

attribute-id ::=
        <NAME> [ boundaries ** ]
        [ DEFAULT ( <NAME> | <NUMBER> ) |
        "(" <NAME> | <NUMBER>
        { ";" <NAME> | <NUMBER> } * ")" ]

functions ::=
        FUNCTIONS { funct ";" } +

funct ::=
        <NAME> IS
        DOMAIN <NAME> { "," <NAME> } ";" *
        RANGE <NAME> ";"
        GRAPH map-list |
        [ <NAME> ] IMPORTED

map-list ::=
        map { "," map } +

map ::=  domain-interval IS <NAME> | <NUMBER>

domain-interval ::=
        single-domain-interval |
        "("  single-domain-interval  { "," single-domain-
        interval } * ")"

single-domain-interval ::=
        ( <NAME> | <NUMBER> ) |
        ( <NAME> | <NUMBER> "." "." <NAME> | <NUMBER> ** )

okfacets ::=
        { OKFACET okfacet } *
```

```
exfacets ::=
        { EXFACET exfacet } *

okfacet ::=
        <NAME> [ IS_INST ] IS { rule } +

exfacet ::=
        <NAME> [ IS_INST ] IS { rule } +

rule ::= <NAME> IS
        PRECONDS precond { "," precond } * ";"
        ACTIONS action-list-with-enter | action-list-without-
        enter
        [ RELATIONS relation { "," relation } * ";" ]

action-list-with-enter ::=
        ENTER <NAME> { "," action } * ";"

action-list-without-enter ::=
        action { "," action } *

precond ::=
        named-precond | precond-1

precond-1 ::=
        ( ACTIVE <NAME> ** | comparison )

named-precond ::=
        "(" precond-1 {"," precond-1 } * label ")"

label ::=
        FACET | <NAME>

action ::=
        comparison | "(" comparison { "," comparison } * ")"

relation ::=
        label IS temp-spec {"," temp-spec } * WITH label

temp-spec ::=
        =  |  < | > | m | mi | f | fi | s | si | d | di | o |
        oi

comparison ::=
        quantity ( IS qvalue-set ) |
        ( comparator compare-with )

quantity ::=
        (<NAME> [ "." index ** ] [ "." <NAME> ** ] ) |
        <NUMBER>

comparator ::=
        = | \= | > | < | >= | <=
```

```
compare-with ::=
          <NUMBER> |
          (  [ <dollar-sign> ] <NAME> [ "(" quantity {"," quan-
          tity } * ")" |
          ( "." index ] "." name ] "(" quantity {"," quantity
          } * ")" ** ) ]

qvalue-set ::=
          "{" <NAME> | <NUMBER> {"," <NAME> | <NUMBER> } *  "}"
          |
          ( <NAME> | <NUMBER> "." "." <NAME> | <NUMBER> ** )

init-clause ::= INITIALLY <NAME> ";"
```

Die mit ** versehenen Konstrukte sollten nicht benutzt werden. Sie haben entweder kein Lispäquivalent oder sie werden nicht in die Auswertung einbezogen.

II.1.1.3 Aufsetzsymbole

Der Hiqualmodellparser verfügt über folgende Aufsetzsymbole, die es ihm erlauben sich nach einem Syntaxfehler zu erholen und in der Überprüfung der syntaktischen Korrektheit weiterzufahren :

TYPES PORTS VARIABLES ATTRIBUTES FUNCTIONS OKFACET EXFACET INITIALLY.

Im Fehlerfall liest die Fehlerfunktion weiter bis eines dieser Aufsetzsymbole auftritt und übergibt dann die Kontrolle wieder an den Parser.
Wenn ein syntaktischer Fehler auftritt, wird keine interne Repräsentation erzeugt und die Überprüfung der statischen Semantik abgebrochen.

II.1.1.4 Fehlermeldungen

Im syntaktischen Fehlerfall wird die Position (Zeile und Spalte) des Fehlers gemeldet sowie das Konstrukt, in dem der Fehler aufgetreten ist.
Im Fall eines semantischen Fehlers erscheint eine präzise Meldung über die beteiligten Elemente (Ports, Variable, Wertebereiche etc.). Alle Fehlermeldungen werden nach Standard-

output geschickt.

II.1.2 Statische Semantik

II.1.2.1 Überprüfte Elemente

Die Überprüfung der statischen Semantik geschieht, um Laufzeitfehler beim Parsen und bei der Berechnung des Modellgraphen zu verhindern.
Im Hiqualmodellparser beschränkt sie sich ausschließlich auf die Überprüfung von Definition und Definfitionsbereich.
Folgende Tests werden durchgeführt :

- Im Types-Konstrukt importierte Typen müssen in der Hiqual-Datenbasis vorhanden sein.
- Typen, die in den Ports-, Variables- oder Attributekonstrukten referiert werden, müssen in der Typesklausel definiert worden sein.
- Das Gleiche gilt für Typen, die in der Domain- oder Graphklausel des Funct-Konstrukts benutzt werden.
- In der Graphklausel des Funct-Konstrukts wird überprüft, ob der Graph vollständig und eindeutig ist, d.h. ob die angegeben Domaintupel das Domain vollständig abdecken und ob jedes Tupel nur einmal vorkommt.

- In den Comparison-Konstrukten der Preconditions der Regeln muß folgendes gelten :
  - die linke Seite muß ein In-port, Variable oder Attribut sein;
  - die rechte Seite muß entweder ein In-port, eine Variable, ein Attribut, eine Funktion oder ein Wert aus dem Wertebereich der linken Seite sein.
  - Ist die rechte Seite eine Funktion, muß die Anzahl der Parameter mit der Definition der Funktion übereinstimmen und die Parameter müssen Variable, In-ports, Attribute oder qualitative Werte aus den angegebenen Domains sein.

- In den Actions von Regeln muß gelten :
  - Enter-actions müssen zu definierten Faceten führen, in den Zuweisungen müssen die linken Seiten Outports oder Variable sein, die rechten Seiten müssen In-ports, Attribute, Variable, Funktionen oder qualitative Werte aus dem Wertebereich der linken Seite sein.
  - FÜr die Parameter in Funktionsaufrufen gelten die gleichen Regeln wie oben.
  - Relationen in Regeln dürfen nur zwischen Labels, die

in der selben Regel definiert sind, angegeben werden.

- Die Init-Klausel muß auf eine definierte Facette verweisen.
- In allen Konstrukten wird überprüft, ob nichts doppelt definiert wird, d.h. Regeln, Facetten, Ports etc. müssen eindeutige Namen haben.

II.1.2.2 Auswertung

Die statische Semantik wird während des Parsevorgangs im selben Durchlauf wie die Syntax überprüft.
Ein auftretender semantischer Fehler führt , nach einer entsprechenden Fehlermeldung, zum Abbruch der Semantiküberprüfung.

II.1.3 Aufruf des Parsers

Der Modellparser kann über das StandardmenÜ im Hiqual-Lisp-Interaction-window aufgerufen werden (Option Modell Instanz erzeugen) oder über die Funktion /hiqual-model-instance/ mit einem String als Parameter. Der String muß folgender Syntax genügen :

```
string ::=
        <MODEL-INSTANCE-NAME> IS <MODEL-TYPE-NAME>
        [ WITH attribute-def-list ]

attribute-def-list ::=
        <ATTR-NAME> "=" <QVALUE>
        {"," <ATTR-NAME> "=" <QVALUE> } *
```

Der Aufruf über die Funktion /hiqual-model-instance/ eignet sich für den Batchbetrieb .
In beiden Fällen werden Syntax und Sematik überprüft und es wird eine Instanz des Modelltyps erzeugt.

II.1.4 Funktionsweise und benutzte Module

Der Modellparser ist ein top-down LL(1)-Parser, der, in einem Durchgang, Hiqualprogramme nach der beschriebenen Syntax, aus-

gehend vom Startsymbol Model, aus einer Datei parst.
Dabei werden Syntax und Semantik gleichzeitig überprüft; parallel dazu wird die im Hiqualprogramm enthaltene Information in eine interne Lispdarstellung umgesetzt und die zeitlichen Relationen werden eingefügt.
Am Modellparser beteiligt sind die Module MODEL-INSTANCE, MODEL-TYPE-PARSER, MODEL-ANKNÜPFUNGEN, SCANNER und STACK.

II.2 Aggregat-Parser

II.2.1 Syntax für Hiqualaggregate

II.2.1.1 Grammatik

```
aggregate ::=
        TYPE <NAME> IS AGGREGATION "(" name-list ")"
        ROOT <NAME> ";"
        MODELS model-paramater-list ";"
        [ WITH attribute-restrictions-list ";" ]
        [ CONNECTIONS connection-list ";" ]
        [ REPRESENTATIONS representation-list ";"]
        STRUCTURE-MAP structure-mapping-list | INHERIT ** ";"
        [ TEMPORAL-MAP temporal-mapping-list ";" ]
        [ EPISODES episode-list ";" ** ]

name-list ::=
        <NAME> { "," <NAME> } *

model-parameter-list ::=
        model-parameter { "," model-parameter } *

model-parameter ::=
        name-list IS | ARE <NAME>

attribute-restrictions-list ::=
        attribute-restriction { "," attribute-restriction } *

attribute-restriction ::=
        prefixed-attribute   comp-op   prefixed-attribute-or-
        value

prefixed-attribute ::=
        model-parameter-name prefix-sign attribute-name

model-parameter-name ::=
        <NAME>

prefix-sign ::=
        "@"

attribute-name ::=
        <NAME>
```

```
prefixed-attribute-or-value ::=
        prefixed-attribute | value

value ::=
        <NUMBER> | <NAME>

connection-list ::=
        connection { "," connection } *

connection ::=
        prefixed-in-port WITH prefixed-out-port

prefixed-in-port ::=
        model-parameter-name prefix-sign port-bez

port-bez ::=
        <NAME>

prefixed-out-port ::=
        model-parameter-name prefix-sign port-bez

representation-list ::=
        representation *

representaion ::=
        <NAME> IS
        DOMAIN <NAME> ";"
        RANGE prefixed-type-list ";"
        GRAPH rep-graph ";"

prefixed-type-list ::=
        prefixed-type { "," prefixed-type } *

prefixed-type ::=
        model-parameter-name prefix-sign type-id

type-id ::=
        <NAME>

rep-graph ::=
        rep-graph-el { "," rep-graph-el } *

rep-graph-el ::=
        value IS value-list

value-list ::=
        value { "," value } *

structure-mapping-list ::=
        structure-mapping { "," structure-mapping } *

structure-mapping ::=
        port-bez ( IS prefixed-port ) | ( representation-name
```

```
        prefixed-port-list )

representation-name ::=
        <NAME>

prefixed-port-list ::=
        prefixed-port { "," prefixed-port } *

temporal-mapping-list ::=
        temporal-mapping { "," temporal-mapping } *

temporal-mapping ::=
        state-or-action-ref-list  IS  | ARE start-or-end-spec
        WITH prefixed-state-or-action-ref
        [ IS end-spec WITH prefixed-state-or-action-ref ]

state-or-action-ref-list ::=
        state-or-action-ref { "," state-or-action-ref } *

state-or-action-ref ::=
        comp | facet-port-or-label

comp ::= "(" port-or-variable comp-op value ")"

comp-op ::=
        > | < | = | => | =< | \=

prefixed-state-or-action-ref ::=
        prefixed-comparison | prefixed-facet-port-or-label

prefixed-facet-port-or-label ::=
        model-parameter-name prefix-sign facet-port-or-label

facet-port-or-label ::=
        <NAME>

prefixed-comparison ::=
        "(" prefixed-port-or-variable comp-op value ")"

model-parameter-name ::=
        <NAME>

start-or-end-spec ::=
        SB | MI | = | SE | M

end-spec ::=
        SE | M

prefixed-port-or-variable ::=
        model-parameter-name prefix-sign port-or-variable

port-or-variable ::=
        <NAME>
```

```
episode-list ::=
        episode { "," episode } *

episode ::=
        <NAME> IS ( episode-macro | episode-expr )

episode-macro ::=
        MACRO <NAME> "(" name-list ")"

episode-expr ::=
        root { IS temp-spec-list WITH subtree-expr } +

root ::= prefixed-state-or-action-ref

subtree-expr ::=
        "(" episode-expr ")" | leaf

leaf ::= prefixed-state-or-action-ref

temp-spec-list ::=
        temporal-spec { "," temporal-spec } *

temporal-spec ::=
        < | > | M | MI | S | SI | D | DI | O | OI | F | FI |
        =
```

Die mit ** versehenen Konstrukte sollten nicht benutzt werden. Sie haben kein Lispäquivalent und werden nicht in die Auswertung mit einbezogen.

II.2.1.2 Aufsetzsymbole

Im Hiqualaggregatparser wird das Semikolon als Aufsetzsymbol benutzt.
Wenn ein Fehler auftritt, liest die Fehlerfunktion weiter bis zum nächsten ";" und gibt dann die Kontrolle an den Parser zurück.
Nachdem ein syntaktischer Fehler aufgetreten ist, wird keine interne Struktur mehr aufgebaut und keine statische Semantik mehr kontrolliert.

II.2.1.3 Fehlermeldungen

Im syntaktischen Fehlerfall wird die Position des Fehlers (Zeile und Spalte) und das Konstrukt, in dem der Fehler aufgetreten ist ausgedruckt.
Wenn ein semantischer Fehler auftritt, erscheint eine präzise Meldung über die beteiligten Elemente (Ports, Variable, Wertebereiche, etc.) .
Alle Fehlermeldungen werden nach Standard-output geschrieben.

II.2.2 Statische Semantik

II.2.2.1 Überprüfte Elemente

Die statische Semantik wird überprüft, um Laufzeitfehler beim Parsen und bei der Auswertung des Aggregats zu verhindern.
Folgende Anforderungen werden überprüft :

- Alle Modellinstanzen, mit denen eine Aggregatinstanz gebildet wird, müssen definiert und von dem Typ sein, der in der Aggregattypedefinition angegeben wurde.
- Die angegebenen Attributrestrictionen müssen auf die aktuellen Werte der Modellinstanzen zutreffen.
- Connections und Structure-maps ohne Representation dürfen nur zwischen Ports mit gleichen Wertebereichen angegeben werden.
- Die Definition der Representations muß vollständig und eindeutig sein.
- Die Ports, Attribute, Facetten oder Labels, die in der Temporal_Map angegeben werden, müssen existieren.
- Alle Ports der Blätter des Aggregates müssen genau einmal in einer Connection oder der rechten Seite einer Structure-map auftreten.
- Alle Ports der Wurzel müßen genau einmal in der linken Seite einer Structure-map auftreten.

II.2.2.2 Auswertung

Die statische Semantik für Aggregatinstanzen wird in einem separaten Pass nach der Syntax überprüft.
Ein auftretender Fehler führt zum Abbruch der Überprüfung und einer entsprechenden Fehlermeldung.

II.2.3 Aufruf des Parsers

Der Hiqualaggregatparser kann im Hiqual-Lisp-Interaction-Window über das StandardmenÜ aufgerufen werden (Option Aggregat-Instanz erzeugen). Die Modellinstanzen, mit denen die Aggregatinstanz gebildet wird, werden über Menu eingegeben.
Der Aufruf kann auch über die Funktion /hiqual-aggregate-instance/ mit einem String als Parameter erfolgen. Der String muß folgender Syntax genügen :

```
string ::=
        <AGGREGATE-INSTANCE-NAME> IS <AGGREGATE-TYPE-NAME>
        WITH model-instance-list

model-instance-list ::=
        <MODEL-INSTANCE-NAME> { "," <MODEL-INSTANCE-NAME> } *
```

Der Aufruf über /hiqual-aggregate-instance/ eignet sich für den Batchbetrieb.
In beiden Aufrufmodi werden Syntax und Semantik überprüft und es wird eine interne Struktur erzeugt.

II.2.4 Funktionsweise und benutzte Module

Der Hiqualaggregatparser ist ein Dreipassparser. Im ersten Pass wird die Syntax überprüft und das Gerüst für die interne Struktur erzeugt. Im zweiten Durchgang werden die formalen Parameter durch die Namen der Modellinstanzen ersetzt. Erst im dritten Anlauf wird die statische Semantik kontrolliert.
Die SyntaxÜberprüfung wird durch einen top-down LL(1)-Parser ausgeführt, der die Aggregatbeschreibungen gemäß der obigen Syntax, ausgehend vom Startsymbol Aggregate, aus einer Datei

liest.
Am Aggregatparser beteiligt sind die Module AGGREGATE-TYPE-PARSER, AGGREGATE-ANKNUEPFUNGEN, AGGREGATE-INSTANCE, SCANNER und STACK.

II.3 Funktions-Parser

II.3.1 Syntax für Hiqual Funktionen

II.3.1.1 Grammatik

Die Syntax für Funktionen, die in die Hiqualdatenbasis eingefügt werden sollen, ist die gleiche, die für Funktionen innerhalb von Modelldefinitionen gilt.
An den Schluß der Definition kann noch ein Kommentar angefügt werden.

II.3.1.2 Aufsetzsymbole

Dieser Parser kennt keine Aufsetzsymbole. Im Fehlerfall wird bis zum Symbol "END_OF_FILE" gelesen und eine entsprechende Meldung ausgegeben.

II.3.2 Statische Semantik

Die statische Semantik wird überprüft, wenn die Funktion mit dem Import-Konstrukt aus der Datenbasis in eine Modelldefinition eingefügt wird.

II.3.3 Aufruf des Parsers

Der Parser wird im Lisp-top-level über die Funktion /hiqual-function-define/ mit einem String als Parameter, welcher der Grammatik, ausgehend vom Symbol Funct entspricht, aufgerufen.

II.3.4 Funktionsweise und benutzte Module

Der String, der als Parameter eingegeben wurde, wird geparst, auf syntaktische Korrektheit geprüft und die darin enthaltene Information in die Datei
>HIQUAL>DATA>FUNCTIONS><FUNKTIONSNAME>.HIQUAL geschrieben. Sie kann dort über den Standardeditor gelesen und verändert werden.
Beteiligt sind die Module FUNCTION-DEFINE, STACK und SCANNER.

II.4 Typ-Parser

II.4.1 Syntax für Hiqual Typen

II.4.1.1 Grammatik

In die Hiqualdatenbasis können nur Typen eingefügt werden, die dem Konstrukt simple-type aus der Grammatik für Hiqualmodelle entsprechen.

type ::= <NAME> IS simple-type

An den Schluß der Definition kann noch ein Kommentar angehängt werden.

II.4.1.2 Aufsetzsymbole

Dieser Parser kennt keine Aufsetzsymbole. Im Fehlerfall wird bis zum Zeichen "END_OF_FILE" gelesen und eine Fehlermeldung ausgegeben.

II.4.2 Statische Semantik

Die statische Semantik wird überprüft, wenn der Typ aus der Datenbasis mit dem Import-Konstrukt in eine Modelldefinition eingefügt wird.

II.4.3 Aufruf des Parsers

Der Parser wird im Lisp-top-level über die Funktion /hiqual-type-define/ mit einem String als Parameter, welcher obiger Grammatik genügt, aufgerufen.

II.4.4 Funktionsweise und benutzte Module

Der String, der als Parameter eingegeben wurde, wird geparst, auf syntaktische Korrektheit untersucht und die darin enthaltene Information wird in die Datei >HIQUAL>DATA>TYPES><TYPENAME>.HIQUAL geschrieben. Sie kann dort über den Standardeditor gelesen und verändert werden. Beteiligt sind die Module TYPE-DEFINE, SCANNER und STACK.

II.5 Stack

Der Stack wird von allen Parsern in Hiqual benutzt.
Er ist als Liste implementiert und bietet folgende Operationen an :

TOP ist durch eine Variable realisiert, die nach jeder Operation neu belegt wird;
POP, PUSH und SHOW sind durch Funktionen realisiert.

Wenn versucht wird POP auf den leeren Stack anzuwenden, wird eine Fehlermeldung erzeugt. Es folgt ein Tag ABORT-ALL, der vom aufrufenden Programm aufgefangen und verarbeitet werden muß.

Der Stack ist im Modul STACK implementiert.

II.6 Scanner

Der Scanner ist als deterministischer endlicher Automat implementiert. Er ist charakterisiert durch die folgende Grammatik zur Erkennung aller erlaubten Grundsymbole (tokens).

```
token          ::= identifier |  keyword |  number |  lonely-
                   chars | special-chars | END_OF_FILE

identifier     ::= letter (digit | _ | letter) *
keyword        ::= <Liste aller Hiqual-Keywords>
number         ::= (digit) +
lonely-chars   ::= ( | ) | { | } | ; | , | ' | . | @
special-chars  ::= \= | = | => | > | < | <= | >= | =<

letter         ::= A |...| Z | a |...| z
digit          ::= 0 |...| 9
```

Besonderheiten:

- In der Eingabegrammatik sind neben den Groß- auch Klein-Buchstaben zugelassen. Sie werden im Hiqual-System jedoch nicht unterschieden. Der Scanner transformiert alle Klein- zu Großbuchstaben.
- Ein Quote leitet einen Kommentar im Quelltext ein. Beim Erkennen dieses Zeichens wird der Rest der Zeile ignoriert.

Aufruf des Scanners mit (NEXT-ITEM).
Rückgabe des erkannten Grundsymbols als Wert der Variablen *next-item*. Unter der Property TYP der Variablen ist der Typ des erkannten Tokens vermerkt. Mögliche Typen sind:
Name, Keyword, Character, Spezchar, Number, Error und EOF.
Error bedeutet, daß ein unzulässiges Token erkannt und zurückgegeben wird.
EOF ist nur möglich, wenn das Token END_OF_FILE als File-Ende-Kennzeichen übergeben wird.

III Der Modellgraph

Die Modellgraphgenerierung läßt sich als kombiniertes Such- und Matchproblem charakterisieren. Auf dieser Basis ist sie in diesem Kapitel beschrieben.

Die Lösung eines Suchproblem erfordert im allg. folgende Komponenten:

- Eine Datenbasis, die eine Menge von Zuständen verwaltet; auch als Suchraum bezeichnet;
- Eine Menge von Operatoren, die einen existierenden Zustand in einen neuen Zustand überführen und damit die Datenbasis verändern;
- Einen Kontrollmechanismus, der die Anwendung der Operatoren steuert.

Im Falle der MG-Generierung heißt das:
Der Suchraum ist der gewünschte MG, nachdem der Suchprozeß terminiert.

III.1 Begriffe und Überblick

PVA bezeichnet ein Element aus der Menge der definierten Ports, Variablen und Attribute. Während der Graphgenerierung sind mit jedem PVA dynamisch auch jeweils gewisse Werte assoziert. Diese Werteassoziationen werden als **Assertions** verwaltet. Die syntaktische Form einer Assertion ist (<pva> <comparator> <value>), wobei <comparator> für ein in Hiqual definierter Vergleichsoperator steht und <value> einen Wert aus dem Wertebereich von <pva> repräsentiert.
Bsp.: (Inport_1 > drei).
Conditions haben die gleiche Form wie Assertions, stehen jedoch für Wertebelegungen, die im Precondition-Teil einer Regel definiert sind, d.h. eine Condition bezeichnet nicht den tatsächlichen, sondern den möglichen Wert einer PVA.
Die Menge der zu einem Zeitpunkt gültigen bzw. bekannten Assertions ist jeweils in einer **Situation** zusammengefaßt. Eine Situation enthält für jedes PVA maximal eine Assertion. Existiert ein PVA, dem noch keine Assertion zugewiesen wurde, dann ist die entsprechende Situation unvollständig. Wie bereits in Kapitel I. erwähnt, ist eine Situation als ein qualitatives zeitliches Intervall zu interpretieren, in dem sich keine der darin enthaltenen Assertions ändert.
Zu einer Situation zählt auch die aktuelle Facette. Sie bezeichnet die Facette, in der sich das Modell während der Ver-

haltensanalyse gerade befindet. Dementsprechend sind alle Regeln innerhalb der aktuellen Facette **aktive** Regeln. Im MG charakterisiert eine Situation einen Knoten. Kanten bezeichnen den Übergang von einer Situation S-i zu einer Nachfolgersituation S-j. Im MG sichtbar sind nur die Cf-Intervall Darstellungen der Assertions, die sich beim Übergang von S-i nach S-j geändert haben bzw. neu hinzugekommen sind.
Die **Initialsituation** enthält mindestens die Initialfacette, die in der INITIALLY-Klausel der MSP definiert ist. Zusätzlich kann sie noch Assertions von Attributen enthalten, denen ein Initialwert bei der Modellinstanziierung zugewiesen wurde. Für Ports und Variablen sind in diesem Stadium noch keine Wertebelegungen bekannt, d.h. eine Initialsituation ist immer unvollständig.
Die Expansion (Berechnung der direkten Nachfolgersituationen) einer Situation ist durch zwei Kriterien motiviert:

- Welche äußeren Größen (Werte an den Inports) können sich änderen?
- Welche Auswirkungen haben diese Änderungen auf das Modellverhalten?

Die Änderung von äußeren Größen wird durch Anwendung eines **Komlementierungsoperators** (KOP) simuliert. Dies ist der einzige Operator, der dem Kontrollmechanismus zur Verfügung steht.
Die Analyse der Auswirkungen auf das Modell nach jeder Operatoranwendung übernimmt ein **Regelinterpreter.** Er testet jede **aktive** Regel auf Anwendbarkeit und erzeugt gegebenenfalls eine neue Situation.

Anwendung des Komplementierungsoperators:
Die einzigen äußeren Größen, die sich außerhalb der Kontrolle eines Modells ändern können, sind Werte von Inports. Der Operator ist deshalb nur auf Inport-Assertions anwendbar. Als Ergebnis einer Anwendung liefert der KOP die Komplement-Assertion einer gegebenen Inport-Assertion.
Bsp.: KOP((Inport_1 > drei)) = (Inport_1 ≤ drei).
Welche Ports sich ändern, eventuell gleichzeitig mit anderen Ports, ist nicht bekannt. Es müssen deshalb alle möglichen Kombinationen betrachtet werden.

Ein Permutationsmechanismus **PERMUT** generiert systematisch jede Änderungskonstellation von Inports. Auf jedes Inport, das sich innerhalb der Konstellation ändern soll, wendet PERMUT den KOP an und erzeugt auf diese Weise jeweils neue Eingangsgrößen.
Eine Substitution der entsprechenden alten Assertions durch die neuen Assertions aus der aktuellen Änderungskonstellation ergibt bei jeder einzelnen Permutation einen neuen **Situationstyp.** Jeder neue Situationstyp wird anschließend dem Regelinterpreter übergeben, der die Anwendbarkeit der aktiven Regeln unter dem Situationstyp überprüft und somit die Auswirkungen einer Änderungskonstellation auf das Modellverhalten ableitet.

Die Menge der aktiven Regeln charakterisieren die zugelassenen

Reaktionen eines Modells auf eine konkrete Situation innerhalb der aktuellen Facette. In diesem Sinne ist die Aufgabe des **Regelinterpreters** die Ableitung von Auswirkungen auf der Basis einer neuen oder bereits existierenden Situation.
Folgende Vorausetzungen müssen vor bzw. bei einer Regelanwendung erfüllt sein:

- Eine Regel kann nur feuern, wenn jede Condition aus der Preconditionliste erfüllt ist. Die Erfüllbarkeit einer Condition wird durch einen Matchingprozeß ermittelt, bei dem eine Condition gegen ihre entsprechende Assertion aus dem Situationstyp gematcht wird.
- Die gleiche Regel kann nicht mehrfach hintereinander feuern, solange sich kein Ereignis aus der Preconditionliste seit der letzten Regelanwendung geändert hat. Diese Forderung leitet sich aus dem Ereignisbegriff ab. Solange sich keines der in der Precondition beteiligten Ereignisse ändert, kann eine weitere Regelanwendung auch kein neues Ereignis definieren.
- Es feuert immer nur genau eine Regel, d.h. es gibt keine parallele Ausführung von Regeln.

Erfüllt eine aktive Regel die oben genannten Bedingungen, dann ist es eine unter dem neuen Situationstyp **reaktive** Regel.
Der Regelinterpreter untersucht sukzessive jede aktive Regel auf ihre reaktive Eigenschaft.
Jede anwendbare Regel impliziert die Generierung einer neuen **Situationsinstanz** aus dem zugrundeliegenden Situationstyp. Eine Situationsinstanz (bisher und im folgenden auch als Situation bezeichnet) entsteht durch Substitution von bereits existierenden Assertions aus dem Situationstyp durch die von einer Regelanwendung implizierten jeweiligen neuen Assertions. Falls innerhalb eines Situationstyps für ein PVA noch keine Assertion existiert, jedoch auf der lhs oder rhs einer feuerenden Regel eine entsprechende Condition oder Action definiert ist, dann wird die Situationsinstanz um die entsprechende Assertion erweitert.

Die Assertions, die beim Übergang von der alten zur neuen Situation entstehen, werden in die explizite Cf-Intervall Darstellung transformiert und als sichtbaren Knoten in den Modellgraphen übernommen.
Falls für einen Situationstyp keine reaktive Regel existiert, dann entsteht aus einem Situationstyp unmittelbar eine Situationsinstanz, die nur regelunabhängige Cf-Intervalle enthält und zwar die von PERMUT generierten Assertions einer Änderungskonstellation.

Terminationsbedingung:
Mit jeder neu entstandenen Nachfolgersituation verzweigt der Graph auch in einen neuen alternativen Pfad. Die Suche auf einem Pfad terminiert, falls eine **Terminalsituation** erreicht ist. Sie ist dardurch charakterisiert, daß alle ihre direkten Nachfolgersituationen bereits im expandierten Graphen existieren. Abhängig davon, wo diese Nachfolgersituationen im Graphen

vorkommen, werden zwei Arten von Terminationen unterschieden:
- eine zyklische und
- eine azyklische Termination.

Bei der zyklischen Termination existiert auf dem gleichen Pfad eine andere Situation, die die gleichen direkten Nachfolgersituationen bestitzt wie die Terminalsituation. Liegt die entsprechende Situation nicht auf dem gleichen Pfad, dann wird von einer azyklischen Termination gesprochen.
Mündet jeder begonnene Pfad in einer Terminalsituation, dann terminiert die Modellgraphgenerierung. Da die beiden Terminationsbedingungen ein Verzweigen in andere Pfade erlauben, entsteht aus dem bis dahin baumartigen Suchraum ein Graph.
Der Kontrollmechanismus, der den Aufbau des Modellgraphen steuert, wird als **MGCTR** (Modelgraph-Control) bezeichnet. Er verwaltet die Berechnung der Nachfolgersituationen im Depth-First-Prinzip.

III.2 Aufbau des Modellgraphen

Eine Situation setzt sich aus maximal fünf disjunkten Mengen zusammen. Die Menge der bekannten
- Inport-Assertions,
- Outport-Assertions,
- Variablen-Assertions,
- Attribut-Assertions und der
- aktuellen Facette.

Inport-Assertions können nicht von Regeln manipuliert werden, während alle anderen Größen ausschließlich unter der Kontrolle von Regeln stehen.
Interessant sind die beiden Fragen:
- Wie verhält sich ein Modell bei konstanten Eingangsgrößen?
- Wie verhält sich ein Modell bei wechselnden Eingangsgrößen?

Der Wechsel einer Eingangsgröße ist nur aus einer bekannten Größe ableitbar. Er ergibt sich durch Invertierung der entsprechenden Inport-Assertion durch Anwendung des Komplementierungsoperators.
Welche möglichen Konstellationen von sich ändernden Inport-Assertions sind möglich? Zur Entwicklung aller Möglichkeiten wird der Permutationsmechanismus PERMUT eingesetzt.

III.2.1 Der Permutationsmechanismus PERMUT

Basis für PERMUT ist die Menge der bekannten Inport-Assertions

In-Ass einer Situation S-i. Sei n=|In-Ass|; die Anzahl der Assertions.
Durch Anwendung von KOP auf jede Assertion in In-Ass entsteht eine neue Menge von Assertions New-In-Ass. Sei m=|New-In-Ass|; dann gilt n≤m≤2*n.

Bsp.:
In-Ass = {(p1 = 2),(p2 > 4)},
New-In-Ass := KOP[In-Ass]={(p1 < 2),(p2 > 2),(p2 ≤ 4)}
wobei m>n.

Der Fall m>n ist nur möglich, falls in In-Ass eine Assertion mit Gleichheitsoperator enthalten und der Wertebereich des entsprechenden Ports als geordnete Liste definiert ist.
Die Anzahl der möglichen Änderungskonstellationen ergibt sich aus 2↑m. In einer Schleife von p=0 bis 2↑m - 1 werden nacheinander alle Änderungskonstellationen generiert.

Prinzip:
0. Setze p=0.
1. Konvertiere p in einen Bitvektor (b1,...,bn).
2. Assoziiere Bit bi ,1≤i≤m, mit dem i-ten Element aus New-In-Ass.
3. Vereinige alle Assertions Ai aus New-In-Ass, deren entspr. Bit bi=1 ist, in einer neuen Menge Sub.
4. Falls Sub mehr als eine Assertions für ein Port enthält, gehe nach 7.
5. Generiere durch Substitution der Inport-Assertions aus Si durch die entspr. Assertions aus Sub eine neuen Situationstyp.
6. Übergebe den Situationstyp dem Regelinterpreter.
7. Falls p≤2↑m - 1, dann erhöhe p um 1 und gehe nach 1.

p=0 heißt, daß der Regelinterpreter untersuchen soll, ob sich in einer konstant bleibenden Situation noch weitere Reaktionen ergeben können; falls z.B. mehrere Regeln auf eine Änderungskonstellation anwendbar sind.

III.2.2 Der Regelinterpreter

Ein Situationstyp beschreibt den Zustand eines Modells, nachdem sich die Werte bestimmter Inports geändert haben, jedoch noch keine Regel reagiert hat.
Jede aktive Regel wird auf ihre reaktive Eigenschaft getestet und gegebenenfalls angewandt. Wie dieser Test abläuft und wie eine Regel angewandt wird, ist der Inhalt dieses Abschnitts.
Hier nocheinmal die Kriterien für eine reaktive Regel:
K1: Es kann zu einem Zeitpunkt nur eine Regel feuern.
Dies bedarf keiner weitern Erläuterungen.

K2: Eine Regel darf nur feuern, wenn sich seit der letzten Anwendung dieser Regel mindestens einer der Assertions der an der Precondition beteiligten PVA seitdem geändert hat. Falls sich keiner der Assertions seitdem änderte, dann muß mindestens ein Facettenwechsel stattgefunden haben.
Diese beiden Bedingungen verwaltet eine **Rulehistory**, die sich an die jeweiligen Nachfolgersituationen weitervererbt. Sie allein genügt, um diese Forderungen zu testen.

K3: Eine Regel kann nur feuern, wenn jede einzelne Condition aus der Preconditionliste erfüllt ist, da alle Conditions implizit durch logisch UND verknüpft sind.
Dieser Teil erfordert ausführlichere Erläuterungen.

Erfüllbarkeit einer Condition:
In einer Precondition sind zwei Arten von Conditions zugelassen.
Conditions der Form:
- (<pva> <comparator> <value>), hier als Value-Condition bezeichnet und
- (<pva-1> <comparator> <pva-2>), hier als symbolische Condition referiert.

Nur ein Matchingprozeß gibt Auskunft über die Erfüllbarkeit einer Condition. Da beide Arten von Conditions jeweils einen eigenen Matchingprozeß verlangen, wird die Behandlung der beiden Conditionarten getrennt besprochen.

Erfüllbarkeit für Value-Conditions:
Eine Value-Condition ist erfüllt, falls einer der beiden Bedingungen zutrifft:
- Für das entsprechende PVA der Conditon existiert in der Vorgängersituation bzw. im Situationstyp noch keine Assertion. In diesem Fall wird die Condition uneingeschränkt als erfüllt betrachtet.
- Der Matchingprozeß zwischen der Condition und ihrer entsprechenden Assertion aus dem aktuellen Situationstyp ist erfolgreich, d.h. die Gültigkeitsbereiche der Condition und der Assertion besitzen eine gemeinsame Schnittmenge.

Erfüllbarkeit einer symbolischen Condition:
Der Erfüllbarkeitstest einer symbolischen Condition verlangt eine andere Semantik als für Value-Conditions. In einer symbolischen Condition sollen die aktuellen Assertions von <pav-1> und <pva-2> in Relation zueinander gesetzt werden. Ein Vergleich ist jedoch nur möglich, wenn bereits für beide PVA im Situationstyp eine Assertion existiert. Gilt dies nicht, dann ist auch keine Aussage über die Gültigkeit dieser Condition möglich, d.h. sie wird als nicht erfüllt angenommen.
Existieren hingegen beide Assertions, dann muß ein eigenständiger Matchingprozeß über die Erfüllbarkeit entscheiden.

Bsp.:

Aus der symbolischen Condition (p1 > v1) ensteht die erweiterte Condition
((p1 = 3) > (v1 < 2)); sie wäre in diesem Fall erfüllt.

Um den Entscheidungsprozeß einer neuen Situation durch Anwendung einer reaktiven Regel detailliert darstellen zu können, ist ein Verständnis der beiden Matchingprozesse erforderlich. Sie sind Thema des nächsten Unterabschnitts.

III.2.2.1 Matching

Zuerst ein Rückblick auf die Definition von Wertebereichen für Ports, Variablen und Attribute in der TYPES-Klausel einer MSP. Wertebereiche können als geordnete Listen oder als ungeordnete Mengen definiert werden.
Sei Typ-x=(v1;v2;...;vn) ein geordneter Wertebereich. Dann ist über die < - Relation für Typ-x eine irreflexive u. transitive Ordnung festgelegt, mit vi<vj <=> i<j und 1≤i,j≤n.
Des weiteren gilt analog vi=vj <=> i=j als Äquivalenzrelation.
Sei Typ-y={v1,...,vm} ein ungeordneter Wertebereich. Dann ist für Typ-y nur die Gleichheitsrelation (=) definiert, mit
vi=vj <=> i=j bzw. vi≠vj <=> i≠j und 1≤i,j≤m.
In dieser Implementierung werden die Wertebereiche als endlich angenommen.
Um den Matchingprozeß nur auf numerische Operationen zu reduzieren, werden Wertebereiche in eine andere Darstellung transformiert. Statt jedes Element aus einem Wertebereich explizit zu referieren, genügt es nur auf seinen Index zuzugreifen, der die Position des Elementes innerhalb der Liste bzw. Menge angibt.
Aus (v1;...;vn) wird (1,...,n) und aus
{v1,...,vm} wird {1,...,m}.
Die Liste (1,...,n) besitzt ein minimales und ein maximales Element. Für eine einfache Repräsentation genügt eine Intervalldarstellung der Form [1 n]. Genauso läßt sich auch eine Condition bzw. Assertion transformieren.

Bsp.:
Sei Typ-1=(null;eins;zwei;drei;vier),
Typ-2={blau,grün,rot,gelb}.
Typ-1 ist Wertebereich für p1 und
Typ-2 ist Wertebereich für p2.
(p1 > null) => (p1 [2 5]) heißt Intervallcondition,
(p1 = drei) => (p1 [4 4]) " " ,
(p2 = blau) => (p2 {1}) heißt Mengencondition,
(p2 ≠ gelb) => (p2 {1 2 3}) " ".

Definiere:
sub[(p1 [2 5])]=2 und inf[(p1 [2 5])]=5.

III.2.2.1.1 Matching von Value-Conditions

Sei Ass eine Assertion für ein PVA aus einem Situationstyp,
Cond eine entspr. Condition aus einer Precondition.

MV: Ass x Cond --> Fail-cond1 x Success x Fail-cond2

ist die Matchingfunktion, wobei Fail-cond1, Success und Fail-cond2 wiederum Conditions sind oder Nil.
Success ≠ Nil bedeutet, daß der Gültigkeitsbereich von Cond und Ass sich ganz oder teilweise überdecken. Fail-cond1 und Fail-cond2 können jeweils ≠ Nil sein, wenn sich Ass und Cond nur teilweise überdecken. Sie kennzeichnen die Differenzmenge bei einem teilweise erfolgreichen Match. Mit ihnen wird der mögliche Fall berücksichtigt, daß sich der "tatsächliche" Wert eines PVA nicht in der Schnittmenge der beiden Gültigkeitsbereiche liegt, sondern gerade in der Differenzmenge, so daß die Condition nicht erfüllt ist und eine Regel nicht feuern kann.
Beachte:
Ass und Cond beziehen sich auf das gleiche PVA, d.h. es wird nur auf einen Wertebereich referiert.

Berechnung von MV:

1. Fall: Der Wertebereich ist eine geordnete Liste.

i) **Ass und Cond sind Intervallconditions.**

a) Es existiert keine gemeinsame Überdeckung.
Fail-cond1, Success und Fail-cond2 sind Nil.

b) Es existiert eine gemeinsame Überdeckung.
sub[Success] := MAX(sub[Ass], sub[Cond]),
inf[Success] := MIN(inf[Ass], inf[Cond]),

sub[Cond] > sub[Ass]
<=> sub[Fail-cond1] := sub[Ass] und
inf[Fail-cond1] := sub[Cond]-1,
sonst: Fail-cond1 := Nil.

inf[Cond] < inf[Ass]
<=> sub[Fail-cond2] := inf[Cond]+1 und
inf[Fail-cond2] := inf[Ass],
sonst: Fail-cond2 := Nil.

ii) Entweder Ass oder Cond oder beide sind Mengenconditions. In diesem Fall interpretiere auch Intervallconditions als Mengenconditions. Wende zum Matchen die bekannten mengentheoretischen Operatoren an:

Success := Ass ∩ Cond,

Fail-cond2 := Nil und
Fail-cond1 := {x|x ∈ Ass ∧ x ∉ (Ass ∩ Cond)
∧ Success ≠ Nil}.

2. Fall: Der Wertebereich ist eine ungeordnete Menge.
In diesem Fall können für Ass und Cond nur Mengenconditions auftreten, d.h. das Matching berechnet sich wie im 1. Fall bei ii).

Sind Ass und Cond beide Intervallconditions, dann sind es auch Fail-cond1, Success und Fail-cond2, soweit sie nicht Nil sind. Im anderen Fall ergeben sich ausschließlich Mengenconditions bzw. Nil als Ergebnis.

III.2.2.1.2 Matching von symbolischen Conditions

Beim Matching von Value-Conditions genügte es, die beiden Conditions auf eine gemeinsame Schnittmenge ihrer Gültigkeitsbereiche hin zu untersuchen. Symbolische Conditions geben zusätzlich an, wie sich die beiden Gültigkeitsbereiche zueinander verhalten sollen.

Beisp.:
Sei p1 ein Inport, v1 eine lokale Variable jeweils mit dem Wertebereich Typ-1 wie zuvor definiert.
(p1 > v1) die symbolische Condition,
((p1 > eins) > (v1 < vier)), die erweiterte symbolische Condition, wobei (p1 > eins) als Ass1 und (v1 > vier) als Ass2 bezeichnet sind. Ass1 und Ass2 sind beide als Assertions zu verstehen.
Problem:
Es ist keine direkte eindeutige Aussage möglich.

Bevor das Beispiel weiter aufgelöst wird, zunächst die Definition der Matching-Funktion MS für symbolische Conditions:

MS: Ass1 x Comp x Ass2 --> Refine-conds x Success.

MS ist nur definiert, falls das PVA von Ass1 und Ass2 den gleichen Wertebereich besitzen. Da im MG nur Value-Conditions zugelassen sind, müssen die Conditions von Refine-conds und Success ebenfalls Value-Conditions sein, sofern die beiden Rückgabewerte nicht Nil sind.
Falls Success ≠ Nil, dann enthält es eine Value-Conditions für das PVA in Ass1. Refine-conds enthält Value-Conditions als Verfeinerungen von Ass1 oder Ass2. Refine-conds und Success sind niemals beide ≠ Nil.

Zurück zum obigen Beispiel:
Die Gültigkeitsbereiche von Ass1 und Ass2 überdecken sich nur

teilweise und sind zusätzlich über die > Relation relativiert. Um soweit wie möglich eindeutige Aussagen ableiten zu können, werden die beiden Assertions in disjunkte und gleiche Gültigkeitsbereiche aufgespaltet. Die disjunkten Bereiche lassen sich eindeutig unter dem Comparator in Comp entscheiden. Die Unsicherheit ist nur noch auf die beiden gleichen Gültigkeitsbereiche reduziert. Für die Berechnung von MS gilt jedoch:
Wenn beide Assertions Ass1 und Ass2 verfeinert werden können, dann gebe nur die Verfeinerung von Ass1 zurück. Dies rechtfertigt sich aus der Intention die Verzweigungsrate im Modellgraphen möglichst gering zu halten, da für jede Condition aus Refine-Conds später ein eigener Pfad entsteht. Die endgültige Entscheidung über die Gültigkeit einer symbolischen Condition wird dann in einer weiteren Anwendung von MS getroffen, wobei Ass1 dann eine Verfeinerung der ursprünglichen Ass1 repäsentiert.
Die Gültigkeitsbereiche von Ass1 und Ass2 werden mit der Matchingfunktion MV aufgespaltet und zwar wie folgt:

1) MV(Ass1,Ass2) = (Fail-cond1,Success1,Fail-cond2).
2) MV(Ass2,Ass1) = (Fail-cond3,Success2,Fail-cond4).

Die beiden Anwendungen von MV liefern alle erforderlichen Verfeinerungen von Ass1 und Ass2. Die Ergebnisse hier sind:

Fail-cond1 = Nil, Success = (v1 in-list (zwei drei)),
Fail-cond2 = (p1 = vier),
Fail-cond3 = (v1 in-list (null eins)),
Success2 = (p1 in-list (zwei drei)), Fail-cond4 = Nil.

Der in-list-Operator entsteht bei der Transformation einer Intervallcondition aus dem internen Format in die Assertionform. Falls das Sub und das Inf der Intervallcondition nicht dem kleinsten bzw. dem größten Wert des entsprechenden Wertebereiches entsprechen, dann sind die bisher verwendeten Hiqual-Comparatoren nicht mehr anwendbar.

Für die Berechnung von MS gilt allgemein:

- sind alle Fail-conds und Success-Conditions gleich Nil, dann gilt Refine-conds := Nil, Success := Nil.
- Sind alle Fail-conds = Nil, die beiden Success-Conditions ≠ Nil, dann gilt Refine-conds := Nil u. Success := Success2, die Condition für das PVA von Ass1.
- Ist einer der Fail-conds ≠ Nil, dann unterscheide:
  Existieren Verfeinerungen von Ass1, dann gilt:
  Refine-conds := Vereinigung aller Conditions aus Fail-conds und Success-Conditions, die eine Verfeinerung von Ass1 sind, sonst
  Refine-conds := Vereinigung aller Verfeinerungen von Ass2.

FUr das Beispiel gilt entsprechend:

MS((p1 > eins),>,(v1 < vier)) =
[((p1 in-list (zwei drei)), (p1 = vier)), Nil].

Eine weitere Anwendung von MS mit
MS((p1 = vier),>,(v1 < vier)) = [nil, (p1 = vier)],
ergibt ein erfolgreiches Matching.
MS((p1 in-list (zwei drei)),>,(v1 < vier)) =

[((v1 in-list (null eins)),(v1 in-list (zwei drei))),Nil]
muß durch zwei weitere Anwendungen von MS aufgelöst werden.
MS((p1 in-list (zwei drei),>,(v1 =vier)) = [Nil,Nil]
liefert ein Fail.
MS((p1 in-list (zwei drei)),>,(v1 in-list (zwei drei))) =
[Nil, (p1 in-list (zwei drei))]
ist per Definition erfolgreich.

Allgemein gilt:
Zwei gleiche Gültigkeitsbereiche werden von MS immer erfolgreich gematcht, unabhängig vom Comparator in Comp. Um zwei gleiche Gültigkeitsbereiche endgültig zu entscheiden, müßten alle möglichen Kombinationen von Werten gebildet werden. Da dies aus kombinatorischen Gründen ausscheidet, wird die vereinfachte Annahme eines erfolgreichen Matches getroffen.
Ist eine der beiden Assertions Ass1 oder Ass2 oder beide eine Mengencondition, dann sind auch die Resultate von MS immer Mengenconditions, sofern das Ergebnis nicht Nil ist. Dies resultiert aus der Verwendung von MV, das die gleiche Eigenschaft besitzt. Auf eine Beschreibung der Behandlung von Mengenconditions wird hier verzichtet, da sie sich aus den Eigenschaften von MV ableiten läßt.

III.2.2.2 Generierung von Situationsinstanzen

Die Erfüllbarkeit einer Condition läßt sich in zehn Fälle differenzieren:
Eine Condition ist eine Value-Condition mit

C1: Für das entspr. PVA existiert noch keine Assertion, die Condition gilt als erfüllt.
Falls bereits eine Assertion existiert, dann unterscheide:
C2: MV(Ass,Cond) = (Nil,Nil,Nil),
C3: = (Nil,Success,Nil),
C4: = (Fail-cond1,Success,Nil),
C5: = (Nil,Success,Fail-cond2),
C6: = (Fail-cond1,Success,Fail-cond2).

Eine Condition ist eine symbolische Condition mit:

C7: Sie ist unerfüllt, da eine der notwendigen Assertions noch nicht existiert,
C8: MS(Ass1,Comp,Ass2) = (Nil,Nil),
C9: = (Nil,Success),
C10: = (Refine-conds,Nil).

Neue Siuationsinstanzen entstehen bei einer der vier folgenden Bedingungen:

Sit-1: Eine Regel gilt als erfüllt, wenn für jede Condition aus der Precondition einer der Fälle C1, C3, C4, C5, C6 oder C9 zutrifft.
Für die erfüllte Regel wird eine neue Situationsinstanz (SI) gebildet. Die SI entsteht durch Substitution von Assertions aus dem Situationstyp durch die entsprechende Success-Conditions der einzelnen Matchingprozesse , oder durch Hinzufügen von Conditions zum Situationstyp, für die C1 zutrifft.
Desweiteren wird die neue SI um die im Action-Teil einer Regel vorkommenden Conditions aktuallisiert; entweder durch Substitution oder Hinzufügen der entsprechenden Conditions bzw. Assertions. Dazu gehört auch die eventuell vorhandene Enter-Facette.

Bilde eine Menge von Assertions RUA, die nach den folgenden beiden Kriterien ensteht:

Sit-2: Für jede erfüllte Regel, die Conditions enthält, auf die C4, C5 oder C6 zutrifft, erweitere RUA um die entspr. Fail-conds der jeweiligen Matchingprozesse.
Sit-3: Bei jeder unerfüllten Regel, die eine Condition enthält auf die C10 zutrifft, bilde RUA := RUA ∪ Refine-conds.

Generiere aus jeder Assertion aus RUA eine neue SI durch Substitution der entsprechenden Assertion aus dem Situationstyp mit der Assertion aus RUA. Es entstehen dementsprechend soviele SI wie RUA Assertions enthält.

Sit-4: Falls auf keine aktive Regel Sit-1 bis Sit-3 zutrifft, oder K2 nicht erfüllt ist, dann generiere aus dem Situationstyp unmittelbar eine neue SI.

III.2.2.3 Stetigkeitsannahmen

Mit Hilfe der Continuity Assumption, die ein wesentlicher Bestandteil der gesamten Analyse ist, läßt sich die Zahl der Nachfolgersituationen bedeutend einschränken. Sie findet in folgendem Fall Anwendung:
Änderungen von Inport-Assertions entstehen durch Anwendung des KOP. Von Grenzfällen abgesehen, ist das Ergebnis im allgemeinen eine Assertion als Ungleichung, die einen größeren Gültigkeitsbereich abdeckt.

Bsp.:
Sei Typ-1 = (null;eins;zwei;drei;vier),
p1 ein Inport mit Wertebereich Typ-1, für das die Assertion (p1 = drei) gilt.
KOP[(p1 = drei)] = {(p1 < drei), (p1 = vier)}.

Angenommen es existiert eine reaktive Regel mit einer Precondition (p1 = eins).
Der Matchingprozeß MV ergibt:
MV[(p1 < drei),(p1 = eins)] =
((p1 = null),(p1 = eins),(p1 = zwei)).
Dieses Ergebnis hat die Entstehung von drei Nachfolgersituationen zur Folge.

Unter Verwendung der Stetigkeitsannahme entsteht jedoch nur eine einzige Nachfolgersituation und zwar die für Fail-cond2 = (p1 = zwei).
Der Wert von (p1 = drei) kann sich nicht in Nullzeit, also sprunghaft in (p1 = eins) ändern.
Mit dieser Annahme ist es zulässig, die beiden anderen möglichen Situationen aus der Menge der Nachfolgersituationen zu suspendieren.

Im Zusammenhang mit der Stetigkeit sei noch erwähnt, daß die Stetigkeit von Outport-Ereignissen nicht kontrolliert wird. Dies übernimmt die Aggregatanalyse.

III.2.2.4 Generierung der Cf-Intervalle

Conditions im Precondition- und Actionteil einer Regel definieren lokale Ereignisse, die über zeitliche Relationen miteinander in Verbindung stehen. Der Ereignisbegriff wurde bis jetzt bewußt vermieden, da bei der Modellgraphgenerieung zeitliche Relationen noch nicht von Belang sind. Ereignisse in Form von Assertions sind in Situationen zusammengefaßt, wobei der Übergang von einer Situation S-i zur Situation S-j gekennzeichnet ist durch die Assertions, die sich beim Übergang von S-i nach S-j ändern bzw. neu hinzukommen. Assertions können sich im Kontext einer Regel ändern bzw. redefiniert werden, wenn die gleiche Assertion nacheinander in verschiedenen Regeln auftritt. Assertions können jedoch auch als regelunabhängige Ereignisse auftreten, z. B. im Falle von Inport-Änderungen, für die es keine reaktive Regel gibt.
Jede Assertion, die sich beim Übergang von S-i nach S-j ändert, neu hinzukommt oder redefiniert wird, repräsentiert ein lokales Ereignis, das zu anderen Ereignissen bestimmte vordefinierte oder spezialisierte zeitliche Relationen besitzt. Wie diese Relationen zu interpretieren sind, ist nicht Gegenstand des Modellgraphen. Diese lokalen Ereignisse müssen lediglich als individuelle syntaktische Objekte identifizierbar sein.
Für eine eindeutige Repräsentation wurde die in [1] vorgestellte Notation zur Darstellung von ES-Expressions mit einer Erweiterung übernommen und als Cf-Intervall (Condition-Facet-Intervall) bezeichnet.

Syntaktische Struktur:

<caf>!<inst-name>!<rule-name>!<type>!<id>!<index>, mit

<caf>:
Condition, Action oder Facet der Form:
<pva><comp><value> für Condition bzw. Actions und FACET=<facet> für Facettenbezeichner.
<inst-name>:
Ist der Name der analysierten Modellinstanz.
<rule-name>:
Ist der Name der Regel, in deren Kontext die Condition/Action definiert ist bzw. der Name der Regel, die einen Facettenwechsel verursacht hat oder einfach '-' falls es sich um eine regelunabhängige Assertion handelt.
<type>:
Ist IN (instantaneous) oder NI (non-instantaneous) wie bereits im ersten Kapitel vorgestellt.
<id>:
Ist ein Kennzeichen um ein eventuell gleiches Ereignis auf einem anderen Pfad im MG von diesem zu unterscheiden.
<index>:
Gibt an, wie oft diese Condition bereits auf dem Pfad als Ereignis aufgetreten ist. Haben zwei lokale Ereignisse für das gleiche PVA verschiedene Indizes, dann gibt es zwischen dem Vorkommen des ersten und dem Vorkommen des zweiten mindestens eine Assertion für das gleiche PVA, die sich von den anderen beiden Conditions/Assertions syntaktisch unterscheidet.

Im Modellgraphen sind nur die Cf-Intervall-Darstellungen der Assertions bzw. der Facetten sichtbar, die den Übergang von S-i nach S-j kennzeichnen.
Alle Assertions und Facetten, die im Kontext einer Regel stehen, enthalten im <rule-name>-Teil des Cf-Intervall-Symbols den entsprechenden Regelnamen, in dem sie definiert sind. Dies sind alle (eventuell modifizierte) Conditions aus dem Precondition-Teil und alle Action-Conditions aus dem Action-Teil einer Regel inkl. der Enter-Facette. Alle regelunabhängigen Assertions erhalten ein '-' als Regelsymbol. Die aus Sit-2, Sit-3 und Sit-4 hervorgehenden Cf-Intervalle sind ausschließlich regelunabhängig. In Sit-1 können auch regelabhängige in Verbindung mit regelunabhängigen Cf-Intervallen vorkommen; wenn sich z.B. zwei Inport-Assertions ändern, jedoch nur eine Assertion in einer Precondition überprüft wird. Dann muß die andere Assertion als regelunabhängiges Cf-Intervall notiert werden.
Alle Cf-Intervalle, die zu einem Situationsübergang gehören, sind als ein Knoten im Modellgraphen sichtbar. Eine Ausnahme sind Facettenwechsel. Cf-Intervalle, die Facettenwechsel notieren, sind aus Gründen der Übersichtlichkeit als eigenständige Knoten ausgelegt. Sie sind dem eigentlichen Übergangsknoten unmittelbar nachgestellt.

Aus Effizienz- und Übersichtlichkeitsgründen sind die Cf-Intervalle, die die Conditions aus dem Action-Teil einer Regel repräsentieren, nicht sichtbar, sondern lediglich an die Cf-Intervalle gebunden, die die entsprechenden verursachenden Preconditions darstellen.
Die in der Modellspezifikation definierten zeitlichen Relationen zwischen Ereignissen bzw. Conditions sind den regelabhängigen Cf-Intervallen als Attribut mitgegeben und stehen für eine spätere Bearbeitung bereit. Regelunabhängige Cf-Intervalle besitzen die occurs-in Default-Relationenmenge als Attribut, d.h. sie sind nur mit der umschließenden Facette gekoppelt.

IV Cf-Instance-Pfade

Interessant für eine (zeitliche) Analyse sind besonders zyklische Verhaltensweisen im Modellgraphen. Ziel ist es beispielsweise, die zeitliche Konsistenz über mehrere Zykelwiederholungen nachzuweisen. Im MG sind Zykel nur über eine Rückwärtsverkettung angedeutet. Eine Analyse verlangt jedoch die explizite Präsenz von Zykelwiederholungen, die deshalb aus dem MG erst synthetisiert werden müssen.
Desweiteren können verschiedene Pfade aus dem MG zusammen in einen Pfad gemischt werden, so daß prinzipiell beliebig viele Arten von sequentiellen Verhaltensweisen komponierbar sind. Der MG ist somit als eine endliche Repräsentation von unendlich vielen Verhaltensformen zu verstehen, so daß vor Beginn der eigentlichen zeitlichen Analyse zwei Aufgaben zu lösen sind:

- Synthese einer konkreten Verhaltensweise aus der komprimierten Darstellung des Modellgraphen, einen sogenannten Verhaltens- oder Modellpfad.
- Spezifikation von zeitlichen Relationen, die sich speziell aus dem synthetisierten Pfad ergeben.

Der so entstandene Pfad ist die Grundlage für die Aggregat- und Eventanalyse.

IV.1 Generierung von Cf-Instance-Pfaden

Pfade im Modellgraphen beschreiben eine sequentielle Folge von Ereignissen in Form von Cf-Intervallen, die in zwei Arten von Terminalknoten enden:

- Terminalknoten, die ein zyklisches Verhalten dieses Pfades anzeigen.
- Terminalknoten, die auf ein azyklisches Verhalten zeigen, d.h. sie verzweigen in einen anderen beliebigen Pfad des Graphen, der das Verhalten des ursprünglichen Pfades fortsetzt, selbst aber aus einer anderen Ausgangssituation entstand. Der zweite Pfad kann selbst wieder zyklisch oder azyklisch terminieren.

Der Inspector bietet ein Hilfsmittel, um sich innerhalb des Modellgraphen durch mehrere Pfade "durchzuhangeln". Dabei kann ein zyklischer Pfad (virtuell) mehrfach durchlaufen werden. Der Zykel kann an beliebiger Stelle wieder verlassen und ein neuer Pfad betreten werden. Es ist praktisch möglich, ausgehend von der Wurzel des Graphen, entlang den existierenden gerichteten Kanten auch über die Terminalknoten jeden erreichba-

ren Knoten zu besuchen. Die Reihenfolge der Besuche wird vom Benutzer durch "Notieren" der benutzten Verzweigungspunkte gespeichert. Dazu genügt die Angabe der besuchten Terminalknoten und jeweils einer seiner direkten Nachfolger, zu denen er verzweigen kann. Das "Notieren" eines Knotens geschieht mittels Anklicken eines Cf-Intervalls aus einem Knoten mit der Maus. Die Reihenfolge der notierten Knoten, zusammen mit der Angabe, wie oft eventuell ein Zykel wiederholt werden soll, ergibt eine Pfadspezifikation. Die Anzahl der Zykelwiederholungen läßt sich bei Erreichen eines zyklischen Terminalknotens jeweils mit einem numerischen Argument festlegen.
Aus einer Pfadspezifikation entsteht nach Verlassen des Inspectors ein vollkommen neuer Verhaltenspfad auf der Basis von Cf-Instance-Intervallen; ein sogenannter Cf-Instance-Pfad. Cf-Instance-Intervalle haben die gleiche syntaktische Struktur wie Cf-Intervalle. Sie unterscheiden sich nur in der verwendeten Datenstruktur. Während im MG Cf-Intervalle von Conditions aus dem Action-Teil einer Regel nicht direkt sichtbar sind, gibt es im Cf-Instance-Pfad keine Unterscheidung mehr, so daß alle beteiligten Elemente eines Knotens im Pfad explizit sichtbar sind.
Jedem Cf-Instance-Intervall ist eindeutig ein Cf-Intervall zugeordnet. Wegen der Möglichkeit einen Zykel mehrfach aufzurollen, gilt die Umkehrung nicht, d.h. einem Cf-Intervall können mehrere Cf-Instance-Intervalle entsprechen.
Um mehrere Pfade für das gleiche Modell verwalten und referieren zu können, sind Cf-Instance-Pfade mit einem vom Benutzer definierten Namen bezeichnet. Die Konkatenation aus <Modellnamen> '-' <Pfadnamen> ergibt einen qualifizierten Pfadbezeichner, unter dem ein Cf-Instance-Pfad im Hiqual-System geführt und referiert wird.

IV.2 Pfadspezifische und vordefinierte Relationen

Zunächst ein Rückblick auf die in einer MSP definierbaren Relationen.
Conditions und Facetten innerhalb einer MSP werden jetzt wieder als Ereignisse referiert, da hier ihre zeitlichen Relationen zu anderen Conditions und Facetten von Bedeutung sind. Einen Überblick über definierbare Ereignisse und deren zeitliche Relationen ist in Abschnitt I.1.2 gegeben. Daraus geht hervor, daß es drei Kategorien von Relationen gibt:

- Die Not(starts-before)-Relationenmenge {=,mi,oi,f,d,s,si} zur Festlegung der Ursache-Wirkungsbeziehungen zwischen Precondition-Ereignissen und den verursachten Enter- und Non-Enter-Actions.
- Die Occurs-in-Relationenmenge Not{>,<,m,mi}, die anzeigt in welcher Facette ein Ereignis beobachtbar ist und
- Die Starts-in-Relationenmenge {=,d,oi,s,si}, die angibt,

wann eine Non-Enter-Action in der Enter-Facette startet.

Es ist zu beachten, daß sich die Not(starts-before)- und occurs-in- Relationenmenge von denen in [1] entsprechend definierten Mengen unterscheiden.
Die hier definierte not(starts-before)-Relationenmenge schließt die > Relation aus. Sie kann zwar aus prinzipiellen Erwägungen gerechtfertigt sein, würde aber die praktische Analyse scheitern lassen.
Die > Relation besagt, daß die Wirkung eines Ereignisses - z.B. ein Facettenwechsel - irgendwann nach dem zeitlichen Ende der Ursache beginnen kann. Dies bedeutet für die Analyse im MG, daß beim Übergang von der Situation S-i zur Situation S-j nicht bekannt ist, bei welcher Nachfolgersituation von S-j die neue Facette gültig wird, denn in S-j darf sie wegen der > Relation noch nicht vorkommen. Wenn sie jedoch in irgendeiner späteren Situation gültig wird, welche neuen Ereignisse sind inzwischen eingetreten? Zur Vermeidung dieser praktisch unlösbaren Problematik ist dieser Implementierung die eingeschränkte Relationenmenge zugrundegelegt.
Die Einschränkung der in [1] entsprechenden Relationenmenge not(>,m) (occurs-in-Relationen), um die < und mi Relation beruht auf der Überlegung, daß eine Regel nur feuern kann, wenn sich die verantwortlichen Ereignisse in irgendeiner Weise mit der per Definition umschließenden Facette zeitlich überdecken.

Die mit diesen drei Relationenmengen definierbaren ontologischen Beziehungen is-defined-by, is-caused-by, is-registrated-in und is-effected-in lassen folgende drei Beziehungen offen:

- Relationen zwischen verursachenden Ereignissen aus den jeweiligen Preconditions (Non-Action-Ereignisse);
- Relationen zwischen den verursachten Non-Enter-Actions;
- Die Meets-Relation zwischen Ereignissen für das gleiche PVA.

Relationen zwischen Non-Action-Ereignissen sind das Charakteristikum eines Modellpfades. Sie sind hauptsächlich durch die sich ändernden Inport-Ereignisse bestimmt, d.h. sie können bzw. müssen aus dem Pfad abgeleitet werden.
Relationen zwischen Non-Enter-Actions (für Outports und Variablen) explizit anzugeben oder zu versuchen sie ohne Propagation aus dem Pfad abzuleiten, ergäbe eine redundante Information. Non-Enter-Actions sind über eine Dreiecksbeziehung mit den verursachenden Ereignissen (is-caused-by Ontologie) und der entsprechenden Facette (is-effected-in Ontologie) bereits eng verknüpft, so daß die fehlenden Relationen bereits vorbestimmt sind. Der Versuch, weitere Relationen hinzuzufügen, könnte leicht zu Inkonsistenzen führen.
Die Meets-Relationen zwischen Ereignissen für das gleiche PVA sind ebenfalls ein Charakteristikum des synthetisierten Modellpfades und einfach zu bestimmen.

Ableitung neuer Relationen:
Die Bestimmung der Meets-Relationen bedarf keiner weiteren Erläuterungen. Sie können bei einem sequentiellen Durchlaufen des Pfades sofort abgelesen werden.
Zur Ableitung von zeitlichen Relationen zwischen Non-Action-Ereignissen wurde ein neuer onotologischer Typ definiert: **same-time-points** (st). Analog zu der occurs-in-Relationenmenge soll er ausdrücken, daß sich zwei Ereignisse irgendwie überdecken müssen, um miteinander relativiert werden zu können.
Same-time-points und is-registrated-in greifen somit auf die gleiche Relationenmenge zurück:
Not{<, >, m, mi} = {=, s, si, d, di, o, oi, f, fi},
wobei jedoch für die ST-Ontologie nur echte Teilmengen dieser Relationenmenge vergeben werden, da alle notwendigen Informationen explizit verfügbar sind.
Die jeweiligen Relationen ergeben sich hauptsächlich aus den von PERMUT generierten Änderungskonstellationen. Ableitbare Informationen beziehen sich auf den gleichzeitigen Anfang, Überlappung, Enthaltensein, gleichzeitiges Ende und die Gleichheit zweier Ereignisse.
Der Eventanalyse stehen somit die Relationen aus der MSP, sowie die neu abgeleiteten Relationen zur Verfügung.

Der vollständig entwickelte Cf-Instance-Pfad kann ebenso wie der MG mit dem Inspector besichtigt werden. Jedes Element der Datenstruktur, vor allem die bestehenden Relationen sind für jedes Cf-Instance-Intervall auf Wunsch darstellbar.

V Aggregate

V.0 Motivation

Systeme lassen sich nicht immer auf einfache und adäquate Weise als monolytische Elemente beschreiben.
Komplexität und Struktur vieler Objekte sind abhängig von der Beobachtungsebene.

Beispiel Kraftfahrzeugmotor :

Betrachtungsebene	Strukturelemente
Naive Gebrauchsanweisung	Kontaktschloß, Gaspedal, Kurbelwelle.
Pannenhilfe im Handbuch	Zündkerzen, Zündanlage, Zündkabel, Ölwanne, Vergaser, Dichtungen, Kühlung, Keilriemen etc.
Fachmann	Ventile, Stößel, Nockenwelle, Lager, Spule, Verteiler, Kontakte, Verteilerfinger, Drossel, Zylinderkopfdichtungen etc.

Die dargestellte Funktion ist grob gesehen auf allen Ebenen immer die gleiche. Genauer gesagt impliziert die verfeinerte Ebene das abstrakte Niveau.
In Hiqual wird diese Verfeinerung einer abstrakten Ebene durch hierarchische Systeme von Modellen beschrieben.
Das Kontaktschloß der obersten Ebene wird durch die Zündanlage, die Zündkabel und die Zündkerzen dargestellt.
Die Zündanlage wiederum besteht aus Spule und Verteiler.
Der Verteiler besteht aus Finger und Kontakten usw.
Die hierarchische Darstellung beinhaltet neben der räumlichen Aggregation auch eine zeitliche Komponente. Events auf einer abstrakteren Ebene passieren gleichzeitig mit ihren Verfeinerungen auf einer tieferen Ebene. Wenn das Gasgeben auf einer konkreteren Ebene durch die Vergasermechanik beschrieben wird, impliziert dies, daß beim funktionierenden Motor die Bewegungen des Gaspedals, des Vergasergestänges und des Nadelventils zeitgleich sind.

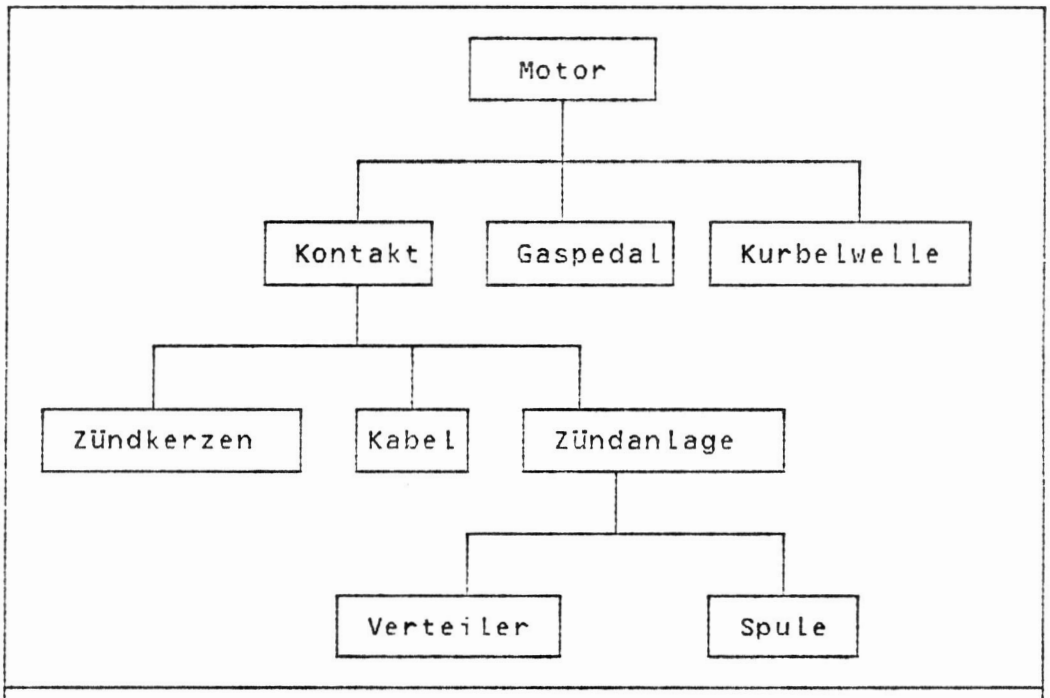


Abb. V.1 Hierarchische Darstellung der Elemente eines Kraftfahrzeugmotors

Hiqual ermöglicht es, selektiv nur die Modelle durch Aggregatbildung zu verfeinern, deren Verhalten auf einer tieferen Ebene den Benutzer interessiert.
Das Verhalten auf der verfeinerten Ebene verändert das Verhalten auf der abstrakteren Ebene nicht , sondern liefert zusätzliche, detailliertere Information für den Benutzer und die darunter liegenden Ebenen. Daher spielt es im obigen Beispiel für das Modell Motor keine Rolle, ob die Zündanlage oder die Kurbelwelle weiter verfeinert werden.
Die so erreichte Modularität ermöglicht es, Aggregate und komplexe Aggregate beliebig aus schon bestehenden Elementen zu komponieren.
Aggregate werden in Hiqual einer Analyse unterworfen, um festzustellen, ob die hierarchischen Beschreibungen wirklich Verfeinerungen voneinander sind.
Für das Aggregat Kfz-Motor wird untersucht, ob das Verhalten des Modells Motor auf der Ebene der Modelle Kontaktschloß, Gaspedal und Kurbelwelle nachvollziehbar ist.
Das abstrakteste Modell eines Aggregats heißt Wurzel, die Verfeinerungen bezeichnet man als Blätter.
Die Verbindung innerhalb eines Aggregats geschieht über die

Ports der Modelle.

V.1 Aufgaben und Probleme der Aggregatanalyse

Die Aggregatanalyse baut auf dem Modellgraphen, der von der Modellanalyse erzeugt wird, auf. Um die Kompatibilität der Blätter mit der Wurzel unter den angegeben Verbindungen zu überprüfen , müßte jede mögliche Situationsfolge aus dem Modellgraphen der Wurzel ihre Entsprechung in den Graphen der Blätter finden.
Der Modellgraph beschreibt eine endliche Anzahl von zyklusfreien Pfaden und abzählbar viele zyklische Pfade.
Interessante, nichttriviale Informationen, was die zeitliche Komponente angeht, sind aber erst von sich widerholenden Ereignissen zu erwarten. Die Auflösung von Zyklen läßt die Verschiebungen im zeitlichen Verhalten verschiedener PVA zueinander erkennen. Die mehrfache Auflösung desselben Zyklus erlaubt es festzustellen ob sich ein stabiles Verhalten einstellt. Da es abzählbar viele interessante Pfade gibt, ist eine vollständige Analyse nicht möglich ist.
Daher wird im Modellgraph der Wurzel ein Pfad vom Benutzer spezifiziert, auf den sich die Aggregatanalyse dann beschränkt.
Der Algorithmus vergleicht die Modelle situationsweise, indem er die Werte der Ports der Wurzel gemäß der Structure_Map, für jede neue Situation des Pfades der Wurzel auf die Ports der Blätter projiziert. Die Blätter reagieren auf diesen Input, versuchen ihn nachzuvollziehen, und kommunizieren ihrerseits über die Connections.
Ein Modell reagiert auf einen Input, indem es sich in seinem Modellgraphen von seiner aktuellen Situation fortschaltet auf eine Nachfolgersituation, wenn diese Transition mit den anliegenden Portwerten vereinbar ist.
Die Connections transferieren ihre Werte vom Out-Port (Sender) zum In-Port (Empfänger), nachdem alle Blätter ihre Inputs von der Wurzel oder den Connections, aus vorhergehenden Zyklen, abgearbeitet haben.
Dieses Verfahren versagt bei der in Abb. V.2 beschriebenen Konfiguration.

E11, E12, E21, E22 sind In-Ports. A1, A2 sind Out-Ports. E12 ist mit A2, E22 mit A1 verbunden. E11 und E21 sind durch die Wurzel belegt.

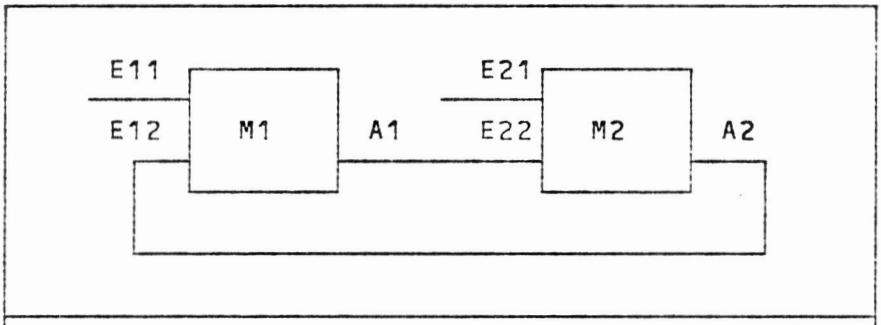


Abb V.2 Rückkoppelungssituation

In der Startsituation werden die Ports E11 und E21 von der Structure_Map belegt. Weil an den Ports E12 und E22 nichts anliegt, können weder M1 noch M2 zu ihren Nachfolgersituationen übergehen. Dadurch werden die Out-Ports A1 und A2 nicht belegt, was wiederum zur Folge hat, daß keine Werte an E12 und E22 anliegen. Es entsteht ein Deadlock. Für solche Konfigurationen sind keine Lösungen bekannt, die das Prinzip " No function in structure " nicht verletzen [5].
Ein zweites Problem stellt der fehlende Takt dar. Die Modelle sind nicht durch ein Zeitsignal, ein Zeitverhältnis oder etwas ähnliches synchronisiert, sie sollen aber einen zeitgleichen Verlauf haben.
Dadurch, daß die Ableitungsschritte der Wurzel von minimaler Länge sind, und diese Auswirkungen sich sofort auf die Blätter übertragen und verarbeitet werden, wird für einen Pseudotakt gesorgt.
Konkreter läßt sich das Problem so formulieren :
Wie lange dauert ein beobachteter Wert eines Modells an, im Vergleich zu anderen Werten die zu anderen Modellen gehören?
Das simple Aggregat aus NAND, NOT und AND (Abb. V.3) soll dies illustrieren.

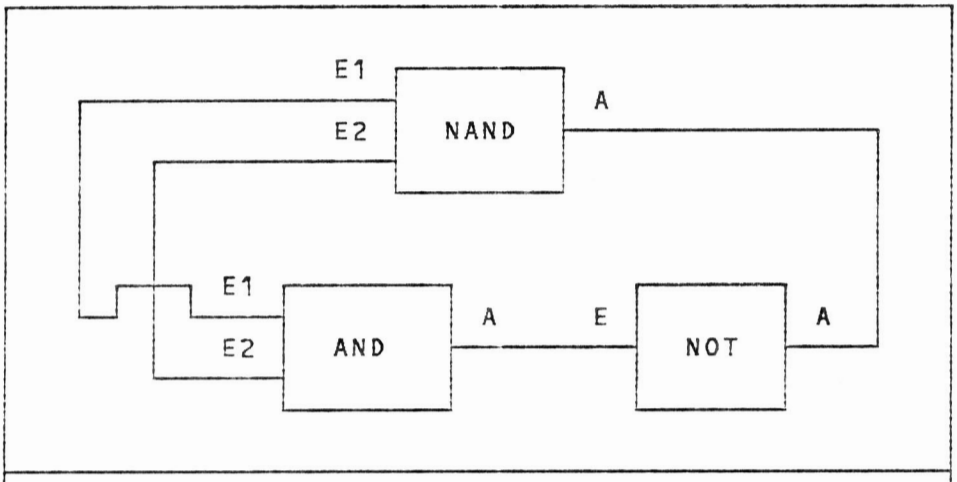


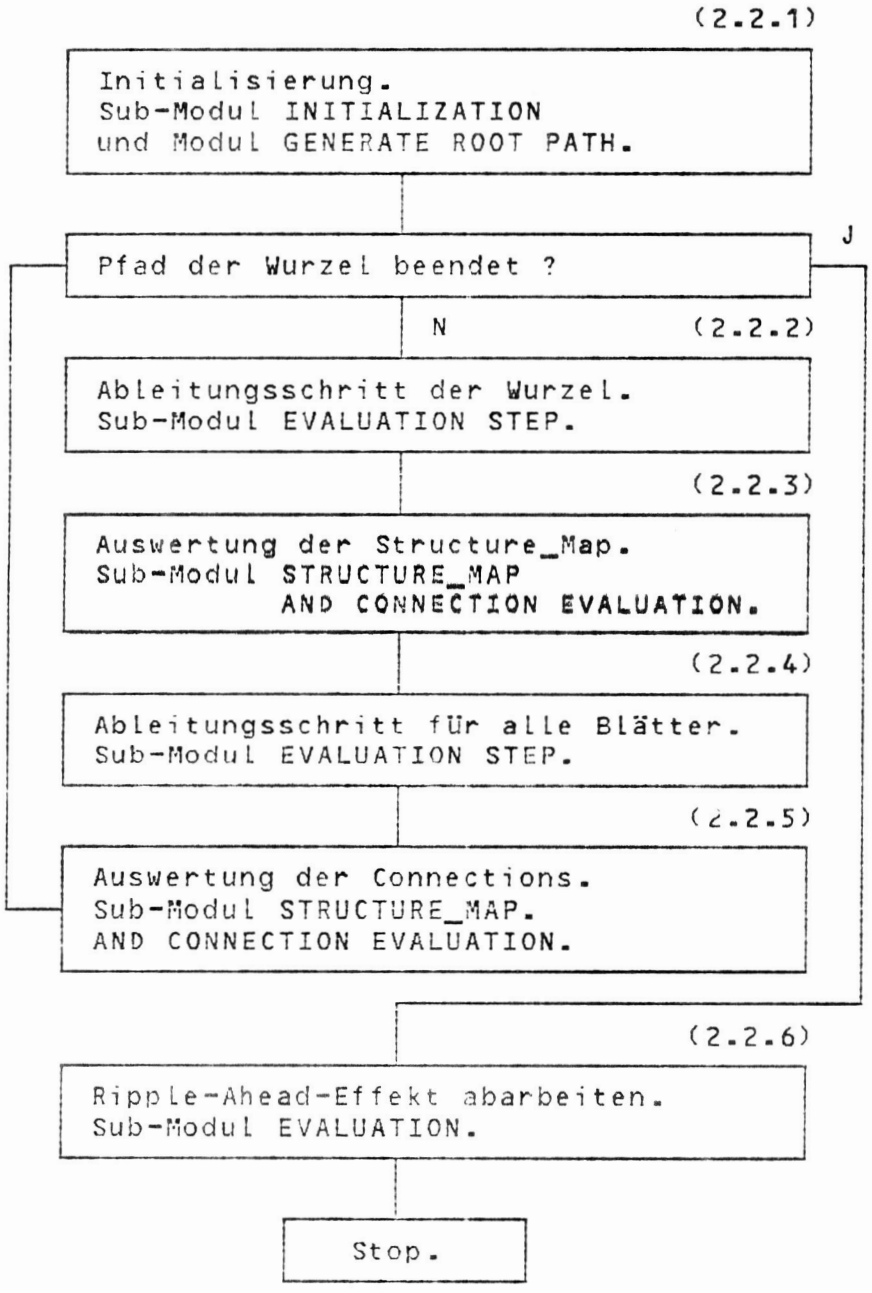
Abb. V.3 Simples Aggregat aus Nand,And und Not.

Das Modell NAND ist die Wurzel, AND und NOT die Blätter.
Die Structure_Map enthält die Verbindungen (E1-Nand E1-AND), (E2-Nand E2-And), (A-And A-Not).
Die Connections sind mit (A-And E-Not) angegeben.
Angenommen, am Ausgang von NAND treten die Werte high, high, high, low auf. Für den Ausgang von AND bedeutet das die Folge low, low, low, high, auf die der Eingang von Not reagieren muß. Der Modellgraph sieht es aber nicht vor, daß E-Not mehrmals auf denselben Input reagiert. Die Aggregatanalyse muß also hier der Eventanalyse vorgreifen und erkennen, daß es sich nur um ein Ereignis handelt.
Angenommen A-Nand nimmt die Werte high, high, low, high, high an. An A-And treten dann die Werte low, low, high, low, low auf. Für E-Not heißt das low, high, low. An A-Not tritt dann high, low, high auf.
Das Problem besteht darin, die Werte am Ausgang von Not zeitlich mit den Werten an A-Nand zu matchen.
Der Pseudotakt, der durch die kleinen Schritte der Wurzel vorgegeben wird, ermöglicht es zu bestimmen, ob Werte auf verschiedenen Ebenen zueinander passen oder ob die Spezifikation inkonsistent ist.
Das Verfahren hat sich an allen Beispielen bewährt, aber es nicht auszuschließen, daß es dennoch Anwendungen gibt, in denen Races oder Hazards auftreten [6].
Damit das Analyseverfahren ohne Backtrack auskommt, müssen die Modelle des Aggregats eindeutig und die Aggregate vollständig sein.
Ein Modell heißt eindeutig, wenn es in seinem Verhalten vollständig durch die In-Portwerte bestimmt wird.

Ein Aggregat ist vollständig, wenn jeder Port in eine Verbindung einbezogen ist. Alle Ports der Wurzel müssen durch die Structure_Map auf die untere Ebene projeziert werden und alle Ports der Blätter entweder von einer solchen Projektion betroffen sein oder in einer Connection vorkommen.

V.2 Das Analyseverfahren

V.2.1 Struktur des Analyseverfahrens



V.2.2 Algorithmen der Analyse

Die Analyse wird im Modul EVALUATE AGGREGATES ausgeführt, welches die im folgenden beschriebenen Sub-Module enthält.

V.2.2.1 Initialisierung

Den Ports durch die Structure_Map oder die Connections zugewiesene Werte, werden in speziellen Datenstrukturen, den Eventhistories gespeichert und von dort an die Ports weiter gegeben. Dies erlaubt es, verschieden schnell ablaufende Modelle oder unstetige Ports auszugleichen.
In der Initialisierungsphase werden die Werte der Eventhistories zurückgesetzt.
Der Pfad der Wurzel wird vom Benutzer im Inspector spezifiziert. Das Modul GENERATE ROOT PATH transformiert die im Cf-Instance-Pfad verwendete Darstellung der Intervalle in eine Listendarstellung.
Wenn ein komplexes Aggregat analysiert wird und der Pfad der Wurzel durch ein übergeordnetes Aggregat spezifiziert worden ist, wird der Pfad unverändert übernommen.
Die Initialisierung wird im Sub-Modul INITIALIZATION ausgeführt.

V.2.2.2 Ableitungsschritt für die Wurzel

Die Wurzel wird in jedem Ableitungsschritt bedingungslos von ihrer aktuellen Situation zur Nachfolgersituation weitergeschaltet. Die Reihenfolge der Situationen ist durch den Pfad der Wurzel eindeutig vorgegeben.
Diese Aufgabe wird im Sub-Modul EVALUATION-STEP ausgeführt.

V.2.2.3 Auswertung der Structure-Map

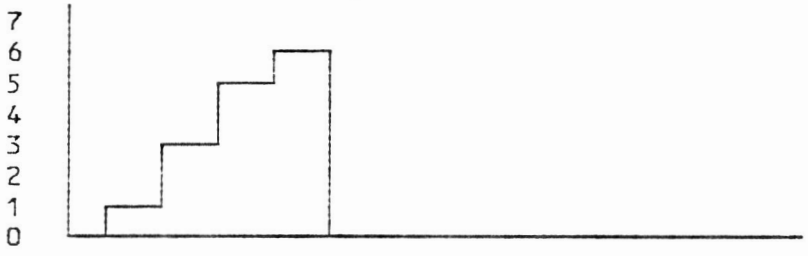
Die Structure_Map bildet Werte, die an Ports der Wurzel anliegen, auf Ports der Blätter ab. Wenn der Verlauf dieser Werte

stetig ist und eine 1:1 Projektion in der Structure_Map spezifiziert wurde, werden sie direkt übergeben.
Andernfalls wird zuerst die angegebene Repräsentationsfunktion auf die Werte angewendet, und dann werden die Werte stetig gemacht die zu geordneten Typen gehören.

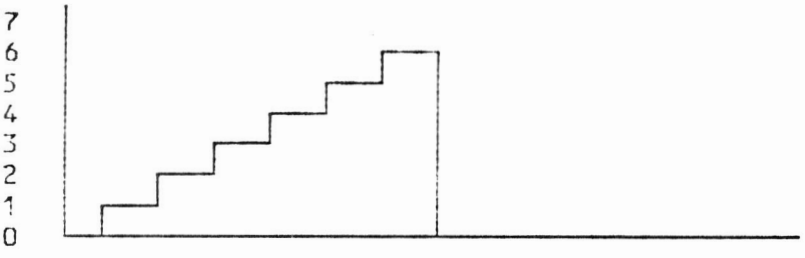
Beispiel :

P1-Wurzel sei ein Port der Wurzel der hintereinander die Werte 1,3,5,6 annimmt.
Die direkte Abbildung gibt die Werte 1,2,3,4,5,6 weiter.
Wenn darauf die Repräsentationsfunktion R1 mit dem Graphen (1 (2)),(2 (4)),(3 (5)),(4 (3)),(5 (3)),(6 (7)) angewendet wird, werden die Werte 2,3,4,5,4,3,4,5,6,7 weitergegeben.

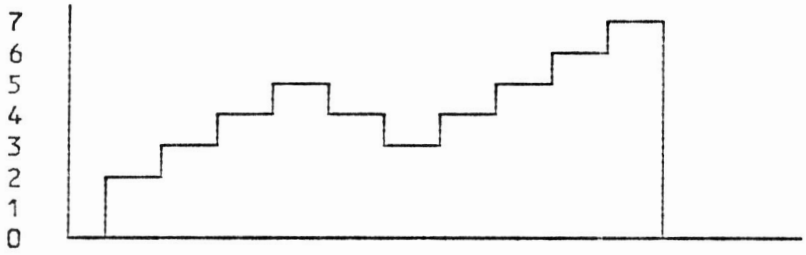
Wertverlauf des Ports der Wurzel :



Wertverlauf mit der direkten Abbildung :



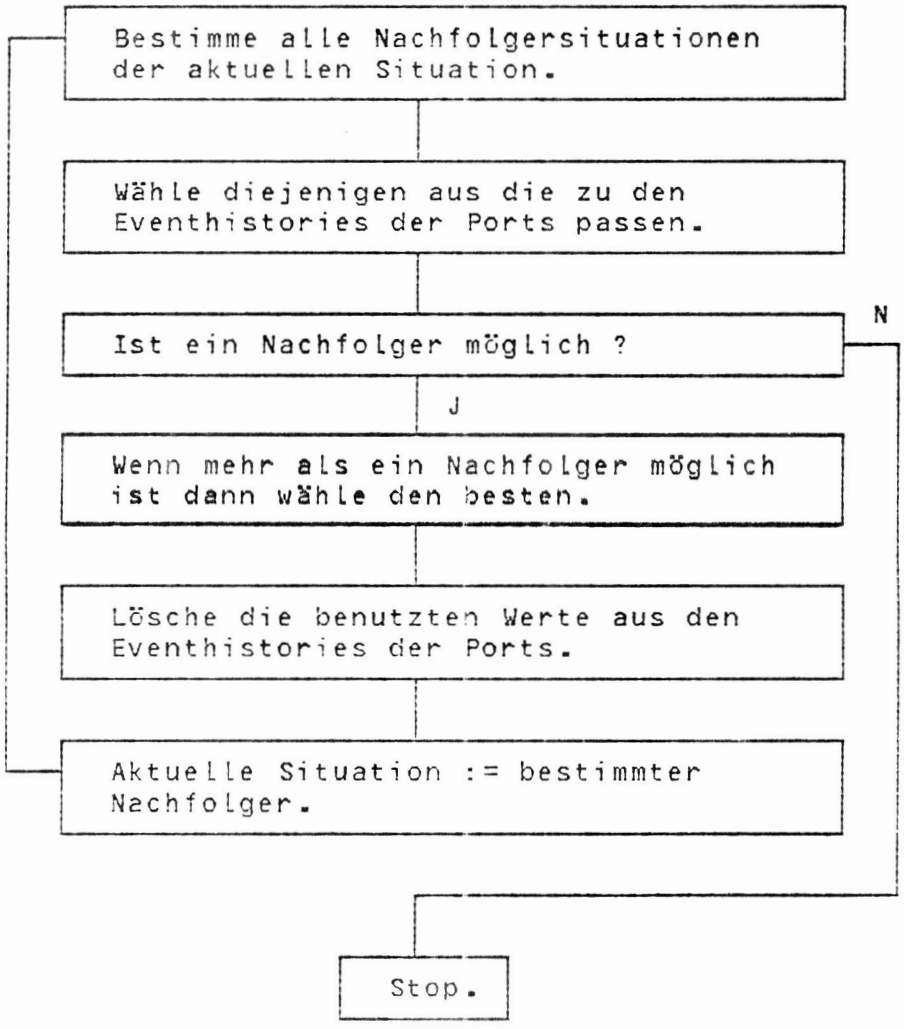
Wertverlauf nach Anwendung der Repräsentation :



Diese Projektion wird im Sub-Modul STRUCTURE-MAP AND CONNECTION EVALUATION ausgeführt.

V.2.2.4 Ableitungsschritt der Blätter

Dieser Teil des Algorithmus ist das zentrale Element der Analyse. Für alle Modelle der unteren Ebene des Aggregats wird dieser Schritt parallel ausgeführt.



In diesem Teil kommen die Eventhistories der Ports zum Tragen. Jede Eventhistory hat zwei Slots. Die Real-Eventhistory enthält die effektiv aufgetretenen Werte. In der Predefined-

Eventhistory werden die Werte vermerkt, die durch die Auswertung der Connections und der Structure_Map vorgegeben sind. Die Real-Eventhistory bildet die Eingabe für diese Auswertung. Nachdem ein Portwert gelesen wurde, um die Transition von der aktuellen Situation zur Nachfolgersituation zu ermöglichen, wird er gelöscht, wenn sein Nachfolgerwert schon bekannt ist. Wenn kein Nachfolgerwert bekannt ist, bleibt der schon benutzte Wert in der Predefined-Eventhistory, da angenommen werden muß , daß er noch gültig ist. Ein Flag merkt sich aber, daß der Wert schon benutzt wurde.
Die Connections und die Structure_Map werden erst ausgewertet, nachdem alle Modelle sich im Graphen soweit wie möglich vorgearbeitet haben. Dadurch kommt es vor, daß in der Predefined-Eventhistory nach der Connectionauswertung, vor dem nächsten Ableitungsschritt, mehrere Werte stehen, von denen der erste schon benutzt wurde. Bei der nächsten Nachfolgerbestimmung werden die Situationen bevorzugt, die mit neueren, noch nicht benutzten Werten arbeiten.
Bei Modellen mit mehreren Ports sind hier natürlich viele Kombinationen möglich und es muß ein Auswahlverfahren angewendet werden.

Folgende Heuristiken werden dazu benutzt :

- Situationen, die Vorbedingungen von Regeln darstellen, werden bevorzugt,
- Situationen die viele PVA beschreiben, werden höher gewichtet,
- Die Nachfolgersituation soll, was die Werte ihrer PVA angeht, nur eine möglichst kleine Differenz aufzeigen.

Die Heuristiken werden auch in dieser Reihenfolge angewendet. Es ist möglich, sich ein Protokoll der Auswahl drucken zu lassen. Wenn *prot-flag* auf T gesetzt ist, wird ein Entscheidungsprotokoll nach Standard-output geschrieben.

Das beschriebene Verfahren wird durch die Sub-Module EVALUATION-STEP und SUCCESSOR-DEFINITION ausgeführt.

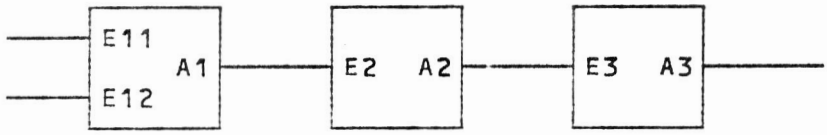
V.2.2.5 Auswertung der Connections

Die Connections werden nach demselben Verfahren wie die Structure_Map im Sub-Modul STRUCTURE-MAP AND CONNECTION EVALUATION ausgewertet. Die Werte werden direkt vom Out-Port an den In-Port weitergegeben, nachdem sie stetig gemacht wurden.

V.2.2.6 Abarbeitung des Ripple-Ahead-Effekts

Die Connections werden nach dem Ableitungsschritt der Blätter ausgewertet. Daten, die sich durch mehrere Modelle hindurcharbeiten müssen (Ripple-Ahead-Effekt), brauchen dazu soviele Ableitungszyklen wie sie Connections benutzten.

Beispiel :



Bis sich eine Veränderung an E11 oder E12 an A3 bemerkbar macht, muß der Auswertungszyklus dreimal durchlaufen werden.

Nachdem der Pfad der Wurzel abgearbeitet wurde, wird im Sub-Modul EVALUATION der Auswertungsschritt für die Blätter so lange aufgerufen, bis jedes dieser Modelle eine stabile Konfiguration erreicht hat.

V.3 Überprüfung und Vervollständigung der Analyse

In der in V.1 und V.2 beschriebenen Analyse muß Rücksicht auf verschieden schnell ablaufende Modelle genommen werden.

Beispiel :

Auf der abstrakten Ebene gibt es einen logischen Pegel mit den Werten low und high. Dieser Pegel wird auf der konkreteren Ebene durch eine Spannung zwischen 0 und 5 Volt dargestellt. Auf der konkreteren Ebene müssen fünfmal so viele Situationen durchlaufen werden, um das Verhalten des Pegels zu simulieren.

Es ist daher kaum möglich, schon während der Analyse festzulegen, welches Cf-Intervall eines Modells zeitgleich ist mit einem anderen Intervall aus einem anderen Modell.
Im Anschluß an die Analyse wird im Modul EVALUATE CONNECTIONS geprüft, ob es für jeden auftretenden Wert auf der einen Seite einer Connection oder Structure_Map eine Entsprechung auf der

anderen Seite gibt.
Gleichzeitig werden die Zeitrelationen zwischen den Cf-Intervallen eingetragen. Wenn Fehler oder Inkonsistenzen auftreten, werden die nicht nachvollziehbaren Verbindungen ausgegeben.
Die genaue zeitliche Kohärenz kann erst in der Eventanalyse berechnet werden. Dies scheint allerdings nur nötig, wenn in den Modellen zeitliche Beschränkungen zwischen Facetten, Preconditions oder Actions zusätzlich zu den implizit geltenden Constraints angegeben wurden, oder wenn die Resultate graphisch dargestellt werden sollen.

V.4 Komplexe Aggregate

V.4.1 Definition

Komplexe hierarchische Strukturen, die aus mehr als zwei Abstraktionsebenen bestehen, sollen nach denselben modularen Prinzipien wie einfache Aggregate aus schon bestehenden Elementen der Hiqualbeschreibung zusammengesetzt werden.
Die rekursive Benutzung des Aggregatkonzeptes bietet sich hier an.
Ein komplexes Aggregat ist eine Menge von Aggregaten, die hierarchisch über gemeinsame Modelle verknüpft sind.
Ein Modell verbindet zwei Aggregate, wenn es im abstrakteren Aggregat Blatt und im konkreteren Aggregat Wurzel ist.

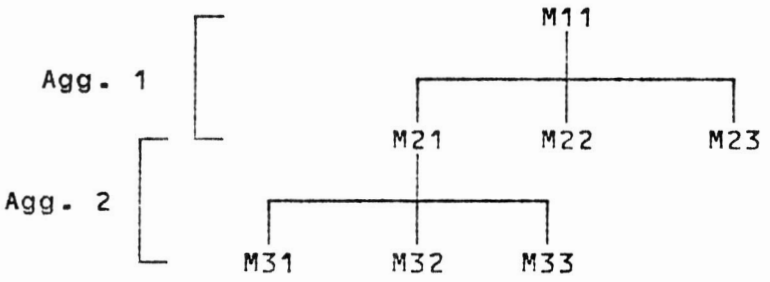


Abb. V.4

Komplexe Aggregate müssen nicht explizit beschrieben werden. Sie können vom System aus allen definierten Aggregaten, die über gemeinsame Modelle verfügen, automatisch zusammengesetzt werden.

V.4.2 Analyse

Aus der Definition der komplexen Aggregate ist ersichtlich, daß die Analyse für simple Aggregate auch auf komplexe Aggregate angewandt werden kann.
Für das Beispiel in Abb. V.4 bedeutet dies, daß das Aggregat 1 wie bisher ausgewertet wird. Der Benutzer spezifiziert einen Pfad für die Wurzel, der dann von den Blättern nachvollzogen wird. Der Pfad des Modells M21, der durch den Pfad des Modells M11 (Wurzel des Aggregats 1) vorgegeben wird, stellt jetzt den Wurzelpfad von Aggregat 2 dar, und wird von den Modellen M31, M32 und M33 nachvollzogen.
Die Auswertungsreihenfolge der Aggregate ist Breadth-first, aber dies ist wegen der Modularität im Prinzip unwichtig.
Komplexe Aggregate werden im Modul COMPLEX-AGGREGATE ausgewertet.

V.5 Grenzen der Analyse

Das Analyseverfahren testet, ob Aktionen der Wurzel in derselben Reihenfolge von den Blättern nachvollzogen werden können. Dies geschieht durch eine Simulation in den Modellen, bei der versucht wird die Aktionen auf den veschiedenen Ebenen schon während der Simulation im richtigen Zeitverhältnis ablaufen zu lassen.
Dadurch wird erreicht, daß das Effect-not-starts-before-cause-Prinzip auf jeden Fall befolgt wird, und die Analyse korrekt und ohne Backtrack arbeitet, wenn die Modelle eindeutig und die Aggregate vollständig sind.
Probleme bereiten der Analyse Ungleichungen, da deren zeitliche Semantik bezüglich der Wertfolgen nicht klar ist.

Beispiel :

Was bedeutet P > 3, mit P ∈ [0 .. 6] ?
heißt das P=4, P=5, P=6,
oder P=4, P=5, P=4,
oder nur P=4 ?

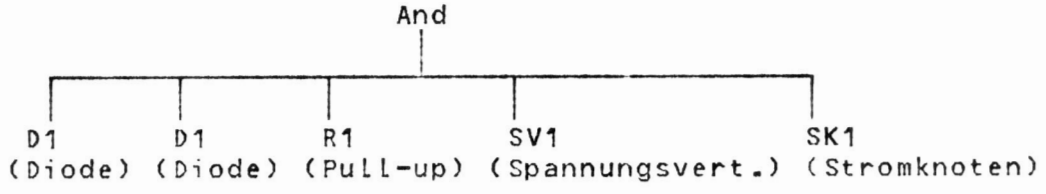
Das gleiche Problem stellt sich auch bei unstetigen Modellen, deren In-Ports von der Modellanalyse stetig gemacht werden. Die hier eingeführten Zwischenintervalle genügen den Anforderungen der Aggregatanalyse nach einzelnen Werten nicht, und

genau wie bei den Ungleichungen, ist ihre zeitliche Bedeutung nicht klar definiert.
Daraus folgt, daß der vorgestellte Algorithmus für vollständige Aggregate aus eindeutigen Modellen, deren In-Ports stetig sind und keine Ungleichungen enthalten, den Anforderungen die an die Analyse gestellt werden, genügt.

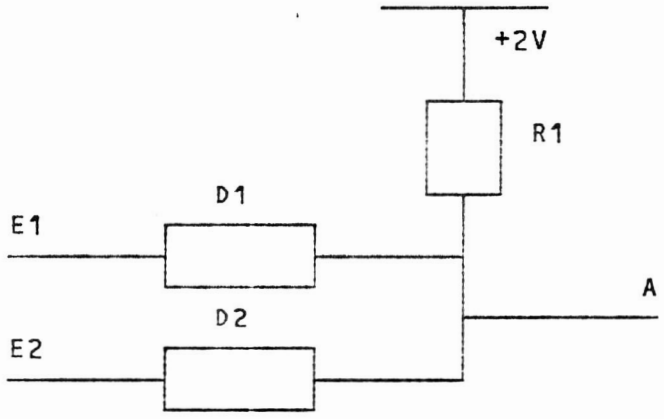
V.6 Beispiel eines Aggregats

Als Beispiel für ein Aggregat, das etwas komplexer ist, soll die Beschreibung eines Zwei-Eingang-Und-Gatters dienen.
Das Und-Gatter wird auf der Hardwareebene durch zwei Dioden und einen Widerstand verfeinert.
In der vorliegenden Implementierung von Hiqual gibt es keine abstrakten Datentypen. Dies kompliziert die Darstellung von Ports, die auf mehrere Werte gleichzeitig reagieren sollen.
In diesem Fall geht es um die Darstellung der Kirchhoffschen Knoten- und Maschenregeln. In realen Netzwerken werden beide Effekte auf demselben Graphen berechnet. Aus den oben genannten Gründen muß der in der Elektrotechnik übliche Strom/Spannungsgraph etwas umständlicher dargestellt werden.
Auf der Hardwareebene unterscheidet man in der Hiqualdarstellung zwei Dioden, einen Pull-up-Widerstand, einen Stromknoten und einen Spannungsverteiler.
Trotz dieser Komplizierung stellt das Aggregat die effektiven Verhältnisse der DDL-Logik gut dar.

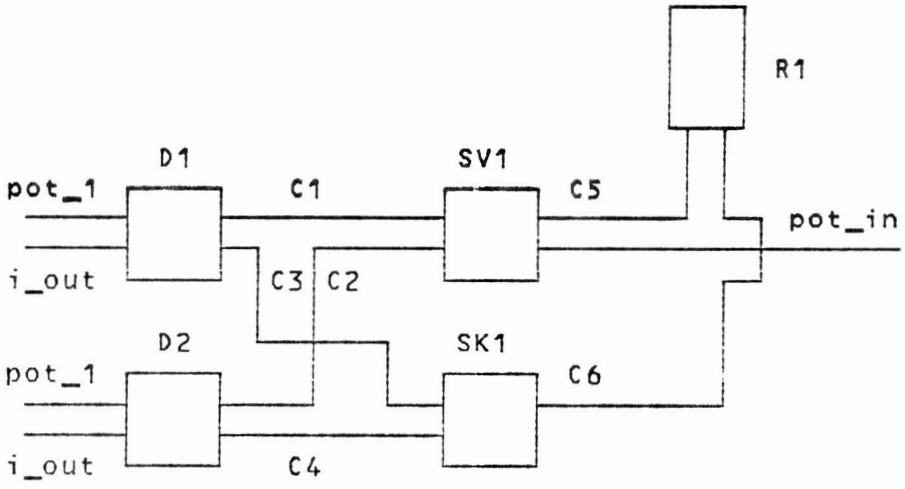
Aggregatschema :



Reales Schaltbild :



Hiqual-Schaltbild :



Hiqual-Beschreibung eines Zwei-Eingang-DDL-And-Gatters.

```
TYPE ddl_and_agg IS AGGREGATION (and,d1,d2,r1,sk1,sv1)

ROOT      and ;

MODELS    and IS bi_and,
          d1,d2 ARE diode,
          r1 IS festwdst,
          sk1 IS stromknoten,
          sv1 IS spannungsverteiler,
          WITH d1@schwellwert = eins,
               r1@festpotential = zwei,
               d2@schwellwert = eins ;

CONNECTIONS    d1@pot_2 WITH sv1@op1,     'C1
               d2@pot_2 WITH sv1@op2,     'C2
               sk1@id1 WITH d1@i_in,      'C3
               sk1@id2 WITH d2@i_in,      'C4
               r1@pot_in WITH sv1@op3,    'C5
               r1@i_in WITH sk1@ir;       'C6

REPRESENTATIONS    rep_1  IS DOMAIN  binary ;
                             RANGE   diode@potential,
                                     diode@strom ;
                             GRAPH   low IS (null,pos),
                                     high IS (zwei,null) ;

                   rep_2  IS DOMAIN  binary ;
                             RANGE
                             spannungsverteiler@potential
                             GRAPH   low IS (eins),
                                     high IS (zwei) ;

STRUCTURE_MAP      a rep_2 (sv1@pot_in),          'STM1
                   e2 rep_1 (d2@pot_1,d2@i_out),  'STM2
                   e1 rep_1 (d1@pot_1,d1@i_out);  'STM3

END ddl_and_agg ;
```

Hiqual-Beschreibung eines Zwei-Eingang-And-Gatters.

```
TYPE bi_and IS MODEL

TYPES  binary IS (low;high) ;

PORTS  IN  e1,e2 ARE binary ;
       OUT a IS binary ;

OKFACET all IS

       r1 IS PRECONDS e1=low,e2=low;
             ACTIONS a=low;

       r2 IS PRECONDS e1=low,e2=high;
             ACTIONS a=low;

       r3 IS PRECONDS e1=high,e2=low;
             ACTIONS a=low;

       r4 IS PRECONDS e1=high,e2=high;
             ACTIONS a=high;

INITIALLY all;

END bi_and;
```

Hiqual-Beschreibung einer Diode mit dem Schwellwert 1V.

```
TYPE diode IS MODEL

TYPES potential IS (null;eins;zwei) ;
      strom is (neg;null;pos) ;

PORTS IN pot_1,pot_2 ARE potential ;
      OUT i_in, i_out ARE strom ;

ATTRIBUTES schwellwert DEFAULT eins IS potential;

OKFACET all IS

        r1 IS PRECONDS pot_1=null,pot_2=null;
              ACTIONS  i_out=null,i_in=null;

        r2 IS PRECONDS pot_1=eins,pot_2=null;
              ACTIONS  i_out=null,i_in=null;

        r3 IS PRECONDS pot_1=zwei,pot_2=null;
              ACTIONS i_out=null,i_in=null;

        r4 IS PRECONDS pot_1=null,pot_2=eins;
              ACTIONS i_out=pos,i_in=neg;

        r5 IS PRECONDS pot_1=eins,pot_2=eins;
              ACTIONS i_out=null,i_in=null;

        r6 IS PRECONDS pot_1=zwei,pot_2=eins;
              ACTIONS i_out=null,i_in=null;

        r7 IS PRECONDS pot_1=null,pot_2=zwei;
              ACTIONS i_out=pos,i_in=neg;

        r8 IS PRECONDS pot_1=eins,pot_2=zwei;
              ACTIONS i_out=pos,i_in=neg;

        r9 IS PRECONDS pot_1=zwei,pot_2=zwei;
              ACTIONS i_out=null,i_in=null;

INITIALLY all;

END diode ;
```

Hiqual-Beschreibung eines Widerstands mit einem auf zwei Volt festgelegten Pol. (Pull-Up-Widerstand)

```
TYPE festwdst IS MODEL

TYPES potential IS (null;eins;zwei);
      strom IS (neg;null;pos);

PORTS IN pot_in IS potential;
         i_in IS strom;

VARIABLES  str IS strom;

ATTRIBUTES festpotential DEFAULT zwei IS potential;

FUNCTIONS gen_strom IS IMPORTED;

OKFACET all IS

        r0 IS PRECONDS pot_in=null,i_in=pos;
              ACTIONS str=gen_strom(pot_in,zwei);

        r1 IS PRECONDS pot_in=eins,i_in=pos;
              ACTIONS str=gen_strom(pot_in,zwei);

        r2 IS PRECONDS pot_in=zwei,i_in=null;
              ACTIONS str=gen_strom(pot_in,zwei);

INITIALLY all;

END festwdst;
```

```
Hiqual-Beschreibung eines Spannungsverteilers mit
einem Eingang und drei Ausgängen.


TYPE spannungsverteiler IS MODEL

TYPES potential IS (null;eins;zwei) ;
      strom IS (neg;null;pos) ;

PORTS IN  pot_in IS potential;
      OUT op1,op2,op3 ARE potential;

OKFACET all IS

        r0 IS PRECONDS pot_in = null;
              ACTIONS op1=null,op2=null,op3=null;

        r1 IS PRECONDS pot_in=eins;
              ACTIONS op1=eins,op2=eins,op3=eins;

        r3 IS PRECONDS pot_in=zwei;
              ACTIONS op1=zwei,op2=zwei,op3=drei;

INITIALLY all;

END festwdst;
```

Hiqual-Beschreibung eines Stromknotens mit zwei Eingängen und einem Ausgang.

```
TYPE stromknoten IS MODEL

TYPES  potential IS (null;eins;zwei);
       strom IS (neg;null;pos) ;

PORTS  IN  id1,id2 ARE strom;
       OUT ir IS strom;

OKFACET all IS

        r1 IS PRECONDS id1=null,id2=null;
              ACTIONS ir=null;

        r2 IS PRECONDS id1=neg,id2=null;
              ACTIONS ir=pos;

        r3 IS PRECONDS id1=null,id2=neg;
              ACTIONS ir=pos;

INITIALLY all;

END stromknoten;
```

VI Eventanalyse

VI.0 Motivation

Die Eventanalyse verwandelt die Verhaltensanalyse des Modellgraphen in eine Zeitanalyse.
Im Modellgraph wird das Verhalten eines Modells durch Cf-Intervalle, die zu Situationen zusammengefaßt sind, beschrieben.

```
TYPE UND_GATTER IS MODEL

     TYPES BINARY IS ( LOW ; HIGH ) ;

     PORTS IN E1,E2 ARE BINARY ;
           OUT A IS BINARY ;

     OKFACET ZUSTAND IS

          R0 IS PRECOND E1=LOW , E2=LOW;
                ACTIONS A=LOW;

          R1 IS PRECOND E1=LOW , E2=HIGH;
                ACTIONS A=LOW ;

          R2 IS PRECOND E1=HIGH , E2=LOW;
                ACTIONS A=LOW ;

          R3 IS PRECOND E1=HIGH , E2=LOW;
                ACTIONS A=HIGH ;

     INITIALLY ZUSTAND ;

END UND_GATTER ;
```

Abb.VI.0 Modell UND-GATTER

Die meisten Modelle haben mehrere PVA. Die Transition von einer Situation zur nächsten ist möglich, wenn sich der Wert eines PVA ändert, oder beim Übergang von einer Facette zur näch-

sten . Beim Facettenwechsel können zwei Situationen mit identischen Assertions aufeinanderfolgen.
Das Ziel der Eventanalyse ist es, die mehrfach beschriebenen identischen Werte eines PVA aus konsekutiven Situationen, die höchstens durch einen Facettenwechsel getrennt sind, zu einem Event zusammenzufassen. Dabei werden die Zeitrelationen, die diese Cf-Intervalle in ihren Situationen beschreiben, auf das Event übertragen und als Relationen zu anderen Events ausgedrückt.

Das Modell UND-GATTER (Abb. VI.0) soll dies illustrieren. Der Pfad folge den Regeln R0, R1, R3, R2.

Dies entspricht den Situationen :

(FACET=ZUSTAND) --> (E1=LOW,E2=LOW)
--> (E1=LOW,E2=HIGH) --> (E1=HIGH,E2=HIGH)
--> (E1=HIGH,E2=LOW) .

Die Eventanalyse soll folgendes Bild liefern :

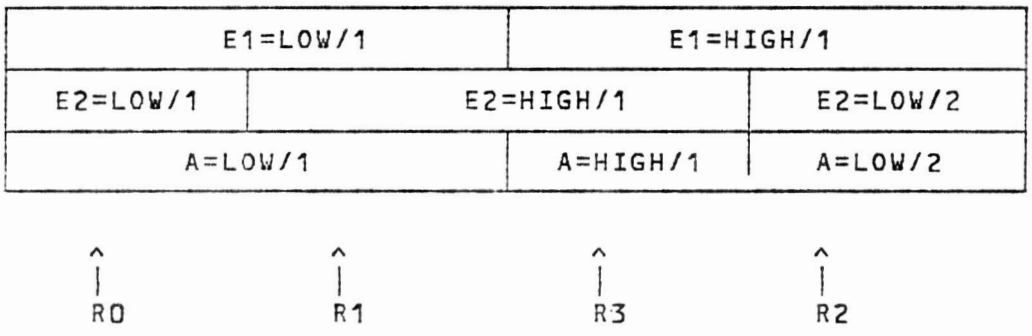


Abb. VI.1

Diese räumliche Darstellung entspricht einer von vielen möglichen zeitlichen Aggregationen.
Die zeitlichen Relationen, die zwischen den Cf-Intervallen einer Regel und der umgebenden oder der zu betretenden Facette gelten, werden auf Events übertragen. Diese Relationen werden durch eine Constraintpropagation vervollständigt und auf Konsistenz geprüft.
Zwischen den Events in unserem Beispiel gelten folgende Relationen :

E1=low/1 —— SI ——> E2=low/1
E1=low/1 —— O ——> E2=high/1
E1=low/1 —— < ——> E2=low/2
E1=high/1 —— > ——> E2=low/1
E1=high/1 —— OI ——> E2=high/1
E1=high/1 —— F ——> E2=low/2
etc.

Die Eventanalyse wurde hier nur für einen Modellpfad beschrieben, sie kann aber auch für Aggregate nach genau dem gleichen Schema ausgeführt werden. Dann werden die Relationen, die in den Connections und der Structure_Map beschrieben sind, hinzugenommen.

VI.1 Analyseverfahren

VI.1.1 Struktur des Analyseverfahrens

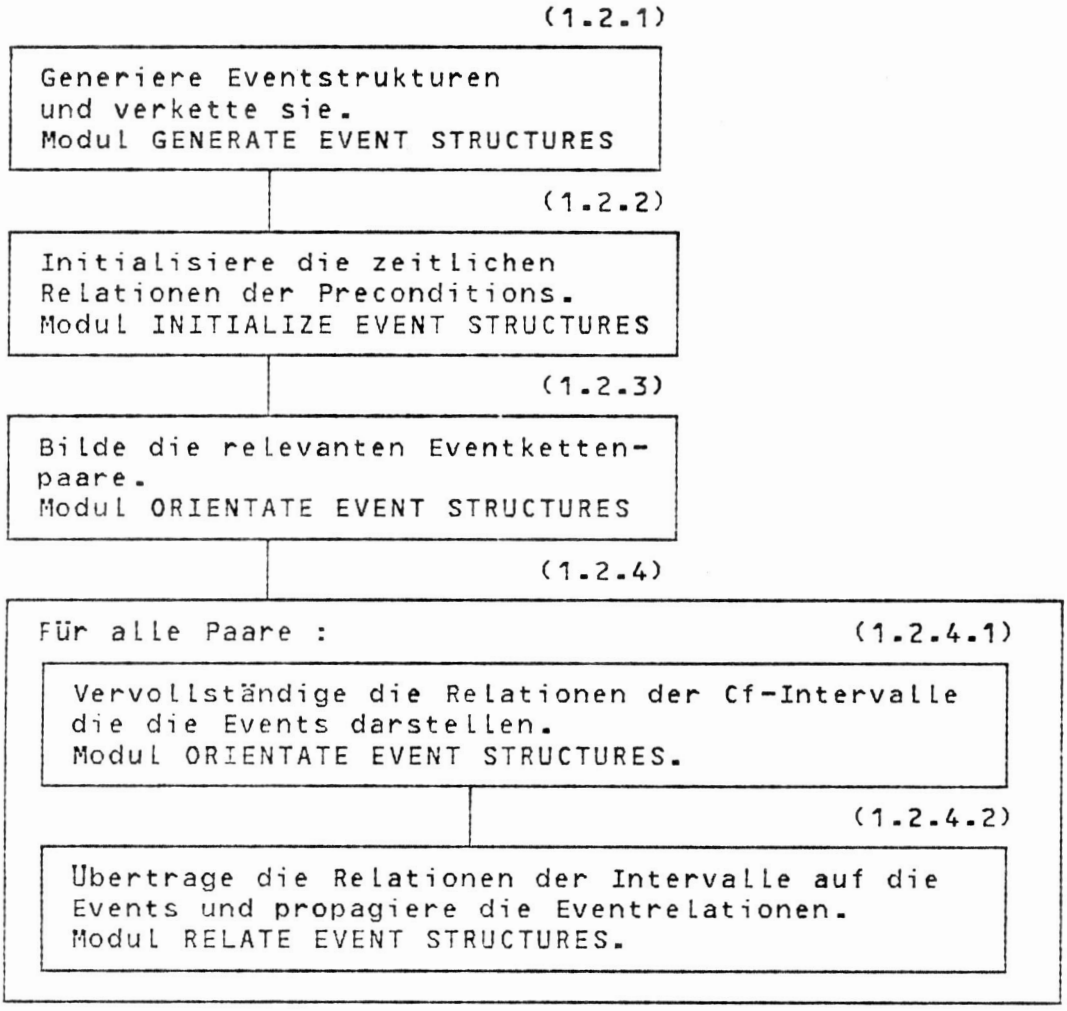


Abb. VI.2

VI.1.2 Algorithmen der Analyse

VI.1.2.1 Erzeugung und Verkettung der Eventstrukturen

Die Analyse wird mit einer Liste von Modellpfaden aufgerufen, die entweder vom Benutzer spezifiziert werden oder aus der Aggregatanalyse stammen.
Jeder Pfad wird für jeden PVA von der ersten bis zur letzten Situation durchwandert und dabei werden die auftretenden Werte (Events) aufgeschrieben.
Für jeden neuen Wert wird eine neue Eventstruktur gebildet, die durch mehrere Cf-Intervalle aus sukzessiven Situationen beschrieben wird.
Die Eventstrukturen, die ein PVA beschreiben, sind über Nachfolgerpointer verkettet.

Beispiel :

UND-GATTER mit demselben Pfad wie oben.
Dazu gehören die Eventketten :

E1: E1=low/1, E1=high/1
E2: E2=low/1, E2=high/1, E2=low/2
A: A=low/1, A=high/1, A=low/2.

VI.1.2.2 Initialisierung

In der Initialisierungsphase werden sämtlichen Eventketten, deren erste Events Facetten oder Vorbedingungen von Regeln beschreiben, die Relationen gleichzeitiger Anfang und gleichzeitiges Ende mitgegeben.
Auf diese Art wird ein Zeitpunkt Null definiert, vor dem keine Werte existieren.
Der gemeinsame Endpunkt der Nicht-Out-Actions beschreibt das Ende des betrachteten Verhaltens und ist an sich keine Notwendigkeit für die Analyse.

VI.1.2.3 Paare von Eventketten

Die Constraintpropagation nach Allen [2] [3] ist ein Algorithmus mit kubischer Komplexität. Der für den interaktiven Betrieb sehr große Zeitbedarf dieses Verfahrens für Relationenmengen die 10-20 Kanten übersteigen , macht es sinnvoll, ein Verfahren zu suchen, bei dem die Propagation öfters mit kleinen Relationenmengen aufgerufen wird, anstatt einmal mit allen Relationen.
Das Problem dabei ist, daß Inkonsistenzen in den Relationen die sich erst weitläufig entwickeln, eventuell nicht mehr aufgedeckt werden, da der Propagation nicht notwendigerweise die ganze Menge der Relationen in einem Mal übergeben wird, die gebraucht würde, um den Widerspruch aufzudecken.
Um die Zeit bei der Erzeugung der Eventrelationen auf ein vernünftiges Maß zu reduzieren, werden immer nur zwei Eventketten gleichzeitig gegeneinander in Relation gesetzt und propagiert.
Die Anzahl der Eventkettenpaare wächst quadratisch mit der Anzahl der PVA. Daher wird hier eine Auswahl getroffen.
Innerhalb eines Modells werden alle Eventkettenpaare genommen; modellübergreifend, in einfachen oder komplexen Aggregaten, werden nur jene Eventkettenpaare genommen, die durch die Connections oder die Structure-Map vorgegeben sind, und alle Facettenpaare, um modellübergreifend die Initialisierung sicherzustellen.
Dieses Verfahren reduziert den Zeitbedarf beträchtlich (in einem Beispiel wurde der Faktor 10 beobachtet).
Der Zeitbedarf wächst jetzt nur noch linear mit der Anzahl der Modelle in einem Aggregat, quadratisch mit der Anzahl der PVA innerhalb eines Modells und kubisch nur noch mit der Länge des zu analysierenden Verhaltens.

VI.1.2.4 Verarbeitung der zeitlichen Relationen

Im Modellgraph und im Modellpfad werden die zeitlichen Relationen zwischen Intervallen angegeben, so wie sie der Benuzter oder die Defaultwerte (siehe Kapitel I.1.2. und I.1.3.) spezifizieren.
Für jeweils zwei Eventketten werden die zeitlichen Informationen aus den Intervallen auf die Events übertragen.

VI.1.2.4.1. Vervollständigung der Relationen zwischen den Intervallen

Im Modellgraph werden Relationen nur innerhalb einer Regel und zur definierenden Facette (inner rule) oder zur definierten Facette (enter rule) eingetragen.
Der Cf-Instance-Pfad enthält zusätzlich noch die in IV.2 beschriebenen Relationen.

Beispiel :

Modell UND-GATTER mit dem Pfad R0, R1, R2, R3

Daraus resultiert die Eventkonfiguration :

Events:	E1=low/1		E1=high/1	
Interv:	E1=low/R0	E1=low/R1	E1=high/R2	E1=high/R3
Events:	E2=low/1	E2=high/1	E2=low/2	E2=high/2
Interv:	E2=low/R0	E2=high/R1	E2=low/R2	E2=high/R3

Abb. VI.3

Um Information über das Verhältnis des Events E1=low/1 zum Event E2=low/1 zu erhalten, müssen die zeitlichen Relationen der Cf-Intervalle E1=low/R1 und E2=low/R0 bekannt sein.
Das Modul ORIENTATE EVENT STRUCTURES erreicht dies durch eine Propagation der Cf-Intervalle der beiden Eventketten gegeneinander. Um den Zeitbedarf zu reduzieren, wird selektiv propagiert.
In diesem Beispiel werden folgende Mengen propagiert :

{E1=low/R0, E1=low/R1, E2=low/R0, E2=high/R1}, {E1=high/R2, E1=high/R3, E2=low/R2, E2=high/R3}, {E2=low/R0, E1=low/R0}, {E2=high/R1, E1=low/R1}, {E2=low/R2, E1=high/R2}, {E2=high/R3, E1=high/R3}.

Dadurch wird erreicht, daß die Cf-Intervalle zweier Events die zeitlich konkurrent sind, also einen nichtleeren gemeinsamen Zeitabschnitt haben, vollständig durch Relationen verbunden

werden.

VI.1.2.4.2 Übertragung der Relationen auf die Events

Das Modul ORIENTATE EVENT STRUCTURES hat die Relationen der Cf-Intervalle vervollständigt. Daraus generiert das Modul RELATE EVENT STRUCTURES Anfangs- und Endinformationen für die Events, aus denen dann die Eventrelationen gewonnen werden.

Modell UND-GATTER mit dem Pfad R0, R1, R2, R3 wie in Abbildung VI.3 .

Die Cf-Intervalle haben die Relationen :

```
E1=low/R0  -- S, SI, = --> E2=low/R0
E1=low/R1  -- MI ————————> E2=low/R0
E1=low/R0  -- M  ————————> E2=high/R1
E1=low/R0  -- F, Fi, = --> E2=high/R1
```

Daraus folgen die Anfangsinformationen :

```
Anf(E1=low/1) = Anf(E2=low/1)
Anf(E1=low/1) < Anf(E2=high/1)
```

und die Endinformationen :

```
End(E1=low/1) > End(E2=low/1)
End(E1=low/1) = End(E2=high/1) .
```

Dazu gehören die Eventrelationen :

```
E1=low/1 -- SI --> E2=low/1
E1=low/1 -- FI --> E2=high/1
E2=low/1 -- S --> E1=low/1
E2=high/1 - F --> E1=low/1 .
```

Die Eventrelationen werden für alle Events generiert, die sich auf den beiden Eventketten "gegenüberliegen". Für das Beispiel in Abb. VI.3 heißt dies, daß für das Paar E1=low/1, E2=high/1 aber nicht für E1=low/1, E2=high/2 Eventrelationen generiert werden.
Alle generierten Relationen beider Eventketten werden der Propagation übergeben, um dort vervollständigt und auf Konsistenz geprüft zu werden.
Die so erzeugten Zeitrelationen werden den Events als Eventrelationen mitgegeben.

VI.2 Probleme der Eventanalyse

Die Zeit, welche für die Eventanalyse benötigt wird, konnte zwar reduziert werden, aber auf Kosten der Aussagekraft des Verfahrens.
Die selektive Propagierung kann Inkonsistenzen übersehen, die eine vollständige Propagation finden würde.
Trotzallem treten schon bei kleineren komplexen Aggregaten (10 Modelle) und recht kurzem beobachteten Verhalten (10 Regeln der Wurzel) Rechenzeiten von mehr als 30 Minuten auf.
Aus diesem Grund gibt es im Observer die Möglichkeit, eine vollständige Propagation im Hintergrund ausführen zu lassen.
Uns ist allerdings kein Beispiel bekannt, in dem hierdurch zusätzliche Informationen aufgetreten wären.

VII Der Observer

Das Ergebnis der Eventanalyse für Modelle, Aggregate und komplexe Aggregate ist ein Constraintnetz (CN). Qualitative zeitliche Intervalle in Form von Eventstrukturen sind in diesem Netz über die bekannten zeitlichen Relationen miteinander verbunden.
Für die Darstellung des CN sind die Eventketten von Bedeutung. Eventketten besitzen alle den gleichen zeitlichen Anfang und ein gleiches Ende. Zusätzlich sind diese Ketten untereinander über zeitliche Relationen miteinander gekoppelt. Diese Kopplung ist verantwortlich für die Dauer der einzelnen Intervalle bzw. Events und richtet sich ausschließlich nach den bestehenden Relationen.
Ausgehend von dieser Betrachtungsweise stellt sich die Frage, wie sich die einzelnen Intervalle einer Kette in Abhängigkeit der jeweils gültigen Relationen zu anderen Intervallen in anderen Ketten orientieren. Orientieren heißt hier, das Verhalten bzgl. Dauer, Anfang und Ende zu anderen Intervallen zu ermitteln und in einer graphisch aufbereiteten Form sichtbar zu machen. Dies umfaßt die Aufgabe des Observers.

VII.1 Eventorientierung

Nach Aufruf des Observers stehen dem Benutzer über ein Menü zwei Modi zur Verfügung um das CN zu orientieren:

1) Orientierung nur bzgl. der Meets-Relation.
   Alle Eventketten werden unabhängig von ihrer Kopplung zu anderen Eventketten ausgegeben. Dies ist für den Fall einer inkonsistenten Relationenmenge gedacht, die eine Orientierung wie in 2) beschrieben nicht zuläßt. Mit dieser Orientierung ist trotzdem eine Inspektion der Events und deren Relationen möglich. Das Verfahren ist das gleiche wie in 2) vorgestellt, jedoch nur mit einer eingeschränkten Relationenmenge, die nur die Meets-Relationen der Events der jeweiligen Kette enthält.
2) Event-Orientierung mit allen Allen-Relationen.
   Dies ist der normale Modus zur Ausgabe des Constraintnetzes, in dem alle Abhängigkeiten berücksichtigt werden. Der Mechanismus ist nachfolgend beschrieben.

VII.1.1 Ableitung von Assertions aus Relationen

Die grundlegende Idee des Orientierungsmechanismus ist, daß jede einzelne Allen-Relation in eine äquivalente Menge von Assertions transformierbar ist, die den Beginn und das Ende zweier Intervalle in Relation zueinander setzen.

Seien I1 und I2 zwei Intervalle mit
A(I1) = Anfang von I1 und
E(I1) = Ende von I1.
Dann gilt:
I1 m I2: E(I1) = A(I2),
I1 < I2: E(I1) < A(I2),
I1 = I2: A(I1) = A(I2) ∧ E(I1) = E(I2),
I1 o I2: E(I1) > A(I2) ∧ E(I1) < E(I2) ∧ A(I1) < A(I2),
I1 d I2: A(I1) > A(I2) ∧ E(I1) < E(I2),
I1 s I2: A(I1) = A(I2) ∧ E(I1) < E(I2),
I1 f I2: A(I1) > A(I2) ∧ E(I1) = E(I2).
Analoges gilt für die jeweiligen inversen Relationen.

Im einfachsten Fall würde jede existierende Relation aus dem CN in eine äquivalente Assertionmenge transformiert werden und anschließend eine Interpretation gesucht, die die entstandene Assertionmenge erfüllt. Eine Interpretation für eine Assertionmenge bedeutet, daß jedem Intervall aus dem CN ein absoluter Anfangs- und Endwert zugeordnet ist, so daß jede existierende Assertion erfüllt ist.
Diese Methode ist jedoch nur anwendbar, wenn zwischen allen möglichen Eventpaaren höchstens eine Relation existiert. Zwischen zwei Events gelten jedoch im allg. eine Menge von Relationen, die sich teilweise sogar widersprechen. Die Auswahl der Assertions aus den bestehenden Relationen muß deshalb sorgfältig geschehen.

I1 ——Rel-list——> I2 beschreibt den Zusammenhang, daß I1 über die Relationenmenge Rel-list in Beziehung zu I2 steht. In Rel-list können prinzipiell alle 13 Allen-Relationen enthalten sein.
Aus der Menge der 13 möglichen Relationen werden 16 Teilmengen definiert, aus denen sich noch eindeutige Informationen über den Anfang bzw. über das Ende zweier relativierter Intervalle ableiten lassen.
Aus dem oben aufgelisteten Zusammenhang zwischen einer zeitlichen Relation und ihrer entsprechenden Assertionmenge, ist ersichtlich, daß einige Relationen bzgl. dem Anfang oder dem Ende die gleiche Information besitzen. Die Relationen o,d und s zwischen zwei Intervallen I1 und I2 haben beispielsweise die Information E(I1) < E(I2) gemeinsam. Die Idee für die Teil-

mengengenerierung ist, alle Relationen, die eine gemeinsame Information besitzen zu einer Menge zusammenfassen, wobei Mengen unterschiedlicher Aussagekraft möglich sind, d.h. je weniger Relationen in einer Menge enthalten sind, desto mächtiger ist ihr Informationsgehalt.
Im folgenden sind sind diese Teilmengen, zusammen mit den daraus jeweils resultierenden Assertions aufgelistet:

REL1: {<}: E(I1) < A(I2).
REL2: {>}: A(I1) > E(I2).
REL3: {m}: E(I1) = A(I2).
REL4: {mi}: A(I1) = E(I2).
REL5: {o}: A(I1) < A(I2) ∧ E(I1) < E(I2) ∧ E(I1) > A(I2).
REL6: {oi}: A(I1) > A(I2) ∧ E(I1) > E(I2) ∧ E(I1) < A(I2).

REL7: {o,di,fi,<,m}: A(I1) < A(I2).
REL8: {=,s,si}: A(I1) = A(I2).
REL9: {oi,d,f,>,mi}: A(I1) > A(I2).
REL10: {o,di,fi,<,m,=,s,si}: A(I1) ≤ A(I2).
REL11: {oi,d,f,>,mi,=,s,si}: A(I1) ≥ A(I2).

REL12: {o,d,s,<,m}: E(I1) < E(I2).
REL13: {=,f,fi}: E(I1) = E(I2).
REL14: {oi,di,si,>,mi}: E(I1) > E(I2).
REL15: {o,d,s,<,m,=,f,fi}: E(I1) ≤ E(I2).
REL16: {oi,di,si,>,mi,f,fi}: E(I1) ≥ E(I2).

Die oben genannte Beziehung I1 ——Rel-list——> I2 wird nach folgenden Regeln in entsprechende Assertions transformiert:

ASS1: Ist Rel-list gleich einer der Mengen aus REL1-REL6, dann übernehme die Assertions, die der entsprechenden Menge zugeordnet sind.

Sind aus ASS1 keine Assertions ableitbar, dann versuche folgende beiden Regeln anzuwenden:

ASS2: Ist Rel-list in einer der Mengen REL7-REL11 als Teilmenge enthalten, dann übernehme die Assertions, die zu der kleinsten Menge aus REL7-REL11 gehören, die Rel-list noch enthält.
ASS3: Analog zu ASS2 für die Mengen REL12-REL16. Vereinige die in ASS2 und ASS3 abgeleiteten Assertions.

Falls keine der drei Regeln anwendbar ist, dann ist aus Rel-list keine widerspruchsfreie Information bzgl. dem Anfang oder dem Ende zweier Intervalle abzuleiten. Dies ist genau dann der Fall, wenn Rel-List einer der fogenden Relationenpaare enthält:

(<,>); (m,mi); (d,di); (o,oi).

Sie generieren jeweils Assertions, die sich gegenseitig ausschließen.
Die entstandenen Assertions werden noch in eine evaluierbare

Form gebracht und für eine spätere Verwendung an I1 gebunden. Nach diesem Schema entsteht aus allen bestehenden relationalen Beziehungen eine Menge von Assertions, für die anschließend als Ergebnis einer Eventorientierung eine Interpretation bestimmt werden muß.

VII.1.2 Berechnung der Eventorientierung

Grundlage für die Eventorientierung ist ausschließlich die zuvor generierte Menge von Assertions. Jede Assertion ist ein in Lisp evaluierbarer Ausdruck, mit den möglichen Komparatoren <,≤,=,≥,> und den Intervallfunktionen (Eventstructure-begin <eventstructure>) und (Eventstructure-end <eventstructure>).
Alle Assertions sind implizit mit UND verknüpft. Die Mindestdauer eines Intervalls ergibt sich aus der maximal notwendigen Printlänge IPL für ein Intervallsymbol.
Zu Beginn der Orientierung wird jedes Intervall initialisiert mit:

Event-structure-begin(<event>) := 1,
Eventstructure-end(<event>) := IPL.

Prinzip der Orientierung:

1) Durchlaufe sukzessive die Menge der Assertions. Ist während des Durchlaufs eine Assertion nicht erfüllt, dann korrigiere die entsprechende Assertion, so daß sie nach der Korrektur als erfüllt gilt.
   Die Korrektur bezieht sich dabei auf die Anfangs oder Endwerte der an der Assertion beteiligten Events.
2) Ist ein Durchlauf beendet und wurde mindestens eine Assertion korrigiert, dann gehe nach 1),
   sonst terminiere mit einer erfolgreichen Orientierung.

Prinzip einer Assertionkorrektur:
Der Beginn und das Ende eines in einer Assertion beteiligten Intervalls darf nur nach oben korrigiert werden, d.h. selektiere das entsprechende Event aus der Assertion, das nach diesem Kriterium die Assertion bzgl. seinem Anfang oder Ende verletzt. Vergrößere den Anfangs- bzw. Endwert um den Betrag, der zur Erfüllung der Assertion notwendig ist. Wurde der Anfang eines Intervalls nach oben korrigiert, dann führe ein Durationtest durch. Prüfe ob die Intervallmindestdauer noch gilt. Setze gegebenenfalls das Ende des Intervalls zu Anfang(I) + IPL.

Ist die zugrundeliegende Assertionmenge konsistent, also erfüllbar, dann terminiert dieses Verfahren mit einer korrekten Orientierung.
Inkonsistete Assertionmengen treten nur auf, wenn das entspr.

CN inkonsistent ist. In diesem Fall würde die Orientierung nicht terminieren. Eine inkonsistente Assertionmenge ist jedoch während der Orientierung nach einem einfachen Verfahren erkennbar:
Anfangs- und Endwerte von Intervallen werden bei der Korrektur nur nach oben verschoben. Bei einer inkonsistenten Assertionmenge hat eine Korrektur von bestimmten Assertions die Folge, daß andere wieder unerfüllt und somit wieder korrigiert werden. Es läßt sich jedoch eine obere Schranke für die höchst zulässigen Anfangs- und Endwerte angeben und zwar:

Anzahl aller Events aus dem CN * IPL.

Diese Schranke entspricht dem Fall, daß alle Intervalle im CN in einer Eventkette auftreten, wobei jedes Event die Ausdehnung IPL besitzt. Im CN ergibt sich die Zahl der Eventketten aus der Zahl der beteiligten PVA (n) plus der Zahl der beteiligten Modelle (m); jedes Modell ist für eine Facettenfolge verantwortlich. Zu jedem Event E-v aus einer Eventkette gibt es in allen anderen Eventketten jeweils ein anderes Event E-j, so daß sich E-v und E-j in einem bestimmten zeitlichen Abschnitt überdecken. Der Korrekturmechanismus für die Assertions hat die Eigenschaft, die Anfangs- und Endwerte der jeweiligen Events nur um soviel zu erhöhen, wie unbedingt notwendig ist. Dies bedeutet, daß eines der sich zeitlich überdeckenden Events nur die Ausdehnung IPL besitzt. Sei E-p dieses Event mit der Minimalausdehnung IPL, dann existiert auf der gleichen oder auf einer anderen Eventkette ein Event E-q, so daß ENDE(E-q)=ENDE(E-p)+IPL.
Anschaulich heißt das:
Wähle innerhalb der vollständig orientierten Eventketten einen beliebigen Zeitpunkt t und vermerke alle zu diesem Zeitpunkt gültigen Events. Dann gibt es zum Zeitpunkt t+IPL mindestens ein Event aus der vermerkten Menge, das bis zu diesem Zeitpunkt von einem anderen Event abgelöst worden ist.
Für die Orientierung bedeutet das: Es gibt eine konsekutive Folge von Events (nicht notwendigerweise für das gleiche PVA) in der jedes Event nur die Mindestausdehnung IPL besitzt. Gibt es nur eine Eventkette im CN, dann kann sie nur die maximale Ausdehnung der oben genannten Schranke annehmen. Im Falle von n+m Eventketten ist die maximale Länge aller Eventketten wegen der zeitlichen Parallelität auf jeden Fall kleiner als diese Schranke.
Ist die Schranke überschritten, dann liegt eine inkonsistente Assertionmenge vor und somit auch eine inkonsistente Relationenmenge im CN.
Auf diese Weise können ohne eine Propagation unter Umständen Inkonsistenzen erkennbar sein, die die Eventanalyse nicht aufdecken konnte.
Allerdings ist es nicht zwingend, daß aus einer inkonsistenten Relationenmenge auch eine inkonsistente Assertionmenge entsteht. Wegen des Ausschlusses einer Relationenmenge zwischen zwei Events, die einer der Paare (<,>); (m,mi); (o,oi) oder (d,di) enthält, kann möglicherweise gerade eine an einer Inkonsistenz beteiligte Relation ausgeschlossen werden.

Der Observer berechnet für eine erfüllbare Assertionmenge nur genau eine Interpretation. Andere mögliche Interpretationen sind nicht abrufbar. Es besteht zwar die Möglichkeit das Maß für eine Eventüberlappung anzugeben, dies liefert aber nur eine Variante der ursprünglichen Interpretation.

VII.2 Weitere Features

Neben der graphischen Ausgabe eines CN nach der Eventorientierung gibt es noch einige andere abrufbare Operationen. Z.B.:

- Aufruf der Propagation für ein CN.
- Hardcopy des gesamten CN nach der Eventorientierung.

Die Beschreibung aller verfügbaren Operationen mit ihren Optionen ist mit der HELP-Taste innerhalb des Observers abrufbar. Auf sie wird deshalb hier verzichtet.

VIII Benutzeranleitung für Hiqual

VIII.0 Implementierung

Die vorliegende Version von Hiqual wurde auf einer Symbolics 36xx in Zetalisp implementiert und läuft zur Zeit unter der Betriebssystemversion 6.0 .
Sie benötigt inklusive der Servicedateien und der Beispiele 611 Blöcke Speicheplatz. Dies entspricht etwa 1.5 Megabyte.

VIII.1 Definition von Modellen und Aggregaten

Modell- und Aggregattypen werden über den Standardeditor [7] als Textdateien eingegeben.
Ihre Definitionen werden jeweils als einzelne Dateien abgespeichert. Aggregatdateinamen müssen folgende Form haben :
>HIQUAL>DATA>AGGREGATES><aggregatname>.HIQUAL.<vers-nr> .
Modelldateien müssen einen Namen der Form :
>HIQUAL>DATA>MODELS><modellname>.HIQUAL.<vers-nr> haben.
Die Syntax und die Anforderungen der statischen Semantik für Modelltypen und Aggregattypen werden in den Kapiteln II.1 und II.2 erläutert.

VIII.2 Definition von Funktionen und Typen

Standardfunktionen und oft benutzte Typen können in der Datenbasis von Hiqual abgespeichert werden.
Dazu werden im Lisp-Interpreter die Funktionen /hiqual-function-define/ und /hiqual-type-define/ benutzt. Die Syntax für die Parameter dieser Funktionen ist in den Kapiteln II.3 und II.4 erklärt.
Die Hiqualfunktionen werden in Dateien abgespeichert für die die Namenskonventionen
>HIQAUL>DATA>TYPES><typname>.HIQUAL.<vers-nr> resp.
>HIQUAL>DATA>FUNCTIONS><funkt-name>.HIQUAL.<vers-nr> gelten.
Die Definitionen in den Dateien können editiert und verändert werden.

VIII.3 Definition von Batchjobs

Häufig benutzte Modellinstanzen und Aggregatinstanzen können in Dateien zusammengefaßt werden. In diese Dateien werden die Aufrufe für die Funktionen /hiqual-model-instance/ und /hiqual-aggregate-instance/ geschrieben. Sobald diese Dateien geladen werden, werden die entsprechenden Instanzen generiert. Diese Dateien heißen Sourroundings und werden im Directory >HIQUAL>DATA>SOURROUNDINGS><name>.HIQUAL.<vers-nr> abgelegt. Anstelle der Funktionen /hiqual-model-instance/ und /hiqual-aggregate-instance/ können auch die Kürzel /hmi/ oder /hai/ benutzt werden.

VIII.4 Verarbeitung von bestehenden Definitionen

Zur Erzeugung von Modellinstanzen, Aggregatinstanzen, Modellpfaden, Eventstrukturen und komplexen Aggregaten, sowie zu deren Auswertung, bietet Hiqual ein eigenes Window an.
Das Hiqual-Lisp-Interaction-Window (HLIW) arbeitet menüorientiert. Das HLIW wird nach dem Laden von Hiqual (mit LOGIN HIQUAL) mit SELECT K und dem Aufruf der Funktion /hw/ aktiviert.
Das Kommandomenu im HLIW bietet alle Möglichkeiten an, die zur Verarbeitung von Hiqualelementen definiert sind.

Optionen :

Modell Instanz erzeugen :

Der Benutzer wählt per Menü aus der Liste der definierten Modelltypen ein Modell aus, und gibt den Namen der neuen Instanz sowie etwaige Attributbeschränkungen an. Es wird eine Instanz des Types erzeugt und der Modellgraph wird expandiert.

Aggregat Instanz erzeugen :

Der Benutzer wählt aus der Liste der Aggregattypen einen aus, gibt den Namen der neuen Instanz an, sowie die Namen der Modellinstanzen, die in die Aggregatinstanz eingefügt werden sollen.

Inspector aufrufen :

Es wird eine Modellinstanz, und ein Pfad oder Modellgraph ausgewählt. Der Pfad oder Graph wird dann auf dem Inspector ausgegeben.
Im Inspector kann durch die HELP-Taste eine Liste der Optionen ausgegeben werden. Der Inspector sollte nicht durch SELECT K verlassen werden, da dann die Kontrolle im Inspector bleibt und der Prozess blockiert wird.

Aggregat auswerten :

Für eine definierte Aggregatinstanz wird ein Wurzelpfad im Inspector angegeben. Die Aggregatanalyse testet dann, ob dieser Pfad von den Blättern, unter den angegebenen Spezifikationen, nachvollzogen werden kann. Auftretende Inkonsistenzen werden gemeldet.

Directory Wartung :

Aus den Directories, die die Aggregat-, Modell-, Funktions-, Typ- oder Sourroundingdefinitionen enthalten, können Dateien gelöscht werden.
Diese Dateien werden nur zum Löschen vorgemerkt. Effektiv gelöscht werden sie erst beim nächsten Expunge. Das Autoexpungeintervall ist auf einen Tag festgelegt.

Modellpfad erzeugen :

Für ein Modell wird im Inspector ein Cf-Instance-Pfad spezifiziert.

Umgebung erzeugen :

Aus dem Directory Sourroundings wird eine Datei geladen. Dabei werden die enthaltenen Funktionsaufrufe ausgeführt.
Mit dieser Option werden Batchjobs gestartet. Auftretende Parsfehler werden zwar gemeldet, aber die Ausführung wird nicht unterbrochen. Deshalb sollten Batchjobs nur mit korrekten Definitionen aufgerufen werden.

Definierte Elemente :

Auf dem Bildschirm erscheint eine Liste der bisher definierten Modellinstanzen, der Aggregatinstanzen mit ihren Aufrufparametern, der komplexen Aggregate und der beendeten Hintergrundpropagationen.

Komplexes Aggregat auswerten :

Aus allen definierten Aggregatinstanzen werden diejenigen ausgewählt , die in die Konzeption des komplexen Aggregats einbezogen werden.
Wenn mehrere Hierarchien möglich sind, muß der Benuzter eine auswählen. Für die Wurzel des komplexen Aggregats wird im Inspector ein Pfad spezifiziert.

Aufruf des Observers :

Im Observer kann man sich Eventstrukturen von Modellen, Aggregaten oder komplexen Aggregaten ansehen.
Im Observer gibt die HELP-Taste Auskunft Über bestehende Optionen. Der Observer sollte, wie der Inspector, nicht mit SELECT K verlassen werden.

Eventstrukturen erzeugen

Für den Cf-Instance-Pfad eines Modells, oder den Auswertungspfad eines einfachen oder komplexen Aggregats können Eventstrukturen erzeugt werden.
Achtung! Die Eventstrukturerzeugung hat einen kubische Aufwand und nimmt ohne weiteres für ein Modell mit 3 PVA und einem Pfad aus 16 Regeln eine Viertelstunde in Anspruch.

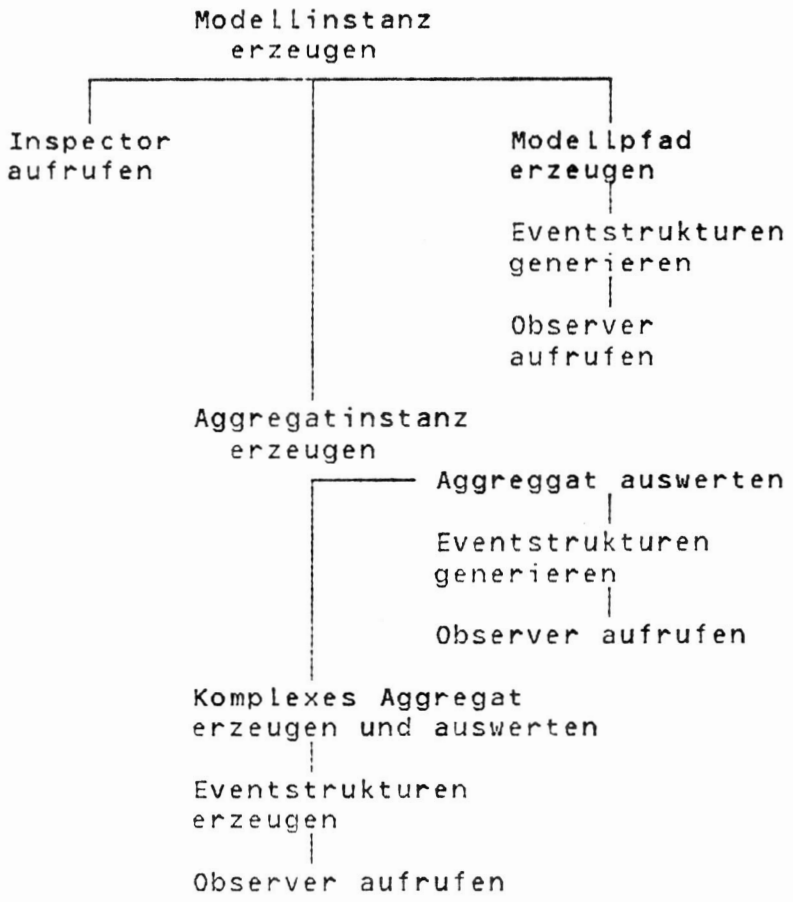
Existierende Files :

Eine Datei aus den Directories Models, Aggregates, Types, Funktions oder Sourroundings wird auf dem Bildschirm ausgegeben.

Exit :

Das KommndomenÜ des HLIW wird deaktiviert. Das Window arbeitet jetzt wie ein Lisp-Interpreter. Das KommandomenÜ wird mit der Funktion /hw/ reaktiviert.

VIII.5 Reihenfolge der Operationen



IX Quellen.

[1]: Hans Voss (1985):
Representing and Analyzing Time and Causality
in HIQUAL Models,
Informatik Fachberichte 118, Seite 259-271,
Springer Verlag

[2]: James F. Allen (1983):
Maintaining Knowledge about
Temporal Intervals,
CACM, Vol. 26, No. 11, Seite 832-843.

[3]: Gebhard Przyrembel,
Dieter Zöllner,
Ewald Matheis (1986):
Der Propagierungsalgorithmus in HIQUAL,
Studienarbeit, Fachbereich Informatik,
Universität Kaiserslautern.

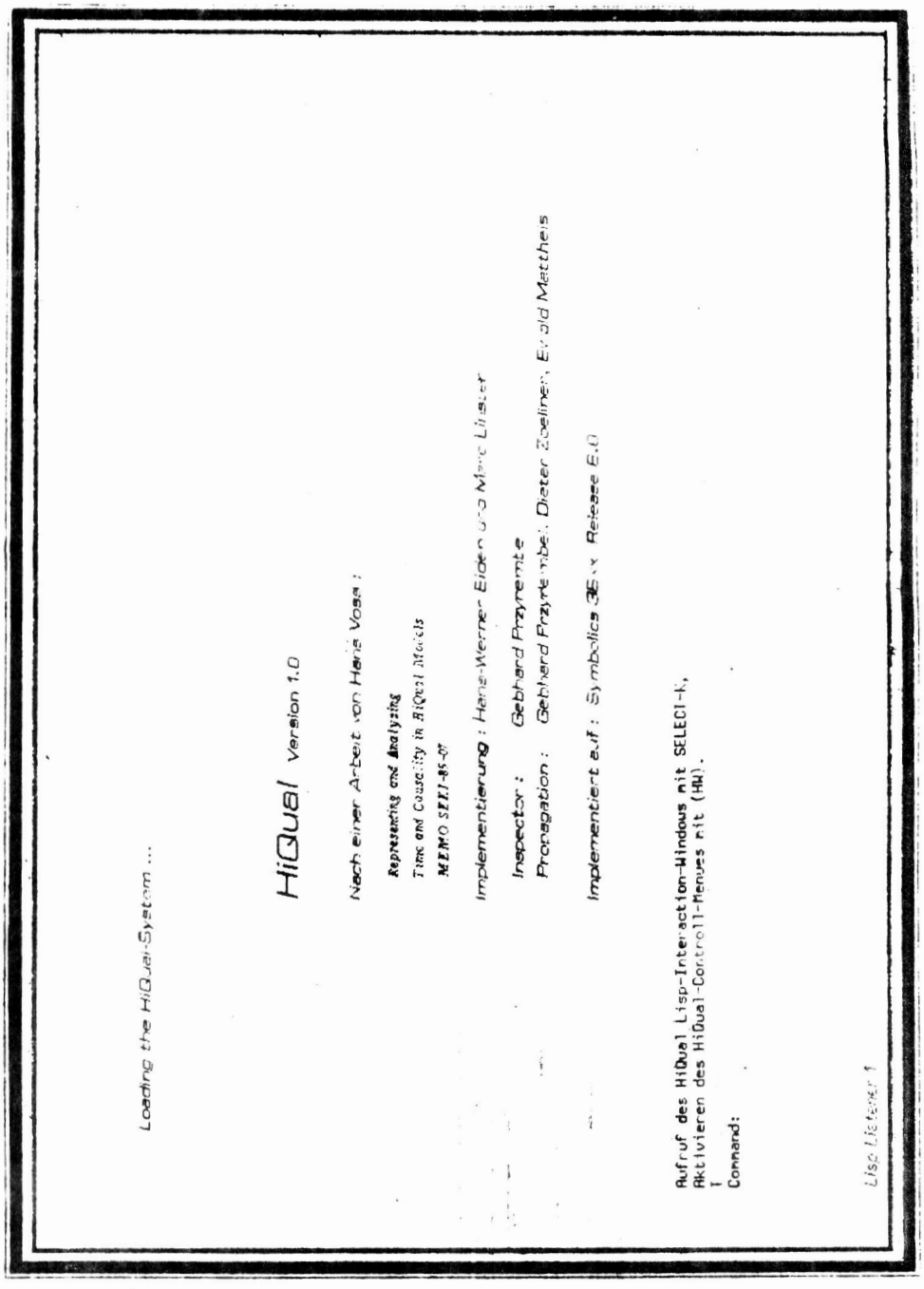
[4]: Gebhard Przyrembel (1986):
Der HIQUAL-Inspector,
Teil der in [3] genannten Studienarbeit.

[5]: Johann De Kleer, John S. Brown,
A Qualitative Physics Based On Confluences,
AI-Journal 1984.

[6]: Samuel C. Lee,
Digital Circuits And Logic Design,
Prentice Hall, 1976.

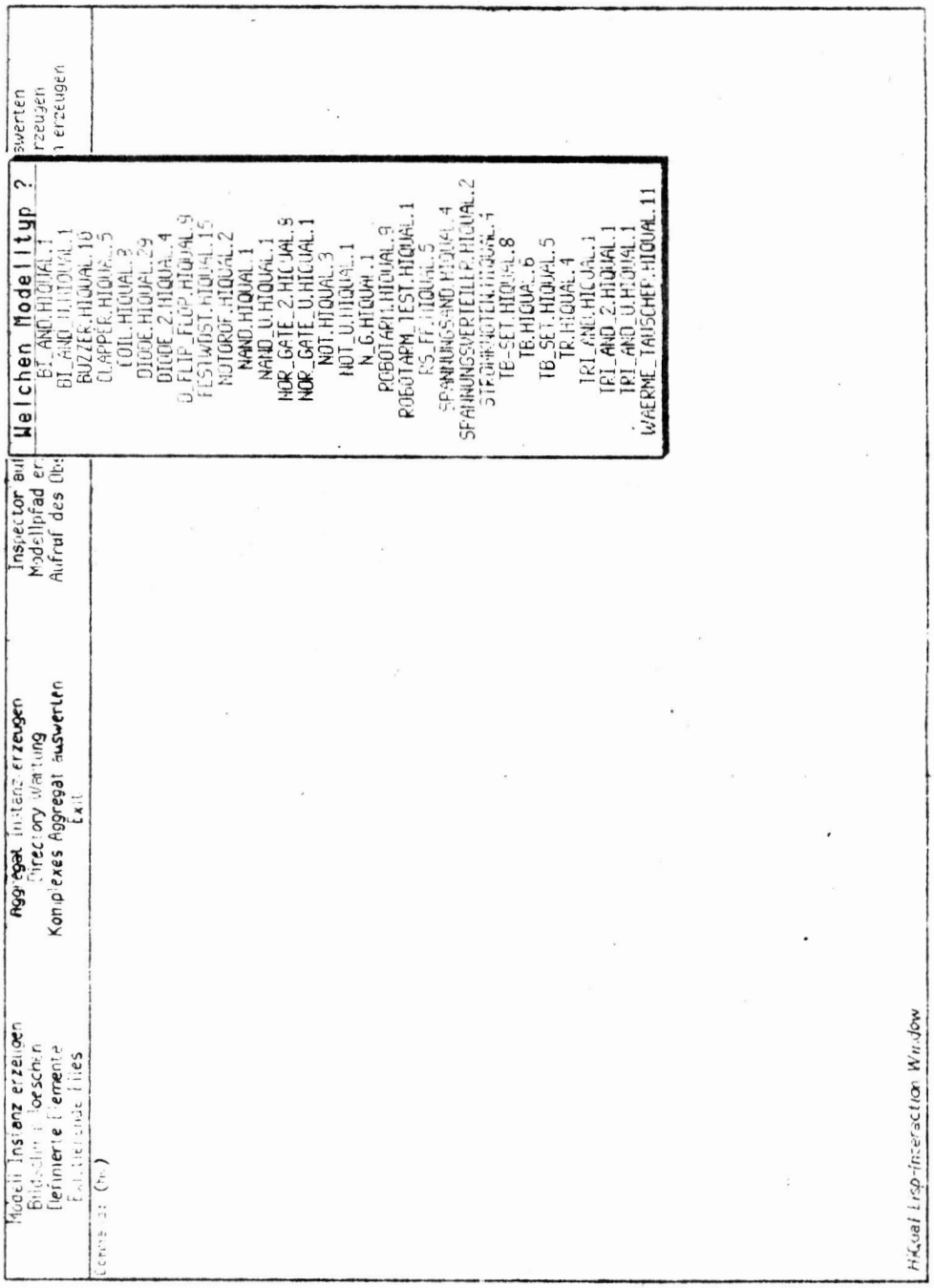
[7]: Text Editing And Processing,
Symbolics Lisp Machine Manual 3.

X Beispiel einer Analyse.



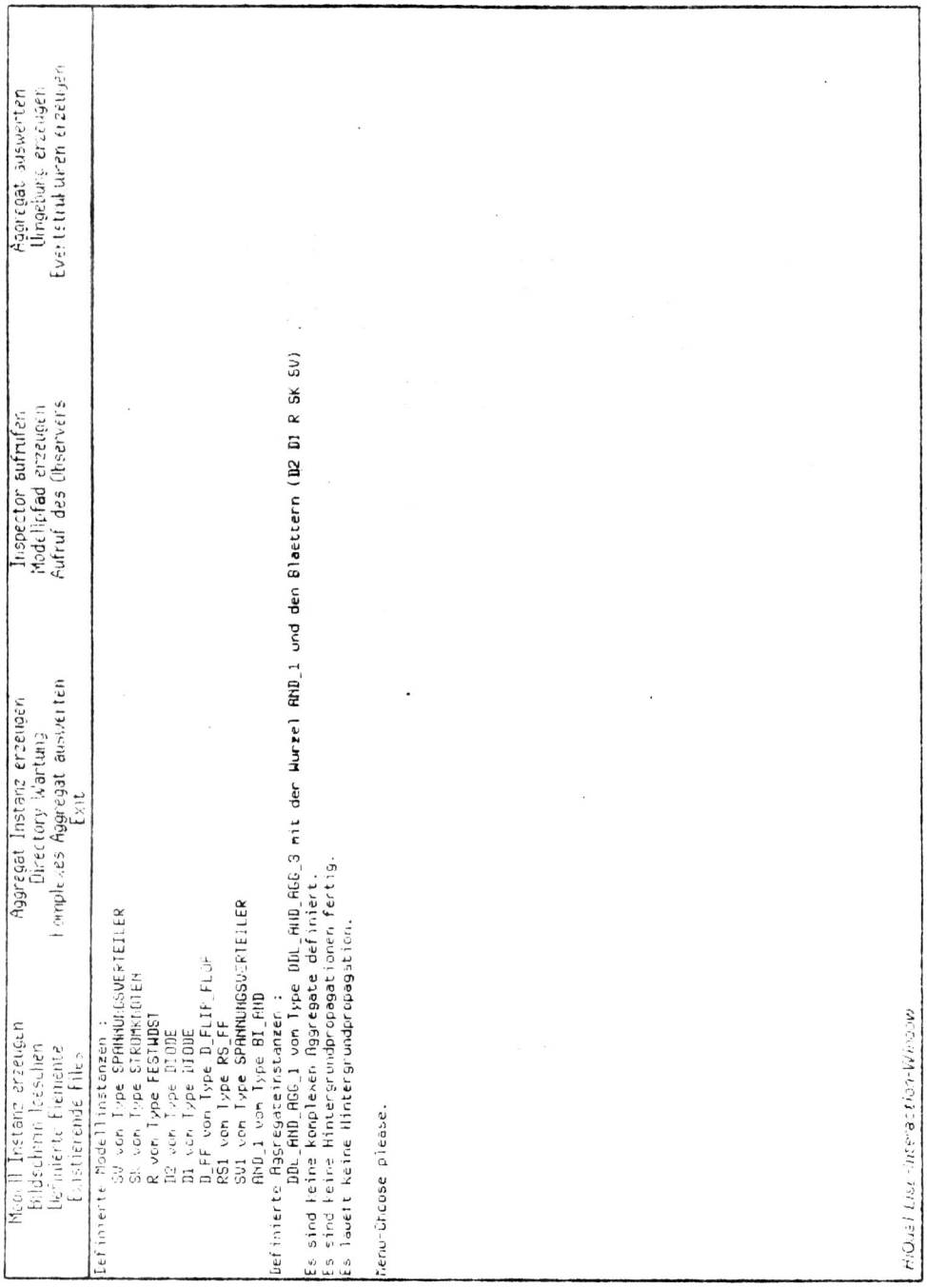
Zustand nach dem Laden des Systems mit LOGIN HIQUAL.

Kommandomenu nach dem Selektieren des Hiqual-Lisp-Interaction-Windows mit SELECT K.

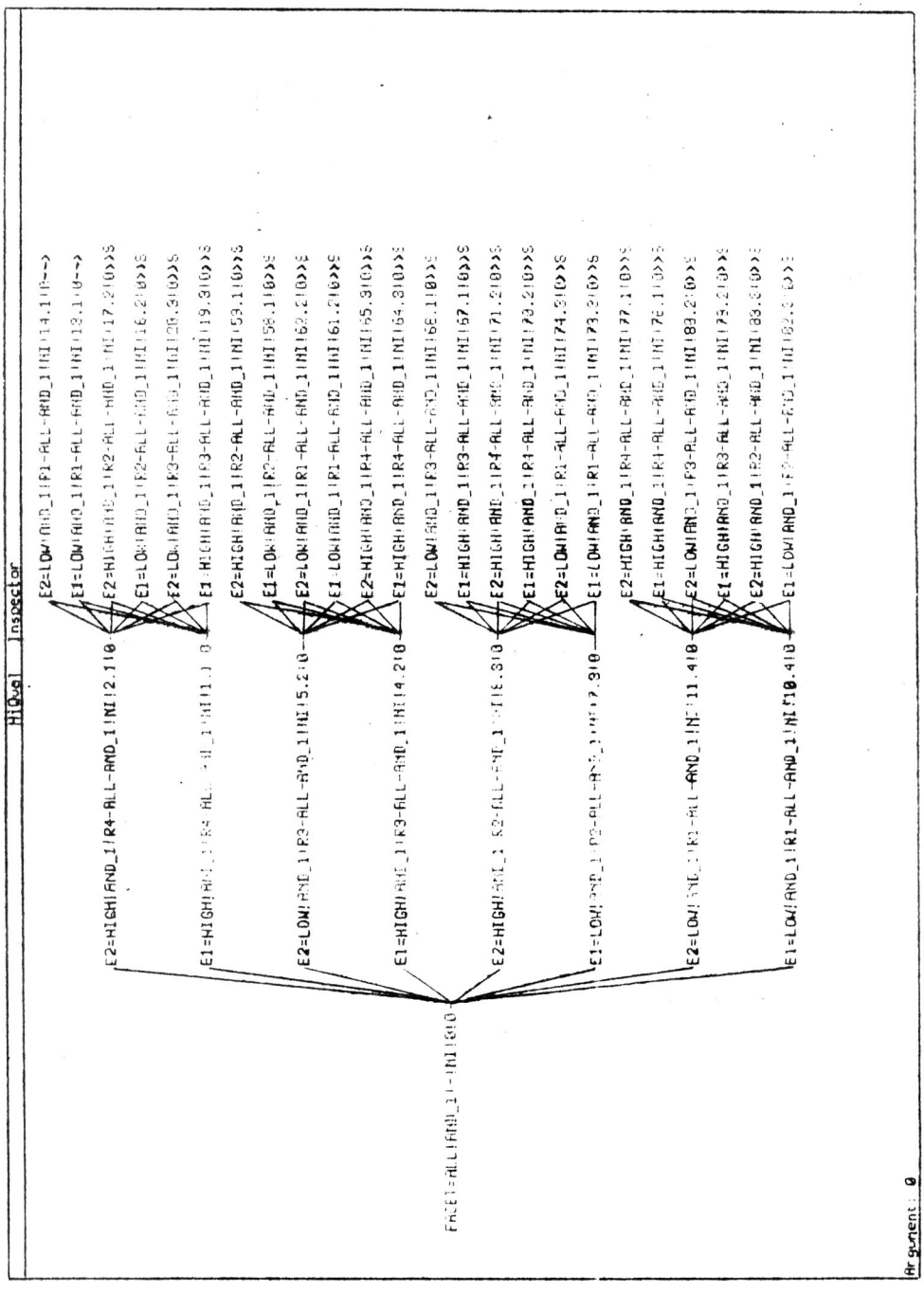


Das Kommandomenü wurde mit (HW) aktiviert.
Sichtbar ist der Zustand nach der Option MODELL INSTANZ ERZEUGEN.

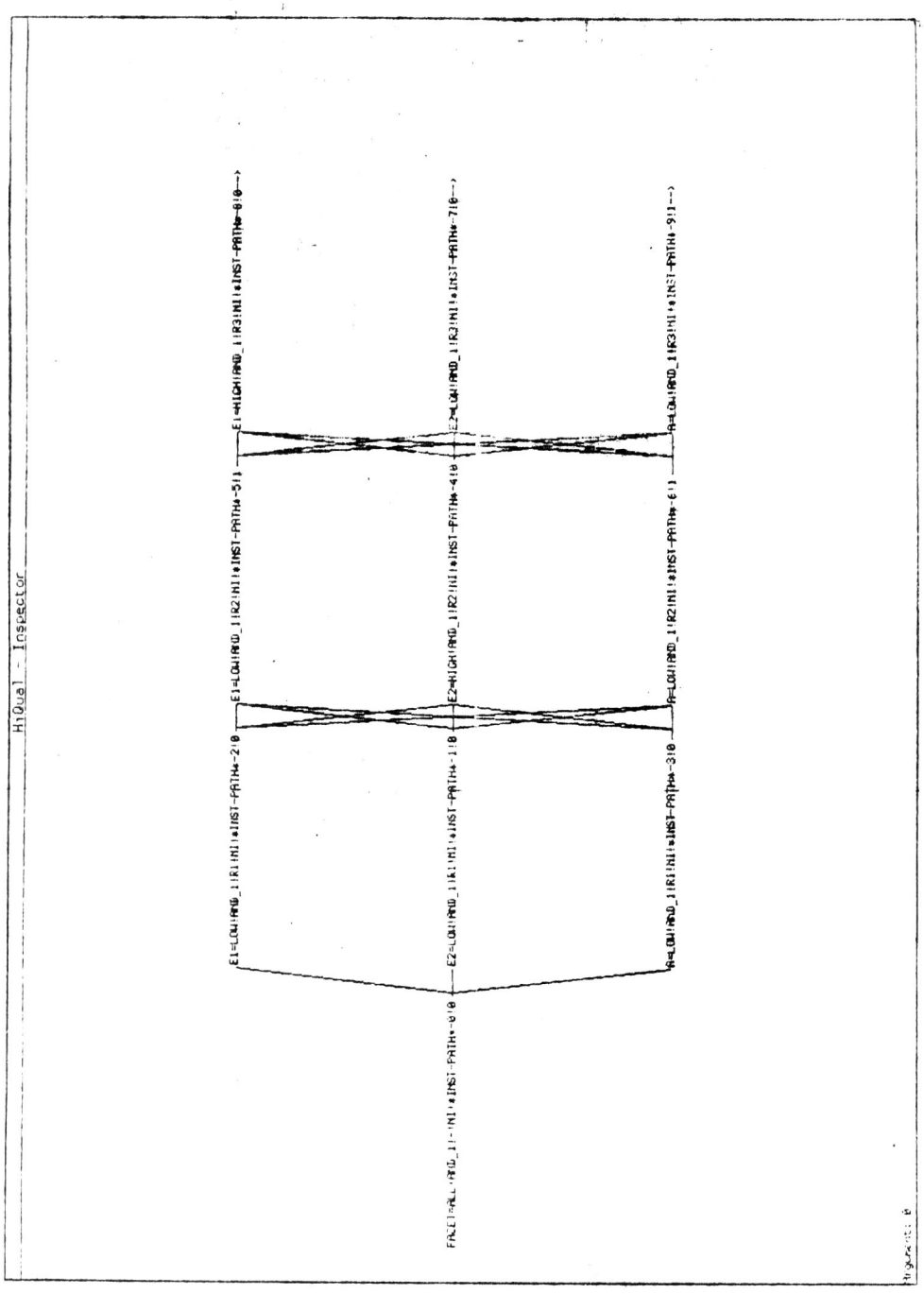
Instanziierung des Modelltyps Spannungsverteiler und Analyse des erzeugten Modells SV1.



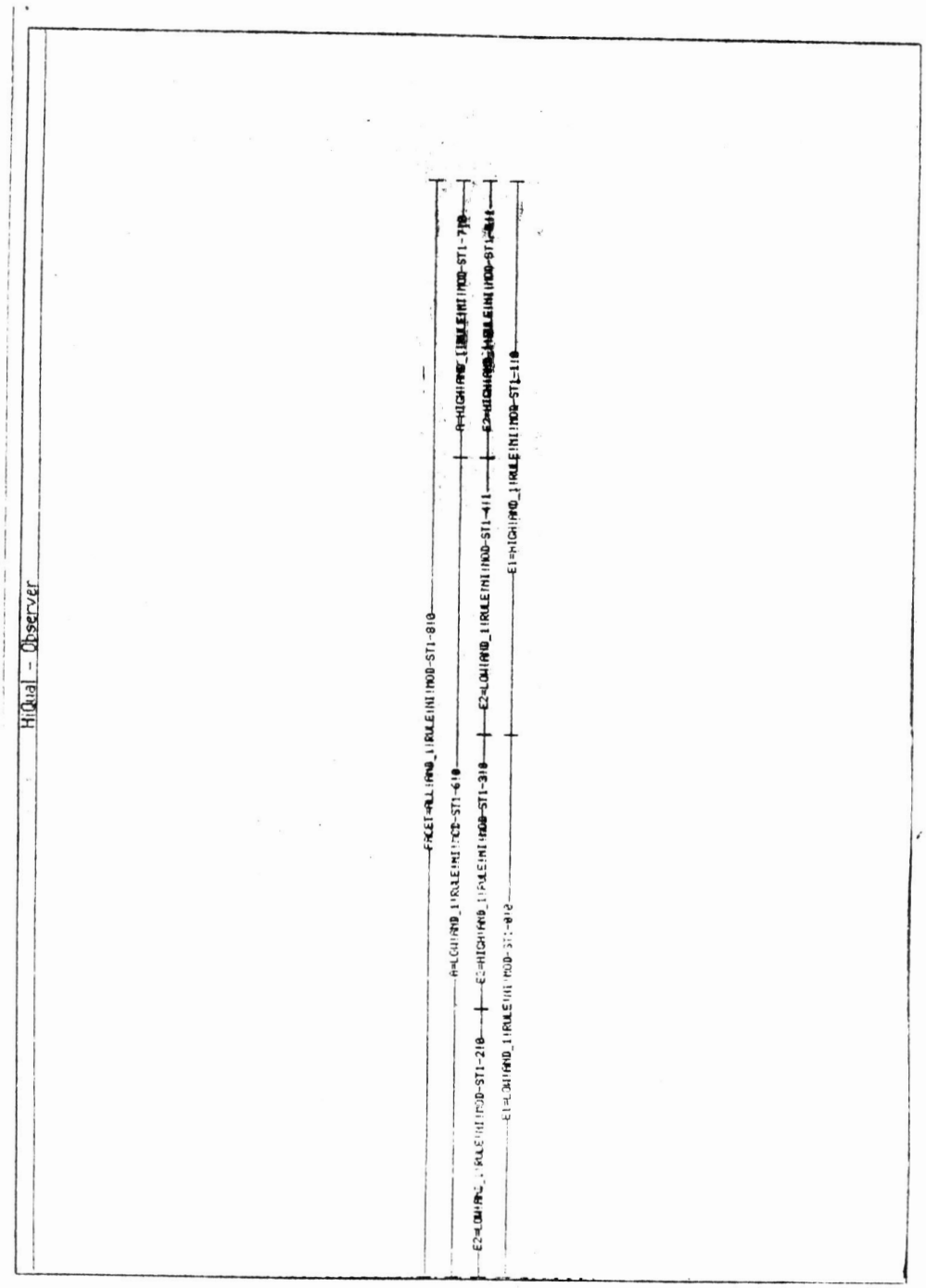
Zustandsanzeige des Systems mit der Option DEFINIERTE ELEMENTE.



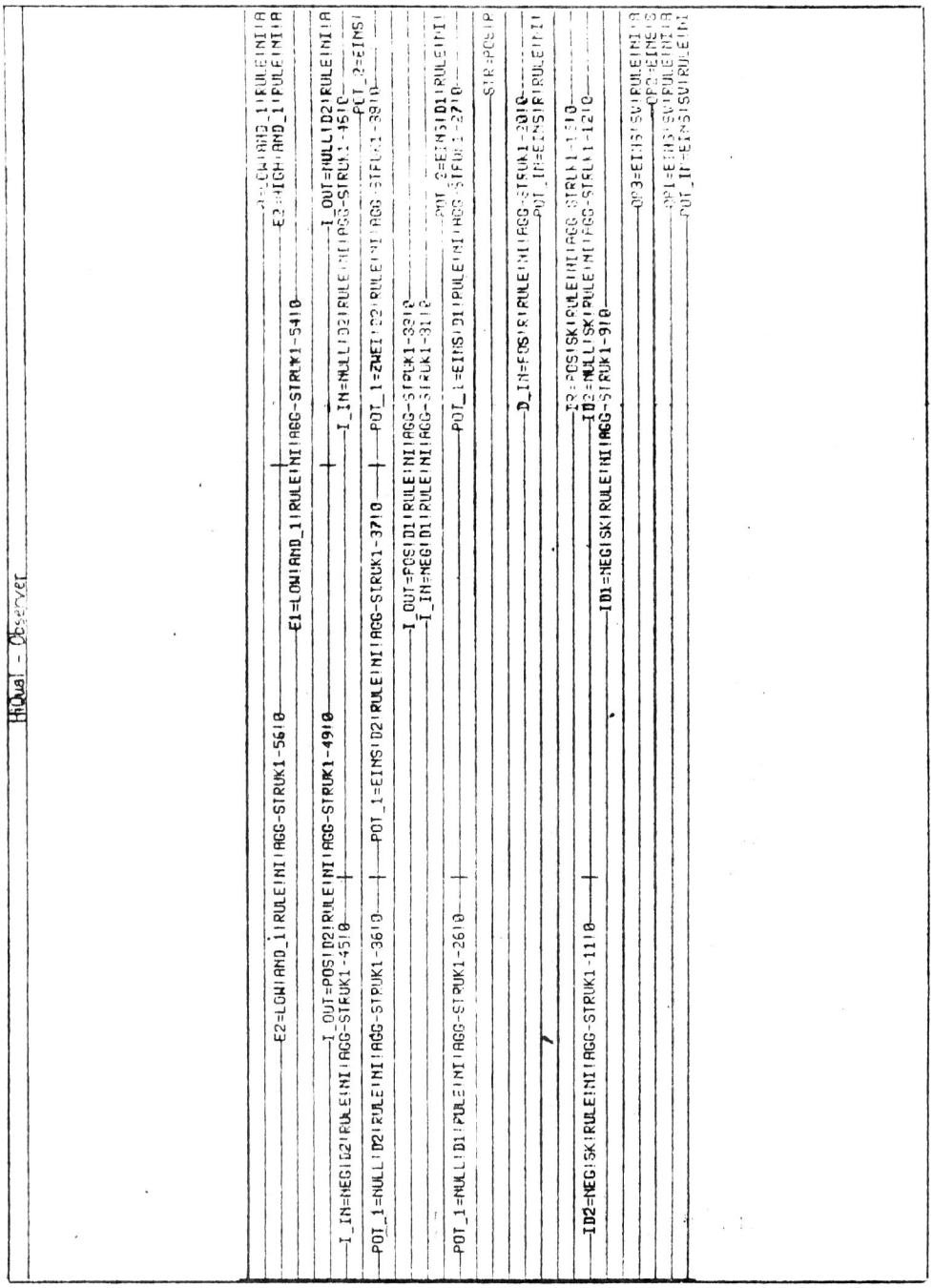
Ausschnitt aus dem Modellgraphen des Modells AND im Inspector.



Ausschnitt eines Modellpfades für das Modell AND.



Darstellung des selben Pfades im Observer nach der Eventanalyse.



Darstellung der Eventstrukturen für das Aggregat DDL_AND_AGG_1 mit dem zuvor gezeigten Wurzelpfad.