Spezifikation, Semantik und Überwachung von Objektlebensläufen in Datenbanken

Gunter Saake

Informatik, Abt. Datenbanken
Technische Universität Braunschweig
Postfach 3329
D–3300 Braunschweig

Vorversion Januar 1988

Abstract

Hier steht der Text des Abstracts und der Zusammenfassung.

Inhaltsverzeichnis

1 Einführung

Der Entwurf und die Implementierung von großen Softwaresystemen ist eine kosten- und arbeitsintensive Aufgabe, die sich in der Regel über einen langen Zeitraum erstreckt und von mehreren Personen durchgeführt werden muß. Ausgehend von einer Anforderungsanalyse wird das Gesamtsystem in mehreren aufeinander folgenden Phasen entwickelt, wobei jede dieser Phasen durch eine Beschreibung des Systemverhaltens, der Systemarchitektur und weiterer Eigenschaften dokumentiert wird.

Fehler in den frühen Entwurfsphasen, die erst in einer späteren Entwurfsphase oder gar erst nach Implementierung des Systems entdeckt werden, erfordern einen aufwendigen Neuentwurf des Gesamtsystems oder von einzelnen Systemkomponenten. Nur eine formale Beschreibung des Softwaresystems mit fester Semantik ermöglicht das Erkennen von derartigen Inkonsistenzen und unvollständigen Beschreibungen in frühen Entwurfsphasen.

Wünschenswert ist eine formale Systembeschreibung bereits in einer möglichst frühen Entwurfsphase. Jede Entwurfsphase wird als Referenz für die in den nächsten Phasen erfolgenden Verfeinerungen verwendet. Durch eine formale Beschreibung ist eine eindeutige Referenzschnittstelle zwischen den Entwurfsphasen gegeben. Ein Überblick über verschiedene formale Beschreibungsmethoden für komplexe Softwaresysteme kann in [CHJ86] gefunden werden.

Dem *Datenbankentwurf* kommt im Rahmen des Entwurfs großer Softwaresysteme eine besondere Bedeutung zu, da alle Systemkomponenten auf die in der Datenbank zentral gehaltenen Daten zugreifen müssen. Auf die Festlegung der Struktur dieser zentral gespeicherten Informationen, ihrer möglichen Entwicklungen während der Laufzeit des Gesamtsystems und auf die Beschreibung der Datenbankschnittstelle sollte also besondere Sorgfalt gelegt werden.

Entgegen der Relevanz des Datenbankentwurfs für den Gesamtentwurf sind die theoretischen Grundlagen, Methoden und Werkzeuge für dieses Gebiet im Vergleich zum klassischen Softwareentwurf nicht sehr weit entwickelt. Dieses Defizit reicht vom weitgehenden Fehlen geeigneter formaler semantischer Grundlagen für Datenbanksprachen bis hin zum Mangel an Techniken und Systemen zur methodischen Unterstützung des Datenbankentwurfs. In dieser Arbeit werden für ein Teilgebiet des Datenbankentwurfs, der *Spezifikation der erlaubten Änderungen des Datenbankinhalts*, die semantischen Grundlagen vorgestellt und Techniken für die Entwurfsunterstützung diskutiert.

Der Datenbankentwurf läßt sich in folgende Phasen aufteilen, die dem bekannten 'Lifecycle' von Softwaresystemen entsprechen (angelehnt an [MDL87]):

- Die *Anforderungsanalyse* beinhaltet das Zusammenstellen der informalen Anforderungen an die Funktionalität der Datenbank. Diese Anforderungen bilden die Grundlage für

- den *konzeptionellen Entwurf*, der die Struktur der Datenbank und ihre Schnittstelle formal beschreibt. Das Ergebnis des konzeptionellen Entwurfs ist das sogenannte *konzeptionelle Schema*, das wiederum die Grundlage für die folgenden Entwurfsphasen bildet.

- Der *logische Entwurf* und die anschließende *Implementierung* übertragen das konzeptionelle Schema in ein Schema eines Datenmodells (z.B. des relationalen datenmodells) und realisieren dieses logische Schema ergänzt um die in der Schnittstellenspezifikation beschriebenen Operationen.

In den Entwurfsphasen werden unterschiedliche Entwurfssprachen und formale Modellbildungen eingesetzt. Zur Unterstützung des Datenbankentwurfs sind Entwurfsmethoden und -umgebungen nötig, um einen effizienten Entwurfsprozeß zu ermöglichen (vergl. die Vorschläge in [Ce83, BDRZ84, ADD85] und in [SNHE87, EHNSE88]). Unterstützt werden muß neben dem Entwurf einer Phase, z.B. durch Konsistenzprüfungen, auch der Übergang zwischen den einzelnen Phasen, z.B. in Form von Korrektheitskriterien für Verfeinerungen oder Transformationsmethoden.

Der konzeptionelle Entwurf dient als Schnittstelle zwischen den Anwendern des Systems, die ihre Wünsche in der Anforderungsanalyse artikuliert haben, und den Erstellern des Softwaresystems. Demzufolge muß der Entwurf einerseits für die Anwender verständlich sein, die oft nicht mit formalen Spezifikationsmethoden vertraut sind, andererseits dient er als formale Anforderungsbeschreibung für die späteren Entwurfsphasen. Die Semantik eines konzeptionellen Schemas muß somit formal definiert sein, um die Verifikation der Implementierung zu ermöglichen. Gleichzeitig sollten aber die verwendeten Konzepte so "natürlich" sein, daß sie auch für Nichtspezialisten intuitiv verständlich sind.

Der Entwurf einer Datenbank beinhaltet als zentrale Aufgabe die Modellierung der Informationsstruktur für einen Ausschnitt der realen Welt. Die Struktur der Daten, die für die Anwendung relevanten Objekte und statisches Weltwissen über die erlaubten Datenbankinhalte werden festgelegt und formal in einem konzeptionellen oder logischen Datenmodell beschrieben. Für die Modellierung der Informationsstruktur gibt es bereits viele Entwurfssprachen und Methoden (ein Überblick kann in [MDL87] gefunden werden). Für die konzeptionelle Entwurfsphase werden z.B. erweiterte Entity-Relationship-Modelle [HNSE87] oder funktionale Datenmodelle vorgeschlagen.

Der Entwurf des dynamischen Verhaltens einer Datenbank, d.h. die Beschreibung der erlaubten Änderungen der in ihr gespeicherten Informationen und der vom System zur Verfügung gestellten Änderungstransaktionen, ist beim Datenbankentwurf bisher wenig berücksichtigt worden. Änderungstransaktionen werden bisher — unabhängig von den besonderen Anforderungen an Datenbanken — mit Methoden des allgemeinen Softwareentwurfs entwickelt. Die aus der Weltmodellierung resultierenden Einschränkungen an die Transaktionsauswirkungen werden dabei nicht explizit berücksichtigt.

In [Se80] wurde eine Spezifikationslogik für den konzeptionellen Entwurf von Informationssystemen vorgestellt, in der sowohl deskriptive Anforderungen an das zeitliche Verhalten eines Systems als auch die Wirkungen von Änderungsoperationen spezifiziert werden können. Dieser Logikkalkül wurde in [FS86] weiter verfeinert und um formale Beweistechniken erweitert. Verwandte Ansätze für den konzeptionellen Datenbankentwurf wurden daraufhin u.a. in [CCF82, GMS83, ELG84, SFNC84, CS87] vorgeschlagen. Gemeinsam ist diesen Ansätzen, daß eine temporale Prädikatenlogik für die Beschreibung des dynamischen Datenbankverhaltens eingesetzt wird.

Die abstrakte Beschreibung der erlaubten zeitlichen Entwicklungen der gespeicherten Informationen in temporaler Logik und darauf basierender Entwurfsmethoden ist Inhalt dieser Arbeit. Die zeitliche Entwicklung wird durch die aus Änderungen resultierende Folge von Datenbankzuständen repräsentiert, die jeweils eine konsistente Ausprägungen der spezifizierten Informationsstrukturen sind. Die temporale Logik ermöglicht Aussagen der Art 'das Gehalt einer Person an einer Firma darf nicht sinken', die von den Änderungstransaktionen eingehalten werden müssen.

Ein Spezifikationskalkül für zeitliche Entwicklungen muß mächtige semantische Konzepte zur Beschreibung temporaler Abhängigkeiten zur Verfügung stellen. Eine weitere Anforderung ist die Existenz formaler Beweistechniken zur Untersuchung einer Spezifikation auf Konsistenz. Für die Entwurfsunterstützung sind zusätzlich Verfahren wichtig, die einen methodischen Entwurf unterstützen. Hierzu gehören graphische Darstellungen der spezifizierten zeitlichen Abläufe, konsistenzprüfende Werkzeuge und die Unterstützung von Änderungen und Verfeinerungen des Entwurfs.

Im konzeptionellen Datenbankentwurf werden einzelne Datenbankzustände als Modelle einer Prädikatenlogik aufgefaßt, die die statischen Integritätsbedingungen erfüllen. Die *temporale Prädikatenlogik* ist als Spezifikationskalkül für die zeitlichen Änderungen einer Datenbank besonders geeignet, da sie die mehrsortige Prädikatenlogik erster Stufe, die als Spezifikationskalkül für den Strukturanteil einer Datenbank verbreitet ist, als Teilkalkül enthält.

Temporale Logiken sind Erweiterungen der klassischen Logik um zeitbezogene Aussagen wie 'ab jetzt gilt immer ...'. Temporale Logiken werden bisher hauptsächlich als Spezifikations- und Verifikationskalküle im Bereich der Programmentwicklung und Programmsemantik [MP79, MP81, Mo86, Krö87], der Prozeßkommunikation und der verteilten Systeme [CE81, MW84, CES86] eingesetzt. In diesen Anwendungen ist eine temporale Aussagenlogik ausreichend.

Für den Datenbankentwurf hingegen ist eine Erweiterung auf eine temporale Prädikatenlogik notwendig. Eine zusätzliche Anforderung ist die formale Behandlung des Einfügens und Löschens von Daten in die Datenbank, die in dieser Arbeit in der vorgestellten *finit-linearen temporalen Logik* integriert ist.

Inspiriert von den Vorschlägen in [SFNC84, CS87, SSE87] wird eine *mehrschichtige objektorientierte Entwurfsmethodik* für den konzeptionellen Datenbankentwurf vorgeschlagen. In dieser Entwurfsmethodik wird ein konzeptionelles Schema in aufeinander aufbauenden Schichten spezifiziert, die von der Festlegung der elementaren Datentypen bis hin zu einer Spezifikation von Anwendungstransaktionen reichen. Für jede Schicht wird eine den speziellen Anforderungen angemessene Spezifikationslogik benutzt, die sowohl semantisch als auch syntaktisch auf den unterliegenden Schichten aufbaut.

Neben der 'horizontalen' Entwurfsstrukturierung durch die mehrschichtige Spezifikation wird der Gesamtentwurf 'vertikal' dadurch gegliedert, daß die statischen und dynamischen Eigenschaften einzelner informationstragender Einheiten, der sogenannten *Datenbankobjekte*, in Spezifikationsmodule zusammengefaßt werden. Dieser *objektorientierte* Entwurf ermöglicht neben dem strukturierten Entwurf mit sauberen Schnittstellen auch lokale Konsistenzprüfungen und die Entwicklung wiederverwendbarer Teilentwürfe.

Die finit-lineare temporale Logik wird in dieser Entwurfsmethodik für die deskriptive Beschreibung der zeitlichen Änderungen des Datenbankinhalts eingesetzt. Sie umfaßt die Spezifikationskalküle für die statischen Eigenschaften von Datenbankinhalten als Teilkalküle und ihre semantische Modellbildung durch Folgen von Datenbankzuständen, und ist gleichzeitig die Interpretationsstruktur für die Spezifikation von Anwendungstransaktionen.

Nach der Wahl einer Spezifikationssprache muß ihre Eignung für die Lösung der auftretenden Entwurfsprobleme untersucht werden. Neben der erwähnten formalen Semantik sind Methoden zur benutzergerechten Aufarbeitung der Spezifikation, Grundlagen der Strukturierung der Spezifikation und der Konsistenzprüfung notwendig. Für den Übergang zu verfeinerten Entwurfsphasen bis hin zur Implementierung und Betriebsphase sind Transformationsmethoden und Korrektheitskriterien zu entwickeln.

Ziel dieser Arbeit ist es, den ganzen Bereich der Spezifikation des dynamischen Verhaltens von Datenbanken in temporaler Logik abzudecken. Der Schwerpunkt ist die formale Semantik der Spezifikation und die Einbindung in eine strukturierte Spezifikationsmethodik für Datenbank-Anwendungen. Für andere Teilbereiche, z.B. für eine methodische Implementierung der Spezifikation, werden nur die grundlegenden Verfahren und Anforderungen vorgestellt.

Diese Arbeit ist wie folgt gegliedert : In Kapitel 2 werden die formalen Grundlagen und Notationen der Prädikatenlogik rekapituliert. Aufbauend auf diesen Festlegungen wird eine *modale Prädikatenlogik* definiert, die die Formulierung von logischen Aussagen über mehrere prädikatenlogische Modelle, die einzelnen Datenbankzuständen entsprechen, ermöglicht. In Kapitel 3 werden, basierend auf modallogischen Konzepten, temporale Logiken eingeführt. Es wird die finit-lineare temporale Logik festgelegt, die im folgenden als Spezifikationskalkül für das dynamische Datenbankverhalten verwendet wird.

Im Kapitel 4 werden aus temporalen Formeln konstruierte Transitionsgraphen eingeführt, die eine operationale Beschreibung der spezifizierten Abläufe erlauben. Ihre formale Semantik wird untersucht und die Äquivalenz zur deskriptiven Semantikdefinition der temporalen Logik gezeigt. Es werden Verfahren zur Manipulation von Transitionsgraphen und auf diese aufbauende Verfahren zur Konsistenzüberprüfung temporaler Formeln vorgestellt.

Die Einbindung des temporalen Spezifikationskalküls in eine Entwurfsmethodik für konzeptionelle Datenbankschemata wird in Kapitel 5 untersucht. Es zeigt sich, daß die aus temporalen Formeln konstruierten Transitionsgraphen sowohl als graphische Darstellung der spezifizierten Änderungen als auch als Grundlage für konsistenzprüfende Werkzeuge eingesetzt werden können.

Der Inhalt des abschließenden Kapitels 6 befaßt sich mit dem Aspekt der Implementierung der spezifizierten Datenbankdynamik. Schwerpunkte hierbei sind die algorithmischen Grundlagen eines *Integritätsmonitors*, die die Integration der Überwachung temporaler Formeln in ein Datenbankmanagementsystem ermöglichen.

2 Theoretische Grundlagen

Das Kapitel über die theoretische Grundlagen setzt Grundkenntnisse der mathematischen Logik und anderer üblicher mathematische Notationen wie den Begriff der Algebra als bekannt voraus. Insbesondere werden in den Definitionen Begriffe und Notationen der elementaren Mengenlehre ohne explizite Einführung des Formalismus benutzt. Die wichtigsten weitergehenden Begriffe, die in dieser Arbeit benötigt werden, werden definiert und kurz erläutert, um eine einheitliche Notation einzuführen.

Für die Spezifikation von Datenbankobjekten und speziell ihrem dynamischen Verhalten werden Erweiterungen der klassischen Prädikatenlogik erster Stufe benutzt. Die der Prädikatenlogik zugrundeliegenden Strukturen und Begriffsbildungen werden in einem ersten Unterabschnitt eingeführt. Es folgt ein Abschnitt über allgemeine Modallogiken und ihnen zugrundeliegende Modelle. In einem dritten Abschnitt werden Belegungen von Variablen in Modalstrukturen und durch sie definierte Quantifizierungen eingeführt und diskutiert.

Für eine ausführlichere Einführung in die Thematik der einzelnen Abschnitte sind Lehrbücher zur mathematischen Logik [BN77, GOT83] und zur Theorie der Modallogiken [HC68, Ch80] geeignet. Weiterführende Literatur wird an den entsprechenden Stellen angegeben.

2.1 Prädikatenlogik

Die mathematische Logik wurde als Grundlage für das exakte mathematische Arbeiten mit *Aussagen* entwickelt. Aussagen sind Sätze einer Sprache, denen ein Wahrheitsgehalt zugeordnet werden kann. Ist diese Sprache formal definiert, wird von einem *Logikkalkül* gesprochen. Der mathematischen Logik liegt eine einfache formale Sprache zur Formulierung von Aussagen zugrunde. Wesentliche Sprachelemente sind elementare Aussagen über die Objekte von Interesse und die logische Verknüpfung von Aussagen zu einer neuen Aussage.

Jedem Satz dieser formalen Sprache wird einer der zwei Wahrheitswerte *wahr* oder *falsch* zugeordnet. Der Wahrheitswert der elementaren Aussagen ist dabei abhängig von den in der speziellen Anwendung betrachteten Objekten, während die Bedeutung der logischen Verknüpfungen festen Gesetzen folgt.

Im Gegensatz zu vielen klassischen Anwendungen der mathematischen Logik zeichnen sich die betrachteten Objekte von Interesse und somit die elementaren Aussagen über diese Objekte im Datenbankentwurf durch eine hohe Komplexität aus. Aus diesem Grund wird ein *mehrsortiger Prädikatenkalkül mit Gleichheit* eingeführt.

In einem solchen Prädikatenkalkül sollen Aussagen über einen bestimmten Weltausschnitt formuliert werden. Die Struktur der betrachteten Objekte von Interesse und damit die Syntax der elementaren Aussagen wird mit einer *Signatur* festgelegt.

Definition 2.1.1 Eine *Signatur* $\Sigma = (S, \Omega, \Pi)$ besteht aus

1. einer Menge $S = \{s, r, \ldots\}$ von Sorten,
2. einer Menge $\Omega = \{f\colon s_1 \times \ldots \times s_n \to s_0,\ g\colon r_1 \times \ldots \times r_m \to r_0, \ldots\}$ von Funktionssymbolen mit ≥ 0 Argumentsorten und einer Wertsorte,
3. einer Menge $\Pi = \{p\colon s_1 \times \ldots \times s_n,\ q\colon r_1 \times \ldots \times r_m, \ldots\}$ von Prädikatensymbolen mit ≥ 0 Argumentsorten. □

Die Teilsignatur $(S, \Omega, \emptyset)$ beschreibt die Struktur einer mehrsortigen Algebra mit den Sorten S und den in Ω festgelegten Operationen auf diesen Sorten. Die Angabe der Argumentsorten und der Wertsorten in Ω legt die Syntax der erlaubten Terme fest.

Sei zusätzlich zu der Signatur $\Sigma = (S, \Omega, \Pi)$ eine Menge von sortengebundenen Variablen $X = \{x\colon s_1,\ y\colon s_2, \ldots\}$ mit $s_i \in S$ gegeben. Die Menge aller Terme zu einer Signatur Σ und einer Variablenmenge X wird als $T_\Sigma(X)$ bezeichnet.

Definition 2.1.2 Die *Terme* einer Sorte s in $T_\Sigma(X)$ sind definiert durch :

1. Jede Variable $x\colon s$ ist ein Term der Sorte s.
2. Sind $t_1\colon s_1, \ldots, t_n\colon s_n$ Terme der Sorten $s_1, \ldots, s_n$ und ist $f\colon s_1 \times \ldots \times s_n \to s$ ein Funktionssymbol, so ist $f(t_1\colon s_1, \ldots, t_n\colon s_n)\colon s$ ein Term der Sorte s.
3. Keine anderen Zeichenketten sind Terme in $T_\Sigma(X)$. □

Nullstellige Funktionssymbole werden auch als Konstanten einer Sorte bezeichnet. Sie erlauben die Konstruktion von Termen ohne Variablen, die als konstante Terme bezeichnet werden.

Die Menge der Prädikatensymbole Π legt die elementaren Aussagen des Prädikatenkalküls fest. Diese elementaren Aussagen werden als *atomare Formeln* bezeichnet. Für alle Sorten $s \in S$ ist ein Prädikatensymbol $= \colon s \times s$ mit fester Interpretation vorgegeben und in der Signatur als zweistelliges Prädikatensymbol enthalten.

Die *Gleichheitsprädikate* werden in der Signatur bei den Prädikatensymbolen Π nicht explizit aufgeführt, da sie in der Definition der Semantik von Formeln anders behandelt werden müssen als die in Π explizit aufgeführten Prädikatensymbole. Die Menge der atomaren Formeln zu einer Signatur Σ und einer Variablenmenge X wird mit $AF_\Sigma(X)$ bezeichnet.

Definition 2.1.3 Die *atomaren Formeln* in $AF_\Sigma(X)$ sind definiert durch :

1. Sind $t_1\colon s$ und $t_2\colon s$ Terme der gleichen Sorte s aus $T_\Sigma(X)$, so ist $t_1\colon s = t_2\colon s$ eine atomare Formel.

2. Sind $t_i\colon s_i$ Terme der Sorten s_i aus $T_\Sigma(X)$ und ist $p\colon s_1 \times \ldots \times s_n$ ein Prädikatensymbol, so ist $p(t_1\colon s_1, \ldots, t_n\colon s_n)$ eine atomare Formel.

3. Keine anderen Zeichenreihen sind atomare Formeln in $AF_\Sigma(X)$. □

Nullstellige Prädikate werden oft auch als *Aussagenvariablen* bezeichnet. Aus den atomaren Formeln werden durch logische Verknüpfungen und Quantifizierung über Variablen die Formeln des Prädikatenkalküls gebildet. Die Menge aller prädikatenlogischen Formeln für eine gegebene Signatur $\Sigma = (S, \Omega, \Pi)$ und einer Variablenmenge X wird mit $P_\Sigma(X)$ bezeichnet.

Definition 2.1.4 Die *Formeln* in $P_\Sigma(X)$ sind definiert durch :

1. Die Symbole **true** und **false** sind Formeln.

2. Jede atomare Formel in $AF_\Sigma(X)$ ist eine Formel in $P_\Sigma(X)$.

3. Sind φ, ψ Formeln in $P_\Sigma(X)$ und x eine Variable der Sorte $s \in S$, so sind auch $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \Rightarrow \psi)$, $(\varphi \iff \psi)$, $\neg\varphi$, $(\exists x\colon s)\varphi$ und $(\forall x\colon s)\varphi$ Formeln in $P_\Sigma(X)$.

4. Keine anderen Zeichenreihen sind Formeln in $P_\Sigma(X)$. □

Um die Lesbarkeit von Formeln zu erhöhen, werden alle üblichen logischen Symbole eingeführt, obwohl sie zum Teil auch als Abkürzungen für andere Formeln definiert werden können, so z.B. $(\varphi \vee \psi)$ als $\neg(\neg\varphi \wedge \neg\psi)$ oder **false** als $\neg$**true**. Eine für den Prädikatenkalkül ausreichende Menge von logischen Symbolen wäre z.B. $\neg$, $\wedge$ und $(\forall x\colon s)$.

Ebenfalls zur Erhöhung der Lesbarkeit von Signaturen und Formeln werden weitere Abkürzungskonventionen eingeführt. Bei Konstanten $k\colon \to s$ in Ω wird das "$\to$"–Symbol in der Signatur weggelassen. In Termen werden die Sortenangaben weggelassen, falls keine Mißverständnisse zu erwarten sind. Klammern werden zur Strukturierung gesetzt und zur Verbesserung der Lesbarkeit weggelassen. Hierbei gelten die üblichen Vorrangregeln für die logischen Konnektive (die Operatoren $\neg$, $(\exists x\colon s)$ und $(\forall x\colon s)$ binden stärker als alle binären Operatoren, $\wedge$ bindet stärker als $\vee$, etc.).

Die klassische Prädikatenlogik mit nur einer Objektsorte ist in den Definitionen mit einer einelementigen Sortenmenge $S = \{s\}$ enthalten. Die Aussagenlogik kann durch eine leere Sortenmenge $S = \emptyset$ erhalten werden — da keine Sorten vorhanden sind, gilt ebenfalls $\Omega = \emptyset$ und Π enthält nurmehr nullstellige Prädikate, die die Aussagenvariablen der Aussagenlogik bilden.

Eine allgemeinere Formulierung einer Aussagenlogik erhält man durch eine leere Variablenmenge $X = \emptyset$, d.h. nur konstante Terme und aus ihnen gebildete atomare Formeln sind möglich. Dieser Kalkül ist zu einer gegebenen Signatur durch $P_\Sigma(\emptyset)$ charakterisiert. In beiden Fällen können keine quantifizierten Formeln gebildet werden. Die erste Auffassung der Aussagenlogik ist ein Spezialfall der zweiten, so daß im folgenden mit Aussagenlogik zu einer Signatur Σ der Kalkül $P_\Sigma(\emptyset)$ gemeint ist.

Die bisherigen Definitionen legen die Syntax des Prädikatenkalküls fest. Die Zuordnung der Wahrheitswerte zu den Formeln wird als *Semantik der Formeln* bezeichnet. Der Wahrheitswert einer Formel hängt von der Interpretation der Signatur Σ durch eine Σ–Struktur und der Belegung der Variablen aus X ab.

Definition 2.1.5 Eine *Interpretation* A einer Signatur $\Sigma = (S, \Omega, \Pi)$ ist eine Σ–*Struktur* $A(\Sigma) = (A(S), A(\Omega), A(\Pi))$, die definiert ist durch

$A(S)$: eine nichtleere Trägermenge $A(s)$ für jede Sorte $s \in S$,

$A(\Omega)$: eine Funktion $A(f)\colon A(s_1) \times \ldots \times A(s_n) \to A(s_0)$ für jedes Funktionssymbol $f\colon s_1 \times \ldots \times s_n \to s_0 \in \Omega$,

$A(\Pi)$: eine Relation $A(p) \subseteq A(s_1) \times \ldots \times A(s_n)$ für jedes Prädikatensymbol $p\colon s_1 \times \ldots \times s_n \in \Pi$.

Die Menge aller Σ–Strukturen für eine Signatur Σ wird als Menge der *möglichen Interpretationen* $\Im_\Sigma$ bezeichnet. □

Für eine gegebene Σ–Struktur $A(\Sigma) = (A(S), A(\Omega), A(\Pi))$ wird eine Belegung der Variablen aus X durch eine Belegungsfunktion β festgelegt.

Definition 2.1.6 Eine *Belegung* β der Variablen in X ist eine Funktion

$$\beta\colon X \to \bigcup_{s \in S} A(s) \ ,$$

die jeder Variablen $x\colon s \in X$ einen Wert $\beta(x\colon s) \in A(s)$ aus der Trägermenge der passenden Sorte zuordnet. Die *Menge aller Belegungen* einer Variablenmenge X für eine gegebene Σ–Struktur $A(\Sigma)$ wird mit $\mathit{BEL}(X, A)$ bezeichnet.

Diese Definition wird auf Mengen von Σ–Strukturen durch Vereinigung der einzelnen Mengen von Belegungen erweitert. So bezeichnet $BEL(X, \Im_\Sigma)$ die Menge aller für eine gegebene Signatur möglichen Belegungen. $\Box$

Ist die Interpretation der Signatur und die Belegung der Variablen bestimmt, kann die Auswertung von Termen rekursiv über den syntaktischen Aufbau definiert werden. Dies entspricht der intuitiven Termauswertung, bei der der Gesamtwert eines Ausdrucks ausgehend von den kleinsten Bestandteilen, deren Wert direkt bestimmt ist, errechnet wird. In den folgenden Definitionen wird der Übersicht halber auf die explizite Sortenangabe bei Variablen und Termen verzichtet.

Definition 2.1.7 Für eine gegebene Σ–Struktur A und eine Belegung der Variablen β ist die *Termauswertung* definiert durch

$$\begin{aligned}(A,\beta)[\![x]\!] &= \beta(x) \text{ für } x \in X\\ (A,\beta)[\![f(t_1,\ldots,t_n)]\!] &= A(f)((A,\beta)[\![t_1]\!],\ldots,(A,\beta)[\![t_n]\!])\end{aligned}$$

Der Ausdruck $(A,\beta)[\![t]\!]$ bezeichnet hierbei den Wert des Termes t bei gegebener Σ–Struktur A und Belegung β. Die Auswertung von Konstanten ist wiederum als Spezialfall enthalten. $\Box$

Aufbauend auf der Termauswertung kann nun der Wahrheitswert einer Formel definiert werden. Eine Formel heißt *gültig* für eine Σ–Struktur A und eine Belegung β, wenn ihr der Wahrheitswert *wahr* zugeordnet ist. Die Gültigkeit einer Formel φ bezüglich gegebenen A und β wird durch $(A,\beta) \models \varphi$ abgekürzt, $\not\models$ ist entsprechend das Symbol für "nicht gültig".

Definition 2.1.8 Für eine gegebene Σ–Struktur A und eine Belegung der Variablen β ist die *Gültigkeit einer Formel* $\varphi \in P_\Sigma(X)$, kurz $(A,\beta) \models \varphi$, definiert durch

$$
\begin{array}{lll}
(A,\beta) \models \mathbf{true} & & \\
(A,\beta) \not\models \mathbf{false} & & \\
(A,\beta) \models t_1 = t_2 & \text{gdw.} & (A,\beta)[\![t_1]\!] = (A,\beta)[\![t_2]\!]. \\
(A,\beta) \models p(t_1,\ldots,t_n) & \text{gdw.} & ((A,\beta)[\![t_1]\!],\ldots,(A,\beta)[\![t_n]\!]) \in A(p). \\
(A,\beta) \models \neg\varphi & \text{gdw.} & \text{nicht } (A,\beta) \models \varphi. \\
(A,\beta) \models \varphi \wedge \psi & \text{gdw.} & (A,\beta) \models \varphi \text{ und } (A,\beta) \models \psi. \\
(A,\beta) \models \varphi \vee \psi & \text{gdw.} & (A,\beta) \models \neg(\neg\varphi \wedge \neg\psi). \\
(A,\beta) \models \varphi \Rightarrow \psi & \text{gdw.} & (A,\beta) \models \neg\varphi \vee \psi. \\
(A,\beta) \models \varphi \iff \psi & \text{gdw.} & (A,\beta) \models (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi). \\
(A,\beta) \models (\forall x)\varphi & \text{gdw.} & \text{für alle Belegungen } \beta' \in \mathit{BEL}(X,A), \text{ die sich von } \beta \\
& & \text{nur im Wert für } x \text{ unterscheiden, gilt :} \\
& & (A,\beta') \models \varphi. \\
(A,\beta) \models (\exists x)\varphi & \text{gdw.} & (A,\beta) \models \neg((\forall x)\neg\varphi).
\end{array}
$$

$\square$

Durch diese Definition ist die Gültigkeit einer Formel für den allgemeinen Fall definiert. Es folgen einige Notationen für wichtige Spezialfälle.

Definition 2.1.9 Die Behandlung einer Variablen x in einer Formel hängt davon ab, ob ihre Vorkommen in der Formel durch einen Quantoren, also einem Existenz– ($\exists$) oder Allquantor ($\forall$), gebunden sind. Für diese Vorkommen ist die Belegung der Variable x durch eine Belegung β unerheblich.

- Eine Auftreten einer Variable x heißt *gebunden* in einer Formel φ, wenn dieses Auftreten Teil einer Unterformel $(\forall x)\varphi'$ oder$(\exists x)\varphi'$ von φ ist.
- Eine Variable x heißt *frei* in einer Formel φ, wenn sie ein nicht gebundenes Auftreten in φ hat.
- Eine Formel φ heißt *geschlossen*, wenn sie keine freien Variablen enthält.

Die Gültigkeit von geschlossenen Formeln ist unabhängig von einer Belegung β. Allgemein werden folgende Kurznotationen benutzt :

$$
\begin{array}{rll}
A \models \psi & \text{gdw.} & (A,\beta) \models \psi \text{ für alle Belegungen } \beta. \\
\models \psi & \text{gdw.} & A \models \psi \text{ für alle } \Sigma\text{–Strukturen } A.
\end{array}
$$

- Gilt $\models \psi$, so heißt ψ *allgemeingültig* oder *Tautologie*.

- Gilt $\models \neg\psi$, so heißt ψ *unerfüllbar* oder *Widerspruch*. $\square$

Diese und die nun folgenden Definitionen und Notationen sind prinzipiell unabhängig vom verwendeten Kalkül. Sie werden in den weiteren Kapiteln für Erweiterungen der Prädikatenlogik übernommen, ohne daß sie explizit mit den nötigen Modifikationen neu definiert werden.

Definition 2.1.10 Jede Σ–Struktur $A \in \Im_\Sigma$ legt eindeutig die Menge der Formeln aus $P_\Sigma(X)$ fest, die in A für alle Belegungen β gültig sind. Eine Menge von Σ–Strukturen $\mathbf{A} \subseteq \Im_\Sigma$ legt die Menge der Formeln fest, die in allen $A \in \mathbf{A}$ gültig sind. Diese Formelmenge wird als *Theorie* von A bzw. $\mathbf{A}$ bezeichnet, kurz $TH(A)$ bzw. $TH(\mathbf{A})$. $TH(A)$ ist ein Spezialfall der folgenden Definition von $TH(\mathbf{A})$.

$$TH(\mathbf{A}) = \left\{\varphi \;\middle|\; A \models \varphi \text{ für alle } A \in \mathbf{A}\right\}$$

Umgekehrt legt eine Menge von geschlossenen Formeln $\Phi \subset P_\Sigma(X)$ diejenige Menge von Σ–Strukturen fest, in der diese Formeln gültig sind. Diese Menge wird als die Menge der *Modelle* $MOD(\Phi) \subseteq \Im_\Sigma$ von Φ bezeichnet.

$$MOD(\Phi) = \{A \mid A \models \varphi \text{ für alle } \varphi \in \Phi\}$$

Ist eine Formel $\varphi \in P_\Sigma(X)$ in allen Modellen von Φ gültig, wird sie als *logische Konsequenz* von Φ, kurz $\Phi \models \varphi$, bezeichnet. Diese Definition kann auch auf Formeln mit freien Variablen erweitert werden, in dem statt Σ–Strukturen Kombinationen von Σ–Strukturen und Belegungen betrachtet werden.

$$\Phi \models \varphi \quad \text{gdw.} \quad \text{für alle } \Sigma\text{–Strukturen } A \text{ und Belegungen } \beta \text{ gilt :}$$
$$\Big((A,\beta) \models \psi \text{ gilt für alle } \psi \in \Phi\Big) \;\Rightarrow\; \Big((A,\beta) \models \varphi\Big).$$

$\square$

Insbesondere legt eine Menge von Formeln Φ die Menge $TH(MOD(\Phi))$ fest, die von ihr *erzeugte Theorie*. Diese Menge ist eine semantische Festlegung aller logischen Konsequenzen von Φ.

Für praktische Zwecke ist eine syntaktische Festlegung der logischen Konsequenzen durch *Deduktionssysteme* von größerem Interesse, da diese eine algorithmische Methode zur Erzeugung der logischen Konsequenzen bieten. Deduktionssysteme leiten aus einer vorgegebenen Menge von Formeln Φ und aus gegebenen logischen Axiomen durch Anwendung von syntaktischen Ableitungsregeln die logischen Konsequenzen aus Φ ab. Ein Deduktionssystem heißt vollständig, wenn alle logischen Konsequenzen ableitbar sind, und es heißt konsistent, wenn nur diese ableitbar sind.

Für die in diesem und den folgenden Kapiteln vorgestellten Kalküle existieren konsistente und vollständige Deduktionssysteme, auf die an den entsprechenden Stellen mittels Literaturangaben hingewiesen wird. Deduktionssysteme für die vorgestellte Prädikatenlogik, z.B. Resolutionsverfahren, können in Lehrbüchern über mathematische Logik [BN77, Ll84] gefunden werden. Die dort vorgestellten Deduktionssysteme sind oft für die klassische Prädikatenlogik mit einer Sorte eingeführt, können aber direkt auf den mehrsortigen Fall verallgemeinert werden.

2.2 Modallogiken und Kripkestrukturen

Der bisher vorgestellte Prädikatenkalkül ermöglicht die Formulierung von Aussagen über einen festen statischen Weltausschnitt. Formeln werden dementsprechend in einer einzelnen Σ–Struktur ausgewertet. Aussagen über mehrere Welten, z.B. über verschiedene alternative Welten oder über eine Welt zu verschiedenen Zeitpunkten, sind nur durch eine umständliche Erweiterung der Struktursignatur möglich.

Eine übliche Methode der Modellierung von Zeitaspekten ist es z.B., jedes Funktions- und Prädikatensymbol um eine zusätzliche Argumentsorte von Zeitpunkten zu erweitern. Dies beeinträchtigt jedoch die Lesbarkeit von Signaturen und Formeln, erfordert eine zusätzliche explizite Axiomatisierung des intuitiven Zeitbegriffs und erhöht die Fehleranfälligkeit beim Arbeiten mit Formeln.

Schon früh wurden darum *modale Logikkalküle* vorgeschlagen, um eine elegante Formulierung von derartigen Aussagen zu ermöglichen. Den modalen Kalkülen liegen als Interpretationen nicht einzelne Σ–Strukturen zugrunde, sondern Mengen derartiger Σ–Strukturen, die oft als verschiedene *Welten* einer Modallogik bezeichnet werden. Zwischen diesen Welten existiert eine *Relevanzbeziehung* (auch Erreichbarkeits– oder Sichtbarkeitsbeziehung genannt). Für die Beschreibung eines Weltausschnittes zu verschiedenen Zeitpunkten kann diese Relevanzbeziehung zum Beispiel die Bedeutung ‘folgt zeitlich auf’ haben. Des weiteren ist oft eine dieser Welten als die *aktuelle* oder im Moment relevante ausgezeichnet. Diese Interpretationen werden als Modalstrukturen oder auch Kripkestrukturen bezeichnet. In den Arbeiten von Kripke [Kri63a, Kri63b] wurde diese Grundidee erstmals mathematisch exakt formalisiert.

Definition 2.2.1 Eine *Kripkestruktur* K_Σ zu einer Signatur Σ besteht aus

1. einer Menge $\mathcal{W} = \{\alpha, \gamma \ldots\}$ von Welten,

2. einer Relevanzrelation $R \subseteq \mathcal{W} \times \mathcal{W}$,

3. einer Weltinterpretation $I \colon \mathcal{W} \to \Im_\Sigma$, die den einzelnen Welten Σ–Strukturen zuordnet,

4. und einer ausgezeichneten aktuellen Welt $\alpha_0 \in \mathcal{W}$. $\square$

Kripkestrukturen können als gerichtete Graphen dargestellt werden, in denen die Welten als Knoten und die Relevanzrelation als Kanten dargestellt werden. Jedem Knoten ist eine Σ–Struktur als Weltinterpretation zugeordnet.

Der modale Logikkalkül baut auf dem Prädikatenkalkül auf. Die Formeln des Prädikatenkalküls werden in der aktuellen Welt ausgewertet. Zusätzlich können neue modallogische Formeln durch Benutzung der beiden Modaloperatoren $\Box$ und $\Diamond$ gebildet werden. Die Formel $\Box\varphi$ drückt aus, daß φ in allen zur aktuellen Welt relevanten Welten gültig sein muß, während $\Diamond\varphi$ die entsprechende Existenzforderung darstellt.

Die Menge aller modallogischen Formeln zu einer Signatur Σ und einer Variablenmenge X wird mit $M_\Sigma(X)$ bezeichnet.

Definition 2.2.2 Die Syntax der *modallogischen Formeln* in $M_\Sigma(X)$ ist definiert durch

1. Jede prädikatenlogische Formel $\varphi \in P_\Sigma(X)$ ist eine modallogische Formel.

2. Sind φ, ψ modallogische Formeln und x eine Variable der Sorte s, so sind auch $\neg\varphi$, $\varphi \Rightarrow \psi$, $\varphi \iff \psi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $(\hat{\forall} x \colon s)\varphi$ und $(\hat{\exists} x \colon s)\varphi$ Formeln in $M_\Sigma(X)$. Diese Formeln werden als *Modalerweiterungen* der Prädikatenlogik bezeichnet. $\hat{\exists}$ und $\hat{\forall}$ werden als globaler Existenz– bzw. Allquantor bezeichnet. Durch sie gebundene Formeln werden als *global quantifiziert* bezeichnet.

3. Sind φ, ψ modallogische Formeln, so sind auch $\Box\varphi$ und $\Diamond\varphi$ Formeln in $M_\Sigma(X)$.

4. Keine anderen Zeichenketten sind Formeln in $M_\Sigma(X)$. $\square$

Die umständlich erscheinende doppelte Definition der aussagenlogischen Konnektive[1] ist notwendig, um ihren Gebrauch innerhalb von quantifizierten prädikatenlogischen Formeln und außerhalb der neuen modalen Operatoren zu ermöglichen.

[1] Gemäß ihrer formalen Semantik sind die prädikatenlogischen Konnektive und ihre Modalerweiterungen unterschiedliche Operatoren, da sie auf verschiedenen Strukturen definiert sind. Auf eine unterschiedliche syntaktische Notation (z.B. durch $\hat{\vee}$, $\hat{\wedge}$ etc.) wurde aus Gründen der Lesbarkeit von Formeln verzichtet. Die teilweise auftretende Mehrdeutigkeit (z.B. in **true** $\vee$ **false** für $\vee$) für aussagenlogische Konnektive wird hierbei in Kauf genommen, da beide Interpretationen zu derselben Semantik führen.

Die Semantikdefinition für modallogische Formeln ist eine Erweiterung der Definition 2.1.8. Die Definition ist aufgeteilt in die Definition für die sogenannte Modalerweiterung der Prädikatenlogik und die Definition der Semantik der Modaloperatoren, um eine direkte Übernahme des ersten Teiles in der Semantikdefinition von temporalen Formeln über Kripkestrukturen im nächsten Kapitel zu erleichtern. Die Definition der Semantik der globalen Quantifizierungen folgt gesondert im Kapitel 2.3.

Eine *globale Belegung* $\hat{\beta}$ ordnet jeder Weltinterpretation in einer Kripkestruktur eine Belegungsfunktion im prädikatenlogischen Sinne zu.

Definition 2.2.3 Eine *globale Belegung* $\hat{\beta}$ ist eine Menge von Belegungen, die für eine gegebene Kripkestruktur K_Σ jeder Welt aus $\mathcal{W}$ eine Belegung im prädikatenlogischen Sinn zuordnet :

$$\hat{\beta} = \left\{ \beta_\alpha : X \rightarrow \bigcup_{s \epsilon S} I(\alpha)(s) \;\middle|\; \alpha \,\epsilon\, \mathcal{W} \right\}$$

wobei gilt

$$\beta_\alpha(x{:}\, s) \,\epsilon\, I(\gamma)(s) \wedge R(\alpha, \gamma) \Rightarrow \beta_\alpha(x{:}\, s) = \beta_\gamma(x{:}\, s).$$

Ist eine Variable x in einer Welt α an einen Wert aus der passenden Trägermenge gebunden, so auch in allen zu α relevanten Welten, die diesen Wert ebenfalls enthalten. Die einzelnen β_α werden als *lokale Belegungen* bezeichnet. □

Durch die Notation der globalen Belegungen ist sichergestellt, daß eine Variable in jeder Weltinterpretation einen definierten Wert annimmt. Probleme können darum bei leeren Trägermengen für einzelne Weltinterpretationen auftreten. Ferner ist durch die Definition sichergestellt, daß eine Wechsel der Wertzuordnung durch die lokalen Belegungen nur dann auftritt, wenn er aufgrund des Fehlens des Wertes in einer Weltinterpretation notwendig wird. Bei identischen Trägermengen für alle Weltinterpretationen entsprechen globale Belegungen direkt der Belegungsdefinition der Prädikatenlogik, die jeder Variablen genau einen Wert zuordnet.

Definition 2.2.4 Sei φ' eine prädikatenlogische Formel aus $P_\Sigma(X)$ und seien φ und ψ modallogische Formeln. Die Gültigkeit einer Formel aus $M_\Sigma(X)$ für eine gegebene Kripkestruktur K_Σ und eine globale Belegung $\hat{\beta}$ in einer Welt $\alpha \,\epsilon\, \mathcal{W}$ ist definiert durch

$$\begin{array}{lll}
(K_\Sigma, \hat{\beta}) \models_\alpha \varphi' & \text{gdw.} & (I(\alpha), \beta_\alpha) \models \varphi'. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \neg\varphi & \text{gdw.} & \text{nicht } (K_\Sigma, \hat{\beta}) \models_\alpha \varphi. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \wedge \psi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \varphi \text{ und } (K_\Sigma, \hat{\beta}) \models_\alpha \psi. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \vee \psi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg(\neg\varphi \wedge \neg\psi). \\
(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \Rightarrow \psi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg\varphi \vee \psi. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \iff \psi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi).
\end{array}$$

Die Semantik der Modalerweiterung der prädikatenlogischen Operatoren entspricht also direkt den Festlegungen in der Definition 2.1.8. Die Semantik der Modaloperatoren wird wie folgt festgelegt :

$$
\begin{array}{lll}
(K_\Sigma, \hat{\beta}) \models_\alpha \Box\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\gamma \varphi \text{ gilt für alle zu } \alpha \text{ relevanten} \\
 & & \text{Welten } \{\gamma \mid (\alpha, \gamma) \mathrel{\epsilon} R\}. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \Diamond\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg\Box\neg\varphi
\end{array}
$$

□

Aufgrund der Quantifizierungen über die relevanten Welten in der Definition des $\Box$–Operators werden "$\Box$" und "$\Diamond$" auch als *modale Quantoren* bezeichnet. Zwischen ihnen herrscht dieselbe Dualität $\Box\varphi \iff \neg\Diamond\neg\varphi$ wie zwischen dem Allquantor und dem Existenzquantor.

2.3 Quantifizierung in Kripkestrukturen

Während sich die Semantik einer modalen Aussagenlogik direkt durch eine Quantifizierung über die zu einer Welt relevanten anderen Welten definieren läßt, ist die Erweiterung auf eine Prädikatenlogik erster Stufe problematisch. Insbesondere führt das fehlende intuitive Verständnis der Wechselwirkung zwischen Quantifizierung einerseits über Variablenbereiche und andererseits über relevante Welten in der Literatur zu unterschiedlichen Semantikdefinitionen für quantifizierte Formeln. Ein Beispiel hierfür ist die Diskussion über die notwendige Gültigkeit der sogenannten *Barcan'schen Formel* $((\forall x)\Box\varphi) \Rightarrow (\Box(\forall x)\varphi)$ in Kripkestrukturen einer Theorie (siehe z.B. [HC68]).

Ein schwerwiegenderes Problem ist aber, daß in der obigen, sehr allgemeingehaltenen Definition unterschiedliche Trägermengen für Sorten in den einzelnen Welten möglich sind. Unterschiedlichen Trägermengen sind eine Möglichkeit zur semantischen Modellierung des Einfügens und Löschens in Datenbanken (siehe Kapitel 5). Eine alternative Vorgehensweise zur Modellierung des Einfügens und Löschens von Objekten besteht darin, in jeder Welt eine Teilmenge einer festen Trägermenge als aktuelle Objekte auszuzeichnen [LEG85, LS87, Li87]. Auf diesen Ansatz wird in dieser Arbeit nicht näher eingegangen.

Die variierenden Trägermengen verhindern eine direkte modifizierte Übernahme der Definition der Bedeutung des Allquantors aus Definition 2.1.8, da in den einzelnen Welten unterschiedliche lokale Belegungen möglich sind. Viele Erweiterungen von modalen Aussagenlogiken zu modalen Prädikatenlogiken gehen darum von einer festen Sortenträgermenge für alle Welten aus, z.B. [HC68]. Eine frühe Diskussion dieser Problematik kann in [Kri63b] gefunden werden.

In dieser Arbeit wird ein anderer Weg eingeschlagen, diese Problematik in den Griff zu bekommen. Es wird zwischen einer Quantifizierung innerhalb einer Weltinterpretation gemäß der Definition der Prädikatenlogik und Quantifizierungen innerhalb einer Kripkestruktur unterschieden. Diese etwas umständlich erscheinende Vorgehensweise ist auch durch die im Kapitel 5 beschriebene Spezifikationsmethodik für Datenbankschemata motiviert, in der im wesentlichen nur Quantifizierungen in einzelnen Datenbankzuständen (die modalen Welten entsprechen) eingesetzt werden. Die Definitionen werden so allgemein gehalten, daß andere Ansätze, z.B. feste Trägermengen für alle Welten einer Kripkestruktur, als Spezialfälle in den Definitionen berücksichtigt sind.

Um eine semantisch fundierte Definition von Variablenquantifizierungen in Kripkestrukturen zu erreichen, müssen entweder die globalen Belegungen, über die die Quantifikationen variieren, oder die 'Reichweite' der Belegungen (d.h. genauer der Bereich der Kripkestruktur, der für die Auswertung unter dieser Belegung benutzt wird) in einer Kripkestruktur eingeschränkt werden. In dieser Arbeit werden beide Ansätze verfolgt und formal eingeführt. Der Grund dafür ist, daß für verschiedene temporal erweiterte Modallogiken Einschränkungen auf eine der beiden Alternativen zwingend sind.

Beide Ansätze zur Variablenquantifizierung in Kripkestrukturen mit unterschiedlichen Trägermengen führen zu Quantifizierungen, die die folgenden Forderungen erfüllen :

- Für feste Trägermengen in allen Weltinterpretationen haben beide Ansätze dieselbe Semantik, die der direkten Erweiterung der prädikatenlogischen Definition auf globale Belegungen anstatt einfacher Belegungen entspricht.
- Die Zuordnung eines Elementes der Sortenträgermenge zu einer Variablen besteht für alle Weltinterpretationen, die für die Auswertung der gebundenen Formelbestandteile benutzt werden. Diese Weltinterpretationen sind durch Erreichbarkeit über die Relevanzrelation festgelegt.

Die beiden Ansätze zur Variablenquantifizierung in Kripkestrukturen, die globale Quantifizierung und die Reduktquantifizierung, werden in den folgenden Abschnitten 2.3.1 und 2.3.2 definiert. Im Anschluß wird ihre Eignung als Grundlage für die Spezifikation im konzeptionellen Datenbankentwurf diskutiert.

2.3.1 Globale Quantifizierung

Als erste Möglichkeit für Variablenquantifizierungen in Kripkestrukturen wird die *globale Quantifizierung* eingeführt, die die Menge der in Frage kommenden Belegungen einschränkt, aber die gebundene Formel für diese Belegungen in der unveränderten Kripkestruktur auswertet.

Die Definition der globalen Belegungen ermöglicht Belegungen in Kripkestrukturen, die derselben Variable in verschiedenen Weltinterpretationen über ihre lokalen Belegungen verschiedene Werte zuordnen. Für die Definition der globalen Quantifizierung sind diejenigen globalen Belegungen interessant, für die dies nicht gilt. Mit R^* wird die transitive Hülle der Relation R bezeichnet.

Definition 2.3.1 Eine Belegung β heißt *gemeinsame Belegung bezüglich einer Welt* α für eine Kripkestruktur K_Σ, kurz $\beta \in GEM(K_\Sigma, \alpha)$, falls sie in den möglichen Belegungen aller von α aus erreichbaren Welten enthalten ist. β ist somit eine Belegung, die folgenden Bedingungen genügt :

1. β ist eine mögliche Belegung in α, also : $\beta \in BEL(X, I(\alpha))$.
2. Alle Variablen in X werden in Elemente der Trägermengen abgebildet, die allen von α aus über die Relevanzbeziehung erreichbaren Welten gemeinsam sind :

$$\beta \colon X \to \bigcup_{s \in S} \left(\bigcap_{R^*(\alpha,\gamma)} I(\gamma)(s) \right)$$

Jede gemeinsame lokale Belegung $\beta \colon X \to \bigcup_{s\in S} I(\alpha)(s) \in GEM(K_\Sigma, \alpha)$ induziert eine globale Erweiterung $[\beta]$ von β, für die gilt :

$$([\beta]_\gamma = \beta) \quad \text{für alle Welten } \{\gamma \mid R^*(\alpha, \gamma)\}$$

Die Erweiterung ist nur für gemeinsame lokale Belegungen definiert und nur für von α über R^* erreichbare Welten festgelegt. □

Die gemeinsamen lokalen Belegungen bestimmen durch ihre globalen Erweiterungen diejenigen globalen Belegungen, die in allen von der Welt α aus über die Relevanzrelation (eventuell über mehrere Stufen) erreichbaren Weltinterpretationen den Variablen dieselben Werte wie in α zuordnen. Die Zuordnung der Werte zu den Variablen bleibt also auch bei der Auswertung geschachtelter modaler Quantoren erhalten.[2]

[2] Für eine gegebene modale Formel bräuchten eigentlich anstatt der überhaupt erreichbaren Welten (also der über R^* erreichbaren) nur diejenigen Welten betrachtet werden, die in einer festen maximalen Relevanzstufe für α liegen, nämlich der maximalen Schachtelungstiefe modaler Quantoren in der quantifizierten Formel. Dies ergibt eine größere Anzahl von in Frage kommenden globalen Belegungen.

Dieser Ansatz wurde nicht weiter verfolgt, da die Relevanzbeziehung für die in Kapitel 3 eingeführten temporalen Quantoren transitiv ist und für die temporalen Logiken somit beide Varianten zusammenfallen.

Definition 2.3.2 Sei φ eine modallogische Formel und $x\colon s \in X$ eine Variable aus X. Die Gültigkeit einer global quantifizierten Formel aus $M_\Sigma(X)$ für eine gegebene Kripkestruktur K_Σ und eine globale Belegung $\hat{\beta}$ in einer Welt $\alpha \in \mathcal{W}$ ist definiert durch

$$
\begin{array}{lll}
(K_\Sigma, \hat{\beta}) \models_\alpha (\hat{\forall} x)\varphi & \text{gdw.} & (K_\Sigma, [\beta']) \models_\alpha \varphi \text{ für alle gemeinsamen Belegungen} \\
 & & \beta' \in \mathit{GEM}(K_\Sigma, \alpha)\text{, die sich von } \beta_\alpha \text{ nur im Wert von} \\
 & & x \text{ unterscheiden.} \\
(K_\Sigma, \hat{\beta}) \models_\alpha (\hat{\exists} x)\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg((\hat{\forall} x)\neg\varphi).
\end{array}
$$

□

In dieser Definition der globalen Allquantifizierung über ganzen Kripkestrukturen sind zwei intuitiv plausibel erscheinende Festlegungen getroffen worden :

- Die für die Auswertung einer Variablenquantifizierung benutzten globalen Belegungen ordnen Variablen in allen betroffenen Weltinterpretationen den gleichen, definierten Wert zu.
- Die Welt α, bezüglich der die Formel ausgewertet wird, ist als die für die Quantifizierung relevante Welt ausgezeichnet.

Da die Semantik der modallogischen Formeln jetzt vollständig festgelegt ist, kann jetzt auch die Gültigkeit einer Formel unabhängig von einer aktuellen Welt definiert werden.

Definition 2.3.3 Eine modallogische Formel φ heißt *gültig in einer Kripkestruktur* K_Σ unter einer globalen Belegung $\hat{\beta}$, kurz $(K_\Sigma, \hat{\beta}) \models \varphi$, falls sie in allen Welten in $\mathcal{W}$ gültig ist. □

Die Benutzung von globalen Erweiterungen gemeinsamer Belegungen für die globale Quantifizierung hat oft unerwünschte Folgen. In vielen Fällen existieren gar keine globale Erweiterungen, da die betroffenen Werte aus den Sortenträgermengen in über R^* erreichbaren Welten nicht enthalten sind.

Die Alternative ist, Formeln nur in den Weltinterpretationen auszuwerten, in denen die zugrundeliegende Belegung den Variablen nur definierte Werte zuordnet. Diese Methode entspricht besonders für die Definition der Gültigkeit von Formeln mit freien Variablen eher der intuitiven Bedeutung von Formeln. Für diese Alternative wird nicht die Menge der in Frage kommenden Belegungen, sondern die Menge der für die Auswertung der quantifizierten Formel benutzten Welten eingeschränkt.

Auf die globale Quantifizierung kann aber nicht vollständig zugunsten der im folgenden Abschnitt 2.3.2 beschriebenen *Reduktquantifizierung* verzichtet werden, da die in Abschnitt 3.3 beschriebene Manna–Pnueli–Logik keine Reduktquantifizierung zuläßt.

2.3.2 Reduktquantifizierung

Die Reduktquantifizierung wird als alternative Methode der Variablenquantifizierung eingeführt und dient als theoretische Grundlage der hier und in Kapitel 3.4 erfolgenden Definition der Gültigkeit von Formeln mit freien Variablen.

Die Gültigkeit von geschlossenen modallogischen Formeln ist unabhängig von einer globalen Belegung $\hat{\beta}$. Der Wert einer modallogischen Formel mit einer freien Variablen x kann nach obiger Definition globaler Belegungen verschiedene Werte in verschiedenen Welten annehmen, falls der Wert x bezüglich einer lokalen Belegung in einer anderen Welt nicht enthalten ist. Um willkürliche Formelauswertungen bei der Definition von belegungsunabhängiger Gültigkeit zu vermeiden, werden folgende Vereinbarungen getroffen.

Definition 2.3.4 Der *Definitionsbereich* (oder auch *Existenzbereich*) einer Belegung $\beta \in BEL(X, I(\alpha))$ in einer Kripkestruktur $K_\Sigma = (\mathcal{W}, R, I, \alpha_0)$ bezüglich der Welt α oder kurz das *β–Redukt* von K_Σ, notiert als $K_\Sigma|^\alpha_\beta$, ist eine Kripkestruktur definiert durch

1. eine Weltenmenge $\mathcal{W}' := \{\gamma \in \mathcal{W} \mid \beta \in BEL(X, I(\gamma))\}$,
2. eine Relevanzrelation $R' := R|_{\mathcal{W}' \times \mathcal{W}'}$
3. eine auf die neue Weltenmenge eingeschränkte Weltinterpretation $I' := I|_{\mathcal{W}'}$ und
4. die aktuelle Welt $\alpha'_0 := \alpha$. □

Die Reduktquantoren $\exists^*$ und $\forall^*$ variieren über alle Belegungen einer Welt α, und durch sie gebundene Formeln werden in den durch die jeweilige Belegung definierten β–Redukten ausgewertet. Die Reduktquantoren wurden bisher nicht in die syntaktische Festlegung der modalen Formeln aufgenommen, da sie im Gegensatz zur Globalquantifizierung in Modallogiken nicht definiert sind, die die Bildung der Definitionsbereiche von Belegungen zur Formelauswertung nicht erlauben.

Definition 2.3.5 Sei φ eine modallogische Formel und $x\colon s \epsilon X$ eine Variable aus X. Dann sind auch $(\forall^* x)\varphi$ und $(\exists^* x)\varphi$ modallogische Formeln in $M_\Sigma(X)$. □

Definition 2.3.6 Sei φ eine modallogische Formel und $x\colon s \epsilon X$ eine Variable aus X. Die Gültigkeit einer reduktquantifizierten Formel aus $M_\Sigma(X)$ für eine gegebene Kripkestruktur K_Σ und eine globale Belegung $\hat{\beta}$ in einer Welt $\alpha \epsilon \mathcal{W}$ ist definiert durch

$$
\begin{array}{lll}
(K_\Sigma, \hat{\beta}) \models_\alpha (\forall^* x)\varphi & \text{gdw.} & (K_\Sigma|^{\alpha}_{\beta}, [\beta]) \models_\alpha \varphi \text{ für alle Belegungen } \beta \epsilon \\
 & & \mathit{BEL}(X, I(\alpha)), \text{ die sich von } \beta_\alpha \text{ nur im Wert von} \\
 & & x \text{ unterscheiden.} \\
(K_\Sigma, \hat{\beta}) \models_\alpha (\exists^* x)\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg((\forall^* x)\neg\varphi).
\end{array}
$$

□

Analog zur prädikatenlogischen Definition wird die Semantik von modallogischen Formeln mit freien Variablen über eine implizite Allquantifizierung gemäß der Reduktquantifizierung festgelegt.

Definition 2.3.7 Eine Formel φ heißt *gültig in einer Welt* α einer Kripkestruktur K_Σ, notiert als $(K_\Sigma) \models_\alpha \varphi$, wenn gilt

$$(K_\Sigma) \models_\alpha \varphi \quad \text{gdw.} \quad (K_\Sigma|^{\alpha}_{\beta}, [\beta]) \models_\alpha \varphi \text{ für alle } \beta \epsilon \mathit{BEL}(X, I(\alpha))$$

Die Notationen $(K_\Sigma) \models \varphi$, $\models_\alpha \varphi$ und $\models \varphi$ können analog zu den bisherigen Notationen eingeführt werden. □

2.3.3 Vergleich der beiden Ansätze

Die vorgestellten Ansätze der globalen Quantifizierung und der Reduktquantifizierung ermöglichen beide die Variablenquantifizierung in Kripkestrukturen, ohne teilweise partielle lokale Belegungen für einzelne Weltinterpretationen einzubeziehen. Derartige teilweise partiellen Belegungen würden entweder eine dreiwertige Logik mit zum Teil undefinierten atomaren Formeln oder eine explizite Festlegung der Auswertung von atomaren Formeln mit undefinierten Variablen erfordern (vergl. z.B. die Festlegungen in [Li87] für die Auswertung von Formeln in der leeren Zustandsfolge).

Eine explizite Festlegung des Wertes atomarer Formeln mit undefinierten Variablen, z.B. durch die Auswertung zu **false**, kann unter anderem dazu führen, daß prädikatenlogische Theoreme nur dann gelten, wenn alle Variablen definiert sind. Ein Beispiel ist die Formel $(x \leq y) \vee (y \leq x)$. Dies führt zu Problemen beim formalen Einsatz von Deduktionssystemen und auch beim intuitiven Verständnis der Gültigkeit von Formeln. Unter diesem Gesichtspunkt sind beide vorgestellten Quantifizierungsmethoden als semantische Grundlage für die Spezifikation geeignet.

Für die Eignung der beiden Ansätze für den Datenbankentwurf ist insbesondere die Behandlung von neu eingefügten und gelöschten Informationen von Interesse. Die Entwicklung einer Datenbank wird hierbei als spezielle Kripkestruktur betrachtet, deren Welten zeitlich angeordnet sind und deren Weltinterpretationen den Datenbankinhalten entsprechen (siehe Kap. 5). Die zeitliche Anordnung spiegelt sich in der Erreichbarkeit über die Relevanzrelation wieder (vergl. Kap. 3).

Die globale Quantifizierung ermöglicht zwar die intuitiv korrekte Handhabung von 'eingefügten' Werten einer Sorte, aber führt bei Löschungen zu Anomalien. Werte, die ab einem bestimmten Zeitpunkt nicht mehr in den Weltinterpretationen enthalten sind, werden in den globalen Quantifizierungen nicht berücksichtigt. Neu eingefügte Werte werden ab dem ersten Auftreten in den globalen Quantifizierungen korrekt berücksichtigt.

Die Reduktquantifizierung ermöglicht hingegen die formale Handhabung sowohl von Einfügungen wie auch von Löschungen. Die Auswertung für eine Belegung erfolgt nur in dem Teil der Kripkestruktur bzw. der Datenbankentwicklung, in der diese Belegung den Variablen aktuelle Werte zuweist. Diese Vorteile sprechen eindeutig für den Einsatz der Reduktquantifizierung. Deshalb wurde die Gültigkeit von Formeln mit freien Variablen auch über die Reduktquantifizierung eingeführt.

Die Bedeutung der globlen Quantifizierung ist darin begründet, daß die Spezifikation in der von Manna und Pnueli eingeführten temporalen Logik ([MP79, MP81], Kap. 3.3) den Einsatz der Reduktquantifizierung nicht ermöglicht. Der Grund dafür ist, daß die Formelauswertung in dieser Logik nur für unendliche Kripkestrukturen definiert ist — und nicht für endliche Redukte dieser Kripkestruktur. Da diese Logik auch in der Spezifikation von Datenbankentwicklungen benutzt wird [ELG84, LEG85, LS87, Li87], werden in dieser Arbeit beide Ansätze behandelt.

Die Verwendung der klassischen Manna–Pnueli–Logik, in der keine Reduktquantifizierung möglich ist, erfordert zusätzliche Festlegungen für die Formelsemantik bei Löschungen, die nicht in der Logik selber begründet sind. In [SL87, LS87] werden rein syntaktische Kriterien für Formeln angegeben, die ein Löschen von Informationen in dem gerade aktuellen Datenbankzustand erlauben. Die dortigen Festlegungen werden durch die in dieser Arbeit vorgestellte finit-lineare temporale Logik mit Reduktquantifizierung nachträglich semantisch gerechtfertigt.

3 Temporale Logik

Ziel der Spezifikation von Objektlebensläufen ist eine abstrakte, deskriptive Beschreibung des zeitlichen Verhaltens von Objekten. Das zeitliche Verhalten äußert sich durch das Verändern von Objekteigenschaften im Lauf der Zeit. Speziell im Datenbankentwurf ist hierfür eine Erweiterung der Prädikatenlogik erster Stufe als Spezifikationssprache wünschenswert, da diese Logik als Spezifikationssprache für die statische Struktur des Datenbankinhalts besonders geeignet ist.

Bereits in der Einleitung des Kapitels über Modallogiken wurde auf die Möglichkeit hingewiesen, das zeitliche Verhalten direkt in der klassischen Prädikatenlogik zu spezifizieren. Hierbei wird jedes Prädikat und jede Funktion in der Signatur um eine zusätzliche Argumentsorte der vorgegebenen geordneten Zeitsorte *time* erweitert, um die Gültigkeit einer atomaren Formel bzw. den Wert eines Terms zu einem Zeitpunkt aus *time* beschreiben zu können. Als Standardinterpretation für *time* wird dabei oft eine Untermenge der ganzen Zahlen gewählt.

Problematisch und fehleranfällig wird dieser Ansatz besonders im klassischen Fall einer einsortigen Logik, da Zeitpunkte und andere Objekte nicht unterschieden werden. Eine ausreichend ausdrucksstarke temporale Aussagenlogik ist derart gar nicht zu formulieren, da Aussagen wie 'ab jetzt gilt immer Aussage A' nur durch eine Variablenquantifizierung der Form $(\forall t\colon time)((t \geq t_0) \Rightarrow A(t))$ formuliert werden können (t_0 sei eine Konstante der Sorte *time*, die den aktuellen Zeitpunkt modelliert).

Um nur solche Σ–Strukturen als Interpretationen eines derartigen temporal erweiterten Prädikatenkalküls zuzulassen, die das beabsichtigte zeitliche Verhalten aufweisen, muß dies durch eine zusätzliche Axiomatisierung festgelegt werden. Die zusätzliche Axiomatisierung führt entweder zu einem erhöhten Spezifikationsaufwand, wenn sie dem Ersteller einer Spezifikation überlassen wird, oder zu einer versteckten Einschränkung der gültigen Formeln, die die Fehleranfälligkeit ebenfalls erhöht.

Schon früh wurden deshalb Erweiterungen der klassischen Logiken um Zeitaspekte vorgeschlagen, um eine natürlichere Modellierung von zeitlichen Veränderungen zu ermöglichen [Pr67, RU71]. Den meisten dieser Ansätze liegen als Interpretationsstrukturen Kripkestrukturen zugrunde.

Wünschenswert ist eine Erweiterung der Prädikatenlogik um zusätzliche temporale Operatoren, die eine elegante direkte Formulierung von Aussagen mit zeitlichem Bezug ermöglichen. Eine Auswahl derartiger temporaler Operatoren ist

always A	immer gilt A
sometime A	irgendwann gilt A
A	jetzt gilt A
next A	im nächsten Zustand gilt A (dies setzt die Existenz eines eindeutig bestimmten nächsten Systemzustands voraus)
A **until** B	A gilt solange bis B eintritt
...	...

Auf Kripkestrukturen läßt sich eine allgemeine temporale Logik definieren, die den intuitiven Zeitbegriff widerspiegelt. Mehrere spezielle Ansätze werden im folgenden vorgestellt und die Wahl eines Ansatzes als Spezifikationskalkül motiviert. Dieser Logikkalkül, eine zuerst von Z. Manna und A. Pnueli in [MP81] vollständig beschriebene erweiterte lineare temporale Logik, wird im folgenden formal eingeführt und anderen vergleichbaren Ansätzen gegenübergestellt. Die ursprüngliche Definition der Semantik dieses Kalküls wird in dieser Arbeit dahingehend modifiziert, daß auch endliche Zeitabläufe als Interpretationen auftreten können.

Der Rest des Kapitels ist wie folgt gegliedert. Der erste Abschnitt 3.1 beschäftigt sich mit den verschiedenen Modellbildungen, in denen temporale Aussagen interpretiert werden. Diese Modellbildungen werden im Abschnitt 3.2 auf die in Kapitel 2 eingeführten modallogischen Konzepte zurückgeführt. Die Modellbildung einer *linearen Zeit* führt zu der Definition einer linearen temporalen Logik in Kapitel 3.3, die an die Vorschläge von Manna und Pnueli [MP81] angelehnt ist.

Die ursprüngliche Definition dieser Logik basiert ausschließlich auf unendlichen Kripkestrukturen. Die Erweiterung der Formelsemantik auf endliche Zustandsfolgen führt zur Definition der *finit-linearen temporalen Logik* in Abschnitt 3.4, die in den anschließenden Kapiteln ausführlich analysiert und als Spezifikationslogik für das dynamische Datenbankverhalten eingesetzt wird.

3.1 Modelle für temporale Logikkalküle

Kripkestrukturen können dahingehend interpretiert werden, daß die verschiedenen Elemente in $\mathcal{W}$ den Zustand einer Welt zu verschiedenen Zeitpunkten modellieren. Die Relevanzbeziehung — bzw. in einigen Ansätzen auch mehrere unterschiedliche Relevanzbeziehungen über derselben Weltenmenge gleichzeitig — beschreiben dabei eine Art von zeitlicher Relevanz, z.B. die zeitliche Anordnung oder den zeitlich nächsten Zustand. Die verschiedenen Ansätze für temporale Logikkalküle unterscheiden sich hauptsächlich in den Anforderungen an die Relevanzbeziehung(en) und darauf aufbauend in den möglichen temporalen Operatoren.

Die Art der benutzten Relevanzbeziehungen hängt sowohl von dem abstrakten Verständnis des Zeitbegriffs als auch von der jeweils beabsichtigten Anwendung ab. Allgemein existiert eine Beziehung "folgt zeitlich auf" zwischen den einzelnen Welten, die allerdings oft nicht explizit angegeben wird. Diese Beziehung muß den Forderungen einer zeitlichen Anordnung der Zeitpunkte genügen, d.h. insbesondere daß sie transitiv ist. Oft wird zusätzlich Antisymmetrie und Reflexivität gefordert. Ist dies der Fall, bildet die Relevanzbeziehung eine Ordnungsrelation auf der Weltenmenge. Prinzipiell können die Weltenmenge $\mathcal{W}$ und die Relevanzbeziehung R durch eine beliebige Menge mit Ordnungsrelation, einer sogenannten Halbordnung, festgelegt werden.

Für einen temporalen Logikkalkül sind folgende Festlegungen nötig :

1. **Ist die Weltenmenge $\mathcal{W}$ abzählbar oder überabzählbar ?**

   Der Fall einer überabzählbaren Weltenmenge, z.B. $\mathcal{W}$ isomorph zu den reellen Zahlen, wurde in Zusammenhang mit Informatikanwendungen, die sich in der Regel mit diskreten Systemen befassen, bisher wenig untersucht. Von Interesse ist diese Modellbildung für kontinuierliche dynamische Systeme.

   Eng verbunden mit dieser Fragestellung ist die Frage, ob der Zeitablauf in diskreten endlichen Schritten modelliert werden kann, ob also stets ein direkter zeitlicher Folgezustand bzw. eine endliche Menge von direkten Folgezeitpunkten existiert.

2. **Ist die Weltenmenge $\mathcal{W}$ endlich oder unendlich ?**

   Die in der Literatur vorgestellten Ansätze für temporale Logiken gehen fast alle von einer unendlichen Weltenmenge aus, die isomorph zu den ganzen bzw. natürlichen Zahlen oder zu Pfaden in einer unendlichen Baumstruktur ist. Diese Ansätze werden in den Abschnitten 3.2 und 3.3 näher erläutert.

   Ein neuer Ansatz für eine temporale Logik für endliche Weltenmengen ist Inhalt von Abschnitt 3.4.

3. **Gibt es einen Anfangszeitpunkt t_0, der vor allen anderen Zeitpunkten liegt (bzw. als symmetrische Forderung einen Endzeitpunkt) ?**

   Derartige Zeitpunkte bilden obere bzw. untere Schranken in einer aus der Weltenmenge und der Relevanzrelation gebildeten Halbordnung. Temporale Logiken mit Anfangszeitpunkten modellieren eine Art 'zukunftsgerichtete' Beschreibung eines dynamischen Systems, daß sich ausgehend von einem initialen Zustand weiterentwickelt, und entsprechen mehr der menschlichen Intuition von einer zeitlichen Entwicklung.

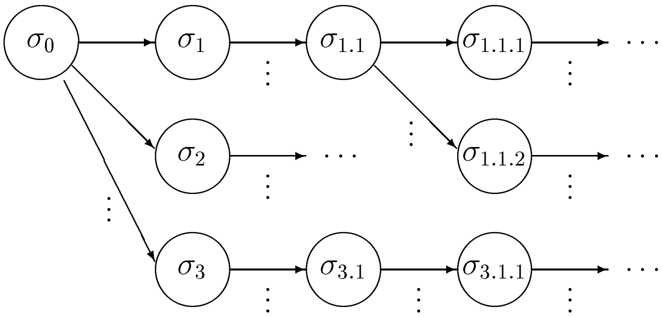


Abbildung 1: Interpretationsstruktur einer verzweigten temporalen Logik

Die in Informatikanwendungen verbreiteten temporalen Logikkalküle (eine aktuelle Übersicht geben z.B. [Mo86, Krö87] im Bereich der Programmspezifikation) gehen von abzählbar unendlichen Weltenmengen mit eindeutig bestimmten direkten Folgezeitpunkten aus. Diese Kalküle werden in Kripkestrukturen mit zwei Relevanzbeziehungen interpretiert, der zeitlichen Ordnungsrelation und der Relation 'ist direkter zeitlicher Nachfolger'. Die Semantik der temporalen Formeln wird bezüglich eines Anfangs– (oder seltener End–) Zeitpunktes definiert.

In einigen Ansätzen werden zwei derartige Kalküle, einer mit Anfangszeitpunkt und einer mit Endzeitpunkt, zu einem einzigen Kalkül verschmolzen, um temporale Quantifizierung über Vergangenheit und Zukunft in einem Kalkül zu ermöglichen (z.B. in [Se80, Ku84]). Dies wirft aber semantische Probleme und vor allem Schwierigkeiten bei einer algorithmischen Überwachung derartiger Spezifikationen auf, da zu jedem Zeitpunkt Aussagen über die gesamte Vergangenheit und Zukunft der spezifizierten Systeme formuliert werden können.

Bei den temporalen Logikkalkülen über nichtendlichen Strukturen lassen sich in der Literatur insbesondere zwei verbreitete Ansätze unterscheiden :

1. *Verzweigte temporale Logik* (in der englischsprachigen Literatur *branching time logic*, [La80, BMP81, CE81, ES85, CES86]). Die zugrundeliegenden Halbordnungen werden durch die Präfixbeziehungen von Pfaden in einer unendlichen Baumstruktur gebildet. Ein Ausschnitt einer derartigen Struktur ist in Abbildung 1 dargestellt.

   Die Längen der Pfadpräfixe entsprechen den Zeitpunkten, und Welten mit Präfixen gleicher Länge entsprechen alternativen Zukünften. Es wird oft eine maximale Anzahl für mögliche Söhne in der Baumstruktur gefordert, um immer eine endliche Menge direkter zeitlicher Nachfolger zu gewährleisten.

   Verzweigte temporale Logiken modellieren die Vorstellung einer nichtdetermi-

$$\sigma_0 \longrightarrow \sigma_1 \longrightarrow \sigma_2 \longrightarrow \sigma_3 \longrightarrow \cdots$$

Abbildung 2: Interpretationsstruktur einer linearen temporalen Logik

nistischen Zukunft und haben sich insbesondere in der Spezifikation von nichtdeterministischen Prozessen und Systemen bewährt.

2. *Lineare temporale Logik* (in der englischsprachigen Literatur *linear temporal logic*, [MP81, Wo81, MW84, Mo86, Krö87]). Die Interpretationen der linearen temporalen Logik sind unendliche Folgen von Zuständen $\hat{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$ beginnend mit einem aktuellen Zustand σ_0. Der Anfang einer derartigen Folge ist in Abbildung 2 dargestellt, wie in Abbildung 1 ist nur die Beziehung 'unmittelbarer zeitlicher Nachfolger' durch Kanten dargestellt. Die Weltenmenge ist isomorph zu den natürlichen Zahlen, wodurch die Benennung der Welten $\mathcal{W} = \{\sigma_i \mid i \in I\!N\}$ motiviert wird. Eine derartige Logik liegt der in Kapitel 3.3 vorgestellten temporalen Logik nach Manna/Pnueli zugrunde.

   Der lineare Ansatz basiert auf der Idee einer deterministischen (möglicherweise allerdings noch unbekannten) Zukunft auf. Einsatzgebiete sind somit insbesondere die Spezifikation von deterministischen Programmen und Prozessen [MP81, MW84, Krö87] und einer durch deterministische Änderungsaktionen definierten Datenbankentwicklung [ELG84].

Auch die Erweiterung auf eine temporale Prädikatenlogik kann auf verschiedene Arten erfolgen. Neben der in dieser Arbeit vorgestellten Vorgehensweise, Kripkestrukturen mit unterschiedlichen Weltinterpretationen als formale Modellbildung zu verwenden (vergl. Abschnitt 2.3), wird in [Krö87, Sz87] eine Erweiterung auf eine temporale Prädikatenlogik vorgestellt, in der als Modellbildung eine feste Σ–Struktur betrachtet wird und die zeitabhängige Information durch eine Folge von Belegungen von sogenannten lokalen Variablen ausgedrückt wird. In dieser Arbeit wird dieser Ansatz, motiviert durch die Auffassung von Datenbankzuständen als verschiedene Interpretationen einer Signatur (Abschnitt 5.3), nicht weiter verfolgt.

In Kapitel 3.2 wird eine auf der Modallogik aufbauende temporale Logik vorgestellt, die temporale Logikkalküle mit Anfangszeitpunkt und direkten Folgezeitpunkten umfaßt. Als Modellbildung für die prädikatenlogische Erweiterung werden die in Abschnitt 2.2 vorgestellten Kripkestrukturen über einer gegebenen Signatur benutzt.

3.2 Temporale Erweiterung der Modallogik

Eine ausdrucksfähige temporale Logik muß zumindest die Formulierung von einfachen zeitbezogenen Aussagen der Form "immer gilt A" oder "jetzt gilt A" ermöglichen. Derartige Aussagen können direkt in Formeln der eingeführten Modallogik umgesetzt werden, wenn die Relevanzbeziehung R die "zeitliche Relevanz" korrekt widerspiegelt. Dann kann z.B. "immer gilt A" durch "$\Box A$" und "jetzt gilt A" durch die Formel "A" ausgedrückt werden.

Ist R eine Ordnungsrelation auf $\mathcal{W}$, so wird das intuitive Verständnis der zeitlichen Abfolge von Welten korrekt wiedergegeben. Eine zusätzliche Einschränkung der als Interpretationen erlaubten Kripkestrukturen durch weitere Axiomenschemata kann diese Anforderungen berücksichtigten. Z.B. wird die Transitivität der Ordnungsrelation durch das Axiomenschemata $\Box(\varphi) \Rightarrow \Box(\Box(\varphi))$ garantiert (φ ist ein Platzhalter für eine beliebige modallogische Formel). Das durch geeignete Axiomatisierung erhaltene modale Logiksystem wird in der Literatur mit S4 bezeichnet. Die Axiomatisierung von S4 kann z.B. in [HC68, Se80] gefunden werden, wo auch weitergehende Einschränkungen der Relevanzbeziehung diskutiert werden.

Dieser verbreitete Ansatz hat den Nachteil, daß Aussagen der Art "im nächsten Zeitpunkt gilt A" nicht formuliert werden können. Dieser Nachteil kann durch eine andersartige Erweiterung der Modallogik zu einer temporalen Logik vermieden werden.

Die Grundidee des hier verwendeten Ansatzes ist, daß die Relevanzbeziehung "zeitlicher Nachfolger" durch die Beziehung "direkter zeitlicher Nachfolger" eindeutig bestimmt ist und aus ihr abgeleitet werden kann. Für eine zweistellige Relation R sei die reflexive, transitive Hülle von R mit R^* bezeichnet. Ist R die Relation "direkter zeitlicher Nachfolger", dann ist die Relation "zeitlicher Nachfolger" durch R^* definiert.

Aufbauend auf der Definition der Modallogik wird nun eine temporale Logik auf allgemeinen Kripkestrukturen $K_\Sigma = (\mathcal{W}, R, I, \alpha_0)$ definiert. Die Relevanzbeziehung R soll hierbei die Bedeutung der Beziehung "direkter zeitlicher Nachfolger" haben und sollte deshalb antisymmetrisch sein. Die allgemeine zeitliche Nachfolgerbeziehung wird aus dieser Relation abgeleitet. Diese Einschränkung auf antisymmetrische Relationen wird in der allgemeinen Definition aber nicht benötigt.

Die Modaloperatoren '$\Box$' und '$\Diamond$' werden in **allnext** und **existsnext** umbenannt, um ihre temporale Bedeutung als Quantifizierung über die direkten Folgezeitpunkte hervorzuheben. Sie werden als *Nextoperatoren* bezeichnet.

Die Definition der temporalen Formeln ist eine Erweiterung der Definition 2.2.2 der modallogischen Formeln auf Seite 16. Die Menge der temporalen Formeln zu einer Signatur Σ und einer Variablenmenge X wird mit $TF_\Sigma(X)$ bezeichnet.

Definition 3.2.1 Die *temporalen Formeln* in $TF_\Sigma(X)$ sind definiert durch :

1. Jede prädikatenlogische Formel $\varphi \in P_\Sigma(X)$ ist eine temporale Formel.
2. Sind φ, ψ temporale Formeln und x eine Variable der Sorte s, so sind auch $\neg\varphi$, $(\varphi \Rightarrow \psi)$, $(\varphi \iff \psi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\hat{\forall} x{:}\, s)\varphi$ und $(\hat{\exists} x{:}\, s)\varphi$ Formeln in $TF_\Sigma(X)$. Diese Formeln werden als *Temporalerweiterungen* der Prädikatenlogik bezeichnet.
3. Ist φ eine temporale Formel, so sind auch **allnext** φ, **existsnext** φ, **always** φ und **sometime** φ Formeln in $TF_\Sigma(X)$.
4. Keine anderen Zeichenketten sind Formeln in $TF_\Sigma(X)$.

Statt der globalen Quantoren können auch die Reduktquantoren zur Variablenquantifikation herangezogen werden, falls keine Einschränkungen an die Kripkestrukturen gestellt werden, die der Reduktquantifizierung widersprechen (vergl. Kapitel 3.3). □

Die Semantikdefinition kann zum Teil direkt aus der Definition 2.2.4 der modallogischen Formeln von Seite 17 übernommen werden. Es gelten die gleichen Vereinbarungen für die globale Quantifizierung über Kripkestrukturen.

Definition 3.2.2 Seien φ' eine prädikatenlogische Formel aus $P_\Sigma(X)$, φ und ψ temporale Formeln und $x{:}\,s \in X$ eine Variable aus X. Die Gültigkeit einer Formel aus $TF_\Sigma(X)$ für eine gegebene Kripkestruktur $K_\Sigma = (\mathcal{W}, R, I, \alpha_0)$ und eine globale Belegung $\hat{\beta}$ in einer Welt $\alpha \in \mathcal{W}$ ist definiert durch

$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi'$	gdw.	$(I(\alpha), \beta_\alpha) \models \varphi'$
$(K_\Sigma, \hat{\beta}) \models_\alpha \neg\varphi$	gdw.	nicht $(K_\Sigma, \hat{\beta}) \models_\alpha \varphi$.
$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \wedge \psi$	gdw.	$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi$ und $(K_\Sigma, \hat{\beta}) \models_\alpha \psi$.
$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \vee \psi$	gdw.	$(K_\Sigma, \hat{\beta}) \models_\alpha \neg(\neg\varphi \wedge \neg\psi)$.
$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \Rightarrow \psi$	gdw.	$(K_\Sigma, \hat{\beta}) \models_\alpha (\neg\varphi \vee \psi)$.
$(K_\Sigma, \hat{\beta}) \models_\alpha \varphi \iff \psi$	gdw.	$(K_\Sigma, \hat{\beta}) \models_\alpha (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$.
$(K_\Sigma, \hat{\beta}) \models_\alpha (\hat{\forall} x)\varphi$	gdw.	$(K_\Sigma, [\beta']) \models_\alpha \varphi$ für alle gemeinsamen Belegungen β', die sich von β_α nur im Wert von x unterscheiden.
$(K_\Sigma, \hat{\beta}) \models_\alpha (\hat{\exists} x)\varphi$	gdw.	$(K_\Sigma, \hat{\beta}) \models_\alpha \neg((\forall x)\neg\varphi)$.

Die Semantik der Temporalerweiterung der prädikatenlogischen Operatoren ist identisch zur Definition 2.2.4 der Modalerweiterung auf Seite 17.

$$
\begin{array}{lll}
(K_\Sigma, \hat{\beta}) \models_\alpha \textbf{allnext}\,\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\gamma \varphi \text{ gilt für alle zu } \alpha \text{ relevanten} \\
 & & \text{Welten } \{\gamma \mid R(\alpha, \gamma)\}. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \textbf{existsnext}\,\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg\,\textbf{allnext}\,\neg\varphi \\
(K_\Sigma, \hat{\beta}) \models_\alpha \textbf{always}\,\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\gamma \varphi \text{ gilt für alle zu } \alpha \text{ bezüglich} \\
 & & R^* \text{ relevanten Welten } \{\gamma \mid (\alpha, \gamma) \in R^*\}. \\
(K_\Sigma, \hat{\beta}) \models_\alpha \textbf{sometime}\,\varphi & \text{gdw.} & (K_\Sigma, \hat{\beta}) \models_\alpha \neg\,\textbf{always}\,\neg\varphi
\end{array}
$$

Die Semantik der temporalen Operatoren **allnext** und **existsnext** ist identisch zur Semantik von '$\Box$' und '$\Diamond$' in Definition 2.2.2. Die naheliegende Definition der *temporalen Quantoren* **always** und **sometime** durch die Modaloperatoren in $K'_\Sigma = (\mathcal{W}, R^*, I, \alpha_0)$ ist nicht möglich, da in ihrem Wirkungsbereich geschachtelt einer der Nextoperatoren auftreten kann und die ursprüngliche Kripkestruktur K_Σ aus K'_Σ nicht wiedergewonnen werden kann.

Die Semantik der Reduktquantoren kann ebenfalls direkt übertragen werden. $\Box$

Satz 3.2.3 Die Beziehung zwischen der Quantifizierung über die direkten zeitlichen Nachfolger und den temporalen Quantoren wird durch folgende Äquivalenzen deutlich :

$$\textbf{always}\,\varphi \iff \varphi \wedge \textbf{allnext}(\textbf{always}\,\varphi) \tag{1}$$
$$\textbf{sometime}\,\varphi \iff \varphi \vee \textbf{existsnext}(\textbf{sometime}\,\varphi) \tag{2}$$

Beweis : Es muß nur eine der Äquivalenzen bewiesen werden, da die andere dann aus den Dualitäten der Operatoren folgt. **always** φ fordert die Gültigkeit von φ in allen Welten, die von der aktuellen Welt α entlang der Relevanzbeziehung R erreichbar sind (Definition von R^*). Dieses sind gerade die aktuelle Welt (Formelteil φ) und die von den direkten Nachfolgern aus erreichbaren Welten (Formelteil **allnext**(**always** φ)). $\Box$

Diese Äquivalenzen werden als *temporale Rekursionsregeln* bezeichnet, da sie die Gültigkeit einer temporalen Quantifizierung in einen für die aktuelle Welt relevanten Teil und einen für die zukünftigen Welten relevanten Teil aufteilen.

3.3 Lineare temporale Logik nach Manna/Pnueli

Die in Kapitel 3.2 vorgestellte temporale Logik stellt nur wenige Mindestanforderungen an die unterliegenden Interpretationsstrukturen. In Informatikanwendungen sind lineare und verzweigte temporale Logiken mit Anfangszustand verbreitet, denen weiter eingeschränkte Kripkestrukturen als Interpretationsstrukturen zugrundeliegen. Derart eingeschränkte Kripkestrukturen werden als *temporale Strukturen* bezeichnet.

Definition 3.3.1 Eine *temporale Struktur* ist eine Kripkestruktur K_Σ mit einer *antisymmetrischen* transitiven, reflexiven Hülle R^* der Relevanzbeziehung R, d.h. es gilt :

$$R^*(\alpha,\gamma) \wedge R^*(\gamma,\alpha) \Rightarrow (\alpha = \gamma)$$

Eine temporale Struktur heißt

- *endlich*, falls $|\mathcal{W}| \,\epsilon\, I\!N$ ist,
- *abbrechend*, falls es eine Welt $\alpha \,\epsilon\, \mathcal{W}$ gibt, so daß

$$\{\gamma \mid R(\alpha,\gamma)\} = \emptyset,$$

- *linear*, falls für alle Welten $\alpha \,\epsilon\, \mathcal{W}$ gilt

$$|\{\gamma \mid R(\alpha,\gamma)\}| \leq 1.$$

□

Definition 3.3.2 Für lineare temporale Strukturen ist die *Länge* der Struktur, kurz $|K_\Sigma|$, durch die Anzahl der von der aktuellen Welt über die Relevanzbeziehung erreichbaren Welten definiert :

$$|K_\Sigma| = \Big|\{\gamma \mid R^*(\alpha_0,\gamma)\}\Big|$$

□

In [Em83] werden verschiedene alternative Charakterisierungen temporaler Strukturen vorgestellt und verglichen. Die definierten temporalen Strukturen sind gerade die Kripkestrukturen, in denen R und $\mathcal{W}$ einen azyklischen Graph bilden. Den in Abschnitt 3.1 erwähnten verzweigten temporalen Logiken liegen nichtabbrechende, unendliche temporale Strukturen zu grunde. Die linearen temporalen Logiken nach [MP81] basieren auf unendlichen, nichtabbrechenden, linearen temporalen Strukturen.

Als Interpretationen einer temporalen Spezifikationslogik für Lebensläufe von informationstragenen Objekten[3], die durch deterministische Änderungen von Objekteigenschaften entstehen, bieten sich lineare temporale Strukturen an. Ein Vorteil linearer

[3]Der Begriff ‘Objekt’ wird im folgenden synonym zu der Begriffsbildung ‘Wert einer Sorte, deren Trägermenge sich zeitabhängig ändert, und dessen Eigenschaften zeitpunktabhängig sind’ benutzt. Diese Begriffsbildung ist motiviert durch den beabsichtigten Einsatz der temporalen Logik zur Spezifikation des Verhaltens von Datenbankobjekten (Kapitel 5).

Strukturen liegt darin, daß sich leicht zusätzliche temporale Operatoren wie **until** in die Logik einbauen lassen. In [Wo81, Wo82] werden beliebige reguläre Ausdrücke als temporale Operatoren in eine lineare temporale Logik eingebaut. [Mo86] baut zusätzliche temporale Operatoren ein, die Konzepten höherer Programmiersprachen entsprechen, und benutzt die erhaltene Logik als ausführbare Programmspezifikation.

Die von Z. Manna und A. Pnueli eingeführte, um zusätzliche Operatoren erweiterte lineare temporale Logik nach [MP81] wird laut [Krö87] aufgrund ihrer Verbreitung in der Programm– und Prozeßspezifikation bereits als *Manna–Pnueli–Theorie* bezeichnet.

Definition 3.3.3 Eine um zusätzliche Operatoren erweiterte temporale Logik heißt *MP–* oder *Manna–Pnueli–Logik*, falls gilt :

1. Als Interpretationen sind nur lineare, nicht abbrechende unendliche temporale Strukturen erlaubt (Abbildung 2 auf Seite 29). Die Weltenmenge bildet eine zu den natürlichen Zahlen isomorphe Folge von Welten. Die Welten werden als *Zustände* σ_i bezeichnet und die temporale Struktur wird als *Zustandsfolge* $\hat{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$ bezeichnet.

2. Reduktquantifizierung mit $\forall^*$ und $\exists^*$ ist nicht erlaubt, da das β–Redukt einer unendlichen Zustandsfolge selber endlich sein kann und die Manna–Pnueli–Logik die Semantik einer temporalen Formel für abbrechende Zustandsfolgen nicht festlegt. $\square$

Definition 3.3.4 Als Notationen werden eingeführt :

- Als abkürzende Schreibweise wird für die Weltinterpretation $I(\sigma_i)$ eines Zustands σ_i abkürzend nur σ_i geschrieben, sofern die Bedeutung aus dem Zusammenhang hervorgeht.

- Die Menge der nichtleeren Folgen von Zuständen aus $\Im_\Sigma$ wird mit ${\Im_\Sigma}^+$ bezeichnet. $\square$

In der Manna–Pnueli–Logik gibt es nur einen Nextoperator (als $\bigcirc$ oder **next** notiert), da in linearen, nicht abbrechenden temporalen Strukturen die Äquivalenz **allnext** $\varphi \iff$ **existsnext** φ gilt. Die Operatoren **always** und **sometime** werden in [MP81] als $\square$ und $\diamond$ in Anlehnung an die Modalquantoren notiert.

Definition 3.3.5 In der Manna–Pnueli–Logik sind die zweistelligen temporalen Operatoren **until** und **before** definiert durch

- *Syntax :*

  Sind φ und ψ temporale Formeln, dann auch die Formeln φ**until**ψ und φ**before** ψ.

- *Semantik :* Sei $j = \min(\{k \mid (\hat{\sigma}_k, \hat{\beta}) \models \psi\} \cup \{\infty\})$.

$$
\begin{array}{lll}
(\hat{\sigma}, \hat{\beta}) \models \varphi\ \mathbf{until}\ \psi & \text{gdw.} & \text{für alle } i,\ 0 \leq i < j, \text{ gilt : } (\hat{\sigma}_i, \hat{\beta}) \models \varphi. \\
(\hat{\sigma}, \hat{\beta}) \models \varphi\ \mathbf{before}\ \psi & \text{gdw.} & \text{ein } i \text{ existiert, } 0 \leq i < j, \text{ so daß} \\
 & & (\hat{\sigma}_i, \hat{\beta}) \models \varphi.
\end{array}
$$

$\hat{\sigma}_i$ bezeichnet die Teilfolge $\langle \sigma_i, \sigma_{i+1}, \ldots \rangle$ der Zustandsfolge $\hat{\sigma}$ beginnend mit den i–ten Zustand σ_i. □

Die neuen temporalen Operatoren werden auch z.B. als (**always** A **until** B) für (A **until** B) notiert, um die enge Beziehung zwischen dem **always** und dem **until** Operator zu betonen. Möglich ist auch die Definition verwandter temporaler Operatoren, die den Zustand σ_j in den Wirkungsbereich der Quantifizierung einschließen [LS87, Li87].

Die Definition der Semantik der neuen temporalen Operatoren wurde in der für den Manna/Pnueli–Ansatz typischen abkürzenden Notation formuliert. Die Gültigkeit wird jeweils bezüglich des Anfangszustandes σ_0 definiert. Die Relation R ist implizit durch die Reihenfolge der natürlichen Zahlen gegeben und R^* durch die Ordnungsrelation $\leq$. Da R^* antisymmetrisch ist, kann die Gültigkeitsdefinition bezüglich der anderen Zustände auf die jeweilige Restfolge $\hat{\sigma}_i$ beschränkt werden. Diese Notation kann leicht in die entsprechende Notation einer allgemeinen linearen temporalen Logik umgesetzt werden.

Satz 3.3.6 In einer MP–Logik gelten folgende temporalen Rekursionsregeln :

$$
\begin{array}{rcll}
\mathbf{always}\,\varphi & \Longleftrightarrow & \varphi \wedge \mathbf{next}(\mathbf{always}\,\varphi) & (3) \\
\mathbf{sometime}\,\varphi & \Longleftrightarrow & \varphi \vee \mathbf{next}(\mathbf{sometime}\,\varphi) & (4) \\
\varphi\,\mathbf{until}\,\psi & \Longleftrightarrow & \psi \vee (\varphi \wedge \mathbf{next}(\varphi\,\mathbf{until}\,\psi)) & (5) \\
\varphi\,\mathbf{before}\,\psi & \Longleftrightarrow & (\neg\psi \wedge \varphi) \vee (\neg\psi \wedge \mathbf{next}(\varphi\,\mathbf{before}\,\psi)) & (6)
\end{array}
$$

Beweis : Die ersten beiden Äquivalenzen folgen aus den Rekursionsregeln für die allgemeine temporale Logik. Der Beweis der anderen beiden Äquivalenzen kann analog geführt werden (siehe z.B. [LS87]). □

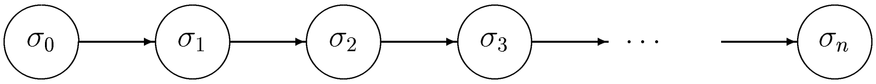


Abbildung 3: Endliche lineare temporale Struktur

Die Manna–Pnueli–Logik hat aufgrund ihrer Vorteile — insbesondere mächtige, intuitiv verständliche temporale Operatoren und die Möglichkeit der Umsetzung in eine operationale Beschreibung der Spezifikation (siehe Kapitel 4) — weite Verbreitung gefunden.

Ein Nachteil in vielen Anwendungen ist allerdings, daß die Semantik für endliche Zustandsfolgen nicht definiert ist. Insbesondere ist die Semantik von $\textbf{next}\,\varphi$ in einer Zustandsfolge der Länge 1 offen. Endliche Zustandsfolgen treten bei "abgeschlossenen Objektlebensläufen" auf, d.h. Objekte entstehen und vergehen im Lauf der Zeit. Derartige Objekte sind nur für eine endliche Teilzustandsfolge in der jeweiligen Trägermenge ihrer Sorte enthalten.

Bei Reduktquantifizierungen wäre in diesen Fällen das β–Redukt einer unendlichen Zustandsfolge für eine Belegung β selbst nur endlich, weshalb in der Definition der Manna–Pnueli–Logik auf Reduktquantifizierungen verzichtet werden mußte. Die Gültigkeit von Formeln mit freien Variablen kann aus demselben Grunde nicht aus Kapitel 2.3 übernommen werden.

3.4 Finit-lineare temporale Logik

Die Semantik einer Manna–Pnueli–Logik mit einem Nextoperator kann nicht direkt auf endliche lineare temporale Strukturen übertragen werden. Eine derartige temporale Struktur der Länge n ist in Abbildung 3 dargestellt. Die Bedeutung einer Formel der Form $\textbf{next}\,\varphi$ im letzten Zustand σ_n ist bisher undefiniert. Die Problematik wird deutlich anhand der temporalen Rekursionen :

$$\begin{aligned} \textbf{always}\,\varphi &\iff \varphi \wedge \textbf{next}(\textbf{always}\,\varphi) \\ \textbf{sometime}\,\varphi &\iff \varphi \vee \textbf{next}(\textbf{sometime}\,\varphi) \end{aligned}$$

Im letzten Zustand σ_n einer endlichen Zustandsfolge ist die beabsichtigte Bedeutung von **sometime** und **always**

$$\textbf{sometime}\,\varphi \iff \varphi \quad \text{und} \quad \textbf{always}\,\varphi \iff \varphi\ ;$$

d.h. in σ_n müßte für alle Formeln φ gelten :

$$\begin{aligned} \textbf{next}(\textbf{sometime}\,\varphi) &\iff \textbf{false} \\ \textbf{next}(\textbf{always}\,\varphi) &\iff \textbf{true} \end{aligned}$$

Dies entspricht nicht der Vorstellung, daß die Bedeutung einer mit **next** gebundenen Formel durch eine Quantifizierung über Folgezustände definiert ist.

Für endliche lineare temporale Strukturen muß also auch in der Manna–Pnueli–Logik zwischen einem schwachen und starken Nextoperator unterschieden werden. Hierzu bieten sich die beiden Nextoperatoren **allnext** und **existsnext** der temporalen Logik aus Kapitel 3.2 an. Diese Logik wird als *finit-lineare temporale Logik* bezeichnet zur Unterscheidung von der klassischen Manna–Pnueli–Logik aus Kapitel 3.3. Die Definition der Operatoren **until** und **before** kann direkt modifiziert auf endliche Folgen übernommen werden.

In der finit-linearen temporalen Logik ist Reduktquantifizierung erlaubt und semantisch abgesichert. Die Definition der Gültigkeit für Formeln mit freien Variablen wird derart modifiziert, daß über *alle* in der Zustandsfolge auftretenden Belegungen variiert wird. Dies ist möglich, da in linearen Strukturen für jedes Redukt eindeutig ein Anfangszustand identifiziert wird.

Definition 3.4.1 Eine Formel φ heißt *gültig in einer Zustandsfolge* $\hat{\sigma}$, notiert als $\hat{\sigma} \models \varphi$, wenn gilt

$$(\hat{\sigma} \models \varphi) \quad \text{gdw.} \quad (\hat{\sigma}|_{\beta}^{\sigma_{\mu.\beta}}, [\beta]) \models \varphi \text{ für alle } \beta \in \bigcup_i \mathit{BEL}(X, I(\sigma_i))$$

wobei $\mu.\beta$ den ersten Zeitpunkt angibt, an dem die Belegung β allen Variablen definierte Werte zuordnet :

$$\mu.\beta = \min\{i \mid \beta \in \mathit{BEL}(X, I(\sigma_i))\}$$

Da der Anfangszustand $\sigma_{\mu.\beta}$ durch die Belegung β eindeutig bestimmt ist, kann er in der Notation des β–Redukts und der globalen Erweiterung von β weggelassen werden. Die Zustandsfolge $\hat{\sigma}|_{\beta}^{\sigma_{\mu.\beta}}$ oder kurz $\hat{\sigma}|_{\beta}$ heißt auch *Existenzintervall* der Belegung β. $\Box$

Diese Definition folgt der Vorstellung, daß eine temporale Formel mit freien Variablen für *alle auftretenden* Belegungen während der jeweiligen zeitlichen Existenz der betroffenen Werte erfüllt werden muß. Diese Modellbildung bildet die Grundlage der Spezifikation von Objektlebensläufen in Kapitel 5 und der auf dieser basierenden Überwachungsmethoden für dynamische Integritätsbedingungen in Kapitel 6.

Die modallogische Reduktquantifizierung variiert im Gegensatz zu dieser Definition der belegungsunabhängigen Gültigkeit nur über die Belegungen des initialen Zustands. Die Modifikation der Reduktquantifizierung auf alle noch später aktuell werdenden Belegungen ist für die finit-lineare temporale Logik ebenfalls möglich und wird als *erweiterte Reduktquantifizierung* bezeichnet. Auf die formale Definition wird an dieser Stelle verzichtet. Intuitiv bezieht sich die erweiterte Reduktquantifizierung auf alle in der relativen Zukunft auftauchenden Objekte, während die einfache Reduktquantifizierung nur über die gerade aktuellen Objekte variiert. Die beiden Reduktquantifizierungen entsprechen den Quantifizierungen über aktuelle und mögliche Objekte, wie sie in [LEG85] vorgeschlagen wurden.

Satz 3.4.2 Die temporalen Rekursionsregeln für die finit-lineare temporale Logik lauten

$$\mathbf{always}\,\varphi \iff \varphi \wedge \mathbf{allnext}(\mathbf{always}\,\varphi) \tag{7}$$
$$\mathbf{sometime}\,\varphi \iff \varphi \vee \mathbf{existsnext}(\mathbf{sometime}\,\varphi) \tag{8}$$
$$\varphi\,\mathbf{until}\,\psi \iff \psi \vee (\varphi \wedge \mathbf{allnext}(\varphi\,\mathbf{until}\,\psi)) \tag{9}$$
$$\varphi\,\mathbf{before}\,\psi \iff (\neg\psi \wedge \varphi) \vee (\neg\psi \wedge \mathbf{existsnext}(\varphi\,\mathbf{before}\,\psi)) \tag{10}$$

Ferner gelten u.a. folgende Beziehungen zwischen den Nextoperatoren und den aussagenlogischen Konnektiven $\wedge$ und $\vee$:

$$\neg\,\mathbf{allnext}\,\varphi \iff \mathbf{existsnext}\,\neg\varphi \tag{11}$$
$$\mathbf{allnext}(\varphi \wedge \psi) \iff \mathbf{allnext}\,\varphi \wedge \mathbf{allnext}\,\psi \tag{12}$$
$$\mathbf{existsnext}(\varphi \vee \psi) \iff \mathbf{existsnext}\,\varphi \vee \mathbf{existsnext}\,\psi \tag{13}$$
$$\mathbf{allnext}(\varphi \vee \psi) \iff \mathbf{allnext}\,\varphi \vee \mathbf{allnext}\,\psi \tag{14}$$
$$\mathbf{existsnext}(\varphi \wedge \psi) \iff \mathbf{existsnext}\,\varphi \wedge \mathbf{existsnext}\,\psi \tag{15}$$
$$\dots$$

Beweis : Die temporalen Rekursionsregeln und die Dualität der Nextoperatoren folgen direkt aus den entsprechenden Definitionen, ebenso die Vertauschbarkeit der Nextoperatoren mit den Konnektiven $\wedge$ und $\vee$. Interessant sind die letzten beiden Zeilen, da sie nur für *lineare* temporale Logiken gelten. Die Nextoperatoren sind durch implizite All– bzw. Existenzquantifizierung über die Menge der Folgezustände des aktuellen Zustands definiert und der $\vee$–Operator z.B. ist außer bei einelementigen oder leeren Quantifizierungsbereichen nicht mit der Allquantifizierung in der Reihenfolge vertauschbar. $\square$

In den folgenden Kapiteln wird die finit-lineare temporale Logik als semantische Grundlage für die Spezifikation in temporaler Logik verwendet. Diese Entscheidung wird motiviert durch folgende Punkte :

- Für unendliche Zustandsfolgen entspricht die formale Semantik der finit-linearen temporalen Logik der klassischen Definition der Manna–Pnueli–Logik.
- Reduktquantifizierung ist möglich, und die auf ihr basierende Gültigkeit von Formeln mit freien Variablen führt zu einer intuitiv einleuchtenden Semantik (siehe Kapitel 6, Integritätsüberwachung).
- Bei der Spezifikation dynamischer Integrität sollte soweit wie möglich auf die Benutzung des **next** verzichtet werden, um ein abstraktes Transaktionskonzept zu ermöglichen (siehe Kapitel 5, Spezifikationsmethodik). Der Ersteller einer Spezifikation wird nicht mit der Problematik verschiedener Nextoperatoren belastet.

Die finit-lineare temporale Logik kann genau wie die klassische Manna–Pnueli–Logik um weitere temporale Operatoren erweitert werden, um eine bequemere Formulierung zeitlicher Forderungen zu ermöglichen. Eine Auswahl für Informatikanwendungen geeigneter temporaler Operatoren kann z.B. in [Krö87, Li87] gefunden werden.

Die Semantik der neuen Operatoren kann durch Äquivalenzen zu Formeln der finit-linearen temporalen Logik (vergl. [Krö87, Li87]) oder durch Angabe temporaler Rekursionsregeln [Wo81, CCF82] festgelegt werden. Dabei muß das gewünschte Verhalten bezüglich des Löschens von Objekten besonders beachtet werden. Beide Semantikfestlegungen erlauben den Einsatz aller in den folgenden Kapiteln beschriebenen Verfahren.

Von den in der Literatur beschriebenen Vorschlägen für temporale Operatoren lassen sich als einzige die Vorschläge aus [SMV83] nicht direkt durch die in dieser Arbeit vorgestellten temporalen Operatoren ausdrücken. Der Grund dafür ist, daß die Auswertung der dort vorgestellten *intervall-logischen* Formeln abhängig ist vom Kontext, in dem die Formel steht. In [Li87] wird ein Transformationsalgorithmus vorgestellt, der Formeln der Intervall-Logik in Formeln der Manna–Pnueli–Logik transformiert.

4 Temporale Logik und Transitionsgraphen

In Kapitel 3 wird die Gültigkeit von temporalen Formeln in einer temporalen Struktur unter Bezugnahme auf *alle* Welten dieser Struktur definiert. Für temporale Strukturen mit einer unendlichen Weltenmenge bietet diese Definition keinen Ansatz für ein effektives Verfahren, das die Gültigkeit einer Formel für eine vorgegebene temporale Struktur bestimmt.

In den folgenden Abschnitten werden Verfahren zur effizienten Gültigkeitsüberprüfung temporaler Formeln vorgestellt. Ausgehend von einem Algorithmus, der die Gültigkeit einer Formel unter Verwendung der temporalen Rekursionsregeln überprüft (Abschnitt 4.1), werden Prüfschemata für temporale Formeln konstruiert und deren formale Semantik untersucht (Abschnitte 4.2, 4.3 und 4.4). In den folgenden Abschnitten 4.5 und 4.6 werden die Eigenschaften dieser Prüfschemata untersucht und einige nützliche Verfahren für den Umgang mit ihnen vorgestellt.

4.1 Überprüfung der Gültigkeit temporaler Formeln

Enthält eine temporale Formel keine globalen Quantifizierungen oder Reduktquantifizierungen — dies gilt z.B. automatisch bei Beschränkung auf eine temporale Aussagenlogik — kann die Formel unter Ausnutzung der temporalen Rekursionsregeln in einen "aktuellen" und einen durch Nextoperatoren gebundenen "zukünftigen" Formelteil aufgespalten werden. Für eine lineare temporale Logik kann die Gültigkeit einer Formel φ in einer nichtabbrechenden temporalen Struktur nach folgendem Algorithmus bestimmt werden.

Algorithmus 4.1.1 *(Modellchecker)* Gegeben sei eine temporale Formel φ, eine Zustandsfolge $\hat{\sigma} = \langle \sigma_0, \ldots \rangle$ passender Signatur und eine Belegung der freien Variablen in φ durch Objekte in $\hat{\sigma}$.

(1.) Enthält φ einen temporalen Operator **always**, **sometime**, **until** oder **before**, der nicht durch ein Vorkommen eines Nextoperators gebunden ist, löse ihn in φ unter Verwendung der passenden temporalen Rekursionsregel auf.

(2.) Führe Schritt (1.) aus, bis φ keinen derartigen temporalen Operator mehr enthält.

(3.) Werte alle nicht durch einen Nextoperator gebundenen atomaren Formeln und quantifizierten Formeln im aktuellen Zustand aus, und ersetze sie in φ durch den erhaltenen Wahrheitswert **true** oder **false**.

(4.) Vereinfache φ durch boole'sche Umformungen außerhalb der temporalen Quantoren, so weit wie möglich.

(5.) Ist $\varphi =$ **true**, ist die ursprüngliche Formel erfüllt und der Algorithmus bricht ab.

(6.) Ist $\varphi =$ **false**, ist die ursprüngliche Formel nicht erfüllt und der Algorithmus bricht ab.

(7.) Streiche alle äußersten Nextoperatoren (die also nicht durch einen anderen Nextoperator gebunden sind) in φ unter Berücksichtigung der logischen Struktur der Formel (gegebenenfalls müssen Klammerungen ergänzt werden), setze $\hat{\sigma} = \langle \sigma_1, \ldots \rangle$ und fahre bei Schritt (1.) fort.

Der Algorithmus terminiert natürlich nur, falls die Umformungen in Schritt (4.) irgendwann die Formel zu **true** oder **false** auswerten. □

Dieser Algorithmus kann für alle in Kapitel 3 vorgestellten temporalen Logiken eingesetzt werden. Die Erweiterung auf verzweigte temporale Logiken ergibt sich direkt durch Abarbeitung aller Verzweigungen.

Für die finit-lineare temporale Logik, d.h. für abbrechende lineare temporale Strukturen, muß der Algorithmus um den Schritt (3.1.) erweitert werden, der den Sonderfall eines Zustands ohne Nachfolger berücksichtigt.

(3.1.) Hat der gerade bearbeitete aktuelle Zustand keinen Nachfolger, so ersetze jede Teilformel **allnext** ψ durch **true** und jede Teilformel **existsnext** ψ durch **false**.

Für unendliche Zustandsfolgen braucht der Algorithmus 4.1.1 nicht zu terminieren. Falls er terminiert, ist die Gültigkeit bzw. Nichterfülltheit der untersuchten Formel gezeigt, im anderen Fall ist oft keine Aussage möglich. Die Formel **always** A z.B. kann nur widerlegt, nicht bewiesen werden, da jede sie erfüllende unendliche Zustandsfolge zu einer nichtterminierenden Berechnungsfolge führt. Auf die Behandlung derartiger Berechnungsfolgen und die aus ihr folgenden Notationen von *vorläufiger Gültigkeit* (d.h. "bis jetzt nicht widerlegt") wird in den folgenden Abschnitten noch genauer eingegangen.

Varianten dieses Algorithmus bilden die Grundidee aller sogenannter "Modellchecker", also der Algorithmen, die ein gegebenes Modell (eine temporale Struktur oder ein Muster für eine temporale Struktur) gegen eine temporale Formel verifizieren. Beispiele für rein aussagenlogische Anwendungen sind [Wo82, MW84] für lineare temporale

Logiken und [ES85, CES86] für verzweigte temporale Logiken. [CF84, Ve85] erweitern diese aussagenlogischen Methoden für Manna–Pnueli–Logik auf eine temporale Erweiterung der dem Resolutionsverfahren zugrundeliegenden Klausellogik. Die in [Saa85, LSE86, LS87, SL87] vorgestellte Theorie der Transitionsgraphen verwirklicht einen derartigen Algorithmus für die klassische Manna–Pnueli–Logik mit obigen Einschränkungen für Variablenquantifizierungen. Dieser Ansatz wird in den folgenden Abschnitten auf die finit-lineare temporale Logik übertragen.

Der Algorithmus 4.1.1 berechnet für jeden Übergang zu einer Nachfolgewelt die aktuell zu überprüfende Formel neu. Die Berechnungen bestehen zu einem großen Teil aus aufwendigen Formeltransformationen, die oft für mehrere Übergänge unter gleichen Bedingungen wiederholt werden müssen.

Die durch Anwendung der temporalen Rekursionsregeln konstruierten Formeln bestehen nur aus aussagenlogisch verknüpften Teilformeln der Ursprungsformeln, wie leicht anhand des Formelaufbaus und der Rekursionsregeln gezeigt werden kann. Insbesondere ist jede durch einen temporalen Quantor gebundene Teilformel in einer der berechneten aktuellen Formeln auch Teilformel der Ursprungsformel. Es gibt nur endlich viele paarweise nicht äquivalente Formeln dieser Art (dies kann zum Beispiel durch Transformation in eine Normalform gezeigt werden, siehe Abschnitt 4.3).

Diese Eigenschaft der berechneten aktuellen Formeln bildet die Grundlage für die in diesem Kapitel beschriebene Konstruktion von Transitionsgraphen aus temporalen Formeln. Die zur Berechnung der jeweiligen aktuellen Formel nötigen Umformungen werden für alle möglicherweise als Zwischenergebnis vorkommenden Formeln *unabhängig von einer konkreten Zustandsfolge* berechnet. Das Ergebnis ist ein *Prüfschema* für eine temporale Formel φ. Dieses Prüfschema kann zur Überprüfung beliebiger Zustandsfolgen bezüglich φ benutzt werden, ohne daß in jedem Schritt eine aufwendige Formelumformung nötig ist.

Transitionsgraphen sind eine Darstellung derartiger Prüfschemata durch gerichtete Graphen. Die Konstruktion von Transitionsgraphen, ihre formale Semantik und weitere Eigenschaften werden in den folgenden Abschnitten genauer untersucht. Für die Manna–Pnueli–Logik über unendlichen Zustandsfolgen wurden Transitionsgraphen als *Modellgraphen* für Entscheidungsverfahren auf Erfüllbarkeit in verteilten Prozessen eingeführt [MW81, MW84]. Ihr Einsatz im Datenbankbereich wurde erstmals in [Saa85] vorgeschlagen und in [SL87, LS87] näher untersucht.

Der Schwerpunkt dieser Arbeit liegt auf der Erweiterung der temporalen Logik auf endliche Zustandsfolgen. Sätze und Algorithmen für den neu auftretenden Spezialfall endlicher Folgen werden eingehend untersucht und bewiesen, bei bekannten Eigenschaften von Transitionsgraphen für unendliche Zustandsfolgen wird neben einer kurzen Erläuterung nur auf die Literaturstellen verwiesen.

4.2 Transitionsgraphen

Transitionsgraphen sind eine Notation für aus temporalen Formeln konstruierte Prüfschemata. Die Knoten des Graphen entsprechen Äquivalenzklassen von zu überprüfenden Formeln. Einer *Anfangsmarkierung* von Knoten entspricht eine Disjunktion von Formeln, die zu der zu prüfenden Formel äquivalent ist. Diese Anfangsmarkierung besteht abhängig von der Konstruktionsmethode aus einem oder mehreren Knoten. Mehrere Knoten entsprechen dem alternativen (bzw. nichtdeterministischen) Auswerten der Einzelformeln.

Der Modellchecker–Algorithmus 4.1.1 bestimmt die Gültigkeit der zum aktuellen Zeitpunkt zu überprüfenden Formel durch Auswertung der aktuellen, nichttemporalen Bestandteile und errechnet dabei gleichzeitig die neue zu überprüfende Formel für den nächsten Zeitpunkt. Diese Übergänge zwischen Formeln werden im Transitionsgraphen durch mit nichttemporalen Formeln beschriftete gerichtete Kanten dargestellt. Wie bei der Anfangsmarkierung können auch diese Übergänge zu mehreren alternativen, neu zu überprüfenden Formeln führen (und somit zu einer nichtdeterministischen Abarbeitung des Prüfschemas).

Für die finit-lineare temporale Logik werden die Knoten (bzw. die entsprechenden Formeln) bestimmt und gekennzeichnet, die analog zu Schritt (3.1.) im Algorithmus 4.1.1 eine korrekte Auswertung der Formeln im letzten Zustand einer Zustandsfolge ermöglichen. Dieser Schritt ermöglicht die Behandlung von abbrechenden Zustandsfolgen gemäß der Semantik der finit-linearen temporalen Logik und ist eine echte Erweiterung der aus der Literatur bekannten Transitionsgraphen für die Manna–Pnueli–Logik [MW84, Saa85, LS87].

Definition 4.2.1 Ein *Transitionsgraph* $T = (G, \nu, \eta, m_0, F)$ über einer Signatur Σ und einer Variablenmenge X besteht aus

1. einem gerichteten Graphen $G = \langle N, E \rangle$ mit Knoten N und Kanten E,
2. einer Knotenbeschriftung $\nu \colon N \to \mathit{TF}_\Sigma(X)$, die den Knoten temporale Formeln zuordnet,
3. einer Kantenbeschriftung $\eta \colon E \to P_\Sigma(X)$, die den Kanten prädikatenlogische Formeln zuordnet,
4. einer nichtleeren Anfangsmarkierung $m_0 \subseteq N$
5. und einer Menge von finalen Knoten $F \subseteq N$. $\square$

Die Überprüfung einer temporalen Formel mit einem aus ihr konstruierten Transitionsgraphen als Prüfschema erfolgt nach folgendem nichtdeterministischen Algorithmus. Dieser Algorithmus kann leicht deterministisch formuliert werden, indem statt einzelner Knoten Mengen von Knoten betrachtet werden. Die Eigenschaft eines Transitionsgraphen, Prüfschema einer temporalen Formel zu sein, wird erst später formal definiert.

Algorithmus 4.2.2 Gegeben seien ein Transitionsgraph T, der ein Prüfschema für die zu überprüfende Formel φ ist, eine Zustandsfolge $\hat{\sigma} = \langle \sigma_0, \ldots \rangle$ passender Signatur und eine Belegung der freien Variablen in den Knoten– und Kantenbeschriftungen durch Objekte in $\hat{\sigma}$.

(1.) Wähle einen Knoten aus der Anfangsmarkierung als aktuellen Knoten.

(2.) Bestimme die Gültigkeit der (nichttemporalen !) Beschriftungen der vom aktuellen Knoten ausgehenden Kanten im aktuellem Zustand.

(3.) Ist keine dieser Kantenbeschriftungen gültig, so bricht der Algorithmus ab.

(4.) Wähle eine Kante mit gültiger Beschriftung aus; deren Zielknoten wird aktueller Knoten.

(5.) Ist der aktuelle Knoten mit **false** beschriftet, so bricht der Algorithmus ab.

(6.) Ist der aktuelle Knoten mit **true** beschriftet, so ist φ erfüllt, und der Algorithmus bricht ab.

(7.) Hat der gerade bearbeitete Zustand keinen Nachfolger, so ist φ erfüllt, falls der aktuelle Knoten ein finaler Knoten aus F ist. Der Algorithmus bricht ab.

(8.) Setze $\hat{\sigma} := \hat{\sigma}_1$. Fahre bei (2.) fort.

Eine Formel ist als gültig bewiesen, falls der Algorithmus in Schritt (6.) oder (7.) erfolgreich abbricht. Eine Formel ist als ungültig bewiesen, falls *alle* alternativen Ausführungen erfolglos abbrechen (Schritt (3.), (5.) oder (7.)). □

Dieser Algorithmus beschreibt das Arbeiten mit als Transitionsgraphen vorliegenden Prüfschemata. Eine Formalisierung des Algorithmus und der Beweis seiner Korrektheit folgt im Kapitel 4.4. Die Vorgehensweise des Algorithmus wird am folgenden Beispiel deutlich.

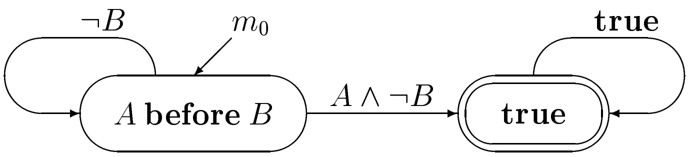


Abbildung 4: Transitionsgraph für die Formel A **before** B

Beispiel 4.2.3 Abbildung 4 auf Seite 45 zeigt die graphische Darstellung eines (von mehreren möglichen !) Transitionsgraphen für die temporale Formel A **before** B (A und B seien atomare Formeln einer Signatur Σ).

Der Transitionsgraph gemäß Abbildung 4 ist definiert durch

$$
\begin{aligned}
G &= \langle N = \{0,1\}, E = \{(0,0),(0,1),(1,1)\}\rangle \\
\nu &= \{\nu(0) = A\,\mathbf{before}\,B,\ \ \nu(1) = \mathbf{true}\} \\
\eta &= \{\eta((0,0)) = \neg B, \eta((0,1)) = A \wedge \neg B, \eta((1,1)) = \mathbf{true}\} \\
m_0 &= \{0\} \\
F &= \{1\}
\end{aligned}
$$

In der graphischen Darstellung werden finale Knoten durch doppelte Umrandung und die Knoten der (in der Regel einelementigen) Anfangsmarkierung werden mit m_0 markiert.

Der Transitionsgraph ist ein Prüfschema für die Formel A **before**B , d.h. er überprüft, ob die atomare Formel A an einem Zeitpunkt vor dem ersten Erfülltsein von B gültig ist. Dies entspricht einer Berechnungsfolge im Algorithmus 4.2.2, die mit Schritt (6.) im mit **true** beschrifteten Knoten 1 terminiert. Bei dieser Berechnungsfolge gilt in jedem abgearbeiteten Zustand $\neg B$ aufgrund der betrachteten Kantenbeschriftungen.

Die Formel A**before**B ist genau dann nicht gültig, falls entweder B in einem Zustand gültig ist, der zeitlich vor dem ersten Zustand liegt, in dem A gültig ist, oder falls die Zustandsfolge abbricht, bevor A das erste Mal gültig wurde. Im ersten Fall bleibt jede Berechnungsfolge zwangsläufig stationär im nichtfinalen Knoten 0 und führt zu einer Terminierung in Schritt (3.) aufgrund fehlender gültiger Kantenbeschriftungen. Im zweiten Fall führen alle Berechnungsfolgen ebenfalls auf den aktuellen Knoten 0 und zu einer korrekten Feststellung der Nichterfülltheit in Schritt (7.). □

Im Beispiel wurde anhand einer Analyse des Graphen gezeigt, daß dieser Graph ein korrektes Prüfschema für die Formel A **before** B ist. Der Zusammenhang zwischen

Transitionsgraphen und temporalen Formeln wird in den folgenden Definitionen deutlich.

Definition 4.2.4 Ein Transitionsgraph heißt *konsistent beschriftet*, wenn für alle Knoten $k \in N$ gilt :

$$\nu(k) \iff \left(\bigvee_{\substack{l \in F \\ (k,l) \in E}} (\eta((k,l)) \wedge \textbf{allnext}\, \nu(l)) \right) \vee \left(\bigvee_{\substack{l \in (N-F) \\ (k,l) \in E}} (\eta((k,l)) \wedge \textbf{existsnext}\, \nu(l)) \right)$$

$\square$

Diese Definition garantiert korrekte Übergänge entlang der Kanten zu neuen aktuellen Knotenbeschriftungen und entspricht somit der beabsichtigten Verwendung der Transitionsgraphen als Prüfschemata für temporale Formeln.

Definition 4.2.5 Ein Transitionsgraph T heißt *Prüfschema* für eine temporale Formel φ, wenn gilt :

1. T ist konsistent beschriftet.
2. φ ist äquivalent zur Disjunktion der Beschriftungen von Knoten in m_0. $\square$

Gemäß dieser Begriffsbildung müßte gezeigt werden, daß der Algorithmus 4.2.2 für T unter diesen Bedingungen dieselben Zustandsfolgen zu **true** bzw. **false** auswertet wie der Modellchecker–Algorithmus 4.1.1. Da beide Algorithmen informal festgelegt sind, ist höchstens eine Plausibilitätbetrachtung möglich. Diese kann — da beide Algorithmen im ersten Schritt von derselben Formel ausgehen — durch vollständige Induktion über Anfangsstücke der Zustandsfolge geschehen. Im Abschnitt 4.4 wird die Akzeptanz von Zustandsfolgen durch Transitionsgraphen formalisiert und die Korrektheit der Formelüberwachung durch Transitionsgraphen gezeigt.

In den folgenden Abschnitten werden Verfahren vorgestellt, die aus vorgegebenen temporalen Formeln Transitionsgraphen konstruieren. Die Konstruktion ist immer möglich, falls eine Überprüfung der Gültigkeit mit dem allgemeinen Modellchecker 4.1.1 möglich ist, also für temporale Formeln ohne globale oder Reduktquantifizierung.

Die Korrektheit der auf Transitionsgraphen beruhenden Beweise von Formeln in Zustandsfolgen wird gezeigt, und Notationen für die semantische Behandlung von "laufenden" Beweisen werden eingeführt, die von besonderer Wichtigkeit für die Überprüfung von temporalen Formeln in unvollständigen Zustandsfolgen sind. Derartige unvollständige Zustandsfolgen entsprechen einer Datenbankentwicklung bis zu einem aktuellen Zustand mit einer noch offenen Zukunft.

4.3 Konstruktion von Transitionsgraphen

Der allgemeine Modellchecker–Algorithmus 4.1.1 ermöglicht die Überprüfung von beliebigen temporalen Formeln ohne globale Quantifizierungen. Für diese Klasse von temporalen Formeln wird nun ein Konstruktionsverfahren für Transitionsgraphen vorgestellt. Als Vorbereitung werden eine *Normalform* für temporale Formeln definiert, ein Algorithmus zur Umformung in diese Normalform skizziert und Eigenschaften dieser Normalform untersucht.

4.3.1 Disjunktive Normalform

Das Grundprinzip des allgemeinen Modellchecker–Algorithmus ist die Anwendung der Rekursionsregeln, um den aktuellen Anteil der Formel von dem zukünftigen, echt temporalen Anteil abzutrennen. Durch sukzessive Anwendung der Rekursionsregeln kann jede Formel in eine äquivalente Formel umgeformt werden, in der alle temporalen Operatoren durch einen Nextoperator gebunden sind. Diese Formeln sind somit Aussagen über die Restzustandsfolge $\hat{\sigma}_1$ bzw. im Falle von **existsnext** zusätzlich über deren Existenz. Durch weitere Umformung gemäß den Gesetzen der Aussagenlogik können Formeln in eine Normalform gebracht werden, die Ausgangspunkt zur Konstruktion von Transitionsgraphen ist.

Definition 4.3.1 Prädikatenlogische *Grundformeln* einer temporalen Formel $\varphi \in TF_{\Sigma}(X)$ sind alle atomaren Formeln und durch Quantoren ($\exists$ oder $\forall$) gebundenen Teilformeln von φ, für die gilt :

1. Sie sind Teilformeln von φ.
2. Sie haben mindestens ein Vorkommen außerhalb einer Variablenquantifizierung, d.h. sie treten nicht nur als echte Teilformeln von Formeln $\exists(x)\psi$ bzw. $\forall(x)\psi$ auf.

Aufgrund der Syntaxdefinition von temporalen Formeln aus $TF_{\Sigma}(X)$ sind Grundformeln nichttemporal. □

Definition 4.3.2 Eine Formel $\varphi \in TF_\Sigma(X)$ ohne globale Quantifizierungen ist in *disjunktiver Normalform*, kurz $\varphi \in DNF_\Sigma(X)$, wenn sie folgende Form hat :

$$\bigvee_k \left(\zeta_k \wedge \textbf{allnext}\, \gamma_k \; [\wedge\, \textbf{existsnext}\, \delta_k]\right)$$

wobei

- jedes ζ_k eine Konjunktion von Grundformeln von φ oder deren Negationen ist,
- jedes γ_k eine Konjunktion von Formeln γ_{k_j} oder $\neg\gamma_{k_j}$ ist, wobei jedes γ_{k_j} entweder eine Grundformel von φ oder durch einen temporalen Operator gebunden ist,
- und jedes δ_k eine Konjunktion von Formeln δ_{k_j} oder $\neg\delta_{k_j}$ ist, wobei jedes δ_{k_j} entweder eine Grundformel von φ oder durch einen temporalen Operator gebunden ist.

Die eckige Klammer drückt die Optionalität des durch **existsnext** gebundenen Formelteils für gewisse k an. ζ_k oder γ_k äquivalent zu **true** können für einzelne k in der Notation weggelassen werden, um die Lesbarkeit zu erhöhen. Dies gilt nicht für die δ_k, da **existsnext true** nicht äquivalent zu **true** ist (für abbrechende Zustandsfolgen). $\square$

Die Umformung in disjunktive Normalform erfolgt durch fortgesetzte zielgerichtete Anwendung der temporalen Rekursionsregeln und von Formeltransformationen gemäß den bekannten Regeln der Aussagenlogik. Detaillierte Algorithmen zur Umformung in eine entsprechende Normalform für die klassische Manna–Pnueli–Logik sind in [MW84, Saa85, Schu87] und [LSE86, LS87] beschrieben. Diese Algorithmen können leicht für die finit-lineare temporale Logik modifiziert werden.

Satz 4.3.3 Jede Formel $\varphi \in TF_\Sigma(X)$ ohne globale Quantifizierungen läßt sich durch Anwendung der temporalen Rekursionsregeln 3.4.2 von Seite 38, der Dualitäts– und Verträglichkeitsregeln der Nextoperatoren (ebenfalls 3.4.2) und aussagenlogischer Transformationen in eine äquivalente Formel in disjunktiver Normalform $DNF(\varphi)$ umwandeln. Die Anzahl k der nicht äquivalenten Zeilen von $DNF(\varphi)$ nach obiger Definition ist endlich.

Beweisskizze : Da diese Arbeit Grundkenntnisse der formalen Logik voraussetzt, wird auf eine ausführliche Angabe des Transformationsalgorithmus verzichtet. Stattdessen werden die interessanten Zwischenschritte näher betrachtet und die Endlichkeit der konstruierten Normalform gezeigt.

Zuerst werden alle nicht durch einen Nextoperator gebundenen temporalen Quantoren durch Anwendung der Rekursionsregeln aufgelöst. Bei geschachtelten temporalen Quantoren muß dieser Schritt entsprechend oft wiederholt werden. Die folgende Umformung in eine disjunktive Form erfordert nur bekannte aussagenlogische Transformationen und Verschmelzungen unter Berücksichtigung der Sonderregeln für die Nextoperatoren.

Jede Anwendung der Rekursionsregeln liefert eine Formel, die aus durch aussagenlogische Konnektive und Nextoperatoren verbundenen Teilformeln der Ursprungsformel besteht. Darum können als γ_{k_j} bzw. δ_{k_j} neben Grundformeln nur durch einen temporalen Quantor gebundene Teilformeln der ursprünglichen Formel φ auftreten. Die Menge der Grundformeln wird durch die angegebenen Manipulationen ebenfalls nicht vergrößert.

Nach Verschmelzung identischer Bestandteile und aussagenlogischer Vereinfachung können prinzipiell nur endlich viele verschiedene Zeilen einer Normalform gebildet werden. □

Beispiel 4.3.4 Für Grundformeln φ und ψ ergeben die Transformationen gemäß der Rekursionsregeln 3.4.2 auf Seite 38 direkt Formeln in disjunktiver Normalform.

Um die disjunktive Normalform der Formel **always**$(A \Rightarrow$ **sometime** $B)$ zu berechnen, muß in einem ersten Schritt der **always**–Operator gemäß der Rekursionsregel aufgelöst und die Implikation durch eine Disjunktion ausgedrückt werden. Dies führt als Zwischenergebnis zu der Formel

$$(\neg A \vee \textbf{sometime}\, B) \wedge \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B))$$

Die Auflösung von **sometime** B und anschließende aussagenlogische Transformationen überführen die Formel in disjunktive Normalform. Die drei Zeilen der disjunktiven Normalform für **always**$(A \Rightarrow$ **sometime** $B)$ lauten

$$\begin{array}{lll} & \neg A & \wedge\, \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B)) \\ \vee & B & \wedge\, \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B)) \\ \vee & & \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B)) \\ & & \wedge\, \textbf{existsnext}(\textbf{sometime}\, B) \end{array}$$

□

Satz 4.3.5 Ist φ eine Formel der temporalen Prädikatenlogik mit einer Darstellung in *DNF* gemäß Definition 4.3.2, so sind folgende Aussagen für eine gegebene Zustandsfolge $\hat{\sigma}$ mit einer Länge > 1 äquivalent (die δ_k–Bestandteile müssen natürlich nur bei ihrer Existenz berücksichtigt werden) :

1.
$$(\hat{\sigma}, \hat{\beta}) \models \varphi$$

2. Es existiert ein k, so daß $(\sigma_0, \beta) \models \zeta_k$ und $(\hat{\sigma}_1, \hat{\beta}) \models \gamma_k \wedge \delta_k$.

3.
$$(\hat{\sigma}_1, \hat{\beta}) \models \bigvee_{\substack{k \\ (\sigma_0,\beta)\models\zeta_k}} \gamma_k \wedge \delta_k$$

Die lokale Belegung β_{σ_0} in σ_0 wurde mit β abgekürzt.

Beweis: Da ein Nachfolgezustand laut Voraussetzung existiert, haben **existsnext** und **allnext** dieselbe Semantik. Der Formelteil **allnext** $\gamma_k \wedge$ **existsnext** δ_k kann deshalb entsprechend der klassischen Manna–Pnueli–Logik als **next**$(\gamma_k \wedge \delta_k)$ notiert werden. Fehlende Bestandteile müssen dabei durch **true** ergänzt werden.

Die Äquivalenz der drei Aussagen folgt nun direkt aus der Semantikdefinition der temporalen Prädikatenlogik : Die Formel φ hat die Form einer Disjunktion von Formeln $\zeta_k \wedge$ **next**$(\gamma_k \wedge \delta_k)$. Aus der Definition des $\wedge$-Operators und der aus der Definition des $\vee$-Operators folgenden Tatsache, daß in einer Disjunktion nicht gültige Teile gekürzt werden dürfen, folgt :

$$(\hat{\sigma}, \hat{\beta}) \models \bigvee_{\substack{k \\ (\sigma_0,\beta)\models\zeta_k}} \mathbf{next}(\gamma_k \wedge \delta_k)$$

Dies entspricht gemäß der Definition der Nextoperatoren der dritten Behauptung. Die Disjunktion über den Index k wiederum kann unter Benutzung sukzessiver Anwendung der Definition von $\vee$ in die Existenzforderung der zweiten Behauptung umgeformt werden. □

Satz 4.3.6 Eine Formel in disjunktiver Normalform ist in einer Zustandsfolge $\hat{\sigma} = \langle\sigma_0\rangle$ der Länge 1 genau dann gültig, falls eine Zeile k vorkommt, so daß $(\sigma_0, \beta_{\sigma_0}) \models \zeta_k$ und ein Bestandteil **existsnext** δ_k in der Zeile nicht existiert.

Beweis : Der Beweis erfolgt analog zum obigen Satz direkt aus der Definition von **allnext** und **existsnext** für Zustände ohne Nachfolgezustände. Der Formelteil **allnext** γ_k wird dabei zu **true** ausgewertet und der Formelteil **existsnext** δ_k zu **false**, falls er existiert. □

4.3.2 Konstruktionsalgorithmus

Nachstehend ist ein auf die Konstruktion der disjunktiven Normalform basierender Konstruktionsalgorithmus für Prüfschemata angegeben. Der Konstruktionsalgorithmus 4.3.7 für Transitionsgraphen aus beliebigen temporalen Formeln ohne globale Quantifizierungen ist auf Seite 51 angegeben. Der Transitionsgraph wird in den

Algorithmus 4.3.7 Gegeben sei eine Formel $\varphi \epsilon TF_\Sigma(X)$ ohne globale Quantifizierungen. Der zugehörige Transitionsgraph T_φ wird wie folgt aus φ konstruiert.

(1) Starte mit dem leeren Graphen G als T_φ;
(2) Füge einen Knoten 0 mit $\nu(0) = \varphi$ in T_φ ein;
    Setze $m_0 = \{0\}$;
    Setze $F = \{\}$;
(3) **for each** Knoten $n \epsilon N$ in T_φ
    do Transformiere $\nu(n)$ in $DNF_\Sigma(X)$;
        for each $(\zeta \wedge$ **allnext** $\gamma)$ in $DNF(\nu(n))$
        do if Für kein $m \epsilon F$ gilt $\nu(m) = \gamma$
                then Füge neuen Knoten m mit $\nu(m) = \gamma$ in K ein;
                    $F := F \cup \{m\}$;
            endif;
            Sei m der Knoten in F mit $\nu(m) = \gamma$;
            Füge Kante (n, m) mit $\eta((n, m)) = \zeta$ in E ein;
        enddo;
        for each $(\zeta \wedge$ **allnext** $\gamma \wedge$ **existsnext** $\delta)$ in $DNF(\nu(n))$
        do if Für kein $m \epsilon N - F$ gilt $\nu(m) = \gamma \wedge \delta$
                then Füge neuen Knoten m mit $\nu(m) = \gamma \wedge \delta$ in $N - F$ ein;
            endif;
            Sei m der Knoten in $N - F$ mit $\nu(m) = \gamma \wedge \delta$;
            Füge Kante (n, m) mit $\eta((n, m)) = \zeta$ in E ein;
        enddo;
        if Mehrere Kanten existieren von n zu einem Knoten m
            then Ersetze sie durch eine einzige Kante mit der Disjunktion der
                alten Beschriftungen als neue Beschriftung;
        endif;
    enddo.

Abbildung 5: Konstruktionsalgorithmus für Transitionsgraphen

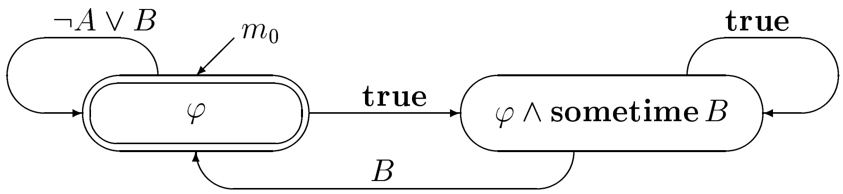


Abbildung 6: Transitionsgraph für $\varphi :=$ **always**$(A \Rightarrow$ **sometime** $B)$

Schritten (1) und (2) initialisiert. Er enthält danach genau einen Knoten, der mit der Eingabeformel beschriftet ist, und der die Anfangsmarkierung bildet.[4] Während der Konstruktion wird ein gerichteter Multigraph (mehrere Kanten zwischen zwei Knoten) als Datenstruktur für Zwischenschritte verwandt.

Basierend auf der jeweiligen disjunktiven Normalform der Knotenbeschriftungen werden Kanten zu anderen Knoten eingefügt, die bei Bedarf dem Graphen neu hinzugefügt werden müssen. Entsprechend der Behandlung abbrechender Zustandsfolgen im Modellchecker–Algorithmus werden die finalen Knoten aus den Zeilen der disjunktiven Normalform gebildet, in denen kein **existsnext**–Bestandteil vorkommt. Die Korrektheit des Konstruktionsverfahrens wird im folgenden Kapitel 4.3 gezeigt.

Beispiel 4.3.8 Der Transitionsgraph zur Formel

$$\varphi := \textbf{always}(A \Rightarrow \textbf{sometime}\, B)$$

ist in Bild 6 auf Seite 52 abgebildet. Der Algorithmus 4.3.7 erzeugt in diesem Fall zwei mit φ beschriftete Knoten, einen finalen und einen nicht finalen, um den Spezialfall der leeren Zustandsfolge zu modellieren. Der Übersicht halber wurden die beiden Knoten in diesem Beispiel in einem finalen Knoten zusammengefaßt.

Die Knoten und ihre Beschriftung, ihre Aufteilung auf die Mengen m_0, F und $(N-F)$ und die Beschriftung der vom mit φ beschrifteteten Knoten ausgehenden Kanten folgt direkt aus der in Beispiel 4.3.4 auf Seite 49 angegebenen disjunktiven Normalform. Die Normalformbildung und die Konstruktionsschritte müssen für den zweiten Knoten wiederholt werden. □

Satz 4.3.9 Der Konstruktionsalgorithmus 4.3.7 terminiert.

Beweis : Die Transformation in $DNF_{\Sigma}(X)$ erzeugt gemäß Satz 4.3.3 jeweils nur eine endliche Anzahl von Zeilen. Da bei der Erzeugung neuer Knoten nur Formeln als Beschriftung auftreten können, die aus einer Konjunktion von (eventuell negierten) Unterformeln von φ bestehen (iterative Anwendung der Argumentation von Satz 4.3.3),

[4] Der initiale, mit der Eingabeformel beschriftete Knoten ist nicht final, da die Semantik temporaler Formeln nur für Kripkestrukturen mit einer nichtleeren Weltenmenge definiert wurde.

sind nur endlich viele unterschiedliche Knotenbeschriftungen möglich. Voraussetzung ist dabei die weitestgehende aussagenlogische Vereinfachung der disjunktiven Normalform. □

Satz 4.3.10 Der konstruierte Transitionsgraph T_φ ist ein Prüfschema für die Formel φ.

Beweis : Der Knoten 0 ist mit φ beschriftet. Nach Definition 4.2.5 von Seite 46 muß nur noch gezeigt werden, daß T_φ konsistent beschriftet ist. Knoten– und Kantenbeschriftungen werden aus den Zeilen der disjunktiven Normalform gebildet. Folgende Äquivalenz für temporale Formeln zeigt, daß die Konstruktion über die Normalform zu konsistenten Beschriftungen führt :

$$\textbf{allnext}\, \psi \wedge \textbf{existsnext}\, \varphi \quad \Longleftrightarrow \quad \textbf{existsnext}(\psi \wedge \varphi)$$

In einem Zustand ohne Nachfolger werden beide Seiten dieser Äquivalenz zu **false** ausgewertet. Gibt es einen Nachfolger, muß die mit ihm beginnende Restzustandsfolge in beiden Fällen die Formel $\psi \wedge \varphi$ erfüllen.

Einzelne Zeilen der disjunktiven Normalform erfüllen aufgrund dieser Äquivalenz bereits die Forderungen des für Kanten- und Knotenbeschriftungen formulierten Konsistenzkriterium. Das Verschmelzen von Kanten erhält diese Eigenschaften gemäß der Definition der Disjunktion. Es folgt, daß aufgrund der Benutzung der disjunktiven Normalform im Algorithmus 4.3.7 die konstruierten Graphen konsistent beschriftet sind. □

Für die klassische Manna–Pnueli–Logik ohne globale Quantifizierung sind mehrere alternative Konstruktionsalgorithmen für konsistent beschriftete Transitionsgraphen bekannt. In [LSE86, LS87, SL87] werden die Graphen ebenfalls durch Umformungen in eine disjunktive Normalform konstruiert. Ein äquivalenter Algorithmus [MW84, Saa85, Schu87] benutzt Tableau–Methoden nach [Sm68] für die Konstruktion der Kanten und Knoten.

Eine neuartige Konstruktionsmethode wird in [FL87, Li87] vorgestellt. Dort werden konsistent beschriftete Transitionsgraphen entsprechend der Formelstruktur aus elementaren Graphen, die den atomaren Formeln zugeordnet sind, aufgebaut. Jedem aussagenlogischen Konnektiv und jedem temporalen Operator ist eine Graphenmanipulation zugeordnet, die aus gegebenen konsistent beschrifteten Graphen für die Teilformeln einen konsistent beschrifteten Graphen für die neue komplexere Formel konstruiert.

Im Gegensatz zu den durch Normalformbildung konstruierten sind diese Transitionsgraphen deterministisch, d.h. die Beschriftungen der von einem Knoten ausgehenden

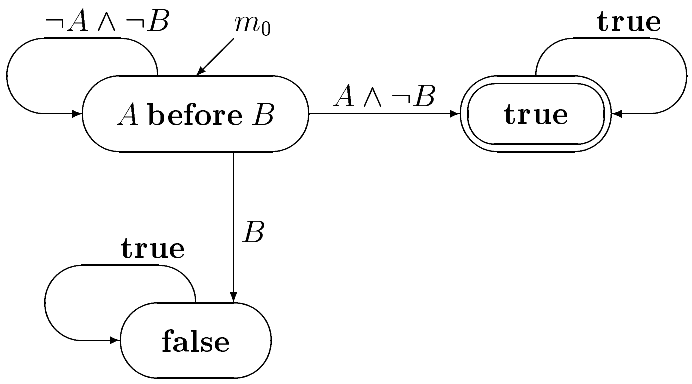


Abbildung 7: Deterministischer Transitionsgraph für A **before** B

Kanten schließen sich gegenseitig aus, und in jedem Zustand ist mindestens eine der Kanten gültig. Um diese Bedingung einzuhalten, muß ein Fehlerknoten, der mit **false** beschriftet ist, eingeführt werden. Die Determiniertheit ist notwendig, um z.B. die dem '$\neg$'–Operator zugeordnete Graphmanipulation zu definieren.

Der mit **false** beschriftete explizite Fehlerknoten gehört in der erweiterten Manna–Pnueli–Logik nicht zu den finalen Knoten, da zwar

$$\textbf{false} \iff \textbf{true} \wedge \textbf{allnext false} \wedge \textbf{existsnext false}$$

gilt, aber nicht

$$\textbf{false} \iff \textbf{true} \wedge \textbf{allnext false}.$$

Der statische Anteil muß in deterministischen Graphen für den Fehlerknoten **true** sein — bei der Konstruktion von Transitionsgraphen durch Normalformen treten derartige Knoten nicht auf.

Deterministische Transitionsgraphen sind auch für die Überprüfung temporaler Formeln von besonderem Interesse, da in ihnen der einer Zustandsfolge zugeordnete Berechnungspfad für die Gültigkeitsüberprüfung mit Prüfschemata eindeutig bestimmt ist.

Beispiel 4.3.11 Ein deterministischer, konsistent beschrifteter Transitionsgraph für die Formel A **before** B ist in Bild 7 abgebildet. Die aus dem ersten Knoten herausführenden Kanten erfüllen die obigen Forderungen — ihre Disjunktion ist äquivalent zu **true** und die Kantenbeschriftungen der drei Kanten schließen sich paarweise aus. $\square$

Im Abschnitt 4.5 wird ein Verfahren vorgestellt, daß die Konstruktion von deterministischen, konsistent beschrifteten Transitionsgraphen aus den durch Algorithmus 4.3.7 konstruierten nichtdeterministischen Graphen ermöglicht.

4.4 Formale Semantik

Die Semantik von Transitionsgraphen als Prüfschemata für temporale Formeln wurde bis jetzt durch den eher informal formulierten Algorithmus 4.2.2 von Seite 44 festgelegt. In diesem Kapitel wird eine formale operationale Semantik für Transitionsgraphen angegeben und die Korrektheit von Formelbeweisen mittels Transitionsgraphen bewiesen.

Der Algorithmus 4.2.2 wählt nichtdeterministisch einen Pfad durch den Graphen aus, dessen Kanten in den entsprechenden Zuständen gültig sind. Ein deterministisch arbeitender Algorithmus muß entweder den Transitionsgraphen in einen deterministischen Graphen umformen (siehe Kapitel 4.5) oder jeweils *alle* möglichen Pfade gleichzeitig verfolgen, bis die Gültigkeit der Formel entschieden werden kann. Da die für die Überprüfung relevante Information über den abgearbeiteten Teil der Zustandsfolge durch die aktuell zu überprüfenden Formeln ausgedrückt wird, genügt es hierbei, nur die Endknoten der korrekten Pfade zu betrachten. Diese Knotenmenge wird als *aktuelle Markierung* bezeichnet.

Um die Lesbarkeit von Definitionen, Sätzen und deren Beweisen zu erhöhen, wird in den Notationen nicht explizit zwischen den einzelnen lokalen Belegungen (β_{σ_i}) einer globalen Belegung $(\hat{\beta})$ unterschieden. Betrachtet werden nur globale Belegungen, die globale Erweiterung einer gemeinsamen Belegung bezüglich des Anfangszustands σ_0 sind. Mit anderen Worten, jede betrachtete globale Belegung induziert für jeden Zustand σ_i dieselbe lokale Belegung β.

Definition 4.4.1 Für einen Transitionsgraphen T zur Signatur Σ ist die *Markierung* $m_T(t, \hat{\sigma}, \hat{\beta})$ zum Zeitpunkt t für eine Zustandsfolge $\hat{\sigma}$ unter einer Belegung $\hat{\beta}$ durch folgende (partielle) Funktion

$$m_T\colon I\!N \times \Im_\Sigma{}^+ \times \mathit{BEL}(X, \Im_\Sigma) \to \mathcal{P}(N)$$

definiert :

$$\begin{array}{rcl} m_T(0, \hat{\sigma}, \hat{\beta}) & = & m_0 \\ m_T(i+1, \hat{\sigma}, \hat{\beta}) & = & \left\{ \begin{array}{ll} \mathrm{trans}(m_T(i, \hat{\sigma}, \hat{\beta}), \sigma_i, \beta) & \text{falls } i \leq |\hat{\sigma}| \\ \perp & \text{sonst} \end{array} \right. \end{array}$$

wobei trans: $\mathcal{P}(N) \times \Im_\Sigma \times BEL(X, \Im_\Sigma) \to \mathcal{P}(N)$ definiert ist durch

$$\begin{aligned}\text{trans}(L, \sigma, \beta) \;\; = \;\; \{k \in N \mid \;\; & (\exists l \in L) \colon (l,k) \in E \wedge (\sigma, \beta) \models \eta((l,k)) \\ & \wedge \;\; \nu(k) \neq \textbf{false}\}\end{aligned}$$

Die Definition von 'trans' wird auch als *Transitionsregel* bezeichnet. □

Die Markierung $m_T(t, \hat{\sigma}, \hat{\beta})$ enthält genau die Endknoten von Pfaden der Länge t durch den Transitionsgraphen T, die in dem Sinne korrekt sind, daß zum Zeitpunkt i die i–te Kante des Pfades unter der Belegung $\hat{\beta}$ gültig ist. Die Markierungen werden iterativ durch Anwendung der Transitionsregel berechnet.

Definition 4.4.2 Für die Überprüfung von Formeln mit Transitionsgraphen sind die (Anfangsstücke von) Zustandsfolgen von Interesse, die zu nicht leeren Markierungen führen. Bei endlichen Zustandsfolgen muß nach der vollständigen Abarbeitung der Folge mindestens ein finaler Knoten in der Markierung enthalten sein.

1. Ein Transitionsgraph T *akzeptiert* ein Anfangsstück der Länge i einer Zustandsfolge $\hat{\sigma}$ *vorläufig* unter der Belegung $\hat{\beta}$ genau dann, wenn die i–te Markierung $m_T(i, \hat{\sigma}, \hat{\beta})$ nicht leer ist.

2. Ein Transitionsgraph T *akzeptiert* eine endliche Zustandsfolge $\hat{\sigma}$ unter der Belegung $\hat{\beta}$, kurz $(\hat{\sigma}, \hat{\beta}) \vdash T$, wenn die letzte Markierung einen finalen Knoten enthält :
$$(\hat{\sigma}, \hat{\beta}) \vdash T \;\text{ gdw. }\; m_T(|\hat{\sigma}| + 1, \hat{\sigma}, \hat{\beta}) \cap F \neq \emptyset$$

3. Ein Transitionsgraph T *akzeptiert* eine (endliche oder unendliche) Zustandsfolge unter der Belegung $\hat{\beta}$ *nicht*, kurz $(\hat{\sigma}, \hat{\beta}) \not\vdash T$, wenn entweder zu einem Zeitpunkt t die aktuelle Markierung leer ist, oder die Zustandsfolge endlich ist und die letzte Markierung keinen finalen Knoten enthält :
$$\begin{aligned}(\hat{\sigma}, \hat{\beta}) \not\vdash T \quad \text{gdw.} \quad & (\exists t \in I\!N) \colon m_T(t, \hat{\sigma}, \hat{\beta}) = \emptyset \\ & \vee \; (|\hat{\sigma}| \in I\!N \wedge \text{ nicht } (\hat{\sigma}, \hat{\beta}) \vdash T)\end{aligned}$$

□

Für unendliche Zustandsfolgen ist nach diesen Definitionen nur die Nichtakzeptanz definiert. Auf die Fälle, in denen die Akzeptanz einer unendlichen Zustandsfolge schon nach einem endlichen Teilstück entschieden werden kann, wird später eingegangen (dieses ist z.B. der Fall, falls irgendwann einer der markierten Knoten mit **true** beschriftet ist).

Die Benutzung des '$\vdash$'–Symbols, das oft die Ableitung der Gültigkeit von Formeln in Deduktionssystemen bezeichnet, für die Akzeptanz von Zustandsfolgen wird durch die folgenden Sätze motiviert, die die Beziehung zwischen der Gültigkeit von Formeln und der Akzeptanz durch aus ihnen konstruierte Transitionsgraphen herstellen.

Für beliebige Zustandsfolgen $\hat{\sigma} = \langle \sigma_0, \ldots \rangle$ gilt folgende Beziehung zwischen der vorläufigen Akzeptanz von Anfangsstücken $\langle \sigma_0, \ldots, \sigma_i \rangle$ von $\hat{\sigma}$ durch Transitionsgraphen und der Gültigkeit von Formeln :

Satz 4.4.3 Sei T_φ ein Prüfschema für die Formel φ. Für beliebige Zustandsfolgen $\hat{\sigma}$ gilt für eine feste Belegung $\hat{\beta}$ und für alle $n \leq |\hat{\sigma}|$:

$$(\hat{\sigma}, \hat{\beta}) \models \varphi \quad \text{gdw.} \quad T_\varphi \text{ akzeptiert } \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \text{ unter } \hat{\beta} \text{ vorläufig} \\ \wedge (\hat{\sigma}_n, \hat{\beta}) \models \bigvee_{k \,\epsilon\, m(n, \hat{\sigma}, \hat{\beta})} \nu(k)$$

Beweis : Der Beweis erfolgt durch vollständige Induktion über die Länge der vom Transitionsgraphen akzeptierten Anfangsstücke von Zustandsfolgen. Die Disjunktion der Beschriftungen der Knoten in der aktuellen Markierung wird im folgenden als *aktuelle Formel* bezeichnet.

Die Anfangsmarkierung m_0 kann nicht leer sein, wenn T_φ ein Prüfschema ist. Der Graph T_φ akzeptiert also die leere Zustandsfolge. Da der Transitionsgraph ein Prüfschema für φ ist, gilt, da $m_0 = m(0, \hat{\sigma}, \hat{\beta})$, außerdem

$$\varphi \iff \bigvee_{k \,\epsilon\, m(0, \hat{\sigma}, \hat{\beta})} \nu(k)$$

Die Zustandsfolge $\langle \sigma_0, \ldots, \sigma_{-1} \rangle$ (für $i = 0$) entspricht der leeren Folge. Der Satz gilt somit für $n = 0$. Nun muß der Induktionsschritt von i auf $i + 1$ gezeigt werden. Es gelte

$$T_\varphi \text{ akzeptiert } \langle \sigma_0, \ldots, \sigma_{i-1} \rangle \text{ unter } \hat{\beta} \text{ vorläufig } \wedge (\hat{\sigma}_i, \hat{\beta}) \models \bigvee_{k \,\epsilon\, m(i, \hat{\sigma}, \hat{\beta})} \nu(k)$$

Laut Voraussetzung ($n \leq |\hat{\sigma}|$) existiert der Zustand σ_i. Aufgrund der Definition 4.2.5 von Seite 46 der Prüfschemaeigenschaften und der Kommutativität der Disjunktion ist die aktuelle Formel äquivalent zu der Disjunktion ($*$)

$$
\begin{aligned}
& \left(\bigvee_{\substack{k \,\epsilon\, m(i,\hat{\sigma},\hat{\beta}) \\ l \epsilon F \wedge (k,l) \epsilon E}} \Big(\eta((k,l)) \wedge \mathbf{allnext}\, \nu(l) \Big) \right) \\
\vee \quad & \left(\bigvee_{\substack{k \,\epsilon\, m(i,\hat{\sigma},\hat{\beta}) \\ l \epsilon (N-F) \wedge (k,l) \epsilon E}} \Big(\eta((k,l)) \wedge \mathbf{existsnext}\, \nu(l)) \Big) \right)
\end{aligned}
\qquad (*)
$$

Die aktuelle Formel ist genau dann erfüllt, wenn erstens mindestens eine der Zeilen der Disjunktion $(*)$ eine in σ_i gültige Kantenbeschriftung enthält und zweitens die zugehörige Knotenbeschriftung in der Restzustandsfolge erfüllt ist. Der erste Teil dieser Bedingung erzwingt eine nichtleere Markierung $m(i+1, \hat{\sigma}, \hat{\beta})$ und damit die vorläufige Akzeptanz von $\langle \sigma_0, \ldots, \sigma_i \rangle$. Der zweite Teil der Bedingung fordert, daß die neue aktuelle Formel erfüllt sein muß.

Es gilt also

$$
T_\varphi \text{ akzeptiert } \langle \sigma_0, \ldots, \sigma_i \rangle \text{ unter } \hat{\beta} \text{ vorläufig } \;\wedge\; (\hat{\sigma}_{i+1}, \hat{\beta}) \models \bigvee_{k \,\epsilon\, m(i+1,\hat{\sigma},\hat{\beta})} \nu(k)
$$

und aufgrund der Induktion über i ist der Satz bewiesen. □

Für die klassische Manna–Pnueli–Logik kann der Beweis des entsprechenden Satzes in [LS87] gefunden werden. Der Satz 4.4.3 ist von besonderer Bedeutung für die optimistische Überwachung von temporalen Formeln bei noch unbekannter Zukunft (dem typischen Zustand während des Betriebs einer Datenbank), da er ein Kriterium für die *potentielle* Gültigkeit von Formeln in Zustandsfolgen liefert (siehe Kapitel 6).

Satz 4.4.4 Ist T_φ ein Prüfschema für die Formel φ, dann gilt für endliche Zustandsfolgen $\hat{\sigma}$:

$$
(\hat{\sigma}, \hat{\beta}) \vdash T_\varphi \;\text{ gdw. }\; (\hat{\sigma}, \hat{\beta}) \models \varphi
$$

Beweis :

"$\Rightarrow$" Sei $n = |\hat{\sigma}|$. Zuerst wird

$$
((\hat{\sigma}, \hat{\beta}) \vdash T_\varphi) \Rightarrow ((\hat{\sigma}, \hat{\beta}) \models \varphi)
$$

bewiesen. Der Transitionsgraph akzeptiert aufgrund Definition 4.4.2 jedes Anfangsstück von $\hat{\sigma}$ vorläufig (die Transitionsregel überführt die leere Menge wieder in die leere Menge) und die letzte Markierung $m_{T_\varphi}(n+1, \hat{\sigma}, \hat{\beta})$ enthält einen finalen Knoten.

Aufgrund Satz 4.4.3 gilt für die aus einem Zustand bestehende Restzustandsfolge $\hat{\sigma}_n = \langle \sigma_n \rangle$:

$$(\langle \sigma_n \rangle, \hat{\beta}) \models \bigvee_{k \,\epsilon\, m_{T_\varphi}(n, \hat{\sigma}, \hat{\beta})} \nu(k)$$

Da T_φ ein Prüfschema für φ ist, ist diese aktuelle Formel ebenfalls äquivalent zu der Disjunktion $(*)$ (Definition 4.2.5). Für eine Zustandsfolge der Länge 1 vereinfacht sich diese Formel zu

$$\bigvee_{\substack{k \,\epsilon\, m(i, \hat{\sigma}, \hat{\beta}) \\ l \epsilon F \wedge (k,l) \epsilon E}} \eta((k, l))$$

Da T_φ laut Voraussetzung die Zustandsfolge $\hat{\sigma}$ akzeptiert, ist ein finaler Knoten in $m_{T_\varphi}(n+1, \hat{\sigma}, \hat{\beta})$ enthalten und somit eine dieser Kantenbeschriftungen in σ_n erfüllt. Es folgt die Gültigkeit von φ in $\hat{\sigma}$.

"$\Leftarrow$" Es muß nun noch gezeigt werden, daß die Gültigkeit von φ die Akzeptanz durch T_φ impliziert.

Aufgrund des Satzes 4.4.3 kann sich die Beweisführung wieder auf die Betrachtung des letzten Zustands σ_n der Zustandsfolge $\hat{\sigma}$ beschränken. Es gilt (Satz 4.4.3)

$$\Big((\hat{\sigma}, \hat{\beta}) \models \varphi\Big) \quad \Rightarrow \quad \begin{array}{l} T_\varphi \text{ akzeptiert } \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \text{ unter } \hat{\beta} \text{ vorläufig} \\ \wedge\ (\hat{\sigma}_n, \hat{\beta}) \models \displaystyle\bigvee_{k \,\epsilon\, m(n, \hat{\sigma}, \hat{\beta})} \nu(k) \end{array}$$

Die aktuelle Formel ist wieder äquivalent zur Disjunktion $(*)$. Da φ in $\hat{\sigma}$ gültig ist, ist die aktuelle Formel in $\hat{\sigma}_n$ gültig (Satz 4.4.3) und somit eine der Zeilen von $(*)$. Da σ_n der letzte Zustand von $\hat{\sigma}$ ist, kann diese Zeile keinen **existsnext**–Bestandteil haben. Es folgt, daß die neue Markierung einen Knoten aus F enthält und somit $\hat{\sigma}$ von T_φ akzeptiert wird. $\square$

Wenn für die Auswertung der Kantenbeschriftungen ein vollständiges und konsistentes Beweissystem für die Prädikatenlogik benutzt wird, folgt aus diesem Satz, daß das Beweisen mit Transitionsgraphen für endliche Zustandsfolgen ein konsistentes und vollständiges Prüfsystem für eine große Klasse temporaler Formeln ist.

Für unendliche Zustandsfolgen gilt zusätzlich :

Satz 4.4.5 Sei T_φ ein Prüfschema für die Formel φ. Für eine beliebige Zustandsfolge $\hat{\sigma}$ gilt :

$$((\hat{\sigma}, \hat{\beta}) \not\vdash T_\varphi) \quad \Rightarrow \quad ((\hat{\sigma}, \hat{\beta}) \not\models \varphi)$$

Beweis : Für endliche Zustandsfolgen ist diese Behauptung bereits bewiesen. Unendliche Zustandsfolgen werden nicht akzeptiert, falls ein Zeitpunkt n existiert, an dem die aktuelle Markierung leer ist (Definition 4.4.2). Die aktuelle Formel (leere Disjunktion) ist **false**. Aus Satz 4.4.3 folgt die Ungültigkeit von φ, da die Formel **false** von keiner Zustandsfolge erfüllt werden kann. □

Definition 4.4.6 Für einen Transitionsgraphen T, in dessen Kanten– und Knotenbeschriftungen freie Variablen vorkommen können, ist die *belegungsunabhängige Akzeptanz* definiert durch

$$\Big(\hat{\sigma} \vdash T\Big) \;\text{ gdw. }\; \Big((\hat{\sigma}|_\beta, [\beta]) \vdash T \text{ für alle } \beta \in \bigcup_i BEL(X, I(\sigma_i))\Big)$$

□

Intuitiv bedeutet diese Definition, daß der Lebenslauf aller einzelnen “Objekte” getrennt überprüft wird. Diese Festlegung entspricht der Semantik der Reduktquantifizierung in der finit-linearen temporalen Logik und der durch sie definierten belegungsunabhängigen Gütigkeit von temporalen Formeln. Existiert ein Prüfschema für eine temporale Formel, gilt der folgende Zusammenhang zwischen belegungsunabhängiger Gültigkeit und Akzeptanz für temporale Formeln.

Satz 4.4.7 Ist für alle $\beta \in \bigcup_i BEL(X, I(\sigma_i))$ das jeweilige β–Redukt endlich (also $\Big|\, \hat{\sigma}|_\beta \,\Big| \in I\!N$), so gilt

$$\hat{\sigma} \models \varphi \quad \text{gdw.} \quad \hat{\sigma} \vdash T_\varphi$$

Beweis : Die Behauptung folgt direkt aus der Definition 3.4.1 der belegungsunabhängigen Gültigkeit von Seite 37 und dem Satz 4.4.4 über die Akzeptanz von Zustandsfolgen unter einer festen Belegung. □

Definition 4.4.8 (Erweiterung von Definition 4.4.2) Ein konsistent beschrifteter Transitionsgraph T *akzeptiert eine unendliche Zustandsfolge* $\hat{\sigma}$ wenn für ein $i \in I\!N$ gilt

$$\Big(\exists k \in m_T(i, \hat{\sigma}, \hat{\beta})\Big) : \nu(k) \text{ ist eine Tautologie unter } \hat{\beta}$$

□

Satz 4.4.9 Sei T_φ ein Prüfschema für φ. Die Akzeptanz einer unendlichen Zustandsfolge $\hat{\sigma}$ impliziert die Gültigkeit von φ in $\hat{\sigma}$.

Beweis : Eine Tautologie (für eine feste Belegung oder belegungsunabhängig) ist in *allen* Zustandsfolgen erfüllt, also auch in der Restzustandsfolge $\hat{\sigma}_i$. Da der Transitionsgraph konsistent beschriftet ist, gilt Satz 4.4.3 für die Formel $\varphi := \bigvee_{k \,\epsilon\, m_0} \nu(k)$. Die Akzeptanz für unendliche Zustandsfolgen besagt, daß die aktuelle Formel irgendwann eine Tautologie ist. Aus beiden Aussagen folgt die Gültigkeit von φ in $\hat{\sigma}$. □

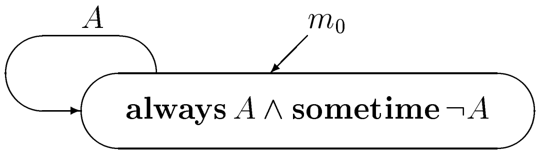
Die Definition der Akzeptanz von unendlichen Zustandsfolgen kann auch für endliche Zustandsfolgen modifiziert werden, um die Ausführung eines Beweises beim frühestmöglichen Zustand der Zustandsfolge abschließen zu können. Für die Überwachung von temporalen Formeln während der Laufzeit eines Systems (siehe Kapitel 6) ermöglicht es diese Eigenschaft, bestimmte Belegungen aus der laufenden Überprüfung herauszunehmen, falls die aktuelle Markierung z.B. einen mit **true** beschrifteten Knoten enthält.

Die Betrachtung von Tautologien *unter einer festen Belegung* ermöglicht eine zusätzliche differenzierte Behandlung von Sonderfällen, z.B. der temporalen Formel **always**$(f(x) = f(y))$, die für eine Belegung β mit $\beta(x) = \beta(y)$ eine Tautologie ist.

4.5 Reduktion von Transitionsgraphen

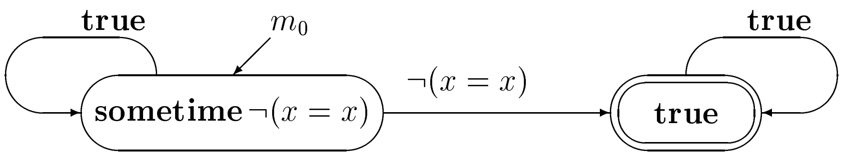
Das Beweisen von temporalen Formeln mit Transitionsgraphen als Prüfschemata ist — zumindestens für endliche Zustandsfolgen — ein konsistentes und vollständiges Beweisverfahren für eine große Klasse temporaler Formeln. Da der "temporale" Anteil der Formeln im Konstruktionsalgorithmus 4.3.7 rein syntaktisch durch Eliminierung der temporalen Quantifizierungen mittels der temporalen Rekursionsregeln analysiert wird, kann die Unerfüllbarkeit einiger temporaler Formeln in Beweisen mit den konstruierten Prüfschemata nicht berücksichtigt werden.

Beispiel 4.5.1 Aus der Formel $\varphi :=$ **always** $A \wedge$ **sometime** $\neg A$ wird der folgende Transitionsgraph konstruiert :



Da die Äquivalenz $\textbf{sometime}\,\neg A \iff \neg\,\textbf{always}\,A$ gilt, ist die Formel φ ein temporallogischer Widerspruch. Der Graph akzeptiert keine Zustandsfolge, da kein finaler Knoten in ihm enthalten ist. Der Beweis der Nichterfülltheit von φ erfolgt aber erst beim ersten Zustand, in dem $\neg A$ gilt, bzw. im letzten Zustand einer endlichen Folge — für unendliche Zustandsfolgen, in denen in jedem Zustand A gilt, erfolgt dieser Beweis nie.

Ein zweites Beispiel ist die Formel $\textbf{sometime}\,\neg(x = x)$, die ebenfalls unerfüllbar ist. Aus ihr wird der folgender Graph konstruiert :



Auch dieser Transitionsgraph akzeptiert keine Zustandsfolge — die Kantenbeschriftung $\neg(x = x)$ ist unerfüllbar und der **sometime**–Knoten ist nicht final. □

Ziel der *Reduktion* von Transitionsgraphen ist es, unerfüllbare Teilgraphen aus dem Gesamtgraphen zu entfernen. Reduzierte Transitionsgraphen erkennen unvermeidbare zukünftige Fehler (d.h. zwingende Nichtakzeptanz der Restzustandsfolge) zum frühest möglichen Zeitpunkt.

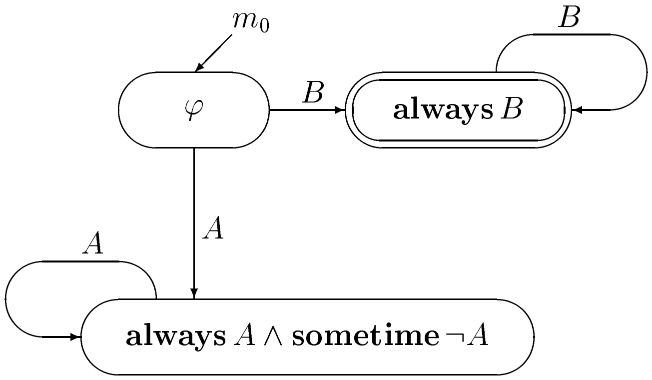
Beispiel 4.5.2 Die Formel

$$\varphi := (\textbf{always}\,A \wedge \textbf{sometime}\,\neg A) \vee \textbf{always}\,B$$

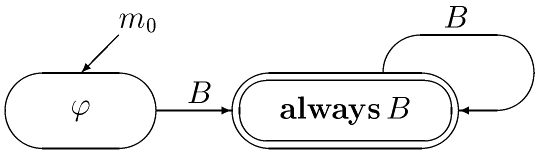
enthält als Teilformel die unerfüllbare Formel $(\textbf{always}\,A \wedge \textbf{sometime}\,\neg A)$. Der konstruierte Transitionsgraph T_φ ist in Bild 8 auf Seite 63 abgebildet. Die Nichtgültigkeit einer (möglicherweise unendlichen) Zustandsfolge, in der in jedem Zustand $A \wedge \neg B$ gilt, kann wiederum erst nach vollständiger Abarbeitung der Folge gezeigt werden.

Der aus dem Knoten mit der Beschriftung $(\textbf{always}\,A \wedge \textbf{sometime}\,\neg A)$ konstruierte Teilgraph entspricht einer unerfüllbaren Formel und kann darum durch ein Reduktionsverfahren entfernt werden. Der Graph nach erfolgter Reduktion ist ebenfalls in Bild 8 abgebildet. □

Die Reduzierung von Prüfschemata hat also das Ziel, daß ein Prüfschema für eine Formel φ nach der Reduktion eine endliche Zustandsfolge nur dann vorläufig akzeptiert, wenn diese Zustandsfolge zu einer Zustandsfolge verlängert werden kann, in der φ gültig ist.



Prüfschema für $\varphi :=$ (**always** $A \wedge$ **sometime** $\neg A) \vee$ **always** B



Reduziertes Prüfschema für die Formel φ

Abbildung 8: Beispiel für Reduktion von Transitionsgraphen

Definition 4.5.3 Ein Prüfschema T_φ heißt *reduziert*, wenn gilt

$$\Big(T_\varphi \text{ akzeptiert } \quad \langle \sigma_0, \ldots \sigma_i \rangle \quad \text{unter } \hat{\beta} \text{ vorläufig}\Big)$$
$$\Rightarrow \qquad \Big((\exists \hat{\sigma}' \,\epsilon\, (\Im_\Sigma{}^+ \cup \lambda))(\langle \sigma_0, \ldots \sigma_i \rangle \circ \hat{\sigma}', \hat{\beta}) \models \varphi\Big)$$

wobei λ die leere Zustandsfolge und $\circ$ die Konkatenation von Folgen bezeichnet. Es wird im folgenden unterschieden zwischen Reduziertheit bezüglich der Existenz endlicher Zustandsfolgen und allgemeiner Reduziertheit. □

Die Reduktion kann in der Regel nur in Abhängigkeit von einer festen Belegung vorgenommen werden. Der Reduktionsvorgang kann belegungsunabhängig vorgenommen werden, wobei obige Charakterisierung der Reduziertheit dann nicht auf alle Belegungen zutrifft.

Aus den Beispielen geht hervor, daß verschiedene Techniken zur Reduktion von Transitionsgraphen anwendbar sind. Merkmale der Teilgraphen, die unerfüllbaren Formeln entsprechen, sind in den Beispielen unerfüllbare Kantenbeschriftungen und das Fehlen von Kanten zu finalen Knoten.

Satz 4.5.4 Das Entfernen von unerfüllbaren Kanten aus einem Transitionsgraphen hat keinen Einfluß auf die Menge der durch den Transitionsgraphen akzeptierten

Zustandsfolgen. Sei T' der Transitionsgraph, der sich aus T durch Streichen von (unter $\hat{\beta}$) unerfüllbaren Kanten ergibt. Es gilt für alle i und alle $\hat{\sigma}$:

$$m_T(i, \hat{\sigma}, \hat{\beta}) = m_{T'}(i, \hat{\sigma}, \hat{\beta})$$

Beweis : Diese Eigenschaft ergibt sich direkt aus der Definition der Transitionsregel. Die jeweils neue Markierung wird nur aufgrund von gültigen Kantenbeschriftungen errechnet. □

Eine alternative Möglichkeit, unerfüllbare Kantenbeschriftungen zu vermeiden, ist das direkte Einbringen der Erfüllbarkeitskriterien für atomare Formeln in den Konstruktionsprozeß [Saa85, LS87]. Hierzu wird das Prüfschema für eine Formel φ aus der Formel $\varphi \wedge \textbf{always}\,\rho$ konstruiert, wobei die nichttemporale Formel ρ die für φ relevanten Aussagen über die Erfüllbarkeit der nichttemporalen Bestandteile enthält.

Enthält eine temporale Formel φ zum Beispiel die beiden atomaren Formeln A und B, die sich gegenseitig ausschließen, würde das Prüfschema für die Formel

$$\varphi \wedge \textbf{always}((A \Rightarrow \neg B) \wedge (B \Rightarrow \neg A))$$

konstruiert werden. Die Bestandteile der Formel ρ müssen aus den prädikatenlogischen Axiomen erzeugt werden, die die einzelnen Zustände einer Folge einschränken.

Satz 4.5.5 Das Entfernen von Knoten aus Transitionsgraphen, von denen kein finaler Knoten aus über Kantenzüge erreichbar ist, hat keinen Einfluß auf die Akzeptanz endlicher Zustandsfolgen. Insbesondere können Knoten aus $(N - F)$ entfernt werden, von denen keine Kante ausgeht.

Beweis : Die Akzeptanz von endlichen Zustandsfolgen ist dann gegeben, wenn in der letzten Markierung mindestens ein finaler Knoten enthalten ist. Die nicht finalen Knoten in der letzten Markierung haben keinen Einfluß auf die Akzeptanz.

Ein finaler Knoten ist genau dann in der letzten Markierung, wenn ein Pfad durch den Graphen existiert, der an einem Knoten der Anfangsmarkierung beginnt, dessen Kanten im jeweiligen Zustand der Zustandsfolge gültig sind und an dem finalen Knoten endet. Knoten, von denen aus kein Kantenzug zu einem finalen Knoten führt, können nicht an derartigen Pfaden beteiligt sein. □

Diese beiden Sätze bilden die Grundlage für den folgenden Reduktionsalgorithmus bezüglich endlicher Zustandsfolgen.

Algorithmus 4.5.6 Gegeben sei ein Transitionsgraph $T = (G = \langle N, E \rangle, \nu, \eta, m_0, F)$. Der *reduzierte Transitionsgraph* $RT = (G' = \langle N', E' \rangle, \nu', \eta', m'_0, F')$ wird aus T folgendermaßen konstruiert. E^* bezeichnet die reflexive, transitive Hülle der Kantenrelation E.

(1) Entferne alle Kanten aus E mit unerfüllbaren Beschriftungen;
(2) $N' := \{k \in N \mid (\exists k' \in F)(k, k') \in E^*\}$;
(* N' enthält die Knoten, von denen aus ein finaler Knoten erreichbar ist. *)
(3) $N' := \{k \in N' \mid (\exists k' \in m_0 \cap N')(k', k) \in E^*\}$;
(* Aus N' werden die Knoten entfernt, die nicht von einem Anfangsknoten aus erreichbar sind. *)
(4) (* Anpassung der übrigen Komponenten *)
$E' := E|_{N'}$;
$\nu' := \nu|_{N'}$;
$\eta' := \eta|_{E'}$;
$m_0' := m_0 \cap N'$;
$F' := F \cap N'$.

Entgegen der ursprünglichen Definition kann der Algorithmus einen *leeren* Transitionsgraphen erzeugen, wenn die Knoten der Anfangsmarkierung aus dem Graphen entfernt werden. Der leere Transitionsgraph akzeptiert keine Zustandsfolge. □

Der Schritt (1) des Algorithmus kann unter einer gegebenen Belegung oder belegungsunabhängig vorgenommen werden. Bei einer *belegungsunabhängigen Reduktion* werden in Schritt (1) in der Regel weniger Kanten entfernt. Insbesondere verbleiben dabei diejenigen Kanten im Graphen, deren Beschriftungen nur für einige Belegungen unerfüllbar sind. Aufgrund des Berechnungsaufwands wird in den folgenen Kapiteln oft nur die belegungsunabhängige Reduktion eingesetzt, obwohl die erhaltenen Prüfschemata für bestimmte Belegungen nicht vollständig reduziert sind.

Möglich ist ebenfalls eine Reduzierung für ausgewählte *Klassen von Belegungen*. Enthält eine Formel z.B. die atomare Formel $(x = y)$, so kann für die Belegungen, die x und y denselben Wert zuweisen, und für die übrigen Belegungen getrennt reduziert werden.

Satz 4.5.7 Sei T ein Transitionsgraph und $\hat{\sigma}$ eine endliche Zustandsfolge. Es gilt :

$$(\hat{\sigma}, \hat{\beta}) \vdash T \text{ gdw. } (\hat{\sigma}, \hat{\beta}) \vdash RT$$

Beweis : Der Schritt (1) des Reduktionsalgorithmus hat keinen Einfluß auf die Akzeptanz (Satz 4.5.4). Aufgrund Satz 4.5.5 hat das Eliminieren von Knoten aus $(N - F)$, von denen kein Pfad zu einem finalen Knoten führt (Schritt (2)), ebenfalls keinen Einfluß auf die Akzeptanz. In einer aktuellen Markierung können aufgrund der Definition der Transitionsregel nur Knoten enthalten sein, die von einem Knoten aus m_0 aus erreichbar sind (Korrektheit von Schritt (3)). □

Satz 4.5.8 Eine temporale Formel φ mit einem Prüfschema T_φ ist genau dann von einer endlichen Zustandsfolge unter der Belegung $\hat{\beta}$ erfüllbar, wenn die belegungsabhängige[5] Reduktion bezüglich $\hat{\beta}$ nach Algorithmus 4.5.6 nicht den leeren Transitionsgraphen als Ergebnis RT_φ hat.

Beweis : Der leere Transitionsgraph akzeptiert laut Definition keine Zustandsfolge.

Im reduzierten, nicht leeren Transitionsgraph existiert mindestens ein Pfad mit i Kanten von einem Knoten der Anfangsmarkierung zu einem finalen Knoten, da im Schritt (2) alle Knoten entfernt werden, von denen kein Pfad zu einem finalen Knoten führt, und in Schritt (3) alle (finalen) Knoten entfernt werden, die nicht von einem Knoten aus m'_0 erreichbar sind.

Da in Schritt (1) des Algorithmus alle unter $\hat{\beta}$ nichterfüllbaren Kantenbeschriftungen entfernt wurden, läßt sich aus dem Pfad direkt eine Zustandsfolge $\hat{\sigma}'$ der Länge i konstruieren, die von RT_φ unter $\hat{\beta}$ akzeptiert wird. Hierzu wird jeder Kante des Pfades eine Weltinterpretation zugeordnet, die die Kantenbeschriftung erfüllt. Da die Kantenbeschriftung erfüllbar ist, existiert jeweils eine derartige Weltinterpretation. Die erhaltene Zustandsfolge erfüllt φ unter $\hat{\beta}$.

Es gilt

$$(\hat{\sigma}', \hat{\beta}) \vdash RT_\varphi \;\text{ gdw. }\; (\hat{\sigma}', \hat{\beta}) \vdash T_\varphi \;\text{ gdw. }\; (\hat{\sigma}', \hat{\beta}) \models \varphi$$

aufgrund der Sätze über die Korrektheit der Reduktionsschritte und der Äquivalenz von Akzeptanz und Gültigkeit für endliche Zustandsfolgen. □

Satz 4.5.9 Der Reduktionsalgorithmus 4.5.6 konstruiert aus einem Prüfschema T_φ ein korrekt reduziertes Prüfschema RT_φ für endliche Zustandsfolgen (bezüglich einer Belegung $\hat{\beta}$).

Beweis : Ein Prüfschema heißt reduziert, wenn aus der Akzeptanz eines Anfangstücks $\hat{\sigma}_0^i = \langle \sigma_0, \dots, \sigma_i \rangle$ einer Zustandsfolge folgt, daß eine Fortsetzung $\hat{\sigma}'$ dieses Anfangsstücks existiert, so daß die Gesamtfolge $\hat{\sigma}_0^i \circ \hat{\sigma}'$ die Formel φ erfüllt.

Ist in der letzten Markierung $m_{RT_\varphi}(i+1, \hat{\sigma}_0^i, \hat{\beta})$ ein finaler Knoten enthalten, so erfüllt $\hat{\sigma} = \lambda$ den Satz aufgrund von Satz 4.4.4 (Äquivalenz von Akzeptanz und Gültigkeit für Prüfschemata).

Andernfalls existiert ein Knoten $k \in m_{RT_\varphi}(i+1, \hat{\sigma}_0^i, \hat{\beta})$, vom dem aus ein Pfad zu einem finalen Knoten führt, auf dem alle Kantenbeschriftungen erfüllbar sind (Reduktionsalgorithmus 4.5.6). Es kann somit die Konstruktion einer erfüllenden Zustandsfolge aus Satz 4.5.8 angewandt werden.

[5] Notwendig ist eigentlich nicht die Reduktion für die feste Belegung, sondern daß diese Belegung zu einer Klasse von Belegungen gehört, für die in der Reduktion alle nichterfüllbaren Kanten entfernt wurden. Als Beispiel müssen für eine Belegung, die x und y verschiedene Werte zuweisen, alle Kanten als unerfüllbar entfernt werden, die mit $(x = y)$ beschriftet sind.

Eine temporale Formel ist genau dann erfüllbar, wenn das reduzierte Prüfschema nicht leer ist (Satz 4.5.8). Da die Reduktion im Schritt (3) des Reduktionsalgorithmus auf E^* basiert, gilt diese Eigenschaft auch für von Knoten ausgehende Teilgraphen des reduzierten Prüfschemas, die nicht entfernt werden. □

Die Reduktion von Transitionsgraphen ist also korrekt für endliche Zustandsfolge in dem Sinne, daß reduzierte Prüfschemata ebenfalls genau die Zustandsfolgen akzeptieren, in denen die zugrundeliegende Formel gültig ist. Der Reduktionsalgorithmus ist allerdings nicht korrekt bezüglich der Nichtakzeptanz unendlicher Zustandsfolgen.

Beispiel 4.5.10 Die Formel $\varphi = \textbf{always}(\textbf{sometime}\, A \wedge \textbf{sometime}\, \neg A)$ ist eine temporale Formel, die zwar in unendlichen Zustandsfolgen erfüllt sein kann (z.B. von einer unendlichen Zustandsfolge, in denen abwechselnd A und $\neg A$ erfüllt sind), aber von keiner endlichen Zustandsfolge erfüllbar ist. Gilt nämlich im letzten Zustand A, so ist die in jeder Restzustandsfolge zu erfüllende Bedingung **sometime** $\neg A$ nicht erfüllt und umgekehrt.

Das Prüfschema T_φ enthält keine finale Knoten und der reduzierte Transitionsgraph RT_φ ist leer — dies entspricht der Tatsache, daß φ in keiner endlichen Zustandsfolge erfüllt sein kann. □

Es existieren also temporale Formeln, die nur von unendlichen Zustandsfolgen erfüllt sein können. Derartige Formeln können in allgemeinen Spezifikationen durchaus sinnvoll sein — sogenannte Lebendigkeitsforderungen ('liveness requirements') werden typischerweise durch ähnliche Formeln ausgedrückt.

Das Ziel der Spezifikation mittels temporaler Logik ist in dieser Arbeit das formale Beschreiben der Lebensläufe von informationstragenden Objekten in einer Datenbank. Da diese Objekte in der Regel nur eine begrenzte Lebensdauer im zu entwerfenden Gesamtsystem haben, ist im Normalfall die beschriebene Reduktionsmethode für die Entwurfsunterstützung und Überwachung geeignet (vergl. Kapitel 5 und 6).

Reduktionsverfahren für unendliche Zustandsfolgen sind naturgemäß komplexer als diejenigen für endliche Zustandsfolgen. Für endliche Zustandsfolgen genügt es, zu zeigen, daß es für jede mögliche aktuelle Markierung möglich ist, eine Markierung zu erreichen, die eine Beendigung des Beweises der zugrundeliegenden Formel ermöglicht. Derartige Markierungen sind in Prüfschemata genau diejenigen, die finale Knoten enthalten. Daraus folgt ein Reduktionsalgorithmus, der einzig auf der Struktur des Transitionsgraphen beruht.

Bei der Reduktion für unendliche Zustandsfolgen muß für jede mögliche Markierung gezeigt werden, daß eine *unendliche Folge von Folgemarkierungen* existiert, in der jede temporale Existenzforderung jeder in ihr vorkommenden aktuellen Formel erfüllt wird. *Temporale Existenzforderungen* sind Teilformeln wie **sometime**... oder ... **before** ..., die Aussagen über die Existenz zukünftiger Zustände bzw. Restzustandsfolgen ausdrücken.

Ein Algorithmus zur Reduktion von Transitionsgraphen für unendliche Zustandsfolgen kann in [MW84, LS87] gefunden werden. Hierbei werden bei der Konstruktion des Graphen zusätzliche Kantenmarkierungen erzeugt, die die Erfüllung von temporalen Existenzforderungen bei Knotenübergängen kennzeichnen. Die Reduktion entfernt dann diejenigen Knoten, die eine temporale Existenzforderung beinhalten, für die auf keinem von diesem Knoten ausgehenden Pfad eine Kante mit den passenden Kantenmarkierungen auftaucht.[6]

Ein ähnlicher Ansatz — allerdings für verzweigte temporale Logiken — wird in [ES85, VW86] vorgestellt. Dort wird die Erfüllbarkeit der temporalen Knotenbeschriftungen auf das Wortproblem für bestimmte Klassen von endlichen Automaten über unendlichen Baumstrukturen zurückgeführt.

Da temporale Existenzforderungen Formeln entsprechen, deren disjunktive Normalform einen **existsnext**–Bestandteil hat, arbeitet der Reduktionsalgorithmus 4.5.6 in vielen Fällen auch für unendliche Zustandsfolgen korrekt (z.B. für **always** $A \wedge$ **sometime** $\neg A$), in einigen Fällen werden allerdings zu viele Knoten entfernt.

Die Reduktion von Transitionsgraphen kann belegungsunabhängig ein optimiertes Prüfschema für eine temporale Formel liefern und diese Formel gleichzeitig auf Unerfüllbarkeit hin untersuchen. Bei der belegungsunabhängigen Reduktion kann allerdings eine Formel unter einer festen Belegung unerfüllbar sein, obwohl der Reduktionsalgorithmus nicht den leeren Graph als Ergebnis hat.

Die belegungsunabhängige Reduktion kann in einer Entwurfsumgebung als Analysewerkzeug eingesetzt werden (siehe Kapitel 5) und zur Optimierung eines Integritätsmonitors zur Definitionszeit der Datenbank benutzt werden (siehe Kapitel 6). Die Reduktion für Klassen von Belegungen kann ebenfalls als Analysewerkzeug für die Erkennung unerfüllbarer Integritätsbedingungen und zur Optimierung des Integritätsmonitors eingesetzt werden. Offen ist bisher dabei die Bestimmung der Belegungsklassen für eine gegebene temporale Formel, für die dieser Mehraufwand bei der Analyse und Überwachung Vorteile hat.

Die belegungsabhängige Reduktion kann während der laufenden Überprüfung einer temporalen Formel eingesetzt werden, um die Nichtakzeptanz einer Zustandsfolge

[6] Die Reduktion für endliche Zustandsfolgen ist *nicht* ein Spezialfall der Reduktion für unendliche Zustandsfolgen, wie das Beispiel **sometime allnext false** zeigt. Diese Formel ist genau dann erfüllt, falls die Zustandsfolge endlich ist, ist aber von keiner unendlichen Zustandsfolge erfüllbar (und müßte bei einer entsprechenden Reduktion entfernt werden).

unter einer Belegung zum frühest möglichen Zeitpunkt zu erkennen. Diese Vorgehensweise verhindert während der Überprüfung durch einen Integritätsmonitor, daß die Überprüfung einer dynamischen Integritätsbedingung unter einen festen Belegung eine zwangsläufige zukünftige Verletzung dieser Integritätsbedingung nicht berücksichtigt.

4.6 Transitionsgraphen und endliche Automaten

Die Definition der Struktur eines Transitionsgraphen (der gerichtete Graph G, die Anfangsmarkierung m_0 und die Menge der finalen Knoten F) entspricht der Struktur eines endlichen, erkennenden Automaten. Auch die Transitionsregel eines Transitionsgraphen ähnelt stark der Zustandsüberführungsfunktion eines nichtdeterministischen Automaten, wenn die Formeln der Kantenbeschriftungen als Symbole eines Eingabealphabetes aufgefaßt werden.

Da die Theorie der endlichen erkennenden Automaten ein gründlich untersuchtes Forschungsgebiet der theoretischen Informatik ist (vergl. z.B. [Sal78] als Lehrbuch), lohnt sich eine eingehende Betrachtung der Beziehungen zwischen Transitionsgraphen und endlichen Automaten. Auf eine ausführliche Behandlung der Grundlagen der Automatentheorie wird an dieser Stelle verzichtet, hier sei auf die einschlägigen Lehrbücher verwiesen.

Definition 4.6.1 Ein (nichtdeterministischer) *endlicher erkennender Automat* oder *endlicher Akzeptor EA* wird beschrieben durch die Menge seiner Zustände S, ein Eingabealphabet I, eine Menge von Anfangszuständen $S_0 \subseteq S$, eine Menge von Endzuständen $S_F \subseteq S$ und die Überführungsfunktion $\delta\colon S \times I \to 2^S$. Ein endlicher Akzeptor heißt deterministisch, falls genau ein Anfangszustand existiert ($S_0 = \{s_0\}$) und die Überführungsfunktion jeweils genau einen Zustand ergibt, also $\delta\colon S \times I \to S$.

Ein endlicher erkennender Automat *erkennt* (oder auch *akzeptiert*) ein Wort $\omega = w_1 \circ w_2 \circ \cdots \circ w_n \,\epsilon\, I^*$ genau dann, wenn eine Folge $s_0, s_1, \ldots, s_n$ von Zuständen $s_i \,\epsilon\, S$ existiert mit

$$s_0 \,\epsilon\, S_0 \quad \wedge \quad s_n \,\epsilon\, S_F \quad \wedge \quad \text{für alle } i,\ 0 \leq i < n, \text{ gilt } s_{i+1} \,\epsilon\, \delta(s_i, w_{i+1})$$

Die Überführungsfunktion δ wird oft als Menge von Produktionsregeln $s_i w \to s_j$ notiert. Eine Produktion $s_i w \to s_j$ wird genau dann aufgeführt, wenn $s_j \,\epsilon\, \delta(s_i, w)$.

Die von einem endlichen Akzeptor *EA* erkannte *Sprache* $\mathcal{L}(EA)$ ist die Menge der von *EA* akzeptierten Worte aus I^*. □

Bis auf das Eingabealphabet lassen sich die Komponenten des endlichen Akzeptors direkt korrespondierenden Strukturkomponenten eines Transitionsgraphen zuordnen. Die oben erwähnte Wahl der Kantenbeschriftungen als Eingabealphabet ist aus mehreren Gründen nicht von Vorteil. Kantenbeschriftungen sind beliebige nichttemporale Formeln aus $P_\Sigma(X)$, im allgemeinen Fall ergibt das selbst für eine endliche Menge atomarer Formeln ein potentiell unendliches Alphabet. Schwerwiegender ist, daß bei dieser Wahl Abhängigkeiten zwischen Kantenbeschriftungen (z.B. von "A" und "$A \vee B$") nicht berücksichtigt werden können — es sei denn, man nehme als Alphabet gegen logische Konsequenz abgeschlossene Mengen von Formeln.

Als Symbole eines Eingabealphabets für endliche Automaten, die Transitionsgraphen entsprechen, bieten sich statt dessen Wahrheitswertkombinationen für eine Menge von Grundformeln GF aus $P_\Sigma(X)$ an. Diese Zuordnungen können als Zeilen z einer Wahrheitstafel $WT(GF)$ für die Formeln in GF aufgefaßt werden. Jeder Zeile $z \in WT(GF)$ ist eindeutig ein konjunktiver kanonischer Term $\zeta(z) \in P_\Sigma(X)$ zugeordnet, der genau dann gültig ist, wenn die Grundformeln die Wahrheitswerte gemäß z annehmen. In $\zeta(z)$ tritt jede Grundformel genau einmal auf, und sie ist genau dann negiert, falls ihr in z der Wert *falsch* zugeordnet ist.

Für einen gegebenen Transitionsgraphen T werden als Grundformeln GF_T die Grundformeln der Kantenbeschriftungen gemäß Definition 4.3.1 gewählt, also die auftretenden atomaren und quantifizierten Formeln, die nicht selbst Bestandteile einer quantifizierten Teilformel sind. Die Menge GF_T ist für einen Transitionsgraphen stets endlich und somit auch das Alphabet $WT(GF_T)$. Für ein Prüfschema T_φ können diese Mengen direkt durch Zerlegung der Formel φ konstruiert werden (Definition 4.2.1).

Jedem Zustand σ_i einer Zustandsfolge ist unter einer Belegung $\hat{\beta}$ genau eine Zeile z zugeordnet, die durch Auswerten der einzelnen Grundformeln in σ_i unter $\hat{\beta}$ berechnet werden kann. Diese zugeordnete Zeile wird mit $z(\sigma_i, \hat{\beta})$ bezeichnet. Unter dieser Voraussetzung kann die Akzeptanz von Zustandsfolgen durch den Transitionsgraphen T auf das Erkennen von Wörtern über $WT(GF_T)$ durch einen geeigneten endlichen Automaten zurückgeführt werden. Der Einfachheit halber wird auch von der Akzeptanz von Zustandsfolgen durch den Automaten gesprochen, wenn die Akzeptanz der zugeordneten Wörter gemeint ist.

Da die atomaren und quantifizierten Grundformeln i.a. nicht logisch unabhängig sind, kann das Alphabet $WT(GF)$ auf die Menge der möglichen z (d.h. derjenigen z, für die $\zeta(z)$ erfüllbar ist) beschränkt werden, ohne die Akzeptanz von Zustandsfolgen durch den endlichen Automaten zu beeinflussen.

Definition 4.6.2 Sei ein Transitionsgraph $T = (G = \langle N, E\rangle,\ \nu, \eta, m_0, F)$ mit Grundformeln GF_T gegeben. Der *korrespondierende endliche Akzeptor* $EA(T) =$

(S, I, S_0, S_F, δ) über dem Alphabet $I = WT(GF_T)$ ist definiert durch :

$$\begin{array}{rcl} S & := & N \\ I & := & WT(GF_T) \\ S_0 & := & m_0 \\ S_F & := & F \\ \delta & := & \left\{ s_i z \to s_j \;\middle|\; z \,\epsilon\, I \;\wedge\; (s_i, s_j) \,\epsilon\, E \;\wedge\; (\zeta(z) \Rightarrow \eta((s_i, s_j))\ ) \right\} \end{array}$$

□

Satz 4.6.3 Der korrespondierende endliche Akzeptor $EA(T)$ akzeptiert ein endliches Wort $\omega = z(\sigma_0, \hat{\beta}) \circ \cdots \circ z(\sigma_n, \hat{\beta})$ über I genau dann, wenn der zugrundeliegende Transitionsgraph T die Zustandsfolge $\hat{\sigma} = \langle \sigma_0, \ldots, \sigma_n \rangle$ unter $\hat{\beta}$ akzeptiert.

Beweis : Akzeptiert $EA(T)$ das Wort ω, so existiert eine Folge $s_0, s_1, \ldots, s_n$ von Zuständen $s_i \,\epsilon\, S$ mit

$$s_0 \,\epsilon\, S_0 \quad \wedge \quad s_n \,\epsilon\, S_F \quad \wedge \quad \text{für alle } i,\ 0 \leq i < n, \text{ gilt } s_{i+1} \,\epsilon\, \delta(s_i, z(\sigma_i, \hat{\beta}))$$

Dieser Folge ist ein Pfad durch den Transitionsgraphen T zugeordnet, da die Überführungsfunktion δ gemäß der Definition nur entlang von Kanten aus E abbildet. Der Pfad startet an einem Knoten der Anfangsmarkierung und endet an einem finalen Knoten des Transitionsgraphen. Nach Definition 4.4.2 der Akzeptanz durch Transitionsgraphen muß gezeigt werden, daß dieser finale Knoten in der letzten Markierung enthalten ist — dies ist genau dann der Fall, wenn die Beschriftung der i–ten Kante des Pfades im i–ten Zustand σ_{i-1} gültig ist (Definition der Transitionsregel 4.4.2).

Es gilt $s_{i+1} \,\epsilon\, \delta(s_i, z(\sigma_i, \hat{\beta}))$ und $\sigma_i \models \zeta(z(\sigma_i, \hat{\beta}))$. Aufgrund der Definition der Überführungsfunktion existiert die Kante (s_i, s_j) und es gilt $\zeta(z(\sigma_i, \hat{\beta})) \Rightarrow \eta((s_i, s_j))$. Daraus folgt die Akzeptanz durch den Transitionsgraphen.

Analog existiert bei Akzeptanz durch den Transitionsgraphen eine Knotenfolge aus den einzelnen aktuellen Markierungen, so daß jeweils der nächste Knoten im Ergebnis der Transitionsfunktion für den vorigen Knoten enthalten ist und der letzte Knoten final ist. Diese Knotenfolge definiert die Folge von Zuständen aus S, die die Akzeptanz durch den endlichen Automaten festlegt. □

Jedem endlichen Akzeptor mit einem Alphabet $I = WT(GF)$ kann ein äquivalenter Transitionsgraph zugeordnet werden. Knoten, Anfangsmarkierung und finale Knoten werden wieder direkt übernommen. Die Kanten des Graphen G sind durch die Überführungsfunktion festgelegt :

$$E := \{(s_i, s_j) \mid (s_i z \to s_j) \; \epsilon \; \delta\}$$

Die Kantenbeschriftung wird aus der Disjunktion der Formeldarstellungen derjenigen Zeilen der Wahrheitstafel gebildet, die in den Produktionsregeln der Überführungsfunktion den entsprechenden Zustandsübergang festlegen :

$$\eta((s_i, s_j)) := \bigvee_{s_i z \to s_j} \zeta(z)$$

Die Knotenbeschriftung kann beliebig gewählt werden, da sie auf die Akzeptanz keinen Einfluß hat — der erhaltene Transitionsgraph ist dann natürlich in der Regel nicht konsistent beschriftet.

Diese Definitionen ermöglichen die Transformation von Transitionsgraphen in endliche Akzeptoren, die Analyse und Manipulation der erhaltenen Automaten mit Methoden der Automatentheorie und anschließende Rücktransformation in Transitionsgraphen. Die Transformationen zwischen Transitionsgraphen und endlichen Akzeptoren erhalten in beiden Richtungen die Akzeptanzbeziehung zu Zustandsfolgen.

Sind die zugrundeliegenden Transitionsgraphen Prüfschemata für Formeln (d.h. insbesondere konsistent beschriftet), so besteht ein besonderes Interesse daran, auch nach der Rücktransformation eine sinnvolle, konsistente Knotenbeschriftung zu erhalten. Es folgen einige nützliche Anwendungen von automatentheoretischen Algorithmen, ihre Bedeutung für Transitionsgraphen und besonders für Prüfschemata, und die konsistenzerhaltende Behandlung der Knotenbeschriftungen bei diesen Verfahren.

Die Algorithmen zur Manipulation von endlichen erkennenden Automaten werden jeweils nur kurz skizziert, die Beweise ihrer Korrektheit bezüglich der erkannten Sprachen können aus der Literatur entnommen werden.

Satz 4.6.4 Durch Konstruktion des Potenzmengenautomaten kann aus einem Prüfschema T_φ ein *deterministischer Potenzmengentransitionsgraph* DT_φ konstruiert werden. Wird als Knotenbeschriftung eines Knoten k die Disjunktion der Beschriftungen der in k enthaltenen Ursprungsknoten gewählt, so ist DT_φ ebenfalls ein Prüfschema für φ. Der der leeren Menge zugeordnete Fehlerknoten wird mit **false** beschriftet.

Beweisskizze : Die Knoten des Potenzmengenautomaten sind Teilmengen der Potenzmenge 2^S der Knotenmenge S des zugrundeliegenden nichtdeterministischen Automaten. Die Überführungsfunktion bildet jede dieser Zustandsmengen für ein Symbol $w \; \epsilon \; I$ genau in die Menge von Zuständen ab, die von Knoten dieser Menge durch den Übergang mit w erreicht werden. Der neue Anfangszustand wird durch die Menge der ursprünglichen Anfangszustände gebildet, die neue Menge finaler Knoten enthält genau die Teilmengen von S, die mindestens einen alten finalen Knoten enthalten.

Der Knoten des Potenzmengenautomaten (und des aus ihm rücktransformierten Transitionsgraphen), der in einer Berechnungsfolge erreicht wird, enthält genau alle Knoten des nichtdeterministischen Automaten (Transitionsgraphens), die Endknoten für Berechnungsfolgen für dasselbe Wort (dieselbe Zustandsfolge) im nichtdeterministischen Automaten (Transitionsgraphen) sind.

Alternative Berechnungsfolgen im nichtdeterministischen, konsistent beschrifteten Transitionsgraphen entsprechen aufgrund der Definition konsistenter Beschriftungen dem alternativen Auswerten von Zeilen einer Disjunktion. Die Beschriftung der Knoten des Potenzmengentransitionsgraphen mit den Disjunktionen der Knotenbeschriftungen führt aus diesem Grund wieder zu konsistent beschrifteten Graphen, falls der zugrundeliegende Graph konsistent beschriftet ist. Zusammen mit der Definition der Anfangsmarkierung folgt der Erhalt der Prüfschemaeigenschaft. $\square$

Satz 4.6.5 Seien φ^1, φ^2 temporale Formeln und $T_{\varphi^2}, T_{\varphi^2}$ Prüfschemata für diese Formeln. Mit Hilfe automatentheoretischer Algorithmen lassen sich aus T_{φ^1} und T_{φ^2} Prüfschemata für $\varphi^1 \vee \varphi^2$, $\varphi^1 \wedge \varphi^2$ und $\neg\varphi^1$ konstruieren.

Beweisskizze : Es werden jeweils nur die zugeordneten automatentheoretischen Algorithmen skizziert und die Wahl der neuen Knotenbeschriftung motiviert. Die Komponenten der beiden Akzeptoren $EA(T_{\varphi^1})$ und $EA(T_{\varphi^2})$ werden durch die hochgestellten Indizes 1 und 2 unterschieden.

- Dem $\vee$–Operator entspricht die Vereinigung der Sprachen von endlichen Akzeptoren. Der Akzeptor, der die Vereinigung zweier Sprachen erkennt, wird durch disjunkte Vereinigung der zugrundeliegenden Akzeptoren gebildet, wobei die neue Anfangsmarkierung die Vereinigung der alten Anfangsmarkierungen ist. Intuitiv entspricht das dem parallelen (oder alternativen) Arbeiten zweier Automaten.

  Die Knotenmarkierung kann direkt übernommen werden. Beide (unveränderten) Teiltransitionsgraphen sind weiterhin konsistent beschriftet und die neue Anfangsmarkierung ist äquivalent zu $\varphi^1 \vee \varphi^2$.

- Dem $\wedge$–Konnektiv entspricht die Bildung des Schnittes zweier Sprachen. Hierzu wird ein Akzeptor konstruiert, dessen Zustandsmenge das Kreuzprodukt $S^1 \times S^2$ der beiden Zustandsmengen der Akzeptoren ist. Anfangszustände, Endzustände und Überführungsfunktion werden folgendermaßen definiert :

$$S_0 \;:=\; \{(s_1, s_2) \mid s_1 \,\epsilon\, S_0^1 \wedge s_2 \,\epsilon\, S_0^2\}$$

$$
\begin{aligned}
S_f &:= \{(s_1, s_2) \mid s_1 \in S_F^1 \wedge s_2 \in S_F^2\} \\
\delta &:= \left\{(s_1, s_2)z \rightarrow (s_1', s_2') \;\middle|\; (s_1 z \rightarrow s_1') \in \delta^1 \wedge (s_2 z \rightarrow s_2') \in \delta^2\right\}
\end{aligned}
$$

Als Knotenmarkierung für (s_1, s_2) wird die Konjunktion $\nu(s_1) \wedge \nu(s_2)$ der Knotenbeschriftungen der beiden Bestandteile gewählt. Aufgrund der Distributivitätsgesetze (Distribution des $\wedge$ über Disjunktionen) ist $\varphi^1 \wedge \varphi^2$ äquivalent zu der Disjunktion der Knotenbeschriftungen in m_0. Auf diese Weise kann ebenfalls leicht der Erhalt der Eigenschaft "konsistent beschriftet" während der Konstruktion gezeigt werden.

- Für die Konstruktion des Akzeptors für $I^* - \mathcal{L}(EA(T_{\varphi^1}))$ muß in einem ersten Schritt der deterministische Akzeptor konstruiert werden.[7] In einem zweiten Schritt werden im Ergebnisautomaten die Mengen der Endzustände und der Nichtendzustände vertauscht.

  Die Knotenbeschriftungen des Potenzmengentransitionsgraphen werden negiert. Der Beweis, daß die Konstruktionsschritte (für deterministische Graphen !) die Prüfschema–Eigenschaft erhält, erfolgt durch Analyse der einzelnen Disjunktionen über die von Knoten ausgehenden Kanten (von denen in jedem Schritt jeweils genau eine Zeile einen gültigen nichttemporalen Bestandteil hat). Der Beweis kann in abgewandelter Form in [FL87] gefunden werden. □

Der Satz 4.6.5 definiert eine Algebra für Transitionsgraphen, deren Operatoren den aussagenlogischen Konnektiven zugeordnet sind. Ähnliche Operatoren sind Bestandteil der inkrementellen Konstruktionsmethode für deterministische Transitionsgraphen, die in [FL87, Li87] beschrieben ist. Dort werden zusätzlich rekursiv definierte Operatoren für temporale Quantoren eingeführt, die im Gegensatz zu den aussagenlogischen Operatoren zusätzlich zur Graphstruktur die Knotenbeschriftung berücksichtigen müssen.

Diese Algebra für Transitionsgraphen ermöglicht eine schrittweise Erweiterung bzw. Änderung von Transitionsgraphen ohne vollständige Neukonstruktion der Graphen. Sie ermöglicht somit z.B. ein effizientes Hinzufügen von dynamischen Integritätsbedingungen zum Datenbankschema zur Datenbanklaufzeit, ohne daß historische Information für die alten Integritätsbedingungen verlorengeht (vergl. Kapitel 6).

[7] Die nichtdeterministische Auswahl eines Berechnungspfades ausgehend von einem Knoten entspricht der nichtdeterministischen Auswahl einer Zeile einer disjunktiven Normalform der letzten Knotenbeschriftung. Nach den De Morganschen Regeln entspricht eine negierte Disjunktion einer Konjunktion. Aus diesem Grund muß der Transitionsgraph im ersten Schritt deterministisch gemacht werden, um eine korrekte Behandlung von disjunktiven Formeln zu gewährleisten.

Ähnlich wie in der Automatentheorie ermöglicht die Minimierung von Transitionsgraphen Aussagen über die Äquivalenz von Transitionsgraphen (Existenz von Graphisomorphismen) und somit auch über die Äquivalenz von Formeln bezüglich akzeptierter Zustandsfolgen — allerdings unter der oben motivierten Einschränkung auf endliche Zustandsfolgen.

Im Gegensatz zu den bisher vorgestellten Algorithmen kann die Zustandsminimierung für endliche Automaten (z.B. [Sal78]) allerdings nicht direkt übernommen werden. Für endliche Zustandsfolgen kann bezüglich der finalen Knoten minimiert werden, aber diese Minimierung kann nicht für unendliche Zustandsfolgen übernommen werden. Die Argumente gegen die Übernahme entsprechen denen für unterschiedliche Reduktionsverfahren für endliche und unendliche Zustandsfolgen. Für unendliche Zustandsfolgen müssen im Minimierungsalgorithmus alle Knoten wie Endzustände behandelt werden.

Im minimierten Automaten sind die Knoten Äquivalenzklassen bezüglich der akzeptierten Restworte. Als Knotenbeschriftung eines Knotens des minimierten Transitionsgraphen kann somit eine der Knotenbeschriftungen der in der Äquivalenzklasse enthaltenen Knoten genommen werden.

5 Spezifikation von Objektlebensläufen

Während des Entwurfs größerer Softwaresysteme kommt dem Entwurf der zentralen Datenbank, in der die für alle Systemkomponenten relevanten Informationen gehalten werden, eine besondere Rolle zu. Entwurfsfehler beim Entwurf dieser Systemkomponente wirken sich zwangsläufig auf alle anderen Komponenten aus, da sie alle auf die in der Datenbank gehaltenen Informationen angewiesen sind. Die Datenbank selber, d.h. die gespeicherten Informationen, zusammen mit den auf diesen Informationen erlaubten Operationen, wird im folgenden als *Datenbank-Anwendung* bezeichnet. Diejenige Systemkomponente, die die zentrale Datenbank verwaltet, muß die Konsistenz der gespeicherten Informationen gewährleisten und den anderen Systemkomponenten eine Schnittstelle anbieten, die einen implementierungsunabhängigen Zugriff auf die in der Datenbank gespeicherten Informationen ermöglicht.

Der Entwurf von Softwaresystemen wird in der Regel in aufeinander folgenden Phasen vollzogen, die von einer Analyse der Anforderungen über mehrere Zwischenschritte hin zu einer Implementierung des Systems führen. Die erste von diesen Entwurfsphasen, die in einer formalen Spezifikationssprache formuliert wird und Aussagen über die Konsistenz des Gesamtentwurfs ermöglicht, wird als *konzeptioneller Entwurf* bezeichnet. Der konzeptionelle Entwurf ist eine deskriptive, logikorientierte abstrakte Beschreibung des Systems und dient als Referenz für die Korrektheit der späteren Entwurfsphasen.

In diesem Kapitel wird eine Spezifikationsmethodik und -sprache für den konzeptionellen Entwurf von Datenbank-Anwendungen vorgestellt, die einen abstrakten, implementierungsunabhängigen Entwurf sowohl der statischen Struktur der Datenbank als auch der dynamischen Entwicklung zusammen mit den erlaubten Manipulationen ermöglicht. Der Schwerpunkt liegt auf der deskriptiven Beschreibung der erlaubten Entwicklungen der gespeicherten Informationen und deren formaler Semantik sowie auf der Beschreibung von entwurfsunterstützenden Werkzeugen für diesen Teilaspekt.

Auf die Besonderheiten des Entwurfs von Datenbank-Anwendungen im Vergleich zum allgemeinen Systementwurf wird im ersten Abschnitt 5.1 dieses Kapitels eingegangen. Da dieses allgemeine Gebiet nicht Schwerpunkt der vorliegenden Arbeit ist, wird nur ein kurzer Überblick über das Thema mit Verweisen auf aktuelle einführende Literatur geboten.

In den Abschnitt 5.2 und 5.3 wird eine Spezifikationsmethodik skizziert, die einen konsistenten Entwurf von Datenbank-Anwendungen ermöglicht. Der Schwerpunkt liegt hierbei auf dem strukturierten Entwurf und einer formalen Semantik einer derartigen Spezifikation, in der als Modellbildung für das dynamische Verhalten einer Datenbank Kripkestrukturen benutzt werden. Die Konstruktion von linearen Kripkestrukturen als Modelle für die konzeptionelle Spezifikation ist Inhalt von Abschnitt 5.3.

Die letzten beiden Teilkapitel beschäftigen sich mit Konsistenzkriterien für den dynamischen Anteil eines konzeptionellen Schemas. Es werden auf die Konstruktion von Prüfschemata basierende Verfahren für Konsistenzprüfungen vorgestellt, die eine Untersuchung der Entwicklungsschicht auf Erfüllbarkeit und auf Verträglichkeit zur Aktionsschicht ermöglichen.

5.1 Entwurf von Datenbank-Anwendungen

Um ein mehrfaches, redundantes Verwalten von Informationen, die von mehreren Komponenten eines komplexen Systems benötigt werden, und eine daraus resultierende Fehleranfälligkeit zu vermeiden, werden derartige Informationen in einer zentralen *Datenbank* verwaltet. Die Datenbank wird von einer speziellen Systemkomponente, dem *Datenbankmanagementsystem (DBMS)*, verwaltet, die den anderen Systemkomponenten den Zugriff auf die Informationen über eine implementierungsunabhängige Schnittstelle ermöglicht und die *Konsistenz* der gespeicherten Informationen gewährleistet.

Die Konsistenz kann auf verschiedenste Weise verletzt werden, z.B. durch fehlerhafte Eingaben, Programm- und Maschinenfehler, Widersprüche zur Weltmodellierung, etc. Die formale Angabe der Konsistenz- oder auch *Integritätsforderungen* umfaßt ein weites Spektrum (siehe z.B. [Reu87] als Überblick). In dieser Arbeit steht die Gewährleistung der *semantischen Integrität* im Vordergrund, d.h. die Berücksichtigung von sogenanntem semantischem Weltwissen über die Anwendung.

Der Entwurf einer Datenbank-Anwendung beinhaltet

- die Beschreibung der *Struktur* der gespeicherten Daten,
- die Angabe von *(statischen) Integritätsbedingungen*, die die erlaubten Ausprägungen ("Datenbankinhalte") der beschriebenen Struktur einschränken,
- die Angabe von *dynamischen Integritätsbedingungen*, die die erlaubten Änderungen des Datenbankinhalts einschränken,
- und den Entwurf der *Änderungstransaktionen*, die den anderen Systemkomponenten zur Manipulation der gespeicherten Daten angeboten werden.

Die Beschreibung einer Datenbank-Anwendung durch diese Komponenten wird als *Datenbankschema* bezeichnet. Die Sprache, in der ein Datenbankschema beschrieben ist, und auch die Art der Schemainformationen variieren während der Phasen des

Entwurfs. So müssen die statischen und dynamischen Integritätsbedingungen, da ihre Verwaltung von den meisten implementierten DBMS nicht unterstützt wird, in der Regel während des Entwurfsprozesses im Entwurf der Änderungstransaktionen berücksichtigt werden [Li87].

Der Entwurf einer Datenbank-Anwendung erfolgt in mehreren aufeinanderfolgenden Phasen, die den Phasen des "Software Lifecycle" des allgemeinen Softwareentwurfs entsprechen [MDL87]. In einem ersten Schritt, der sogenannten *Anforderungsanalyse*, muß der Informationsbedarf der geplanten Anwendung ermittelt und analysiert werden. Das Ergebnis dieses Schrittes ist eine informale Beschreibung der Anforderungen an die Datenbank-Anwendung.

Im folgenden *konzeptionellen Entwurf* wird eine formale Beschreibung der Datenbank und der Änderungstransaktionen erstellt. Der konzeptionelle Entwurf ist unabhängig von dem zur Implementierung benutzten DBMS (und dem von diesem unterstützten Datenmodell). Der konzeptionelle Entwurf ist die Grundlage für die weiteren Entwurfsphasen und Referenz für die Korrektheit der Implementierung.

Als nächste Entwurfsschritte folgen ein weiter detaillierter Entwurf und die *Implementierung* der Datenbank-Anwendung unter Benutzung eines existierenden DBMS. Diese Schritte werden als *logischer Entwurf* und *physischer Entwurf* bezeichnet. Der logische Entwurf beschreibt die Struktur der Informationen in einem Datenmodell, z.B. dem relationalen, das von einem implementierten DBMS unterstützt wird. Im physischen Entwurf werden Einzelheiten der physischen Repräsentation der Informationen festgelegt, so die Festlegung der Zugriffspfade u.ä., sowie die spezifizierten Änderungstransaktionen implementiert.

Die formale Spezifikation[8] der Datenbank-Anwendung erfolgt in der Phase des konzeptionellen Entwurfs. Die besonderen Anforderungen an eine Sprache für die Spezifikation eines konzeptionellen Datenbankschemas folgen aus der Stellung des konzeptionellen Entwurfs zwischen der informalen Anforderungsanalyse und der konkreten Implementierung der Anwendung :

- Die Spezifikationssprache muß eine formale Semantik haben, um eine Überprüfung des Entwurfs auf Konsistenz zu ermöglichen und gleichzeitig eine formale Basis für die Verifikation des logischen und physischen Entwurfs zu bilden.

- Die Sprache muß mächtige semantische Konzepte zur Umsetzung der informalen Anforderungen anbieten.

[8]Mit dem Begriff *Spezifikation* wird in dieser Arbeit ein formaler Entwurf bezeichnet, der in einem Logikkalkül formuliert ist, dessen mathematische Semantik Aussagen über die Konsistenz des Entwurfs ermöglicht.

- Die Sprache muß einen strukturierten Entwurf unterstützen, um auch für größere Anwendungen eingesetzt werden zu können.

- Da die Anforderungsanalyse im Gegensatz zum konzeptionellen Entwurf von Anwendern und nicht von Datenbankentwurfsspezialisten erstellt wird (oder zumindest in enger Zusammenarbeit mit den Anwendern), sollte der Entwurf auch für Nichtspezialisten intuitiv verständlich sein. Dies ermöglicht eine bessere Rückkopplung zwischen den Anwendern und den Entwerfern der Datenbank-Anwendung.

- In diesem Zusammenhang sind auch graphische Darstellungen der entworfenen Strukturen und dynamischen Abläufe von Vorteil.

Eine Spezifikationssprache und -methodik für den konzeptionellen Entwurf, die diesen Anforderungen genügt, wird in Abschnitt 5.2 vorgestellt. Der Schwerpunkt liegt auf der deskriptiven Beschreibung des dynamischen Verhaltens und auf dem Entwurf unterstützender Werkzeuge. Konkrete Sprachvorschläge für eine derartige Entwurfsmethode können in [HNSE87, SNHE87] bzw. [SSE87] gefunden werden.

Wichtig für den Entwurfsprozeß ist eine methodische Überführung von Entwürfen von einer Entwurfsphase in die nächste. Für Teile des konzeptionellen Datenbankschemas, z.B. für den Strukturanteil, ist eine (halb–)automatische Transformation durch entwurfsunterstützende Werkzeuge denkbar und wünschenswert [HNS86]. In Kapitel 6 dieser Arbeit werden Vorgehensweisen diskutiert, wie der Anteil der dynamischen Integritätsspezifikation eines konzeptionellen Datenbankschemas in der Implementierung berücksichtigt werden kann.

Im Vergleich zum Entwurf von Programmsystemen liegt beim Entwurf von Datenbank-Anwendungen der Schwerpunkt auf der Modellierung der Struktur der gespeicherten Informationen und weniger auf den operationellen Aspekten der Transaktionsimplementierung. Der Grund hierfür ist unter anderem, daß Zugriffsalgorithmen und Datenverwaltung oft durch ein fertig entworfenes und implementiertes DBMS angeboten werden, das in das Gesamtsystem eingebunden wird. Entwurfsziel ist somit, eine auf dieses DBMS zugeschnittene Beschreibung der Informationsstrukturen und Hauptzugriffspfade zu erstellen.

In der Praxis vollzieht sich der Entwurf einer Datenbank-Anwendung natürlich nicht ohne Rückkopplungen zwischen den Entwurfsphasen, und Entwurfsänderungen treten in jeder Phase des "Lebenslaufes" einer Datenbank-Anwendung auf — zum Teil erst während des Betriebes des Gesamtsystems.

Eine Datenbank-Anwendung ist ein *reaktives System* [Pn86], d.h. daß die zeitliche Entwicklung der gespeicherten Informationen in der Regel nicht zur Spezifikationszeit bekannt ist und somit auch nicht selbst Inhalt der Spezifikation ist. Statt dessen wird die Veränderung erst zur Laufzeit durch — prinzipiell in rein willkürlicher Reihenfolge erfolgende — Aufrufe der Änderungstransaktionen bestimmt. Inhalt der Spezifikation sind somit eher die *Einschränkungen* für die dynamische Entwicklung als konkrete Berechnungsabläufe, die im allgemeinen Softwareentwurf oft im Vordergrund stehen.

Im Unterschied zum Software-Lifecycle allgemeiner Programmsysteme ist zudem eine größere Bedeutung der Unterstützung von Entwurfsänderungen während der Systemlaufzeit zu beachten. Auf der einen Seite ist es in vielen Anwendungen notwendig, die einmal in die Datenbank aufgenommenen Informationen während der gesamten Systemlaufzeit konsistent zu speichern. Änderungen des Entwurfs der Datenbank müssen also sorgfältig durchgeführt werden. Andererseits besteht oft der Bedarf, den Entwurf der Datenbank-Anwendung zu ändern oder zu erweitern, da alle Systemkomponenten auf die Datenbank zugreifen und somit alle Änderungen des Gesamtsystems potentiell eine Änderung der Anforderungen an die Datenbank-Anwendung bedeuten.

5.2 Objektorientierte konzeptionelle Spezifikation

Für realistische Anwendungen ist der konzeptionelle Datenbankentwurf eine komplexe Aufgabe. Die Struktur des für die Anwendung relevanten Weltausschnittes muß auf einem möglichst hohen Abstraktionssniveau modelliert werden — diese Aufgabe umfaßt z.B. in einer ingenieurwissenschaftlichen Anwendung das Spektrum von elementaren Daten wie Materialkoeffizienten oder geometrischen Koordinaten bis hin zu komplex strukturierten Objekten wie kompletten Fertigungsanlagen.

Auf der anderen Seite sollte auch die Dynamik des beschriebenen Weltausschnittes auf einem vergleichbar hohen Abstraktionsniveau beschrieben werden, angefangen von rein deskriptiven Beschreibungen der erlaubten (oder erwünschten) Änderungen bis hin zu ganzen Arbeitsabläufen. Desweiteren müssen die Spezifikationskalküle 'verträglich' sein in dem Sinne, daß die Spezifikationen der einzelnen Komponenten auf Konsistenz zueinander überprüfbar sind und auch die Entwerfer eines konzeptionellen Schemas keine Umgewöhnungsschwierigkeiten beim Entwurf der verschiedenen Komponenten haben.

Um diesen Anforderungen an eine Datenbankspezifikation gerecht zu werden, wurde von verschiedenen Forschergruppen eine *mehrschichtige* Spezifikationsmethodik vorgeschlagen (u.a. [SFNC84, CS87, HNSE87, SSE87]). Ein konzeptionelles Schema wird in mehreren Schichten spezifiziert, die semantisch aufeinander aufbauen. Für jede Schicht wird ein Spezifikationskalkül angeboten, der eine der Komplexität der zu beschreibenden Informationen angemessenen Ausdrucksfähigkeit hat. Die Spezifikationskalküle bauen aufeinander auf sowohl bezüglich der semantischen Modellbildung als auch bezüglich der formalen Sprachdefinition.

Eine mehrschichtige Spezifikationsmethodik für konzeptionelle Datenbankschemata, in der die beschriebene temporale Logik zur Beschreibung des aktionsunabhängigen dynamischen Datenbankverhaltens benutzt wird, wird im folgenden kurz vorgestellt. Eine benutzergerechte Entwurfssprache für einen mehrschichtigen konzeptionellen Datenbankentwurf wird in [HNSE87, SNHE87] vorgestellt. Einzelheiten zur formalen Semantik einer derartigen mehrschichtigen Spezifikation können in [Eh86, EDG86] und [SSE87, ESS88] gefunden werden. An dieser Stelle werden hauptsächlich die Aspekte vorgestellt, die die dynamische Integrität betreffen.

Die Spezifikation eines konzeptionellen Datenbankschemas wird in vier Entwurfsschichten aufgeteilt, die *Daten-*, *Objekt-*, *Entwicklungs-* und *Aktionsschicht*. Die beiden ersten Schichten beschreiben die Struktur des modellierten Weltausschnittes und die erlaubten Ausprägungen dieser Struktur. Sie legen die Signatur einer Prädikatenlogik fest, in der die als Ausprägungen erlaubten Σ–Strukturen durch zusätzliche Axiome festgelegt sind. Die folgenden zwei Schichten beschreiben die zwei Aspekte des dynamischen Verhaltens der zu spezifizierenden Datenbank-Anwendung : die deskriptive Charakterisierung der erlaubten Zustandsänderungen unabhängig von konkreten Änderungsaktionen und die abstrakte Beschreibung der zur Anwendung gehörigen Änderungsaktionen.

Um einen strukturierten Entwurf zu ermöglichen, wird die Spezifikation in Module aufgeteilt, die die statischen und dynamischen Eigenschaften von informationstragenden Datenbankobjekten einer Objektsorte beschreiben. Beziehungen zwischen verschieden Objekten werden explizit spezifiziert, so daß Konsistenzprüfungen lokal in den Objektmodulen durchgeführt werden können.

Die Spezifikationskalküle der vier Schichten und ihre Semantik werden an dieser Stelle nur kurz anhand von Beispielen[9] skizziert. In [HNSE87, SNHE87] ist ein komplexes Beispiel für ein derartiges konzeptionelles Schema angegeben, das in einem durch zusätzliche Konzepte angereicherten mehrschichtigen Spezifikationskalkül formuliert ist. Eine ausführliche Beschreibung der benutzten semantischen Konzepte kann in [EDG86, SSE87, ESS88] gefunden werden.

```
DATATYPE point BASED ON real;
SORTS point;
OPERATIONS distance :       (point × point): real;
           xcoord, ycoord : (point): real;
           createpoint :    (real × real): point;
           add :            (point × point): point;
VARIABLES p,q : point;
          x,y,x1,y1 : real;
EQUATIONS
    x = xcoord(createpoint(x,y));
    y = ycoord(createpoint(x,y));
    distance(createpoint(x,y),createpoint(x1,y1))
      = sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1));
    add(p,q)
      = createpoint(xcoord(p)+xcoord(q),ycoord(p)+ycoord(q));
```

Abbildung 9: Spezifikation eines Datentyps durch Gleichungen

5.2.1 Datenschicht

Auf der *Datenschicht* wird die Struktur der elementaren Informationseinheiten spezifiziert. Diese elementaren Informationseinheiten oder *Datenwerte* sind nicht eigenständig als zu speichernde Informationen von Interesse, sondern tauchen konzeptionell nur als Eigenschaften (sogenannte *Attribute*) von informationstragenden Objekten auf. Zur Struktur der Daten gehört eine Aufteilung in Sorten und auf diesen erlaubten Operationen, deren Wirkung geeignet spezifiziert werden muß. Eine Datensorte mit zugehörigen Operationen wird als *Datentyp* bezeichnet.

Beispiele für Datentypen sind die ganzen Zahlen mit Operationen wie $+$ oder $*$, Punkte in der Zahlenebene mit zugehörigen Operationen, oder Aufzählungen mit einer Ordnungsrelation $<$. Ein Beispiel für eine Datentypspezifikation des Datentyps `points`, der als Wertebereich die Punkte der Zahlenebene hat, ist in Abbildung 9 gegeben.

Die Datentypen zeichnen sich durch zwei Eigenschaften aus, die sie von den in der Objektschicht spezifizierten informationstragenden Objekten unterscheidet : Sie sind durch die Summe ihrer beobachtbaren Eigenschaften eindeutig identifiziert, und ihre Eigenschaften ändern sich nicht während der Laufzeit der Datenbank-Anwendung.

[9] Als Spezifikationssprache wurden für die Daten- und Objektschicht Sprachkonzepte aus Spezifikationssprachen für abstrakte Datentypen [EM85] und aus OBLOG ([SSE87]) gewählt. Für die Spezifikation der Aktionsschicht wurde die Notation für die Transaktionsspezifikation aus [Li87] modifiziert übernommen.

```
OBJECTTYPE person;
IMPORT DATATYPES text, integer, datum, fstatus;
IMPORT OBJECTTYPES firma;
SORTS person;
KEY name : text;
    gebdat : datum;
KEYCONSTRAINTS
    FORALL(p : person) uppercase(first(name(p)));
ATTRIBUTES
    status : fstatus; (* Familienstand *)
    arbeitgeber : firma;
    gehalt : integer;
    lohnsteuer : integer;
CONSTRAINTS
    FORALL(p,q : person) gehalt(q) < gehalt(p)
                         IMPLIES lohnsteuer(q) < lohnsteuer(p);
```

Abbildung 10: Spezifikation der Objektklasse `person`

Zur Spezifikation von Datentypen und als formale Semantik kann die algebraische Gleichungsspezifikation von abstrakten Datentypen dienen, die z.B. in [EM85] beschrieben ist. Die Spezifikation durch (implizit allquantifizierte) Gleichungen ist eine Teilsprache des in Kapitel 2.1 beschriebenen mehrsortigen Prädikatenkalküls mit Gleichheit, für die effektive Methoden zur Konstruktion von Modellen, Ausführungswerkzeugen und konsistenzüberprüfenden Werkzeugen bekannt sind.

5.2.2 Objektschicht

Auf der *Objektschicht* werden die Struktur der informationstragenden Objekte beschrieben und die als Datenbankinhalt erlaubten Objektpopulationen durch statische Integritätsbedingungen eingeschränkt. Beispiele für Klassen derartiger Objekte sind Personen, Dokumente, Landkarten, etc. von denen Informationen (und Beziehungen zwischen diesen Informationen) in der Datenbank gespeichert werden sollen.

Da sich die Eigenschaften derartiger Objekte im Laufe der Datenbanklaufzeit typischerweise ändern können, wird ein zeitunabhängiger Identifikationsmechanismus benötigt. Dies geschieht durch die Angabe sogenannter *Schlüsselattribute*. Für Personen kann der Identifizierungsschlüssel[10] z.B. der Name zusammen mit dem Geburtsdatum sein.

[10] Sowohl die Struktur der informationstragenden Objekte als auch ihre Identifizierungsschlüssel sind Modellierungsentscheidungen für die konkrete Anwendung. Sie müssen also nicht mit den Gegebenheiten der realen Welt übereinstimmen, die modelliert werden sollen.

```
BEHAVIOUR OBJECTTYPE person;
DYNAMIC CONSTRAINT familienstandwechsel;
VARIABLES p : person;
   ALWAYS (status(p)= ledig IMPLIES
           ((status(p) = verheiratet BEFORE status(p) = geschieden)
            OR ALWAYS (status(p) = ledig) )
   AND ALWAYS(( NOT(status(p) = ledig)
              IMPLIES ALWAYS NOT(status(p) = ledig));
DYNAMIC CONSTRAINT gehaltsentwicklung;
VARIABLES p : person, f : firma, g : integer;
   ALWAYS (arbeitgeber(p) = f AND gehalt(p) = g
   IMPLIES ALWAYS (arbeitgeber(p) = f IMPLIES NOT gehalt(p) < g));
```

Abbildung 11: Beispiel für dynamische Integritätsbedingungen

Die Spezifikation einer Objektklasse besteht aus einer Signatur, in der der Sortenname und die Attributfunktionen angegeben werden, den Schlüsselattributen und aus statischen Integritätsbedingungen, die die Ausprägungen der Signatur einschränken. Die Wertebereiche der Attribute sind sowohl Daten- als auch Objektsorten. In einer Entwurfssprache für konzeptionelle Schemata sollten zusätzlich abkürzende Notationen und weitere semantische Konzepte (wie Generalisierung, Spezialisierung, Beziehungen zwischen Objektklassen etc.) zur Verfügung gestellt werden [MDL87, HNSE87], deren Semantik aber durch den hier vorgestellten Ansatz ausdrückbar ist [ESS88].

Da der konzeptionelle Entwurf die Diskussionsgrundlage zwischen den Anwendern, die die Anforderungsanalyse erstellen, und den Datenbankspezialisten ist, sollte die konzeptionelle Entwurfssprache für die Objektschicht eine graphische Darstellung der modellierten Objekte ermöglichen, z.B. in Form eines (erweiterten) Entity-Relationship-Diagramms wie in [HNSE87, SNHE87] vorgeschlagen.

Eine an die Spezifikationssprache für Datentypen angelehnte Spezifikation der Objektklasse `person` ist in Abbildung 10 gegeben. Die Beispielspezifikation besteht neben der Angabe des Objektsortennamen aus der Angabe der informationstragenden Attributfunktionen, z.B. dem Geburtsdatum oder der beschäftigenden Firma, und aus in Prädikatenlogik formulierten statischen Integritätsbedingungen ("Constraints").

In der Beispielspezifikation werden alle Eigenschaften einer Person durch die Werte von Funktionen ausgedrückt. Auf die formale Semantik einer derartigen Spezifikation wird in Abschnitt 5.3 eingegangen.

5.2.3 Entwicklungsschicht

In der *Entwicklungsschicht* werden die erlaubten Entwicklungen der gespeicherten Informationen unabhängig von konkreten Änderungsaktionen durch *dynamische Integritätsbedingungen* beschrieben. Ein Beispiel für eine dynamische Integritätsbedingung ist die Forderung, daß sich der Familienstand einer Person direkt nur von 'ledig' nach 'verheiratet' verändern kann und danach niemals wieder 'ledig' werden kann. Diese Integritätsbedingung ist zum Teil in Abbildung 11 spezifiziert. Eine dynamische Integritätsbedingung muß von allen in der Aktionsschicht spezifizierten Änderungsaktionen respektiert werden. In Abbildung 11 ist zusätzlich eine etwas komplexere dynamische Integritätsbedingung angegeben, die aussagt, daß das Gehalt einer Person in derselben Firma nur steigen darf — auch wenn die betreffende Person zwischenzeitig die Firma verlassen hatte.

Die Entwicklungsschicht ist semantisch eindeutig von der Objektschicht getrennt, da ihr als semantische Modellbildung Folgen von in der Objektschicht beschriebenen Datenbankzuständen und als Spezifikationskalkül temporale Logik zugrundeliegen. Um einen strukturierten Entwurf zu ermöglichen, sollten dynamische Integritätsbedingungen ebenfalls so weit möglich objektklassenlokal angegeben werden [SSE87].

Dynamische Integritätsbedingungen sind beliebige Formeln der finit-linearen temporalen Logik ohne globale oder Reduktquantifizierung über der durch die Daten- (Sorten und Operationen) und Objektschicht (Objektsorten und Attributfunktionen) festgelegten Signatur. Eine Erweiterung um Reduktquantifizierung bzw. erweiterte Reduktquantifizierung ist möglich und semantisch abgesichert, deren Berücksichtigung in späteren Entwurfsphasen aber bisher nicht untersucht worden.

5.2.4 Aktionsschicht

Auf der *Aktionsschicht* werden die Änderungstransaktionen beschrieben, die anderen Systemkomponenten als Schnittstelle zur Manipulation des Datenbankinhalts zur Verfügung gestellt werden und die die Konsistenz der gespeicherten Informationen bewahren müssen. Auf die Aktionsschicht wird in dieser Arbeit nicht näher eingegangen, da hier die deskriptive Entwicklungsschicht im Mittelpunkt steht.

Als Spezifikationssprache für die Aktionsschicht wird eine Spezifikation mit Vor- und Nachbedingungen zusammen mit einer Anwendbarkeitsbedingung vorgeschlagen [Li87]. Diese Wahl ist dadurch motiviert, daß derartige Spezifikationen mit der vorgestellten temporalen Logik verträglich sind (siehe Abschnitt 5.3) und diese Spezifikationstechnik im Softwareentwurf verbreitet ist. Eine Beispielspezifikation ist in Abbildung 12 wiedergegeben.

```
ACTIONS OBJECTTYPE person;
ACTION heirat(n :text, g :datum);
VARIABLES p :person;
ON (EXISTS (person p)(name(p)=n AND gebdat(p)=g));
PRE name(p)=n AND gebdat(p)=g AND NOT status(p)=verheiratet;
POST status(p)=verheiratet.
```

Abbildung 12: Beispiel einer Aktionsspezifikation

Um eine formale Semantik mit eindeutigem Änderungseffekt für eine Aktionenspezifikation zu erreichen, müßte zusätzlich eine *Rahmenregel* angegeben werden, die gewährleistet, daß die Wirkung der Aktion *nur* die explizit aufgeführten Änderungen umfaßt [Li87]. Im Beispiel der Aktion '`heirat`' soll nur der Familienstand eine neuen Wert erhalten, alle anderen Informationen in der Datenbank dürfen nicht verändert werden.

Für die Aktionsschicht können als Spezifikationskalkül auch andere Ansätze benutzt werden, z.B. erweiterte Petrinetzverfahren (u.a. in [RD82, Ce83, Rei85, SK85], eine Übersicht kann in [Schm87] gefunden werden), die zusätzlich eine graphische Darstellung der Aktionsspezifikation ermöglichen, oder andere modale Kalküle [GMS83, KMS85].

Die Spezifikation der Aktionen kann in zwei Schichten aufgeteilt werden, in die Spezifikation von *Elementaraktionen* und die Spezifikation *beliebiger Anwendungstransaktionen* [HNSE87, SNHE87]. Die Elementaraktionen realisieren elementare Änderungen, wie Einfügen eines Objektes oder Ändern des Wertes einer Attributfunktion. Sie betreffen jeweils nur Einzelobjekte und müssen nur die modellinhärenten und Einzelobjekte betreffende Integritätsbedingungen respektieren. Die angebene Beispielsaktion `heirat` ist eine derartige Elementaraktion.

Beliebige *Anwendungstransaktionen* können ganze Objektklassen manipulieren und müssen alle statischen und dynamischen Integritätsforderungen respektieren. Ein Beispiel für die angebene Objektspezifikation ist die Festlegung neuer Lohnsteuersätze für *alle* gespeicherten Personen. Besonders in Nichtstandard-Datenbank-Anwendungen z.B. im Ingenieurwesen erfordern komplexe Integritätsbedingungen zusammen mit komplexen Änderungstransaktionen diese Aufteilung.

Für komplexe Anwendungstransaktionen werden Spezifikationen mit Vor– und Nachbedingungen schnell unübersichtlich. Wichtige Aspekte wie nebenläufige Abläufe, Verwendung bereits spezifizierter Aktionen und Ablaufpläne können dabei nicht berücksichtigt werden. Um diese Aspekte zu berücksichtigen, ist die Einführung einer weiteren Spezifikationsschicht, der *Prozeßschicht*, zu überlegen, die auf den anderen Schichten aufbaut [SNHE87, SFSE88]. Als semantische Grundlage kommen dafür Modelle für nebenläufige Prozesse in Frage (z.B. [Ho85]).

Die Unterscheidung zwischen einfachen Elementaraktionen und beliebigen Änderungstransaktionen ist — neben dem Komplexitätsunterschied — auch in den späteren Entwurfsphasen begründet. Elementaroperationen werden in die Datenbank-Komponente des Gesamtsystems integriert und können z.T. automatisch aus der Spezifikation erstellt werden. Die Berücksichtigung allgemeiner komplexer Integritätsbedingungen würde zu einem Aufblähen der Parameterliste führen und für den Benutzer unvermutete Seiteneffekte haben.

Beliebige Anwendungstransaktionen werden in der Regel in anderen Systemkomponenten realisiert und benutzen die Elementaroperationen als Zugriffsschnittstelle auf die Datenbank. Die Respektierung komplexer statischer und dynamischer Integritätsbedingungen durch die Anwendungstransaktionen muß dann entweder in deren Entwurf berücksichtigt werden oder durch eine Komponente des DBMS, dem *Integritätsmonitor*, gewährleistet werden (Kapitel 6).

5.2.5 Objektorientierter Entwurf

Um einen *strukturierten Entwurf* der Datenbank-Anwendung zu ermöglichen, wird die Spezifikation der drei obersten Schichten so weit möglich *objektorientiert* vorgenommen. Für eine Objektklasse werden die Angaben der Objektschicht, die dynamischen Integritätsbedingungen und die diese Objektklasse betreffenden elementaren Änderungstransaktionen in einem Spezifikationsmodul zusammengefaßt. Besondere Sprachkonzepte ermöglichen den Zugriff auf Datentypmodule und andere Objektmodule, um deren Sorten als Wertebereiche von Attributen bzw. Variablen in Integritätsbedingungen benutzen zu können und deren Daten- und Objektoperationen in Integritätsbedingungen und Aktionenspezifikationen verwenden zu können.

Da Änderungstransaktionen in der Regel mehrere Objekte verschiedener Klassen verändern, muß die Interaktion zwischen den Aktionsspezifikationen verschiedener Objektmodule explizit spezifiziert werden.

Ein Beispiel für eine strukturierte Spezifikationssprache, die diese Anforderungen an einen objektorientierten Entwurf erfüllt, ist die Spezifikationssprache OBLOG, die in [SSE87] vorgestellt wurde. Die Interaktion zwischen den Aktionsspezifikationen verschiedener Objektmodule wird in OBLOG durch Interaktionsgleichungen modelliert, die Aktionen verschiedener Objektmodule miteinander in Beziehung setzen.

Neben der Strukturierung des Gesamtentwurfs durch eine Modularisierung in lokal auf Konsistenz zu prüfende Teilschemata ermöglicht die Aufteilung in Objektmodule die abstrakte Modellierung von *konzeptionell unabhängigen* Informationen und Aktionen. Alle Änderungsaktionen, die nicht explizit auf ein bestimmtes Objekt zugreifen, haben keinen Einfluß auf die Informationen dieses Objektes. Die konzeptionelle Unabhängigkeit ermöglicht eine implementierungsunabhängige Behandlung von Problemstellungen wie der Äquivalenz von verschiedenen Sequenzen von Aktionen.

Um eine konzeptionelle Unabhängigkeit der Entwicklungsspezifikation zu erreichen, muß die Klasse der als dynamische Integritätsbedingungen zugelassenen temporalen Formeln weiter eingeschränkt werden. Aktionen, die unabhängige Objekte / Objektklassen betreffen, sollen nur Einfluß auf die Gültigkeit der dynamischen Integritätsbedingungen der explizit aufgeführten Objekte haben.

Die konzeptionelle Unabhängigkeit von dynamischen Integritätsbedingungen führt zur Notation von *iterationsinvarianten* temporalen Formeln [Li87], deren Gültigkeit unabhängig von Iteration einzelner Zustände in einer Zustandsfolge ist.

Definition 5.2.1 Eine temporale Formel heißt *iterationsinvariant*, wenn ihre Gültigkeit in einer Zustandsfolge nicht von der endlichen Iteration einzelner Zustände der Zustandsfolge abhängt. Die formale Definition kann in [Li87] gefunden werden. □

Die Iterationsinvarianz kann insbesondere durch den Gebrauch von Nextoperatoren verletzt werden, während der ausschließliche Einsatz der über ganze Zustandsfolgen quantifizierenden temporalen Operatoren **always**, **sometime**, **before** und **until** die Iterationsinvarianz garantiert [Li87, Hü88].

Die Eigenschaft der Iterationsinvarianz kann auch anhand der Struktur des aus einer Formel konstruierten Prüfschemas bestimmt werden. Hierzu muß überprüft werden, ob für jeden Knoten k, der Zielknoten mindestens einer Kante ist, eine Kante (k, k) existiert, deren Beschriftung von jeder auf diesen Knoten führenden Kante impliziert wird. Derartige Transitionsgraphen werden ebenfalls als iterationsinvariant bezeichnet.

5.3 Semantische Modellbildung

Die Daten- und Objektschicht einer Spezifikation definieren eine Signatur $\Sigma = \Sigma_D + \Sigma_O$, die aus der Vereinigung der Signaturen der Einzelschichten besteht. Die Ausprägungen einer Datenbank oder *Datenbankzustände* sind Σ–Strukturen für diese Signatur, die die Gleichungen E_D (E von englisch *Equations*) der Datentypspezifikation und die statischen Integritätsbedingungen C_O (C von englisch *Constraints*) der Objekttypspezifikation erfüllen.

Die Entwicklungsschicht beschreibt die erlaubten *Datenbankentwicklungen*, die durch lineare temporale Strukturen zur Signatur Σ modelliert werden. Korrekte Datenbankentwicklungen bestehen aus erlaubten Datenbankzuständen und erfüllen die dynamischen Integritätsbedingungen der Entwicklungsschicht C_T. Die zeitunabhängige Identifikation der Objekte wird explizit durch die Objektschlüssel festgelegt.

In der Regel gibt es viele unterschiedliche Σ–Strukturen für eine gegebene Signatur und gegebene logische Axiome. Dasselbe gilt erst recht für die temporalen Strukturen. Es stellt sich die Frage, ob bestimmte dieser Strukturen als "Standardinterpretationen" ausgezeichnet werden können bzw. ob andere von vornherein ausgeschlossen werden können.

Aus der Anwendung für eine Datenbankspezifikation sind eine Reihe impliziter Einschränkungen vorgegeben, die bei der Konstruktion von semantischen Modellen berücksichtigt werden müssen. So sollte für alle Datenbankzustände die Interpretation der Datenschicht fest sein — der Wert eines Datentermes $5 + 3$ und dessen Eigenschaften sind weder von der konkreten Datenbankausprägung noch vom Zeitpunkt der Betrachtung abhängig. Die Schlüsseleigenschaft der eindeutigen Objektidentifikation muß ebenfalls gewährleistet sein.

Verschiedene Konstruktionsmethoden für Datenbankzustände, die diese Eigenschaften garantieren, wurden in [Eh86, EDG86] und [SSE87, ESS88] vorgestellt. In dieser Arbeit werden im folgenden die diesen Ansätzen zugrundeliegenden theoretischen Überlegungen kurz vorgestellt, wobei der Schwerpunkt auf dem Übergang von Datenbankzuständen zu Datenbankentwicklungen liegt. Die formale Semantik der Modellkonstruktion kann aus der angegebenen Literatur entnommen werden.

Für die Datentypen wird eine feste Semantik angenommen, die z.B. durch die initiale Algebra für die Datentypspezifikation [EM85] festgelegt sein kann. Diese Semantik ist eine Σ–Struktur $A_D \,\epsilon\, \Im_{\Sigma_D}$, die die Gleichungen in E_D erfüllt :

$$A_D \models e \quad \text{für alle} \quad e \,\epsilon\, E_D$$

Ein Datenbankzustand[11] σ ist eine Σ–Struktur $\sigma \,\epsilon\, \Im_\Sigma$ zur Signatur $\Sigma = \Sigma_D + \Sigma_O$, der die *Datentypalgebra* A_D als Teilstruktur enthält. In der Theorie der algebraischen Datentypspezifikation wird diese Aussage durch Bildung einer Reduktalgebra zur Signatur Σ_D formuliert :

$$A_D = \sigma|\Sigma_D$$

Die Semantik der Datentypen ist konstant während der Laufzeit der Datenbank-Anwendung. Dies entspricht der Unterscheidung zwischen Daten- und Objekttypen,

[11]Für die Datenbankzustände und Datenbankentwicklungen werden die Notationen für Welten und Folgen der erweiterten MP–Logik gewählt, obwohl in diesem Kapitel weitere Einschränkungen für diese Interpretationsstrukturen eingeführt werden.

in der die Datentypen die Struktur der elementaren Informationseinheiten bestimmen, die unabhängig von der aktuell gespeicherten Information ist. Elemente aus der Trägermenge einer Datensorte sind in der Datenbank nur als Attributwert von Objekten gespeichert und die Auswertung der Datenoperationen soll während der Implementierung fest in das DBMS integriert werden.

Zur Festlegung der Interpretationsstrukturen für die Objektschicht wird die Spezifikation dieser Schicht in zwei aufeinanderaufbauende Konstruktionsschritte aufgeteilt. In einem ersten Schritt wird aus der Angabe der Sorten und der Schlüsselattribute das *Universum der möglichen Objekte* konstruiert. Jedes Trägerelement dieses Universums charakterisiert ein eindeutig durch den Objektschlüssel bestimmtes Objekt einer Objektsorte, das in bestimmten Datenbankzuständen enthalten sein kann.

Die zustandsabhängige Information über ein Objekt wird durch die Werte der Nichtschlüsselattribute repräsentiert. Ein Datenbankzustand enthält als Träger der Objektsorten eine Teilstruktur des Universums möglicher Objekte, die sogenannten *aktuellen Objekte*, zusammen mit den zusätzlichen, zustandsabhängigen Attributwerten.

Beide Bestandteile der Objektspezifikation können mit statischen Integritätsbedingungen versehen sein. Integritätsbedingungen für die Schlüsselspezifikation schränken die möglichen Objekte ein, die anderen die erlaubten Datenbankzustände.

Die Objektsorten zusammen mit den Schlüsselattributen bilden die *Schlüsselsignatur* Σ_{KEY}. Das *Universum der möglichen Objekte* zur einer Objektsignatur enthält für eine Objektsorte intuitiv genau alle Objekte, die unterschiedliche Werte für die Schlüsselattribute haben. Für datenwertige Schlüsselattribute ist die Gleichheit der Werte durch die Datentypalgebra A_D definiert, für objektwertige Schlüsselattribute wird die Gleichheit der Objekte rekursiv durch die Gleichheit der Schlüsselattribute der Zielobjektsorte definiert.

Enthält die Schlüsselsignatur keine Zyklen in den Attributfunktionen, können die möglichen Objekte auf Kreuzprodukte von Datenwerten zurückgeführt werden und das Universum kann direkt durch eine initiale Konstruktion gebildet werden [SSE87]. Der allgemeine Fall mit Zyklen wurde in [Eh86, EDG86] untersucht und eine geeignete Universumskonstruktion vorgestellt.

Statische Integritätsbedingungen, die nur Schlüsselattribute betreffen, können zum Teil in der Universumskonstruktion berücksichtigt werden. In diesem Fall dürfen aber nicht beliebige prädikatenlogische Formeln betrachtet werden, damit die Universumskonstruktion eine eindeutige Algebra UNIV als Ergebnis hat [Eh86, EDG86]. Bei einer initialen Konstruktion ermöglicht z.B. die ausschließliche Verwendung von Gleichungen die Konstruktion eines eindeutig bestimmten Universums.

Die in einem Datenbankzustand enthaltenen *aktuellen Objekte* sind eine Teilmenge der möglichen Objekte, für die alle Werte der Schlüsselattribute ebenfalls aktuelle Objekte sind. Für datenwertige Schlüsselattribute gilt dies aufgrund der Tatsache, daß die Datentypalgebra in jedem Zustand enthalten ist. Für die Objektsorten gilt dieses genau dann, wenn die Objekte zusammen mit den Schlüsselattributfunktionen eine Unteralgebra von UNIV bilden. Für jeden Datenbankzustand σ muß also gelten :

$$\sigma \Big| \Sigma_{\mathrm{KEY}} \subseteq \mathrm{UNIV}$$

Zusätzlich wird noch gefordert, daß jeder Datenbankzustand σ alle statischen Integritätsbedingungen erfüllt :

$$\sigma \models \varphi \;\text{ für alle }\; \varphi \,\epsilon\, C_O$$

Mit diesen Konstruktionen sind die korrekten Datenbankzustände festgelegt.

Definition 5.3.1 Jeder *korrekter Datenbankzustand* σ muß

- die Datentypalgebra A_D als Unteralgebra enthalten,
- als aktuelle Objekte der Objektsorten eine wohldefinierte Untermenge der möglichen, durch die Schlüsselattribute eindeutig identifizierten, Objekte haben,
- und alle statischen Integritätsbedingungen erfüllen. □

In [SFSE88] wird die Schlüsselsignatur um sogenannte *konstante Attribute* angereichert, die während der Laufzeit ihren Wert nicht ändern dürfen und bei der Universumskonstruktion aufgenommen werden. Die Schlüsseleigenschaft wird bei der Universumskonstruktion dadurch berücksichtigt, daß die aus der Signatur mit zusätzlichen konstanten Attributen konstruierten Universen das gemäß obiger Methode zur Schlüsselsignatur konstruierte Universum UNIV als Teilstruktur enthalten müssen. Die Eigenschaft eines Attributs, konstant zu sein, kann allerdings auch durch geeignete dynamische Integritätsbedingungen ausgedrückt werden.

Die semantische Modellbildung für den dynamischen Anteil der Datenbankspezifikation, also für die Entwicklungs- und die Aktionsschicht, sind Folgen von korrekten Datenbankzuständen $\hat{\sigma} \,\epsilon\, \Im_{\Sigma}{}^{+}$. Jeder Zustand σ_i einer Folge muß die obigen drei Bedingungen für Datenbankzustände erfüllen. Eine derartige Folge heißt *Datenbankentwicklung zur Signatur* $\Sigma = \Sigma_D + \Sigma_O$.

Definition 5.3.2 Eine *korrekte Datenbankentwicklung* $\hat{\sigma}$ ist eine Folge von korrekten Datenbankzuständen, die alle dynamischen Integritätsbedingungen C_T der Entwicklungsschicht erfüllt :

$$\hat{\sigma} \models \varphi \;\text{ für alle }\; \varphi \,\epsilon\, C_T$$

□

Durch die Verwendung der finit-linearen temporalen Logik ist die gesonderte Behandlung des Löschens bzw. Einfügens von Datenbankobjekten nicht nötig, da deren Semantik durch die Definition der belegungsunabhängigen Gültigkeit über Reduktquantifizierung festgelegt ist.

Als Interpretationsstrukturen für Datenbankentwicklungen werden unendliche Zustandsfolgen genommen, da die Spezifikation der Datenbankdynamik zwar das Löschen von Objekten beinhaltet, aber *nicht* das endgültige Beenden der Datenbank-Anwendung. Endliche Folgen von Datenbankzuständen tauchen allerdings als Interpretationsstrukturen bei der Semantikdefinition von dynamischen Integritätsbedingungen mit freien Variablen auf.

In dem vorgestellten funktionalen Objektmodell müssen nur die explizit angegebenen dynamischen Integritätsbedingungen betrachtet werden. In komplexeren Objektmodellen müssen möglicherweise analog zu den modellinhärenten statischen Integritätsbedingungen auch *modellinhärente dynamische Integritätsbedingungen* beachtet werden [SNHE87]. Ein Beispiel für inhärente dynamische Integritätsbedingungen ist die erwähnte Festlegung von konstanten Attributen oder von Alternativschlüsseln.

Eine Datenbankentwicklung heißt *ausführbar*, wenn jedem Zustandsübergang $\sigma_i \rightarrow \sigma_{i+1}$ eine Aktion a zugeordnet werden kann, deren Spezifikation von diesem Übergang erfüllt wird. Eine Aktion a wird durch eine Anwendbarkeitsbedingung ω, eine Vorbedingung φ und eine Nachbedingung ψ spezifiziert (alle drei sind prädikatenlogische Formeln). Die Spezifikation wird als $\mathbf{on}\,\omega : \{\varphi\}a\{\psi\}$ notiert. Aktionen können auch Parameter haben, auf deren Berücksichtigung wird aber im folgenden verzichtet.

Sei die Aktion a durch die Spezifikation $\mathbf{on}\,\omega : \{\varphi\}a\{\psi\}$ spezifiziert. Ein Zustandspaar (σ, σ') erfüllt diese Spezifikation genau dann, wenn erstens ω in σ gültig ist und daß, wenn $\sigma \models \varphi$ gilt, dann auch $\sigma' \models \psi$ gelten muß. Der Zustandsübergang heißt dann auch *durch a ausführbar*. Unter Benutzung der temporalen Logik kann dieser Zusammenhang auf Folgen von Datenbankzuständen erweitert werden.

Definition 5.3.3 Eine Folge von Datenbankzuständen $\hat{\sigma}$ heißt *ausführbar*, wenn für jeden Datenbankzustand σ_i in dieser Folge eine Aktion a mit der Spezifikation $\mathbf{on}\,\omega : \{\varphi\}a\{\psi\}$ existiert, so daß

$$\hat{\sigma}_i \models \omega \wedge (\varphi \Rightarrow \mathbf{allnext}\,\psi).$$

□

Die Zustandsfolgen, die als erlaubte Ausprägungen eines konzeptionellen Datenbankschemas auftreten dürfen, müssen beide Bestandteile der Dynamikspezifikation erfüllen :

Definition 5.3.4 Die Semantik einer vierschichtigen Spezifikation ist durch die Menge der *ausführbaren korrekten Datenbankentwicklungen* festgelegt. $\square$

Um auch für die Aktionsspezifikation eine umfassendere mathematische Grundlage für weitergehende formale Untersuchungen auf Konsistenz, Einhaltung von Lebendigkeitsforderungen etc. zu haben, die eine direkte Erweiterung der Spezifikationskalküle der unteren drei Schichten ist, kann die Spezifikation der Aktionen durch eine *Aktionenlogik* ausgedrückt werden. Eine Aktionenlogik enthält für jede Aktion a einen Modalquantor $[a]$ (siehe z.B. [GMS83, KMS85, SNHE87]). Analog zu den beiden Nextoperatoren können auch zwei Modalquantoren für jede Aktion eingeführt werden, um zwischen terminierenden und möglicherweise nichtterminierenden Aktionsausführungen zu unterscheiden.

Die Spezifikation durch Vor- und Nachbedingungen ohne Anwendbarkeitsbedingung (ohne **on** ω– Formelbestandteil) würde in der Aktionenlogik in der Form $\varphi \Rightarrow [a]\psi$ ausgedrückt. Der temporale Nextoperator muß in dieser Aktionenlogik für korrekte, nichtabbrechende ausführbare Datenbankentwicklungen äquivalent zur Quantifizierung über alle Aktionen sein :

$$\hat{\sigma} \models \mathbf{allnext}\,\varphi \ \text{ gdw. } \quad \text{für alle } a \text{ gilt : } \ \hat{\sigma} \models [a]\varphi$$

Wie bereits erwähnt, muß der Effekt einer Aktion durch die zusätzliche Angabe einer Rahmenregel eingeschränkt werden, um die Aktionswirkung auf die durch die Nachbedingung explizit festgelegten Änderungen zu beschränken. Diese Rahmenregel muß ebenfalls von dem Zustandsübergang erfüllt werden. Auf die Konstruktion einer expliziten Rahmenregel, die eine minimale Änderung der Zustände bewirkt, wird in [Li87] eingegangen.

Die Semantik des dynamischen Anteils der mehrschichtigen Schemaspezifikation läßt sich wie folgt charakterisieren :

Eine zeitliche Abfolge von Datenbankzuständen muß

- zu jedem Zeitpunkt einen korrekten Datenbankzustand als Weltinterpretation haben,
- die dynamischen Integritätsbedingungen C_T erfüllen
- und durch die Aktionen ausführbar sein.

Modelle für die Gesamtspezifikationen sind die *ausführbaren korrekten Datenbankentwicklungen*. Als Ziele für einen methodischen Entwurf sind die beiden Extreme bezüglich der Einschränkungen der Datenbankdynamik interessant :

1. Erlaubt sind alle typischerweise durch ein DBMS zur Verfügung gestellten Änderungsaktionen, die Einschränkungen werden ausschließlich durch dynamische Integritätsbedingungen formuliert.

   Dieses Entwurfsziel ist von Bedeutung, falls ein DBMS mit integriertem Integritätsmonitor zur Verfügung steht [Hü88]. Dieser Ansatz ermöglicht die flexible Handhabung von Änderungsaktionen, die erst zur Systemlaufzeit ad hoc formuliert werden, und entspricht dem verbreiteten Entwurfsansatz, die eigentliche Datenbank und die Dynamik der Änderungstransaktionen getrennt zu entwerfen.

2. Die Menge der dynamischen Integritätsbedingungen ist leer, alle Einschränkungen werden durch die Aktionsspezifikation berücksichtigt.

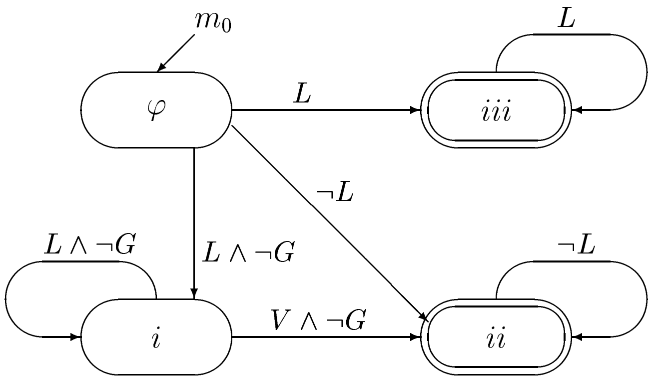
   Hierbei wird kein Integritätsmonitor benötigt. Da bis jetzt keine DBMS mit Integritätsmonitor für dynamische Integritätsbedingungen implementiert sind, ist dieser Ansatz im allgemeinen vorzuziehen. Allerdings müssen alle Änderungstransaktionen zur Entwurfszeit festgelegt werden bzw. neue Änderungstransaktionen den Entwurfsprozeß vollständig neu durchlaufen. Methodische Vorschläge für diesen Ansatz werden in [Li87] vorgestellt.

Diese beiden entgegengesetzten Entwurfsziele spiegeln sich in den beiden Implementierungsvorschlägen für die Überwachung von dynamischen Integitätsbedingungen in Datenbank-Anwendungen wieder, der Überwachung durch einen Integritätsmonitor ohne Einschränkungen an die Transaktionen und der Überwachung durch integritätsrespektierende Transaktionen, die beide in Kapitel 6 vorgestellt werden.

5.4 Entwurf der Entwicklungsschicht

Der Entwurf eines konzeptionellen Datenbankschemas ist eine komplexe Aufgabe, für die der Einsatz von methodischen Entwurfstechniken und entwurfsunterstützender Werkzeuge wünschenswert sind. In diesem Abschnitt werden Methoden zur Unterstützung des Entwurfs der dynamischen Integritätsbedingungen vorgestellt, die sowohl als Vorschläge für einen methodischen Entwurfsprozeß und für Analysetechniken als auch als Grundlage für Werkzeuge einer integrierten Datenbankentwurfsumgebung [EHNSE88] geeignet sind.

Auch wenn die dynamischen Integritätsbedingungen lokal zu Objektklassen angegeben werden, ist die Semantik einer Entwicklungsschicht selbst für Entwurfsspezialisten nicht mehr intuitiv überschaubar. Einfache, nicht geschachtelte, temporale Formeln teilen die Objekte typischerweise in zwei bis drei Äquivalenzklassen bezüglich offener



Abkürzungen :

L	$=$	`status(p) = ledig`
V	$=$	`status(p) = verheiratet`
G	$=$	`status(p) IN (geschieden,verwitwet)`
φ	$=$	`familienstandwechsel` (Integritätsbedingung aus Abb. 11, Seite 84)
i	$=$	$\varphi \wedge (L\ \mathbf{before}\ G)$
ii	$=$	$\varphi \wedge (\mathbf{always}\ \neg L)$
iii	$=$	$\varphi \wedge (\mathbf{always}\ L)$

Abbildung 13: Graphische Darstellung der Entwicklungsspezifikation

zu beweisender Formelteile ein. Realistische Integritätsforderungen hingegen bestehen aus aussagenlogischen Verknüpfungen mehrerer temporaler Formeln, die oft noch zusätzlich durch temporale Operatoren gebunden sind (siehe [Li87] für ein größeres Anwendungsbeispiel).

Unabhängig von Konsistenzfragen bieten die Prüfschemata für die dynamischen Integritätsbedingungen die Möglichkeit der graphischen Darstellung der durch die Formel spezifizierten Objektdynamik. Dies ermöglicht insbesondere für geschachtelte temporale Operatoren eine schnelle Kontrolle, ob die temporale Formel auch wirklich die beabsichtige Einschränkung ausdrückt.

Beispiel 5.4.1 In Abbildung 13 ist das Prüfschema für die Integritätsbedingung `familienstand` aus der Abbildung 11 graphisch dargestellt. Erst aus der graphischen Darstellung wird die Relevanz des Formelbestandteils $L \Rightarrow (\ldots \vee \mathbf{always}\, L)$ deutlich. Würde die Teilformel "$\vee\, \mathbf{always}\, L$", die den Knoten iii des Transitionsgraphen verursacht, in der Gesamtintegritätsbedingung fehlen, können Personen erst dann gelöscht werden, nachdem sie den Familienstand von 'ledig' nach 'verheiratet' gewechselt haben — eine Einschränkung, deren Spezifikation sicher nicht beabsichtigt wäre. □

Die graphische Darstellung ermöglicht in diesem Beispiel eine schnelle Kontrolle der Entwicklungsspezifikation. Sinnvoll ist hierbei nicht nur die Darstellung der Originalprüfschemata, sondern auch die der reduzierten Prüfschemata sowie die Darstellung der zugehörigen deterministischen Transitionsgraphen, da der volle Umfang der nichtdeterministischen Abarbeitung oft schwierig zu überblicken ist. Ein Vorschlag für ein auf der graphischen Darstellung von Transitionsgraphen basierendes Entwurfswerkzeug kann in [Me87] gefunden werden.

Für ein konkretes Datenbankobjekt müssen für Konsistenzüberprüfungen nicht nur die Integritätsbedingungen, die der Objektklasse explizit zugeordnet sind, betrachtet werden, sondern auch diejenigen Integritätsbedingungen, die auf dieses Objekt über objektwertige Attributfunktionen zugreifen, und die impliziten modellinhärenten Integritätsbedingungen (vergl. Kapitel 5.2).

Definition 5.4.2 Die für die Konsistenzüberprüfung der Entwicklungsspezifikation für eine Objektklasse O *relevanten dynamischen Integritätsbedingungen* $IB(O)$ sind

- alle expliziten und modellinhärenten dynamischen Integritätsbedingungen, in denen eine Variable der Objektsorte frei oder gebunden vorkommt,
- und die Formeln **always** φ für alle expliziten und modellinhärenten statischen Integritätsbedingungen φ, in denen eine Variable der Objektsorte frei oder gebunden vorkommt. □

Für die Konsistenzüberprüfung müssen auch die modellinhärenten statischen und dynamischen Integritätsbedingungen berücksichtigt werden, die bei der semantischen Modellbildung benutzt wurden, so z.B. die zeitpunktunabhängige Semantik der Datenschicht und der zeitpunktunabhängige Identifikationsmechanismus durch Schlüsselattribute.

Definition 5.4.3 Die Entwicklungsschicht heißt *konsistent* für eine Objektklasse, wenn der reduzierte Transitionsgraph für die temporale Formel φ, die die Konjunktion aller für die Objektklasse relevanten dynamischen Integritätsbedingungen ist, nicht leer ist.

Bei einer objektorientierten Entwurfsmethodik muß dieser Transitionsgraph zusätzlich iterationsinvariant sein. □

In der praktischen Konsistenzüberprüfung brauchen nicht alle relevanten dynamischen Integritätsbedingungen in die Konstruktion der Transitionsgraphen eingebracht werden. Im allgemeinen ist diese Vorgehensweise allein deshalb nicht sinnvoll, da alle Datentypaxiome und statischen Integritätsbedingungen zu einer temporalen Formel umgeformt werden müßten.

Algorithmus 5.4.4 Eine dynamische Integritätsbedingung φ heißt *erfüllbar*, wenn folgender Algorithmus nicht den leeren Transitionsgraph konstruiert :

1. Bestimme alle expliziten und modellinhärenten dynamischen Integritätsbedingungen, für die gilt :
   - Sie betreffen — direkt oder über Attributfunktionen — dieselben Objektklassen wie φ.
   - Sie betreffen — direkt oder indirekt über weitere statische oder dynamische Integritätsbedingungen — dieselben Objekteigenschaften (Attribute) wie φ.
2. Konstruiere den Transitionsgraph für die Konjunktion dieser Formeln.
3. Reduziere den Transitionsgraph unter Berücksichtigung der statischen Integritätsbedingungen und Datentypaxiome. □

Dieser Algorithmus ermöglicht z.B. eine getrennte Analyse der beiden dynamischen Integritätsbedingungen in Abbildung 11 auf Seite 84, da diese ausschließlich auf unterschiedliche Objekteigenschaften Bezug nehmen, deren mögliche Wertekombinationen nicht durch andere, statische oder dynamische, Integritätsbedingungen eingeschränkt sind.

Wird der Algorithmus zur Überprüfung auf Erfüllbarkeit als methodische Vorgehensweise durch einen Entwurfsspezialisten durchgeführt, kann die Menge der für diese Konsistenzüberprüfung relevanten Integritätsbedingungen anhand weiterer Entwurfsinformationen zusätzlich eingeschränkt werden.

Die Ergebnisse der Konsistenzüberprüfung — reduzierte Transitionsgraphen, die alle Integritätsforderungen berücksichtigen — sind ein wichtiges Hilfsmittel für eine Bewertung des Entwurfs der Datenbankdynamik. Sie sind *Musterlebensläufe* für die Objekte (bzw. Objektkombinationen) einer Objektklasse.

In ein entwurfsunterstützendes System sollte eine graphische Darstellung der Ergebnisse der Reduktion als Komponente aufgenommen werden [Me87, SNHE87]. Aus diesen graphisch dargestellten Musterlebensläufen lassen sich neben der Erfüllbarkeit der Spezifikation oft direkt weitere wichtige Eigenschaften ablesen, z.B. ob ein Attributwert nicht geändert werden kann oder ob bestimmte Attribute aufgrund der Entwicklungsspezifikation nur vorbestimmte Wertfolgen annehmen können.

Diese Eigenschaft ist deshalb von besonderer Wichtigkeit, da derartige Objekteigenschaften für komplexe Formelmengen nur durch Beweisverfahren für temporale Logiken [MP81, CF84, AM85, Ve85] gezeigt werden können. Eine intuitive Vorstellung der

Semantik einer komplexen temporalen Formel ist oft schon bei kleinen Spezifikationen nicht mehr möglich. Um eine (nichttriviale) Eigenschaft einer Spezifkation aber beweisen zu können, muß zuerst diese Objekteigenschaft bekannt und formal formuliert sein. In solchen Fällen ist eine intuitiv überschaubare graphische Darstellung der möglichen Objektlebensläufe sicher von Vorteil.

5.5 Konsistenz der Aktions- zur Entwicklungsschicht

Die Semantik einer vierschichtigen Spezifikation sind die ausführbaren korrekten Datenbankentwicklungen, d.h. diejenigen Folgen von Datenbankzuständen, die sowohl die Entwicklungsschichtspezifikation als auch die der Aktionen erfüllen. Die beiden Spezifikationsschichten sind in der vorgestellten Spezifikationsmethode unabhängig voneinander — sie können sich also widersprechen, obwohl die Spezifikationen selber in sich konsistent sind.[12]

Verfahren zur Überprüfung einer Aktionsspezifikation auf Verträglichkeit mit den statischen und dynamischen Integritätsforderungen sind erstmals in [Ku84, Ku85] vorgestellt worden. Dort wird aus einer Aktionsspezifikation mit Vor- und Nachbedingungen ein sogenanntes *Transitionsdiagramm* erstellt, das die Form eines einfachen Petrinetzes (ohne Nebenläufigkeit) mit objektwertigen Markierungen hat. Die Berechnungsfolgen im Transitionsdiagramm werden anschließend auf Verträglichkeit mit den dynamischen Integritätsbedingungen untersucht.

Ein implizites Verfahren zur Untersuchung einer Aktionsspezifikation auf Konsistenz zur Entwicklungsschicht ist in [Li87] beschrieben. Die Vor- und Nachbedingungen von Aktionen werden schrittweise dahingehend verfeinert, daß sie die dynamischen Integritätsbedingungen einhalten (vergl. Abschnitt 6.4). Eine den Integritätsforderungen widersprechende Aktion kann daran erkannt werden, daß nach Vereinfachung der verfeinerten Spezifikation die Aktion in keinem Datenbankzustand anwendbar ist.

In diesem Abschnitt werden Notationen und Untersuchungsverfahren vorgestellt, die Aussagen über die "Qualität" der aus der Entwicklungs- und Aktionsschicht bestehenden Dynamikspezifikation ermöglichen. Da Spezifikationsmethodik und formale Semantik der Aktionenspezifikation in dieser Arbeit nicht vertieft wurden, können formale Kriterien für die Konsistenz der Dynamikspezifikation nur angedeutet werden.

Da sich die beiden unabhängigen Spezifikationen der Entwicklungen und der Aktionen widersprechen können, sollten sie auf Konsistenz zueinander untersucht werden. Hierbei sind vor allem drei Fragen von Interesse, die für die Konsistenz des Gesamtentwurfs relevant sind :

[12] Die Überprüfung der Aktionsschicht auf Konsistenz ist nicht Inhalt dieser Arbeit.

1. Erzwingt bereits die Aktionenspezifikation die dynamischen Integritätsbedingungen ? Oder widerspricht sie den Forderungen der Entwicklungsschicht ? Existiert insbesondere für jede zu erfüllende temporale Existenzforderung eine endliche ausführbare Zustandsfolge, die diese Existenzforderung erfüllt ?

2. Sind alle spezifizierten Aktionen irgendwann anwendbar oder widerspricht die Spezifikation einiger Aktionen den dynamischen Integritätsbedingungen ?

3. Existiert überhaupt eine korrekte ausführbare Datenbankentwicklung für die Gesamtspezifikation ?

Die erste Frage läßt sich durch eine *aktionenabhängige* Reduktion von Transitionsgraphen entscheiden.

Definition 5.5.1 Für eine Objektklasse O sei die Menge $\mathcal{T}(O)$ der die Objektklasse O betreffenden Transaktionen mit Spezifikationen der Form $\mathbf{on}\,\omega : \{\pi\}a\{\psi\}$ und die Menge $IB(O)$ der für die Objektklasse O relevanten dynamischen Integritätsbedingungen (siehe Kap. 5.2 bis 5.4) festgelegt. Die Menge $\mathcal{T}(O)$wird als *Aktionenspezifikation* für O bezeichnet. Alle Spezifikationen von Aktionen in $\mathcal{T}(O)$ seien erfüllbar.

Ferner sei die nichtleere Teilmenge $\mathcal{T}_I(O) \subset \mathcal{T}(O)$ der *Einfügeaktionen* und die Teilmenge $\mathcal{T}_F(O) \subset \mathcal{T}(O)$ der *Löschaktionen* festgelegt.[13] □

Definition 5.5.2 Eine Aktionenspezifikation $\mathcal{T}(O)$ *erfüllt* eine dynamische Integritätsbedingung $\varphi \,\epsilon\, IB(O)$, falls folgender Algorithmus nicht den leeren Transitionsgraphen erzeugt :

1. Konstruiere den Transitionsgraph T_φ für φ.

2. Entferne alle Kanten e aus T_φ, die von einem initialen Knoten $k \,\epsilon\, m_0$ ausgehen und für die keine Einfügeaktion $a \,\epsilon\, \mathcal{T}_I(O)$ mit der Spezifikation $\mathbf{on}\,\omega : \{\pi\}a\{\psi\}$ existiert, deren Nachbedingung ψ die Beschriftung $\eta(e)$ impliziert.

3. Streiche alle nicht an einem Knoten der Anfangsmarkierung startenden Kanten $e = (s_i, s_j)$ aus T_φ, für die keine Aktion $a \,\epsilon\, \mathcal{T}(O)$ mit der Spezifikation $\mathbf{on}\,\omega :$ $\{\pi\}a\{\psi\}$ existiert, so daß gilt :

[13] Bei der Beschränkung auf Elementaroperationen oder der Verwendung einer strukturierten, objektorientierten Spezifikationssprache wie OBLOG [SSE87] sind die Komponenten $\mathcal{T}(O)$, $\mathcal{T}_I(O)$ und $\mathcal{T}_F(O)$ syntaktisch festgelegt. Im allgemeinen Fall müssen sie analog zur Festlegung der relevanten Integritätsbedingungen durch ein Analyseverfahren bestimmt werden.

- Die Disjunktion der Kantenbeschriftungen der auf den Knoten s_i führenden Kanten widerspricht nicht $\omega \wedge \pi$.
- Die Nachbedingung ψ impliziert die Kantenbeschriftung $\eta(e)$, d.h. die Nachbedingung der Aktion ist einschränkender als die Kantenbeschriftung.

4. Streiche alle Knoten k aus F, für die keine Löschaktion $a \in \mathcal{T}_F(O)$ mit der Spezifikation $\mathbf{on}\,\omega : \{\pi\}a\{\psi\}$ existiert, so daß $\omega \wedge \pi$ die Disjunktion der Kantenbeschriftungen der auf k führenden Kanten impliziert.

5. Reduziere den erhaltenen Graphen.

Für einen iterationsinvarianten Transitionsgraphen T_φ müssen bei einem objektorientierten Entwurf in Schritt 3 nur Kanten (s_i, s_j) mit $s_i \neq s_j$ betrachtet werden.

Je nachdem, ob die Löschung von Objekten der betroffenen Klasse erlaubt sein soll oder nicht, muß die Reduktion bezüglich endlicher oder nichtendlicher Zustandsfolgen durchgeführt werden. Ist der reduzierte Graph nicht leer, läßt sich eine Folge von (anwendbaren) Aktionen angeben, die die Integritätsbedingung φ erfüllt.[14] □

Für viele Fälle ist die Forderung, daß die Aktionenspezifikation die betreffenden Kanten des Transitionsgraphen *impliziert*, zu stark. Beispiele dafür sind Zwischenschritte der Spezifikation oder die Beschränkung auf Elementaraktionen. Die Aktionen sind dann für die Erzwingung der dynamischen Integritätsbedingung nicht ausreichend einschränkend spezifiziert. In jedem Fall ist aber die beschriebene Eigenschaft ein nützliches Qualitätsmerkmal für eine vierschichtige Spezifikation.

Auch in den Fällen, in denen die Erfüllung einer Integritätsbedingung durch die Aktionenspezifikation nicht gegeben ist, sollte eine Implementierung der Aktionen möglich sein. Für praktische Entwurfszwecke ist daher die folgende Eigenschaft relevant :

Definition 5.5.3 Eine Aktionenspezifikation $\mathcal{T}(O)$ *widerspricht nicht* einer Integritätsbedingung $\varphi \in IB(O)$, wenn in dem Algorithmus in Definition 5.5.2 in den Schritten 2., 3. und 4. statt der Implikation gefordert wird, daß der jeweilige Teil der Aktionenspezifikation der Kantenbeschriftung nicht widerspricht.

In diesem Fall läßt sich eine Verfeinerung der Aktionenspezifikation angeben, die die Integritätsbedingungen erfüllt. □

[14] Genaugenommen müssen die möglichen Parameter einer Aktion sowie die belegungsabhängigen Effekte mit berücksichtigt werden. Darauf wird in diesem Abschnitt verzichtet, da hier nur ein Überblick über die grundlegenden Verfahrensweisen und Notationen gegeben werden soll.

Diese Eigenschaft ist eine notwendige Bedingung dafür, daß — bezüglich einer Integritätsbedingung einer Objektklasse — überhaupt ausführbare Datenbankentwicklungen möglich sind.

Selbst wenn die Aktionenspezifikation eine dynamische Integritätsbedingung erfüllt, führen nicht alle Folgen von Aktionen notwendigerweise zu ausführbaren Datenbankentwicklungen. Gefordert ist im Prinzip nur, daß mindestens eine derartige Folge existiert.

Als Entwurfsziel ist darum oft erwünscht, daß die Aktionenspezifikation die dynamischen Integritätsbedingungen erzwingt, d.h. daß jede (ausführbare) Folge von Aktionen, die den Lebenslauf eines Objektes festlegt, die Einhaltung der Entwicklungsspezifikation garantiert. Dieses Ziel kann i.a. nur durch schrittweise Transformation der Aktionenspezifikation erreicht werden, da die erste Festlegung der Aktionen durch eine Spezifikation in der Regel nur die beabsichtigten Änderungen festlegt und die Einhaltung der Integritätsbedingungen nicht beinhaltet. Methodische Vorschläge für die zielgerichtete Transformation einer Aktionenspezifikation werden in [Li86, Li87] vorgestellt.

Definition 5.5.4 Eine Aktionenspezifikation $\mathcal{T}(O)$ *erzwingt* eine dynamische Integritätsbedingung $\varphi \in IB(O)$, wenn $\mathcal{T}(O)$ die Formel φ erfüllt (Def. 5.5.2) und für jeden Knoten s_i und jede Aktion a mit der Spezifikation $\mathbf{on}\,\omega : \{\pi\}a\{\psi\}$ gilt:

1. Ist a eine Einfügeaktion, dann existiert ein Knoten $s' \in m_0$ und eine Kante (s', s'') mit $\psi \Rightarrow \eta((s', s''))$.
2. Wenn $\omega \wedge \pi$ der Disjunktion der Kantenbeschriftungen der auf s_i führenden Kanten nicht widerspricht, dann existiert auch eine Kante (s_i, s') mit $\psi \Rightarrow \eta((s_i, s'))$.
3. Ist a eine Löschaktion, so widerspricht $\omega \wedge \pi$ nur dann der Disjunktion der Kantenbeschriftungen der auf s_i führenden Kanten nicht, wenn s_i ein finaler Knoten ist.

Für jede (endliche, mit einer Einfügeaktion startende und mit einer Löschaktion endende) Folge von Aktionen läßt sich dann ein korrekter Pfad durch den Transitionsgraphen T_φ angeben, der die Formel φ erfüllt. □

Die bisherigen Notationen und Definitionen behandelten den ersten Fragenkomplex, ob die Integritätsbedingungen durch die spezifizierten Aktionen erfüllt werden können. Für die Konsistenz des Gesamtschemas nicht zwingend ist die komplementäre

Frage nach der Ausführbarkeit einer bestimmten Aktion — die Gesamtspezifikation kann trotzdem konsistent sein. Während des Entwurfsprozesses ist natürlich die Erkenntnis, daß z.B. eine spezifizierte Transaktion "`entlasse Angestellten`" nie ausführbar ist, unabhängig von der Frage nach der Konsistenz des Gesamtentwurfs von großem Interesse.

Definition 5.5.5 Folgende Bedingungen sind notwendig für die *Ausführbarkeit* einer Aktion a mit der Spezifikation $\mathbf{on}\,\omega : \{\pi\}a\{\psi\}$ bezüglich einer dynamischen Integritätsbedingung $\varphi \in IB(O)$ mit reduziertem Prüfschema RT_φ :

- Für alle Einfügeaktionen $a \in \mathcal{T}_I(O)$ muß gelten :

  Die Nachbedingung ψ von a widerspricht nicht der Disjunktion der Kantenbeschriftungen der von Knoten aus m_0 ausgehenden Kanten.

- Für alle Änderungsaktionen $a \in \mathcal{T}(O)/(\mathcal{T}_I(O) \cup \mathcal{T}_F(O))$ muß gelten :

  Es gibt einen Pfad (s, s', s'') entlang der Kanten (s, s') und (s', s'') durch RT_φ, so daß sowohl $\omega \wedge \pi \wedge \eta((s, s'))$ als auch $\psi \wedge \eta((s', s''))$ erfüllbar sind.

- für alle Löschaktionen $a \in \mathcal{T}_F(O)$ muß gelten :

  Es existiert ein finaler Knoten in RT_φ, auf den eine Kante führt, deren Beschriftung nicht $\omega \wedge \psi$ widerspricht.

Diese Überprüfung muß für alle $\varphi \in IB(O)$ bzw. für deren Konjunktion vorgenommen werden. □

Für jede ausführbare Aktion a muß es daher einen Übergang im reduzierten Prüfschema geben, der nicht der Aktionenspezifikation widerspricht. Mit anderen Worten, es gibt eine korrekte Datenbankentwicklung (die nicht ausführbar sein muß !) bis zu einem gewissen Zeitpunkt, so daß die Aktion anschließend prinzipiell ausführbar ist. Es folgt *nicht* aus der prinzipiellen Ausführbarkeit der Einzelaktionen, daß damit auch eine *ausführbare* Datenbankentwicklung existiert. Die Bedingungen sind demnach nur notwendige Kriterien für die prinzipielle Ausführbarkeit von Einzelaktionen.

Mit den bisherigen Definitionen lassen sich nun notwendige Kriterien für eine global ausführbare Datenbankentwicklung angeben. Die dritte Frage vom Kapitelanfang läßt sich so zum Teil beantworten. Es wird vorausgesetzt, daß die einzelnen Schichten in sich konsistent, also ihre Spezifikationen erfüllbar sind.

Satz 5.5.6 Für eine objektorientierte vierschichtige Spezifikation sind folgende Punkte Voraussetzungen für die Existenz einer globalen ausführbaren Datenbankentwicklung, in der konzeptionell unabhängige Aktionen in der Reihenfolge vertauscht werden können.

1. Die Aktionenspezifikationen $\mathcal{T}(O)$ erfüllen für jede Objektsorte O die Konjunktion der für O relevanten dynamischen Integritätsbedingungen.

2. Alle dynamischen Integritätsbedingungen haben iterationsinvariante reduzierte Prüfschemata.

3. Es existiert ein initialer Datenbankzustand.

Beweisskizze : Die erste Bedingung garantiert lokale ausführbare Datenbankentwicklungen bezüglich einer festen Objektklasse. Der zweite Punkte garantiert, daß die Aktionen für verschiedene Objektklassen konzeptionell unabhängig betrachtet werden können.

Die dritte Forderung ist speziell dann von Interesse, wenn die Wahl der "leeren Datenbank", d.h. leere Träger für jede Objektsorte, durch statische Integritätsbedingungen ausgeschlossen wird. In diesem Falle sind neben der Existenz eines initialen Datenbankzustands auch Fragen nach der Eindeutigkeit oder der endlichen Darstellung wichtig. □

Zusammenfassend kann gesagt werden, daß die Konstruktion von Prüfschemata aus dynamischen Integritätsbedingungen ein Ansatzpunkt für die Konsistenzüberprüfung und Qualitätsbewertung einer konzeptionellen Dynamikspezifikation bietet. Die Überprüfung der hier vorgestellten Kriterien kann in eine entwurfsunterstützende Umgebung eingebunden werden oder als formale Grundlage der Bewertung von Spezifikationen dienen.

Ein objektorientierter Entwurf ermöglicht hier eine semantisch abgesicherte lokale Überprüfung dieser Kriterien, die sonst in einem großen Entwurf allein vom Aufwand her nicht durchführbar wäre. Deutlich wird auch der Vorteil, der durch eine deskriptive, mathematisch fundierte Spezifikation der Benutzeraktionen bereits in der konzeptionellen Entwurfsphase entsteht — so kann eine integritätsverletzende Transaktion oder eine zu stark einschränkende Integritätsbedingung oft schon in dieser frühen Entwurfsphase erkannt werden.

6 Überwachung dynamischer Integrität

Die finit-lineare temporale Logik ist ein formaler Spezifikationskalkül, der die deskriptive Beschreibung der dynamischen Entwicklung einer Datenbank-Anwendung ermöglicht. Die Konstruktion von Prüfschemata erlaubt während des Entwurfs eine graphische Darstellung der spezifizierten Entwicklungen für einzelne Objekte und gleichzeitig Aussagen über die Konsistenz der Entwicklungsspezifikation eines konzeptionellen Schemas. Offen ist bisher allerdings die Berücksichtigung dieser deskriptiven Spezifikation in der Implementierung der Datenbank-Anwendung.

Durch die Konstruktion von Transitionsgraphen aus temporalen Formeln ist eine operationale Semantik für temporale Formeln gegeben, die als Ansatzpunkt für eine Implementierung der Überwachung der dynamischen Integritätsbedingungen geeignet ist. In diesem Kapitel werden zwei komplementäre Vorgehensweisen für die Implementierung der Integritätsüberwachung vorgestellt, nämlich die Überwachung durch eine spezielle Komponente eines Datenbankmanagementsystems, dem sogenannten *Integritätsmonitor*, und die Erzwingung der Integritätsforderungen durch die implementierten Transaktionen.

Abschnitt 6.1 diskutiert die Anforderungen an eine Integritätsüberwachung und stellt die beiden Ansätze vor. Die Abschnitte 6.2 und 6.3 behandeln die algorithmischen Grundlagen der Überwachung durch einen Integritätsmonitor und Hinweise zu seiner Integration in ein DBMS. Der Abschnitt 6.4 skizziert die Integration der Integritätsüberwachung in die Transaktionsimplementierung.

In den folgenden Abschnitten werden die aus dem Bereich der Implementierung von Datenbank-Anwendungen bekannten Fachbegriffe benutzt, die oft nicht dem bisherigen Gebrauch im Bereich des konzeptionellen Entwurfs entsprechen. Ein *Datenbankschema* in einem Datenmodell beinhaltet hier ausschließlich die Strukturbeschreibung und eingeschränkte statische Integritätsbedingungen, die vom verwendeten DBMS unterstützt werden. Eine *Transaktion* ist eine Änderungsaktion, die einen konsistenten Datenbankzustand in einen neuen konsistenten Datenbankzustand überführt und dabei alle Integritätsforderungen berücksichtigt.

Auf die Systemarchitektur- und Verfahrensvorschläge zur Integration der Überwachung statischer Integritätsbedingungen wird nicht eingegangen. Eine Literaturübersicht kann in [Reu87, Li87] gefunden werden.

6.1 Implementierung der spezifizierten Objektdynamik

In der Entwicklungsschicht werden die erlaubten dynamischen Vorgänge deskriptiv durch temporale Formeln spezifiziert. Beschrieben werden die Eigenschaften *vollständiger* Objektlebensläufe, die entweder abgeschlossene Lebensläufen von Einzelobjekten, beginnend mit dem Einfügen in die Datenbank bis zum Löschen, oder sogar unendliche Lebensläufen sind.

In einer Implementierung der spezifizierten Datenbank-Anwendung sind zu einem bestimmten Systemzeitpunkt jeweils nur Anfangsstücke von Objektlebensläufen bekannt. Über deren Fortsetzungen können prinzipiell keine Aussagen gemacht werden, da sie erst durch die in willkürlicher Reihenfolge stattfindenden Transaktionsaufrufe festgelegt werden. Ziel der Überwachung von dynamischen Integritätsbedingungen ist es, alle Folgen von Transaktionen, die die Integritätsbedingungen verletzen, zurückzuweisen oder sie gar nicht erst zu ermöglichen.

In einer Datenbank-Anwendung ist es natürlich nicht akzeptabel, wenn ganze Folgen von Transaktionen zurückgewiesen werden müssen. Eine Überwachung von Integritätsbedingung sollte möglichst nur diejenige Transaktion zurückweisen, die die Integritätsverletzung hervorruft — und diese Zurückweisung sollte möglichst direkt erfolgen, um den Zugriff auf die nun inkonsistente Datenbank durch andere Transaktionen zu verhindern.[15]

Da es aus diesen praktischen Gründen nicht möglich ist, die vollständig abgeschlossenen Objektlebensläufe auf Integritätsrespektierung zu untersuchen (diese Vorgehensweise würde außerdem die Speicherung der gesamten Objekthistorie verlangen), muß die Überprüfung der dynamischen Integritätsbedingungen parallel zu den anderen Aufgaben des DBMS unternommen werden. Noch offene zeitliche Forderungen z.B. der Form "**sometime** A" müssen dabei optimistisch behandelt werden.

Definition 6.1.1 Eine dynamische Integritätsbedingung heißt *potentiell gültig* in einer endlichen Zustandsfolge $\hat{\sigma}$, wenn $\hat{\sigma}$ zu einer Zustandsfolge $\hat{\sigma}' = \hat{\sigma} \circ \hat{\sigma}''$ verlängert werden kann, die diese Integritätsbedingung erfüllt. □

Diese Definition formalisiert die "optimistische" Behandlung von offenen temporalen Existenzforderungen. Für eine temporale Formel φ ohne globale und Reduktquantifizierung ist die vorläufige Gültigkeit unter einer festen Belegung gemäß Satz 4.5.9 von Seite 66 äquivalent zur vorläufigen Akzeptanz durch das reduzierte Prüfschema

[15] Dies ist in der Regel nur möglich, wenn die Reihenfolge der Transaktionen beliebig ist und die überwachten Integritätsbedingungen konzeptionell unabhängig sind. Für allgemeine Entwicklungsspezifikationen sind 'Sackgassen' möglich, also Datenbankzustände, die von keiner spezifizierten Transaktion unter Einhaltung aller dynamischen Integritätsbedingungen in einen neuen Zustand überführt werden können, selber aber das Ergebnis einer Transaktion sind, die keine der Einzelintegritätsbedingungen verletzt haben.

Beim Auftreten einer solchen Situation müssen entweder mehrere Transaktionen zurückgesetzt werden oder die Menge der verwalteten dynamischen Integritätsbedingungen muß abgeändert werden.

RT_φ. Dies gilt natürlich nur für belegungsabhängige Reduktion. Für den praktischen Einsatz ist jedoch nur die belegungsunabhängige Reduktion bzw. die Reduktion für wenige Belegungsklassen mit vertretbarem Aufwand durchzuführen.

Da dynamische Integritätsbedingungen derartige temporale Formeln mit freien Variablen sind, können sie durch den Einsatz der aus ihnen konstruierten reduzierten Prüfschemata überwacht werden. Aufgrund von Definition 3.4.1 der belegungsunabhängigen Gültigkeit von temporalen Formeln muß diese Überprüfung für alle Belegungen der freien Variablen mit aktuellen Datenbankobjekten vorgenommen werden. Jede Transaktion erfordert die Ausführung einer Transition im Prüfschema — bei iterationsinvarianten Prüfschemata müssen dabei nur diejenigen Transaktionen berücksichtigt werden, die die Objekteigenschaften verändern.

Diese Vorgehensweise erfordert eine Erweiterung des logischen bzw. physischen Datenbankschemas der Implementierung um die für Integritätsüberwachung benötigte historische Information. Die aktuelle Markierung des reduzierten Prüfschemas beinhaltet die minimal benötigte historische Information über ein Objekt, die für die Überprüfung einer temporalen Formel benötigt wird. In [Hü88] wird die nötige Schemaerweiterung für das relationale Datenmodell vorgestellt.

In vielen praktischen Fällen kann die aktuelle Markierung direkt aus anderen Objekteigenschaften konstruiert werden, so z.B. in der Beispielintegritätsbedingung `familienstand` von Seite 84, in der die aktuelle Markierung aus dem Attribut `status` abgeleitet werden kann. In diesen Fällen ist eine Schemaerweiterung nicht notwendig. Stattdessen muß für die Integritätsüberwachung eine Berechnungsvorschrift für die Markierung bekannt sein.

Wie bereits in Kapitel 5 erwähnt, kann die Überwachung einer dynamischen Integritätsbedingung auf zwei Arten erfolgen. Die Überwachung der dynamischen Integritätsbedingung kann durch eine spezielle Systemkomponente erfolgen, die eine (möglicherweise variable) Menge von Integritätsbedingungen verwaltet, die in einer an das benutzte Datenmodell orientierten temporalen Logik formuliert ist. Diese Systemkomponente, der Integritätsmonitor, berechnet transaktionsabhängig für die aktuellen Belegungen die Markierungen in den reduzierten Prüfschematas und weist Transaktionen zurück, die eine Integritätsbedingung verletzen. Die algorithmischen Grundlagen und Hinweise zur Integration in ein implementiertes DBMS werden in den Abschnitten 6.2 und 6.3 diskutiert.

Die zweite Methode zur Integritätsüberwachung überläßt die Integritätsüberwachung den implementierten Transaktionen. Die Transaktionen überprüfen, ob die von ihnen vorgenommenen Änderungen die Integritätsbedingungen betreffen und reagieren geeignet auf Konflikte zu Integritätsbedingungen. Diese Transaktionen werden als *integritätsrespektierend* bezeichnet.

Die für die Überprüfung temporaler Formeln benötigte historische Information muß bei Bedarf von den integritätsrespektierenden Transaktionen aktualisiert werden. Formale Richtlinien für die Erzeugung integritätsrespektierender Transaktionen sind Inhalt von Abschnitt 6.4.

Jeder dieser beiden Vorgehensweisen hat sowohl Vor- als auch Nachteile. Einige ihrer jeweiligen Vorteile werden im folgenden gegenübergestellt.

- Ein Integritätsmonitor ermöglicht bis zu einem gewissen Grade das Ändern und Neueinfügen von dynamischen Integritätsbedingungen zur Systemlaufzeit.
- Ein Integritätsmonitor ermöglicht die Handhabung von willkürlichen Ad-hoc-Transaktionen und das Austesten von in der Implementierung befindlichen Transaktionen mit zum Teil unbekannten Auswirkungen.
- Ein Integritätsmonitor garantiert ein einheitliches Systemverhalten bei Integritätsverletzungen.
- Integritätsrespektierende Transaktionen führen eine Integritätsprüfung nur dann durch, wenn diese wirklich notwendig ist.
- Integritätsrespektierende Transaktionen können die Überprüfung der Integritätsbedingungen auf die effizientest mögliche Art verwirklichen.
- Integritätsrespektierende Transaktionen ermöglichen eine flexible Reaktion auf Integritätsverletzungen.

Der Ansatz der integritätsrespektierenden Transaktionen kann nur dann als einziger Überwachungsmechanismus eingesetzt werden, wenn alle auftretenden Transaktionen daraufhin verifiziert sind — ein Zustand, der bei flexiblen Datenbank-Anwendungen wohl eher zu den Ausnahmen zählt. Die Argumente der Effektivität und der Flexibilität sprechen allerdings eindeutig für diese Vorgehensweise.

Für einen praktischen Systemvorschlag ist eine Kombination der beiden Ansätze wahrscheinlich die optimale Lösung. "Sichere" Transaktionen, die die Integritätsbedingungen respektieren, aktivieren den Integritätsmonitor nicht und erreichen dadurch ein effizienteres Systemverhalten. Aufgrund des hohen Aufwandes werden nur häufig wiederkehrende Transaktionen dahingehend verifiziert. Seltener auftretende Transaktionen bzw. ad hoc formulierte Änderungen werden vom Integritätsmonitor überwacht und gegebenenfalls zurückgewiesen. Der Monitor kann ebenfalls in der Testphase für integritätsrespektierende Transaktionen eingesetzt werden.

Eine derartige hybride Integritätsüberwachung ermöglicht gleichzeitig eine flexible Handhabung von Schemaänderungen während der Systemlaufzeit. Änderungen und

Ergänzungen sowohl der Integritätsbedingungen als auch der Transaktionen können vorgenommen werden, ohne daß sofort eine zeitraubende Reimplementierung bzw. Verifikation aller Transaktionen nötig wird.

In den folgenden Abschnitten werden nur diejenigen Aspekte der Implementierung untersucht, die die Grundlagen für die Berücksichtigung der in der Entwicklungsschicht spezifizierten dynamischen Integritätsbedingungen bilden. Zusätzlich zu den vorgestellten algorithmischen Grundlagen für die Integritätsüberwachung sind im Zusammenhang mit dynamischen Integritätsbedingungen u.a. die folgenden Punkte relevant :

- Die Implementierung eines korrekten, zeitpunktunabhängigen Identifikationsmechanismus für die betroffenen Datenbankobjekte,
- die Repräsentation der Objekte und Datenwerte im konkreten Datenmodell,
- die zusätzliche Berücksichtigung modellinhärenter Integritätsbedingungen bei der Integritätsüberwachung und
- die Umsetzung der atomaren und prädikatenlogischen Formeln in einen Anfragekalkül für das zugrundeliegende DBMS.

Auf diese zusätzlichen Implementierungsaspekte wird in dieser Arbeit nicht näher eingegangen.

6.2 Überwachungsprinzip eines Integritätsmonitors

Ein Monitor für dynamische Integrität benutzt reduzierte Prüfschemata, um die dynamischen Integritätsbedingungen für einzelne Datenbankobjekte zu überwachen. Aufgrund der Festlegung der als dynamische Integritätsbedingungen erlaubten temporalen Formeln ist die Konstruktion von Prüfschemata immer möglich und die Formeln betreffen jeweils die Lebensläufe einzelner Objekte bzw. Objektkombinationen.

Für eine Integritätsbedingung muß der Monitor prinzipiell für alle Belegungen der freien Variablen (mit aktuell gespeicherten Datenbankobjekten) die aktuelle Markierung im Prüfschema verwalten und nach jedem Zustandsübergang neu berechnen. Sowohl die Menge der in Frage kommenden Belegungen als auch die Anlässe der Neuberechnung können und müssen stark eingeschränkt werden, um die Überwachung mit einem sinnvollen Aufwand durchführen zu können.

An dieser Stelle werden nur die algorithmischen Grundlagen der Integritätsüberwachung vorgestellt und einige Hinweise gegeben, an welchen Stellen Effizienzsteigerung möglich und sinnvoll ist. Konkrete Vorschläge für Effizienzsteigerungen können im folgenden Abschnitt 6.3 sowie in [Hü88] gefunden werden.

Zuerst müssen einige Notationen festgelegt werden, die in der Formulierung und Diskussion der Algorithmen benötigt werden.

Definition 6.2.1 Datenbankzustände sind Σ–Strukturen für das gegebene Datenbankschema. Für die Integritätsüberwachung werden Übergänge zwischen Datenbankzuständen betrachtet. Die Überprüfung erfolgt jeweils nach dem Zustandsübergang. Der aktuelle Datenbankzustand nach dem Übergang wird mit σ und der alte Zustand vor dem Übergang mit $\sigma_{\mathtt{old}}$ bezeichnet.

Betrachtet werden jeweils Belegungen β der freien Variablen $\mathit{VAR}(\mathit{IB})$ einer Integritätsbedingung IB durch Objekte in den Datenbankzuständen σ oder $\sigma_{\mathtt{old}}$. Die Menge dieser Belegungen wird mit $\mathit{BEL}(\mathit{VAR}(\mathit{IB}), \sigma \cup \sigma_{\mathtt{old}})$ bezeichnet. Allgemein können als abkürzende Notation die Mengenoperationen auf den Belegungsmengen durch die entsprechende Notation auf den Datenbankzuständen als zweites Argument von *BEL* ausgedrückt werden.

Für jede Integritätsbedingung IB wird eine Schemaerweiterung um die Markierung $m_{\mathit{IB}}(\mathit{VAR}(\mathit{IB}))$ eingeführt, in der für eine Belegung der Variablen in $\mathit{VAR}(\mathit{IB})$ eine Markierung für das reduzierte Prüfschema $\mathit{RT}_{\mathit{IB}}$ abgespeichert werden kann. Die Realisierung hängt vom verwendeten Datenmodell ab. □

Für jede dynamische Integritätsbedingung muß eine *Initialisierung* durchgeführt werden, in der die notwendigen Prüfschemaberechnungen und Initialisierungen der Markierungen vorgenommen werden. Der Algorithmus `INIT` für die Initialisierung der Überwachnug einer Integritätsbedingung ist in Abbildung 14 auf Seite 110 skizziert.

Die Initialisierung kann zur Definitionszeit des physischen Datenbankschemas (z.T. auch in früheren Entwurfsstadien, insbesondere sinnvoll im Zusammenhang mit einer Konsistenzprüfung) oder beim Neueinfügen einer Integritätsbedingung erfolgen. Im letzteren Fall werden alle schon in der Datenbank vorhandenen Objekte als neu in die Datenbank eingefügt betrachtet.

Für alle initialisierten Integritätsbedingungen muß dann nach jedem durch eine Transaktion hervorgerufenen Zustandsübergang überprüft werden, ob sie durch diesen Übergang verletzt worden ist. Dies kann durch den in Abbildung 15, Seite 111, formulierten Algorithmus `MONITOR` geschehen, der nach jeder (Änderungs-) Transaktion aufgerufen werden muß. Der Aufruf muß auch für den initialen Datenbankzustand ausgeführt werden, um die Integritätsüberwachung zu initialisieren.

Die eigentliche Überwachung einer Integritätsbedingung IB erfolgt im Algorithmus `MONITOR` dadurch, daß die Integritätsbedingung für alle Belegungen der Variablen

Algorithmus 6.2.2 Gegeben sei eine Integritätsbedingung IB. Die *Initialisierung* der Integritätsüberwachung für IB geschieht durch folgenden Algorithmus.

algorithmus `INIT`(IB) :

Berechne RT_{IB};
if RT_{IB} ist leerer Transitionsgraph
then Fehler : inkonsistente Integritätsbedingung
else Erweitere Schema um $m_{IB}(VAR(IB))$;
  if Datenbank ist bereits initialisiert
  then Rufe `MONITOR` für IB mit $BEL(VAR(IB), \sigma_{\mathtt{old}}) = \emptyset$ auf;
  endif;
endif.

□

Abbildung 14: Initialisierung der Integritätsüberwachung

in IB mit Datenbankobjekten gesondert überprüft wird. Für diese Überprüfung wird fortlaufend bei jedem Zustandsübergang die neue Markierung im reduzierten Prüfschema errechnet und bei einer Nichtakzeptanz der bisherigen Zustandsfolge eine Fehlerbehandlung durch `ERROR` erfolgt.

Markierungen für neu eingefügte Objekte werden vor der ersten Anwendung der Transitionsregel initialisiert. Die Löschung von Objekten wird nur akzeptiert, wenn die Markierung einen finalen Knoten enthält. Der Algorithmus verwirklicht die Überprüfung der *belegungsunabhängigen Akzeptanz* durch belegungsunabhängig reduzierte Prüfschemata für alle dynamischen Integritätsbedingungen. Er überwacht daher auch die potentielle Gültigkeit dieser dynamischen Integritätsbedingungen, sofern die belegungsabhängige Reduktion nicht einen weiter reduzierten Graphen ergeben würde.

Die Überprüfung der potentiellen Gültigkeit ohne Einschränkungen kann nur erreicht werden, wenn alle belegungsabhängigen zusätzlichen Reduktionen ebenfalls berücksichtigt werden. Dies kann durch die Reduktion für Belegungsklassen geschehen, falls die Klasseneinteilung den belegungsabhängigen Effekt vollständig abdeckt. Sinnvoll ist hier zum Beispiel, für diejenigen Belegungen gesondert zu reduzieren, die verschiedenen Variablen das gleiche Objekt zuordnen. In diesem Fall kann der Wert vieler atomarer Formeln in der Reduktion berücksichtigt werden (z.B. $(x = y)$ oder $\mathtt{gehalt}(x) \leq \mathtt{gehalt}(y)$ für $\beta(x) = \beta(y)$). Die Reduktion bezüglich Belegungsklassen ist bereits zur Entwurfs- bzw. Initalisierungszeit möglich und kann zu diesem

Algorithmus 6.2.3 (Integritätsmonitor) Für jede der überwachten Integritätsbedingungen *IB* wurde das reduzierte Prüfschema $RT_{IB} = (G', \nu', \eta', m'_0, F')$ konstruiert und das Datenbankschema um die Markierung m_{IB} erweitert (Algorithmus `INIT`). Die Transitionsfunktion für RT_{IB} wird mit trans_{IB} bezeichnet und muß durch Datenbankanfragen gemäß der Definition 4.4.1 der Transitionsregel implementiert werden.

algorithmus `MONITOR`

for all *IB* mit $RT_{IB} = (G', \nu', \eta', m'_0, F')$ und trans_{IB}
for all $\beta \in BEL(VAR(IB), \sigma \cup \sigma_{\mathtt{old}})$
do
    if $\neg(\beta \in BEL(VAR(IB), \sigma))$ (* gelöschte Objekte *)
    then if $m_{IB}(\beta) \cap F' \neq \emptyset$
        then `ERROR`; (* Integritätsbedingung *IB* verletzt *)
        else Lösche $m_{IB}(\beta)$;
        endif;
    endif;
    if $\neg(\beta \in BEL(VAR(IB), \sigma_{\mathtt{old}})$ (* neu eingefügte Objekte *)
    then Erzeuge Markierung $m_{IB}(\beta)$;
        $m_{IB}(\beta) := m_0{}'$;
    endif;
    if $\beta \in BEL(VAR(IB), \sigma)$ (* aktuelle Objekte *)
    then $m_{IB}(\beta) := \mathrm{trans}_{IB}(m_{IB}(\beta))$;
        if $m_{IB}(\beta) = \emptyset$
        then `ERROR`; (* Integritätsbedingung *IB* verletzt *)
        endif;
    endif;
enddo.

Die nicht näher spezifizierte Aktion `ERROR` muß die ausgeführte Transaktion mit all ihren Auswirkungen zurücksetzen und den Zustandsübergang rückgängig machen. Zu diesen Auswirkungen gehören auch bereits erfolgte Aktualisierungen bzw. Einfügungen und Löschungen von Markierungen $m_{IB}(\beta)$ durch den Algorithmus `MONITOR`. □

Abbildung 15: Integritätsmonitor

Zeitpunkt bereits Hinweise auf "kritische" Belegungsklassen liefern.

Eine alternative Methode, die belegungsabhängigen Effekte in die Überwachung zu integrieren, ist es, während der Laufzeit des Integritätsmonitors jeweils für die neu eingefügten Belegungen eine belegungsabhängige Reduktion durchzuführen. Der reduzierte Graph muß dann ebenfalls abgespeichert werden oder es muß bei einer Modifikation der Markierung neu reduziert werden, um Transitionen in unerfüllbare Teilgraphen zu vermeiden.

Im Algorithmus `MONITOR` werden für alle Integritätsbedingungen und alle möglicherweise betroffenen Objektkombinationen die Markierungen in den jeweiligen Prüfschemata berechnet und aktualisiert. Dieser für den praktischen Einsatz nicht akzeptable Aufwand muß geeignet reduziert werden, um ein annehmbares Systemverhalten zu ermöglichen. Ansätze zur Effizienzsteigerung sind Inhalt des folgenden Abschnitts 6.3.

6.3 Ansatzpunkte zur Effizienzsteigerung

Der Algorithmus `MONITOR` beschreibt das algorithmische Grundprinzip für ein System, das dynamische Integritätsbedingungen überwachen kann. Sowohl der Berechnungsaufwand als auch der Platzbedarf dieser ersten Algorithmenfassung sind für den praktischen Einsatz zu hoch, wie im folgenden an Beispielen klar gemacht wird.

Als Maß für den Berechnungsaufwand wird die Anzahl der Schleifendurchläufe im Algorithmus `MONITOR` angenommen. Unter der Bedingung, daß die Datenbankzustände σ und $\sigma_{\texttt{old}}$ identische Trägermengen für die Objektsorten besitzen, entspricht der Aufwand eines Durchlaufs grob dem Aufwand für die Berechnung

$$m_{IB}(\mathit{VAR}(IB)) := \mathrm{trans}_{IB}(m_{IB}(\mathit{VAR}(IB))).$$

Dies kann als Abschätzung für den Aufwand bei beliebigen Änderungen angenommen werden, da die in der Regel kleineren Mengen der gelöschten und eingefügten Objekte einen vergleichbaren Aufwand (bei geeigneter Implementierung jeweils ein Lese- und Schreibzugriff auf die Datenbank) pro Schleifendurchlauf erfordern.

Der Gesamtaufwand kann prinzipiell durch zwei Maßnahmen verringert werden, nämlich durch die Verringerung der Anzahl der Schleifendurchläufe und durch die Verringerung des Aufwands für eine Einzelberechnung.[16] Der Aufwand einer Einzelberechnung wird dominiert durch den Aufwand für den Lese- und Schreibzugriff auf die

[16]Zusätzliche implementierungsabhängige Optimierungen, wie geeignete Optimierung des Datentransfers zwischen Hintergrundspeicher und dem Datenbankmanagementsystem oder Zugriffsunterstützung durch Zugriffspfade, sind ebenfalls möglich.

Objektinformationen und die Markierungen. Z.B. sollte ein Schreibzugriff nur erfolgen, wenn sich die Markierung durch die Transitionsregel geändert hat. Der Aufwand für die Berechnung der Transitionsregel kann durch die Verwendung deterministischer Prüfschemata deutlich verringert werden.

Ein wesentlich größerer Optimierungsspielraum besteht bei der Verringerung der Anzahl der Schleifendurchläufe. Um diese Anzahl zu verringern, bieten sich eine Reihe von Verfahren an, die nun kurz vorgestellt werden. Auf zwei besonders vielversprechende Ansätze wird anschließend genauer eingegangen.

1. Die Anzahl der zu überprüfenden *Integritätsbedingungen* kann dadurch verringert werden, daß aus mehreren Integritätsbedingungen für dieselbe Objektsorte bzw. Sortenkombination nur *ein* Prüfschema konstruiert und überwacht wird.

2. Belegungen, für die die Integritätsbedingungen vollständig erfüllt sind, können aus der Überprüfung herausgenommen werden. Dies ist dann der Fall, wenn die Markierung für diese Belegung einen mit einer Tautologie beschrifteten Knoten enthält (vergl. Kapitel 4.4).

   Während einer großen Zeitspanne der Überwachung haben für viele Integritätsbedingungen die meisten Belegungen die gleiche Markierung. In diesen Fällen kann diese Markierung als Default-Markierung angenommen werden, so daß nur die anderen Markierungen explizit abgespeichert werden müssen.

3. Die Anzahl der betrachteten Belegungen kann für viele Formeln stark eingeschränkt werden, da sie für die Gültigkeit der Integritätsbedingung nicht relevant sind. Ein Beispiel sind Formeln, bei denen in der Prämisse einer Implikation ein bestimmter Wert an eine Variable gebunden wird. In diesem Fall sind die anderen Belegungskombinationen nicht relevant für die Überwachung.

   Die Integritätsbedingung `gehaltsentwicklung` aus Abbildung 11 von Seite 84 ist ein Beispiel für eine derartige Integritätsbedingung. Für ihre Gültigkeit sind nur diejenigen Kombinationen von Personen und Firmen relevant, bei denen irgendwann einmal ein Arbeitsverhältnis bestand. Ebenfalls müssen für diese Kombinationen nur Gehälter betrachtet werden, die die betroffene Person einmal von dieser Firma bezogen hat.

   Eine derartige Einschränkung des Belegungsraumes ist unumgänglich für die Belegungen von Datenvariablen, da aufgrund der vorgestellten Semantik von Datenbankzuständen alle ihre möglichen Werte aktuell sind — und das sind, sieht man von implementierungsabhängigen Einschränkungen ab, z.B. für `integer` oder `text` abzählbar unendlich viele.

Methoden zur Einschränkung des Belegungsraumes werden in Abschnitt 6.3.1 diskutiert.

4. In vielen Transaktionen werden nur die Attribute einiger weniger Objekte verändert, so daß die Integritätsüberprüfung auf diese Objekte beschränkt werden kann. Voraussetzung hierfür sind allerdings iterationsinvariante Prüfschemata. Durch zusätzliches Wissen über die möglichen Auswirkungen einer Transaktion kann die Menge der zu überprüfenden Objekte und Integritätsbedingungen weiter eingeschränkt werden.

   Methoden zur Auswahl der zu überprüfenden Belegungen für eine Integritätsbedingung sind Inhalt des Abschnittes 6.3.2.

Bevor die beiden vielversprechendsten Ansatzpunkte für die Optimierung, die Einschränkung des Belegungsraumes und die Transaktionsanalyse, genauer vorgestellt werden, sei noch darauf hingewiesen, daß der Algorithmus `MONITOR` hochgradig parallelisierbar ist, da alle Berechnungen in den Schleifendurchläufe unabhängig voneinander sind. Konflikte können allerdings bei dem Lesezugriff auf zu überprüfende Objekte entstehen.

6.3.1 Einschränkung des Belegungsraumes

Während für Objektvariablen aufgrund der Semantik von Datenbankzuständen nur die aktuell gespeicherten Objekte als Belegung der Variablen in Frage kommen, sind für Datenvariablen prinzipiell alle Werte des Datentyps aktuell. Es ist in einer implementierten Integritätsüberwachung praktisch unmöglich, alle aktuellen Belegungen von Datenvariablen in der Integritätsbedingung zu berücksichtigen. Der *Belegungsraum* muß in diesen Fällen geeignet eingeschränkt werden.

Die Einschränkung des Belegungsraum für Objektvariablen ist ebenfalls von großem Interesse für die Überwachungsimplementierung, da für die Überwachung einer Formel mit mehreren freien Objektvariablen mit Prüfschemata ansonsten die Anzahl der zu verwaltenden Markierungen dem Kreuzprodukt der Objektsortenausprägungen entspricht.

Die folgenden Vorschläge für Belegungsraumeinschränkungen sind sowohl für Datenvariablen als auch für Objektvariablen geeignet. Die Einschränkung des Belegungsraumes wirkt sich neben der Verringerung des Überwachungsaufwandes auch bei dem für die Überwachung zusätzlich benötigten Speicherplatz positiv aus und ist deshalb auch für die Methode der integritätsrespektierenden Transaktionen (Abschnitt 6.4) von Interesse.

Beispiel 6.3.1 Für die folgenden Überlegungen wird die Integritätsbedingung $\varphi =$ `gehaltsentwicklung` (Abb. 11, S. 84) als begleitendes Beispiel gewählt. Die Integritätsbedingung φ fordert, daß das Gehalt einer Person bei derselben Firma nur steigen darf.

Die Belegungen der freien Variablen in φ variieren über die Objektsorten `person` und `firma` und den Datentyp `integer` für das Gehalt. Insbesondere der Datentyp `integer` erfordert eine Einschränkung des Belegungsraumes, aber auch die Beschränkung auf diejenigen Kombinationen von Personen und Firmen ist von Vorteil, zwischen denen irgendwann ein Arbeitsverhältnis bestand. □

Unabhängig von dem konkreten Prüfschema kann der Wertebereich einer Variablen oft auf die konkreten Ausprägungen einer bestimmten Attributfunktion eingeschränkt werden. In anderen Fällen, z.B. bei Werten vom Datentyp `datum`, ist bereits bei der Formulierung der dynamischen Integritätsbedingung bekannt, daß eine verwendete Variable nur Werte eines bestimmten Wertebereichs annehmen kann.

Beispiel 6.3.2 Zahlt eine Firma nur Gehälter in z.B. zehn festen Gehaltsstufen aus, so ist eine Beschränkung der Variablen `g` auf diese Werte möglich. Für die Belegungen der Variablen `f` müssen nur Firmen betrachtet werden, die irgendwann `arbeitgeber` einer Person waren. □

Eine Einschränkung der Belegungen einer Variablen auf eine formal festgelegte Untermenge der möglichen Belegungen kann durch zusätzliche Sprachmittel bei der Definition der Integritätsbedingung geschehen. Ein auf der Datenbanksprache SQL basierender Sprachvorschlag für ein relationales Datenbankmanagementsystem kann in [Hü88] gefunden werden.

Die bisherige Einschränkung des Belegungsraumes auf Teilbereiche berücksichtigt nicht den aktuellen Stand der Integritätsüberwachung. In vielen Fällen müssen für konkrete Belegungen bestimmter Variablen nur in wenigen Situationen auch die zugehörigen Markierungen abgespeichert werden. Während des größten Teils der Überwachung ist ihr Wert nicht von Interesse.

Beispiel 6.3.3 In der Beispielintegritätsbedingung $\varphi =$ `gehaltsentwicklung` sind Markierungen für die Kombinationen von Personen und Firmen nur dann relevant für die Integritätsüberwachung, wenn die Person einmal bei der Firma angestellt war. Für jede dieser Kombination muß nur das höchste einmal gezahlte Gehalt als Belegung für die Variable `g` gespeichert werden. □

Ein vielversprechender Ansatz für die Berücksichtigung nur der für die Überwachung relevanten Belegungen ist die Möglichkeit, für bestimmte Variablen auch Belegungen mit Platzhaltern zu betrachten. Eine Belegung mit Platzhaltern wird während der Integritätsüberwachung als Repräsentant für all diejenigen Belegungen genommen, die für die Überwachung nicht relevant sind.

In [Hü88] wird eine Klasse von Integritätsbedingungen vorgestellt, für die der Integritätsmonitor die Variablen, die mit Platzhaltern belegt werden können, bestimmen und Belegungen mit Platzhaltern durch modifizierte Prüfschemata überwachen kann. Die Belegungen mit Platzhaltern werden dort als *implizite Belegungen* bezeichnet.

Die Beispielintegritätsbedingung φ gehört zu dieser Formelklasse. Allgemein gehören all diejenigen Formeln zu dieser Klasse, in denen Variablen in der Prämisse einer Implikation für die weitere Überwachung an aktuelle Werte gebunden werden. Derartige Formeln kommen in den aus der Literatur bekannten Beipielen für dynamische Integritätsbedingungen häufig vor (u.a. in [ISO82, Ku84, ELG84, LS87, Li87]), so daß sich der erhöhte Aufwand für die Integration der Bestimmung und Behandlung impliziter Belegungen in ein integritätsüberwachendes System sicherlich lohnt.

6.3.2 Transaktionsanalyse

Ziel der Transaktionsanalyse ist es, nur diejenigen Belegungen in die Integritätsüberwachung einzubeziehen, für die eine Integritätsverletzung möglich ist. Voraussetzung dafür sind iterationsinvariante Prüfschemata[17] und eine Analyse der Integritätsbedingungen daraufhin, welche Änderungen von Objekten bzw. Objekteigenschaften ihre Gültigkeit beeinträchtigen können.

Die Erfüllung dieser Bedingungen folgt direkt aus einem objektorientierten Datenbankentwurf, bei dem die möglicherweise betroffenen Objekte konkret angeben werden müssen und nur iterationsinvariante Integritätsbedingungen zugelassen sind.

Im allgemeinen Fall müssen die Integritätsbedingungen daraufhin untersucht werden. Für jede Integritätsbedingung wird dabei eine Menge von *kritischen Änderungen* bestimmt, bei denen eine Integritätsverletzung möglicherweise auftreten kann und die somit zu einem Aufruf des Monitors für die betroffenen Objekte führen muß. Im einfachsten Fall werden nur die Objektsorten bestimmt, die für freie Variablen ein-

[17] Einige nichtiterationsinvariante Prüfschemata können ebenfalls berücksichtigt werden, wenn die Transitionsregel bei der Überwachung jeweils solange iteriert wird, bis die Markierung konstant bleibt. Dem entspricht die Vorstellung, daß das beliebige Einschieben einer Transaktion ohne Wirkung keinen Einfluß auf die Entwicklung der Datenbank hat. Allerdings lassen sich Integritätsbedingungen angeben, für die keine stabile Markierung erreicht wird. Offen ist auch, ob dieser 'Trick' nicht der Erwartungshaltung des Datenbankbenutzers an das Systemverhalten widerspricht.

gesetzt werden können, sowie Objektsorten, über die in der Formel prädikatenlogisch quantifiziert wird.

Für jede Transaktion muß nun eine Transaktionsanalyse durchgeführt werden, in der in einer Art Logbuch die Änderungen wie Einfügen von Objekten oder Änderungen von Attributfunktionen eingetragen werden. Aufgrund dieses Logbuchs und der für die Integritätsbedingung kritischen Änderungen werden diejenigen Belegungen ausgewählt, für die eine Integritätsüberprüfung nach dieser Transaktion nötig ist.

Das erstellte Logbuch muß nicht alle konkreten Änderungen explizit aufführen. Für die Überwachung genügt eine Obermenge der betroffenen Änderungen, in der die tatsächlichen Änderungen enthalten sind, um eine deutliche Verringerung des Überwachungsaufwands zu erreichen. Eine einfach zu implementierende Transaktionsanalyse könnte zum Beispiel nur die Objektklassen bestimmen, auf die die Transaktion schreibend zugegriffen hat.

Die Transaktionsanalyse kann sowohl direkt zur Laufzeit der Transaktion durch das DBMS erfolgen als auch bereits während des Entwurfs oder der Implementierung der Transaktion (hier besteht allerdings das Problem, wie aktuelle Aufrufparameter berücksichtigt werden können).

6.4 Integritätsrespektierende Transaktionen

Die Vorteile einer effizienten Integritätsüberwachung mit flexibler Behandlung von Integritätsverletzungen sprechen eindeutig für die Verwendung von integritätsrespektierenden Transaktionen zur Integritätsüberwachung. Ein starkes Argument für diesen Ansatz ist natürlich außerdem die Tatsache, daß existierende Datenbankmanagementsysteme keinen Integritätsmonitor als Systemkomponente enthalten. Dieser Monitor müßte als zusätzliche Schnittstelle zwischen dem DBMS und den anderen Systemkomponenten in Form eines Anwendungsprogramms implementiert werden.

Der Ansatz der integritätsrespektierenden Transaktionen kann aber nur dann für eine korrekte Implementierung mit Erfolg eingesetzt werden, wenn die beiden folgenden Fragestellungen erfolgreich gelöst werden :

1. Gibt es eine formale Methode, mit der eine gegebene Transaktion daraufhin verifiziert werden kann, ob sie alle dynamischen Integritätsbedingungen respektiert ?

2. Gibt es formale Methoden, um eine Transaktion mit vorgegebener Wirkung dahingehend zu modifizieren, daß sie integritätsrespektierend ist ?

Die Verifikation von Transaktionen gegen statische Integritätsbedingungen wird bereits von [GM79] vorgeschlagen. Dort wird eine statische Integritätsbedingung sowohl als Vorbedingung als auch als Nachbedingung einer Transaktion eingesetzt und anschließend die Transaktion gegen diese Spezifikation verifiziert. Die Transaktion ist dabei in einer um Datenbankzugriffe erweiterten konventionellen Programmiersprache implementiert. Bewiesen werden muß somit für statische Integritätsbedingungen, daß die Transaktion einen konsistenten Datenbankzustand, in dem die statische Integritätsbedingung erfüllt ist, wieder in einen konsistenten Datenbankzustand überführt. Für eine dynamische Integritätsbedingung kann in ähnlicher Weise verfahren werden. Berücksichtigt werden muß dabei der aktuelle Stand der Formelauswertung, z.B. in Form der Markierung des reduzierten Prüfschemas, und die zu erfolgende Aktualisierung der zur Integritätsüberwachung benötigten historischen Information.

Für die Verifikation einer Transaktion in bezug auf die dynamischen Integritätsbedingungen muß die bisherige Geschichte der Datenbank berücksichtigt werden. Die Verifikation hat, wie der Integritätsmonitor, die für die Integritätsüberwachung benötigte historische Information zu berücksichtigen und ihre Aktualisierung durch die Transaktion zu garantieren.

Die benötigte historische Information kann in der Spezifikation wieder durch Schemaerweiterungen m_{IB} ausgedrückt werden. Ist dies der Fall, muß verifiziert werden, daß die Transaktion dieselbe Wirkung hat, als wenn sie ohne Änderungen der m_{IB} ausgeführt worden wäre und anschließend der Algorithmus `MONITOR` aufgerufen worden wäre. Die Transaktion muß also die Kantenbeschriftungen der Prüfschemata erfüllen und, wenn nötig, die Markierungen aktualisieren.

Die Integration der Wirkung des Integritätsmonitors in eine Spezifikation mit Vor- und Nachbedingungen führt direkt zu folgender Definition einer zusätzlichen Transaktionsspezifikation. Betrachtet werden Transaktionen $t(p)$ mit Parametern p.

Definition 6.4.1 Eine Transaktion $t(p)$ *respektiert* eine dynamische Integritätsbedingung *IB* mit reduziertem Prüfschema $RT_{IB} = (G, \nu, \eta, m_0, F)$, wenn sie die folgenden zusätzlichen Spezifikationen erfüllt.

Das Datenbankschema wurde um die Knotenmarkierung m_{IB}, wie in Abschnitt 6.2 beschrieben, erweitert. Die Transitionsregel sei durch einen Term des verwendeten Prädikatenkalküls ausgedrückt, der als Ergebnis die berechnete neue Markierung hat.

$\sigma_{t(p)}$ bezeichnet einen beliebigen Datenbankzustand nach Anwendung von $t(p)$ auf einen Datenbankzustand, der das Ergebnis einer bisher korrekten Datenbankentwicklung ist und deshalb ausschließlich nichtleere Markierungen m_{IB} enthält.

1. Für alle Belegungen β der Variablen in *IB* mit neu eingefügten Datenbankob-

jekten muß gelten

$$\{\mathbf{true}\}\ t(p)\ \{m_{IB}(\beta) \neq \emptyset\ \wedge\ m_{IB} = \mathrm{trans}_{IB}(m_0, \sigma_{t(p)}, \beta)\}.$$

Diese Spezifikation erzwingt die korrekte Initialisierung der Integritätsüberwachung für neu eingefügte Objekte.

2. Für alle Belegungen β der Variablen in *IB* mit geänderten Datenbankobjekten muß gelten

$$\{m = m_{IB}(\beta)\}\ t(p)\ \{m_{IB}(\beta) \neq \emptyset\ \wedge\ m_{IB} = \mathrm{trans}_{IB}(m, \sigma_{t(p)}, \beta)\}.$$

Die Überprüfung der Kantenbeschriftungen auf Erfülltheit, die den Kern der Integritätsüberwachung ausmacht, ist in der Berechnungsvorschrift der Transitionsregel enthalten. Als zusätzliche Vorbedingung z.B. für Vereinfachungen kann die Disjunktion der Kantenbeschriftungen $\eta((k,k))$ für $k \in m_{IB}(\beta)$ herangezogen werden.

3. Für alle Belegungen β der Variablen in *IB* mit gelöschten Datenbankobjekten muß gelten

$$\{m_{IB}(\beta) \cap F = \emptyset\}\ t(p)\ \{\mathbf{false}\}.$$

Eine Löschung darf nur vorgenommen werden, wenn die betroffenen Markierungen finale Knoten enthalten. □

Für die Verifikation müssen die Transitionsregel und der Test auf eine leere Markierung durch logische Aussagen ersetzt werden. Dies kann durch eine Fallunterscheidung für die Knotenmarkierungen und die von ihnen ausgehenden Kanten erfolgen. Auch eine Berechnungsvorschrift für Markierungen ist anstelle der Markierung m_0 einsetzbar. Der Zugriff auf den neuen Datenbankzustand muß dabei aufgelöst werden in Zugriffe auf die für die Überwachung relevanten, durch Prädikate beschriebenen, Zustandsinformationen.

Die Verifikation einer Transaktion gegen eine derartige Spezifikation ist nur möglich, wenn eine formale Semantik für die Implementierungssprache festgelegt ist, die einen formalen Beweis der Spezifikationserfüllung ermöglicht. Die Transaktionssprachen sind stark abhängig vom zur Implementierung benutzten Datenmodell und vom verwendeten DBMS. Für viele existierende Systeme ist deshalb die Anwendung formaler Beweise aufgrund fehlender formaler Semantik nicht möglich, so daß keine formale Verifikation von implementierten Transaktionen für diese Systeme durchgeführt werden kann.

Dieses Problem ist zum Teil dadurch lösbar, daß die Verifikation in früheren Entwurfsphasen vorgenommen bzw. vorbereitet wird, für die eine Spezifikationssprache mit formaler Semantik benutzt wird. Für die Phase des konzeptionellen Entwurfs entspricht dies der Eigenschaft, daß die Aktionenspezifikation die dynamischen Integritätsbedingungen erzwingt (Def. 5.5.4 von Seite 101). Es bleibt allerdings das Problem, die implementierten Transaktionen gegen die Aktionsspezifikation zu verifizieren.

Bereits in Abschnitt 5.5 wurde erwähnt, daß die direkte Berücksichtigung aller dynamischen Integritätsbedingungen beim konzeptionellen Entwurf von Aktionen zu Spezifikationen führen kann, in denen die eigentliche Spezifikation der Aktionswirkung aufgrund der integritätserzwingenden Spezifikationsteile nicht mehr deutlich wird. Auch wurde bisher nichts über eine Methode zur Erstellung integritätserzwingender Aktionsspezifikationen gesagt.

Wünschenswert ist eine Entwurfsmethode, die den Entwerfer eines konzeptionellen Datenbankschemas nur die Spezifikation der beabsichtigten Aktionswirkung erstellen läßt, und die Aktionenspezifikation anschließend in eine integritätserzwingende Aktionenspezifikation transformiert. Eine derartige Transformation wurde erstmals in [Li85, Li86] für eine eingeschränkte Klasse von temporalen Formeln vorgeschlagen. In [Li87] wurde dieses Verfahren auf beliebige Formeln einer linearen temporalen Logik ohne Variablenquantifizierungen erweitert.

Die Transformation basiert ebenfalls auf der Konstruktion von Prüfschemata aus temporalen Formeln. Ausgehend von einer konzeptionellen Spezifikation der beabsichtigten Aktionswirkung wird dafür die Aktionenspezifikation schrittweise um die Kantenbeschriftungen des Prüfschemas als zusätzliche Transaktionenspezifikation erweitert und anschließend vereinfacht. Wie bei der Verifikation muß auch hier die Aktualisierung der historischen Überwachungsinformationen in die Spezifikation integriert werden. Einzelheiten der Transformationen können in [Li86, Li87] gefunden werden.

7 Ausblick

Die in dieser Arbeit vorgestellte deskriptive Spezifikation von Objektlebensläufen in finit-linearer temporaler Logik ermöglicht durch die Konstruktion von Prüfschemata neben Konsistenzprüfungen auch eine direkte Implementierung der Objektdynamik. Die finit-lineare temporale Logik kann als Teil einer mehrschichtigen Familie von Spezifikationssprachen für Datenbank-Anwendungen eingesetzt werden.

Die deskriptiven temporalen Beschreibungen können in Prüfschemata für die dynamischen Integritätsbedingungen umgeformt werden, die von einer Komponente der implementierten Datenbank-Anwendung zur Integritätsüberwachung benutzt werden. Um eine effiziente Überwachung der potentiellen Gültigkeit temporaler Formeln zu ermöglichen, müssen insbesondere für die Vorschläge für eine Reduktion für Belegungsklassen (Abschnitt 4.5) und für die Überwachung impliziter Belegungen (Abschnitt 6.3.1) die theoretischen und algorithmischen Grundlagen weiter untersucht werden.

Neben der Spezifikation von Datenbank-Anwendungen kann die finit-lineare temporale Logik auch in den anderen Gebieten der Informatik verwendet werden, in denen die lineare temporale Logik nach Manna/Pnueli bisher mit Erfolg eingesetzt wurde. Beispiele hierfür sind die Spezifikation und Verifikation von Programmen [MP79, MP81, Mo86, Krö87] und parallelen Prozessen [MW84] oder die Wissensmodellierung und das zeitliche Schlußfolgern in Bereichen der künstlichen Intelligenz [MH69, Schw84, SK85].

Die Vorteile der vorgestellten finit-linearen temporalen Logik im Vergleich zu den aus der Literatur bekannten linearen temporalen Logiken sind dabei

- die Erweiterung auf eine temporale Prädikatenlogik mit einer intuitiv einleuchtenden Behandlung von Formeln mit freien Variablen,
- damit zusammenhängend die Möglichkeit der einfachen und erweiterten Reduktquantifizierung,
- eine formale Semantik für abbrechende Zustandsfolgen
- und somit die Möglichkeit, Systeme von sich verändernden Populationen dynamischer Objekte zu beschreiben.

Diese Vorteile sind nicht nur für die formale Beschreibung von Datenbank-Anwendungen interessant, sondern ermöglichen auch die Spezifikation von objektorientierten Programmen und verteilten Prozessen. Bei Einsatz der finit-linearen temporalen Logik können Eigenschaften von dynamischen Objekten oder Prozessen betrachtet werden, die während der Laufzeit erzeugt und möglicherweise terminieren und danach als gelöscht betrachtet werden.

Die konstruierten Prüfschemata können bei derartigen Prozessen als Ablaufstruktur einer möglichen Implementierung interpretiert werden [CE81, Wo82] und z.B. auf Lebendigkeitseigenschaften oder Terminierung untersucht werden [MW84].

Die vorgestellte Semantik der Spezifikation von Datenbank-Anwendungen liefert eine mathematische Modellbildung für ein Struktur und Dynamik beschreibendes Datenbankschema. Es bleiben allerdings einige Fragen offen, die für den Datenbankentwurf und für die semantische Modellbildung einer implementierten Datenbank-Anwendung von Interesse sind.

In der vorgestellten mehrschichtigen Datenbankspezifikation entspricht ein Datenbankzustand einem Modell der durch die Schemasignatur festgelegten Prädikatenlogik. Von Vorteil wäre eine Erweiterung um Möglichkeiten, auch disjunktives Wissen und Ableitungsregeln berücksichtigen zu können. Diese Erweiterungen führen zum sogenannten *Theorieansatz* für Datenbankzustände, in dem ein Datenbankzustand einer Menge von prädikatenlogischen Modellen entspricht [GMN84]. Ein Datenbankzustand ist dann durch die Theorie einer Menge von gespeicherten Formeln festgelegt (vergl. Abschnitt 2.1).

Um diesen Ansatz auf Datenbankentwicklungen erweitern zu können, müssen abgewandelte Kripkestrukturen betrachtet werden, in denen als Weltinterpretationen prädikatenlogische Theorien statt Modelle betrachtet werden.[18] Offen ist dabei sowohl die Semantik einer Entwicklungsspezifikation wie auch die formale Beschreibung von Änderungen des Datenbestandes.

Eine weitere offene Frage ist die Berücksichtigung von Schemaänderungen während der Datenbanklaufzeit in der semantischen Modellbildung. Die der Datenbankentwicklung zugeordnete temporale Struktur müßte ab diesem Zeitpunkt Datenbankzustände einer neuen Signatur haben, wobei der Inhalt des ersten neuen Datenbankzustands aber durch den vorherigen Datenbankzustand (zur alten Signatur) eindeutig festgelegt ist. Der bisherige formale Rahmen bietet keine Erweiterungsmöglichkeiten zur Behandlung von Schemaänderungen.

Für den Teilbereich der Überwachung dynamischer Integritätsbedingungen bietet die in Abschnitt 4.6 beschriebene Algebra für die aussagenlogischen Verknüpfung von Prüfschemata einen ersten Ansatz, Änderungen der Menge von dynamischen

[18] Diese abgewandelten Kripkestrukturen dürfen nicht mit der Theorie einer Menge von modallogischen bzw. temporalen Formeln verwechselt werden. Eine derartige Theorie bestimmt die Menge von Kripkestrukturen, die die Formeln erfüllen — ein für die Semantik von einzelnen Datenbankentwicklungen sicher ungeeigneter Ansatz.

Integritätsbedingungen während der Laufzeit zu berücksichtigen. Alle drei dort beschriebenen Verfahren ermöglichen es, die Markierungen (also die historische Überwachungsinformation) einer bereits überwachten Integritätsbedingung in der Überwachung einer erweiterten bzw. modifizierten neuen Integritätsbedingung weiterhin zu berücksichtigen.

Für die Semantik der Änderungsaktionen wurde ein einfaches formales Modell gewählt, daß für die Spezifikation komplexer Systeme um mächtigere Konzepte erweitert werden muß. Denkbar sind Erweiterungen auf verteilte Prozesse bzw. Petrinetze oder auf hierarchisch strukturierte komplexe Transaktionen. Für diese Erweiterungen fehlen bisher noch die semantische Grundlagen im Verhältnis zu den vorgestellten Konzepten.

Für die Unterstützung des konzeptionellen Entwurfs von Datenbank-Anendungen ist die Entwicklung entwurfsunterstützender Werkzeuge und methodischer Vorgehensweisen notwendig [EHNSE88]. Neben der Unterstützung eines strukturierten objektorientierten Entwurfs der beschriebenen vier Spezifikationsschichten sind auch geeignete Beschreibungsverfahren für komplexe Anwendungstransaktionen zu entwickeln, die möglicherweise eine große Anzahl von Objekten mit komplexer Wirkung manipulieren. Offen ist bisher auch die Einbindung von konzeptionellen Beschreibungen weiterer Eigenschaften von Datenbank-Anwendungen wie z.B. realzeitspezifische Aspekte oder die Verwaltung von Benutzerrechten.

Um die Eignung der Implementierungsvorschläge aus Kapitel 6 für die Implementierung von Datenbank-Anwendungen beurteilen zu können, ist eine weitere Verfeinerung der integritätsüberwachenden Algorithmen und eine anschließende Prototyp-Implementierung für ein existierendes Datenbankmanagementsystem erforderlich. Dieses System muß anhand realistischer Anwendungen ausgetestet werden. Geplant ist die Implementierung eines Integritätsmonitors für ein relationales System [EHNSE88, Hü88].

Gleichzeitig muß die Ausdrucksfähigkeit der Spezifikation in temporaler Logik und ihre Akzeptanz durch den Systementwerfer und — im Falle des Integritätsmonitors — den Systembenutzer kritisch untersucht werden. Hierzu sind Beispielentwürfe und Implementierungen für reale Anwendungen notwendig.

Literatur

[ADD85] Albano, A., DeAntonellis, V.,DiLeva, A.(eds.): Computer–Aided Database Design : The DATAID Projekt. North–Holland Amsterdam 1985.

[AM85] Abadi, M., Manna, Z.: Nonclausal Temporal Deduction. *Proc. Logics of Programs, Brooklyn (R.Parikh, ed.).* LNCS 193, Springer–Verlag 1985, **1–15**.

[BCG87] Browne, M.C., Clarke, E.M., Grümberg, O.: Characterizing Kripke Structures in Temporal Logic. *Proc. Int. Joint Conf. on Theory and Practice of Software Development TAPSOFT'87 (H.Ehrig, et al., eds.).* Pisa, LNCS 249, Springer–Verlag, Berlin 1987, **256–270**.

[BDRZ84] Brägger, R.P., Dudler, A., Rebsamen, J., Zehnder, C.A.: Gambit : An Interactive Database Design Tool for Data Structures, Integrity Constraints, and Transactions. *Proc. 1st Int. Conf. on Data Engineering.* Los Angeles 1984, **339–407**.

[BMP81] Ben–Ari, M., Manna, Z., Pnueli, A.: The Temporal Logic of Branching Time. *Proc. 8th ACM Symp. on Principles of Programming Languages.* New York 1981, **164–176**.

[BN77] Bergmann, E., Noll, H.: Mathematische Logik mit Informatik–Anwendungen. Springer–Verlag, Berlin 1977.

[CCF82] de Castilho, J.M.V., Casanova, M.A., Furtado, A.L.: A Temporal Framework for Database Specification. *Proc. Int. Conf. on Very Large Databases.* Mexico City 1982, **280–291**.

[Ce83] Ceri, S.: Methodology and Tools for Data Base Design. North–Holland Amsterdam 1983.

[CE81] Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching–Time Temporal Logic. *Proc. Workshop Logics of Programs (D.Kozen, ed.).* LNCS 131, Springer–Verlag, Berlin 1981, **52–71**.

[CES86] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems.* Vol. 8, No. 2, 1986, **244–263**.

[CF84] Cavalli, A.R., Fariñas del Cerro, L.: A Decision Method for Linear Temporal Logic. *7th International Conf. on Automated Deduction (R.E.Shostak, ed.).* LNCS 170, Springer–Verlag, Berlin 1984, **113–127**

[Ch80] Chellas, B.F.: Modal Logic : An Introduction. Cambridge Univ. Press 1980.

[CHJ86] Cohen, B., Harwood, W.T., Jackson, M.I.: The Specification of Complex Systems. Addison–Wesley, Wokingham (England) 1986.

[CS87] Carmo, J., Sernadas, A.: A Temporal Logic Framework for a Layered Approach to Systems Specification and Verification. *Proc. IFIP WG 8.1 Conf. on "Temporal Aspects of Information Systems"*. Sophia–Antipolis 1987, **31–46**.

[EDG86] Ehrich, H.-D., Drosten, K., Gogolla, M.: Towards an Algebraic Semantics for Database Specification. *Proc. IFIP Work. Conf. on Knowledge and Data "DS–2"*. Albufeira (Portugal) 1986.

[Eh85] Ehrich, H.-D.: Spezifikation konzeptioneller Schemata mit abstrakten Datentypen und Versionen. *Fachgespräch "Entwurf von Informationssystemen — Methoden und Modelle" (H.C.Mayr, B.E.Meyer, Hrsg.)*. Ev. Akad. Tutzing 1985, **1–19**.

[Eh86] Ehrich, H.-D.: Key Extensions of Abstract Data Types, Final Algebras and Database Semantics. *Proc. Workshop on Category Theory and Computer Programming (D.Pitt et al, eds.)*. Springer–Verlag, Berlin 1986, **412–433**.

[EHNSE88] Engels, G., Hohenstein, U., Neugebauer, L., Saake,G., Ehrich, H.-D.: Konzeption einer integrierten Datenbank–Entwurfsumgebung. *Proc. Data Dictionaries und Entwicklungswerkzeuge für Datenbank–Anwendungen*. Zürich 1988.

[ELG84] Ehrich, H.-D., Lipeck, U.W., Gogolla, M.: Specification, Semantics and Enforcement of Dynamic Database Constraints. *Proc. Int. Conf. on Very Large Databases*. Singapore 1984, **301–308**.

[Em83] Emerson, E.A.: Alternative Semantics for Temporal Logics. *Theoretical Computer Science*. 26, 1983 **121–130**.

[EM85] Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1. Equations and Initial Semantics. Springer–Verlag, Berlin 1985.

[ES85] Emerson, E.A., Sistla, A.P.: Deciding Full Branching Time Logic. *Information & Control*. Vol. 61, No. 3, 1985, **175–201**.

[ESS88] Ehrich, H.–D., Sernadas, A., Sernadas, C.: Semantics of Object–Oriented Databases : Complex Objects. *zur Veröffentlichung eingereicht*. 1988.

[FL87] Feng, D., Lipeck, U.W.: Deterministische Überwachung temporaler Formeln. *Informatik–Bericht Nr. 87–06*. Technische Universität Braunschweig, 1987.

[FS86] Fiadeiro, J., Sernadas, A.: The INFOLOG Linear Tense Propositional Logic of Events and Transactions. *Information Systems*. Vol. 11, No. 1, 1986, **61–85**.

[GM79] Gardarin, G., Melkanoff, M.: Proving Consistency of Database Transactions. *Proc. Int. Conf. on Very Large Databases*. Rio de Janeiro 1979, **291–299**.

[GMN84] Gallaire, H., Minker, J., Nicolas, J.-M.: Logic and Databases : A Deductive Approach. *Computing Surveys*. Vol. 16, No. 2, 1984, **153–185**.

[GMS83] Golshani, F., Maibaum, T., Sadler, M.: A Modal System of Algebras for Database Specification and Query/Update Language Support. *Proc. Int. Conf. on Very Large Databases.* Florence 1983, **331–339**.

[GOT83] Glubrecht, J.–M., Oberschelp, A., Todt, G.: Klassenlogik. Bibliographisches Institut, Mannheim 1983.

[HC68] Hughes, G.E., Cresswell, M.G.: An Introduction to Modal Logic. Methuen and Co Ltd. , London 1968.

[HNS86] Hohenstein, U., Neugebauer, L., Saake, G.: An Extended Entity-Relationship Model for Non–Standard Databases. *Proc. "Workshop über Relationale Datenbanken" (A.Heuer, ed.).* Lessach, Bericht Technische Univ. Clausthal–Zellerfeld Nr. 86–3, **185–211**.

[HNSE87] Hohenstein, U., Neugebauer, L., Saake, G., Ehrich, H.-D.: Three Level Specification of Databases Using an Extended Entity Relationship Model. *GI/GI/SI–Fachtagung "Entwurf von Informationssystemen, Informationsbedarfsermittlung und -analyse", Linz (R.R.Wagner, R.Traunmüller,H.C.Mayr, eds.).* Informatik–Fachbericht Bd. 143, Springer–Verlag, Berlin 1987, **58–88**.

[Ho85] Hoare, C.A.R.: Communicating Sequential Processes. Prentice–Hall, Englewood Cliffs 1985.

[Hü88] Hülsmann, K.: Entwurf eines Systems zur Überwachung dynamischer Integritätsbedingungen. *Diplomarbeit.* Techn. Universität Braunschweig 1988.

[ISO82] ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and the Information Base. *J.J. van Griethuysen, ed.* ISO, New York 1982.

[KMS85] Khosla, S., Maibaum, T.S.E., Sadler, M.: Database Specification. *Proc. IFIP Conf. on Database Semantics (DS–1) 1985 (T.B.Steel, R.Meersmann, eds.).* North Holland, Amsterdam 1986 **141–158**.

[Kri63a] Kripke, S.A.: Semantical Analysis of Modal Logic I — Normal Modal Propositional Calculi. *Z. f. math. Logik und Grundl. d. Mathematik.* Bd. 9, 1963, **67–96**.

[Kri63b] Kripke, S.A.: Semantical Considerations on Modal Logic. *Acta Philosophica Fennica – Modal and Many-valued Logics.* Helsinki 1963, **83–94**.

[Krö87] Kröger, F.: Temporal Logic of Programs. Springer–Verlag, Berlin 1987.

[Ku84] Kung, C.H.: A Temporal Framework for Database Specification and Verification. *Proc. Int. Conf. on Very Large Databases.* Singapore 1984, **91–99**.

[Ku85] Kung, C.H.: On Verification of Database Temporal Constraints. *Proc. ACM SIGMOD Int. Conf. on Management of Data.* Austin 1985, **169–179**.

[La80] Lamport, L.: "Sometime" is sometimes "not Never" — On the Temporal Logics of Programs. *7th ACM Symp. on Principles of Programming Languages.* 1980, **174–185**.

[LEG85] Lipeck, U.W., Ehrich, H.-D., Gogolla, M.: Specifying Admissibility of Dynamic Database Behaviour Using Temporal Logic. *Proc. IFIP Work. Conf. on Theoretical and Formal Aspects of Information Systems (A.Sernadas et al., eds.)* North–Holland, Amsterdam 1985, **145–157**.

[Li85] Lipeck, U.W.: Schrittweise Spezifikation des dynamischen Verhaltens von Datenbanken. *Proc. Fachgespräch "Entwurf von Informationssystemen — Methoden und Modelle" (H.C.Mayr, B.E.Meyer, Hrsg.).* Ev. Akad. Tutzing 1985, **20–38**.

[Li86] Lipeck, U.W.: Stepwise Specification of Dynamic Database Behaviour. *Proc. Int. ACM–SIGMOD Conf. on Management of Data (C.Zaniolo, ed.).* Washington 1986, **387–397**.

[Li87] Lipeck, U.W.: Zur dynamischen Integrität von Datenbanken : Grundlagen der Spezifikation und Überwachung. *Habilitationsschrift (eingereicht).* Braunschweig 1987.

[Ll84] Lloyd, J.W.: Foundations of Logic Programming. Springer–Verlag, Berlin 1984.

[LSE86] Lipeck, U.W., Saake, G., Ehrich, H.-D.: Monitoring Dynamic Database Integrity by Transition Graphs. *Bericht Nr. 86–08.* Technische Universität Braunschweig, 1986.

[LS87] Lipeck, U.W., Saake, G.: Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems.* Vol. 12, No. 3, 1987, **255–269**.

[MDL87] Mayr, H.C., Dittrich, K.R., Lockemann, P.C.: Datenbankentwurf. *Datenbank–Handbuch. (P.C.Lockemann, J.W.Schmidt, eds.).* Springer–Verlag Berlin 1987. **481–557**

[Me87] Meyer, B.: Graphische Entwurfsumgebung für Transitionsgraphen. *Diplomarbeit.* Technische Universität Braunschweig 1987.

[MH69] McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence 4 (B.Meltzer, D.Michie, eds.).* University Press, Edinburgh 1969, **463–502**.

[Mo86] Moszkowski, B.: Executing Temporal Logic Programs. Cambridge University Press, Cambridge 1986.

[MP79] Manna, Z., Pnueli, A.: The Modal Logic of Programs. *6th Colloguium on Automata, Languages and Programming (H.A.Maurer, ed.).* LNCS 71, Springer–Verlag, Berlin 1979, **385–409**.

[MP81] Manna, Z., Pnueli, A.: Verification of Concurrent Programs : The Temporal Framework. *The Correctness Problem in Computer Science (R.S.Boyer, J.S.Moore, eds.)*. Academic Press, London 1981, **215–273**.

[MW81] Manna, Z., Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications. *Proc. Workshop Logics of Programs (D.Kozen, ed.)*. LNCS 131, Springer–Verlag, Berlin 1981, **253–281**.

[MW84] Manna, Z., Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*. Vol. 6, 1984, **68–93**.

[Pn86] Pnueli, A.: Specification and Development of Reactive Systems. *Information Processing '86 (H.-J.Kugler, ed.)*. North–Holland, Amsterdam 1986, **845–858**.

[Pr67] Prior, A.: Past, Present and Future. Oxford University Press 1967.

[RD82] Richter, G., Durchholz, R.: IML–Inscribed Petri–Nets. *Proc. IFIP Work. Conf. on Comparative Review of Information Systems Design Methodologies (T.W.Olle, A.A.Verijn–Stuart, eds.)*. North–Holland, Amsterdam 1982, **335–368**.

[Rei85] Reisig, W.: Petri Nets. Springer–Verlag, Berlin 1985.

[Reu87] Reuter, A.: Maßnahmen zur Wahrung von Sicherheits- und Integritätsbedingungen. *Datenbank–Handbuch. (P.C.Lockemann, J.W.Schmidt, eds.)*. Springer–Verlag Berlin 1987. **337–479**

[RU71] Rescher, W., Urquhart, A.: Temporal Logic. *Library of Exact Philosophy 3*. Springer–Verlag, Wien 1971.

[Saa85] Saake, G.: Konstruktion von Transitionsgraphen aus Temporalen Formeln zur Integritätsüberwachung in Datenbanken. *Diplomarbeit*. Technische Universität Braunschweig, 1985.

[Sal78] Salomaa, A.K.: Formale Sprachen. Springer–Verlag, Berlin 1978.

[Schm87] Schmidt, C.: Einsatz von Petri–Netzen beim Entwurf von Informationssystemen. *Diplomarbeit*. Technische Universität Braunschweig 1987.

[Schu87] Schurgel, A.: Implementierung eines Algorithmus zur Konstruktion von Transitionsgraphen aus temporalen Formeln. *Diplomarbeit*. Technische Universität Braunschweig 1987.

[Schw84] Schwind, C.B.: Temporal Logic in Artificial Intelligence. *8th German Workshop on Artificial Intelligence GWAI–84, (J.Laubsch, ed.)*. Springer–Verlag, Berlin 1984, **238–264**.

[Se80] Sernadas, A.: Temporal Aspects of Logical Procedure Definition. *Information Systems*. Vol. 5, 1980, **167–187**.

[SFNC84] Schiel, U., Furtado, A.L., Neuhold, E.J., Casanova, M.A.: Towards Multi–Level and Modular Conceptual Schema Specifications. *Information Systems*. Vol. 9, 1984, **43–57**.

[SFSE88] Sernadas, A., Fiadeiro, J., Sernadas, C., Ehrich, H.–D.: Abstract Object Types : A Temporal Perspective. *in Vorbereitung*. 1988.

[SL87] Saake, G., Lipeck, U.W.: Foundations of Temporal Integrity Monitoring. *Proc. IFIP WG 8.1 Conf. on "Temporal Aspects of Information Systems"*. Sophia–Antipolis 1987, **239–253**.

[SK85] Solvberg, A., Kung, C.H.: On Structural and Behavioral Modeling of Reality. *Proc. IFIP Conf. on Database Semantics (DS–1) 1985 (T.B.Steel, R.Meersmann, eds.)*. North Holland, Amsterdam 1986 **205–221**.

[Sm68] Smullyan, R.M.: First Order Logic. Springer–Verlag, New York 1968.

[SMV83] Schwartz, R.L., Melliar–Smih, P.M., Vogt, F.H.: An Interval Logic for Higher–Level Temporal Reasoning. *Proc. 2nd ACM Symp. on Princ. of Distributed Computing*. Montreal 1983, **173–186**.

[SNHE87] Saake, G., Neugebauer, L., Hohenstein, U., Ehrich, H.–D.: Konzepte und Werkzeuge für eine Datenbank–Entwurfsumgebung. *Informatik–Bericht Nr. 87–05*. Technische Universität Braunschweig, 1987.

[SSE87] Sernadas, A., Sernadas, C., Ehrich, H.–D.: Object–Oriented Specification of Databases : An Algebraic Approach. *Proc. Int. Conf. on Very Large Databases*. Brighton 1987.

[Sz87] Szalas, A.: A Complete Axiomatic Characterization of First–Order Temporal Logic of Linear Time. *Theoretical Computer Science*. 54, 1987, **199–214**.

[Ve85] Venkatesh, G.: A Decision Method for Temporal Logic Based on Resolution. *Proc. Found. of Software Technology and Theoretical Computer Science. New Delhi (S.N.Maheshwari, ed.)*. Springer–Verlag, 1985, **272–289**.

[VW86] Vardi, M.Y., Wolper, P.: Automata–Theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Sciences*. Vol. 32, 1986, **183–221**.

[Wo81] Wolper, P.: Temporal Logic can be more expressive. *Proc. of the 22th IEEE Symp. on Found. of Computer Science*. Nashville, Tenn., 1981, **340–348**.

[Wo82] Wolper, P.: Specification and Synthesis of Communicating Processes Using an Extended Temporal Logic. *Proc. of the 9th ACM Symposium on Principles of Programming Languages*. Albuquerque, New Mexico 1982, **20–33**.

Verzeichnis der Abbildungen

Index

Abkürzungen

Index