Interner Bericht

E I T e L –
ein intervallbasierter
temporallogischer Ansatz

Klaus Spies und Otto Mayer

März 1992
219/92

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

E I T e L –
ein intervallbasierter temporallogischer Ansatz

Klaus Spies und Otto Mayer

März 1992
219/92

AG Grundlagen der Programmierung
Leiter: Prof. Dr. O. Mayer

1 Einführung

In vielen Bereichen der Informatik spielt Zeit und ihre Modellierung eine wichtige Rolle. Stellvertretend für alle anderen Bereiche seien nur folgende genannt:

Bei **Datenbankanfragen** treten in natürlicher Weise zeitabhängige Probleme auf. ("Ich möchte eine Liste aller Angestellten, die eingestellt wurden *bevor* ..., und deren Gehalt sich erhöhte *während* Dr. Schmitt Abteilungsleiter war.")

Bei der **Analyse bzw. Verifikation digitaler Schaltwerke** muß z.B. entschieden werden, welche Input- und Output-Werte die einzelnen Komponenten vor und nach bestimmten Signalen haben.

Bei der **Programmverifikation** betrachtet man Zusicherungen vor und nach Ausführung eines Statements. Dabei tritt bei der Behandlung paralleler Prozesse zusätzlich das Problem der Gleichzeitigkeit auf.

Innerhalb der **KI** tritt das Problem des Umgangs mit Zeit häufig dort auf, wo Vorgänge des täglichen Lebens nachgebildet werden sollen.

Beispiele hierfür sind:

- natürlichsprachliche Systeme, Zeitverlaufsformen (Tenses) in der Sprache ("*Bevor* ich zu Bett ging rauchte ich noch eine Zigarette *während* ich die Zeitung durchblätterte.")
- Expertensysteme mit zeitabhängigen Regeln in der Wissensbasis (" Wenn die kritische Temperatur *länger* als 10 Sekunden überschritten wird ... ")
- Plan-erzeugende Systeme (z.B. "Blocks-World")

Aus obigen Beispielen ergibt sich die Notwendigkeit und Wichtigkeit von Kalkülen für den Umgang mit Zeit. Folgerichtig sind in den letzten Jahren auf allen genannten Gebieten eine Vielzahl von Arbeiten erschienen. Dabei wurden auch verstärkt Anstrengungen unternommen, (logische) Programmiersprachen zu entwickeln, die die Formulierung und Behandlung zeitspezifischer Probleme unterstützen. Die verschiedenen Ansätze unterscheiden sich dabei zunächst einmal in der Art und Weise, wie Zeit betrachtet wird. Je nach Philosophie und Pragmatik betrachtet das zugrundeliegende Zeitmodell diese als linear oder verzweigend, diskret oder dicht, endlich oder unendlich, sich nur in die Zukunft oder auch in die Vergangenheit ausdehnend etc.

Ein weiteres wichtiges Unterscheidungsmerkmal der verschiedenen Ansätze ist ihre jeweilige Stellung zur "klassischen" logischen Programmierung (Prolog). Grundlegend für die logische Programmierung ist dabei die Erkenntnis, daß sich Formeln der Prädikatenlogik erster Stufe als Anweisungen eines Programms interpretieren lassen. Dadurch wird es möglich, Prädikatenlogik erster Stufe direkt als Programmiersprache zu verwenden. Insbesondere Kowalski hat durch seine prozedurale Interpretation von Logikprogrammen wesentlich zum Durchbruch der logischen Programmierung beigetragen ([4]). Auf ihn geht auch die Formel $Algorithm = Logik + Control$ zurück ([5]), die einfach gesagt, ausdrückt, daß ein Algorithmus grundsätzlich aus einer Logikkomponente besteht, die bestimmt, *was* getan werden

muß, und aus einer Kontrollkomponente, die bestimmt, *wie* dies getan wird. Während bei herkömmlichen (imperativen) Programmiersprachen diese beiden Komponenten eine Einheit bilden, beschreiben Logikprogramme nur die logische Komponente eines Algorithmus. Die oben erwähnte prozedurale Interpretation von Formeln ermöglicht es, die Kontrolle (Ausführung des Programms) dem Interpreter zu überlassen. Bei der Behandlung von Problemen, bei denen Zeit modelliert werden muß, tritt hier nun die Schwierigkeit auf, daß es in der Prädikatenlogik erster Stufe nur die Möglichkeit gibt, Zeit durch Parametrisierung zu modellieren. Dies führt einerseits oft zu erheblichen Komplexitätsproblemen, andererseits scheint dieser Ansatz für viele Problembereiche recht unnatürlich zu sein. (Man denke hier z.B. an Zeitverlaufsformen in natürlichen Sprachen.) Bei der Überwindung dieser Schwierigkeiten kann man nun zwei Vorgehensweisen unterscheiden. Einerseits wird versucht, Prolog-nahe Systeme, d.h. Systeme mit Prolog-ähnlicher, um Temporal-Operatoren erweiterter Syntax und Prolog-ähnlicher Programmabarbeitung (modifizierte Resolution, Unifikation), oder gar nur um temporale Operatoren erweiterte Prolog-Systeme zu entwickeln (vgl. z.B. [1], [2], [3], [6], [9]). Andererseits kann man versuchen, einer geeigneten Temporallogik (bzw. einer bestimmten Teilmenge davon) unmittelbar, ohne den "Umweg" über Prolog und die dabei verwendeten Abarbeitungsprinzipien, eine eigene, der temporalen Natur der Logik besser angepaßte operationale Semantik zu geben, die unter Umständen völlig anders geartet ist als die oben angesprochenen klassischen Prinzipien, die Prolog und den daraus abgeleiteten Systemen zugrunde liegen ([7]). Diese zweite Richtung wird denn auch von uns verfolgt.

Die grundlegende Idee stammt dabei von Moszkowski [5] , der mit der Sprache Tempura auch die erste auf dieser Idee basierende Sprache entwickelt hat. Tempura und die dieser Sprache zugrundeliegende Idee soll hier kurz beschrieben werden.

Tempura basiert auf der sogenannten Intervall-Temporallogik (ITL). Diese ist eine Modallogik und somit eine um bestimmte modale Operationen wie z.B. $\Box$ (always) und $\circ$ (next) erweiterte klassische Prädikatenlogik. ITL-Formeln werden auf endlichen Folgen von Zuständen (Intervallen) interpretiert. Die modalen Operatoren verbinden dabei Aussagen über Intervalle mit Aussagen über deren Teilintervalle. So bedeutet z.B. $\Box$ "auf allen Teilintervallen", $\circ$ "auf dem nächsten Teilintervall". Eine ganz wesentliche Besonderheit von ITL ist in diesem Zusammenhang der sogenannte Chop-Operator (;), ein binärer Operator, der die sequentielle Komposition von temporalen Formeln bezeichnet. Durch "chop" lassen sich die verschiedensten sequentiellen Konstrukte wie z.B. For- und While-Schleifen als temporallogische Formeln über Zeitintervallen ausdrücken. Chop ist ein sehr starkes Konzept (es lassen sich alle anderen temporalen Operatoren in ITL auf chop und next zurückführen), das in dieser Form bisher in keiner anderen Temporallogik auftritt. Zu erwähnen ist an dieser Stelle noch, daß ITL eine rein zukunftsorientierte Logik ist. Aussagen über Vergangenheit können also hier nicht gemacht werden.

In Prolog wird einer bestimmten Teilmenge der Prädikatenlogik, nämlich der Hornklauselogik, eine operationale Semantik gegeben. Man kann nun auch einer bestimmten Teilmenge von ITL eine operationale Semantik geben. Diese basiert jedoch nicht auf Unifikation und Resolution, wie dies bei allen Prolog-ähnlichen Ansätzen der Fall ist, sondern auf dem Prinzip der Transformation. Dieses besagt grob gesprochen folgendes:

Bei Abarbeitung eines Tempura-Programms w wird dieses reduziert auf einen Teil w', der den aktuellen Zustand des zugrundeliegenden Intervalls (d.h. sozusagen den aktuellen Zeitpunkt) spezifiziert, und einen Teil w'', der auf dem verbleibenden (Rest-) Teilintervall erfüllt (wahr) ist. Diese Reduktion wird in jedem Zustand wiederholt, und zwar solange, bis schließlich das aktuelle Teilintervall nur noch aus einem Zustand besteht und es daher kein weiteres Restteilintervall mehr gibt. Damit ist die Transformation dann vollständig durchgeführt: w ist danach umgeformt zu einer Konjunktion von Teilformeln der Form:

$$w \equiv \wedge_{i=0..n} \circ^i w_i$$

Dabei bezeichne $\circ^i w$ die i-fache Anwendung des Operators $\circ$ auf die Formel w. Die Teilformeln w_i enthalten dabei keine temporalen Elemente mehr und bestimmen jeweils vollständig einen Zustand σ_i eines Intervalls $\sigma = \langle \sigma_0 \ldots \sigma_n \rangle$. Das Ergebnis der Abarbeitung eines Tempura-Programms ist dann gerade dieses Intervall σ. Es hat die Eigenschaft, daß w auf σ erfüllt ist.

Die Sprache Tempura ist von Moszkowski leider nur sehr informell beschrieben (vgl. dazu [7]). Eine präzise - operationale oder denotationale - Semantikbeschreibung liegt nicht vor. Aus diesem Grunde wird von uns eine eigene, auf Moszkowskis Prinzip der Transformation basierende Temporallogische Programiersprache EXTELL entwickelt, d.h. formal sauber beschrieben und entsprechend implementiert.
Die von uns entwickelte Sprache EXTELL soll sich darüberhinaus in zwei wesentlichen Punkten von Tempura unterscheiden: Zum einen enthält EXTELL auch **nichtdeterministische** Sprachkonstrukte wie $\vee$ (oder) und $\Diamond$ (sometimes). Zum anderen enthält EXTELL **vergangenheitsorientierte** Sprachkonstrukte wie z.B. (always-in-the-past), $\bullet$ (previous) etc., was die (pragmatische) Ausdrucksfähigkeit der Sprache EXTELL gegenüber Tempura beträchtlich erhöht.

Tempura basiert auf der von Moszkowski vorgestellten Temporal-Logik ITL (Interval Temporal Logic). Diese ist eine rein zukunftsorientierte Logik, d.h. Formeln werden in ITL auf endlichen Intervallen (= Folgen von Zuständen) interpretiert, deren erster Zustand immer den aktuellen Gegenwartszustand bezeichnet. Demgemäß beziehen sich die Temporalen Operatoren wie z.B. always immer nur auf die Zukunft: always ist eigentlich always-in-the-future.

Ein grundlegendes Problem für die Entwicklung einer Tempura-ähnlichen Programmiersprache mit (auch) vergangenheitsorientierten Operatoren ist nun, eine geeignete Logik zu finden bzw. zu entwickeln. Diese muß einerseits alle gewünschten Eigenschaften von ITL haben, d.h. es muß eine intervallbasierte Logik mit dem next- und vor allem auch dem chop-Operator sein, andererseits muß die Logik auch geeignete Vergangenheitsoperatoren besitzen. Ein weiteres entscheidendes Kriterium ist, daß in der Logik geeignete Rekurrenzrelationen für die benutzten temporalen Konstrukte gelten, damit das Prinzip der Transformation bei der Definition der operationalen Semantik der Sprache EXTELL angewandt werden kann. Genau diese Eigenschaften hat die im folgenden vorgestellte Logik EITeL.

2 Syntax von EITeL

EITeL ist eine Modallogik und als solche eine um Modaloperatoren erweiterte Prädikatenlogik. EITeL-Formeln bauen sich daher induktiv wie folgt auf:

- Alle Formeln des Prädikatenkalküls 1. Stufe sind EITeL-Formeln.
- Ist w eine EITeL-Formel, so sind auch $\Box w, \Diamond w$ and $\circ w$, sowie $\blacksquare w, \Diamond_p w$ und $\bullet w$ EITeL-Formeln.
- Sind w_1 und w_2 EITeL-Formeln, so sind auch $w_1; w_2$ und $w_1 ;_p w_2$ EITeL-Formeln.

Wir wollen obige Definition noch etwas präzisieren und Alphabet, Ausdrücke und Formeln der Sprache der erweiterten Intervall-Temporal-Logik genau definieren.

Definition 2.1 (Alphabet) Ein *Alphabet* der erweiterten Intervall-Temporal-Logik besteht aus

- einer Menge $\{'(' , ')' , ' \mid ' , '\langle' , '\rangle' , ',' , ' : ' , ' <' \}$ von (Trenn-) Zeichen
- einer Menge $L = \{'\neg' , '\wedge'\}$ von *logischen Verknüpfungen*
- dem *Existenzquantor* $'\exists'$
- einer Menge $\mathcal{M} = \{ 'o' , '\Box' , ' \bullet ' , '\blacksquare' , ',' , ';' , ' ;_p ' \}$ von *Modaloperatoren*
- einer abzählbaren Menge ZV von *Zustandsvariablen*
- einer abzählbaren Menge SV von *statischen Variablen*
- für jedes $n \in \mathbb{N}_0$ einer abzählbaren Menge $\mathcal{F}_n$ von *n-stelligen Funktionssymbolen.* $\mathcal{F}_0$ heißt auch die Menge der *Konstanten*
- für jedes $n \in \mathbb{N}_0$ einer abzählbaren Menge $\mathcal{P}_n$ von n-stelligen *Prädikatssymbolen.* (Insbesondere soll $'='\in \mathcal{P}_2$ gelten)
- den Symbolen *'true'*,*' start'*, und *'more'*, (die man als 0-stellige Prädikatssymbole auffassen kann).

Bemerkung: Zustandsvariable werden künftig durch Großbuchstaben A,B,C,... , statische Variable durch Kleinbuchstaben a,b,c,... bezeichnet. Für Funktionssymbole werden meistens die Buchstaben f,g,h,... verwendet. Prädikatssymbole sind i.A. p,q,r,... . Alle diese Symbole können auch indiziert auftreten.

Definition 2.2 (Terme) *Terme* sind induktiv definiert durch:

- Jede Zustandsvariable $V \in ZV$ ist ein Term.
- Jede statische Variable $v \in SV$ ist ein Term.
- Ist $f \in \mathcal{F}$ ein n-stelliges Funktionssymbol, $n \geq 0$, und sind $t_1, ..., t_n$ Terme, so ist $f(t_1, ..., t_n)$ ein Term.

- Ist t ein Term, so sind $\circ t$ und $\bullet t$ Terme.
- Listenkonstruktionen:
  - einfache *Listenkonstruktion:*
    Sind $t_0, ..., t_{n-1}$ Terme, $n \geq 1$, so ist $\langle t_0, ..., t_{n-1} \rangle$ ein Term.
  - *iterative Listenkonstruktion:*
    Sind t_1 und t_2 Terme und ist $v \in ST$ eine statische Variable, so ist $\langle t_1 : v < t_2 \rangle$ ein Term.
  - *Listenlänge:*
    Ist t ein Term, so ist $\mid t \mid$ ein Term.
  - *Indizierung:*
    Sind t_1 und t_2 Terme, so ist $t_1(t_2)$ ein Term.

Mit TERM bezeichnen wir die kleinste Menge, welche die gemäß obiger Definition bildbaren Terme enthält.

Definition 2.3 (atomare Formeln)

- *true*, *start* und *more* sind atomare Formeln.
- Ist $p \in \mathcal{P}_n$ ein n-stelliges Prädikatssymbol und sind $t_1, ..., t_n$ Terme, so ist $p(t_1, ..., t_n)$ eine atomare Formel.

Bezeichnung: Statt atomare Formeln verwenden wir meistens den Begriff Atom.

Definition 2.4 (Formeln) Formeln werden induktiv definiert wie folgt:

- Jedes Atom ist eine Formel. (Insbesondere ist $t_1 = t_2$ eine Formel, falls t_1, t_2 Terme sind).
- Sind w, w_1 und w_2 Formeln, so sind $\neg w$ und $w_1 \wedge w_2$ Formeln.
- Quantifizierung: Ist w eine Formel und $V \in ZV$ eine Zustandsvariable, bzw. $v \in SV$ eine statische Variable, so sind $\exists V : w$ und $\exists v : w$ Formeln.
- Next und Previous: Ist w eine Formel, so sind $\circ w$ und $\bullet w$ Formeln.
- Always: Ist w eine Formel, so sind $\Box w$ und $\blacksquare w$ Formeln.
- Chop und Past-Chop: Sind v_1 und v_2 Formeln, so sind $w_1; w_2$ und $w_1;_p w_2$ Formeln.

Mit FORM bezeichnen wir die kleinste Menge, welche die gemäß obiger Definition bildbaren Formeln enthält.

Bemerkung: Im allgemeinen sollen unter den Prädikaten insbesondere $<, \leq$ etc. sowie das 2-stellige Prädikat *list* vorhanden sein, so daß dann $t_1 < t_2, t_1 \leq t_2, list(t_1, t_2)$ atomare Formeln sind, falls t_1 und t_2 Terme sind (vgl. hierzu 3.2).

3 Semantik

3.1 Grundlegende Definitionen

Definition 3.1.1 (Zustand) Sei $V := ZV \cup SV$ die Menge der (Zustands- und statischen) Variablen eines Alphabets der Sprache der erweiterten Intervall-Temporal-Logik.
D sei eine abzählbare Datenmenge.
Ein Zustand ist eine Abbildung $f : V \rightarrow D$.
Die Menge der Zustände wird mit Σ bezeichnet: $\Sigma = \{f : V \rightarrow D\}$.

Definition 3.1.2 (Modell, Interpretation) Ein Modell M ist ein Tripel $M = (D, \Sigma, \mathcal{M})$, wobei D und Σ wie in Definition 3.1.1 sind und $\mathcal{M}$ eine Abbildung ist, die jedem k-stelligen Funktionssymbol f eine Abbildung $\mathcal{M}(f) : D^k \rightarrow D$ und jedem n-stelligen Prädikatssymbol p eine Abbildung $\mathcal{M}(p) : D^n \rightarrow \{true, false\}$ zuordnet. $\mathcal{M}$ heißt dann Interpretation.

Bemerkung:

1. Im folgenden bestehe D immer aus der Menge der Integerzahlen, den boole'schen Werten *true* und *false* sowie aus der Menge der - möglicherweise geschachtelten - endlichen Listen aus diesen Werten:
   $D = Integer \cup \{true, false\} \cup List$, wobei *List* definiert ist wie folgt:
   - $\langle\rangle \in List$
   - Ist $i \in Integer$, dann ist $\langle i \rangle \in List$
   - $\langle true \rangle, \langle false \rangle \in List$
   - Sind $a_0, .., a_{n-1} \in List$, so ist auch $\langle a_0, ..., a_{n-1} \rangle \in List$.
2. Weiterhin nehmen wir im folgenden immer an, daß $\mathcal{M}$ den Standardfunktionen $+, *$ etc. sowie den Standardprädikaten $<, \leq$ etc. immer die jeweilige natürliche Interpretation (hier Addition, Multiplikation, bzw. kleiner, kleiner oder gleich etc.) zuordnet.

Definition 3.1.3 Ein Modell $M = (D, \Sigma, \mathcal{M})$, das 1. und 2. aus obiger Bemerkung genügt, heißt auch Standard-Modell. $\mathcal{M}$ heißt dann Standard-Interpretation.

Definition 3.1.4 (Intervall) Die Menge der endlichen Folgen $\sigma_0 ... \sigma_n$ von Zuständen $\sigma_i \in \Sigma$ wird mit Σ^+ bezeichnet. Ist $\sigma = \sigma_0 ... \sigma_n \in \Sigma^+$, so heißt $\mid \sigma \mid = \mid \sigma_0 ... \sigma_n \mid := n$ die Länge von σ.
Ein Intervall $I \in \Sigma^+ \times \mathbb{N}_0$ ist ein Paar (σ, i), bestehend aus einer endlichen Folge $\sigma \in \Sigma^+$ von Zuständen und einem Index $i \in \mathbb{N}_0$ mit $0 \leq i \leq \mid \sigma \mid$.
σ_i heißt aktueller Gegenwartszustand des Intervalls (σ, i).
$\sigma_P := \sigma_0 ... \sigma_i$ heißt Vergangenheitsanteil, $\sigma_F := \sigma_i ... \sigma_{|\sigma|}$ heißt Zukunftsanteil des Intervalls (σ, i).
$|(\sigma, i)| := |\sigma|$ heißt die Länge des Intervalls (σ, i).

Bemerkung: Die Länge eines Intervalls (σ, i) ist definiert als die Anzahl der Zustände von σ minus 1, d.h. man kann die Länge eines Intervalls auffassen als die Anzahl der Zustandsübergänge im Intervall.

Definition 3.1.5 (Zulässiges Intervall) Sei $M = (D, \Sigma, \mathcal{M})$ ein Modell. Ein Intervall $(\sigma, i) = (\sigma_0 ... \sigma_{|\sigma|}, i)$ heißt zulässiges Intervall, falls gilt: $\sigma_i(v) = \sigma_j(v)\ \forall i, j \leq |\ \sigma\ |\ \forall v \in SV$, d.h. statische Variable haben auf zulässigen Intervallen konstante Werte.
Die Menge der zulässigen Intervalle bezeichnen wir mit I.
Im folgenden meinen wir mit Intervall immer zulässiges Intervall.

3.2 Interpretation von Termen und Formeln auf Intervallen

Wir wollen nun Terme und Formeln auf (zulässigen) Intervallen interpretieren.

Definition 3.2.1 Sei $M = (D, \Sigma, \mathcal{M})$ ein Modell und $(\sigma, i) = (\sigma_0 ... \sigma_{|\sigma|}, i) \in \Sigma^+ \times N_0$ ein zulässiges Intervall.
Wir definieren eine Abbildung $\mathcal{M}_{(\sigma,i)}$ induktiv wie folgt:

- auf Termen:
  - (a) Ist $V \in ZV$ eine Zustandsvariable, so ist $\mathcal{M}_{(\sigma,i)}(V) := \sigma_i(V)$.
  - (b) Ist $v \in SV$ eine statische Variable, so ist $\mathcal{M}_{(\sigma,i)}(v) := \sigma_i(v)\ (= \sigma_j(v)\ \forall_j \leq |\ \sigma\ |)$.
  - (c) Ist f ein n-stelliges Funktionssymbol und sind $t_1, ..., t_n$ Terme, so ist $\mathcal{M}_{(\sigma,i)}(f(t_1, ..., t_n)) := \mathcal{M}(f)(\mathcal{M}_{(\sigma,i)}(t_1), ..., \mathcal{M}_{(\sigma,i)}(t_n))$.
  - (d) Ist t ein Term, so ist $\mathcal{M}_{(\sigma,i)}(\circ t) := \mathcal{M}_{(\sigma,i+1)}(t)$, falls $|\ \sigma\ | \geq i + 1$; $\mathcal{M}_\sigma(\circ t)$ bleibt undefiniert, falls $|\ \sigma\ | = i$.
  - (e) $\mathcal{M}_{(\sigma,i)}(\bullet t) := \mathcal{M}_{(\sigma,i-1)}(t)$, falls $i \geq 0$; $\mathcal{M}_{(\sigma,i)}(\bullet t)$ bleibt undefiniert, falls $i = 0$.
  - (f) Sind $t_0, ..., t_n$ Terme, so ist $\mathcal{M}_{(\sigma,i)}(\langle t_0, ..., t_n \rangle) := \langle \mathcal{M}_{(\sigma,i)}(t_0), ..., \mathcal{M}_{(\sigma,i)}(t_n) \rangle$.
  - (g) Sind t_1 und t_2 Terme und ist v eine statische Variable, so ist $\mathcal{M}_{(\sigma,i)}(\langle t_1 : v < t_2 \rangle) := \langle \mathcal{M}_{(\sigma,i)}(t_1) : \mathcal{M}_{(\sigma,i)}(v) < \mathcal{M}_{(\sigma,i)}(t_2) \rangle$.
  - (h) Sind t_1 und t_2 Terme, so ist $\mathcal{M}_{(\sigma,i)}(t_1(t_2)) := \mathcal{M}_{(\sigma,i)}(t_1)_{\mathcal{M}_{(\sigma,i)}(t_2)}$, wobei die rechte Seite der Gleichung die übliche Indizierung bedeutet (Zählweise: von links, beginnend mit 0).
  - (i) Ist t ein Term, so ist $\mathcal{M}_{(\sigma,i)}(|\ t\ |) := |\ \mathcal{M}_{(\sigma,i)}(t)\ |$, wobei die Betragsstriche auf der rechten Seite die Listenlänge bezeichnen.
- auf Formeln:
  - (j) $\mathcal{M}_{(\sigma,i)}(true) = true$;
    $\mathcal{M}_{(\sigma,i)}(more) = true$ gdw. $|\sigma| > i$;
    $\mathcal{M}_{(\sigma,i)}(start) = true$ gdw. $i = 0$.
  - (k) Ist p ein n-stelliges Prädikat und sind $t_1, ..., t_n$ Terme, so ist $\mathcal{M}_{(\sigma,i)}(p(t_1, ..., t_n)) := \mathcal{M}(p)(\mathcal{M}_{(\sigma,i)}(t_1), ..., \mathcal{M}_{(\sigma,i)}(t_n))$.

(l) Sind t_1 und t_2 Terme, so ist $\mathcal{M}_{(\sigma,i)}(t_1 = t_2) = true$, gdw. $\mathcal{M}_{(\sigma,i)}(t_1) = \mathcal{M}_{(\sigma,i)}(t_2)$.

(m) Wir nehmen auch an, daß die Prädikate $<, \geq$, etc. mit der natürlichen Interpretation kleiner, größer oder gleich etc. versehen werden. Ferner soll gelten: Sind t_1, t_2 Terme, so ist $\mathcal{M}_{(\sigma,i)}(list(t_1, t_2)) = true$, falls $\mathcal{M}_{(\sigma,i)}(t_1)$ eine Liste der Länge $\mathcal{M}_{(\sigma,i)}(t_2)$ ist.

(n) Ist w eine Formel, so ist $\mathcal{M}_{(\sigma,i)}(\neg w) = true$, gdw. $\mathcal{M}_{(\sigma,i)}(w) = false$.

(o) Sind w_1 und w_2 Formeln, so ist $\mathcal{M}_{(\sigma,i)}(w_1 \wedge w_2) = true$, gdw. $\mathcal{M}_{(\sigma,i)}(w_1) = true$ und $\mathcal{M}_{(\sigma,i)}(w_2) = true$.

(p) Ist w eine Formel und V eine (statische oder Zustands-) Variable, so ist $\mathcal{M}_{(\sigma,i)}(\exists V : w) = true$, gdw. es ein Intervall $(\sigma', i) \in I$ gibt mit $\sigma \sim_V \sigma'$ und $\mathcal{M}_{(\sigma',i)}(w) = true$. Dabei ist die Äquivalenzrelation $\sim_V$ auf Σ^+ wie folgt definiert:

$$\sigma \sim_V \sigma' \text{ falls } \mid \sigma \mid = \mid \sigma' \mid \text{ und } \sigma_j(W) = \sigma'_j(W) \ \forall W \neq V \ \forall j \leq \mid \sigma \mid.$$

(q) Ist w eine Formel, so ist $\mathcal{M}_{(\sigma,i)}(\circ w) = true$, gdw. $\mid \sigma \mid > i$ und $\mathcal{M}_{(\sigma,i+1)}(w) = true$.

(r) Ist w eine Formel, so ist $\mathcal{M}_{(\sigma,i)}(\Box w) = true$, gdw. für alle $j \geq i$ gilt: $\mathcal{M}_{(\sigma,j)}(w) = true$.

(s) Ist w eine Formel, so ist $\mathcal{M}_{(\sigma,i)}(\bullet w) = true$, gdw. $i > 0$ und $\mathcal{M}_{(\sigma,i-1)}(w) = true$.

(t) Ist w eine Formel, so ist $\mathcal{M}_{(\sigma,i)}(\blacksquare w) = true$, gdw. für alle $0 \leq j \leq i$ gilt: $\mathcal{M}_{(\sigma,j)}(w) = true$.

(u) Sind w_1 und w_2 Formeln, so ist $\mathcal{M}_{(\sigma,i)}(w_1; w_2) = true$, gdw. es ein j gibt mit $i \leq j \leq \mid \sigma \mid$ und $\mathcal{M}_{(\sigma_0,\ldots,\sigma_j,i)}(w_1) = true$ und $\mathcal{M}_{(\sigma_0\ldots\sigma_{|\sigma|},j)}(w_2) = true$.

(v) Sind w_1 und w_2 Formeln, so ist $\mathcal{M}_{(\sigma,i)}(w_1 \ ;_p w_2) = true$, gdw. es k und j gibt mit $k \leq j \leq i$ und $\mathcal{M}_{(\sigma_0\ldots\sigma_j,k)}(w_1) = true$ und $\mathcal{M}_{(\sigma_0\ldots\sigma_{|\sigma|},j)}(w_2) = true$.

3.3 Erläuterungen zur Definition 3.2.1

Zu a) und b) Der Wert einer Variablen auf einem Intervall wird durch ihren Wert auf dem aktuellen Gegenwartszustand bestimmt.

Zu c) Der Term $f(t_1, \ldots, t_n)$ wird interpretiert, indem man die Interpretation des Funktionssymbols auf die Interpretation der Terme t_1 bis t_n (über σ) anwendet.

Zu d) $\circ t$ wird interpretiert, indem man t auf demjenigen Intervall interpretiert, das sich ergibt, wenn man den auf den aktuellen Gegenwartszustand σ_i unmittelbar folgenden Zustand σ_{i+1} als neuen aktuellen Gegenwartszustand nimmt. Man interpretiert $\circ t$ sozusagen, indem man t *im nächsten Augenblick* interpretiert.

Zu e) Analog zu $\circ t$ wird $\bullet t$ interpretiert, indem man t im *vorherigen Augenblick* interpretiert.

Zu f) bis i) Listenkonstrukte werden interpretiert, indem man Listen aus den interpretierten Termen bildet; analog interpretiert man die Indizierung und Betragsbildung.

Zu j) *true* wird natürlich als *Wahrheitswert true* interpretiert und immer erfüllt.
Das 0-stellige Prädikat *more* wird auf (σ, i) genau dann erfüllt, wenn σ_i noch nicht der letzte Zustand des Intervalls (σ, i) ist.
Analog wird das 0-stellige Prädikat *start* auf (σ, i) genau dann erfüllt, wenn σ_i der erste Zustand des Intervalls (σ, i), also $i = 0$ ist.

Zu k) bis o) Analog c).

Zu p) Die Definition ist klar: alle Variablen außer V müssen sich auf σ' und σ *gleich verhalten.*

Zu q) $\circ w$ ist erfüllt, wenn w im *nächsten* Zeitpunkt (vgl. d) erfüllt ist.

Zu r) $\Box w$ ist erfüllt, falls w in *allen zukünftigen Zeitpunkten* erfüllt ist; genauer: w muß auf allen Intervallen (σ, j) erfüllt sein, die sich ergeben, wenn man statt des aktuellen Gegenwartszeitpunkts einen zukünftigen Zeitpunkt $\sigma_j, j \geq i$, als *aktuellen* Gegenwartszeitpunkt nimmt.

Zu s) $\bullet w$ wird analog zu $\circ w$ interpretiert: $\bullet w$ ist erfüllt, falls w im *vorherigen Zeitpunkt* erfüllt war.

Zu t) Auch hier analog $\Box w$: $\blacksquare w$ ist erfüllt, falls w zu jedem Zeitpunkt des Vergangenheitsteilintervalls erfüllt war.

Zu u) und v) $\mathcal{M}_{(\sigma,i)}(w_1; w_2) = true$ bedeutet: Es gibt einen Index j mit $i \leq j \leq n$ derart, daß w_1 auf dem Teilintervall $(\sigma_0 ... \sigma_j, i)$ und w_2 auf dem Endteilintervall $(\sigma_0, ..., \sigma_{|\sigma|}, j)$ gilt. Der Zustand σ_j ist sowohl der letzte oder späteste für die Interpretation von w_1 relevante Zustand als auch derjenige Zustand oder Zeitpunkt, ab dem dann w_2 gilt.
$';'$ drückt also in gewisser Weise sequentielle Komposition aus. Unter diesem Gesichtspunkt ist $';'$ das sequentielle Gegenstück zu $'\wedge'$, das im temporalen Kontext die parallele Komposition bedeutet. Chop spielt bei der Definition von sequentiellen Konstrukten, wie z.B. for- und while-Schleifen eine wichtige Rolle. Ein weiterer Aspekt ist folgender:
Definiert man $\Diamond w := true; w$ für jede Formel w, so gilt: $\mathcal{M}_{(\sigma,i)}(\Diamond w) = true$ gdw. es ein j gibt mit $i \leq j \leq |\ \sigma\ |$ und $\mathcal{M}_{(\sigma,j)}(w) = true$. Damit gilt dann für jedes w $\mathcal{M}_{(\sigma,i)}(\Box w) = \mathcal{M}_{(\sigma,i)}(\neg\Diamond\neg w)$.
Man kann also $'\Box'$ als aus $';'$ und $'\neg'$ hergeleiteter Operator ansehen. Die grundlegenden zukunftsgerichteten Operatoren von EITel sind also $'\circ'$ und $';'$.
In diesem Sinne ist $'\ ;_p\ '$ das vergangenheitsorientierte Gegenstück zu $';'$: $w_1 ;_p w_2$ ist auf (σ, i) unter $\mathcal{M}$ erfüllt genau dann, wenn es einen Zeitpunkt j in der Vergangenheit gibt, indem $w_1; w_2$ erfüllt war und dabei derjenige Anteil des Intervalls, auf dem w_1 erfüllt ist, vollständig in der Vergangenheit liegt. Dies soll durch die nachstehenden

Abbildungen 1 und 2 verdeutlicht werden.

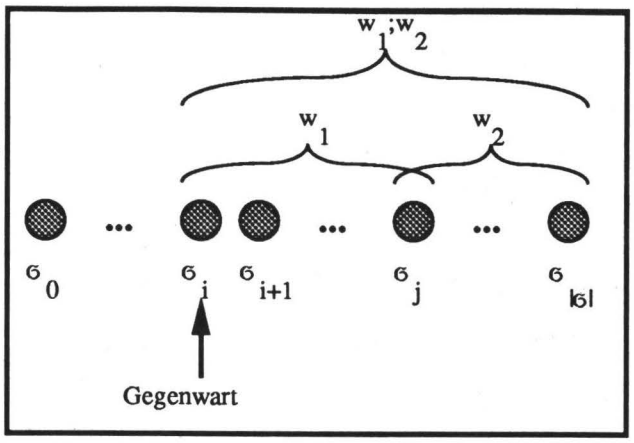

Abb. 1: chop (';')

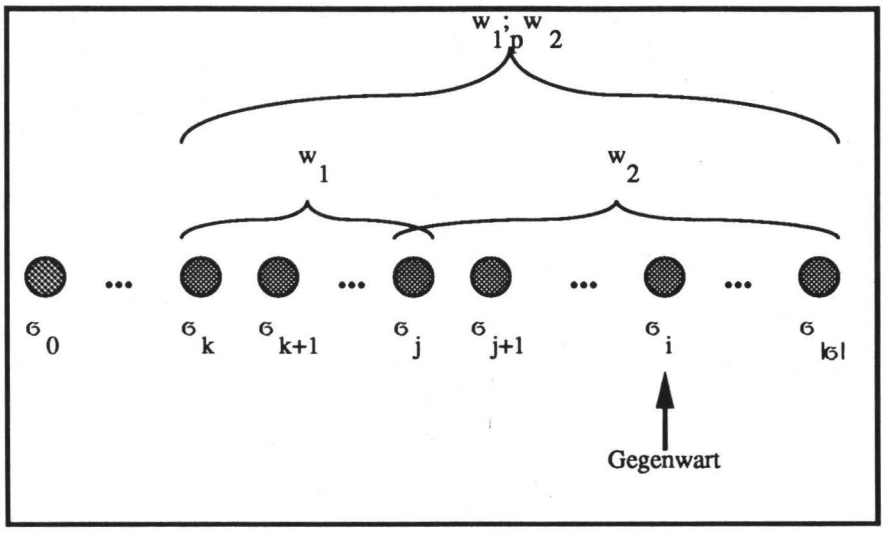

Abb. 2: past_chop (';$_p$')

Definiert man nun wiederum für jede Formel w $\Diamond_p w := true\ ;_p w$, so gilt $\mathcal{M}_{(\sigma,i)}(\blacksquare w) = \mathcal{M}_{(\sigma,i)}(\neg\Diamond_p\neg w)$, so daß man auch hier '$\blacksquare$' als aus '$;_p$ ' und '$\neg$ ' hergeleitetes Konstrukt ansehen kann. Eine andere Möglichkeit der Definition eines chop-Operators für die Vergangenheit, bei der die Symmetrie zwischen Vergangenheit und Zukunft besser berücksichtigt wird, ist folgende:
Wir definieren für jede Formel w_1 und w_2

$$\begin{aligned} \mathcal{M}_{(\sigma,i)}(w_1;^p w_2) &= true \text{ gdw. } \exists\, j \leq i \text{ mit} \\ \mathcal{M}_{(\sigma_0 \ldots \sigma_j,0)}(w_1) &= true = \mathcal{M}_{(\sigma_j \ldots \sigma_i,j)}(w_2) \end{aligned}$$

Dies soll durch die Abbildung 3 verdeutlicht werden.

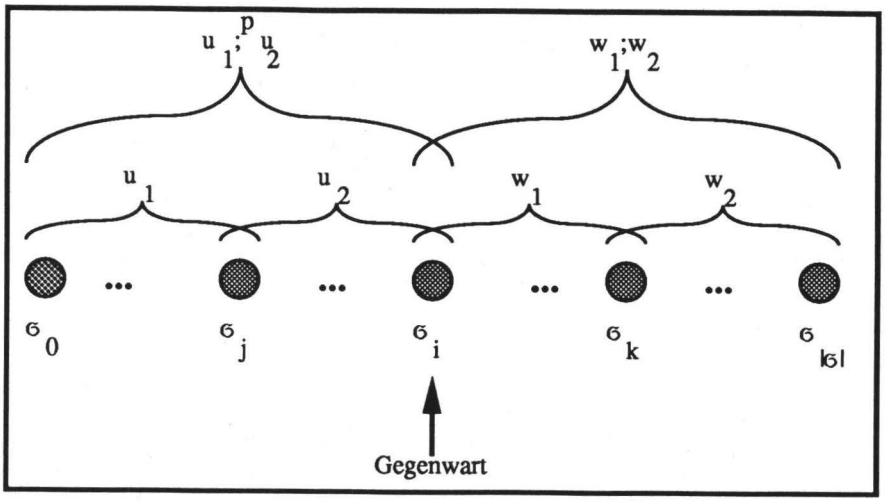

Abb. 3: Eine Alternative für past_chop (';^{p}')

Bei dieser Definition geht allerdings der Zusammenhang mit dem Sometimes- bzw. dem Always-in-the-past-Operator verloren: Es gilt dann **nicht** mehr:

$$\mathcal{M}_{(\sigma,i)}(\blacksquare w) = \mathcal{M}_{(\sigma,i)}(true;_p w)$$

Man bezahlt den Gewinn einer Symmetrie hier also mit dem Verlust einer anderen.

In den Klassischen Temporallogiken wie z.B. der Tense-Logik von Prior [8], gibt es keine Analogie zu den Chop-Operatoren. Dies liegt daran, daß diese punktbasiert sind, d.h. Formeln werden dort immer auf Zeitpunkten ausgewertet, wohingegen EITel intervallbasiert ist (die Chop-Operatoren "inspizieren" jeweils das ganze Intervall).

Definition 3.3.1 Sei $\mathcal{M} = (D, \Sigma, \mathcal{M})$ ein Modell und σ ein zulässiges Intervall. Wir sagen: eine Formel w wird von σ unter der Interpretation $\mathcal{M}$ erfüllt, falls $\mathcal{M}_{(\sigma,i)}(w) = true$ ist. Wir schreiben dann auch $(\sigma, i) \models_M w$. Falls alle zulässigen Intervalle (σ, i) unter $\mathcal{M}$ w erfüllen, so schreiben wir $\models_M w$ und sagen: w ist gültig unter $\mathcal{M}$.
Für $(\sigma, i) \models_M w$ bzw. $\models_M w$ schreiben wir oft einfach $(\sigma, i) \models w$ bzw. $\models v$, falls M fest ist.

Wir führen an dieser Stelle noch einige **Sprechweisen** ein, die wir im folgenden des öfteren benutzen werden.
Sei $\mathcal{M}$ eine Interpretation und $(\sigma, i) = (\sigma_0 \ldots \sigma_n, i) \in I$ ein zulässiges Intervall. Wir sagen, eine Formel w gilt in einem Zustand $\sigma_j, 0 \leq j \leq n$, falls $\mathcal{M}_{(\sigma,j)}(w) = true$. Für Zustand benutzen wir synonym auch oft Zeitpunkt, falls $j < i$ bzw. $j > i$ sagen wir auch die Formel galt im Zustand σ_j bzw. wird im Zustand σ_j gelten.
Bei Termen sprechen wir in analoger Weise vom Wert eines Terms t im Zustand σ_j.
Wir sagen eine Formel w gilt auf einem Teilstück (bzw. Zukunftsteilstück bzw. Vergangenheitsteilstück) von (σ, i), falls $\exists k, l$ mit $0 \leq k \leq l \leq n$ (bzw. $i \leq k \leq l \leq n$ bzw. $0 \leq k \leq l \leq i$) und $\mathcal{M}_{(\sigma_0 \ldots \sigma_l, k)}(w) = true$.
Ist $l = n$, so sagen wir w gilt auf einem Restteilintervall von (σ, i).

4 Abgeleitete Konstrukte

Im folgenden sollen einige abgeleitete Konstrukte nach bestimmten Gruppen geordnet zusammengestellt werden.
Dabei seien immer w, w_1, w_2, w_3 Formeln, t, t_1, t_2 Terme, V eine Zustandsvariable und v eine statische Variable.

4.1 Abgeleitete Boole'sche Operatoren

Die Booleschen Operatoren werden wie üblich definiert:

Definition 4.1.1 (Boolesche Operatoren)

4.1.1.1	Disjunktion	$w_1 \vee w_2 := \neg(\neg w_1 \wedge \neg w_2)$
4.1.1.2	Implikation	$w_1 \supset w_2 := \neg w_1 \vee w_2$
4.1.1.3	Äquivalenz	$w_1 \equiv w_2 := (w_1 \supset w_2) \wedge (w_2 \supset w_1)$
4.1.1.4	false	$false := \neg true$
4.1.1.5	Fallunterscheidung	$if\ w_1\ then\ w_2\ else\ w_3 := (w_1 \supset w_2) \wedge (\neg w_1 \supset w_3)$

Man erhält damit die übliche Semantik (hier natürlich auf Intervallen), nämlich

$\mathcal{M}_{(\sigma,i)}(w_1 \vee w_2) = true$	gdw.	$\mathcal{M}_{(\sigma,i)}(w_1) = true$ oder $\mathcal{M}_{(\sigma,i)}(w_2) = true$
$\mathcal{M}_{(\sigma,i)}(w_1 \supset w_2) = true$	gdw.	$\mathcal{M}_{(\sigma,i)}(w_2) = true$ oder $\mathcal{M}_{(\sigma,i)}(w_1) = false$
$\mathcal{M}_{(\sigma,i)}(w_1 \equiv w_2) = true$	gdw.	$\mathcal{M}_{(\sigma,i)}(w_1) = \mathcal{M}_{(\sigma,i)}(w_2)$
$\mathcal{M}_{(\sigma,i)}(false) = false$		
$\mathcal{M}_{(\sigma,i)}(if\ w_1\ then\ w_2\ else\ w_3) = true$	gdw.	$\mathcal{M}_{(\sigma,i)}(w_1) = \mathcal{M}_{(\sigma,i)}(w_2) = true$ oder $\mathcal{M}_{(\sigma,i)}(w_1) = false$ und $\mathcal{M}_{(\sigma,i)}(w_3) = true$

4.2 Abgeleitete Quantoren

Definition 4.2.1 (Quantoren)

4.2.1.1 Unbeschränkter Allquantor:
$\forall V : w := \neg\exists V : \neg w$
$\forall v : w := \neg\exists v : \neg w$

4.2.1.2 Beschränkter Allquantor:
$\forall v < t : w := \forall v : (0 \leq v < t \supset w)$

4.2.1.3 Beschränkter Existenzquantor:
$\exists v < t : w := \exists v : (0 \leq v < t \wedge w)$

In analoger Weise kann man beschränkte Allquantifizierung und beschränkte Existenzquantifizierung für $\leq, >$ und $\geq$ sowie für $t_1 \leq v < t_2$ etc. einführen.
Man beachte, daß die beschränkte Allquantifizierung und die beschränkte Existenzquantifizierung nur für statische Variable definiert sind.

Die Semantik der abgeleiteten Quantoren ergibt sich direkt aus der Semantik des Existenz-

quantors:

$\mathcal{M}_{(\sigma,i)}(\forall V : w) = true$	gdw.	für jedes Intervall $(\sigma', i) \in I$ mit $\sigma \sim_V \sigma'$ gilt $\mathcal{M}_{(\sigma',i)}(w) = true$
$\mathcal{M}_{(\sigma,i)}(\forall v : w) = true$	gdw.	für jedes Intervall $(\sigma', i) \in I$ mit $\sigma \sim_v \sigma'$ gilt $\mathcal{M}_{(\sigma',i)}(w) = true$
$\mathcal{M}_{(\sigma,i)}(\forall v < t : w) = true$	gdw.	für jedes Intervall $(\sigma', i) \in I$ mit $\sigma \sim_v \sigma'$ gilt $\mathcal{M}_{(\sigma',i)}(w) = true$ oder $\mathcal{M}_{(\sigma',i)}(0 \leq v < t) = false$
$\mathcal{M}_{(\sigma,i)}(\exists v < t : w) = true$	gdw.	es gibt ein Intervall $(\sigma', i) \in I$ mit $\sigma \sim_v \sigma'$ und $\mathcal{M}_{(\sigma',i)}(0 \leq v < t) = \mathcal{M}_{(\sigma',i)}(w) = true$

4.3 Abgeleitete temporale Operatoren

In 3. haben wir die Sometimes-Operatoren '$\Diamond$' und '$\Diamond_p$' schon durch die Chop-Operatoren ';' und ';$_p$' definiert

$$\Diamond w := true; w \text{ bzw.}$$
$$\Diamond_p w := true;_p w$$

In Analogie zu klassischen Modal- und Temporallogiken kann man $\Diamond$ und $\Diamond_p$ auch in EITel definieren vermöge

$$\Diamond w := \neg\Box\neg w \text{ bzw.}$$
$$\Diamond_p w := \neg\blacksquare\neg w$$

Nach 3. erhält man mit beiden Definitionen die gleiche Semantik:

$\mathcal{M}_{(\sigma,i)}(\Diamond w) = true$	gdw.	$\exists j : i \leq j \leq \mid \sigma \mid$ mit $\mathcal{M}_{(\sigma,j)}(w) = true$
$\mathcal{M}_{(\sigma,i)}(\Diamond_p w) = true$	gdw.	$\exists j : 0 \leq j \leq i$ mit $\mathcal{M}_{(\sigma,j)}(w) = true$

Wir definieren also:

Definition 4.3.1 (Sometimes-Operatoren):

4.3.1.1 $\Diamond w := \neg\Box\neg w$
4.3.1.2 $\Diamond_p w := \neg\blacksquare\neg w$

Wir wollen nun noch eine Reihe weiterer temporaler Operatoren definieren.

Definition 4.3.2 (empty, started):

4.3.2.1	*empty*	:=	$\neg more$
4.3.2.2	*started*	:=	$\neg start$
4.3.2.3	*skip*	:=	$\circ empty$

Erläuterungen: Die Formel empty wird von (σ, i) unter der Interpretation $\mathcal{M}$ erfüllt, wenn σ die Länge $\mid \sigma \mid = i$ hat, d.h. der Zukunftsanteil σ_F des Intervalls (σ, i) die Länge 0 hat:

$\mathcal{M}_{(\sigma,i)}(empty) \quad = \quad true \quad gdw. \quad |\sigma| = i$

Das vergangenheitsgerichtete Gegenstück zu empty ist das in 3.2.1 (j) eingeführte 0-stellige Prädikat start, das von(σ, i) unter $\mathcal{M}$ genau dann erfüllt wird, wenn der Vergangenheitsanteil σ_P des Intervalls (σ, i) die Länge 0 hat, d.h. $i = 0$ ist :

$\mathcal{M}_{(\sigma,i)}(start) \quad = \quad true \quad gdw. \quad i = 0$

Daß die jeweiligen Zukunfts- bzw. Vergangenheitsanteile von (σ, i) Länge 0 haben, besagen gerade die jeweiligen Negationen obiger 0-stelliger Prädikate. Es sind dies respective die Prädikate *more* (vgl. 3.2.1 (j)) und *started*, für die demnach gilt:

$\mathcal{M}_{(\sigma,i)}(more) = true \quad$ gdw. $\quad i < |\sigma|$
$\mathcal{M}_{(\sigma,i)}(started) = true \quad$ gdw. $\quad 0 < i.$

Skip schließlich wird von (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn der verbleibende Zukunftsanteil σ_F von (σ, i) gerade noch Länge 1 hat:

$\mathcal{M}_{(\sigma,i)}(skip) = true \quad$ gdw. $\quad |\sigma| = i + 1$ (gdw. $|\sigma_F| = 1$).

Definition 4.3.3 (gets, got, stable, past_stable)

4.3.3.1	t_1 *gets* t_2	:=	$\Box(more \supset (\circ t_1 = t_2))$
4.3.3.2	t_1 *got* t_2	:=	$\blacksquare(started \supset (\bullet t_1 = t_2))$
4.3.3.3	*stable* t	:=	t *gets* t
4.3.3.4	*past_stable* t	:=	t *got* t

Erläuterung: t_1 *gets* t_2 wird von (σ, i) unter $\mathcal{M}$ erfüllt, wenn im weiteren Verlauf des Zeitintervalls σ, d.h. auf dem Zukunftsanteil von (σ, i), der Ausdruck t_1 den Wert des Ausdrucks t_2 mit einer Zeiteinheit Verzögerung annimmt:

$\mathcal{M}_{(\sigma,i)}(t_1 \; gets \; t_2) = true$ gdw. $\forall j : i \leq j < |\sigma| : \mathcal{M}_{(\sigma,j+1)}(t_1) = \mathcal{M}_{(\sigma,j)}(t_2)$

got ist das *Vergangenheitsgegenstück* zu gets: t_1 *got* t_2 wird von (σ, i) unter $\mathcal{M}$ erfüllt, wenn im Vergangenheitsanteil von (σ, i) der Ausdruck t_1 den Wert des Ausdrucks t_2 mit einer Zeiteinheit Verzögerung annimmt:

$\mathcal{M}_{(\sigma,i)}(t_1 \; got \; t_2) = true \quad$ gdw. $\forall j : 0 \leq j < i : \mathcal{M}_{(\sigma,j+1)}(t_1) = \mathcal{M}_{(\sigma,j)}(t_2)$

stable t bzw. *past_stable* t besagt demnach gerade, daß der Wert von t in jedem Zustand des Zukunfts- bzw. des Vergangenheitsanteils von (σ, i) gleich (stabil) ist.

Definition 4.3.4 (halt, begin, final, initial)

4.3.4.1	*halt* w	:=	$\Box(w \equiv empty)$
4.3.4.2	*begin* w	:=	$\blacksquare(w \equiv start)$
4.3.4.3	*final* w	:=	$\Box(empty \supset w)$
4.3.4.4	*initial* w	:=	$\blacksquare(start \supset w)$

Erläuterung: Die Formel *halt* w wird auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn w nur

genau im letzten Zeitpunkt von σ, d.h. auf $(\sigma, \mid \sigma \mid)$ erfüllt wird:

$\mathcal{M}_{(\sigma,i)}(halt\ w)$ = *true* gdw. $\forall j <\mid \sigma \mid$
$\mathcal{M}_{(\sigma,j)}(w)$ = *false* und
$\mathcal{M}_{(\sigma,\mid\sigma\mid)}(w)$ = *true*

Im Unterschied zu *halt* w betrachtet *final* w **nur** den letzten Zeitpunkt von σ, genauer: Der Wert von *final* w hängt nur vom Wert von w auf $(\sigma, \mid \sigma \mid)$ ab.

$$\mathcal{M}_{(\sigma,i)}(final\ w) = true \quad \text{gdw.} \quad \mathcal{M}_{(\sigma,\mid\sigma\mid)}(w) = true.$$

begin und *initial* sind wiederum die vergangenheitsorientierten Gegenstücke zu *halt* und *final*:
Die Formel begin w wird auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn w **nur genau** im Anfangszeitpunkt von σ, d.h. auf $(\sigma, 0)$, erfüllt wird, initial w wird auf (σ, i) erfüllt, wenn w im Anfangszeitpunkt von σ, d.h. auf $(\sigma, 0)$ erfüllt ist (war):

$\mathcal{M}_{(\sigma,i)}(begin\ w)$ = *true* gdw. $\forall j : 0 < j \leq i : \mathcal{M}_{(\sigma,j)}(w) = false$ **und** $\mathcal{M}_{(\sigma,0)}(w) = true$
$\mathcal{M}_{(\sigma,i)}(initial\ w)$ = *true* gdw. $\mathcal{M}_{(\sigma,0)}(w) = true$

Definition 4.3.5 (temporale Gleichheiten)

4.3.5.1 $t_1 \approx t_2$:= $\Box t_1 = t_2$
4.3.5.2 $t_1 \approx_p t_2$:= $\blacksquare t_1 = t_2$

Erläuterung: Der Unterschied zwischen der Gleichheit und den temporalen Gleichheiten liegt in der zeitlichen Komponente: $t_1 = t_2$ bedeutet t_1 ist jetzt, d.h. im aktuellen Gegenwartszeitpunkt, gleich t_2, wohingegen $t_1 \approx t_2$ bedeutet, t_1 ist in Zukunft immer, d.h. in jedem Zeitpunkt des Zukunftsteilstücks σ_F von (σ, i), gleich t_2 und $t_1 \approx_p t_2$ bedeutet t_1 war in Vergangenheit immer, d.h. in jedem Zeitpunkt des Vergangenheitsteilstücks σ_P von (σ, i), gleich t_2. Genauer:

$\mathcal{M}_{(\sigma,i)}(t_1 \approx t_2) = true$ gdw. $\forall j : i \leq j \leq \mid \sigma \mid : \mathcal{M}_{(\sigma,j)}(t_1) = \mathcal{M}_{(\sigma,j)}(t_2)$
$\mathcal{M}_{(\sigma,i)}(t_1 \approx_p t_2) = true$ gdw. $\forall j : 0 \leq j \leq i : \mathcal{M}_{(\sigma,j)}(t_1) = \mathcal{M}_{(\sigma,j)}(t_2)$

Definition 4.3.6 (weak_next, weak_previous)

4.3.6.1 $\circ_w$:= *empty* $\vee \circ w$
4.3.6.2 $\bullet_w w$:= *start* $\vee \bullet w$

Erläuterung: $\circ_w$ *(weak_next)* ist ein sehr wichtiges Konstrukt für die beabsichtigte Anwendung von Teilen von EITel als Temporallogische Programmiersprache. Wie der Name schon besagt, ist *weak_next* eine Abschwächung des next-Operators ($\circ$), und zwar insofern, daß $\circ_w w$ über (σ, i) unter $\mathcal{M}$ auch dann erfüllt ist, wenn der Zukunftsanteil von (σ, i) die Länge 0 hat, d.h. $i = |\sigma|$ ist:

$$\mathcal{M}_{(\sigma,i)}(\circ_w w) = true \text{ gdw. } i = |\sigma| \quad \text{oder } i < |\sigma| \quad \text{und} \quad \mathcal{M}_{(\sigma,i+1)}(w) = true$$

$\bullet_w$ *(weak_previous)* ist wiederum das vergangenheitsorientierte Gegenstück zu $\circ_w$. Im Unterschied zu $\bullet w$ ist $\bullet_w w$ auch dann über (σ, i) unter $\mathcal{M}$ erfüllt, wenn der Vergangenheitsanteil von (σ, i) die Länge 0 hat, d.h. wenn $i = 0$ ist:

$$\mathcal{M}_{(\sigma,i)}(\bullet_w w) = true \text{ gdw. } i = 0 \text{ oder } i > 0 \text{ und } \mathcal{M}_{(\sigma,i-1)}(w) = true$$

Definition 4.3.7 (Intervallänge: length and past_length)

4.3.7.1	$length\, t$	:=	$\exists I : (I = t) \wedge (I\ gets(I-1)) \wedge halt(I = 0)$
4.3.7.2	$past\_length(t)$	:=	$\exists I : (I = t) \wedge (I\ got(I+1)) \wedge begin(I = 0)$

Erläuterung: *length* und *past_length* sind Intervallängenprädikate. $length(t)$ ist auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn der Zukunftsanteil σ_F von (σ, i) die Länge t hat:

$$\begin{array}{lll} \mathcal{M}_{(\sigma,i)}(length(t)) = true & \text{gdw.} & \mathcal{M}_{(\sigma,i)}(t) = |\sigma| - i \\ & \text{gdw.} & |\sigma_F| = \mathcal{M}_{(\sigma,i)}(t)) \end{array}$$

Analog ist $past\_length(t)$ auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn der Vergangenheitsanteil σ_P von (σ, i) die Länge t hat:

$$\begin{array}{lll} \mathcal{M}_{(\sigma,i)}(past\_length(t)) = true & \text{gdw.} & \mathcal{M}_{(\sigma,i)}(t) = i \\ & \text{gdw.} & |\sigma_P| = \mathcal{M}_{(\sigma,i)}(t)) \end{array}$$

Definition 4.3.8 (temporale Wertzuweisung)

$$t_1 \rightarrow t_2 := \exists a : (a = t_1) \wedge final(t_2 = a)$$

Erläuterung: $t_1 \rightarrow t_2$ wird auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn der Wert von t_1 im aktuellen Gegenwartszustand gleich dem Wert von t_2 im letzten Zustand von σ ist:

$$\begin{array}{lll} \mathcal{M}_{(\sigma,i)}(t_1 \rightarrow t_2) & = & true \text{ gdw.} \\ \mathcal{M}_{(\sigma,i)}(t_1) & = & \mathcal{M}_{(\sigma,|\sigma|)}(t_2) \end{array}$$

Dieser Effekt wird in der Definition von $\rightarrow$ durch Verwendung einer *lokalen* Variablen a erreicht, die benutzt wird, um die Werte, die t_1 und t_2 zu verschiedenen Zeitpunkten haben, zu vergleichen.

4.4 Sequentielle Konstrukte

Der wichtigste Operator zur Bildung sequentieller Konstrukte ist der Chop-Operator. Wir definieren hier mit dem Chop-Operator einfache und indizierte for-Schleifen, sowie while- und repeat-Schleifen. Dazu werden zunächst zwei Hilfsprädikate benötigt.

Definition 4.4.1 (das Prädikat end_points)

$$\begin{array}{lll} end\_points(l, n) & := & list(l, n+1) \wedge \\ & & \exists i : (past\_length(i) \wedge l_o = i \wedge length(l_n - i) \wedge (\forall j < n : l_j \leq l_{j+1})) \end{array}$$

Definition 4.4.2 (das Prädikat iteration)

$iteration(l, n, j, w) := \exists i : (past\_length(i) \wedge (length(l_j - i); w; length(l_n - l_{j+1})))$

Erläuterung zu 4.4.1 und 4.4.2: Das Prädikat $end\_points(l, n)$ wird auf (σ, i) unter $\mathcal{M}$ genau dann erfüllt, wenn l eine Liste der Länge n+1 ist, die - intuitiv gesprochen - das verbleibende Zukunftsteilstück des Intervalls (σ, i) in n-Teile teilt durch Auflistung der n+1 Endzustände der n+1 Teilstücke.

$$
\begin{array}{lcl}
\mathcal{M}_{(\sigma,i)}(end\_points(l,n)) & = & true \text{ gdw.} \\
\mathcal{M}_{(\sigma,i)}(list(l, n+1)) & = & true \text{ und} \\
\mathcal{M}_{(\sigma,i)}(l_o) & = & i \text{ und} \\
\mathcal{M}_{(\sigma,i)}(l_{n+1}) & = & |\sigma \text{ und} \\
\mathcal{M}_{(\sigma,i)}(l_j) & \leq & \mathcal{M}_{(\sigma,i)}(l_{j+1}) \ \forall j < n
\end{array}
$$

Das Prädikat $iteration(l, n, j, w)$ ist nun auf (σ, i) unter $\mathcal{M}$ erfüllt gdw. w auf dem j-ten Teilstück der durch $end\_points(l, n)$ gegebenen Zerlegung des Zukunftsteils des Intervalls (σ, i) erfüllt ist (bzw. sein wird). Dies soll durch folgende Abbildung 4 verdeutlicht werden:

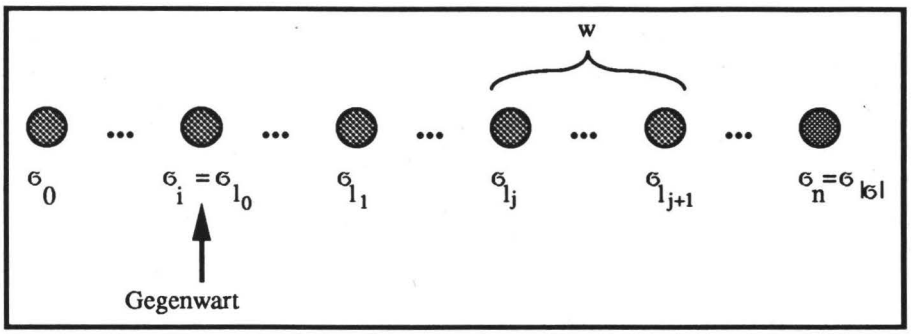

Abb. 4: iteration(l, n, j, w)

Damit können wir nun Schleifen definieren.

Definition 4.4.3 (einfache for-Schleifen)

$for\ t\ times\ do\ w := \exists l, n : n = t \wedge\ end\_points(l, n) \wedge \forall i < n : iteration(l, n, i, w))$ wo i,l und n nicht frei in t oder w auftreten dürfen.

Erläuterung: In der Definition der einfachen for-Schleife werden *lokale* Variable n und l eingeführt. Dabei bekommt n den Wert des Terms t und liefert so die Anzahl der Iteratio-

nen von w auf (σ, i). Das Zukunftsteilstück des Intervalls (σ, i) wird durch $end\_points(l, n)$ in n Teilstücke zerlegt, auf denen dann w vermöge $iteration(l, n, i, w)$ jeweils gilt (*iteriert wird*).
Dies wird durch Abbildung 5 noch einmal verdeutlicht.

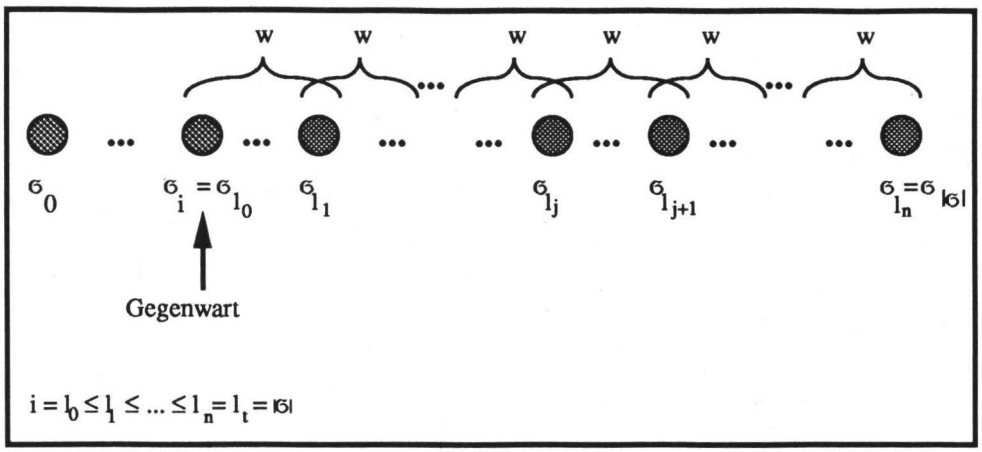

Abb. 5: einfache for-Schleife (for t times do w)

Definition 4.4.4 (Indizierte for-Schleifen)

$$\begin{aligned} for\ v < t\ do\ w\ \ :=\ \ & \exists l, n : (n = t)\ \wedge\ end\_points(l, n) \\ & \wedge\ \forall i < n : iteration(l, n, i, \exists v : (v = i)\ \wedge\ w) \end{aligned}$$

wo i, k und n nicht frei in v, t oder w auftreten.

Definition 4.4.5 (while-Schleifen)

$$\begin{aligned} while\ w_1\ do\ w_2\ \ :=\ \ & \exists i, l, n : (end\_points(l, n) \wedge past\_length(i) \\ & \wedge\ \forall j \leq n : ((j < n) \equiv length(l_j - i); w_1) \\ & \wedge \forall k < n : iteration(l, n, k, w_2)) \end{aligned}$$

Erläuterung: Das *while*-Konstrukt wird auf (σ, i) erfüllt, wenn es eine Zerlegung des Zukunftsteilstücks von (σ, i) in n Teilstücke gibt, derart, daß w_2 (vermöge *iteration*) auf jedem Teilstück v_1, auf jedem durch $end\_points(l, n)$ gegebenen Endteilstück $\sigma_{l_j \ldots} \sigma_n$ außer auf σ_n selbst gilt. Dies wird in der Definition durch den Ausdruck $\forall j < n + i : (j < n) \equiv length(l_j - i); w_1$ gewährleistet.

Abbildung 6 verdeutlicht dies noch einmal.

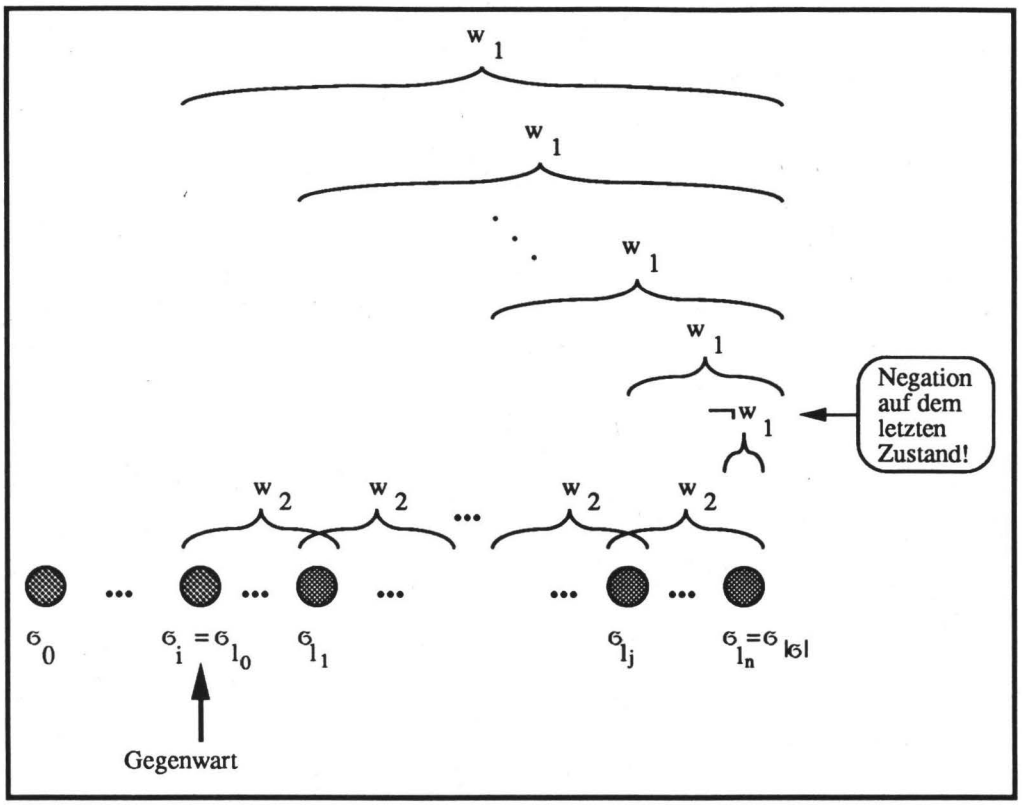

Abb. 6: while-Schleife (while w_1 do w_2)

Mit Hilfe der while-Schleifen werden nun noch in naheliegender Weise repeat-Schleifen definiert:

Definition 4.4.6

$$repeat\ w_1\ until\ w_2 \quad := \quad w_1\ ;\ while(\neg w_2\ do\ w_1)$$

Erläuterung Die Bedeutung des repeat-Konstrukts ergibt sich unmittelbar aus *chop* und *while*.

5 Rekurrenzrelationen

In diesem Abschnitt werden Rekurrenzrelationen für die wichtigsten Konstrukte von EITeL hergeleitet. Damit wird die Grundlage für eine prozedurale Interpretation eines großen Teils der EITeL-Fomeln gelegt.

Satz 5.1
Sei $M = (D, \Sigma, \mathcal{M})$ ein Modell, t, t_1, t_2 seien Terme und w eine Formel. Dann gilt:

5.1.1	$\models$	$\circ w$	$\equiv$	$more \wedge \circ_w w$
5.1.2	$\models$	$\Box w$	$\equiv$	$w \wedge \circ_w \Box w$
5.1.3	$\models$	$t_1\ gets\ t_2$	$\equiv$	$(more \supset ((\circ t_1) = t_2)) \wedge \circ_w (t_1\ gets\ t_2)$
5.1.4	$\models$	$stable\ t$	$\equiv$	$(more \supset ((\circ t) = t)) \wedge \circ_w\ stable\ t$
5.1.5	$\models$	$halt\ w$	$\equiv$	$(empty \equiv w) \wedge \circ_w\ halt\ w$
5.1.6	$\models$	$final\ w$	$\equiv$	$(empty \supset w) \wedge \circ_w\ final\ w$
5.1.7	$\models$	$t_1 \approx t_2$	$\equiv$	$(t_1 = t_2) \wedge \circ_w t_1 \approx t_2$

Beweis:

Zu 5.1.1 Sei (σ, i) ein festes, aber beliebiges zulässiges Intervall. Dann ist
$\mathcal{M}_{(\sigma,i)}(more \wedge \circ_w w) = true$
gdw. $\mathcal{M}_{(\sigma,i)}(more) = true$ und $\mathcal{M}_{(\sigma,i)}(\circ_w w) = true$
gdw. $|\ \sigma\ | > i$ und $\mathcal{M}_{(\sigma,i+1)}(w) = true$
gdw. $\mathcal{M}_{(\sigma,i)}(\circ w) = true$
Mit 4.1.1.3 gilt dann 1.

Zu 5.1.2 Sei (σ, i) ein festes, aber beliebiges zulässiges Intervall. Dann ist
$\mathcal{M}_{(\sigma,i)}(w \wedge \circ_w \Box w) = true$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = true$ und $\mathcal{M}_{(\sigma,i)}(\circ_w \Box w) = true$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = true$ und $|\ \sigma\ | = i$ oder
$\mathcal{M}_{(\sigma,i)}(w) = true$ und $|\ \sigma\ | > i$ und $\mathcal{M}_{(\sigma,i+1)}(\Box w) = true$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = true$ und $|\ \sigma\ | = i$ oder
$|\ \sigma\ | > i$ und $\mathcal{M}_{(\sigma,j)}(w) = true\ \forall j : i < j \leq |\ \sigma\ |$
gdw. $\mathcal{M}_{(\sigma,j)}(w) = true\ \forall j : 1 \leq j \leq |\ \sigma\ |$
gdw. $\mathcal{M}_{(\sigma,j)}(\Box w) = true$
Mit 4.1.1.3 gilt dann 2.

Die Aussagen 3. bis 7. sind nun unmittelbare Folgerungen aus den Definitionen der jeweiligen abgeleiteten Konstrukte und der Aussage 2.

Zu 5.1.3 Nach 4.3.3.1 ist $t_1\ gets\ t_2 = \Box(more \supset (\circ t_1 = t_2))$.
Mit 5.1.2 folgt dann 3.

Zu 5.1.4 Nach 4.3.3.3 ist $stable\ t = t\ gets\ t$.
Mit 5.1.3 folgt dann 4.

Zu 5.1.5 Nach 4.3.4.1 ist $halt\ w = \Box(w \equiv empty)$
Mit 5.1.2 folgt dann 5.

Zu 5.1.6 Nach 4.3.4.3 ist $final\ w = \Box(empty \supset w)$
Mit 5.1.2 folgt dann 6.

Zu 5.1.7 Nach 4.3.5.1 ist $(t_1 \approx t_2) = \Box(t_1 = t_2)$
Mit 5.1.2 folgt dann 7.

Bemerkung: Für den Sometimes-Operator gilt die 5.1.2 entsprechende Rekurrenzrelation $\Diamond w \equiv w \vee \circ_w w$ nicht. Dies liegt daran, daß $\circ_w w$ am Intervallende erfüllt wird, ohne daß w erfüllt worden sein muß.

Beispiel: Sei $M = (D, \Sigma, \mathcal{M})$ ein Standardmodell mit $\Sigma = \{\sigma\} = \{x \to 1\}$. w sei die Formel $x = 2$. Dann ist
$\mathcal{M}_{(\sigma,o)}(\Diamond x = 2) = \mathit{false}$ und
$\mathcal{M}_{(\sigma,o)}((x = 2) \vee \circ_w \Diamond(x = 2)) = \mathit{true}$ da $\mathcal{M}_{(\sigma,o)}(\circ_w \Diamond x = 2) = \mathit{true}$ ist.

Es gilt jedoch

Satz 5.2: Sei $M = (D, \Sigma, \mathcal{M})$ ein Modell und w eine Formel. Dann gilt $\models \Diamond w \equiv w \vee \circ\Diamond w$.

Beweis: Sei (σ, i) ein festes, aber beliebiges zulässiges Intervall. Dann ist
$\mathcal{M}_{(\sigma,i)}(w \vee \circ\Diamond w) = \mathit{true}$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = \mathit{true}$ oder $\mathcal{M}_{(\sigma,i)}(\circ\Diamond w) = \mathit{true}$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = \mathit{true}$ oder $|\sigma| > i$ und
$\mathcal{M}_{(\sigma,i+1)}(\Diamond w) = \mathit{true}$
gdw. $\mathcal{M}_{(\sigma,i)}(w) = \mathit{true}$ oder $|\sigma| > i$ und $\exists j > i : \mathcal{M}_{(\sigma,j)}(w) = \mathit{true}$
gdw. $\exists j \geq i : \mathcal{M}_{(\sigma,j)}(w) = \mathit{true}$
gdw. $\mathcal{M}_{(\sigma,i)}(\Diamond w) = \mathit{true}$

Corollar 5.1: Sei $M = (D, \Sigma, \{\mathcal{M})$ ein Modell und w eine Formel. Dann gilt:
$\models \Diamond w \equiv w \vee (\mathit{more} \wedge \circ_w \Diamond w)$.

6 Ausblick

Ein wesentliches Ziel unserer gegenwärtigen Forschungen ist die Entwicklung und Implementierung einer temporallogischen Programmiersprache mit folgenden Eigenschaften:

- Programme sind syntaktisch nichts weiter als Formeln einer geeigneten, zugrundeliegenden Temporallogik. Diese soll sowohl zukunftsbezogene Operatoren wie always-in-the-future, next etc. als auch vergangenheitsbezogene Operatoren wie always-in-the-past, previous etc. enthalten, sowie über chop die Möglichkeit bieten, sequentielle Konstrukte auszudrücken. Dies wird durch die vorliegende Logik EITeL gewährleistet.
- Berechnet wird ein (Zeit-) Intervall, auf dem die dem Programm entsprechende Formel erfüllt (*wahr* ist.
- Im Unterschied zu den meisten bestehenden temporallogischen Sprachen erfolgt die Berechnung nicht durch Resolution, sondern durch Transformation des Programms unter Verwendung von Rekurrenzformeln.

Durch die vorliegende Arbeit werden die logischen Grundlagen dieses Vorhabens bereitgestellt.

7 Literaturverzeichnis

[1] Abadi, M. and Manna, Z.: Temporal Logic Programming, J. Symbolic Computation, 8 (1989), pp. 277-295.

[2] Gabbay, D.: Modal and Temporal Logic Programming, Temporal Logics and their Applications, Academic Press (1987).

[3] Hrycej, T.: Temporal Prolog, ECAI-88, pp. 296-301.

[4] Kowalski, R.A.: Predicate Logic as Programming Language, IFIP-74, pp. 569-574.

[5] Kowalski, R.A.: Algorithm = Logic + Control, CACM, 22, 7 (1979), pp. 424-436.

[6] Kowalski, R.A. and Sergot, M.J.: A Logic Based Calculus of Events, New Generation Computing 4 (1986), pp. 67-95.

[7] Moszkowski, B.C.: Executing Temporal Logic Programs, Cambridge University Press (1986).

[8] Prior, A.N.: Past, Present and Future, Clarendon Press, Oxford (1967).

[9] Tang, T.G.: Temporal Logic CTL + Prolog, Journal of Automated Reasoning 5 (1989), pp. 49-65.