Inhaltsverzeichnis

Kapitel 1

Motivation und Aufgabenstellung

1.1 Einleitung

Heutige Datenbank–Management–Systeme (DBMS) basieren auf dem Transaktionsparadigma. Jede Folge zusammengehöriger Operationen im Anwendungsgebiet („Miniwelt“) wird als Transaktion (TA) modelliert. Transaktionen realisieren die ACID-Eigenschaften[1] und sind aus Sicht des Systems völlig voneinander unabhängig. Wenn Abhängigkeiten zwischen Transaktionen bestehen, muß die Anwendung diese implementieren. Dieses Vorgehen ist mit einem hohen (Programmier–) Aufwand verbunden für eine Verwaltungsarbeit, die eigentlich nicht im Anwendungsprogramm erledigt werden sollte.

Transaktionen können beispielsweise miteinander gekoppelt sein, wenn sich die Abbildung der Miniwelt auf mehrere, vernetzte DBMS verteilt und schreibende Datenbankzugriffe koodiniert und synchronisiert werden müssen. Beispielsweise dürfen mehrere Transaktionen nur gemeinsam erfolgreich beendet werden; wenn eine von ihnen abbricht, werden alle abgebrochen. Zur Realisierung muß ein Zwei-Phasen-Commit-Protokoll implementiert sein. Ein anderer Fall ist, daß ein Datenbank–Anwendungsprogramm zu viele Objekte modifiziert oder seine Ausführung zu lange dauert, als daß es unter Effizienzgesichtspunkten in *einer* TA realisiert werden könnte. Als ein weiterer Aspekt kann es wünschenswert sein, die Isolation von Transaktionen teilweise aufzuheben und z.B. *dirty read* zuzulassen, die Zuordnung von Transaktionen zu Kompensationstransaktionen zu formalisieren, oder sogar den gesamten Kontrollfluß der Anwendung stabil zu verwalten.

So wurden eine Reihe von Transaktionsmodellen entwickelt, wie z.B. *geschachtelte Transaktionen* [Moss81] und *Sagas* [GaSa87]. Die strukturellen Eigenschaften dieser Modelle lassen sich auf einen Satz von Transaktionsbeziehungen[2] und deren Verwaltung abbilden, also in einem einheitlichen Modell und einer konsistenten Notation beschreiben.

Im Rahmen des ConTract-Modells [ReutWae91], aber auch zur Modellierung von geschachtelten Transaktionen, Sagas oder zur Synchronisation von Transaktionen, z.B. bei einfachen Leseabhängigkeiten (dirty read), sind Transaktionen nicht unabhängig. Es bestehen stattdessen Beziehungen der Art

> *„Wenn T_v in den Zustand* aborted *übergeht, dann auch T_k.“* Diese Beziehung kann z.B. zusammen mit der folgenden zur Modellierung geschachtelter Transaktionen herangezogen werden.

[1] Ein engl. Akronym für *atomare Operation, konsistenzerhaltender Verarbeitungsschritt, isolierte Einheit, dauerhafte Wirkung.*

[2] Im folgenden meist kurz als *Beziehungen* bezeichnet.

„Das Commit von T_v liegt zeitlich nach dem Commit von T_k."

„Wenn T_1 abbricht, muß T_2 gestartet werden." Diese Beziehung löst beispielsweise im Saga–Modell den Start der Kompensation aus.

„T_1 geht genau dann in den Zustand committed *über, wenn auch T_2 erfolgreich ist."* Diese Eigenschaft soll beispielsweise für verteilte Transaktionen eines ConTract–Steps erfüllt sein.

Mit einem kleinen Satz derartiger Beziehungstypen lassen sich nahezu alle bekannten Transaktionstypen strukturell beschreiben [ChRa90]. Bisher existiert noch keine formale Beschreibung der Bedeutung dieser Beziehungen, die notwendig ist, um die Transaktionsverwaltung mit allgemeinen Beziehungsmustern zu operationalisieren. Dazu versucht diese Arbeit einen Beitrag zu leisten, indem sie die Operationale Semantik von Beziehungen näher untersucht.

1.2 Unterschiede zu anderen Transaktionsmodellen

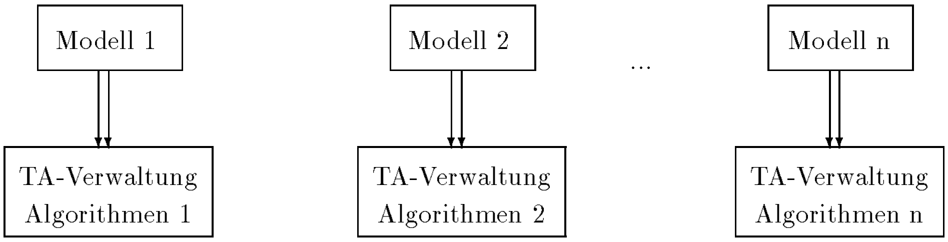

Abbildung 1.1: Transaktionsmodelle

In den erwähnten Transaktionsmodellen werden spezielle Transaktions-Verwaltungsalgorithmen für die jeweils verwendeten Transaktions„typen" beschrieben: bei [Moss81] ist beispielsweise spezifiziert, wie der Ablauf beim *Commit* einer Kind-Transaktion aussieht.

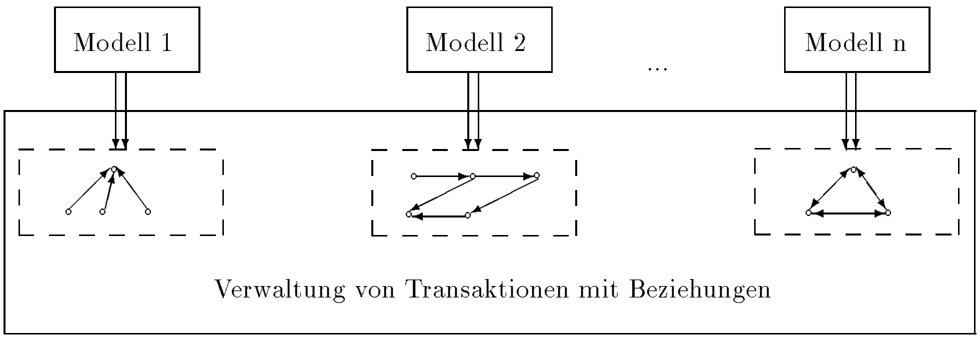

Abbildung 1.2: Transaktionen mit Beziehungen

Im Ansatz der Verwaltung von Transaktionen mit Beziehungen gibt es keine ausgezeichneten Transaktionstypen, sondern gleichartige Transaktionen. Es ist von einer Transaktion z.B. nicht bekannt, daß sie die Rolle einer Vater-Transaktion oder einer Kompensationstransaktion übernimmt. Es sind allein die entsprechenden Beziehungen spezifiziert. Die Transaktionsverwaltung muß mit der Kenntnis dieser Beziehungen die korrekten Aktivitäten entfalten. Im Modell der geschachtelten Transaktionen ist das beispielsweise eine „Abort-Abhängigkeit“ zwischen der (Vater-) Transaktion und anderen (Kind-) Transaktionen (wenn der Vater abbricht, dann auch das Kind), sowie zeitliche Beziehungen (das Commit der Kinder liegt vor dem Commit des Vaters).

Alle spezifizierten Beziehungen zwischen den Transaktionen eines Systems werden in das sogenannte *Transaktions–Beziehungs–Netz* (kurz *TBN*[3]) eingetragen: Ein TBN ist ein Graph. Die Knoten des Graphen sind Transaktionen mit der Binnenstruktur des Zustandsautomaten (siehe Abbildung 1.4). Die gerichteten Kanten des Graphen repräsentieren Beziehungen zwischen (den Zuständen von) zwei Transaktionen. Die Kanten sind mit dem Beziehungstyp (siehe Kapitel 1.3) benannt. Zwischen zwei Knoten des Graphen können mehrere Kanten existieren.

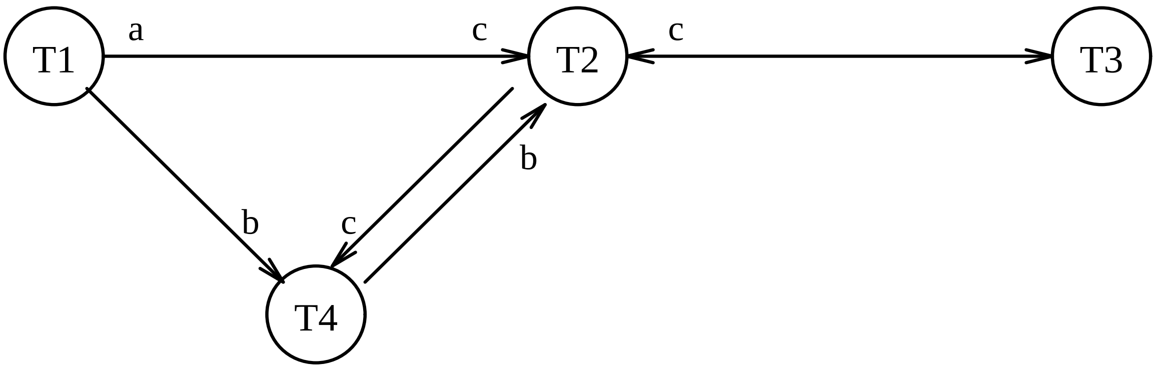

Abbildung 1.3: Beispiel–TBN

Abbildung 1.3 zeigt ein Beispiel für ein Transaktions–Beziehungs–Netz. Der Zustandsübergang einer Transaktion T hat potentiell Auswirkungen auf sämtliche Transaktionen mit direkten und indirekten Abhängigkeitsbeziehungen zu T im TBN.

Zur Beschreibung der Semantik der Beziehungen gehe ich von einem einfachen Transaktions–Zustandsautomaten aus, der allerdings noch erweitert werden wird. Eine Transaktion befindet sich in einem von vier Zuständen:

b(T): Die Transaktion T wurde gestartet und ist aktiv (*running*).

p(t): Die Transaktion T ist *prepared–to–commit*. Sie kann mit `EOT(T)` erfolgreich beendet werden. Diese Eigenschaft ist vom DBMS garantiert. Andererseits ist es möglich, sie mit dem Befehl `ABORT(T)` zurückzusetzen.

a(T): T wurde abgebrochen und ist im Endzustand *aborted*.

c(T): T ist im Endzustand *committed*, d.h. erfolgreich abgeschlossen. Ihre Auswirkungen sind dauerhaft verfügbar.

[3]Manchmal auch *TDN* für *transaction dependency net*.

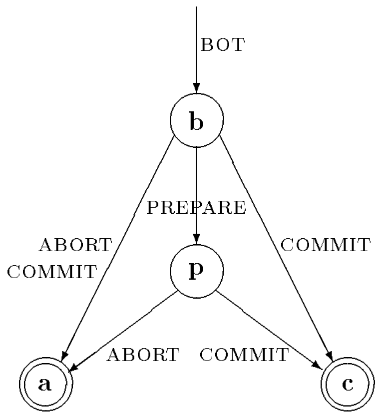

Abbildung 1.4: Transaktions–Automat

Neben dem DBMS hat eine weitere Systemkomponente, der Transaktions–Beziehungsnetz–Manager, Zugriff auf den TA-Automaten. Transaktionen sind nach dem klassischen ACID-Prinzip Objekte mit einem Zustand, der gesteuert durch die Aufrufe „begin–of–transaction“ (`BOT`), „prepare–to–commit“ (`PREPARE`), „end–of–transaction“ (`EOT`) und `ABORT` gemäß dem TA-Automaten aus Abbildung 1.4 geändert werden kann.

Der Transaktionsautomat besitzt folgende relevante Eigenschaften:

- Es gibt genau zwei Endzustände.
- Der Automat ist nicht zyklisch. Wenn eine Transaktion einen Zustandsübergang durchgeführt hat, kann sie nie wieder den vorherigen Zustand erreichen.
- In der Relation *ist–Vorgänger–von* zwischen den Zuständen gibt es eine Halbordnung.

Diese speziellen Eigenschaften ermöglichen einige Schlußfolgerungen, die in Kapitel 2.2.1 diskutiert werden.

1.3 Beziehungstypen

Transaktionsbeziehungen (als Synonyme verwende ich die Begriffe *Beziehung*, *Abhängigkeit* und *dependency*) sind zweistellige Relationen zwischen Transaktionen. Eine Abhängigkeit besteht also immer genau zwischen zwei Transaktionen T_1 und T_2.

Sobald drei oder mehr Transaktionen interagieren, wird dieses System auf paarweise Beziehungen abgebildet: es entsteht ein nichttriviales TBN.

Die zu untersuchenden Beziehungen[4] zwischen Transaktionen lassen sich in drei Kategorien einteilen:

[4] Diese in Transaktionsmodellen verwendeten Beziehungstypen werden im folgenden meist als *elementare Beziehungen* (im Gegensatz zu abgeleiteten, aggregierten Beziehungen) bezeichnet.

1. **Logische Abhängigkeiten:** Dieses in den Transaktionsmodellen häufig verwendete Beziehungsmuster legt Abhängigkeiten des Ausgangs (commit oder abort) einer Transaktion T_1 vom Endzustand einer anderen Transaktion T_2 fest (z.B. im Modell geschachtelter Transaktionen die abort–Abhängigkeit einer Kind–Transaktion T_k von der Vater–Transaktion T_v: *wenn T_v abbricht, dann auch T_k.*).

Notation	prädikatenlog. Definition	Interpretation
$T_1 \rightarrow_c T_2$	`c(T`$_1$`)` $\Rightarrow$ `c(T`$_2$`)`	Wenn T_1 mit *committed* beendet wird, dann muß auch T_2 den Zustand *committed* erreichen.
$T_1 \rightarrow_a T_2$	`a(T`$_1$`)` $\Rightarrow$ `a(T`$_2$`)`	Wenn T_1 mit *aborted* beendet wird, dann muß auch T_2 den Zustand *aborted* erreichen.
$T_1 \leftrightarrow_c T_2$	`c(T`$_1$`)` $\Leftrightarrow$ `c(T`$_2$`)`	T_1 wird genau dann erfolgreich beendet, wenn auch T_2 erfolgreich endet.
$T_1 \leftrightarrow_a T_2$	`a(T`$_1$`)` $\Leftrightarrow$ `a(T`$_2$`)`	T_1 wird genau dann nicht erfolgreich beendet, wenn auch T_2 nicht erfolgreich endet.
$T_1 \; {}_a{\rightarrow}_c \; T_2$	`a(T`$_1$`)` $\Rightarrow$ `c(T`$_2$`)`	Wenn T_1 abbricht, dann muß T_2 den Zustand *committed* erreichen.
$T_1 \; {}_c{\rightarrow}_a \; T_2$	`c(T`$_1$`)` $\Rightarrow$ `a(T`$_2$`)`	Wenn T_1 erfolgreich endet, dann muß T_2 abbrechen.
T_1 — T_2		T_1 und T_2 sind unabhängig; zwischen ihnen besteht die sog. *leere Beziehung*.

Tabelle 1.1: Logische Beziehungen

Bei den logischen Beziehungen wird von den zeitlichen Verhältnissen abstrahiert. Eine logische Beziehung ist erfüllt, wenn beide Transaktionen ihre entsprechenden Endzustände bereits erreicht haben, aber auch dann, wenn garantiert ist, daß die Beziehung irgendwann einmal erfüllt sein wird: spätestens zu dem Zeitpunkt, wenn beide in Beziehung stehenden Transaktionen ihren Endzustand erreicht haben werden. Diese Garantie kann aber in manchen Fällen auch schon früher gegeben werden: wenn T_k aus dem obigen Beispiel nicht vor T_v erfolgreich beendet wird, kann T_k noch abgebrochen werden, falls die Situation `a(T`$_v$`)` eintritt.

2. **Kontrollflußbeziehungen:** Dieser Beziehungstyp ist dadurch charakterisiert, daß ein Übergang im Zustandsautomaten der Transaktion T_1 „sofort“ in die Aktivierung, Beendigung oder in den Abbruch der anderen Transaktion T_2 umgesetzt wird.

Notation	prädikatenlog. Definition	Interpretation
$T_1 \; {}_c{\mapsto}_b \; T_2$	`c(T`$_1$`)` $\Rightarrow$ `b(T`$_2$`)`	Sobald T_1 mit *committed* beendet wird, wird T_2 gestartet.
$T_1 \; {}_a{\mapsto}_b \; T_2$	`a(T`$_1$`)` $\Rightarrow$ `b(T`$_2$`)`	Sobald T_1 abbricht, wird T_2 gestartet.
$T_1 \; {}_b{\mapsto}_c \; T_2$	`b(T`$_1$`)` $\Rightarrow$ `c(T`$_2$`)`	T_2 wird erfolgreich beendet, sobald T_1 startet.
$T_1 \; {}_b{\mapsto}_a \; T_2$	`b(T`$_1$`)` $\Rightarrow$ `a(T`$_2$`)`	T_2 wird abgebrochen, sobald T_1 startet.
$T_1 \mapsto_b T_2$	`b(T`$_1$`)` $\Rightarrow$ `b(T`$_2$`)`	Sobald T_1 startet, startet auch T_2.
$T_1 \leftrightarrow_b T_2$	`b(T`$_1$`)` $\Leftrightarrow$ `b(T`$_2$`)`	T_1 startet genau dann (und im selben Augenblick), wenn auch T_2 startet.

Tabelle 1.2: Kontrollflußbeziehungen

Beispiel zur Verwendung der Kontrollflußbeziehungen: Der Abbruch einer Saga-Teiltransaktion verursacht das Starten der zugehörigen Kompensationstransaktion. Dies wird mit der Beziehung ${}_a{\mapsto}_b$ realisiert.

Die in der Tabelle aufgeführte „prädikatenlogische Definition" der Kontrollflußbeziehungen ist nicht vollständig: Zusätzlich muß gefordert werden, daß z.B. bei der Beziehung ($T_1 \;{}_{c}{\longmapsto}_{b}\; T_2$ `b(T`$_2$`)` unmittelbar auf `c(T`$_1$`)` folgt. Wenn der gewünschte Zielzustand von T_2 bereits früher erreicht wurde, ist die Situation zur Kontrollflußbeziehung ebenfalls konform; In diesem Fall wird auf T_2 natürlich keine Operation ausgeführt. Die Definitionen müssen also nach dem Muster des folgenden Beispiels ergänzt werden: ($T_1 \;{}_{c}{\longmapsto}_{b}\; T_2$) ist definiert durch

$$\texttt{c(T}_1\texttt{)} \Rightarrow \texttt{b(T}_2\texttt{)} \quad \wedge (\Delta t[\texttt{c(T}_1\texttt{)}, \texttt{b(T}_2\texttt{)}] = \text{„klein"} \vee t[\texttt{b(T}_2\texttt{)}] < t[\texttt{c(T}_1\texttt{)}])$$

Kontrollflußbeziehungen, die das Commit der Transaktion T_2 bewirken, werden nicht betrachtet, da das commit einer Transaktion nicht außerhalb der Transaktion selber erzwungen werden kann.

Bei Beziehungen, die den Abbruch einer Transaktion erzwingen können, ist die Klassifizierung in logische Beziehungen und Kontrollflußbeziehungen willkürlich. Der Abbruch ist bei aktiven Transaktionen immer möglich und kann nicht verhindert werden. Er kann folglich einerseits sofort ausgeführt werden, andererseits – von den belegten Systemressourcen abgesehen – ohne negative Auswirkungen auch etwas verzögert werden[5]. Aus diesem Grund behalte ich die Definition der logischen Beziehungen bei und definiere nur diejenigen Beziehungen, die eine Abhängigkeit zum Zustand *running* beschreiben, als Kontrollflußbeziehung.

Aus einer Kontrollflußbeziehung folgt jeweils ihr entsprechendes „logisches Pendant", d.h. eine Beziehung, die nur fordert, *daß* ein Zustand eintritt, aber nicht *wann*: so ist die Beziehung $T_1 \;{}_{c}{\longmapsto}_{b}\; T_2$ beispielsweise noch nicht erfüllt, wenn `c(T`$_1$`)` erreicht und `BOT(T`$_2$`)` aufgerufen wurde, sondern erst, sobald T_2 nachweislich im Zustand *running* (oder in einem seiner Folgezustände) ist. Nur dieser Aspekt der

Der Endzustand von T_2 kann auch durch die Tatsache determiniert werden, ob T_1 im Zustand *running* ist. Beziehungen aus dieser zweiten – soweit mir bekannt nirgends verwendeten – Klasse ordne ich aus Gründen der Symmetrie unter den Kontrollflußbeziehungen ein.

3. Zeitliche Wechselwirkungen: Temporale Beziehungen definieren im Gegensatz zu den logischen Beziehungstypen Abhängigkeiten zwischen Zustands*übergängen*, nicht zwischen Zuständen. Dabei werden jeweils alle Zustandsübergänge in einen Zustand, egal von welchem Zustand ausgehend, gleich behandelt.

Notation	Bedeutung
$T_1 <_c T_2$	Das commit von T_1 liegt zeitlich vor dem commit von T_2.
$T_1 \;{}_{c}{<}_{b}\; T_2$	T_2 wird gestartet, nachdem T_1 erfolgreich beendet wurde.
$T_1 \;{}_{a}{<}_{b}\; T_2$	Der Beginn von T_2 liegt zeitlich nach dem Abbruch von T_1.
$T_1 \;{}_{b}{<}_{c}\; T_2$	T_1 muß gestartet sein, bevor T_2 erfolgreich endet.
$T_1 <_b T_2$	T_1 beginnt zeitlich vor T_2.

Tabelle 1.3: zeitliche Beziehungen

Folgende Frage drängt sich auf: Was soll geschehen, wenn ein spezifizierter Zustandsübergang nicht stattfindet, z.B. wenn wenn `c(T`$_1$`)` vor `c(T`$_2$`)` liegen soll, T_1 aber abbricht? Dieser Fall soll zugelassen werden. Die Beziehung ist dann wirkungslos. Falls dieses Verhalten mit der gewünschten Semantik nicht übereinstimmt, muß zusätzlich die entsprechende logische Beziehung explizit angegeben werden. Im obigen Beispiel wäre das möglicherweise die Beziehung $T_1 \leftrightarrow_c T_2$.

[5] Im Extremfall bis zu dem Zeitpunkt, wo die Transaktion erfolgreich beendet werden will, daraufhin aber abgebrochen wird.

In dieser Studienarbeit gebe ich nur einige Hinweise, wie zeitliche Beziehungen integriert werden könnten. Ihre erschöpfende Behandlung würde den Rahmen der Arbeit sprengen.

An dieser Stelle soll gleich einem möglichen Mißverständnis vorgebeugt werden: Die Pfeile, mit denen die Beziehungen benannt sind, implizieren *nicht*, daß nur Auswirkungen in Pfeilrichtung stattfinden. Ein Zustandsübergang der Transaktion an der Pfeilspitze kann durchaus Einfluß auf die andere Transaktion haben. Als Beispiel kann nochmals die Abort-Abhängigkeit $T_v \rightarrow_a T_k$ dienen: Wenn `c(T`$_k$`)` eintritt, ist der Abbruch von T_v verboten.

1.4 ConTracts und APRICOTS

Die Verwaltung von Transaktionsbeziehungen, wie sie dieser Text beschreibt, ist zur Verwendung als ein Modul im ConTract–System vorgesehen (siehe Abbildung 1.5). In einem Satz zusammangefaßt ist ein *ConTract* ein Anwendungsprogramm, das durch eine Menge von gegebenen Aktionen, sog. *Steps*, und eine explizite Beschreibung des Kontrollflusses gegeben ist, und das unverwundbar durch Systemfehler in jedem Fall ein korrektes Ende erreicht.

Im ConTract-Modell gibt es eine Arbeitsteilung zwischen dem *Transaction Manager* TM [Lutz] (mit dem *Transaktions-Beziehungs-Netz-Manager* TBNM als Komponente) und dem *ConTract-Manager* CM [Wahl] ab. Der CM interpretiert das Skript, das den Kontrollfluß des ConTracts beschreibt, und wickelt entsprechend die einzelnen Verarbeitungsschritte (Steps) ab. Steps beinhalten Transaktionen, die unter Kontrolle des TM stehen. Diese Transaktionen dürfen vom TM keinesfalls – etwa aufgrund von Kontrollflußbeziehungen – aus Sicht des CM willkürlich und zu früh gestartet werden. Beispielsweise kann es passieren, daß eine Transaktion aufgrund der eingetragenen Beziehungen starten könnte, aber der Eingabekontext des Steps dieser Transaktion noch nicht zur Verfügung steht.

Als Konsequenz liegt die Kontrollflußverwaltung ausschließlich beim CM. Kontrollflußbeziehungen sind im aktuellen Prototyp des ConTract-Modells (Apricots – A PRototype Implementation of the COnTract System) auf ihr logisches Pendant reduziert, was `BOT`-Aufrufe angeht.

Der TBNM hält – zumindest aus seiner Sicht – das vollständige Netz aller im System existierenden Beziehungen. Auf Anforderung kann er natürlich Teilnetze „herausreichen“ und (als inverse Operation) integrieren, sodaß eine Migration ermöglicht wird.

Weiterhin läßt sich auf den TBN-Operationen aufbauend ein Protokoll definieren, um ein verteiltes Netz zu implementieren. Zu diesem Thema siehe Kapitel 4.3.3 und [Lutz].

1.5 Aufgabenstellung

In dieser Arbeit werden die Transaktionsbeziehungen analysiert und als globales Ziel eine operationale Semantik für die elementare Beziehungsverwaltung entwickelt. Im einzelnen werden nach der Erörterung der auftretenden Probleme mehrere Lösungsansätze verglichen; einer davon wird als Basis der Operationen zur Verwaltung von allgemeinen Transaktionsbeziehungen ausgewählt und als eine Komponente im ConTract–Prototyp implementiert.

Transaktionen mit Beziehungen sollen verwaltet werden: Als Zwischenebene zwischen Anwendungsprogrammen, die Transaktionen verwenden und Beziehungen eintragen, und dem TM, der (evtl. mehrere) SQL–RM bedient, ist die TBN-Verwaltung eingeschoben. Sie filtert Transaktionskommandos (`BOT`, `COMMIT`, ...) an den TM und setzt sie in zu den Beziehungen konforme Aktivitäten und Kommandos um. Das bedeutet konkret, daß folgende Fragen beantwortet werden müssen, wenn

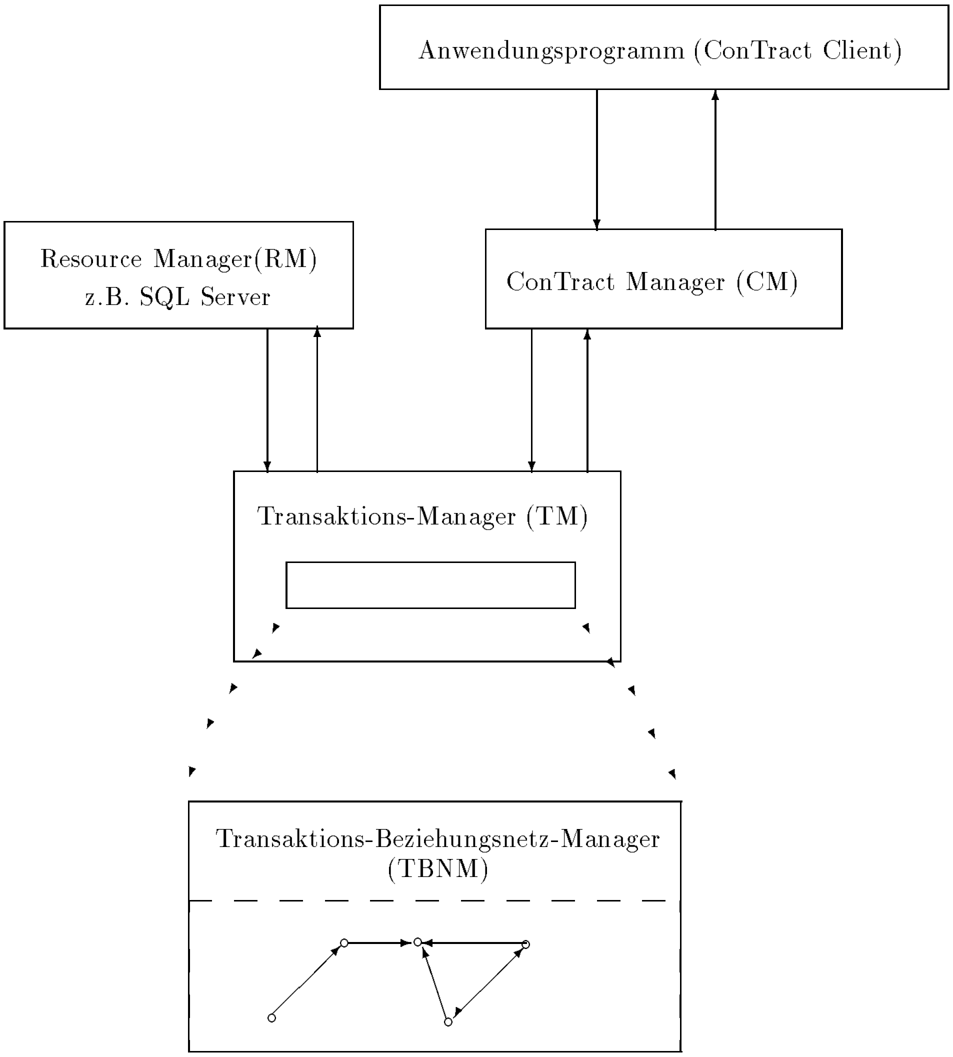

Abbildung 1.5: Ausschnitt aus der ConTract–Systemarchitektur

eine weitere Beziehung hinzugefügt werden soll, oder wenn ein Zustandsübergang einer Transaktion angefordert wird:

1. Steht die Operation im Widerspruch zu den bestehenden Beziehungen oder Zuständen irgendwelcher Transaktionen?

   Die Transaktionsverwaltung darf angeforderte Zustandsübergänge nur dann vollziehen, wenn sie mit dem TBN konform sind. Käme es zu einem Widerspruch, muß die Operation zurückgewiesen werden.

2. Welche abhängigen Zustandsübergänge werden direkt oder indirekt durch die Operation gefordert?

   Einige Transaktionen in der „transitiven Hülle“ der Beziehungswirkungen der Transaktion müssen i.a. bestimmte Zustände erreichen. Wenn diese Zustandsübergänge möglich sind (siehe Frage 1), sollten sie atomar zusammen mit dem auslösenden Übergang vollzogen werden.

3. Welche Restriktionen ergeben sich für zukünftige Zustandsübergänge und Beziehungen?

4. Kann die Operation (einschließlich der abhängigen Zustandsübergänge) sofort ausgeführt, oder muß sie verzögert werden?

5. Welche der verzögerten Zustandsübergänge werden jetzt möglich?

   Eventuell hat die TBN-Verwaltung Zustandsübergänge mehrerer Transaktionen verzögert, und der geforderte Zustandsübergang ist die letzte Bedingung gewesen, die erfüllt werden mußte, um diese Zustandsübergänge zu ermöglichen.

Unter Berücksichtigung dieser Fragen, die in Kapitel 2 erörtert werden, wird eine Datenstruktur für Transaktionsbeziehungen (Kapitel 3) und für das TBN entworfen. Darauf aufbauend werden elementare Operationen spezifiziert (Kapitel 4) und implementiert (siehe Kapitel 5), um

- Beziehungen einzutragen und zu entfernen.
- Zustandsübergänge von Transaktionen in Übereinstimmung mit den Beziehungen im Netz durchzuführen.
- Das Netz ganz oder teilweise zu einem zweiten TBN-Manager, z.B. auf einem weiteren Rechnerknoten, zu migrieren. Diese Operation wird im Rahmen des ConTract-Modells benötigt, um die Unverwundbarkeit der Ausführung im System zu realisieren.

Die ersten beiden Funktionstypen basieren auf einem Verfahren, das von einer Zustandsänderung einer Transaktion ausgeht und über die gegebenen Beziehungen Zustandsänderungen anderer Transaktionen ermittelt. Dieser für die TBN–Verwaltung zentrale Algorithmus ist in Kapitel 4.7 beschrieben.

Kapitel 2

Transaktions–Beziehungs–Netze (TBN)

2.1 Einleitung

In diesem Kapitel werde ich mich genauer mit der Bedeutung und Interpretation der Beziehungen befassen. Anschließend werde ich untersuchen, welche Probleme auftreten, wenn wir mehrere Transaktionen über Beziehungen miteinander verknüpfen. Diese Probleme werden in Konzepte und Strategien der TBN-Verwaltung umgesetzt.

2.2 Beschreibung der Beziehungssemantik

Der Interpretation der Beziehungen (und somit der TBN–Semantik) liegen einige implizite Annahmen zugrunde, die erzwingen, daß nicht triviale Lösungen (wie z.B. keine Transaktion zu starten) gewählt werden können, um alle Probleme zu lösen:

- Normalerweise impliziert ein `BOT`–Aufruf einer Transaktion den Wunsch nach dem Endzustand *committed*.
- Der gewünschte (und vermutete) Ausgang des `EOT`–Aufrufs einer Transaktion ist der Endzustand *committed*.
- Installierte Transaktionen werden üblicherweise starten (und erfolgreich enden).

2.2.1 Transaktionsautomat und Beziehungssemantik

Beziehungen zwischen Transaktionen müssen insbesondere bei den Kontrollflußbeziehungen, die Einfluß auf den Start einer Transaktion haben, bereits zwischen Transaktionen spezifiziert werden können, die noch nicht gestartet, also auch noch nicht im Zustand *running* sind. In herkömmlichen Datenbanksystemen entsteht eine Transaktion als Objekt aber erst in dem Zeitpunkt, in dem sie gestartet wird. Die zwei Schritte *Installiere eine Transaktion* und *Starte eine Transaktion* müssen getrennt werden. Transaktionen im neuen Zustand *installed* bzw. `i(t)` sind von der Transaktionsverwaltung identifizierbar. Ihr Name und Code-Zuordnung werden bereits stabil verwaltet. Aus dem Zustand `i(T)` gibt es einen Übergang in den Zustand `b(T)` über den Aufruf `BOT(T)`.

Vor allen weiteren Überlegungen muß der Transaktions-Zustandsautomat erweitert werden. Betrachten wir folgendes Beispiel: Transaktion T_1 sei *installed*, T_2 *running*, und es gelte die Beziehung $T_1 \; {}_b{\mapsto}_c \; T_2$. In dieser Situation bricht T_2 ab. Ein Abbruch kann vom TM nicht verhindert werden, es gilt also $\mathtt{a(T_2)}$. Ein Start von T_1 würde $\mathtt{c(T_2)}$ fordern, was im Widerspruch zum Abbruch von T_2 stünde. Was kann unternommen werden, damit die Beziehung erfüllt bleibt? T_1 darf *nie* mehr einen Zustandsübergang durchführen (der einzig mögliche wäre ja der nach $\mathtt{b(T_1)}$). Im Zustand *installed* kann sie auch nicht bleiben, da dies kein Endzustand ist, und alle Wege zu Endzuständen abgeschnitten sind. Abhilfe verspricht ein weiterer Endzustand, der *demand-not-run*, oder kurz *dnr* genannt wird. In diesen Zustand gibt es nur einen Übergang von *installed* aus. Die Beziehung aus dem Beispiel oben ist erfüllt, wenn T_1 in diesen Zustand wechselt.

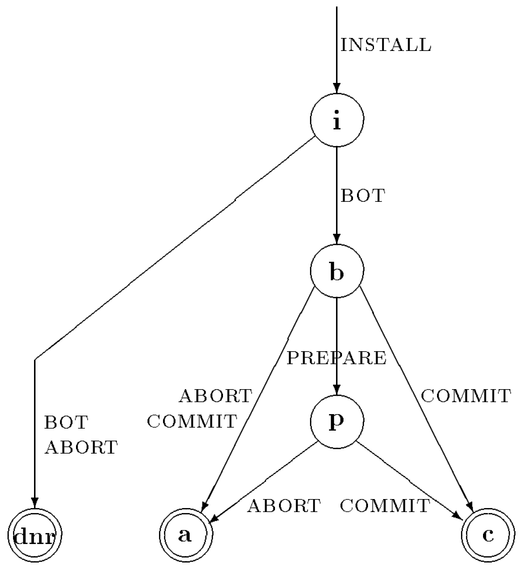

Abbildung 2.1: erweiterter Transaktionsautomat

Dieser dritte Endzustand wirft jedoch ein Problem bei den logischen Beziehungen auf: Bisher konnte man Umkehrschlüsse aufgrund der Regel

$$\neg c(T) \Leftrightarrow a(T)$$

ziehen. Daraus ergab sich eine plausible Symmetrie und Äquivalenz zwischen den logischen Beziehungen. Beispielsweise war $\leftrightarrow_c$ zu $\leftrightarrow_a$ äquivalent, sowie $\rightarrow_c$ zu ${}_a{\leftarrow}$ (siehe auch Kapitel 2.3). Diese Eigenschaften gehen jetzt verloren. Aus *nicht committed* folgt nur noch *aborted oder demand-not-run* bzw. aus *nicht aborted* gerade noch *committed oder demand-not-run*:

$$\begin{aligned} \neg c(T) &\Rightarrow a(T) \vee dnr(T) \qquad \text{bzw.} \\ \neg a(T) &\Rightarrow c(T) \vee dnr(T) \end{aligned}$$

Die oben genannten Äquivalenzen gelten nur noch unter der Voraussetzung *„Wenn beide Transaktionen auch tatsächlich laufen“*. Somit wird die Überprüfung, ob eine Kombination von Zuständen von Transaktionen bei gegebenen Beziehungen erlaubt ist, komplizierter.

Andererseits hat eine Transaktion im Zustand *demand-not-run* eine wesentliche Gemeinsamkeit mit Transaktionen im Zustand *aborted*: sie hat keine Änderungen externalisiert, d.h. für andere sichtbar gemacht. Aus *demand-not-run* sollte deshalb *aborted* folgen. Es ist allerdings nicht sinnvoll, diese beiden Zustände zu einem zusammenzufassen. Es könnte dann nämlich nicht mehr entschieden werden, ob bestimmte Kontrollflußbeziehungen (wie ${}_b{\mapsto}_c$) und zeitliche Beziehungen (wie $<_b$) erfüllt oder erfüllbar sind, sobald die Transaktion in einem Endzustand außer *committed* ist.

Um die Situation zu retten, definiere ich ein Prädikat `abrt(T)`, das genau dann erfüllt ist, wenn die Transaktion T im Zustand *aborted* oder im Zustand *demand-not-run* ist. Unter Verwendung dieses Prädikats gilt wieder die gewünschte Regel

$$\neg c(T) \Leftrightarrow abrt(T)$$

und die logischen Beziehungen behalten ihre uneingeschränkte Semantik. Diesen Vorteil erkauft man sich wiederum mit seltsamen Effekten bei Kontrollflußbeziehungen.

Ein Beispiel dafür ist folgende Fallstudie: Die Transaktionen T_1 und T_2 sind beide *installed*. Sie stehen neben der Beziehung $T_1 \ {}_a{\mapsto}_b \ T_2$ in weiteren Beziehungen zu anderen Transaktionen. Für T_1 wird ein Zustandsübergang nach dnr(T_1) durchgeführt. abrt(T_1) trifft zu. Dies impliziert BOT(T_2), obwohl T_1 im wörtlichen Sinne nicht abgebochen wurde, da T_1 nie gelaufen ist. Ich vermute, daß bei sinnvoller Anwendung der Beziehungen diese Situation nicht oder nur in von den Transaktionsmodellen nicht abgedeckten Fällen eintritt.

Bereits durch logische Beziehungen kann die Situation entstehen, daß für eine Transaktion T_a, die installiert, aber noch nicht aktiv ist, von der TBN-Verwaltung der Abbruch gefordert wird. Hier wäre es Verschwendung von Ressourcen, T_a erst zu starten, um sie daraufhin abbrechen und alle ihre Spuren verwischen zu können. Der Aufruf ABORT(T_a) soll daher für installierte Transaktionen T zugelassen und als ein Übergang in den Zustand *demand-not-run* realisiert werden. Als Seiteneffekt kann auch zur Behandlung der Kontrollflußbeziehungen diese Funktion mitverwendet werden. Der resultierende Zustandsautomat ist in Bild 2.1 dargestellt.

Der Automat ist durch folgende Axiome charakterisiert:

1.	$i(T) \Rightarrow b(T) \vee dnr(T)$
2.	$b(T) \Rightarrow p(T) \vee c(T) \vee a(T)$
3.	$p(T) \Rightarrow c(T) \vee a(T)$
4.	$p(T) \Rightarrow b(T)$
5.	$a(T) \Rightarrow b(T)$
6.	$c(T) \Rightarrow b(T)$
7.	$abrt(T) \Rightarrow a(T) \vee dnr(T)$

Tabelle 2.1: Axiome des erweiterten Transaktions-Zustandsautomaten

Installierte Transaktionen können aktiv werden, ihr Start kann aber auch durch einen sofortigen Abbruch verhindert werden (1 und 7). Eine gestartete Transaktion endet mit *commit* oder *aborted*. Der Weg dorthin kann über den Zustand *prepared* gehen (2 bis 5). Nur Transaktionen, die auch aktiv gewesen sind, können erfolgreich beendet werden (6).

2.2.2 Automatentheoretische Beschreibung

Werden zwei unabhängige Transaktionen T_1 und T_2 betrachtet, kombinieren sich ihre Zustandsautomaten zu dem sog. Potenzautomaten[1] aus Abbildung 2.2. Jeder Zustand in diesem Automat ist ein Tupel `(Zustand(`T_1`), Zustand(`T_2`))`. `d` steht für *demand-not-run*. Da sich die Zustände *running* und *prepared* hinsichtlich der erreichbaren Endzustände nicht unterscheiden, sind sie hier unter `b` zusammengefaßt.

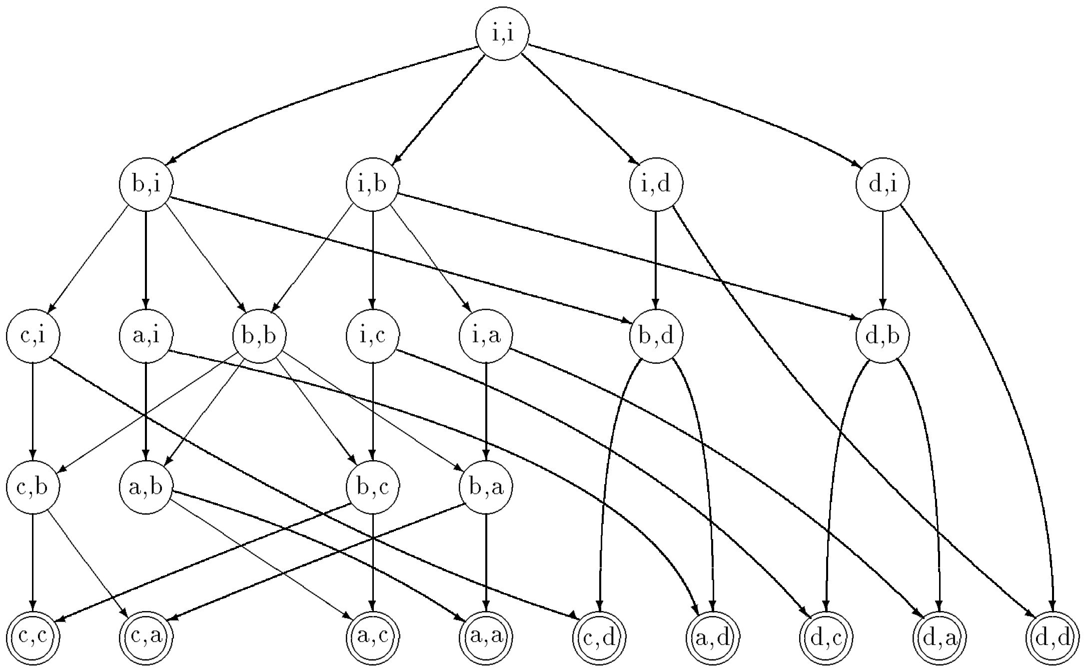

Abbildung 2.2: Transitionsdiagramm bei zwei Transaktionen

Beziehungen zwischen Transaktionen schränken die Übergangsrelation δ dieses Automaten ein: Neue Zustände oder Übergänge können nicht entstehen, aber es werden bestimmte Zustände und Übergänge „verboten" bzw. entfernt. Die einzelnen Beziehungstypen implizieren unterschiedliche Arten von Restriktionen:

- Logische Beziehungen entfernen Endzustände und die Pfade dorthin. Beispiel: Die Beziehung $T_1 \rightarrow_c T_2$ entfernt die Endzustände `(c,a)` und `(c,dnr)`.
- Kontrollflußbeziehungen sowie zeitliche Beziehungen schränken die möglichen Zustandsübergänge ein. Beispiel: Die Beziehung $T_1 <_c T_2$ entfernt die Übergänge $\delta((b,b)) = (b,c)$ und $\delta((i,c)) = (b,c)$.

Rekursiv werden durch das Streichen von Übergängen i.a. weitere Zustände (die nur mit diesen Übergängen) erreichbar sind) und durch das Streichen von Zuständen weitere Übergänge (in diese Zustände) verboten.

[1] Zur Konstruktion siehe [HU].

Auch bei der eingeschränkten Übergangsfunktion des modifizierten Automaten muß mindestens einer der Endzustände erreichbar bleiben (Genau dann können beide Transaktionen im verbliebenen Automat einen Endzustand erreichen). Andernfalls ist die durch den derart eingeschränkten Automaten repräsentierte Beziehung zwischen den beiden Transaktionen *inkonsistent*: Eine Transaktion, die weder erfolgreich beendet `c(T)`, noch abgebrochen werden kann (`a(T)`), die also weder starten darf, deren Startverbot (`dnr(T)`) andererseits auch nicht zulässig ist — so etwas kann nicht erlaubt sein.

Aus automatentheoretischer Sicht ist eine *inkonsistente Beziehung* dadurch definiert, daß im zugehörigen Automaten kein Endzustand erreichar ist.

Es muß beachtet werden, daß der Automat zu einem beliebigen Zeitpunkt modifiziert werden kann, indem z.B. eine weitere Beziehung integriert wird. Insbesondere ist nicht garantiert, daß beide Transaktionen noch im Zustand *installed* sind. Ein Teil des Automaten wird in diesem Fall zusätzlich gestrichen, da er nicht erreichbar ist. Bei dieser Sichtweise gehen die Zustände der Transaktionen in die Beziehungen ein. Auf jeden Fall kann ein Netz von Beziehungen nicht unabhängig von den Zuständen der Transaktionen interpretiert werden.

Bei einem Zustandsübergang einer Transaktion finden im Potenzautomaten so lange weitere Zustandsübergänge statt, wie nur genau ein Zustandsübergang erlaubt ist. Gerade diese Übergänge sind durch die Restriktionen der Beziehung determiniert. Die Zustandsänderungen sind nicht spontan: sie werden vom Aufruf der entsprechenden zustandsändernden Operation auf der betroffenen Transaktion bewirkt. Beispielsweise kann man eine Transaktion T nicht einfach in den Zustand *committed* versetzen — dazu muß erst `EOT(T)` aufgerufen werden.

Gestoppt wird beim ersten Zustand, von dem aus es noch Alternativen gibt. Weitere Zustandsübergänge werden erst wieder von den nächsten Aufrufen an den TM ausgelöst.

2.3 Eigenschaften der Beziehungen als logische Konnektoren

Betrachtet und vergleicht man die Automaten bei den einzelnen Beziehungstypen, so erkennt man einige Zusammenhänge: Beziehungen mit identischem Potenzautomat sind identisch. Beziehungen mit symmetrischem Potenzautomat sind symmetrisch.

2.3.1 Umbenennungsregel für Beziehungsnamen

Bei Beziehungen, die ausschließlich die Indizes a und c enthalten, ist die Benennung nicht eindeutig: Man erhält den zweiten Namen für den Beziehungstyp, indem man (in der ausführlichen Form mit beiden Indizes)

- die Pfeilrichtung umdreht und
- den Index c gegen a, sowie a gegen c vertauscht.

Insbesondere sind die Beziehungen $\leftrightarrow_c$ und $\leftrightarrow_a$ identisch, sowie die Beziehungen $\rightarrow_c$ und ${}_a\!\leftarrow$. Zur Normierung wird nur der jeweils erste Name verwendet.

Die Beziehungen $\leftrightarrow_c$, ${}_a\!\rightarrow_c$, ${}_c\!\rightarrow_a$ und $\leftrightarrow_b$ sind symmetrisch.

2.3.2 Verzögerte Zustandsübergänge

Wie bereits im Kapitel 1.4 erwähnt, können aus dem TBN abgeleitete Forderungen (sog. *demands*) nach einem Zustandsübergang nach *running* (`demand-run`) nicht immer sofort erfüllt werden, müssen also verzögert werden.

Das gleiche Problem stellt sich, wenn an die Transaktionsverwaltung eine Anfrage nach einem Zustandsübergang $X(T_1)$ gestellt wird. Dieser Übergang kann über Beziehungen (beispielsweise $T_1 \;{}_a\!\!\rightarrow_c T_2$) von einer installierten oder laufenden Transaktion T_2 $\mathtt{c(T_2)}$ fordern (`demand-commit`). Sofern die Transaktion T_2 nicht bereits $\mathtt{EOT(T_2)}$ erreicht hat oder zu einem früheren Zeitpunkt eine `EOT`-Anfrage gestellt hatte (die damals verzögert wurde, der jetzt aber entsprochen werden kann), muß der Zustandsübergang nach X verzögert werden.

Handelt es sich bei X um $\mathtt{a(T_1)}$, ist eine Verzögerung – praktisch gesehen – nicht sehr sinnvoll. Der Abbruch einer Transaktion kann nicht verhindert werden, also sollte er gleich ausgeführt werden, damit T_1 nicht unnötig Systemressourcen belegt. Ein sofortiger Abbruch erfordert zwar auch Ressourcen (es werden Nachrichten verschickt), andererseits werden auch beim regulären Ende der Transaktion Nachrichten verschickt. Im Beispiel wird T_2 zur *goldenen Transaktion*, die erfolgreich enden muß. Andernfalls tritt eine inkonsistente Situation auf, die im Beziehungsmodell nicht mehr behandelt werden und nur noch pragmatisch aufgelöst werden kann.

Handelt es sich bei X um $\mathtt{c(T_1)}$, dann bleibt als vernünftige Reaktion der Aufruf $\mathtt{PREPARE(T_1)}$. Ist $\mathtt{p(T_1)}$ erreicht, dann erkennt die TBN-Verwaltung auch zu einem späteren Zeitpunkt, daß T_1 einen commit-Wunsch gestellt hatte. Außerdem ist im Zustand *prepared* garantiert, daß *committed* erreicht werden kann: eine für die korrekte Verwaltung notwendige Eigenschaft (siehe [Lutz]). Dem Commit-Wunsch braucht die TBN-Verwaltung nicht unbedingt nachkommen; falls erforderlich, kann die Transaktion aus dem Zustand *prepared* noch abgebrochen werden.

Beispiel: Betrachten wir zwei laufende Transaktionen mit genau-dann-wenn-commit-Beziehung ($T_1 \leftrightarrow_c T_2$, $\mathtt{b(T_1)}$, $\mathtt{b(T_2)}$).
Die Anfrage $\mathtt{EOT(T_1)}$ wird in $\mathtt{PREPARE(T_1)}$ umgesetzt.
Der Aufruf $\mathtt{PREPARE(T_1)}$ sei erfolgreich, also gilt $\mathtt{p(T_1)}$.
Jetzt wird die Anfrage $\mathtt{EOT(T_2)}$ gestellt. Sie könnte – zusammen mit $\mathtt{EOT(T_1)}$ – ausgeführt werden, da anschließend beide Transaktionen im Zustand *committed* sind. $\mathtt{EOT(T_2)}$ kann aber fehlschlagen und $\mathtt{a(T_2)}$ bewirken. T_2 muß also vorher ebenfalls in den Zustand $\mathtt{p(T_2)}$ gebracht werden, bevor beide Commit–Aufrufe abgesetzt werden dürfen. In diesem Fall enden beide Transaktionen erfolgreich.
Wenn andererseits einer der Aufrufe bis einschließlich der `PREPARE`-Operationen fehlschlägt oder eine der Transaktionen von sich aus abbricht, kann die andere Transaktion (wie von der Beziehungssemantik gefordert) vom Zustand *prepared* aus ebenfalls abgebrochen werden.

2.3.3 Prozedurale Semantik der Beziehungen, TRAMPs

Für den elementaren Fall, daß genau eine Beziehung zwischen zwei sonst unabhängigen Transaktionen besteht, können die durchzuführenden Aktionen in sog. *Transaktions-Management-Prozeduren*, kurz *TRAMPs*, notiert werden.

Für jede Kombination aus Zustands-Übergangswunsch, Beziehungstyp und aktuellem Zustand der Transaktionen ist für jede der beiden Transaktionen festgelegt, welche Maßnahmen zu ergreifen sind. Über einen Verteiler (Tabellen 2.2 und 2.3) wird ein Verweis auf die Aktionen ermittelt, die beim Aufruf einer zustandsändernden Operation (Spalte in den Tabellen) vorgenommen werden, wenn die Transaktion in einer Abhängigkeitsbeziehung (Zeile in den Tabellen) steht. In Anhang A sind als Beispiel die Aktionen der ersten Zeile der ersten Tabelle zusammengestellt.

Abh. / Op.	BOT(T_1)	PREP(T_1)	COMMIT(T_1)	ABORT(T_1)
$T_1 \rightarrow_c T_2$	1	4	7	9
$T_1 \rightarrow_a T_2$	äquiv. $T_2 \rightarrow_c T_1$	...	...	...
$T_1 \leftrightarrow_c T_2$	11	13	14	15
$T_1 \leftrightarrow_a T_2$	äquiv. $T_1 \leftrightarrow_c T_2$	...	...	...
$T_1 \;{}_a{\rightarrow}_c\; T_2$	16	17	18	19
$T_1 \;{}_c{\rightarrow}_a\; T_2$	20	21	22	23
$T_1 \;{}_c{\mapsto}_b\; T_2$	24	26	28	30
$T_1 \;{}_a{\mapsto}_b\; T_2$	32	34	36	38
$T_1 \;{}_b{\mapsto}_c\; T_2$	40	42	44	46
$T_1 \;{}_b{\mapsto}_a\; T_2$	48	50	52	54
$T_1 \mapsto_b T_2$	56	58	60	62
$T_1 \leftrightarrow_b T_2$	70	71	72	73

Tabelle 2.2: Verteiler für TRAMPs (T_1)

Abh. / Op.	BOT(T_2)	PREP(T_2)	COMMIT(T_2)	ABORT(T_2)
$T_1 \rightarrow_c T_2$	2	5	8	10
$T_1 \rightarrow_a T_2$	äquiv. $T_2 \rightarrow_c T_1$	...	...	...
$T_1 \leftrightarrow_c T_2$	äquiv. 11	äquiv. 13	äquiv. 14	äquiv 15
$T_1 \leftrightarrow_a T_2$	äquiv. $T_1 \leftrightarrow_c T_2$	...	...	...
$T_1 \;{}_a{\rightarrow}_c\; T_2$	äquiv. 16	äquiv. 17	äquiv. 18	äquiv 19
$T_1 \;{}_c{\rightarrow}_a\; T_2$	äquiv. 20	äquiv. 21	äquiv. 22	äquiv 23
$T_1 \;{}_c{\mapsto}_b\; T_2$	25	27	29	31
$T_1 \;{}_a{\mapsto}_b\; T_2$	33	35	37	39
$T_1 \;{}_b{\mapsto}_c\; T_2$	41	43	45	47
$T_1 \;{}_b{\mapsto}_a\; T_2$	49	51	53	55
$T_1 \mapsto_b T_2$	57	59	61	63
$T_1 \;{}_c{\rightarrow}_a\; T_2$	äquiv. 70	äquiv. 71	äquiv. 72	äquiv 73

Tabelle 2.3: Verteiler für TRAMPs (T_2)

Wenn Zustandsübergänge verzögert werden müssen, weil eine Bedingung nicht erfüllt ist, bestehen zwei Alternativen bei der Implementierung der TRAMPs:

1. Die Abarbeitung der Prozedur wird unterbrochen und ein Ereignis definiert, das eintritt, wenn die Bedingung erfüllt ist. Die Verabeitung wird – getriggert durch das Ereignis – anschließend wieder aufgenommen.

   Ein Beispiel ist eine TRAMP für den `EOT`-Wunsch der Transaktion T_1 bei der Beziehung $T_1 \rightarrow_c T_2$:

   ```
   EOT_with_dependency(T1, T1 -->c T2)
   {
      switch (state(T2)) {
      case b:
         wait_on_event(T2_finished);
         break;
   ```

```
    case p:
        commit(T1);
        if (c(T1)) commit(T2);
        break;
    case a:
        abort(T1);
        break;
    case c:
        commit(T1);
        break;
}
```

2. Die Prozedur vermerkt in einer Zustandskomponente der Transaktion, welchen Zustand diese erreichen will (im Beispiel `demand-commit`), und stoppt. Die TRAMP der Transaktion auf der Gegenseite ist kooperativ und beachtet solche Wünsche. Im Beispiel wird in der Prozedur `EOT_with_dependency(T2, T1 -->c T2)` dann `commit(T1)` aufgerufen, wenn auch `commit(T2)` ausgeführt wird, und `abort(T1)`, wenn T_2 abbricht.

In Anhang A sind zur Illustration die TRAMPs der Beziehung $\rightarrow_c$ aufgeführt, wobei dort die zweite Alternative gewählt wurde.

Im Detail kann über einige Punkte diskutiert werden: Eine Transaktion im Zustand *installed* mit *demand-commit* ist aufgrund des Transaktionsautomaten implizit auch *demand-run*. Wie sehen in dieser Hinsicht die Verhältnisse aus bei einer Transaktion, die *demand-abort* und (z.B. wegen einer weiteren Beziehung) *demand-run* ist? Diese zwei Forderungen sind durchaus gleichzeitig erfüllbar, indem die Transaktion gestartet und anschließend abgebrochen wird. Dieser umständliche Weg sollte erhalten bleiben, um möglichst viele Situationen zum Netz konform zu halten. *demand-run* kann ja beispielsweise von einer Kontrollflußbeziehung stammen, *demand-abort* dagegen z.B. durch eine Leseabhängigkeit aufgetreten sein.

2.3.4 Verknüpfung von Beziehungen

Bisher haben wir die Situation *zwei Transaktionen mit genau einer Beziehung* betrachtet. Transaktionen stehen i.a. aber nicht in einer isolierten Beziehung zu einer weiteren Transaktion, sondern sind in ein Geflecht heterogener Beziehungen zu einer Reihe von ihrerseits voneinander abhängiger Transaktionen eingebunden. Diese Struktur wird *Transaktions–Beziehungs–Netz* (kurz *TBN*) genannt.

Aus automatentheoretischer Sicht ist dies eine konzeptionell einfache Erweiterung: Es wird der Potenzautomat über alle Transaktionen gebildet. Beim Eintragen einer Beziehung zwischen zwei Transaktionen werden die Kanten zwischen Zuständen des Potenzautomaten gelöscht, die (unter Vernachlässigung der anderen Transaktionen im Zustandsnamen) einer gelöschten Kante des Automaten gemäß Kapitel 2.2.2 entspricht. Natürlich werden ebenfalls i.a. rekursiv weitere Zustände und Übergänge gestrichen.

Ein *inkonsistentes TBN* wird dann genauso wie eine inkonsistente Beziehung über den zugehörigen Potenzautomaten definiert: Genau dann, wenn kein Endzustand erreichbar ist, ist das TBN inkonsistent.

In TBNs lassen sich zwei grundlegende Konstellationen von Beziehungen unterscheiden:

1. Zwischen zwei Transaktionen sind mehrere Beziehungen eingetragen.

2. Ein Zustandsübergang einer Transaktion T hat indirekte Auswirkungen auf Transaktionen, die nicht in direkter Beziehung zu T stehen, aber „transitiv“ über Beziehungen zu weiteren Transaktionen „erreichbar“ sind. Potentielle Kandidaten für diese indirekt geforderten Zustandsübergänge sind alle Transaktionen in der „transitiven[2] Hülle“ der Beziehungen.

   Wenn eine Transaktion ihren Zustand ändert, ohne daß zugleich die über indirekte Auswirkungen geforderten Übergänge anderer Transaktionen ebenfalls stattfinden, widerspricht der Zustandsübergang den Beziehungen und ist inkorrekt. Daher ist es elementar wichtig, die indirekten (transitiven) Abhängigkeiten zu kennen.

Um die Auswirkungen derartiger Kombinationen von Beziehungen formal behandeln zu können, werden zwei Arten der Verknüpfung von Beziehungen, eine Addition $\oplus$ und eine Multiplikation $\odot$, definiert.

2.3.4.1 Addition $\oplus$

Die Summe der Beziehungen $(T_1 \;{}_x\!\!\longrightarrow_y T_2) \oplus (T_1 \;{}_v\!\!\longrightarrow_w T_2)$ ist die schwächste Beziehung zwischen beiden Transaktionen, die sowohl die Restriktionen von ${}_x\!\!\longrightarrow_y$, als auch von ${}_v\!\!\longrightarrow_w$ enthält. Im kombinierten Transitionsdiagramm des Potenzautomaten sind die von jeweils einer Beziehung gelöschten Knoten und Kanten nicht vorhanden[3].

Die Ergebnisse der Addition von jeweils zwei der elementaren Beziehungen sind in Anhang C aufgeführt.

Nicht immer entsteht ein Automat, der einer elementaren Beziehung entspricht. Der Automat kann auch bei Transaktionen, die nur im Zustand *installed* sind, bereits so stark eingeschränkt sein, daß für eine oder sogar beide Transaktionen nur noch jeweils ein Endzustand erreichbar ist.

Zum Beispiel müssen bei $(T_1 \leftrightarrow_c T_2) \oplus (T_1 \,{}_b\!\!\longmapsto_a T_2)$ beide Transaktionen abbrechen, ohne daß dies (wie üblich) durch einen Transaktions-Zustandsübergang ausgelöst worden wäre. Zu diesem Ergebnis kommt man durch folgende Überlegungen: Nehmen wir an, T_1 sei im Zustand *committed*. Aus der Beziehung $(T_1 \leftrightarrow_c T_2)$ folgt, daß auch `c(T₂)` gilt. Das erfolgreiche Beenden von T_1 impliziert aber auch den Start von T_1, was wegen $(T_1 \,{}_b\!\!\longmapsto_a T_2)$ zum Abbruch von T_2 und aufgrund der beidseitigen Commit-Abhängigkeit zum Abbruch von T_1 führt: ein Widerspruch zur Annahme. Brechen beide Transaktionen ab, treten keine Probleme auf.

Die Situation im Beispiel scheint den impliziten Annahmen zu widersprechen, daß jede Transaktion den Zustand *committed* erreichen können sollte, diese Möglichkeit hier aber von vornherein ausgeschlossen ist. Es ist trotzdem sinnvoll, beide Beziehungen gleichzeitig zuzulassen, da diese Konstellation sicher nicht typisch für die eingetragenen Beziehungen ist und immerhin korrekt verwaltet werden kann. Beispielsweise kommen gerade dann, wenn das Transaktionsmodell Kompensationstransaktionen vorsieht, Widersprüche zu den impliziten Annahmen aus Kapitel 2.2 vor: Kompensationstransaktionen werden installiert und es werden Beziehungen zu ihnen eingetragen, die bewirken, das diese Transaktionen bei der korrekten Abwicklung der Anwendung eben *nicht* gestartet, geschweige denn erfolgreich beendet werden.

2.3.4.2 Multiplikation $\odot$

Durch die Multiplikation werden indirekte (transitive) Auswirkungen von Beziehungen bei verketteten Transaktionen ermittelt. Das Produkt $(T_1 \rightarrow_x T_2) \odot (T_2 \rightarrow_y T_3)$ ist die Beziehung zwischen

[2] In Anführungsstrichen deshalb, weil diese Hülle über verschiedenartige Kanten im Graphen gebildet wird.

[3] Die Addition von Beziehungen führt zu einer „Subtraktion“ im Automaten

T_1 und T_3, die sich allein aus der Existenz der Beziehungen $T_1 \rightarrow_x T_2$ und $T_2 \rightarrow_y T_3$ ergibt: Zustandsübergänge von T_1 haben Auswirkungen auf Zustandsübergänge von T_3 und umgekehrt.

Sollte bereits zwischen T_1 und T_3 eine Beziehung bestehen, muß das Produkt zu ihr addiert werden, um die resultierende (Gesamt–)Beziehung[4] zu erhalten.

Bei Transaktionen T_1 bis T_3 mit Zustand *installed* (und solange keine zusätzlichen Beziehungen ins Spiel kommen) können, im Gegensatz zur Addition, bei der Multiplikation von zwei elementaren Beziehungen alle drei Transaktionen alle Endzustände erreichen – nur eben nicht in beliebiger Kombination.

Eine indirekte Wirkung ergibt sich dann, wenn eine Zustandsänderung von T_1 einen Zustandsübergang von T_2 fordert, und dieser wiederum einen von T_3.

Die Ergebnisse der Multiplikation der elementaren Beziehungen sind in Anhang D aufgeführt, woraus einige interessante Ergebnisse in Kapitel 2.3.4.3 zusammengestellt sind. Die Probleme, die sich durch indirekte Wirkungen ergeben, werden in Kapitel 2.6 erörtert.

2.3.4.3 Eigenschaften der Multiplikation von Beziehungen

„Transitive“ Kopplung: In Tabelle 2.4 sind diejenigen Beziehungen aufgeführt, für die gilt[5]: Aus $T_1 \;{}_x{\longrightarrow}_y\; T_2$ und $T_2 \;{}_v{\longrightarrow}_w\; T_3$ folgt $T_1 \;{}_x{\longrightarrow}_w\; T_3$. Als Einschränkung muß beachtet werden, daß sich bei Kontrollflußbeziehungen die „Sofortausführung“ unabhängig davon fortpflanzt bzw. unterbrochen wird. Beispiel: Bei dem TBN $T_1 \;{}_c{\longrightarrow}_a\; T_2 \;{}_a{\longmapsto}_b\; T_3$ und COMMIT(T_1) muß *nicht sofort* BEGIN(T_3) ausgeführt werden, da ${}_c{\longrightarrow}_a$ eine logische Beziehung ist, und daher bereits der Abbruch von T_2 nur „irgendwann“ stattfinden muß. Bei $T_1 \;{}_c{\longmapsto}_b\; T_2 \longmapsto_b T_3$ liegt der Fall anders: COMMIT(T_1) fordert sofort `b(T₂)` und dies wiederum sofort BEGIN(T_3).

$T_1 \;{}_x{\longrightarrow}_y\; T_2 \;{}_v{\longrightarrow}_w\; T_3$	$\Longrightarrow$	$T_1 \;{}_x{\longrightarrow}_w\; T_3$
$T_1 \;{}_x{\longrightarrow}_c\; T_2 \;{}_c{\longrightarrow}_w\; T_3$		$T_1 \;{}_x{\longrightarrow}_w\; T_3$
$T_1 \;{}_x{\longrightarrow}_c\; T_2 \;{}_b{\longrightarrow}_w\; T_3$		$T_1 \;{}_x{\longrightarrow}_w\; T_3$
$T_1 \;{}_x{\longrightarrow}_b\; T_2 \;{}_b{\longrightarrow}_w\; T_3$		$T_1 \;{}_x{\longrightarrow}_w\; T_3$
$T_1 \;{}_x{\longrightarrow}_a\; T_2 \;{}_a{\longrightarrow}_w\; T_3$		$T_1 \;{}_x{\longrightarrow}_w\; T_3$
$T_1 \;{}_x{\longrightarrow}_c\; T_2 \leftrightarrow_c T_3$		$T_1 \;{}_x{\longrightarrow}_c\; T_3$
$T_1 \leftrightarrow_c T_2 \;{}_c{\longrightarrow}_w\; T_3$		$T_1 \;{}_c{\longrightarrow}_w\; T_3$
$T_1 \leftrightarrow_c T_2 \leftrightarrow_c T_3$		$T_1 \leftrightarrow_c T_3$
$T_1 \;{}_x{\longrightarrow}_c\; T_2 \leftrightarrow_b T_3$		$T_1 \;{}_x{\longmapsto}_b\; T_3$
$T_1 \;{}_x{\longmapsto}_b\; T_2 \leftrightarrow_b T_3$		$T_1 \;{}_x{\longmapsto}_b\; T_3$
$T_1 \leftrightarrow_c T_2 \;{}_a{\longrightarrow}_w\; T_3$		$T_1 \;{}_a{\longrightarrow}_w\; T_3$[6]
$T_1 \;{}_x{\longrightarrow}_a\; T_2 \;{}_c{\longleftarrow}_a\; T_3$		$T_1 \;{}_x{\longrightarrow}_c\; T_3$
$T_1 \;{}_x{\longrightarrow}_a\; T_2 \leftrightarrow_c T_3$		$T_1 \;{}_x{\longrightarrow}_a\; T_3$
$T_1 \;{}_x{\longrightarrow}_a\; T_2 \;{}_c{\longleftarrow}\; T_3$		$T_1 \;{}_x{\longrightarrow}_a\; T_3$
$T_1 \;{}_x{\longrightarrow}_c\; T_2 \;{}_a{\longleftarrow}_c\; T_3$		$T_1 \;{}_x{\longrightarrow}_a\; T_3$

Tabelle 2.4: Transitive Auswirkungen

Transitivität: Als Spezialfälle der transitiven Kopplung möchte ich die „echt“ transitiven Beziehungen herausfiltern: Die Beziehungen $\longrightarrow_c$, $\leftrightarrow_c$, ${}_b{\longmapsto}_c$, $\longmapsto_b$ und $\leftrightarrow_b$ sind transitiv, d.h. aus $T_1 \;{}_x{\longrightarrow}_y\; T_2$ und $T_2 \;{}_x{\longrightarrow}_y\; T_3$ folgt $T_1 \;{}_x{\longrightarrow}_y\; T_3$.

[4]Dieses Resultat wird auch als *aggregierte Beziehung* bezeichnet.

[5]x, y, v, w sind Variable, für die a, b oder c eingesetzt werden kann.

[6]Diese (und alle weiteren, hier nicht aufgeführten) Einträge ergeben sich aufgrund der Identität von $\leftrightarrow_c$ und $\leftrightarrow_a$ bzw. der Umbenennungsregel.

Auslöschung: In der Situation $T_1 \;{}_{x}{\rightarrow}_{y}\; T_2 \;{}_{v}{\rightarrow}_{w}\; T_3$ (drei Transaktionen in Reihe miteinander verknüpft) implizieren ausschließlich die Instantiierungen aus Tabelle 2.5 für ${}_{x}{\rightarrow}_{y}$ und ${}_{v}{\rightarrow}_{w}$, daß zwischen T_1 und T_3 keine (indirekte) Abhängigkeit besteht[7].

T_1	${}_{x}{\rightarrow}_{y}$	T_2	${}_{v}{\rightarrow}_{w}$	T_3
T_1	${}_{*}{\rightarrow}_{c}$	T_2	${}_{a}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${}_{c}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${}_{b}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${\leftrightarrow}_{b}$	T_3
T_1	${}_{*}{\mapsto}_{b}$	T_2	${}_{c}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\mapsto}_{b}$	T_2	${\leftrightarrow}_{c}$	T_3
T_1	${}_{*}{\mapsto}_{b}$	T_2	${}_{a}{\rightarrow}_{*}$	T_3
T_1	${\leftrightarrow}_{b}$	T_2	${}_{c}{\rightarrow}_{*}$	T_3

T_1	${}_{x}{\rightarrow}_{y}$	T_2	${}_{v}{\rightarrow}_{w}$	T_3
T_1	${}_{*}{\rightarrow}_{c}$	T_2	${}_{c}{\leftarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{c}$	T_2	${}_{b}{\leftarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${}_{a}{\leftarrow}_{*}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${}_{b}{\leftarrow}_{*}$	T_3
T_1	${}_{*}{\mapsto}_{b}$	T_2	${}_{b}{\leftarrow}_{*}$	T_3
T_1	${}_{*}{\leftarrow}_{c}$	T_2	${}_{c}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\leftarrow}_{c}$	T_2	${}_{b}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\leftarrow}_{a}$	T_2	${}_{a}{\rightarrow}_{*}$	T_3
T_1	${}_{*}{\leftarrow}_{b}$	T_2	${}_{b}{\rightarrow}_{*}$	T_3

Tabelle 2.5: Auslöschung der indirekten Wirkung

Weitere Verallgemeinerungen in dieser Tabelle sind nicht möglich, denn ausdrücklich *nicht* zur „Auslöschung“ der indirekten (transitiven) Wirkung führen unter anderen die Kombinationen der Beziehungen (mit den jeweiligen symmetrischen Pendants) aus Tabelle 2.6.

T_1	${}_{x}{\rightarrow}_{y}$	T_2	${}_{v}{\rightarrow}_{w}$	T_3
T_1	${}_{*}{\rightarrow}_{c}$	T_2	${\leftrightarrow}_{b}$	T_3
T_1	${\leftrightarrow}_{c}$	T_2	${}_{a}{\rightarrow}_{*}$	T_3
T_1	${\leftrightarrow}_{c}$	T_2	${\leftrightarrow}_{b}$	T_3
T_1	${}_{*}{\rightarrow}_{a}$	T_2	${\leftrightarrow}_{c}$	T_3
T_1	${\leftrightarrow}_{b}$	T_2	${\leftrightarrow}_{c}$	T_3
T_1	${\leftrightarrow}_{c}$	T_2	${\rightarrow}_{c}$	T_3
T_1	${\leftrightarrow}_{c}$	T_2	${\mapsto}_{b}$	T_3
T_1	${\leftrightarrow}_{b}$	T_2	${\mapsto}_{b}$	T_3

Tabelle 2.6: Keine Auslöschung der indirekten Wirkung

2.4 Inkonsistente Konstellationen

Wodurch ist ein inkonsistentes TBN charakterisiert? Ein TBN ist *inkonsistent*, wenn mindestens eine Transaktion aus diesem Netz unter Berücksichtigung

- der Beziehungen
- der Zustände der Transaktionen

keinen Endzustand erreichen kann.

[7] $*$ steht als „Wildcard“ für c, a oder b.

Diese Definition ist unnötig umfassend. Sinnvoll ist es, sie nicht auf das gesamte TBN zu beziehen, sondern auf einen einzelnen zusammenhängenden Teilgraphen im TBN. Ein TBN besteht also aus mehreren Teilnetzen, die jeweils unabhängig konsistent oder inkonsistent sind.

Wenn wir als *aggregierte Beziehung* die Beziehung zwischen zwei Transaktionen bezeichnen, die durch Summation der in das TBN eingetragenen Beziehungen zwischen diesen TAs mit allen indirekten Beziehungen zwischen ihnen entsteht, können wir für ein *inkonsistentes TBN-Teilnetz* eine weitere Definition geben:

Ein Teilnetz ist dann *inkonsistent*, wenn es eine inkonsistente aggregierte Beziehung enthält.

Eine Beziehung ist inkonsistent, wenn sie widersprüchliche Zustände einer Transaktion fordert. Diese Tatsache drückt sich darin aus, daß beim Aufbau des diese Beziehung repräsentierenden Zustandsautomaten kein Endzustand erreichbar ist.

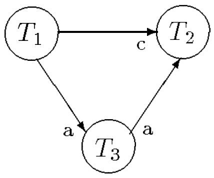

Abbildung 2.3: Beispiel-TBN

Zur Erläuterung reiche ich noch ein Beispiel für eine aggregierte Beziehung nach: Im TBN aus Abbildung 2.3 mit der zwischen T_1 und T_2 eingetragenen Beziehung $(T_1 \rightarrow_c T_2)$ und der indirekten Beziehung $(T_1 \rightarrow_a T_2)$, die durch die Multiplikation $(T_1 \rightarrow_a T_3) \odot (T_3 \rightarrow_a T_2)$ der zu T_3 bestehenden Beziehungen entsteht (und die bekanntermaßen identisch zu $(T_1 \,{}_c\!\leftarrow T_2)$ ist), besteht zwischen T_1 und T_2 die aggregierte Beziehung $(T_1 \leftrightarrow_c T_2)$.

2.5 Nichtdeterminismus

Naheliegenderweise will man die Konsequenzen eines Zustandsüberganges deterministisch bestimmen, um eine eindeutige Handlungsanweisung zu erhalten und keine Ressourcen zur Berechnung irrelevanter Mengen von Zustandsübergängen zu „verschwenden". Wenn man jedoch schrittweise die Konsequenzen realisiert, indem man das Netz durchläuft, kann es passieren, daß man Folgezustandsänderungen vollzieht, von denen man später feststellt, daß sie zu Inkonsistenzen führen. Diese hätte man also besser nicht durchführen sollen.

Andererseits ist eine vollständige Berechnung „im voraus" schwierig, denn es gibt, wie man an der Beispiel-TRAMP auf Seite 16 erkennt, bedingte Aktionen als Konsequenz eines Zustandsüberganges. Wenn rekursiv sämtliche Auswirkungen eines Zustandüberganges berechnet werden sollen, ohne die Aktionen auszuführen, liegt Nichtdeterminismus vor. Eine solche Berechnung „im voraus" ist beispielsweise nötig, um Widersprüche (Inkonsistenzen) in den geforderten Zustandsübergängen rechtzeitig feststellen zu können.

Beispiel: T_1 aus Abb. 2.4 sei *running*. T_3 sei bereits im Zustand *prepared*, nachdem `EOT(T`$_3$`)` verzögert werden mußte, da T_1 noch aktiv ist. Seitdem T_1 gestartet wurde, besteht über die obere Kette von Beziehungen die Anforderung an T_4, den Zustand `c(T`$_4$`)` zu erreichen (und es gilt $abrt(T_2)$, was uns hier aber primär nicht interessiert). In dieser Situation meldet T_1 den Wunsch `EOT(T`$_1$`)` an den

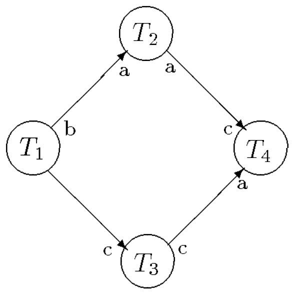

Abbildung 2.4: Beispiel-TBN

TBNM. Führt dieser von links nach rechts die entsprechenden TRAMPs der unteren Beziehungen aus, so wird zuerst gemäß dem Pseudocode auf Seite 16 T_1 beendet — wir nehmen an, dies war erfolgreich, also `c(T`$_1$`)` — und anschließend T_3. Wegen `p(T`$_3$`)` resultiert dieser Schritt garantiert in `c(T`$_3$`)`. Nun fordert die Prozedur von $_c\!\!\longrightarrow_a$ für den Fall `c(T`$_3$`)` den Abbruch von T_4, womit für T_4 widersprüchliche Forderungen bestehen. Die Erkenntnis, daß T_1 hätte abgebrochen werden müssen, um die Konsistenz des TBN zu erhalten, kommt leider zu spät.

Welchen Zusammenhang soll der TBNM herstellen zwischen dem Berechnen der Konsequenzen einer Operation und der Durchführung der zugehörigen Zustandsübergänge? Es stehen drei Möglichkeiten zur Auswahl:

1. Er führt Zustandsübergänge direkt selbst sofort durch. Vorteilhaft ist, daß er direkt die Auswirkungen der zustandsändernden Operationen kennt. Dieser Ansatz hat allerdings einige nachteilige Konsequenzen:
   - Er muß selber mit mehreren verteilten *resource managers* (z.B. SQL-Server) kommunizieren und auf asynchrone Nachrichten warten. Dies steht in einem Spannungsverhältnis zu der Anforderung, daß die Zustandsänderungen, die eine Anforderung auslöst, atomar ablaufen sollten.
   - Andere Systemkomponenten können keinen Einfluß auf den Gang der Berechnung nehmen. Insbesondere muß der TBNM die globale Sicht auf alle Beziehungen im System haben, da er sonst u.U. inkonsistente Aktionen startet.
   - Die langandauernde Berechnung und Ausführung indirekt gefordeter Zustandsänderungen muß wegen der Seiteneffekte auf Transaktionen atomar (unter „Transaktionsschutz“) ablaufen.
2. Die Berechnung der Auswirkungen wird von der Ausführung entkoppelt:
   (a) Zustand (lokal) anschauen, daraufhin
   (b) entscheiden, welche Operation durchgeführt werden soll,
   (c) den gewünschten Zielzustand dem TM melden, der die Operation durchführt und
   (d) dem TBN–Manager wiederum meldet, welcher Zustand erreicht wurde.
   (e) Dieser beginnt rekursiv mit Berechnungen der Auswirkungen der neuen Situation. Diese Rekursion und Iteration der Kommunikation ist notwendig, da z.B. die Operation `EOT` zu verschiedenen Endzuständen führen kann, und die weitere Berechnung von diesem Ergebnis abhängt.

Gegenüber dem ersten Ansatz hat der zweite einige Vorteile:

- Man kann auf einer höheren Ebene (TM) die Konsequenzen interpretieren und bewerten, bevor man sie realisiert, da der Transaktionsmanager Eingriffsmöglichkeiten hat.
- Der TM kann zwischen verteilten TBNM vermitteln.
- Das Logging der Nachrichten zwischen TM und TBNM reicht (aus Sicht des TBNM) für eine gegen Systemfehler unempfindliche Verarbeitung.

Nachteilig ist, daß immer noch viel Kommunikation auftritt.

3. Der TBN-Manager führt selbst gar keine Zustandsübergänge durch, sondern simuliert diese im TBN, solange noch nicht alle indirekten Auswirkungen berechnet wurden. Dabei notiert er die hypothetischen Zustandsübergänge in einer Liste, der sogenannten *Agenda*. Erst in einem zweiten Schritt meldet er dem TM die Agenda mit den gewünschten Zielzuständen. Dieser schickt nur bei Mißerfolg wiederum dem TBNM eine Nachricht, um die Konsequenzen der neuen Situation berechnen zu lassen.

   Der dritte Ansatz besitzt die Vorteile des zweiten. Außerdem benötigt er insbesondere bei erfolgreicher Durchführung weniger Interaktion.

2.5.1 Dynamik

Wenn nichtdeterministisch, d.h. unter Berücksichtigung aller möglichen Ausgänge der zustandsändernden Operationen, die Konsequenzen der initiierenden Operation (ein Zustandsübergang $\mathtt{X(T}_y\mathtt{)}$ oder das Eintragen einer Beziehung ins TBN) berechnet werden, um die indirekt geforderten Zustandsübergänge zu berechnen bzw. Inkonsistenzen festzustellen, bedeutet das letztlich, daß mehrere Wege zu Endzuständen im Potenzautomaten der Transaktionen gesucht werden. Dieses Vorgehen kann optimiert werden, da ein gefundener Weg (möglichst der, der letztlich beschritten wird) ausreicht.

Als Strategie bietet sich an, jeweils optimistisch vom Eintreffen der gewünschten Übergänge auszugehen (gemäß den impliziten Annahmen beispielsweise *committed* bei `EOT(T)`) und jeweils den Pfad, der der Ablehnung entspricht (im Beispiel *aborted*), nicht zu betrachten. Nur wenn bei dem Beschreiten dieses Pfades von ihm abgewichen wird, wird eine Alternative berechnet. Wenn es noch eine gibt, wird ihr gefolgt, sonst ist man in eine „Falle“ gelaufen: Das TBN ist inkonsistent geworden.

Zustandsänderungen auf dem gesamten TBN atomar durchzuführen ist nicht möglich, da bei der Abarbeitung der zustandsändernden Operationen der resultierende Transaktionszustand von Transaktionen T nicht eindeutig festliegt. Insbesondere kann auf `BOT(T)` `c(T)` folgen, aber auch `a(T)`. Zur Lösung des Problems teilt man die Arbeit in mehrere sequentielle Schritte auf, die jeweils atomar durchgeführt werden können und keine derartigen Fallunterscheidungen enthalten. Wird dabei mindestens ein gewünschter Zielzustand nicht erreicht, konnte offensichtlich die initiierende Operation nicht konsistent ausgeführt werden, und es wird — falls möglich, denn es gibt z.B. zu `ABORT(T)` bekanntlich keine Alternative — die oben erwähnte Alternative versucht. Ein wiederholtes „Backtracking“ wird nicht durchgeführt. Schlägt die Alternative nämlich ebenfalls fehl, ist die initiierende Operation inkonsistent.

Dieses Verfahren kann zu an sich vermeidbaren Inkonsistenzen führen, falls Zustandsübergänge, die vor Beginn der Alternativberechnung durchgeführt wurden, zu den Konsequenzen in der Alternative Widersprüche enthalten. Deshalb müssen sämtliche geforderten `EOT`–Aufrufe in einer atomaren Operation und einheitlichem Ergebnis durchgeführt werden können, was mit `PREPARE`-Aufrufen möglich wird. Dieses Thema wird in [Lutz] behandelt.

2.6 Problematik der Verwaltung von TAs mit Beziehungen

2.6.1 Randbedingungen

Um die bisherigen Ergebnisse zusammenzufassen, stellen wir fest, daß die TBN-Verwaltung unter folgenden Randbedingungen operiert:

- Eine „gewünschte“ Operation kann dazu führen, daß
  - dieser Zustandsübergang stattfindet.
  - dieser Zustandsübergang verworfen und dafür ein anderer durchgeführt wird (z.B. auf die Anfrage `EOT(T)` wird `abort(T)` aufgerufen).
  - auf anderen Transaktionen weitere Operationen ausgeführt werden.
  - ein Problem entsteht, wenn die Operation inkonsistent zu den Beziehungen im Netz ist, aber auch die „komplementäre Operation“[8] nicht erlaubt ist.
- Beziehungen können sowohl statisch festliegen, als auch dynamisch zur Laufzeit entstehen und wieder verschwinden.

  Beispiel 1: Die Leseabhängigkeit zu einer schreibenden Transaktion verschwindet, sobald diese im Zustand *committed* ist.

  Beispiel 2: Im Verlauf der Interpretation eines ConTracts wird ein neuer Step, der mit drei Transaktionen $T_{n1}, \ldots, T_{n3}$ realisiert ist, als Nachfolger eines alten Steps mit Transaktion T_a spezifiziert. An Beziehungen wird eingetragen, während T_a noch aktiv ist: Die T_{ni} starten, wenn T_a erfolgreich sein wird ($T_a \;{}_c\!\!\mapsto_b T_{ni}$); Entweder brechen dann die T_{ni} alle ab, oder sie werden alle zusammen den Zustand *committed* erreichen ($T_{n1} \leftrightarrow_c T_{n2} \leftrightarrow_c T_{n3}$).
- Die in die Beziehungsverwaltung umgesetzten Transaktionsmodelle decken nicht alle Fälle ab; sie sind unvollständig bzw. nicht abgeschlossen: Es kann unvermeidbare Inkonsistenzen geben.

  Bsp: In einer Saga bricht eine Transaktion ab. Die Kompensationstransaktion bricht ebenfalls ab. Dieser Fall ist im Saga-Modell nicht vorgesehen und in dessen Umsetzung in das TBN-Modell somit ebenfalls nicht ohne weiteres zu behandeln: Ohne spezielle Maßnahmen entsteht ein inkonsistentes TBN.

  Aus einem inkonsistenten TBN, d.h. einem logischen Widerspruch, folgen beliebige und ebenfalls widersprüchliche Konsequenzen. Daher ist es notwendig, pragmatisch einzugreifen und sich über das TBN hinwegsetzen zu können. Das bedeutet konkret, daß der TM nicht in allen Fällen die von der TBN–Verwaltung geforderten Operationen durchführt, sondern andere Maßnahmen ergreifen muß.
- Inkonsistenzen des TBN, d.h. widersprüchliche Zustandsforderungen, müssen *vor* dem Eintragen einer „gefährlichen“ Beziehung bzw. vor der Ausführung des betreffenden Zustandsüberganges entdeckt werden, um die Operation verhindern und das Netz konsistent halten zu können.

2.6.2 Auswirkungen der Zustandsänderung einer TA auf andere

Zustandsänderungen pflanzen sich nicht nur auf die im TBN direkt benachbarten Transaktionen fort. Sie haben möglicherweise Auswirkungen auf alle über weitere Beziehungen von T aus indirekt erreichbaren Transaktionen.

Über Ketten von Beziehungen aus Tabelle 2.4 entstehen indirekte Beziehungen zu entfernten Transaktionen, die wiederum Zustandsforderungen nach sich ziehen können.

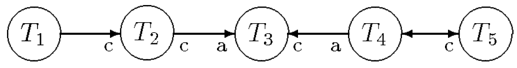

Abbildung 2.5: Beispiel-TBN: direkte Beziehungen

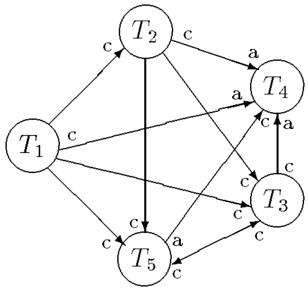

Abbildung 2.6: Indirekte, aggregierte Beziehungen

Beispiel: Im TBN aus Bild 2.5 mit 5 Transaktionen sind insgesamt die indirekten Beziehungen aus Abb. 2.6 „versteckt". Die Summe sämtlicher indirekter Beziehungen zwischen zwei Transaktionen und der zwischen ihnen ursprünglich eingetrangenen Abhängigkeiten bezeichnet man als die *aggregierte Beziehung* zwischen den beiden Transaktionen.

Über Zyklen kann eine Transaktion auf sich selber einwirken. Beispiel: Sobald T_1 aus Abb. 2.7 startet, darf T_1 nicht anders, als erfolgreich sein; das TBN fordert `c(T`$_1$`)`. Dieses TBN ist erlaubt (konsistent), da es (noch) eine Kombination von Endzuständen für alle Transaktionen erreichbar ist, in der die Beziehungen erfüllt sind (siehe aber Kapitel 2.6.3.1).

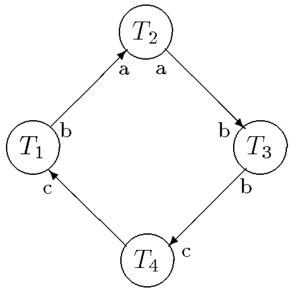

Abbildung 2.7: Ein zyklisches TBN

Was ist mit einer Transaktion, die direkt zu sich selbst in Beziehung steht? Dies ist nur bei bestimmten Beziehungstypen sinnvoll; bei einigen hat die reflexive Anwendung keine Auswirkungen (z.B.

[8] `ABORT` ist komplementär zu `COMMIT`.

$(T \rightarrow_c T)$), bei anderen ist die Situation inkonsistent (z.B. $(T\ {}_a\!\!\mapsto_b T)$: bei unserem zyklenfreien Zustandsgraphen ist das unmöglich). Weitere Beziehungen wie $(T\ {}_a\!\!\rightarrow_c T)$ erlauben es dagegen, mit den Sprachmitteln der Beziehungen *demands* für Transaktionen festzuschreiben: der Abbruch von T ist hier beispielsweise inkonsistent. T muß unabhängig von anderen Transaktionen `c(T)` erreichen.

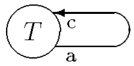

Abbildung 2.8: Eine reflexive Beziehung

Eine Zustandsänderung einer Transaktion T kann somit nicht nur durch den Aufruf einer Operation auf T (z.B. `BOT(T)`) ausgelöst werden, sondern auch durch

- indirekt angestoßene Zustandsübergänge, aber auch allein durch
- das Eintragen einer neuen Beziehung. Als Beispiel soll neben der „alten“ Beziehung $\leftrightarrow_c$ zwischen T und einer anderen Transaktion die Beziehung ${}_b\!\!\mapsto_a$ eingetragen werden (siehe Seite 18).

2.6.3 Atomares Commit mehrerer Transaktionen

Beziehungsnetze, in denen ein Übergang nach `c(T)` einer Transaktion T genau dann erlaubt ist, wenn eine oder mehrere weitere Transaktionen T_i den Zustand *committed* erreichen, werfen für die Transaktionsverwaltung ein Problem auf: T und die Transaktionen T_i müssen in einer atomaren Operation den Zustandsübergang nach `c(T)` bzw. `c(T`$_i$`)` durchführen können, wobei garantiert sein muß, daß entweder alle oder keine dieser Transaktionen den Zustand *committed* erreichen.

Diese Garantie gibt einem das Zwei-Phasen-Commit-Protokoll. Genaueres zu diesem Thema findet man in [Lutz].

Derartige Konstellationen sind nicht theoretischer Natur. Beispielsweise besteht zwischen den Transaktionen eines Steps im ConTract-Modell genau dieser Beziehungstyp (siehe Abb. 2.9).

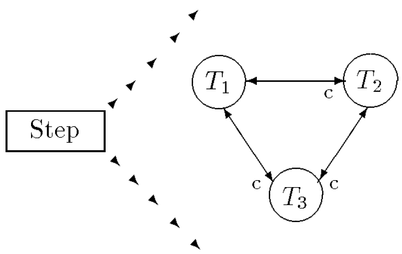

Abbildung 2.9: Transaktionen eines Steps

2.6.3.1 „Gefährliche“ Beziehungstypen

Eine Klasse von sog. *gefährlichen Beziehungen* hat die unangenehme Eigenschaft, daß sie in ein TBN eingetragen werden können, ohne notwendigerweise sofort zu Inkonsistenzen zu führen, aber so starke Restriktionen implizieren, daß nicht in allen Situationen korrekte Zustandsübergänge durchgeführt werden können. Um konkret zu werden: Es handelt sich um Beziehungen, die unter der Voraussetzung eines nicht zu verhindernden Zustandsüberganges (`abort` und `demand-not-run`) das Commit oder den Start[9] einer anderen Transaktion fordern. Dies sind die Beziehungen $_a\!\!\rightarrow_c$ und $_a\!\!\mapsto_b$.

Zusätzlich können auch an sich „harmlose“ Beziehungen gefährlich werden, wenn sie in ein TBN eingetragen werden, in dem manche Transaktionen nicht mehr im Zustand *installed* sind. Beispiel: Es gilt `c(`T_1`)`, `i(`T_2`)`, und $(T_1 \rightarrow_c T_2)$ wird eingetragen.

„Gefährlich“ ist an diesen Konstellationen, daß weder das erfolgreiche Ende einer Transaktion noch – in etwas schwächerer Form – der Start erzwungen werden können.

2.6.3.2 Entdeckung und Vermeidbarkeit inkonsistenter TBN

Inkonsistente TBN (-Cliquen) zu *entdecken*, ist nicht schwierig, wenn man die aggregierten Beziehungen zwischen allen Paaren von Transaktionen berechnet hat: Man untersucht, ob diese Beziehungen mit den konkreten, aktuellen Transaktionszuständen verträglich sind, um Inkonsistenzen zu entdecken.

Um inkonsistente TBN zu *vermeiden*, kann der TBNM vor jedem Eintragen einer weiteren Beziehung probeweise diese Operation durchführen und den obigen Test auf Inkonsistenzen vornehmen. Heißt das Ergebnis „inkonsistent“, sollte der TM die Operation ablehnen.

Ein Problem stellen die gefährlichen Beziehungstypen dar. Verbieten wir ihre Verwendung und verhindern wir den gefährlichen Gebrauch harmloser Beziehungen, schränken wir die Mächtigkeit des Modells extrem ein. Beispielsweise ließe sich das Saga-Modell nicht mehr korrekt darstellen.

Behalten wir sie andererseits bei, lassen wir zu, daß das TBN zu einem beliebigen Zeitpunkt bei irgendeinem Zustandswechsel in einen inkonsistenten Zustand geraten kann, der sich nicht mehr vermeiden läßt.

Wie geht der TBNM mit inkonsistenten Netzen um? Die spontane Idee ist: *Dann bricht er eben alle Transaktionen aus dieser Clique ab. Das ist zwar nicht elegant, aber das Problem ist aus der Welt.* Leider geht es so nicht! „Inkonsistent“ bedeutet ja nicht nur, daß Transaktionen nicht mehr erfolgreich sein dürfen, sondern auch, daß ihnen zusätzlich sogar der Abbruch verwehrt ist. Der TBNM ist „handlungsunfähig“ geworden, was eine inkonsistente Clique angeht.

Das Problem kann nur eine Ebene höher unter pragmatischen Gesichtspunkten gelöst werden: hier ist es vielleicht sinnvoll, z.B. einen ganzen ConTract abzubrechen, alle Beziehungen, die er aufgebaut hatte, zu entfernen und die Transaktionen doch abzubrechen. Solche Entscheidungen kann der TBNM allerdings nicht treffen. Eine ausführliche Diskussion dieser Aspekte würde den Rahmen dieser Arbeit sprengen (siehe [Lutz]).

2.6.4 Entfernen von Informationen aus dem TBN

Wann dürfen Beziehungen gelöscht werden? Anders formuliert: Unter welchen Voraussetzungen darf das TBN Beziehungen und Transaktionen vergessen?

[9] Der Start nur unter der logischen Sicht der Kontrollflußbeziehungen.

Hier soll nicht der einfache Fall betrachtet werden, wo das Löschen von Beziehungen zur Realisierung des TA–Modells dient. Ebenfalls außer Diskussion steht der pragmatische Ansatz, bei inkonsistenten Netzen zur Abhilfe Beziehungen zu entfernen, die zu der Inkonsistenz führen.

Korrekt, aber unrealistisch in einem realen System wäre es, nie etwas zu vergessen.

Der frühestmögliche Zeitpunkt hängt von mehreren Faktoren ab. Im einzelnen sind dies die Zustände der Transaktionen im Netz. Wesentlich ist aber auch die Antwort auf folgende Frage:

Wann dürfen Beziehungen in das TBN eingetragen werden?

Es ist ausdrücklich gefordert, daß Beziehungen dynamisch *zur Laufzeit* der Transaktionen in das TBN ein– und ausgetragen werden können.

Auf den ersten Blick erscheint es widersinnig, das Eintragen von Beziehungen sogar zwischen *beendeten* Transaktionen (Transaktionen in Endzuständen) zuzulassen. Dies muß aus mehreren Gründen dennoch möglich sein:

1. Durch die Dynamik in einem verteilten System (insbesondere wenn nach einem optimistischen Protokoll vorgegangen wird) kann die Reihenfolge von Aktionen „durcheinandergeraten“: Die Nachricht von einem entfernten Rechnerknoten an den TM, daß eine Beziehung eingetragen werden soll, kann von der Anfrage „überholt“ worden sein, daß eine Transaktion zu beenden ist.
2. Speziell die logischen Beziehungen sagen nichts über die zeitlichen Verhältnisse aus. Es kann durchaus sinnvoll sein, die Konsistenz der Verhältnisse „post mortem“ zu beurteilen.
3. Auch bereits beendete Transaktionen können mit *nicht* beendeten Transaktionen in Abhängigkeiten stehen. Die neue Beziehung hat i.a. durchaus Einfluß auf diese installierten oder aktiven Transaktionen.

Da also über den Zeitpunkt, wann weitere Beziehungen eingetragen werden können, keine Einschränkungen vorliegen, und andererseits Beziehungen indirekte Auswirkungen haben können, dürfen *Beziehungen* frühestens dann gelöscht werden, wenn sie keine Implikationen auf die möglichen Zustandsübergänge direkt oder indirekt betroffener Transaktionen mehr haben.

Eine Transaktion T darf dann gelöscht werden, wenn

1. keine aktuellen oder zukünftigen Beziehungen Einfluß darauf haben, welche Zustandsübergänge für diese Transaktion T erlaubt sind.
2. weder der aktuelle noch irgendein zukünftiger Zustand von T über jetzige oder spätere Beziehungen Einschränkungen auf andere Transaktionen ausübt.

Beziehungen und Transaktionen dürfen also aus dem TBN entfernt werden, sobald

1. alle direkt oder indirekt über Beziehungen verbundene Transaktionen (Transaktionen in der „transitiven Hülle“ der Beziehungen) ihren Endzustand erreicht haben, und
2. keine weitere Beziehung mehr zu einer dieser Transaktionen eingetragen werden wird.

Die erste Bedingung kann der TBN–Manager überprüfen. Die zweite läßt sich aber auf dem Abstraktionsgrad des TBN nicht beantworten. Hierzu sind nur die Systemkomponenten, die das Wissen über das realisierte Transaktionsmodell halten, in der Lage.

2.7 Globale Korrektheitsbedingungen für die TBN–Verwaltung

Die TBN–Verwaltung arbeitet korrekt, wenn sie für beliebige zulässige Transaktions-Zustands-Übergangswünsche eine minimale Menge von erforderlichen Zustandsübergängen dieser und weiterer Transaktionen liefert, die alle Restriktionen erfüllt, die durch das TBN gegeben sind. Dabei werden die impliziten Annahmen über Transaktionen (aus Kapitel 2.2) beachtet.

Der TBNM darf keine überflüssigen zustandsändernden Operationen fordern, z.B. allein aufgrund der Beziehung $(T_1 \;{}_{c}{\mapsto}_{b}\; T_2)$ und dem Wunsch `EOT(T`$_1$`) c(T`$_2$`)` zu verlangen.

Die Menge der geforderten Operationen darf keine Widersprüche enthalten und z.B. von einer Transaktion T gleichzeitig `c(T)` und `a(T)` fordern. Daher sind inkonsistente TBN nicht korrekt. Es ist Ermessenssache, ob man die Korrektheit weniger streng definieren und TBN, die momentan konsistent sind, aber gefährliche Beziehungen enthalten, als korrekt bezeichnen will.

Ein TBN ist korrekt, wenn es konsistent ist, d.h. keine inkonsistenten aggregierten Beziehungen impliziert. Wie kann die Korrektheit eines TBN beurteilt werden? Als Lösungsansatz kann man die Cliquen des TBN bilden, indem man die transitive Hülle der Beziehungen mit sämtlichen aggregierten Beziehungen berechnet. Alle, auch die abgeleiteten Beziehungen, stellt man explizit dar. Dann sind die Auswirkungen eines Zustandsüberganges auf alle Transaktionen direkt ablesbar; inkonsistente Forderungen, die als Folge von widersprüchlichen aggregierten Beziehungen entstehen, können erkannt werden.

Nichtleere Beziehungen zwischen Transaktionen gibt es nur innerhalb einer Clique. Transaktionen einer TBN-Clique sind völlig unabhängig von den Transaktionen einer anderen Clique und können unabhängige Konsistenzeigenschaften besitzen.

Kapitel 3

Transaktions-Propagations-Matrizen (TPM)

3.1 Einleitung

In diesem Kapitel stelle ich eine neue Notation zur Darstellung der Beziehungen vor. Diese Notation werde ich ab sofort auch im Text an Stelle der Pfeile verwenden, wo mir dies günstig erscheint. TPMs bieten zum einen eine deklarative Notation der Beziehungssemantik, die zum anderen auch als Grundlage einer operationalen Semantik herangezogen werden kann:

Die Operationen $\oplus$ und $\odot$, die in Kapitel 2.3.4 für Beziehungen definiert wurden, werden entsprechend auf der Datenstruktur TPM definiert. Andererseits läßt sich aus TPMs leicht eine Handlungsanweisung, die vergleichbar zu TRAMPs ist, ableiten. TPMs können ähnlich wie die Potenzautomaten in der Art dynamisch modifiziert werden, daß sich die erreichten Transaktionszustände in ihnen niederschlagen. Einer solchen TPM sieht man an, ob sie eine inkonsistente Beziehung repräsentiert.

Zu ihrer Interpretation ist die Kenntnis des Transaktionszustandsautomaten erforderlich: Nicht jede syntaktisch korrekte TPM stellt eine konsistente Beziehung dar. Vor ihrer Verwendung müssen TPMs normiert werden.

In Kapitel 4 wird auf Grundlage von TPMs ein TBN-Manager entworfen, der die in Kapitel 2 notierten Probleme zu lösen versucht.

3.2 Interpretation der Matrizen

Jede TPM besteht aus zwei Matrizen $TPM(vor)$ und $TPM(zurueck)$, die zusammen die Beziehung zwischen zwei Transaktionen T_1 und T_2 darstellen. Die Matrizen haben jeweils vier Zeilen und Spalten für die (zur Berechnung der Restriktionen relevanten) Transaktionszustände. Die Matrix $TPM(vor)$, die links dargestellt ist, beschreibt die Auswirkungen einer Zustandsänderung von T_1 auf T_2, die jeweils rechte Matrix $TPM(zurueck)$ die Auswirkungen von T_2 auf T_1. Mögliche Einträge in den Tabellen sind `*`, `+`, `-`, `i`, `@`, `n`, `!`. In der Matrix $TPM(vor)$ werden die Einträge wie folgt interpretiert: Wenn T_1 im durch die Zeile z_1 gegebenen Zustand ist, dann gilt für den Zustand von T_2 (aus Spalte z_2) beim Eintrag $TPM(vor, z_1, z_2) =$

„∗“: Der betreffende Zustand von T_2 ist kompatibel, muß aber nicht notwendigerweise erreicht werden. Es ist erlaubt und möglich, daß T_2 den Zustand z_2 erreicht (oder erreicht hat bzw. erreichen wird) unter der Voraussetzung, daß T_1 den Zustand von z_1 erreicht (erreicht hat, erreichen wird).

„+“: Der betreffende Zustand von T_2 ist kompatibel. Diese Zustandskombination darf nicht verboten werden. Sie muß allerdings auch nicht in jedem Fall (auch nicht zeitversetzt) erreicht werden. Wenn allerdings T_1 den Zustand z_1 erreicht hat oder erreichen wird, dann müß T_2 den Zustand z_2 erreichen.

„−“: Diese Zustandskombination ist nicht erlaubt.

„@“: Entspricht dem Eintrag „+“ mit der Erweiterung bzw. zusätzlichen Anforderung, daß der Zustandsübergang von T_2 in den Zustand z_2 sofort ausgeführt (d.h. gegebenenfalls vom TM angestoßen) wird, sobald T_1 den Zustand z_1 erreicht. Dieser Eintrag dient zur Modellierung der Kontrollflußbeziehungen.

„n“: Entspricht dem Eintrag „−“ mit Sofortausführung der komplementären Zustandsänderung.

„i“: Dieser Eintrag denotiert eine inkonsistente, widersprüchliche Anforderung. Der Zustand x_2 wird von T_2 sowohl gefordert als auch verboten.

„!“: Entspricht dem Eintrag „i“ mit zusätzlicher Anforderung Sofortausführung eines der Zustandsübergänge in den Zustand z_2 oder in den Komplementärzustand von z_2. Diese ergänzende Information gegenüber „i“ wird nur der Vollständigkeit halber dargestellt, da „Sofortausführung als orthogonal zu den anderen Forderungen angesehen wird, und bei der Interpretation der Matrizen nicht beachtet.

Wie bei der Erklärung von „∗“ bereits angedeutet, sind die zeitlichen Verhältnisse ausdrücklich nicht festgelegt.

Die Matrix $TPM(zurueck)$ interpretiert man analog als die Auswirkungen von T_2 auf T_1.

Aufgrund der zeilenweisen Symmetrie[1] in den Matrizen sind die Spalten ABORT (komplementär zu COMMIT) und DEMAND-NOT-RUN (komplementär zu BEGIN) redundant. Sie sind in der Implementierung nicht dargestellt. In der Dokumentation dienen sie allerdings der Übersichtlichkeit und der einfacheren Darstellung der Algorithmen. Bei der Integration der temporalen Beziehungen, die nach meinen vorläufigen Überlegungen diese Symmetrie nicht aufweisen, müßten sie zusätzlich integriert werden.

Nehmen wir als Beispiel die TPM der Beziehung ($T_1 \rightarrow_c T_2$). T_1 fordert genau in dem Fall `c(T`$_2$`)`, wenn es selbst den Zustand *committed* erreicht. Da aufgrund der Axiome des Transaktionsautomaten `c(T`$_2$`)` nicht ohne `b(T`$_2$`)` möglich ist, wird der Start von T_2 zusätzlich gefordert. Da dies alle Auswirkungen sind, hat $TPM(vor)$ die Form

T_1 \ T_2	beg	com	abo	dnr
beg	∗	∗	∗	∗
com	+	+	−	−
abo	∗	∗	∗	∗
dnr	∗	∗	∗	∗

[1] +ABORT gdw. −COMMIT, −ABORT gdw. +COMMIT, +BEGIN gdw. −DEMAND-NOT-RUN, −BEGIN gdw. +DEMAND-NOT-RUN

Erfüllt andererseits T_2 das Prädikat `abrt(`T_2`)`, dann darf T_1 nicht erfolgreich sein (sonst käme es nach obiger Argumentation zum Widerspruch). Daher fordert die Beziehung unter dieser Voraussetzung `abrt(`T_1`)`, was sich in den „+"–Einträgen in $TPM(zurueck)$ niederschlägt:

${}_{T_2}\,{}^{T_1}$	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$*$	$*$	$*$	$*$
abo	$*$	$-$	$+$	$*$
dnr	$*$	$-$	$+$	$*$

Die Spalte ABORT enthält also die Restriktionen, die allgemein im Fall „Abbruch der Transaktion" gelten und nicht nur im Spezialfall „Zustand ist *aborted*". Die Spalte DEMAND–NOT–RUN enthält die *zusätzlichen* Restriktionen für den Fall „Die Transaktion wurde nie gestartet".

Würde man als alternative Festlegung für die Spalte ABORT nur den Spezialfall und für die Spalte DEMAND–NOT–RUN *genau* die betreffenden Restriktionen wählen, dann löste sich erstens die Symmetrie auf, und zweitens könnten diese beiden Spalten nicht mehr als konjunktiv verknüpft betrachtet werden: der Vorteil des Prädikats `abrt` ginge verloren, da das Gegenteil von *committed* nicht mehr eindeutig wäre.

Wählen wir auch bei den *Zeilen* die Festlegung „ABORT entspricht `abrt`", dann treten bei einigen der Kontrollflußbeziehungen in der Zeile ABORT „Doppeleinträge" mit zwei Symbolen auf. Da die Eigenschaft `abrt` auch auf Transaktionen zutreffen soll, die gar nicht laufen werden (*demand-not-run*), dieser Zustand aber vom Zustand *aborted* unterschieden werden muß, sind die constraints in der Zeile ABORT nicht eindeutig. In diesem Fall gilt der obere Eintrag für eine TA, die abgebrochen wurde, nachdem sie anfing zu laufen (der „echte" Zustand *aborted*). Der untere Eintrag gilt für Transaktionen, die im Zustand *demand-not-run* sind. Beispielsweise würde die TPM für ${}_b\!\!\mapsto_c$ wie folgt aussehen:

${}_{T_1}\,{}^{T_2}$	beg	com	abo	dnr
beg	$+$	$+$	$-$	$-$
com	$+$	$+$	$-$	$-$
abo	$+$ $*$	$+$ $*$	$-$ $*$	$-$ $*$
dnr	$*$	$*$	$*$	$*$

${}_{T_2}\,{}^{T_1}$	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$*$	$*$	$*$	$*$
abo	$-$	$-$	$+$	$+$
dnr	$-$	$-$	$+$	$+$

Aus diesem Grund ist es sinnvoll, die Zeilen exakt den Zuständen zuzuordnen.

i und ! sind Einträge, die in den elementaren Beziehungen bei installierten, noch nicht gestarteten Transaktionen nicht vorkommen. Sie können in abgeleiteten Beziehungen entstehen (wie zum Beispiel bei folgender Addition: zwischen T_1 und T_2 sollen gleichzeitig die beiden Beziehungen $\rightarrow_c$ und ${}_c\!\!\rightarrow_a$ gelten.

$$(T_1 \rightarrow_c T_2) \oplus (T_1\ {}_c\!\!\rightarrow_a T_2) = \left(T_1 \begin{array}{||c||c|c|c|c||} {}_{T_1}\,{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & * & * & * & * \\ \mathrm{c} & i & i & i & i \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \end{array} \quad \begin{array}{||c||c|c|c|c||} {}_{T_2}\,{}^{T_1} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & * & - & + & * \\ \mathrm{c} & * & - & + & * \\ \mathrm{a} & * & - & + & * \\ \mathrm{d} & * & - & + & * \end{array} T_2 \right)$$

Andererseits können sie ausdrücken, daß eine Transaktion einen zum von der Beziehung geforderten Zustand inkompatiblen Zustand erreicht hat. Beispiel: Zwischen T_1 mit Zustand *installed* und T_2

mit Zustand *aborted* besteht die Beziehung ($T_1 \rightarrow_c T_2$). Die TPM in dieser Situation besitzt folgende Gestalt:

$_{T_1}{}^{T_2}$	b	c	a	d
b	*	*	*	*
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

$_{T_2}{}^{T_1}$	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	*	−	+	*

Einzelne i- und !-Einträge machen eine gesamte Matrix bzw. Beziehung allerdings nicht notwendig inkonsistent (siehe Kapitel 3.4.3). Sie implizieren aber immer Zustandsforderungen (*demands*) für Transaktionen: Eine Transaktion darf nicht in den Zustand s gelangen, wenn die TPM–Zeile s mindestens einen Inkonsistent-Eintrag enthält. Im Beispiel darf T_1 den Zustand *committed* nicht erreichen, ist also `demand-abort`.

3.3 TPMs der elementaren Beziehungen

3.3.1 Logische Beziehungen

besitzen folgende TPMs:

1. $T_1 \rightarrow_c T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	*	*	*
dnr	*	*	*	*

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	*	−	+	*
dnr	*	−	+	*

2. $T_1 \rightarrow_a T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	*	−	+	*
dnr	*	−	+	*

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	*	*	*
dnr	*	*	*	*

(äquivalent zu $T_2 \rightarrow_c T_1$)

3. $T_1 \leftrightarrow_c T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	−	+	*
dnr	*	−	+	*

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	−	+	*
dnr	*	−	+	*

(reflexiv)

4. $T_1 \leftrightarrow_a T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	−	+	*
dnr	*	−	+	*

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	*	*	*	*
com	+	+	−	−
abo	*	−	+	*
dnr	*	−	+	*

(äquivalent zu $T_1 \leftrightarrow_c T_2$)

5. $T_1 \;{}_a\!\!\longrightarrow_c\; T_2$:

(reflexiv)

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	+	+	−	−
dnr	+	+	−	−

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	+	+	−	−
dnr	+	+	−	−

6. $T_1 \;{}_c\!\!\longrightarrow_a\; T_2$:

(reflexiv)

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	*	*	*	*
com	*	−	+	*
abo	*	*	*	*
dnr	*	*	*	*

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	−	+	*
abo	*	*	*	*
dnr	*	*	*	*

3.3.2 Kontrollflußbeziehungen

Bei Kontrollflußbeziehungen hängen die Auswirkungen auf die jeweils andere Transaktion nicht nur von dem zu erreichenden Endzustand, sondern auch vom Erreichen des Zustandes *running* der Transaktion ab.

1. $T_1 \;{}_c\!\!\longmapsto_b\; T_2$:

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	*	*	*	*
com	@	*	*	n
abo	*	*	*	*
dnr	*	*	*	*

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	*	*	*	*
dnr	*	−	+	*

2. $T_1 \;{}_a\!\!\longmapsto_b\; T_2$:

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	@	*	*	n
dnr	@	*	*	n

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	*	*	*	*
dnr	+	+	−	−

3. $T_1 \;{}_b\!\!\longmapsto_c\; T_2$:

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	+	+	−	−
com	+	+	−	−
abo	+	+	−	−
dnr	*	*	*	*

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	−	−	+	+
dnr	−	−	+	+

4. $T_1 \;{}_b\!\!\longmapsto_a\; T_2$:

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	*	−	+	*
com	*	−	+	*
abo	*	−	+	*
dnr	*	*	*	*

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	−	−	+	+
abo	*	*	*	*
dnr	*	*	*	*

5. $T_1 \;\longmapsto_b\; T_2$:

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	@	*	*	n
com	+	*	*	−
abo	+	*	*	−
dnr	*	*	*	*

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	*	*	*	*
com	*	*	*	*
abo	*	*	*	*
dnr	n	−	+	@

6. $T_1 \leftrightarrow_b T_2$:

(reflexiv)

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	$+$	$*$	$*$	$-$
com	$+$	$*$	$*$	$-$
abo	$+$	$*$	$*$	$-$
dnr	$-$	$-$	$+$	$+$

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	$+$	$*$	$*$	$-$
com	$+$	$*$	$*$	$-$
abo	$+$	$*$	$*$	$-$
dnr	$-$	$-$	$+$	$+$

3.3.3 Zeitliche Beziehungen

Die zeitlichen Beziehungen implizieren keine logischen Abhängigkeiten. Falls gewünscht müssen diese gesondert angegeben werden.

Die Tabelleneinträge haben bei einer Beziehung $T_k \to T_m$ folgende Bedeutung: In $TPM(vor, l, n)$ steht der Eintrag $dep_j =$

„<“: Während T_k in den Zustand l übergeht[2], darf T_m noch nicht in Zustand n sein.

„>“: Während T_k in den Zustand l übergeht, muß T_m im Zustand n sein (oder in einem Folgezustand von n).

Werden in einen Tabelleneintrag „<“ und „>“ gleichzeitig eingetragen, ist dieser Eintrag inkonsistent.

Die Einträge werden bei der Normierung der TPM (in `checkTPM()`, siehe Kapitel 3.4.3.1) wie folgt fortgeschrieben:

„<“ gilt transitiv für vorangehende Zustände der ersten Transaktion (bzw. für nachfolgende Zustände der zweiten Transaktion): das entspricht dem Kopieren „nach oben und nach rechts“ in der TPM.

„>“ gilt transitiv für nachfolgende Zustände der ersten Transaktion (bzw. für vorangehende Zustände der zweiten Transaktion): das entspricht dem Kopieren „nach unten und nach links“ in der TPM.

1. $T_1 <_c T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	$*$	$<$	$*$	$*$
com	$*$	$<$	$*$	$*$
abo	$*$	$*$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$>$	$>$	$*$	$*$
abo	$*$	$*$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

2. $T_1 \ {}_a{<}_b \ T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$*$	$*$	$*$	$*$
abo	$<$	$<$	$\stackrel{<}{*}$	$*$
dnr	$<$	$<$	$\stackrel{<}{*}$	$*$

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	$*$	$*$	$>$	$>$
com	$*$	$*$	$>$	$>$
abo	$*$	$*$	$\stackrel{>}{*}$	$\stackrel{>}{*}$
dnr	$*$	$*$	$*$	$*$

3. $T_1 \ {}_c{<}_b \ T_2$:

$_{T_1}{}^{T_2}$	beg	com	abo	dnr
beg	$<$	$<$	$\stackrel{<}{*}$	$*$
com	$<$	$<$	$\stackrel{<}{*}$	$*$
abo	$*$	$*$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

$_{T_2}{}^{T_1}$	beg	com	abo	dnr
beg	$>$	$>$	$*$	$*$
com	$>$	$>$	$*$	$*$
abo	$\stackrel{>}{*}$	$\stackrel{>}{*}$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

[2]„übergeht“ statt „ist“: sonst gäbe es Probleme bei Endzuständen für n.

4. $T_1 \; {}_b{<}_c \; T_2$:

T_1 \ T_2	beg	com	abo	dnr
beg	$\ast$	$<$	$\ast$	$\ast$
com	$\ast$	$\ast$	$\ast$	$\ast$
abo	$\ast$	$\ast$	$\ast$	$\ast$
dnr	$\ast$	$\ast$	$\ast$	$\ast$

T_2 \ T_1	beg	com	abo	dnr
beg	$\ast$	$\ast$	$\ast$	$\ast$
com	$>$	$\ast$	$\ast$	$\ast$
abo	$\ast$	$\ast$	$\ast$	$\ast$
dnr	$\ast$	$\ast$	$\ast$	$\ast$

5. $T_1 <_b T_2$:

T_1 \ T_2	beg	com	abo	dnr
beg	$<$	$<$	$\underset{\ast}{<}$	$\ast$
com	$\ast$	$\ast$	$\ast$	$\ast$
abo	$\ast$	$\ast$	$\ast$	$\ast$
dnr	$\ast$	$\ast$	$\ast$	$\ast$

T_2 \ T_1	beg	com	abo	dnr
beg	$>$	$\ast$	$\ast$	$\ast$
com	$>$	$\ast$	$\ast$	$\ast$
abo	$\underset{\ast}{>}$	$\ast$	$\ast$	$\ast$
dnr	$\ast$	$\ast$	$\ast$	$\ast$

3.4 Operationen auf TPMs

3.4.1 Addition $\oplus$

Die Addition aggregiert die Information von zwei TPMs. Die Summe zweier TPMs drückt das gleichzeitige Bestehen der zwei entsprechenden Beziehungen zwischen den Transaktionen T_1 und T_2 aus. Durch die Addition mit anschließender Überprüfung der TPM wird festgestellt, ob die zwei Beziehungen miteinander verträglich sind: ergibt die Summe eine konsistente TPM, sind offensichtlich beide Beziehungen gleichzeitig zu erfüllen und dürfen daher gleichzeitig eingetragen werden.

Die Summe zweier TPMs wird berechnet, indem — analog zur Matrixaddition in der Algebra — jeweils die Matrizen $TPM(vor)$ und $TPM(zurueck)$ addiert werden. Dabei gelten folgende Rechenregeln:

$\ast$	$\oplus$	$\ast$	=	$\ast$	+	$\oplus$	$\ast$	=	+	$-$	$\oplus$	$\ast$	=	$-$	i	$\oplus$	$\ast$	=	i
$\ast$	$\oplus$	+	=	+	+	$\oplus$	+	=	+	$-$	$\oplus$	+	=	i	i	$\oplus$	+	=	i
$\ast$	$\oplus$	$-$	=	$-$	+	$\oplus$	$-$	=	i	$-$	$\oplus$	$-$	=	$-$	i	$\oplus$	$-$	=	i
$\ast$	$\oplus$	i	=	i	+	$\oplus$	i	=	i	$-$	$\oplus$	i	=	i	i	$\oplus$	i	=	i
$\ast$	$\oplus$	@	=	@	+	$\oplus$	@	=	@	$-$	$\oplus$	@	=	!	i	$\oplus$	@	=	!
$\ast$	$\oplus$	n	=	n	+	$\oplus$	n	=	!	$-$	$\oplus$	n	=	n	i	$\oplus$	n	=	!
$\ast$	$\oplus$	!	=	!	+	$\oplus$	!	=	!	$-$	$\oplus$	!	=	!	i	$\oplus$	!	=	!

@	$\oplus$	$\ast$	=	@	n	$\oplus$	$\ast$	=	n	!	$\oplus$	$\ast$	=	!
@	$\oplus$	+	=	@	n	$\oplus$	+	=	!	!	$\oplus$	+	=	!
@	$\oplus$	$-$	=	!	n	$\oplus$	$-$	=	n	!	$\oplus$	$-$	=	!
@	$\oplus$	i	=	!	n	$\oplus$	i	=	!	!	$\oplus$	i	=	!
@	$\oplus$	@	=	@	n	$\oplus$	@	=	!	!	$\oplus$	@	=	!
@	$\oplus$	n	=	!	n	$\oplus$	n	=	n	!	$\oplus$	n	=	!
@	$\oplus$	!	=	!	n	$\oplus$	!	=	!	!	$\oplus$	!	=	!

Als Teil der Additionsoperation werden anschließend die Einträge in den Matrizen propagiert (siehe Kapitel 3.4.3).

Die Addition ist assoziativ und kommutativ. Das neutrale Element ist die leere Beziehung.

Es gibt keine inverse Operation $\ominus$ zu $\oplus$. Beispiel:

$$\rightarrow_c \oplus \leftrightarrow_c = \leftrightarrow_c \text{ , aber auch } \rightarrow_c \oplus {}_c\!\leftarrow = \leftrightarrow_c$$

Was sollte das Ergebnis von $\leftrightarrow_c \ominus \rightarrow_c$ sein? Sinnvoll wäre ${}_c\!\leftarrow$, aber für die Verwaltung der Beziehungen müssen die „restlichen“ ursprünglich eingetragenen Beziehungen (oder deren Summe) als Ergebnis erscheinen. Dies ist aber offensichtlich (siehe obiges Beispiel) nicht zu erreichen.

3.4.2 Multiplikation ⊙

Wenn $T_1\ dep_{12}\ T_2\ dep_{23}\ T_3$ gilt, dann berechnet die Multiplikation von dep_{12} mit dep_{23} die indirekte Beziehung[3] dep_{13} zwischen T_1 und T_3. Die Zeilen z von $TPM_k(vor)$ dieser Beziehung, also die Auswirkungen von T_1 auf T_3, ermittelt man beginnend mit der leeren TPM wie folgt:

1. Sind in Zeile z der Matrix $TPM_{12}(vor)$ keine Restriktionen für T_2 verzeichnet (d.h. die Einträge sind „$*$“), dann gibt es auch keine Einschränkungen für T_3: setze in $TPM_{13}(vor)$ in Zeile z alle Einträge auf „$*$“.
2. Ist in Zeile z der Matrix $TPM_{12}(vor)$ der Zustand z_2 der TA T_2 gefordert, d.h. der Eintrag $TPM_{12}(vor, z, z_2)$ ist „+“, „i“ oder „!“, dann propagiere[4] die Zeile z_2 aus $TPM_{23}(vor, z, z_2)$ in die Zeile z der Matrix $TPM_{13}(vor)$.

Diese Schritte werden für alle Zeilen wiederholt. Als Teil der Multiplikationsoperation werden anschließend die Einträge in den Matrizen propagiert (siehe Kapitel 3.4.3).

Die Multiplikation ist nicht kommutativ.

Stehen die Faktor-Beziehungen in der „verkehrten Richtung“ zur Verfügung, z.B. $T_1 \rightarrow_c T_2$ und $T_3 \;{}_a\!\!\mapsto_b T_2$, dann kann die indirekte Beziehung zwischen T_1 und T_3 erst berechnet werden, nachdem eine der TPMs (im Beispiel die zweite) invertiert wurde: Die inverse Beziehung (Beziehung in Gegenrichtung) erhält man, indem die Matrizen $TPM(vor)$ und $TPM(zurueck)$ gegeneinander ausgetauscht werden.

Als Beispiel berechnen wir die indirekte Beziehung zwischen T_1 und T_3, wenn die Beziehungen $dep_{12} = (T_1 \rightarrow_c T_2)$ und $dep_{23} = (T_2 \;{}_c\!\!\rightarrow_a T_3)$ mit den entsprechenden TPMs gelten. Wir beginnen mit den Auswirkungen von T_1 auf T_2, also $TPM_{13}(vor)$. In den Zeilen „begin“, „abort“ und „demand–not–run“ von $TPM_{12}(vor)$ stehen keine Restriktionen, also werden in diesen Zeilen auch in $TPM_{13}(vor)$ keine eingetragen. $TPM_{12}(vor)$ enthält in Zeile „commit“ in den Spalten „begin“ und „commit“ den Eintrag „+“. Daher addieren wir die Zeilen „begin“ und „commit“ aus $TPM_{23}(vor)$ zur Zeile „commit“ von $TPM_{13}(vor)$: $TPM_{13}(vor)$ hat die Gestalt

T_1 T_3	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$*$	$-$	$+$	$*$
abo	$*$	$*$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

$TPM_{13}(zurueck)$ beschreibt die Auswirkungen von T_1 auf T_3 und wird wie folgt berechnet: $TPM_{13}(zurueck)$ enthält in den Zeilen „begin“, „abort“ und „demand–not–run“ nur „$*$“–Einträge, da in diesen Zeilen in $TPM_{23}(zurueck)$ keine Einschränkungen verzeichnet sind. $TPM_{23}(zurueck)$ enthält in Zeile „commit“ und Spalte „commit“ den Eintrag „$-$“ (und folglich in Spalte „abort“ „+“, weshalb zur Zeile „commit“ von $TPM_{13}(zurueck)$ die Zeile „abort“ von $TPM_{12}(zurueck)$ addiert wird. Alle Einträge von $TPM_{23}(zurueck)$ wurden bearbeitet, also sieht $TPM_{13}(zurueck)$ wie folgt aus:

T_3 T_1	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$*$	$-$	$+$	$*$
abo	$*$	$*$	$*$	$*$
dnr	$*$	$*$	$*$	$*$

[3] Manchmal auch nicht ganz korrekt „transitive Beziehung“genannt.

[4] Korrekt muß es „addiere“ heißen. Diese Einschränkungen werden zusätzlich zu eventuell schon berechneten Einträgen aufgenommen.

Somit besteht die indirekte Beziehung[5] ($T_1 \ {}_c\!\!\rightarrow_a T_2$).

3.4.3 Propagation von Einträgen innerhalb von TPMs

Aus den Axiomen des Zustandsautomaten der Transaktionen folgt, daß nicht alle syntaktisch erlaubten TPMs sinnvoll sind. Einträge in einer TPM werden daher in der Funktion `checkTPM()` (siehe Kapitel 3.4.3.1) mit dem Ziel propagiert, eine Matrix zu erhalten, die die (maximale) Einschränkung explizit beschreibt, die sich aus der gegebenen TPM ableiten läßt. Beispielsweise folgt aus dem Axiom $c(T) \Rightarrow b(T)$, daß alle Restriktionen, die in der Zeile *beg* gelten, auch in der Zeile *com* anzuwenden sind.

Wenn die beiden Teilmatrizen $TPM(vor)$ und $TPM(zurueck)$ einander widersprechende Einträge enthalten, werden diese Einträge auf „inkonsistent“ gesetzt.

Beispiel: Die vordere Matrix darf für den Fall $c(T_1)$ nicht $a(T_2)$ fordern, wenn die hintere Matrix in Zeile *abo* und Spalte *com* den Eintrag „-“ (= verboten) enthält.

Wenn eine Zeile x der TPM einen Inkonsistent-Eintrag enthält, darf die Transaktion den Zustand x nicht erreichen. Die Transaktion ist also DEMAND–NOT–$< x >$ (und in der Spalte x der anderen Matrixseite wird in allen Zeilen „$-$“ addiert).

Bei der dynamischen Verwaltung der Beziehungen durchlaufen die beteiligten Transaktionen ihren Zustandsautomaten. Der erreichte (bzw. der geforderte) Zustand der ersten TA schränkt die möglichen Zustände der zweiten Transaktion weiter ein. In der TPM stellt man diese Tatsache so dar, daß in $TPM(vor)$ in der Zeile und in $TPM(zurueck)$ in der Spalte des erreichten Zustandes zu allen Einträgen „+“ addiert wird: Dieser erreichte Zustand darf nicht verboten werden. Tritt dieser Fall dennoch ein, wird die Beziehung (und das TBN–Teilnetz) inkonsistent.

Beispiel: $a(T_1)$ bei $T_1 \rightarrow_c T_2$ ergibt die resultierende Matrix

	T_1 \ T_2	beg	com	abo	dnr	T_2 \ T_1	beg	com	abo	dnr	
T_1	beg	$*$	$*$	$*$	$*$	beg	$*$	$-$	$+$	$*$	T_2
	com	i	i	$-$	$-$	com	$*$	$-$	$+$	$*$	
	abo	$+$	$+$	$+$	$+$	abo	$*$	$-$	$+$	$*$	
	dnr	$*$	$*$	$*$	$*$	dnr	$*$	$-$	$+$	$*$	

Die Überprüfung bzw. Realisierung dieser Regeln ergibt den gesuchten Algorithmus:

3.4.3.1 Algorithmus `checkTPM()`

initialisiere beide Transaktionen als DEMAND–$< aktuellerTA - Zustand >$.

wiederhole, bis keine weiteren Änderungen mehr vorgenommen wurden:

- Die Einträge aus der Zeile BEGIN nach den Zeilen COMMIT und ABORT aufaddieren. Diese Operation ist eine Konsequenz aus den Axiomen, die den Transaktionsautomaten beschreiben.
- Wenn $TPM(vor, zeile, spalte) = +$ oder i und
  $TPM(zurueck, spalte, zeile) = -$ oder i,
  dann setze $TPM(vor, zeile, spalte) = i$
  (analog auch in Rückrichtung).

[5] Auch die Anwendung von `checkTPM()` bewirkt keine weiteren Änderungen, da keiner der Fälle vorliegt, wo Einträge propagiert werden müssen.

- Wenn $TPM(vor, zeile, spalte) = i$,
  dann setze T_1 DEMAND–NOT–$< zeile >$
  (analog auch für *zurueck* und T_2).

- Wenn $TPM(vor, zeile_1, spalte) = -$ oder i und
  $TPM(vor, zeile_2, spalte) = -$ oder i und
  $TPM(vor, zeile_3, spalte) = -$ oder i, wobei

$zeile_1$	$zeile_2$	$zeile_3$
BEG	DNR	—
COM	ABO	DNR

  ,
  dann ist T_2 DEMAND–NOT–$< spalte >$
  (analog auch in Rückrichtung und für T_1).

- Wenn $TPM(vor, zeile_1, spalte) = +$ oder i und
  $TPM(vor, zeile_2, spalte) = +$ oder i und
  $TPM(vor, zeile_3, spalte) = +$ oder i, wobei

$zeile_1$	$zeile_2$	$zeile_3$
BEG	DNR	—
COM	ABO	DNR

  ,
  dann ist T_2 DEMAND–$< spalte >$ (analog auch in Rückrichtung und für T_1).

- Wenn T_1 DEMAND-$< x >$ ist, dann setze in allen Zeilen *zeile*
  $TPM(zurueck, zeile, x) = TPM(zurueck, zeile, x) \oplus$„+“.
  In der Komplementärspalte von x addiere entsprechend „$-$“.
  Das Ganze ebenfalls analog für T_2.

3.4.4 Korrektheitsbedingung für TPMs

Genau dann, wenn bei der „matrixinternen“ Propagation im obigen Algorithmus widersprüchliche *demands* auftreten, ist die dargestellte Beziehung inkonsistent. Insbesondere macht ein einzelner „i“–Eintrag in einer Matrix die Beziehung nicht notwendigerweise inkonsistent. Die Transaktion darf den Zustand der Zeile mit dem „i“–Eintrag nicht erreichen. Dieser Eintrag impliziert also nur, daß die Transaktion einen (anderen) Zustand erreichen muß unabhängig davon, welche weiteren Zustandsübergänge die andere Transaktion vollzieht. Im Beispiel auf Seite 38 muß T_1 daher abbrechen.

Alle sonstigen TPM sind konsistent: sie entsprechen zulässigen Restriktionen des gemeinsamen Transaktionsautomaten.

Kapitel 4

Entwurf

4.1 Anforderungen an den TBN–Manager (TBNM)

- Der Zugriff auf TA-Zustände und auf das TBN ist bereits beim Eintragen von Beziehungen im dynamischen Betrieb erforderlich, da sie den Spielraum zusätzlich einschränken, welche Zustandsübergänge erlaubt und welche Beziehungen eingetragen werden können.
- Der TBN-Manager stellt fest, ob es durch das Ausführen der Operationen zu Inkonsistenzen käme. Das kann auch beispielsweise beim Migrieren passieren, wenn eine „importierte“ Transaktion bereits lokal in Beziehungen steht. Der TM entscheidet, ob die Operation verhindert oder trotzdem durchgeführt wird.
- Der TBN-Manager muß zwischen ursprünglich eingetragenen Beziehungen und abgeleiteten, aggregierten Beziehungen unterscheiden. Diese Unterscheidung und ihre Speicherung ist notwendig, wenn Beziehungen gelöscht werden dürfen, bevor alle betroffenen Transaktionen in Endzustände gelangt sind. In diesem Fall müssen die abgeleiteten Beziehungen neu berechnet werden, da einer TPM nicht angesehen werden kann, welche der ursprünglichen Beziehungen in ihr aggregiert sind.
- Es reicht nicht aus, alle aggregierten Beziehungen beim Eintragen einer Beziehung statisch zu berechnen, da die Information in den Matrizen nicht vollständig ist. Bei Zustandsänderungen von Transaktionen müssen die resultierenden Einschränkungen (TPMs) berechnet werden.

4.2 Anforderungen des TBNM an andere Systemkomponenten

- Das Sperren von Transaktionen bzgl. zustandsändernden Operationen muß möglich sein, um eine korrekte und atomare Berechnung der Propagation zu gewährleisten.
- Ein Zustand *installed* für Transaktionen wird eingeführt: Ein global eindeutiger Transaktionsname (GTRID) ist bekannt, und es können Beziehungen zu Transaktionen eingetragen und verwaltet werden, die noch gar nicht aktiv sind. GTRIDs und Transaktionszustände müssen so lange aufgehoben werden, bis
  - alle Transaktionen, die mit ihr in Verbindung stehen, in einen Endzustand gelangt sind

  - sichergestellt ist, daß keine Beziehung, die (mindestens) eine dieser Transaktionen betrifft, eingefügt wird.

- Um für die (Kontrollfluß-)Beziehungen ${}_b\!\!\mapsto_c$, $\mapsto_b$, $\leftrightarrow_b$ eine prozedurale Semantik angeben zu können, wurde der Zustand demand-not-run eingeführt (z.B. darf in der Situation ($T_1 \; {}_b\!\!\mapsto_c T_2$) und den Zuständen `i(T₁)`, `a(T₂)` die Transaktion T_1 nicht starten).

4.3 Aufgabenteilung zwischen TBN–Manager und Transaktions–Manager

Wie im Systemmodell aus Abbildung 1.5 skizziert, ist der Transaktions–Beziehungsnetz–Manager eine Subkomponente des Transaktions–Managers. Ihr Zusammenspiel ist das Thema der folgenden Kapitel.

4.3.1 Was leistet der TBN–Manager?

Der TBNM berechnet Zustandsübergänge von Transaktionen, die durch eine der Operationen „eine neue Beziehung eintragen“ und „Zustand einer Transaktion ändern“ aufgrund der Beziehungen im Netz gefordert werden. Diese Forderungen liefert er in Form einer Liste (Agenda) zurück. Diese Agenda enthält Tupeln aus einer Transaktion und einem demand (Zustands-Übergangs-Anforderung). Die Reihenfolge in dieser Liste ist willkürlich und irrelevant. Fordert der TBNM beispielsweise, daß T_1 erfolgreich beendet wird (wobei T_1 noch nicht gestartet wurde) und daß T_2 abbricht, dann hat die Agenda folgende Gestalt: **((T$_1$,dem-run), (T$_1$,dem-commit), (T$_2$,dem-abort))**

Zusätzlich meldet der TBNM, ob das TBN (genauer: das zusammenhängende Teilnetz) inkonsistent geworden ist.

Der TBNM verwaltet Netze: er aktualisiert sie, indem er Beziehungen einträgt und löscht, und er hält unabhängige Cliquen von Transaktionen.

Er beantwortet Fragen der Art „*Was wäre, wenn ...* “, die der TM an ihn stellt. Alle Aktionen hierzu arbeiten auf einer Kopie des TBN, dem sog. *future state*. Auf Anweisung des TM wird diese Kopie als das neue gültige TBN (*current state*) übernommen.

4.3.2 Was leistet der realisierte TBN–Manager nicht?

Pro Clique kann zu einem Zeitpunkt nur eine Berechnung offen[1] sein. Es gibt genau einen *savepoint*, den aktuellen Netzzustand *current state*, auf den zurückgesetzt werden kann. Der TBN-Manager verwaltet genau eine Kopie des Netzes (*future state*), auf der Berechnungen stattfinden.

Der TBNM arbeitet deterministisch mit der Hypothese, daß alle geforderten Zustandsübergänge (einschließlich desjenigen, mit dem der TBNM aufgerufen wurde) erfolgreich sind. Der TBNM führt kein Backtracking durch (auch nicht bei EOT-Aufrufen: wenn commit nicht möglich sein sollte, meldet er nur „inkonsistent“; Der TM muß explizit eine neue, auf *current state* aufsetzende Berechnung starten, um die Konsequenzen eines Abbruchs zu ermitteln). Der TBNM kann auf einer Menge

[1] Eine *offene Anfrage* ist ein Wunsch nach Zustandsänderungen bzw. neuen Beziehungen, nach deren Bearbeitung der TBNM eine Agenda geliefert hat, die der TM momentan noch nicht vollständig abgearbeitet, d.h. noch nicht zu allen geforderten Zustandsübergängen die entsprechende Operation aufgerufen hat. Diese Agenda wird analog als *offene Agenda* bezeichnet.

von Transaktionen und Beziehungen nicht mehrere, einander widersprechende Anfragen bearbeiten, einzelne Hypothesen isoliert zurücknehmen oder Backtracking durchführen.

Der TBNM führt kein Zwei-Phasen-Commit-Protokoll durch. Aus seiner Sicht ist der Zustand *prepared* identisch zum Zustand *running*: von beiden aus gibt es die Alternative, die Transaktion erfolgreich oder nicht erfolgreich zu beenden. Als *demand* gibt er nie `p(T)`, sondern nur `c(T)` aus.

Ein TBN-Manager hat eine lokale Sicht auf sein Netz. Er hält dieses TBN für systemweit vollständig. Er kennt keine verteilten TBNs und kann nicht mit seinesgleichen kommunizieren.

Die zeitlichen Beziehungen sind in dieser Version nicht integriert. Ihre Behandlung ist aber prinzipiell möglich, indem die Notation und Basisrechenregeln wie angedeutet erweitert werden. Außerdem müssen die Algorithmen so angepaßt werden, daß sich die Restriktionen der zeitlichen Beziehungen untereinander sowie in Wechselwirkungen mit den anderen Beziehungstypen auswirken.

4.3.3 Was ist Aufgabe des Transaktions–Managers?

- Der TM generiert global eindeutige Transaktionsnamen (GTRID) und stellt dem TBNM einen Zugriff auf Daten von Typ „Transaktion“ zur Verfügung.
- Die Transaktionsverwaltung ist verpflichtet, sich die Konsequenzen eines Zustandsübergangs, die sich aus den Beziehungen ergeben, vom TBN–Manager berechnen zu lassen, bevor sie den Zustand einer Transaktion ändert, um die von den Beziehungen geforderten Operationen auf den anderen Transaktionen ebenfalls anzustoßen. Als Grundlage dafür bekommt er vom TBNM nach jedem Aufruf eine Agenda, die die Forderungen des lokalen TBN enthält. Der TM muß diese Agenda abarbeiten und in Aufrufe an die Resource Manager (RM), z.B. Datenbanken, umsetzen. Der TM hat eine Schnittstelle zu anderen Komponenten, insbesondere zum ConTract-Manager und anderen RM, die Beziehungen eintragen und Zustandsübergänge anstoßen.
- Bei den Anfragen $\texttt{RUN(T}_1\texttt{)}$ und $\texttt{EOT(T}_1\texttt{)}$ ist im TM eine Sonderbehandlung vorgesehen, die berücksichtigt, daß die Umsetzung dieser Wünsche verzögert oder durch `ABORT(T)` verhindert werden kann:
  - Falls Inkonsistenzen auftreten: Der TM läßt den TBNM versuchsweise die Auswirkungen von $\texttt{ABORT(T}_1\texttt{)}$ berechnen und arbeitet die gelieferte Agenda ab. Wurde allerdings wiederum „inkonsistent“ gemeldet, ist das TBN inkonsistent.

    Dieses Protokoll ist eine Konsequenz aus der Entscheidung, den TBNM deterministisch zu gestalten.
  - Der TM sortiert die ermittelten Zustandsübergänge auf der Agenda.

    In einer ersten Phase werden die Transaktionen T mit Zustandsübergängen nach *committed* aussortiert. Sind alle diese Transaktionen im Zustand `p(T)`, wird für jede von ihnen `EOT(T)` aufgerufen. Ist mindestens eine noch nicht soweit, dann ruft der TM $\texttt{PREPARE(T}_1\texttt{)}$ auf und wartet ab, bis alle diese Transaktionen T *prepared* sind.

    Falls bisher kein Widerspruch zu der Agenda aufgetreten ist, werden in der zweiten Phase alle noch ausstehenden Zustandsübergänge durchgeführt.

    Wenn Inkonsistenzen aufgetreten sind und der auslösende Änderungswunsch war $\texttt{BOT}(T_1)$ oder $\texttt{EOT}(T_1)$, dann kann der TM den Algorithmus mit DNR(T_1) bzw. ABORT(T_1) ausgehend vom aktuellen Netzzustand (current state) erneut aufrufen (siehe den ersten Punkt dieser Sonderbehandlung).

    Den Zustand *prepared* sieht der TBNM als einen hinreichenden Nachweis dafür an, daß die Transaktion tatsächlich einen EOT-Aufruf abgesetzt hat, um in den Zustand *committed* zu gelangen, und das sofort oder zu einem späteren Zeitpunkt. Es gibt keinen Fall *„prepared–to–abort“*.

Zu diesem Verfahren unter einem allgemeinerem Gesichtspunkt siehe auch [Lutz].

- Der TM schaltet den TBN-Status um zwischen dem aktuellem Zustand (*current state*) und dem Zustand für vorläufige Berechnungen, die explizit übernommen oder verworfen werden können (*future state*). Auf diese Weise kann er Beziehungen vorläufig eintragen und sich die Auswirkungen anzeigen lassen. Je nachdem, ob der TBNM z.B. „inkonsistent“ meldet, kann er die Beziehungen endgültig eintragen, indem er den zukünftigen, hypothetischen Zustand zum aktuellen Zustand macht, oder andere Maßnahmen ergreifen.

  Damit der TM die vollständige und alleinige Kontrolle über den TBN-Zustand behält, muß er die oben erwähnte Sonderbehandlung durchführen. Das Backtracking von „versuche COMMIT“ zu „versuche ABORT“ bedingt das Rücksetzen auf *current state* nach dem ersten Versuch. Wenn der TBNM dies nicht selbst unternimmt, sondern dem TM überläßt, kann dieser u.U. zur Optimierung mehrere Operationen auf *future state* vornehmen, ohne nach jeder Berechnung rücksetzen zu müssen.

  Außerdem kann das hilfreich zur Implementierung eines verteilten TBN sein (siehe nächster Punkt).

- Der TM implementiert eine verteilte Transaktionsverwaltung, falls dies gefordert und sinnvoll sein sollte. Er übernimmt gegebenenfalls die Koordination zwischen TBNM, die auf unterschiedlichen Rechnerknoten laufen. Aus diesem Grund hat er die alleinige Kontrolle über den gesamten Bereich der Zwei-Phasen-Commit-Protokolle.

  Eine verteilte Transaktionsverwaltung mit verteilten TBN muß durch kooperierende TM realisiert werden [Lutz]. Falls das Netz im verteilten Fall nicht alle Beziehungen kennt, die transitiv betroffen sein können, muß der TM für das Migrieren des Teilnetzes sorgen und so den lokalen Fall herbeiführen.

  Als Alternative könnte er jeweils die vom TBNM geforderten Zustandsänderungen an die kooperierenden TMs/TBNMs melden. Diese propagieren die Änderungen in „ihrem“ Netz und senden ihre Ergebnisagenda zurück, wo diese wiederum im lokalen TBN propagiert werden. Das Verfahren muß so lange iteriert werden, bis keine neuen demands mehr gemeldet werden. Die Restriktionen werden jeweils in *future state* kumuliert; so werden auch bei diesem Verfahren Inkonsistenzen entdeckt.

- DerTM weiß, wie eine Migration des TM mit dem TBNM als Komponente durchzuführen ist. Insbesondere muß er offene Agenden von migrierten Cliquen ebenfalls versenden, da diese Teil des TBN–Zustandes sind.

4.4 Schnittstelle zum Transaktions–Manager

Der TBN-Manager stellt Operationen zum Einfügen und Löschen von Beziehungen und Transaktionen in das Netz zur Verfügung, sowie Operationen, um die Auswirkungen von Zustands-Änderungen (`BOT`, `EOT`, `ABORT`) einer Transaktionen zu berechnen. Weiterhin können TBN–Cliquen ein- und ausgegeben werden. Hierfür wird festgestellt, mit welchen Transaktionen eine Transaktion in „nichtleeren“ Abhängigkeiten steht. Nur genau diese dürfen komplett migrieren.

Der TBN-Manager führt zustandsändernde Operationen auf Transaktionen nicht selbst durch, sondern meldet auf einer Agenda, was aus seiner Sicht jeweils erfüllt sein muß: die oben genannten Operationen leiten aus dem TBN und den TA-Zuständen eine Handlungsanweisung (TA-Agenda) ab, die enthält, welche Zustände diese und weitere Transaktionen erreichen müssen. Diese dient dem TM als Grundlage der Berechnung, welche zustandsändernden Operationen auf den Transaktionen durchzuführen sind, um die Operation konsistent durchführen zu können.

Wie behandelt der TM Agenden? Erst wenn eine Agenda komplett abgearbeitet ist, d.h. alle geforderten Zustände erreicht sind, darf der vorläufige Zustand des TBN zum aktuellen gemacht werden.

Bei Anfragen an den TM, die in einer offenen Agenda enthalten sind, ist kein Aufruf des TBNM erforderlich. Er wäre redundant, da die direkten und indirekten Auswirkungen dieser Anfragen bereits als Teilmenge in der zuvor berechneten (und jetzt offenen) Agenda enthalten sind.

Nach jeder Anfrage, deren Forderungen verzögert werden müssen, muß der TBNM auf den aktuellen Zustand zurückgesetzt werden. Der TBNM verwirft dann die vorläufigen Berechnungsergebnisse und vergißt den ursprünglichen Zustandsübergang. Der Grund für dieses Vorgehen ist, daß der Determinismus die in Kapitel 4.3.3 beschriebene Sonderbehandlung bedingt. Diese Sonderbehandlung benötigt ein Rücksetzen auf *current state*, da sonst aus Sicht des TBNM die Konsequenzen sowohl der ursprünglichen Abfrage als auch der (dazu inkompatiblen) Alternative von inkonsistenten Anfragen gleichzeitig verwirklicht werden sollen, was nur ein inkonsistentes TBN ergeben kann.

Ein Problem stellen mögliche Inkonsistenzen dar, wenn der TM für mehrere Anfragen gleichzeitig offene Agenden auf einer Clique hält. Diese Situation kann eintreten, wenn Anfragen an den TM gestellt werden, während Zustandsübergänge von einer vorhergehenden Anfrage verzögert werden. Die offenen Agenden können untereinander inkonsistent sein. Keine Parallelität, also höchstens eine Agenda, zuzulassen, wäre zwar korrekt, allerdings eine Einschränkung (siehe auch Kapitel 4.6.2). Wie stark sich diese Einschränkung auswirkt bzw. ob in der Praxis auf sie verzichtet werden kann, wäre noch zu untersuchen. Dies hängt auf jeden Fall von den verwendeten Transaktionsmodellen und der Größe der Cliquen ab.

4.5 Alternativen der Repräsentation von Beziehungen

Die der Implementierung zugrundeliegende Repräsentation der Beziehungen wurde unter mehreren Alternativen ausgewählt, die an dieser Stelle einander gegenübergestellt werden sollen.

4.5.1 Pfeilnotation

Der naheliegende Ansatz, die Beziehungsnamen direkt und eindeutig zu codieren, ergibt eine kompakte Darstellung, die allerdings einige Nachteile besitzt:

- Es läßt sich schlecht damit mit „rechnen". Bei der Addition (beispielsweise von $\leftrightarrow_c$ mit ${}_c{\mapsto}_b$), aber auch bei der Multiplikation entstehen Restriktionen zwischen Transaktionszuständen, die keiner der elementaren Beziehungen enstprechen: aggregierte Beziehungen sind i.a. nicht darstellbar.
- Eine Handlungsanweisung im konkreten Fall mit mehreren Beziehungen im TBN läßt sich nicht direkt ableiten.
- Inkonsistente Netze, z.B. bei zyklisch angeordneten Beziehungen zwischen mehreren Transaktionen sind nicht leicht zu entdecken.

4.5.2 Verknüpfung der TA-Automaten zu einem globalen Automaten

Die offensichtlich korrekte Methode, ausgehend von dem kombinierten Zustandsautomaten (Seite 13) den Potenzautomaten aller Transaktionen im System zu bilden und diesen den geltenden Beziehungen entsprechend einzuschränken, kommt aus naheliegenden Gründen nicht in Frage: Der Verwaltungsaufwand, dieses unhandliche Gebilde dynamisch zu verwalten, ist prohibitiv hoch:

- An Speicher wird bei n Transaktionen und einem (Einzel-) Transaktionsautomaten mit k Zuständen (siehe Abb. 2.1) Platz für n^k Zustände (bei $k = 6$ und bescheidenen $n = 50$ bereits über 10^{10}) und eine noch größere Anzahl von Übergängen benötigt.
- Bei jeder neu eingetragenen Beziehung müßten an diesem Automat größere Änderungen vorgenommen werden. Insbesondere wäre die Übergangsrelation δ mit dem beschriebenen aufwendigen Verfahren aus Kapitel 2 neu zu bestimmen.

4.5.3 Transaktions–Management–Prozeduren (TRAMPs)

TRAMPs sind selbst ihre Handlungsvorschrift, also ist eine Agenda direkt vorhanden. Kontrollflußbeziehungen stellen kein Problem dar: die geforderten Zustandsübergänge werden einfach direkt ausgeführt.

Problematisch wird es allerdings, wenn mehrere Beziehungen gelten und somit mehrere TRAMPs überlagert werden müssen:

1. Fall: Zwischen zwei Transaktionen gelten mehrere Beziehungen. Dann müssen die für die einzelnen Situationen und Zustandsübergänge spezifizierten Anweisungen aus beiden TRAMPs ausgeführt werden.

   Um unverträgliche Beziehungen zu erkennen, müssen widersprüchliche Anweisungen erkannt werden. Diese Aufgabe ist nicht trivial.

2. Fall: Es bestehen Beziehungen zwischen mehr als zwei Transaktionen. Dann stellen aber *Commit-Abhängigkeiten* ein Problem dar, bzw. allgemein Situationen, in denen Zustandsübergänge verzögert werden. In die TRAMPs ist das Warten bzw. das Auslösen der verzögerten Aktion für den Fall genau einer Beziehung codiert. Diese Aktion kann bei zusätzlichen Beziehungen falsch sein.

   Beispiel: Die Transaktionen sind bereits in den Zuständen `b(T`$_1$`)`, `p(T`$_2$`)`, `b(T`$_3$`)`, und das TBN hat folgende Gestalt:

   $$T_1 \rightarrow_c T_2 \rightarrow_c T_3$$

   Wird jetzt nach `EOT(T`$_1$`)` gefragt und die entsprechende TRAMP aufgerufen, wird T_1 nach *prepared* gebracht und anschließend zusammen mit T_2 erfolgreich beendet, obwohl das der Beziehung zu T_3 widerspricht.

   Gelöst werden könnte das Problem eventuell, indem TRAMPs der benachbarten Transaktionen rekursiv aufgerufen werden. Hierbei sind Vorkehrungen gegen zyklisches Warten und nicht abbrechende Rekursionen zu treffen.

   Widersprüchliche Anweisungen in den aufgerufenen TRAMPs sind ebenfalls schwierig zu entdecken.

4.5.4 Transaktions–Propagations–Matrizen

TPMs haben eine Reihe von Vorteilen:

- Alle vorkommenden Beziehungen, auch die abgeleiteten, sind explizit darstellbar. Aus dem TBN abgeleitete Forderungen sind daher nachvollziehbar. Dieser Vorteil wirkt sich nicht zuletzt bei der Fehlersuche im implementierten System positiv aus.

- Eine Handlungsanweisung (Agenda) ist leicht ableitbar.
- Inkonsistenzen im TBN können entdeckt werden.
- Die commit-Sonderbehandlung (siehe 4.3.3) ist auch im allgemeinen Fall mehrerer Transaktionen möglich.
- Kontrollflußbeziehungen, die die sofortige Ausführung von Zustandsübergängen fordern, sind darstellbar.
- Der Hauptvorteil ist, daß die Möglichkeit besteht, *alle* Auswirkungen und Aktionen erst zu berechnen und in einer Agenda festzuhalten, ohne sie gleich ausführen zu müssen. Falls zu einem späteren Zeitpunkt der Berechnung Inkonsistenzen auftreten, sind noch keine unwiderruflichen Seiteneffekte aufgetreten.

Sie besitzen allerdings einige Nachteile, insbesondere was die Effizienz angeht:

- Es ist eine spezielle Behandlung von Beziehungen einer Transaktion zu sich selbst nötig: Der Algorithmus `checkTPM()`, der die TPM intern normiert, muß für der Fall, daß eine TPM eine reflexive Beziehung repräsentiert, zusätzliche Restriktionen liefern, die im nicht reflexiven Gebrauch inkorrekt sind. Beispielsweise ist die Beziehung ${}_c{\longrightarrow}_a$ nur im reflexiven Gebrauch per se inkonsistent.
- Es werden vergleichsweise aufwendige Berechnungen bei allen Anfragen durchgeführt, um das TBN konsistent zu halten bzw. um die Agenda zu berechnen.
- Zur iterativen Berechnung der Einschränkungen müssen vollständige Cliquen von Transaktionen, die (indirekt) in Beziehung zueinander stehen, gehalten werden.

Aufgrund der Prioritäten bei der Implementierung des Prototypen, wo Durchschaubarkeit wichtiger als hohe Effizienz ist, und weil die Vorteile einfachere Algorithmen gegenüber den anderen Ansätzen versprachen, wurden der Implementierung die TPMs zugrundegelegt.

4.6 Umsetzung der Anforderungen an den TBN–Manager

Die zugrundeliegende Idee ist, die in Form von TPMs und deren Interpretationsvorschrift gegebene Repräsentation der kombinierten Zustandsautomaten von Paaren von Transaktionen zur Darstellung des TBN zu verwenden. Indirekte Restriktionen gehen in das TBN ein, indem der TBNM explizit alle aggregierten Beziehungen berechnet.

Der Propagationsalgorithmus (siehe Kapitel 4.7), der sämtliche Auswirkungen einer Zustandsänderung berechnet, kann sich daher auf vollständig gehaltene Cliquen von Transaktionen mit aggregierten Beziehungen stützen. Er hinterläßt nach jeder vollständigen (fehlerfreien) Berechnung wieder solche vollständigen Informationen.

4.6.1 Statische Korrektheit

Zulässige Cliquen sind die in Kapitel 2.7 definierten korrekten TBN. Da die aggregierten Beziehungen explizit in den TPM des Netzes realisiert sind, erkennt der Algorithmus inkonsistente Netze sofort am Entstehen inkonsistenter TPM beim Eintragen von Beziehungen. Über die Multiplikation werden daraufhin weitere TPMs in der Clique inkonsistent. Der TBNM berechnet, auf welche Transaktionen sich diese Inkonsistenzen fortpflanzen und setzt sie mit allen widersprüchlichen Forderungen auf die Ausgabeagenda.

4.6.1.1 Behandlung von Inkonsistenzen

Entsteht bei der Berechnung von aggregierten Beziehungen eine inkonsistente Beziehung, was der TBNM an resultierenden widersprüchlichen Zustands–Übergangsforderungen (demands) erkennt, liefert der TBNM dem TM eine Inkonsistenz–Meldung in Form eines Flags.

Da alle Berechnungen vorläufig und bedingt vorgenommen werden, hat der TM im Fall von vermeidbaren Inkonsistenzen die Möglichkeit, abzulehnen oder Beziehungen zu entfernen, um die Inkonsistenz aufzulösen. Er kann auf jeden Fall der Anwendung bzw. dem RM, der den fatalen Aufruf getätigt hat, melden, daß die Operation nicht möglich ist, und ihn darauf reagieren lassen. Der TBNM besitzt für diesen Fall keine interne Strategie. Das jeweilige Transaktionsmodell, das der TBNM nicht kennt, bestimmt, welche Maßnahmen der TM bzw. die betroffene Anwendung ergreifen kann.

Bei unvermeidbaren Inkonsistenzen sehe ich als allgemein anwendbare Reaktion nur die Möglichkeit, alle betroffenen Anwendungen (ConTracts) als Notfallbehandlung abzubrechen. Bei langen Anwendungen ist ein Abbruch eigentlich nicht akzeptabel — das war gerade einer der Gründe für den Entwurf des ConTract–Modells. Außerdem kann diese Reaktion zu einem inkonsistenten Stand der Daten der Anwendung führen, da durch einen Abbruch in dieser Form keine Kompensation für bereits beendete Steps des ConTracts vorgenommen wird. Andererseits treten unvermeidbare Inkonsistenzen nur dann auf, wenn das dem TM bzw. der Anwendung zugrundeliegende (nicht abgeschlossene) Transaktionsmodell an seine Grenzen gestoßen ist: Andernfalls läge eine vermeidbare Inkonsistenz vor.

4.6.2 Dynamische Korrektheit

Allgemein kann im dynamischen Ablauf der TBN–Verwaltung nicht garantiert werden, daß das TBN konsistent bleibt, da gefährliche aggregierte Beziehungen entstehen können.

Der TBNM verläßt sich natürlich darauf, daß der TM die Agenda korrekt abarbeitet und nicht beispielsweise die Restriktionen eines Zustandsüberganges, der nicht mit allen Konsequenzen durchgeführt wurde, als aktuellen Zustand übernimmt. So gesehen ist auch die zurückgelieferte Agenda einschließlich dem Stand ihrer Abarbeitung konzeptionell Teil des TBN.

Gerade das gegenteilige Problem tritt auf, wenn eine Beziehung gelöscht wird, ohne daß die Bedingungen für diese Operation aus Kapitel 2.6.4 erfüllt sind, insbesondere noch nicht alle Transaktionen der Clique einen Endzustand erreicht haben. Wird eine Beziehung gelöscht, entspricht das einem Informationsverlust: Restriktionen werden aufgehoben und noch offene Agenden werden teilweise ungültig. Werden diese Agenden allerdings komplett ignoriert, können Transaktionen, die verzögert wurden und bereits im Zustand *prepared* sind, „verwaisen“: Die Information, daß sie beendet werden wollten, steckte u.U. nur in der Agenda.

Beispiel: In der Situation ($T_1 \rightarrow_c T_2$) mit `p(T`$_1$`)` und `b(T`$_2$`)` war offensichtlich eine Anfrage `EOT(T`$_1$`)` vorausgegangen, die der TBNM mit der Agenda `(c(T`$_1$`), c(T`$_2$`))` beantwortet hat. Wird jetzt die Commit–Abhängigkeit gelöscht und die Agenda verworfen, sorgt keine Instanz mehr für das Commit von T_1.

Eine weitere „Belastung“ der dynamischen Konsistenzerhaltung stellen mehrere parallele Anfragen dar. Was ist das korrekte Verhalten von TBNM und TM, wenn nach einer Anfrage, die verzögert wurde, und deren Agenda noch nicht vollständig abgearbeitet, also offen, ist, z.B. der CM einen weiteren Wunsch an den TM heranträgt, der die selbe Clique betrifft?

Setzt der TM den TBNM direkt nach jeder Berechnung auf *current state* zurück und läßt er die Restriktionen jeweils unabhängig berechnen, wird eine Inkompatibilität zwischen diesen Anfragen nicht erkannt, und somit eine drohende Inkonsistenz übersehen. Die Restriktionen der Anfragen müssen also kumuliert werden, wenn die TBN–Clique nicht zur Synchronisation bis zum Ende der Agenda–Abarbeitung (d.h. während eines Aufrufs und seiner Realisierung) vollständig gesperrt bleiben soll. Operationalisiert heißt das im deterministischen Ansatz, daß die Auswirkungen von alten Anfragen mit offenen Agenden zusammen mit denen der hinzugekommenen Anfrage erneut (in *future state*) berechnet werden müssen. Die als Antwort vom TBNM gelieferte Agenda enthält dann die Forderungen, die sich aus der Kombination der Anfrage ergeben; dies sind i.a. mehr als die Vereinigung der Forderungen der Einzelanfragen. Meldet der TBNM „inkonsistent“, dann sind die Anfragen nicht miteinander verträglich.

4.6.3 Garantierbare Eigenschaften von speziellen Netztypen

Sofern

1. keine gefährlichen Beziehungen verwendet und
2. Beziehungen nur in Cliquen eingetragen werden, die ausschließlich Transaktionen im Zustand *installed* enthalten,

dann ist das TBN auch nach der strengeren Definition korrekt, d.h. in allen auftretenden Situationen bleibt die Konsistenz des Netzes erhalten.

4.6.4 Atomizität

Ergebnisse (Agenden) werden erst nach der Berechnung aller Auswirkungen geliefert: unvollständige Zwischenzustände sind nicht sichtbar. Die Schnittstelle zum TM ist so gestaltet, daß jeweils mehrere Zustandsübergangswünsche bzw. Beziehungen in einem Funktionsaufruf übergeben werden können. Diese werden dann als eine Einheit behandelt. Als Einschränkung muß beachtet werden, daß alle betroffenen Transaktionen in *einer* (im Fall der neuen Beziehungen resultierenden) Clique enthalten sind.

4.7 Algorithmus zur Propagation von TA-Zustandsänderungen in TBN

4.7.1 Eingabe

- Eine Liste, die Beziehungen enthält, an denen eine Änderung eingetreten ist.

  Diese Agenda wird von der Funktion `enterDependencies()`, die Beziehungen einträgt, und von der Funktion `enterTAstate()`, die Zustandsübergänge meldet, entsprechend initialisiert (siehe Kapitel 4.8.1 und 4.8.3).

- Ein vollständiges und konsistentes TBN.

  Ein nichtleeres TBN setzt sich aus einer oder mehreren *Beziehungs-Cliquen* zusammen. Eine *Beziehungs-Clique* oder kurz *Clique* ist ein (durch Hüllenbildung entstandenes und zusammenhängendes) Teilnetz in einem TBN, für das garantiert ist: Zu keiner TA, die in der Clique enthalten ist, besteht eine (nichtleere) Abhängigkeit zu einer Transaktion außerhalb der Clique. Eine *minimale Clique* bei einer gegebenen Menge von Beziehungen ist eine Beziehungs-Clique mit minimaler Anzahl von Transaktionen.

Die Funktion `propagateTBN()` arbeitet immer auf einer Kopie des TBN (im sog. *future state* des Netzes). Es kann also abhängig vom Ergebnis auf den vorherigen Zustand (*current state*), der als „before image“ gespeichert bleibt, zurückgesetzt oder das berechnete Netz zum gültigen gemacht werden.

4.7.2 Ausgabe

- Eine Transaktionsagenda, die die vom TBNM geforderten Zustände von Transaktionen enthält.

  Das TBN bleibt nur konsistent, wenn diese Transaktionen den hier aufgeführten Zustand erreichen. Falls dies nicht möglich ist, muß der TM den auslösenden Zustandsübergang bzw. das Eintragen der Beziehung verhindern (und das Netz auf current state zurückgesetzen).

- Ein Flag, das anzeigt, ob Operation die Konsistenz des Netzes erhalten hat.

4.7.3 Algorithmus `propagateTBN()`

Der Algorithmus propagiert Änderungen im TBN in Form von weiteren Zustandsforderungen für Transaktionen und zusätzlichen Einschränkungen von Beziehungen (das entspricht weiteren Einträgen in TPMs) so lange, bis ein Fixpunkt erreicht ist. Dazu geht er seine Eingabe–Liste von Beziehungen und ergänzt jeweils die noch zu bearbeitenden Beziehungen. Die Reihenfolge, in der die Beziehungen in die Liste einsortiert werden, ändert nichts am Ergebnis, da die Addition, die den TPMs weitere Restriktionen hinzufügt, assoziativ ist. Sie ist nur für die Bearbeitungsreihenfolge in der Clique verantwortlich.

Für jede Beziehung aus der Liste werden der Reihe nach alle Teilcliquen aus drei Transaktionen behandelt, die sich aus dieser Beziehung und den zwei Beziehungen zu einer weiteren Transaktion zusammensetzen:

- Er berechnet die aufgrund von ggfs. zusätzlichen Zustandsforderungen eventuell stärker eingeschränkten TPMs der hinzugefügten Beziehungen. Wurden sie (oder die Zustandsforderungen) modifiziert, dann setzt er diese Beziehung auf die Agenda, d.h. geht in einem weiteren Durchlauf von ihr aus.

- Ergaben sich im ersten Schritt keine weiteren Einschränkungen, dann werden die indirekten Beziehungen in dieser Mini–Clique neu berechnet und zur bisherigen aggregierten Beziehung addiert. Wenn diese Operation zu neuen Modifikationen an TPM oder Forderungen führt, dann setzt er die betroffene Beziehung für einen neuen Durchlauf auf die Agenda.

Auf diese Weise werden lokal neue Restriktionen berechnet. Durch das Ergänzen der Liste wird dafür gesorgt, daß sich diese Einschränkungen auf weitere Beziehungen auswirken.

Zuletzt sammelt er diejenigen Transaktionen, bei denen ein Zustand gefordert ist, der nicht dem aktuellen (oder einem vergangenen) Zustand der Transaktion entspricht. Diese Transaktionen werden mit den jeweiligen geforderten Zuständen auf der Ausgabe–Agenda des Algorithmus zusammengestellt, die anschließend dem TM übergeben wird.

```
wiederhole, solange Beziehungen auf der Liste der
                von Zustands"anderungen betroffenen Beziehungen sind:
    nimm die 'oberste' Beziehung (TA1 -->x TA2) aus der Agenda.
    wiederhole f"ur alle TAj, die in der Clique von TA1 stehen:
```

```
        wenn bei der internen Propagation
                    der Beziehung (TA1 -->y TAj)
                    der Zustand von TA1 bzw. TAj modifiziert
                    wird oder die TPM -->y, dann
            setze (TA1 -->y TAj) auf die Liste
        sonst
            berechne die Beziehung (TA2 -->z TAj)
                        aufgrund der transitiven Wirkung
                        "uber TA1 neu:
            wenn der alte Wert (TA2 -->z TAj) ungleich
                        dem neuen Wert ((TA2 -->z TAj) +
                        ((TA2 -->x' TA1) * (TA1 -->y TAj))) ist oder
                        W"unsche nach TA-Zustands"anderungen
                        ge"aussert werden, dann
                setze (TA2 -->z TAj) auf die Liste.
    wiederhole (analog zu oben) f"ur alle TAj,
                die mit TA2 in einer Clique stehen:
        wenn bei der internen Propagation
                    der Beziehung (TA2 -->y TAj)
                    der Zustand von TA2 bzw. TAj modifiziert
                    wird oder die TPM -->y, dann
            setze (TA2 -->y TAj) auf die Liste
        sonst
            berechne die Beziehung (TA1 -->z TAj)
                        aufgrund der transitiven Wirkung
                        "uber TA2 neu:
            wenn der alte Wert (TA1 -->z TAj) ungleich
                        dem neuen Wert ((TA1 -->z TAj) +
                        ((TA1 -->x TA2) * (TA2 -->y TAj))) ist oder
                        W"unsche nach TA-Zustands"anderungen
                        ge"aussert werden, dann
                setze (TA1 -->z TAj) auf die Liste.
"uberpr"ufe f"ur alle TAi der Clique:
    wenn sich der geforderte TA-Zustand vom aktuellen Zustand
                unterscheidet, dann
        setze die TAi mit der 'Zustandsdifferenz' als demand
                    auf die TA-Agenda.
    wenn f"ur mindestens eine der TAi widerspr"uchliche demands
                bestehen, dann
        melde 'inkonsistent'.
```

4.7.4 Anmerkungen zum Algorithmus

Auch Änderungen, die vom *vermuteten* Ausgang einer zustandsändernden Operation (z.B. bei COMMIT) abhängen, müssen bei der Propagation berücksichtigt und in die Agenden eingetragen werden. Sonst werden gegebenenfalls weitere demands nicht entdeckt und die Agenda bleibt unvollständig, wodurch u.U. Inkonsistenzen vorläufig unerkannt bleiben. Im Fall eines COMMIT-Aufrufs bei verzögertem EOT würde somit eine inkorrekte Agenda berechnet und abgearbeitet werden. Aus diesem Grund finden sämtliche Berechnungen auf einer Kopie des aktuellen Beziehungsnetzes statt (im sog. *future state*).

Der Algorithmus ist endlich unter der Voraussetzung, daß sich weder die aktuellen Beziehungen noch die gültigen Transaktionszustände ändern, während er eine Berechnung durchführt: In jedem Schritt können durch die Operationen $\oplus$ und $\odot$ nur weitere Einschränkungen in die Matrizen aufgenommen werden. Es werden keine Restriktionen aufgehoben, also gibt es einen „Fixpunkt" der Iteration.

Die Anzahl m der Zustände, die eine TPMatrix annehmen kann, ist endlich, da es nur endlich viele unterschiedliche konsistente TPMs gibt. Meine Abschätzung stützt sich auf die maximal eingeschränkten TPMs aus Anhang B. Aus ihnen können weitere TPMs erzeugt werden, die gleich viel oder weniger Restriktionen enthalten. Das sind für jeden „i"–Eintrag 7 und für jeden „+"– bzw. „−"–Eintrag 3 Möglichkeiten. Abschätzung: mit

$$
\begin{aligned}
i &= 7 \text{ Anzahl der schwächeren Einträge von } i \\
p &= 3 \text{ Anzahl der schwächeren Einträge von } +, - \\
t_1 &= 8 \text{ Anzahl der TPMs des ersten Verteilungstyps} \\
t_2 &= 5 \text{ Anzahl der TPMs des zweiten Verteilungstyps} \\
rc &> 2 \text{ Faktor durch begin-commit-Restriktion} \\
ra &> 2 \text{ Faktor durch begin-abort-Restriktion} \\
ri &> 2 \text{ Faktor durch Restriktion zw. TPM(vor) und TPM(zurueck)} \\
d &> 5 \text{ Faktor durch doppelt erzeugte TPMs}
\end{aligned}
$$

ergibt sich

$$m = ((p^6 * i^{10}) * t_1 + (p^8 * i^8) * t_2)/(rc * ra * ri * d) < 5 * 10^{10}$$

als obere Grenze. Die Anzahl der Schleifendurchläufe ist somit begrenzt.

4.8 Basisoperationen

4.8.1 Beziehungen eintragen

Der Algorithmus `enterDependencies()`, trägt eine Menge von Beziehungen $T_1 \rightarrow T_2$ in das TBN ein. Dazu muß aus mehreren Cliquen eine Clique gebildet werden, wenn die betroffenen Transaktionen aus unterschiedlichen Cliquen stammen. Anschließend werden die Beziehungen auf die Eingabeliste des Propagationsalgorithmus gesetzt; dieser erledigt den Rest.

```
wiederhole f"ur alle einzutragenen Beziehungen (TA1 -->x TA2):
    wenn sich beide TAs in bisher unabh"angigen
                Teilnetzen (Cliquen) befanden, dann
        verschmelze diese zu einer Clique.
    addiere (TA1 -->x TA2) zur bestehenden Beziehung
                zwischen den beiden TAs.
    wenn dabei Beziehung oder TA-Zust"ande modifiziert
                wurden (checkTPM()), dann
        setze die resultierende Beziehung auf die Eingabeagenda
                    des Propagationsalgorithmus.
```

4.8.2 Beziehungen austragen

Wenn Beziehungen aus dem TBN entfernt werden, werden die Cliquen der betroffenen Transaktionen vollständig aus dem TBN entfernt und alle nicht entfernten ursprünglich eingetragenen Beziehungen erneut eingefügt. Dabei können mehr und andere Cliquen als vorher entstehen. Diese Operation verändert daher auch den *current state* dieser Cliquen. Sämtliche aggregierten TPM in ihnen werden von Grund auf neu berechnet. Dieser Aufwand ist wegen der fehlenden Umkehrfunktion zur Addition $\oplus$ erforderlich, da die zu entfernenden Beziehungen nicht „herausgerechnet" werden können.

Offene Anfragen werden obsolet (siehe auch das Problem in Kapitel 4.6.2).

4.8.3 TA-Zustandsübergangswünsche

Der Algorithmus `enterTAstate()` berechnet die Konsequenzen daraus, daß die Transaktionen T_i eine zustandsändernde Operation durchführen wollen: Der neue Zustand wird jeweils als *demand* vermerkt und alle Beziehungen zu anderen Transaktionen in ihrer Clique werden als Kandidaten für weitere Restriktionen auf die Eingabeliste des Propagationsalgorithmus gesetzt.

```
wiederhole f"ur alle TAi:
    "uberpr"ufe f"ur alle TA2, die in einer Beziehung -->x
                zu TAi stehen:
         wenn der Zustand von TA2 oder die Beziehung (TA2 -->x TAi)
                     mittels checkTPM() modifiziert wurden, dann
             setze die Beziehung -->x auf die Eingabeagenda
                         des Propagationsalgorithmus.
```

4.8.4 Netz migrieren

Der TM mit seinem TBNM muß im ConTract–Modell wie alle anderen Komponenten auch die „Rechnerplattform" wechseln, also migrieren können.

Je nach Implementierung von verteilten TBNM kann zugelassen werden, daß Cliquen auf mehrere TBNM verteilt sind. Es kann aber auch die Beschränkung sinnvoll sein, z.B. um Nachrichten einzusparen, zu fordern, daß Cliquen jeweils vollständig in einem (lokalen) TBN gehalten werden. Dann ist es vernünftig, diese Clique jeweils an den aktuellen „*hot spot*" zu verschieben.

Aus diesen Gründen muß ein TBN bzw. eine TBN–Clique migrieren können. Was heißt migrieren? Sämtliche Beziehungen einer Clique werden aus dem einen TBN in einer atomaren Operation entfernt und externalisiert. Auf der Gegenseite werden sie — wiederum atomar — in ein TBN eingetragen. Die Funktionalität der zweiten Phase stellt bereits die Funktion zum Eintragen von Beziehungen zur Verfügung.

Es werden nur die ursprünglich eingetragenen Beziehungen herausgegeben, um die Nachrichtenlänge zu minimieren. Der TBNM auf der Empfängerseite führt eine Propagation durch, um die aktuellen Transaktionszustände wieder einzuarbeiten, sowie eventuell bereits lokal vorhandene Beziehungen zu berücksichtigen.

4.8.5 Verwerfen einer Operation

Sämtliche Berechnungsergebnis müssen vergessen werden, wenn eine auslösende Operation (Zustandsänderung oder Beziehung eintragen) verworfen oder verzögert wird. Zu diesem Zweck ruft der TM die Funktion `current2future()` auf, die die Zustandsforderungen aller Transaktionen einer Clique mit ihrem Zustand initialisiert und alle TPMs aus *future state* mit denen aus *current state* überschreibt.

4.8.6 TBN aktualisieren

Der TBNM übernimmt nur auf explizite Anforderung in Form eines Aufrufs der Funktion `future2current()` die vorläufigen Modifikationen des TBN aus *future state*, konkret die stärker restringierten TPM, als das momentan gültige TBN (*current state*). Diese Funktion wirkt gezielt auf eine Clique, um in einem TBN unter Garantie unabhängige Transaktionen auch unabhängig verwalten zu können.

Die letzten Funktionen bekommen einen Transaktionsnamen als Parameter, um die betroffene Clique zu identfizieren.

4.8.7 Beispielablauf

Zur Illustration der Interaktion zwischen TM und TBNM betrachten wir die Bearbeitung einer Clique von drei Transaktionen mit gegenseitiger Commit–Abhängigkeit (siehe Abb. 2.9).

1. Der TM installiert die Transaktionen und vergibt an sie die GTRIDs T_1, T_2 und T_3.

2. Es werden zwei Beziehungen eingetragen, indem der TM eine Beziehungsliste aufgebaut und mit folgendem Aufruf des TBNM die Beziehungen vorläufig einträgt: `enterDependencies(((T`$_1$ $\leftrightarrow_c$ `T`$_2$`), (T`$_1$ $\leftrightarrow_c$ `T`$_3$`)))`. Da die Transaktionen bisher nicht in Beziehungen zu irgendwelchen Transaktionen standen (auch nicht untereinander), entsteht eine Clique aus genau diesen Transaktionen. Die zuvor leeren Beziehungen zwischen T_1 und T_2 bzw. T_3 werden durch die Addition der neu eingetragenen Beziehungen modifiziert, also berechnet der TBNM in *future state* mit dem Aufruf `propagateTBN(((T`$_1$ $\leftrightarrow_c$ `T`$_2$`), (T`$_1$ $\leftrightarrow_c$ `T`$_3$`)))` die vollständige Clique mit den indirekten Beziehungen, trägt also auch die errechnete aggregierte Beziehung $(T_2 \leftrightarrow_c T_3)$ ein.

Weil sich diese Beziehung von der zuvor leeren Beziehung zwischen T_2 und T_3 unterscheidet, macht `propagateTBN()` einen weiteren Schleifendurchlauf, in dem zu den anderen beiden Beziehungen die jeweiligen indirekten Auswirkungen addiert werden. Dabei ergeben sich keine Änderungen (insbesondere entstehen auch keine Zustandsforderungen) und der Fixpunkt ist erreicht.

3. Die Rückmeludung des TBNM ist „konsistent“ und die Agenda leer, daher kann der TM die Beziehungen endgültig eintragen, indem er dem TBNM die Operation bestätigt: Er ruft `future2current(T`$_1$`)` auf, sodaß das TBN auch in *current state* die oben erwähnten drei Beziehungen enthält.

4. Die erste Transaktion will starten, also ruft der TM `enterTAstate((T`$_1$`, dem-run))` auf. Dies ist eine Zustandsänderung, die alle Beziehungen betrifft, in denen T_1 steht: `propagateTBM(((T`$_1$ $\leftrightarrow_c$ `T`$_2$`), (T`$_1$ $\leftrightarrow_c$ `T`$_3$`)))` modifiziert die Beziehungen von T_1 durch den Aufruf von `checkTPM()` derart, daß für T_1 der Zustand *demand–not–run* nicht mehr gefordert werden darf, indem in $TPM(zurueck)$ der von T_1 ausgehenden Beziehungen zu allen Einträgen in der Spalte „begin“ ein „+“ und in der Spalte „demand–not–run“ ein „−“ addiert wird und diese

Einträge in der TPM propagiert werden. Anschließend sehen die beiden Beziehungen wie folgt aus:

(T_1

T_1 \ $T_{2/3}$	beg	com	abo	dnr
beg	$*$	$*$	$*$	$*$
com	$+$	$+$	$-$	$-$
abo	$*$	$-$	$+$	$*$
dnr	i	i	i	i

$T_{2/3}$ \ T_1	beg	com	abo	dnr
beg	$+$	$*$	$*$	$-$
com	$+$	$+$	$-$	$-$
abo	$+$	$-$	$+$	$-$
dnr	$+$	$-$	$+$	$-$

$T_{2/3}$)

`propagateTBM()` berechnet allerdings keine weiteren Konsequenzen für die andereren Transaktionen, da in der Zeile „begin“ von $TPM(vor)$ der von T_1 ausgehenden Beziehungen keine Restriktionen enthalten sind, und der TBNM liefert auf der Agenda genau dieses als Forderung: `(T`$_1$`, dem-run)`. Der TM startet T_1. Diese Operation sei erfolgreich, also ruft der TM wieder `future2current(T`$_1$`)` auf, um die Konsequenzen des Zustandsüberganges in den aktuellen Netzzustand zu übernehmen.

5. T_1 möchte sich beenden und meldet dem TM den Wunsch `EOT(T`$_1$`.` Der TM fragt an: `enterTAstate((T`$_1$`, dem-commit))`. `propagateTBN()` modifiziert (in *future state*) alle drei TPMs, bis sie folgende Gestalt haben:

(T_x

T_x \ T_y	beg	com	abo	dnr
beg	$+$	$+$	$-$	$-$
com	$+$	$+$	$-$	$-$
abo	i	i	i	i
dnr	i	i	i	i

T_y \ T_x	beg	com	abo	dnr
beg	$+$	$+$	$-$	$-$
com	$+$	$+$	$-$	$-$
abo	i	i	i	i
dnr	i	i	i	i

T_y)

Alle drei Transaktionen können nur noch den Endzustand *committed* erreichen, wenn das TBN in *future state* zugrunde gelegt wird. Der TBNM liefert als Agenda `((T`$_1$`, dem-commit), (T`$_2$`, dem-run und dem-commit), (T`$_3$`, dem-run und dem-commit))`. Der TM kann diese Agenda nicht vollständig abarbeiten. Daher ruft der TM gerade noch `PREPARE(T`$_1$`)` auf und setzt den TBN-Zustand zurück: `current2future(T`$_1$`)`, wodurch aus Sicht des TBNM alle Transaktionen noch aktiv sind.

6. T_2 soll gestartet werden. Der TM stellt fest, daß dies eine Aktion ist, die der TBNM auf der offenen Agenda gefordert hat, und führt direkt `BOT(T`$_2$`)` aus, ohne eine Funktion des TBNM aufzurufen.

7. T_2 startet nach dem bereits bekannten Schema, bricht dann aber ab. Der TM ruft somit, nachdem er den Widerspruch zur offenen Agenda entdeckt hat, `enterTAstate((T`$_2$`, dem-abort))` auf. `propagateTBN()` berechnet (in *future state*) die unter dieser Voraussetzung geltenden TPMs, wie z.B.:

(T_2

T_2 \ T_3	beg	com	abo	dnr
beg	$*$	$-$	$+$	$*$
com	i	i	i	i
abo	$*$	$-$	$+$	$*$
dnr	i	i	i	i

T_3 \ T_2	beg	com	abo	dnr
beg	$+$	$-$	$+$	$-$
com	i	i	i	i
abo	$+$	$-$	$+$	$-$
dnr	$+$	$-$	$+$	$-$

T_3)

Der TBNM meldet auf der Ausgabeagenda: `((T`$_1$`, dem-abort), (T`$_2$`, dem-abort), (T`$_3$`, dem-abort))`. Der TM kann diese Agenda sofort ausführen: er ruft `ABORT(T`$_1$`)`, `ABORT(T`$_2$`)` und `ABORT(T`$_3$`)` auf und setzt den aktuellen Netzzustand neu: `future2current(T`$_2$`)`.

8. Da dieses Beispiel allmählich zu Ende ist, und keine neuen Beziehungen mehr eingetragen werden, kann der TM alle (ursprünglich eingetragenen) Beziehungen aus dem TBN entfernen: `deleteDependencies(((`$\mathtt{T}_1 \leftrightarrow_c \mathtt{T}_2$`), (`$\mathtt{T}_1 \leftrightarrow_c \mathtt{T}_3$`)))`. Anschließend enthält das Netz weder in *current state* noch in *future state* irgendwelche Beziehungen, insbesondere nicht die aggregierte Abhängigkeit ($T_2 \leftrightarrow_c T_3$).

Kapitel 5

Implementierung

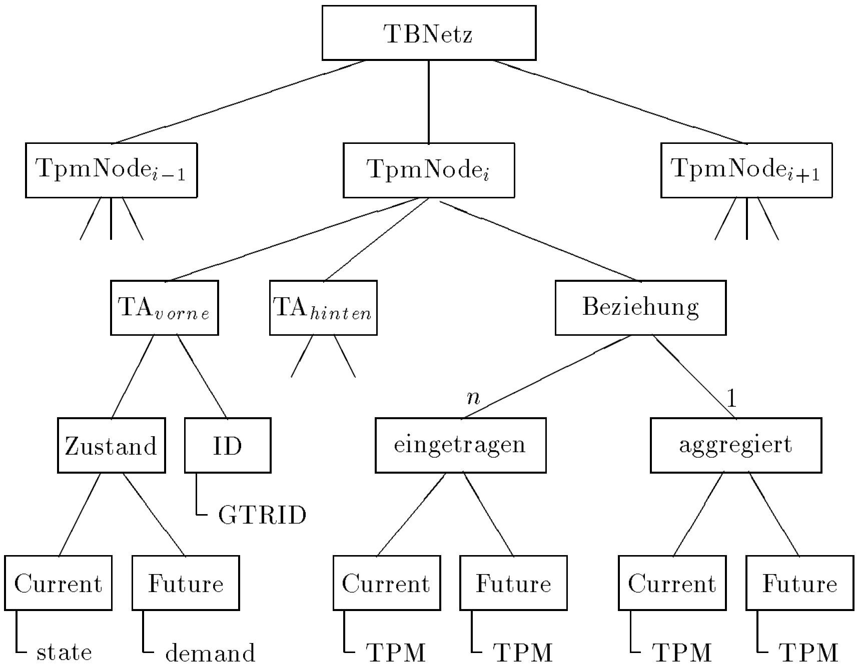

Abbildung 5.1: Aufbau des TBN

Aus konzeptioneller Sicht besteht ein TBN aus einer Menge von Beziehungen (`TpmNode`). Ein Beziehungsknoten repräsentiert die Beziehung $\texttt{TA}_{vorne} \rightarrow_{dependency} \texttt{TA}_{hinten}$. Jede Transaktion besitzt einen Namen **`GTRID`** und einen Zustand. Bei der Abhängigkeit zwischen zwei Transaktionen muß unterschieden werden zwischen den n **`ursprünglich eingetragenen`** Beziehungen und der einen **`aggregierten`**, d.h. abgeleiteten Beziehung. Jede Beziehung ist als Matrix **`Tpm`** repräsentiert.

Für alle Datenstrukturen, die sich dynamisch ändern, wird eine Version für den aktuell gültigen Stand (**`current state`**) gehalten, sowie eine Kopie, auf der Berechnungen stattfinden (**`future`**

state). Die Werte aus **future state** können (müssen aber nicht) vom TM als neuer gültiger Stand eingetragen werden. Dieses Modell ist in Abb. 5.1 dargestellt.

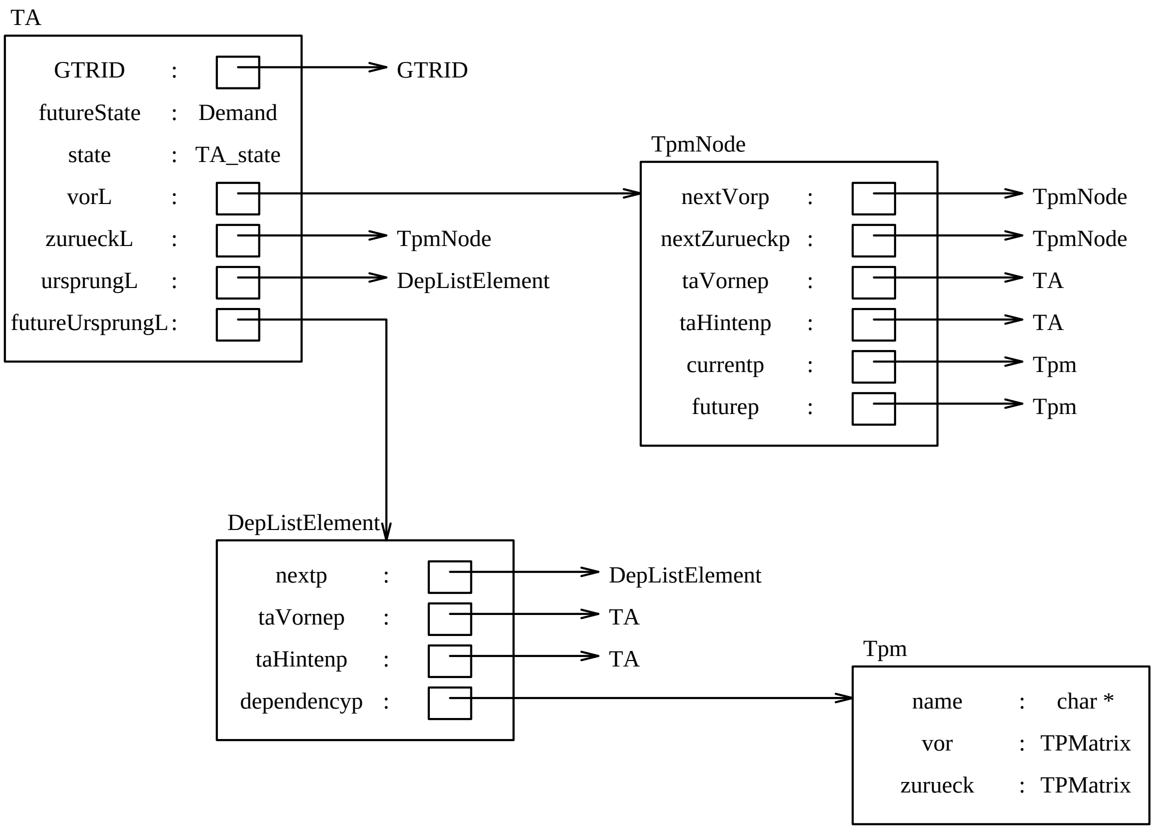

Abbildung 5.2: Datenstruktur `TA`

Die Implementierung des TBN kann man sich als eine Adjazenzmatrix vorstellen, in der die Matrixeinträge erweitert sind. Diese enthalten nicht nur die binäre Information *Kante vorhanden – ja oder nein*, sondern auch Informationen über die Kante. Die Kanten sind gerichtet und mit der aggregiertten TPM benannt. Zwischen zwei Transaktionen kann es mehrere ursprünglich eingetragene Beziehungen geben, deren Auswirkungen in die aggregierte TPM eingehen.

Da zu jeder Beziehung eine inverse Beziehung (mit entsprechender TPM) existiert, können die entgegengesetzt gerichteten Beziehungen zwischen zwei Transaktionen in der aggregierten TPM zusammengefaßt werden. Somit reduziert sich die Adjazenzmatrix auf eine Dreiecksmatrix.

Weil nicht alle Transaktionen im System miteinander in Beziehungen stehen, sondern nur z.B. jeweils die Transaktionen der Steps aus einem ConTract, wird die „transitive" Hülle über den eingetragenen Beziehungen gebildet und für die entstandenen Cliquen von Transaktionen jeweils eine Adjazenzmatrix angelegt.

Implementiert sind die Daten so, daß die Algorithmen einen direkten Zugriff ohne Suchverfahren auf die jeweils benötigten Informationen haben:

Der „Einstieg" in das TBN erfolgt immer über die Struktur `TA` (Bild 5.2). Die Transaktionen werden dem TBNM vom TM zur Verfügung gestellt und sind aus seiner Sicht immer vorhanden. Transaktionen besitzen einen Zeiger `ursprungL` auf eine verkettete Liste von ursprunglich eingetragenen Beziehungen (`DepListElement`), bei denen die Transaktion „links vom Pfeil" (als TA_{vorne}) steht. `futureUrsprungL` enthält diejenigen Beziehungen, die nur vorläufig (in *future state*) eingetragen sind.

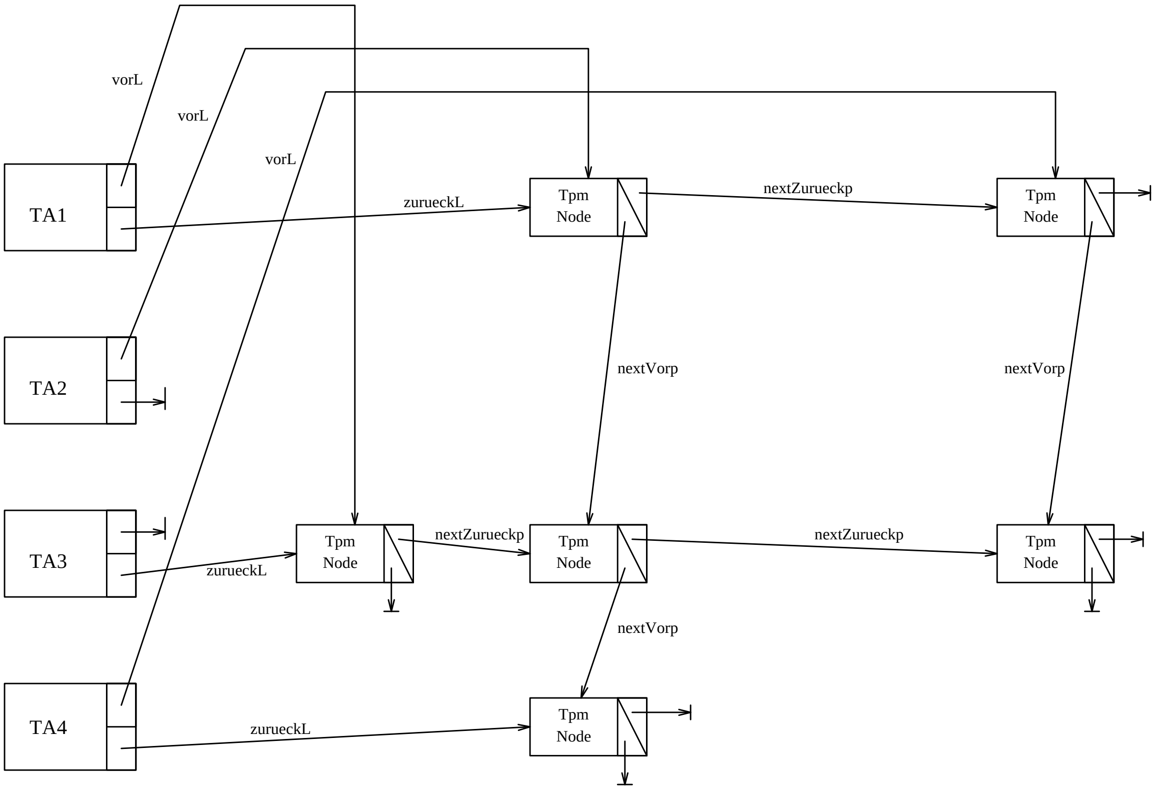

Abbildung 5.3: Verkettung in einem TBN

Von jeder Transaktion ausgehend (`vorL`) sind alle Beziehungsknoten der aggregierten Beziehungen miteinander verkettet, bei denen die Transaktion als TA_{vorne} eingetragen ist. Die Verkettung zwischen den TpmNodes erfolgt über `nextVorp`. Analog sind über `zurueckL` und `nextZurueckp` alle Beziehungen verkettet, auf deren rechter Seite die Transaktion steht. Dies ist exemplarisch in Bild 5.3 dargestellt.

In dieses Beispiel–Netz wurden Beziehungen $TA_1 \rightarrow_c TA_3$, $TA_{2c} \rightarrow_a TA_1$ und $TA_4 \leftrightarrow_c TA_1$ eingetragen. Die restlichen Beziehungen sind bei der Hüllenbildung zur vollständigen Clique entstanden. Die „Richtung" dieser automatisch erzeugten Beziehungen ist willkürlich.

Jeder `TpmNode` verweist (redundant) auf die beiden Transaktionen der Beziehung. Neben den Verkettungszeigern gibt es Zeiger auf die TPMs in den beiden Zuständen. Die ursprünglich eingetragenen Beziehungen sind – wie oben erwähnt – hier allerdings nicht zugeordnet.

Als Ein- und Ausgabeparameter für die Funktionen des TBN werden folgende Daten verwendet:

Transaktionszustände mit den Konstanten `INSTALLED`, `RUNNING`, `FINISHED`, `PREPARED`,

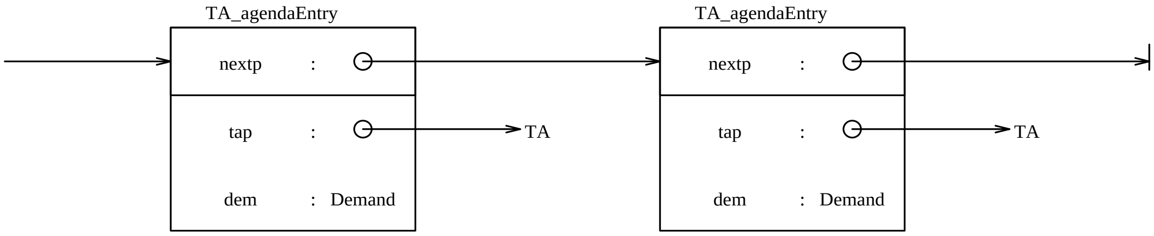

Abbildung 5.4: Beispiel-Datenstruktur `TA_agenda`

`COMMITTED`, `ABORTED`, `DEMAND_NOT_RUN`,

demands mit den konstanten Werten `DEM_RUN`, `DEM_COM`, `DEM_ABO`, `DEM_NR`,

Beziehungsnamen , das sind Zeiger auf Strukturen vom Typ `Tpm`, mit den konstanten Werten `DCLC` (für $\rightarrow_c$), `DCLLC` (für $\leftrightarrow_c$), `DAMB` (für ${}_a{\mapsto}_b$), usw.

Eine kurze Erläuterung zur Mnemonik dieser Konstanten: `D` steht für *demand*. Dann folgt ein Buchstabe für den linken Index am Beziehungspfeil. `L` bezeichnet eine logische „Implikation“, `LL` eine logische „Äquivalenz“. `M` bezeichnet eine „einseitige“ Kontrollflußbeziehung, `MM` eine beidseitige. Als Abschluß steht ein Buchstabe für den rechten Index am Beziehungspfeil.

Die meisten Funktionen haben als Parameter Beziehungslisten (`DepList`) oder Transaktions-Agenden (`TA_agenda`), um in einem Aufruf mehrere Argumente verarbeiten zu können. Wenn beispielsweise mehrere Beziehungen gleichzeitig eingetragen werden, findet *eine* Berechnung der Auswirkungen aller dieser Beziehungen statt (dieses Ergebnis wird auch in einer Agenda zurückgegeben), was effizienter als getrennte Berechnungen ist.

Kapitel 6

Offene Fragen und Ausblick auf zukünftige Arbeiten

6.1 Cliquen und Parallelität

Die Netzverwaltung sollte so erweitert werden, daß auf einer Menge von Transaktionen mehrere Berechnungen parallel stattfinden können.

Beispielsweise sollen mehrere Zustandsübergänge von Transaktionen stattfinden und gleichzeitig eine Beziehung eingetragen werden. Dabei dürfen, falls die Bezeihung abgelehnt wird, weil sie das Netz inkonsistent machen würde, die Zustandsübergänge nicht beeinflußt werden.

Welche Eigenschaften muß eine Clique besitzen, damit korrekte Ergebnisse geliefert werden, wenn gleichzeitig Zustandsübergänge vorgenommen und die Konsistenz einer neuen Beziehung geprüft wird?

Welche Operationen auf TBNs müssen allgemein oder unter bestimmten Voraussetzungen synchronisiert werden, da sie auf keinen Fall parallel ausgeführt werden dürfen?

6.2 Optimierung der Vernküpfungsoperationen

Nach meiner Vermutung können aus den elementaren Beziehungen höchstens wenige hundert unterschiedliche TPMs statt der in Kapitel 4.7.4 berechneten Anzahl entstehen. Wenn für diese die aufwendige Additionsoperation und Multiplikation einschließlich der Normierung durch z.B. einen Tabellenzugriff ersetzt werden könnte, würde das die Propagation wesentlich beschleunigen.

6.3 Optimierung durch algebraische Eigenschaften

Wie lassen sich die Eigenschaften und Rechenregeln der Beziehungen aus Kapitel 2.3 und 2.3.4.3 zur Optimierung des Algorithmus in Kapitel 4.7 verwenden?

Unter welchen Voraussetzungen läßt sich ein ganzes Teilnetz auf eine (eventuell virtuellen) TBN-Knoten reduzieren, was die Auswirkungen auf die Umgebung angeht? Sichere Kandidaten dafür sind Cliquen aus Transaktionen, die untereinander nur in $\leftrightarrow_c$ - Bezeihung stehen.

Wann muß eine solche virtuelle Transaktion expandiert und ihre Binnenstruktur beachtet werden, d.h. woran erkennt man, daß Zustandsänderungen der enthaltenen Transaktionen gefordert werden?

6.4 Löschen von Beziehungen aus dem Netz

Auf der Abstraktionsebene des TBNM kann nicht entschieden werden, wann das Löschen von Beziehungen möglich ist. Implementierte Transaktionsmodelle und das Löschen von Informationen aus dem TBN könnten in ein Konzept gebracht werden. Welche Systemkomponenten (z.B. Anwendungsprogramme oder TM) implementieren die im System eventuell gleichzeitig vorhandenen und über das TBN kooperierenden Transaktionsmodelle? Genau diese Komponenten wären autorisiert, „ihre" Beziehungen zu löschen. Auf welche Bedingungen und Trigger (eventuell mehrere) muß dabei gewartet werden, damit Beziehungen nicht gelöscht werden, solange sie noch indirekte Auswirkungen haben (siehe Kapitel 2.6.4)?

Anhang A

TRAMPs der Commit-Abhängigkeit ($\rightarrow_c$)

Zur Erläuterung: `i-dr` steht für ***demand-run*** *im Zustand* ***installed***; die anderen Kombinationen aus Zustand und einem oder mehreren demands sind analog zu lesen.

1 (BOT(T_1), $T_1 \rightarrow_c T_2$):

```
switch (t1.state) {

case i:
   run(t1);
   switch (t2.state) {
   case dnr, a:
      abort(t1);
      purge(dep);
   case else:
      /* keine Aktion */
   }

case i-dc, i-dr-dc:
   run(t1);
   switch (t2.state) {
   case dnr:
      fatal(demand-not-run-demand-commit-Konflikt);
   case a:
      fatal(abort-demand-commit-Konflikt);
   case else:
      /* keine Aktion */
   }

case i-dr-da:
   run(t1);
   abort(t1);
   purge(dep);
```

```
case else:
   /* wie bei einer isolierten Transaktion */
}
```

2 (BOT(T_2), $T_1 \rightarrow_c T_2$):

```
switch (t2.state) {

case i-dr-da:
   run(t2);
   abort(t2);
   switch (t1.state) {
   case c:
      /* kommt nicht vor */
   case *-dc:
      fatal(aborted-demand-run-Konflikt);
   case a:
      /* keine Aktion */
   case else:
      t1.state |= da;
   }

case else:
   /* wie bei einer isolierten Transaktion */
}
```

4 (PREP(T_1), $T_1 \rightarrow_c T_2$):

```
switch (t1.state) {

case b, r:
   prep(t1);
   switch (t1.state) {
   case a:
      purge(dep);
   case else:
      switch (t2.state) {
      case a:
         abort(t1);
         purge(dep);
      case else:
         /* keine Aktion */
      }
   }
case b-dc, r-dc, p:
   /* siehe (7) */

case else:
```

```
      /* wie bei einer isolierten Transaktion */
   }
```

5 (PREP(T_2), $T_1 \rightarrow_c T_2$):

```
   switch (t2.state) {

   case i, dnr, i-dc, i-dr, i-dr-dc, i-dr-da, p, c, a:
      /* wie bei einer isolierten Transaktion */

   case b, r:
      prep(t2);
      switch (t2.state) {
      case a:
         switch (t1.state) {
         case i, i-dr:
            t1.state |= da;
            purge(dep);
         case dnr:
            purge(dep);
         case i-dc, i-dr-dc, b-dc, p:
            fatal(aborted-demand-commit-Konflikt);
         case b, p:
            abort(t1);
            purge(dep);
         case c:
            /* kommt nicht vor */
            fatal(committed-aborted-Konflikt);
         case a:
            purge(dep);
         }
      case else:
         /* keine Aktion */
      }

   case b-dc, p:
      /* siehe (8) */

   case b-da:
      /* siehe (10) */
   }
```

7 (COMMIT(T_1), $T_1 \rightarrow_c T_2$):

```
   switch (t1.state) {

   case b, b-dc:
      switch (t2.state) {
      case i, i-dc, i-dr, i-dr-dc, b-dc:
         prep(t1);
```

```
        switch (t1.state) {
        case a:
           purge(dep); /* nicht warten */
        case else:
           t1.state |= dc; /* auf Ende von T2 warten */
        }
     case dnr, a:
        /* siehe (9) */
     case p:
        commit(t1, t2); /* Ergebniszustand von T1 ist egal */
        purge(dep);
     case c:
        /* kommt nicht vor */
        commit(t1);
        purge(dep);
     }

  case p:
     switch (t2.state) {
     case i, i-dc, i-dr, i-dr-dc, b-dc:
        t1.state |= dc; /* auf Ende von T2 warten */
     case dnr, a:
        /* siehe (9) */
     case p:
        commit(t1, t2); /* Ergebniszustand von T1 ist egal,
                           da commit(T2) auf jeden Fall klappt */
        purge(dep);
     case c:
        /* kommt nicht vor */
        commit(t1);
        purge(dep);
     }

  case else:
     /* wie bei einer isolierten Transaktion */
  }
```

8 (COMMIT(T_2), $T_1 \rightarrow_c T_2$):

```
  switch (t2.state) {

  case b, b-dc, p:
     commit(t2);
     switch (t2.state) {
     case p:
        t2.state |= dc;
     case c:
        switch (t1.state) {
        case p:
           commit(t1);
           purge(dep);
```

```
        case p:
           commit(t1);
           purge(dep);
        case else:
           /* keine Aktion */
        }
        purge(dep);
     case a:
        switch (t1.state) {
        case dnr:
           purge(dep); /* siehe Kommentar oben */
        case i-dc, i-dr-dc, b-dc, r-dc, p-dc:
           fatal(aborted-demand-commit-Konflikt);
        case i, b, p:
           abort(t1);
           purge(dep);
        case c:
           /* kommt nicht vor */
           fatal(aborted-committed-Konflikt);
        case a:
           purge(dep);
        }
     case else:
        /* kommt nicht vor */
     }

  case else:
     /* wie bei einer isolierten Transaktion */
  }
```

9 (ABORT(T_1), $T_1 \rightarrow_c T_2$):

```
  switch (t1.state) {

  case b, r, p:
     abort(t1);
     purge(dep);

  case else:
     /* wie bei einer isolierten Transaktion */
  }
```

10 (ABORT(T_2), $T_1 \rightarrow_c T_2$):

```
  switch (t2.state) {

  case b, r, p:
     switch (t1.state) {
     case i, i-dr, i-dr-da:
        t1.state |= da;
```

```
      abort(t2);
      purge(dep);
   case dnr:
      abort(t2);
      purge(dep);
   case i-dr-dc, b-dc, p:
      fatal(aborted-demand-commit-Konflikt);
   case b, r, p:
      abort(t1, t2);
      purge(dep);
   case c:
      /* kommt nicht vor */
      fehler(aborted-committed-Konflikt);
   case a:
      abort(t2);
      purge(dep);
   }

case else:
   /* wie bei einer isolierten Transaktion */
}
```

Anhang B

Maximal eingeschränkte TPMs, die nicht im Widerspruch zum TA-Automaten stehen

Beide Transaktionen können in ihrem Zustandsautomaten einen Endzustand erreichen, wenn zwischen ihnen eine Beziehung besteht, die höchstens die Restriktionen aus genau einer der unten aufgezählten Matrizen enthält. Dabei ist vorausgesetzt, daß beide Transaktionen noch im Zustand *installed* sind.

Notiert wurde allein der logische Gesichtspunkt: In dieser Aufzählung wird als unerheblich angesehen, ob Kontrollflußbeziehungen zusätzlich die sofortige Ausführung von Übergängen fordern. Restriktionen durch temporale Beziehungen wurden nicht untersucht.

Jede hier aufgeführte Matrix entspricht einem Weg durch den Potenzautomaten von zwei Transaktionen, das heißt einer Abfolge bzw. Kombination von Zustandsübergängen. Der Inhalt von Einträgen der TPMs, die einer Kombination von Zuständen entsprechen, die nicht auf diesem Weg liegt, ist beliebig und darf daher „inkonsistent“ sein.

Jede zweite TPM in dieser Aufzählung entsteht durch Vertauschen von $TPM(vor)$ und $TPM(zurueck)$ aus der jeweils vorherigen TPM und stellt somit das Äquivalent zu dieser Beziehung „in Gegenrichtung“ dar.

c(T_1) und c(T_2):

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	+	i	i	−
com	+	+	−	−
abo	i	i	i	i
dnr	i	i	i	i

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	+	+	−	−
com	+	+	−	−
abo	i	i	i	i
dnr	i	i	i	i

c(T_1) und c(T_2):

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	+	+	−	−
com	+	+	−	−
abo	i	i	i	i
dnr	i	i	i	i

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	+	i	i	−
com	+	+	−	−
abo	i	i	i	i
dnr	i	i	i	i

c(T_1) und a(T_2):

$T_1 \backslash T_2$	beg	com	abo	dnr
beg	+	i	i	−
com	+	−	+	−
abo	i	i	i	i
dnr	i	i	i	i

$T_2 \backslash T_1$	beg	com	abo	dnr
beg	+	+	−	−
com	i	i	i	i
abo	+	+	−	−
dnr	i	i	i	i

a(T_1) und c(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	+	+	−	−
com	i	i	i	i
abo	+	+	−	−
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	+	i	i	−
com	+	−	+	−
abo	i	i	i	i
dnr	i	i	i	i

a(T_1) und c(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	+	i	i	−
com	i	i	i	i
abo	+	+	−	−
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	+	−	+	−
com	+	−	+	−
abo	i	i	i	i
dnr	i	i	i	i

c(T_1) und a(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	+	−	+	−
com	+	−	+	−
abo	i	i	i	i
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	+	i	i	−
com	i	i	i	i
abo	+	+	−	−
dnr	i	i	i	i

a(T_1) und a(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	+	i	i	−
com	i	i	i	i
abo	+	−	+	−
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	+	−	+	−
com	i	i	i	i
abo	+	−	+	−
dnr	i	i	i	i

a(T_1) und a(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	+	−	+	−
com	i	i	i	i
abo	+	−	+	−
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	+	i	i	−
com	i	i	i	i
abo	+	−	+	−
dnr	i	i	i	i

dnr(T_1) und c(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	+	+	−	−
dnr	+	+	−	−

T_2 \ T_1	beg	com	abo	dnr
beg	−	−	+	+
com	−	−	+	+
abo	i	i	i	i
dnr	i	i	i	i

c(T_1) und dnr(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	−	−	+	+
com	−	−	+	+
abo	i	i	i	i
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	+	+	−	−
dnr	+	+	−	−

dnr(T_1) und a(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	+	−	+	−
dnr	+	−	+	−

T_2 \ T_1	beg	com	abo	dnr
beg	−	−	+	+
com	i	i	i	i
abo	−	−	+	+
dnr	i	i	i	i

a(T_1) und dnr(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	−	−	+	+
com	i	i	i	i
abo	−	−	+	+
dnr	i	i	i	i

T_2 \ T_1	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	+	−	+	−
dnr	+	−	+	−

dnr(T_1) und dnr(T_2):

T_1 \ T_2	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	−	−	+	+
dnr	−	−	+	+

T_2 \ T_1	beg	com	abo	dnr
beg	i	i	i	i
com	i	i	i	i
abo	−	−	+	+
dnr	−	−	+	+

Anhang C

Ergebnisse der Addition der elementaren Beziehungen

$\forall \rightarrow_i, \rightarrow_j \ (T_1 \rightarrow_{neu} T_2) = (T_1 \rightarrow_i T_2) \oplus (T_1 \rightarrow_j T_2)$, sowie $\forall \rightarrow_i, \rightarrow_j \ (T_1 \rightarrow_{neu} T_2) = (T_1 \rightarrow_i T_2) \oplus (T_1 \leftarrow_j T_2)$.

In den Fällen, in denen das Ergebnis wiederum eine elementare Beziehung ist, ist diese symbolisch dargestellt[1]; andernfalls ist die resultierende Matrix ausgedruckt. Beide TAs seien im Zustand **INSTALLED**. Wenn sich aus der Ergebnisbeziehung demands ergeben, sind diese aufgeführt.

C.1 gleich gerichtete Beziehungen addiert

C.1.1 $((T_1 \rightarrow_c T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \rightarrow_c T_2) \oplus (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 \leftrightarrow_c T_2)$

$(T_1 \rightarrow_c T_2) \oplus (T_1 \ {}_a\!\rightarrow_c T_2)$ T_2 demand RUN;
T_2 demand COMMIT;

= T_1

T_1 T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	+	+	−	−
d	+	+	−	−

T_1 T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	i	i	i	i
d	i	i	i	i

T_2

$(T_1 \rightarrow_c T_2) \oplus (T_1 \ {}_c\!\rightarrow_a T_2)$ T_1 demand ABORT;

= T_1

T_1 T_2	b	c	a	d
b	*	*	*	*
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

T_1 T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	*	−	+	*

T_2

$(T_1 \rightarrow_c T_2) \oplus (T_1 \ {}_c\!\mapsto_b T_2)$ = T_1

T_1 T_2	b	c	a	d
b	*	*	*	*
c	@	+	−	n
a	*	*	*	*
d	*	*	*	*

T_1 T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	−	+	*
d	*	−	+	*

T_2

[1] die TPMs der elementaren Matrizen sind auf Seite 33 zu finden.

$(T_1 \rightarrow_c T_2) \oplus (T_1 \;{}_a{\mapsto}_b\; T_2)$ $\quad$ T_2 demand RUN;

$=$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	+	*	*	−	b	*	*	*	*
c	+	+	−	−	c	*	*	*	*
a	@	*	*	n	a	*	−	+	*
d	@	*	*	n	d	i	i	i	i

$(T_1 \rightarrow_c T_2) \oplus (T_1 \;{}_b{\mapsto}_c\; T_2) \quad = \quad (T_1 \;{}_b{\mapsto}_c\; T_2)$

$(T_1 \rightarrow_c T_2) \oplus (T_1 \;{}_b{\mapsto}_a\; T_2)$ $\quad$ T_1 demand ABORT;

$=$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	*	−	+	*	b	*	−	+	*
c	i	i	i	i	c	−	−	+	+
a	*	*	*	*	a	*	−	+	*
d	*	*	*	*	d	*	−	+	*

$(T_1 \rightarrow_c T_2) \oplus (T_1 \mapsto_b T_2) \quad =$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	@	*	*	n	b	*	*	*	*
c	@	+	−	n	c	*	*	*	*
a	*	*	*	*	a	*	−	+	*
d	*	*	*	*	d	n	−	+	@

$(T_1 \rightarrow_c T_2) \oplus (T_1 \leftrightarrow_b T_2) \quad =$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	+	*	*	−	b	+	*	*	−
c	+	+	−	−	c	+	*	*	−
a	*	*	*	*	a	*	−	+	*
d	−	−	+	+	d	−	−	+	+

C.1.2 $((T_1 \leftrightarrow_c T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \;{}_a{\rightarrow}_c\; T_2)$ $\quad$ T_1 demand RUN;
T_1 demand COMMIT;
T_2 demand RUN;
T_2 demand COMMIT;

$=$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	+	+	−	−	b	+	+	−	−
c	+	+	−	−	c	+	+	−	−
a	i	i	i	i	a	i	i	i	i
d	i	i	i	i	d	i	i	i	i

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \;{}_c{\rightarrow}_a\; T_2)$ $\quad$ T_1 demand ABORT;
T_2 demand ABORT;

$=$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	*	−	+	*	b	*	−	+	*
c	i	i	i	i	c	i	i	i	i
a	*	−	+	*	a	*	−	+	*
d	*	−	+	*	d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \;{}_c{\mapsto}_b\; T_2) \quad =$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	*	*	*	*	b	*	*	*	*
c	@	+	−	n	c	+	+	−	−
a	*	−	+	*	a	*	−	+	*
d	*	−	+	*	d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \;{}_a{\mapsto}_b\; T_2)$ $\quad$ T_2 demand RUN;

$=$ T_1 (…) T_2

T_1 \ T_2	b	c	a	d	T_1 \ T_2	b	c	a	d
b	+	*	*	−	b	*	*	*	*
c	+	+	−	−	c	+	+	−	−
a	@	−	+	n	a	*	−	+	*
d	@	−	+	n	d	i	i	i	i

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \,{}_b\!\mapsto_c T_2) =$

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	−	+	*
d	*	−	+	*

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	−	−	+	+
d	−	−	+	+

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \,{}_b\!\mapsto_a T_2)$

T_1 demand ABORT;
T_2 demand ABORT;

=

T_1: T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

T_2: T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \mapsto_b T_2) =$

T_1: T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	+	−	n
a	*	−	+	*
d	*	−	+	*

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	−	+	*
d	n	−	+	@

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \leftrightarrow_b T_2) =$

T_1: T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	*	−	+	*
d	−	−	+	+

T_2: T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	*	−	+	*
d	−	−	+	+

C.1.3 $((T_1 \,{}_a\!\rightarrow_c T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \,{}_a\!\rightarrow_c T_2) \oplus (T_1 \,{}_c\!\rightarrow_a T_2) =$

T_1: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	+	+	−	−
d	+	+	−	−

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	+	+	−	−
d	+	+	−	−

$(T_1 \,{}_a\!\rightarrow_c T_2) \oplus (T_1 \,{}_c\!\mapsto_b T_2)$

T_2 demand RUN;

=

T_1: T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	@	*	*	n
a	+	+	−	−
d	+	+	−	−

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	+	−	−
d	i	i	i	i

$(T_1 \,{}_a\!\rightarrow_c T_2) \oplus (T_1 \,{}_a\!\mapsto_b T_2) =$

T_1: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	@	+	−	n
d	@	+	−	n

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	+	−	−
d	+	+	−	−

$(T_1 \,{}_a\!\rightarrow_c T_2) \oplus (T_1 \,{}_b\!\mapsto_c T_2)$

T_2 demand RUN;
T_2 demand COMMIT;

=

T_1: T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	+	+	−	−
d	+	+	−	−

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	i	i	i	i
d	i	i	i	i

$(T_1 \,{}_a\!\rightarrow_c T_2) \oplus (T_1 \,{}_b\!\mapsto_a T_2) =$

T_1: T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	+	+	−	−
d	+	+	−	−

T_2: T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	−	−	+	+
a	+	+	−	−
d	+	+	−	−

$(T_1\ {}_a\!\!\rightarrow_c T_2) \oplus (T_1 \mapsto_b T_2)$ T_2 demand RUN;

= T_1

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	+	+	−	−
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	+	−	−
d	!	i	i	!

T_2

$(T_1\ {}_a\!\!\rightarrow_c T_2) \oplus (T_1 \leftrightarrow_b T_2)$ T_1 demand RUN;
T_2 demand RUN;

= T_1

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	+	+	−	−
d	i	i	i	i

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	+	+	−	−
d	i	i	i	i

T_2

C.1.4 $((T_1\ {}_c\!\!\rightarrow_a T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1\ {}_c\!\!\mapsto_b T_2)$ = T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	−	+	n
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	*	*	*	*
d	*	−	+	*

T_2

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1\ {}_a\!\!\mapsto_b T_2)$ = T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	*	*	*	*
d	+	+	−	−

T_2

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1\ {}_b\!\!\mapsto_c T_2)$ T_1 demand ABORT;

= T_1

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	−	−	+	+
d	−	−	+	+

T_2

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1\ {}_b\!\!\mapsto_a T_2)$ = $(T_1\ {}_b\!\!\mapsto_a T_2)$

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1 \mapsto_b T_2)$ = T_1

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	−	+	n
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	*	*	*	*
d	n	−	+	@

T_2

$(T_1\ {}_c\!\!\rightarrow_a T_2) \oplus (T_1 \leftrightarrow_b T_2)$ = T_1

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	−	+	−
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	−	+	−
a	*	*	*	*
d	−	−	+	+

T_2

C.1.5 $((T_1\ {}_c\!\!\mapsto_b T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1\ {}_c\!\!\mapsto_b T_2) \oplus (T_1\ {}_a\!\!\mapsto_b T_2)$ T_2 demand RUN;

= T_1

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	@	*	*	n
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	i	i	i	i

T_2

$(T_1 \;{}_c{\mapsto}_b\; T_2) \oplus (T_1 \;{}_b{\mapsto}_c\; T_2) =$

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	@	+	−	n
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	−	−	+	+
d	−	−	+	+

$(T_1 \;{}_c{\mapsto}_b\; T_2) \oplus (T_1 \;{}_b{\mapsto}_a\; T_2) =$

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	@	−	+	n
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	−	−	+	+
a	*	*	*	*
d	*	−	+	*

$(T_1 \;{}_c{\mapsto}_b\; T_2) \oplus (T_1 \mapsto_b T_2) = (T_1 \mapsto_b T_2)$

$(T_1 \;{}_c{\mapsto}_b\; T_2) \oplus (T_1 \leftrightarrow_b T_2) =$

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	@	*	*	n
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	−	−	+	+

C.1.6 $((T_1 \;{}_a{\mapsto}_b\; T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \;{}_a{\mapsto}_b\; T_2) \oplus (T_1 \;{}_b{\mapsto}_c\; T_2)$

T_2 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	−	−	+	+
d	i	i	i	i

$(T_1 \;{}_a{\mapsto}_b\; T_2) \oplus (T_1 \;{}_b{\mapsto}_a\; T_2) =$

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	−	−	+	+
a	*	*	*	*
d	+	+	−	−

$(T_1 \;{}_a{\mapsto}_b\; T_2) \oplus (T_1 \mapsto_b T_2)$

T_2 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	!	i	i	!

$(T_1 \;{}_a{\mapsto}_b\; T_2) \oplus (T_1 \leftrightarrow_b T_2)$

T_1 demand RUN;
T_2 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	@	*	*	n
d	!	i	i	!

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	+	*	*	−
d	i	i	i	i

C.1.7 $((T_1 \;{}_b{\mapsto}_c\; T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \;{}_b{\mapsto}_c\; T_2) \oplus (T_1 \;{}_b{\mapsto}_a\; T_2)$

T_1 demand ABORT;
T_1 demand NOT RUN;

$=$

T_1 \ T_2	b	c	a	d
b	i	i	i	i
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	−	−	+	+
c	−	−	+	+
a	−	−	+	+
d	−	−	+	+

$(T_1\ {}_b{\mapsto}_c\ T_2) \oplus (T_1 \mapsto_b T_2) =$ (T_1

$T_1 \backslash T_2$	b	c	a	d
b	@	+	−	n
c	@	+	−	n
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	−	−	+	+
d	n	−	+	@

T_2)

$(T_1\ {}_b{\mapsto}_c\ T_2) \oplus (T_1 \leftrightarrow_b T_2) =$ (T_1

$T_1 \backslash T_2$	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	*	*	*
d	−	−	+	+

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	−	−	+	+
d	−	−	+	+

T_2)

C.1.8 $((T_1\ {}_b{\mapsto}_a\ T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1\ {}_b{\mapsto}_a\ T_2) \oplus (T_1 \mapsto_b T_2) =$ (T_1

$T_1 \backslash T_2$	b	c	a	d
b	@	−	+	n
c	@	−	+	n
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	−	−	+	+
a	*	*	*	*
d	n	−	+	@

T_2)

$(T_1\ {}_b{\mapsto}_a\ T_2) \oplus (T_1 \leftrightarrow_b T_2)$ T_2 demand ABORT;

= (T_1

$T_1 \backslash T_2$	b	c	a	d
b	+	−	+	−
c	+	−	+	−
a	*	−	+	*
d	−	−	+	+

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	i	i	i	i
a	*	*	*	*
d	−	−	+	+

T_2)

C.1.9 $((T_1 \mapsto_b T_2)) \oplus (T_1 \rightarrow_* T_2)$

$(T_1 \mapsto_b T_2) \oplus (T_1 \leftrightarrow_b T_2) =$ (T_1

$T_1 \backslash T_2$	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	*	*	*	*
d	−	−	+	+

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	n	−	+	@

T_2)

C.2 entgegengesetzt gerichtete Beziehungen addiert

C.2.1 $((T_1 \rightarrow_c T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \rightarrow_c T_2) \oplus (T_1 \leftrightarrow_c T_2) = (T_1 \leftrightarrow_c T_2)$

$(T_1 \rightarrow_c T_2) \oplus (T_1\ {}_a{\rightarrow}_c\ T_2)$ T_2 demand RUN;
T_2 demand COMMIT;

= (T_1

$T_1 \backslash T_2$	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	+	+	−	−
d	+	+	−	−

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	i	i	i	i
d	i	i	i	i

T_2)

$(T_1 \rightarrow_c T_2) \oplus (T_1\ {}_c{\rightarrow}_a\ T_2)$ T_1 demand ABORT;

= (T_1

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	*	−	+	*

T_2)

$(T_1 \rightarrow_c T_2) \oplus (T_2\ {}_c{\mapsto}_b\ T_1)$ =

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	*	*	n
a	*	−	+	*
d	*	−	+	*

$(T_1 \rightarrow_c T_2) \oplus (T_2\ {}_a{\mapsto}_b\ T_1)$ =

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	@	−	+	n
d	@	−	+	n

$(T_1 \rightarrow_c T_2) \oplus (T_2\ {}_b{\mapsto}_c\ T_1)$ =

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	−	−	+	+
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	−	+	*
d	*	−	+	*

$(T_1 \rightarrow_c T_2) \oplus (T_2\ {}_b{\mapsto}_a\ T_1)$

T_1 demand ABORT;

=

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	*	−	+	*

$(T_1 \rightarrow_c T_2) \oplus (T_2 \mapsto_b T_1)$ =

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	*	*	*
d	n	−	+	@

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	*	−	+	*
d	*	−	+	*

$(T_1 \rightarrow_c T_2) \oplus (T_1 \leftrightarrow_b T_2)$ =

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	−	+	*
d	−	−	+	+

C.2.2 $((T_1 \leftrightarrow_c T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \leftrightarrow_c T_2) \oplus (T_1\ {}_a{\rightarrow}_c\ T_2)$

T_1 demand RUN;
T_1 demand COMMIT;
T_2 demand RUN;
T_2 demand COMMIT;

=

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	i	i	i	i
d	i	i	i	i

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	i	i	i	i
d	i	i	i	i

$(T_1 \leftrightarrow_c T_2) \oplus (T_1\ {}_c{\rightarrow}_a\ T_2)$

T_1 demand ABORT;
T_2 demand ABORT;

=

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_2\ {}_c{\mapsto}_b\ T_1)$ =

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	−	+	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	+	−	n
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_2 \;{}_a\!\!\mapsto_b T_1)$

T_1 demand RUN;

$=$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	−	+	*
d	i	i	i	i

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	@	−	+	n
d	@	−	+	n

$(T_1 \leftrightarrow_c T_2) \oplus (T_2 \;{}_b\!\!\mapsto_c T_1) =$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	−	−	+	+
d	−	−	+	+

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_2 \;{}_b\!\!\mapsto_a T_1)$

T_1 demand ABORT;
T_2 demand ABORT;

$=$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	*	−	+	*
c	i	i	i	i
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_2 \mapsto_b T_1) =$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	+	+	−	−
a	*	−	+	*
d	n	−	+	@

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	@	*	*	n
c	@	+	−	n
a	*	−	+	*
d	*	−	+	*

$(T_1 \leftrightarrow_c T_2) \oplus (T_1 \leftrightarrow_b T_2) =$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	*	−	+	*
d	−	−	+	+

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	+	−	−
a	*	−	+	*
d	−	−	+	+

C.2.3 $((T_1 \;{}_a\!\!\rightarrow_c T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \;{}_a\!\!\rightarrow_c T_2) \oplus (T_1 \;{}_c\!\!\rightarrow_a T_2) =$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	+	+	−	−
d	+	+	−	−

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	−	+	*
a	+	+	−	−
d	+	+	−	−

$(T_1 \;{}_a\!\!\rightarrow_c T_2) \oplus (T_2 \;{}_c\!\!\mapsto_b T_1)$

T_1 demand RUN;

$=$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	+	−	−
d	i	i	i	i

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	@	*	*	n
a	+	+	−	−
d	+	+	−	−

$(T_1 \;{}_a\!\!\rightarrow_c T_2) \oplus (T_2 \;{}_a\!\!\mapsto_b T_1) =$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	+	−	−
d	+	+	−	−

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	@	+	−	n
d	@	+	−	n

$(T_1 \;{}_a\!\!\rightarrow_c T_2) \oplus (T_2 \;{}_b\!\!\mapsto_c T_1)$

T_1 demand RUN;
T_1 demand COMMIT;

$=$

T_1:

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	i	i	i	i
d	i	i	i	i

T_2:

$T_1 \backslash T_2$	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	+	+	−	−
d	+	+	−	−

$(T_1 \,{}_a{\rightarrow}_c\, T_2) \oplus (T_2 \,{}_b{\mapsto}_a\, T_1) =$

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	−	−	+	+
a	+	+	−	−
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	∗	−	+	∗
c	∗	−	+	∗
a	+	+	−	−
d	+	+	−	−

$(T_1 \,{}_a{\rightarrow}_c\, T_2) \oplus (T_2 \mapsto_b T_1)$ T_1 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	∗	∗	∗	∗
a	+	+	−	−
d	!	i	i	!

T_1 \ T_2	b	c	a	d
b	@	∗	∗	n
c	@	∗	∗	n
a	+	+	−	−
d	+	+	−	−

$(T_1 \,{}_a{\rightarrow}_c\, T_2) \oplus (T_1 \leftrightarrow_b T_2)$ T_1 demand RUN;
T_2 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	+	∗	∗	−
c	+	∗	∗	−
a	+	+	−	−
d	i	i	i	i

T_1 \ T_2	b	c	a	d
b	+	∗	∗	−
c	+	∗	∗	−
a	+	+	−	−
d	i	i	i	i

C.2.4 $((T_1 \,{}_c{\rightarrow}_a\, T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_2 \,{}_c{\mapsto}_b\, T_1) =$

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	∗	−	+	∗
a	∗	∗	∗	∗
d	∗	−	+	∗

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	@	−	+	n
a	∗	∗	∗	∗
d	∗	∗	∗	∗

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_2 \,{}_a{\mapsto}_b\, T_1) =$

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	∗	−	+	∗
a	∗	∗	∗	∗
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	∗	−	+	∗
a	@	∗	∗	n
d	@	∗	∗	n

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_2 \,{}_b{\mapsto}_c\, T_1)$ T_2 demand ABORT;

$=$

T_1 \ T_2	b	c	a	d
b	∗	−	+	∗
c	∗	−	+	∗
a	−	−	+	+
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	i	i	i	i
a	∗	∗	∗	∗
d	∗	∗	∗	∗

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_2 \,{}_b{\mapsto}_a\, T_1) = (T_2 \,{}_b{\mapsto}_a\, T_1)$

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_2 \mapsto_b T_1) =$

T_1 \ T_2	b	c	a	d
b	∗	∗	∗	∗
c	∗	−	+	∗
a	∗	∗	∗	∗
d	n	−	+	@

T_1 \ T_2	b	c	a	d
b	@	∗	∗	n
c	@	−	+	n
a	∗	∗	∗	∗
d	∗	∗	∗	∗

$(T_1 \,{}_c{\rightarrow}_a\, T_2) \oplus (T_1 \leftrightarrow_b T_2) =$

T_1 \ T_2	b	c	a	d
b	+	∗	∗	−
c	+	−	+	−
a	∗	∗	∗	∗
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	∗	∗	−
c	+	−	+	−
a	∗	∗	∗	∗
d	−	−	+	+

C.2.5 $((T_1 \;{}_{c}{\mapsto}_{b}\; T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \;{}_{c}{\mapsto}_{b}\; T_2) \oplus (T_2 \;{}_{a}{\mapsto}_{b}\; T_1) =$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	*	*	n
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	@	*	*	n
d	@	−	+	n

$(T_1 \;{}_{c}{\mapsto}_{b}\; T_2) \oplus (T_2 \;{}_{b}{\mapsto}_{c}\; T_1) =$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	*	*	n
a	−	−	+	+
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	*	*	*
d	*	−	+	*

$(T_1 \;{}_{c}{\mapsto}_{b}\; T_2) \oplus (T_2 \;{}_{b}{\mapsto}_{a}\; T_1)$

T_1 demand ABORT;

$=$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	!	i	i	!
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	*	−	+	*

$(T_1 \;{}_{c}{\mapsto}_{b}\; T_2) \oplus (T_2 \mapsto_b T_1) =$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	@	*	*	n
a	*	*	*	*
d	n	−	+	@

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	*	*	*	*
d	*	−	+	*

$(T_1 \;{}_{c}{\mapsto}_{b}\; T_2) \oplus (T_1 \leftrightarrow_b T_2) =$

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	@	*	*	n
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	−	−	+	+

C.2.6 $((T_1 \;{}_{a}{\mapsto}_{b}\; T_2)) \oplus (T_1 \leftarrow_* T_2)$

$(T_1 \;{}_{a}{\mapsto}_{b}\; T_2) \oplus (T_2 \;{}_{b}{\mapsto}_{c}\; T_1)$

T_1 demand RUN;
T_1 demand COMMIT;

$=$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	!	i	i	!
d	!	i	i	!

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	+	+	−	−
d	+	+	−	−

$(T_1 \;{}_{a}{\mapsto}_{b}\; T_2) \oplus (T_2 \;{}_{b}{\mapsto}_{a}\; T_1) =$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	−	−	+	+
a	@	*	*	n
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	*	*	*
d	+	+	−	−

$(T_1 \;{}_{a}{\mapsto}_{b}\; T_2) \oplus (T_2 \mapsto_b T_1)$

T_1 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	@	*	*	n
d	!	i	i	!

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	+	*	*	−
d	+	+	−	−

$(T_1 \;{}_{a}{\mapsto}_{b}\; T_2) \oplus (T_1 \leftrightarrow_b T_2)$

T_1 demand RUN;
T_2 demand RUN;

$=$

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	@	*	*	n
d	!	i	i	!

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	+	*	*	−
d	i	i	i	i

C.2.7 $((T_1 \,{}_b\!\!\longmapsto_c\, T_2)) \oplus (T_1 \longleftarrow_* T_2)$

$(T_1 \,{}_b\!\!\mapsto_c\, T_2) \oplus (T_2 \,{}_b\!\!\mapsto_a\, T_1)$ T_1 demand ABORT;

$=$ T_1

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	i	i	i	i
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	−	−	+	+
d	−	−	+	+

T_2

$(T_1 \,{}_b\!\!\mapsto_c\, T_2) \oplus (T_2 \mapsto_b T_1) \quad =$ T_1

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	*	*	*
d	n	−	+	@

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	−	−	+	+
d	−	−	+	+

T_2

$(T_1 \,{}_b\!\!\mapsto_c\, T_2) \oplus (T_1 \leftrightarrow_b T_2) \quad =$ T_1

T_1 \ T_2	b	c	a	d
b	+	+	−	−
c	+	+	−	−
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	−	−	+	+
d	−	−	+	+

T_2

C.2.8 $((T_1 \,{}_b\!\!\longmapsto_a\, T_2)) \oplus (T_1 \longleftarrow_* T_2)$

$(T_1 \,{}_b\!\!\mapsto_a\, T_2) \oplus (T_2 \mapsto_b T_1)$ T_2 demand ABORT;

$=$ T_1

T_1 \ T_2	b	c	a	d
b	*	−	+	*
c	*	−	+	*
a	*	−	+	*
d	n	−	+	@

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	!	i	i	!
a	*	*	*	*
d	*	*	*	*

T_2

$(T_1 \,{}_b\!\!\mapsto_a\, T_2) \oplus (T_1 \leftrightarrow_b T_2)$ T_2 demand ABORT;

$=$ T_1

T_1 \ T_2	b	c	a	d
b	+	−	+	−
c	+	−	+	−
a	*	−	+	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	i	i	i	i
a	*	*	*	*
d	−	−	+	+

T_2

C.2.9 $((T_1 \longmapsto_b T_2)) \oplus (T_1 \longleftarrow_* T_2)$

$(T_1 \mapsto_b T_2) \oplus (T_1 \leftrightarrow_b T_2) \quad =$ T_1

T_1 \ T_2	b	c	a	d
b	@	*	*	n
c	@	*	*	n
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	n	−	+	@

T_2

Anhang D

Ergebnisse der Multiplikation der elementaren Beziehungen

$\forall \rightarrow_i, \rightarrow_j \; (T_1 \rightarrow_{indirekt} T_3) = (T_1 \rightarrow_i T_2) \odot (T_2 \rightarrow_j T_3)$, sowie $(T_1 \rightarrow_{indirekt} T_3) = (T_1 \rightarrow_i T_2) \odot (T_2 \leftarrow_j T_3)$ und $(T_1 \rightarrow_{indirekt} T_3) = (T_1 \leftarrow_i T_2) \odot (T_2 \rightarrow_j T_3)$.

In den Fällen, in denen das Ergebnis wiederum eine elementare Beziehung ist, ist diese symbolisch dargestellt[1]; andernfalls ist die resultierende Matrix ausgedruckt. Alle drei TAs seien im Zustand **INSTALLED**. Aus der Multiplikation von zwei elementaren TPMs ergeben sich keine demands.

D.1 gleich gerichtete Beziehungen multipliziert

D.1.1 $((T_1 \rightarrow_c T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \rightarrow_c T_2) = (T_1 \rightarrow_c T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \leftrightarrow_c T_2) = (T_1 \rightarrow_c T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \,{}_a{\rightarrow}_c\, T_2) = (T_1 \text{—} T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \,{}_c{\rightarrow}_a\, T_2) = (T_1 \,{}_c{\rightarrow}_a\, T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \,{}_c{\mapsto}_b\, T_2) =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_2

$$(T_1 \rightarrow_c T_2) \odot (T_1 \,{}_a{\mapsto}_b\, T_2) = (T_1 \text{—} T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \,{}_b{\mapsto}_c\, T_2) = (T_1 \rightarrow_c T_2)$$

[1] die TPMs der elementaren Matrizen sind auf Seite 33 zu finden.

$$(T_1 \rightarrow_c T_2) \odot (T_1 \, {}_b{\mapsto}_a \, T_2) \quad = \quad (T_1 \, {}_c{\rightarrow}_a \, T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \mapsto_b T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

$$(T_1 \rightarrow_c T_2) \odot (T_1 \leftrightarrow_b T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

D.1.2 $((T_1 \leftrightarrow_c T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 \rightarrow_c T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 \leftrightarrow_c T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_a{\rightarrow}_c \, T_2) \quad = \quad (T_1 \, {}_a{\rightarrow}_c \, T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_c{\rightarrow}_a \, T_2) \quad = \quad (T_1 \, {}_c{\rightarrow}_a \, T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_c{\mapsto}_b \, T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_a{\mapsto}_b \, T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_b{\mapsto}_c \, T_2) \quad = \quad (T_1 \rightarrow_c T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_b{\mapsto}_a \, T_2) \quad = \quad (T_1 \, {}_c{\rightarrow}_a \, T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \mapsto_b T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \leftrightarrow_b T_2) \quad =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

D.1.3 $((T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2)) \odot (T_1 \rightarrow_{*} T_2)$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \rightarrow_{c} T_2) \quad = \quad (T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \leftrightarrow_{c} T_2) \quad = \quad (T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \quad = \quad (T_2 \rightarrow_{c} T_1)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{c}\!\!\mapsto_{b}\, T_2) \quad = \quad \left(T_1 \; \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & + & * & * & - \\ \text{d} & + & * & * & - \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & * & * & * & * \\ \text{d} & + & + & - & - \end{array} \; T_2 \right)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{a}\!\!\mapsto_{b}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{b}\!\!\mapsto_{c}\, T_2) \quad = \quad (T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \,{}_{b}\!\!\mapsto_{a}\, T_2) \quad = \quad (T_2 \rightarrow_{c} T_1)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \mapsto_{b} T_2) \quad = \quad \left(T_1 \; \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & + & * & * & - \\ \text{d} & + & * & * & - \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & * & * & * & * \\ \text{d} & + & + & - & - \end{array} \; T_2 \right)$$

$$(T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \odot (T_1 \leftrightarrow_{b} T_2) \quad = \quad \left(T_1 \; \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & + & * & * & - \\ \text{d} & + & * & * & - \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & * & * & * & * \\ \text{d} & + & + & - & - \end{array} \; T_2 \right)$$

D.1.4 $((T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2)) \odot (T_1 \rightarrow_{*} T_2)$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \rightarrow_{c} T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \leftrightarrow_{c} T_2) \quad = \quad (T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{a}\!\!\rightarrow_{c}\, T_2) \quad = \quad (T_1 \rightarrow_{c} T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{c}\!\!\mapsto_{b}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{a}\!\!\mapsto_{b}\, T_2) \quad = \quad \left(T_1 \; \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & + & * & * & - \\ \text{a} & * & * & * & * \\ \text{d} & * & * & * & * \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & \text{b} & \text{c} & \text{a} & \text{d} \\ \hline \text{b} & * & * & * & * \\ \text{c} & * & * & * & * \\ \text{a} & * & * & * & * \\ \text{d} & * & - & + & * \end{array} \; T_2 \right)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{b}\!\!\mapsto_{c}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \,{}_{b}\!\!\mapsto_{a}\, T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \mapsto_{b} T_2) \quad = \quad (T_1 - T_2)$$

$$(T_1 \,{}_{c}\!\!\rightarrow_{a}\, T_2) \odot (T_1 \leftrightarrow_{b} T_2) \quad = \quad (T_1 - T_2)$$

D.1.5 $((T_1 \,{}_{c}\!\mapsto_b\, T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{a}\!\rightarrow_c\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{c}\!\rightarrow_a\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{c}\!\mapsto_b\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{a}\!\mapsto_b\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{b}\!\mapsto_c\, T_2) \quad = \quad (T_1 \rightarrow_c T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{b}\!\mapsto_a\, T_2) \quad = \quad (T_1 \,{}_{c}\!\rightarrow_a\, T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \mapsto_b T_2) \quad = \quad (T_1 \,{}_{c}\!\mapsto_b\, T_2)$$

$$(T_1 \,{}_{c}\!\mapsto_b\, T_2) \odot (T_1 \leftrightarrow_b T_2) \quad =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_2

D.1.6 $((T_1 \,{}_{a}\!\mapsto_b\, T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{a}\!\rightarrow_c\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{c}\!\rightarrow_a\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{c}\!\mapsto_b\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{a}\!\mapsto_b\, T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{b}\!\mapsto_c\, T_2) \quad = \quad (T_1 \,{}_{a}\!\rightarrow_c\, T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \,{}_{b}\!\mapsto_a\, T_2) \quad = \quad (T_2 \rightarrow_c T_1)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \mapsto_b T_2) \quad = \quad (T_1 \,{}_{a}\!\mapsto_b\, T_2)$$

$$(T_1 \,{}_{a}\!\mapsto_b\, T_2) \odot (T_1 \leftrightarrow_b T_2) \quad =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_2

D.1.7 $((T_1\,{}_b{\mapsto}_c\,T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1\,{}_b{\mapsto}_c\,T_2)$$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\,{}_b{\mapsto}_c\,T_2)$$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) \quad = \quad (T_1\,{}_b{\mapsto}_a\,T_2)$$

$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_c{\mapsto}_b\,T_2) \quad = \quad \Big(T_1$

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

$T_2 \Big)$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_a{\mapsto}_b\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_b{\mapsto}_c\,T_2) \quad = \quad (T_1\,{}_b{\mapsto}_c\,T_2)$$

$$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1\,{}_b{\mapsto}_a\,T_2) \quad = \quad (T_1\,{}_b{\mapsto}_a\,T_2)$$

$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1 \mapsto_b T_2) \quad = \quad \Big(T_1$

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

$T_2 \Big)$

$(T_1\,{}_b{\mapsto}_c\,T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \Big(T_1$

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

$T_2 \Big)$

D.1.8 $((T_1\,{}_b{\mapsto}_a\,T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\,{}_b{\mapsto}_a\,T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) \quad = \quad (T_1\,{}_b{\mapsto}_c\,T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_c{\mapsto}_b\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_a{\mapsto}_b\,T_2) \quad = \quad \Big(T_1$

$T_1 \backslash T_2$	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_1 \backslash T_2$	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

$T_2 \Big)$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_b{\mapsto}_c\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1\,{}_b{\mapsto}_a\,T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1 \mapsto_b T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\,{}_b{\mapsto}_a\,T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad (T_1 \text{---} T_2)$$

D.1.9 $((T_1 \mapsto_b T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \mapsto_b T_2) \odot (T_1 \rightarrow_c T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \leftrightarrow_c T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_a\!\!\rightarrow_c T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_c\!\!\rightarrow_a T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_c\!\!\mapsto_b T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_a\!\!\mapsto_b T_2) = (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_b\!\!\mapsto_c T_2) = (T_1 \,{}_b\!\!\mapsto_c T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \,{}_b\!\!\mapsto_a T_2) = (T_1 \,{}_b\!\!\mapsto_a T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \mapsto_b T_2) = (T_1 \mapsto_b T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \leftrightarrow_b T_2) = \left(T_1 \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & + & * & * & - \\ c & + & * & * & - \\ a & * & * & * & * \\ d & * & * & * & * \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & - & - & + & + \end{array} T_2 \right)$$

D.1.10 $((T_1 \leftrightarrow_b T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \rightarrow_c T_2) = (T_1 — T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \leftrightarrow_c T_2) = \left(T_1 \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & * & - & + & * \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & + & * & * & - \\ a & * & * & * & * \\ d & * & * & * & * \end{array} T_2 \right)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_a\!\!\rightarrow_c T_2) = \left(T_1 \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & + & + & - & - \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & + & * & * & - \\ d & + & * & * & - \end{array} T_2 \right)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_c\!\!\rightarrow_a T_2) = (T_1 — T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_c\!\!\mapsto_b T_2) = (T_1 — T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_a\!\!\mapsto_b T_2) = \left(T_1 \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & + & * & * & - \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & + & * & * & - \end{array} T_2 \right)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_b\!\!\mapsto_c T_2) = (T_1 \,{}_b\!\!\mapsto_c T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \,{}_b\!\!\mapsto_a T_2) = (T_1 \,{}_b\!\!\mapsto_a T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \mapsto_b T_2) = \left(T_1 \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & + & * & * & - \\ c & + & * & * & - \\ a & * & * & * & * \\ d & * & * & * & * \end{array} \quad \begin{array}{c|cccc} T_1 \backslash T_2 & b & c & a & d \\ \hline b & * & * & * & * \\ c & * & * & * & * \\ a & * & * & * & * \\ d & - & - & + & + \end{array} T_2 \right)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \leftrightarrow_b T_2) = (T_1 \leftrightarrow_b T_2)$$

D.2 entgegengesetzt gerichtete Beziehungen multipliziert

D.2.1 $((T_1 \rightarrow_c T_2)) \odot (T_1 \leftarrow_* T_2)$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 \rightarrow_c T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \, {}_a\!\rightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \, {}_c\!\rightarrow_a T_2) \quad = \quad (T_1 \, {}_c\!\rightarrow_a T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \, {}_c\!\mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \, {}_a\!\mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \, {}_b\!\mapsto_c T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \, {}_b\!\mapsto_a T_1) \quad = \quad (T_2 \, {}_b\!\mapsto_a T_1)$$

$$(T_1 \rightarrow_c T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \rightarrow_c T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \left(T_1 \; \cdots \; T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

D.2.2 $((T_1 \leftrightarrow_c T_2)) \odot (T_1 \leftarrow_* T_2)$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_2 \rightarrow_c T_1)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 \leftrightarrow_c T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_a\!\rightarrow_c T_2) \quad = \quad (T_1 \, {}_a\!\rightarrow_c T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \, {}_c\!\rightarrow_a T_2) \quad = \quad (T_1 \, {}_c\!\rightarrow_a T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \, {}_c\!\mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \, {}_a\!\mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \, {}_b\!\mapsto_c T_1) \quad = \quad (T_2 \, {}_b\!\mapsto_c T_1)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \, {}_b\!\mapsto_a T_1) \quad = \quad (T_2 \, {}_b\!\mapsto_a T_1)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \leftrightarrow_c T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \left(T_1 \; \cdots \; T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

D.2.3 $((T_1\ {}_a{\longrightarrow}_c\ T_2)) \odot (T_1 \longleftarrow_* T_2)$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\ {}_a{\rightarrow}_c\ T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_1\ {}_a{\rightarrow}_c\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_1\ {}_c{\rightarrow}_a\ T_2) \quad = \quad (T_2 \rightarrow_c T_1)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2\ {}_c{\mapsto}_b\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2\ {}_a{\mapsto}_b\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2\ {}_b{\mapsto}_c\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2\ {}_b{\mapsto}_a\ T_1) \quad = \quad (T_2\ {}_b{\mapsto}_c\ T_1)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_a{\rightarrow}_c\ T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \left(T_1 \begin{array}{|c||c|c|c|c|} \hline {}_{T_1}\,{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline\hline \mathrm{b} & * & * & * & * \\ \hline \mathrm{c} & * & * & * & * \\ \hline \mathrm{a} & + & * & * & - \\ \hline \mathrm{d} & + & * & * & - \\ \hline \end{array} \quad \begin{array}{|c||c|c|c|c|} \hline {}_{T_1}\,{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline\hline \mathrm{b} & * & * & * & * \\ \hline \mathrm{c} & * & * & * & * \\ \hline \mathrm{a} & * & * & * & * \\ \hline \mathrm{d} & + & + & - & - \\ \hline \end{array} T_2 \right)$$

D.2.4 $((T_1\ {}_c{\longrightarrow}_a\ T_2)) \odot (T_1 \longleftarrow_* T_2)$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1\ {}_c{\rightarrow}_a\ T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\ {}_c{\rightarrow}_a\ T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_1\ {}_a{\rightarrow}_c\ T_2) \quad = \quad (T_1 \rightarrow_c T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_1\ {}_c{\rightarrow}_a\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2\ {}_c{\mapsto}_b\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2\ {}_a{\mapsto}_b\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2\ {}_b{\mapsto}_c\ T_1) \quad = \quad (T_2\ {}_b{\mapsto}_a\ T_1)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2\ {}_b{\mapsto}_a\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_c{\rightarrow}_a\ T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad (T_1 \text{---} T_2)$$

D.2.5 $((T_1\ {}_{c}{\mapsto}_{b}\ T_2)) \odot (T_1 \leftarrow_{*} T_2)$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2 \rightarrow_{c} T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_1 \leftrightarrow_{c} T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_1\ {}_{a}{\rightarrow}_{c}\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_1\ {}_{c}{\rightarrow}_{a}\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{c}{\mapsto}_{b}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{a}{\mapsto}_{b}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{b}{\mapsto}_{c}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{b}{\mapsto}_{a}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_2 \mapsto_{b} T_1) \quad = \quad (T_1 \text{---} T_2)$$

$(T_1\ {}_{c}{\mapsto}_{b}\ T_2) \odot (T_1 \leftrightarrow_{b} T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

$T_2 \Big)$

D.2.6 $((T_1\ {}_{a}{\mapsto}_{b}\ T_2)) \odot (T_1 \leftarrow_{*} T_2)$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2 \rightarrow_{c} T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_1 \leftrightarrow_{c} T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_1\ {}_{a}{\rightarrow}_{c}\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_1\ {}_{c}{\rightarrow}_{a}\ T_2) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{c}{\mapsto}_{b}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{a}{\mapsto}_{b}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{b}{\mapsto}_{c}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2\ {}_{b}{\mapsto}_{a}\ T_1) \quad = \quad (T_1 \text{---} T_2)$$

$$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_2 \mapsto_{b} T_1) \quad = \quad (T_1 \text{---} T_2)$$

$(T_1\ {}_{a}{\mapsto}_{b}\ T_2) \odot (T_1 \leftrightarrow_{b} T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

$T_2 \Big)$

D.2.7 $((T_1\,{}_b\!\!\mapsto_c T_2)) \odot (T_1 \leftarrow_* T_2)$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\,{}_b\!\!\mapsto_c T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_1\,{}_a\!\!\rightarrow_c T_2) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_1\,{}_c\!\!\rightarrow_a T_2) \quad = \quad (T_1\,{}_b\!\!\mapsto_a T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2\,{}_c\!\!\mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2\,{}_a\!\!\mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2\,{}_b\!\!\mapsto_c T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2\,{}_b\!\!\mapsto_a T_1) \quad = \quad \left(T_1 \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & - & - & + & + \\ \mathrm{c} & - & - & + & + \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \\ \hline \end{array} \; \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & - & - & + & + \\ \mathrm{c} & - & - & + & + \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \\ \hline \end{array} T_2 \right)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_c T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \left(T_1 \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & + & * & * & - \\ \mathrm{c} & + & * & * & - \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \\ \hline \end{array} \; \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & * & * & * & * \\ \mathrm{c} & * & * & * & * \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & - & - & + & + \\ \hline \end{array} T_2 \right)$$

D.2.8 $((T_1\,{}_b\!\!\mapsto_a T_2)) \odot (T_1 \leftarrow_* T_2)$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1\,{}_b\!\!\mapsto_a T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1\,{}_b\!\!\mapsto_a T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_1\,{}_a\!\!\rightarrow_c T_2) \quad = \quad (T_1\,{}_b\!\!\mapsto_c T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_1\,{}_c\!\!\rightarrow_a T_2) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2\,{}_c\!\!\mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2\,{}_a\!\!\mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2\,{}_b\!\!\mapsto_c T_1) \quad = \quad \left(T_1 \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & - & - & + & + \\ \mathrm{c} & - & - & + & + \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \\ \hline \end{array} \; \begin{array}{||c||c|c|c|c||} \hline {}_{T_1}{}^{T_2} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{d} \\ \hline \mathrm{b} & - & - & + & + \\ \mathrm{c} & - & - & + & + \\ \mathrm{a} & * & * & * & * \\ \mathrm{d} & * & * & * & * \\ \hline \end{array} T_2 \right)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2\,{}_b\!\!\mapsto_a T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 \!-\! T_2)$$

$$(T_1\,{}_b\!\!\mapsto_a T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad (T_1 \!-\! T_2)$$

D.2.9 $((T_1 \mapsto_b T_2)) \odot (T_1 \leftarrow_* T_2)$

$$(T_1 \mapsto_b T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \;{}_a{\rightarrow}_c\; T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_1 \;{}_c{\rightarrow}_a\; T_2) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_2 \;{}_c{\mapsto}_b\; T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_2 \;{}_a{\mapsto}_b\; T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_2 \;{}_b{\mapsto}_c\; T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_2 \;{}_b{\mapsto}_a\; T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \mapsto_b T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad (T_1 — T_2)$$

$(T_1 \mapsto_b T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

$T_2 \Big)$

D.2.10 $((T_1 \leftrightarrow_b T_2)) \odot (T_1 \leftarrow_* T_2)$

$(T_1 \leftrightarrow_b T_2) \odot (T_2 \rightarrow_c T_1) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_2 \Big)$

$(T_1 \leftrightarrow_b T_2) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_2 \Big)$

$(T_1 \leftrightarrow_b T_2) \odot (T_1 \;{}_a{\rightarrow}_c\; T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

$T_2 \Big)$

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \;{}_c{\rightarrow}_a\; T_2) \quad = \quad (T_1 — T_2)$$

$(T_1 \leftrightarrow_b T_2) \odot (T_2 \;{}_c{\mapsto}_b\; T_1) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_2 \Big)$

$$(T_1 \leftrightarrow_b T_2) \odot (T_2 \ {}_a{\mapsto}_b \ T_1) \quad = \quad \left(T_1 \quad T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

$$(T_1 \leftrightarrow_b T_2) \odot (T_2 \ {}_b{\mapsto}_c \ T_1) \quad = \quad \left(T_1 \quad T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$$(T_1 \leftrightarrow_b T_2) \odot (T_2 \ {}_b{\mapsto}_a \ T_1) \quad = \quad (T_1 — T_2)$$

$$(T_1 \leftrightarrow_b T_2) \odot (T_2 \mapsto_b T_1) \quad = \quad \left(T_1 \quad T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$$(T_1 \leftrightarrow_b T_2) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad (T_1 \leftrightarrow_b T_2)$$

D.2.11 $((T_1 \ {}_c{\leftarrow} \ T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_2 \rightarrow_c T_1)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_a{\rightarrow}_c \ T_2) \quad = \quad (T_1 \ {}_a{\rightarrow}_c \ T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_c{\rightarrow}_a \ T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_c{\mapsto}_b \ T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_a{\mapsto}_b \ T_2) \quad = \quad \left(T_1 \quad T_2 \right)$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_b{\mapsto}_c \ T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \ {}_b{\mapsto}_a \ T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \mapsto_b T_2) \quad = \quad (T_1 — T_2)$$

$$(T_2 \rightarrow_c T_1) \odot (T_1 \leftrightarrow_b T_2) \quad = \quad (T_1 — T_2)$$

D.2.12 $((T_1\,{}_c{\leftarrow}_a\,T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1 \rightarrow_c T_2) = (T_1\,{}_a{\rightarrow}_c\,T_2)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1 \leftrightarrow_c T_2) = (T_1\,{}_a{\rightarrow}_c\,T_2)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) = (T_2 \rightarrow_c T_1)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_c{\mapsto}_b\,T_2) =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_2

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_a{\mapsto}_b\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_b{\mapsto}_c\,T_2) = (T_1\,{}_a{\rightarrow}_c\,T_2)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1\,{}_b{\mapsto}_a\,T_2) = (T_2 \rightarrow_c T_1)$$

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1 \mapsto_b T_2) =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_2

$$(T_1\,{}_a{\rightarrow}_c\,T_2) \odot (T_1 \leftrightarrow_b T_2) =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_2

D.2.13 $((T_1\,{}_a{\leftarrow}_c\,T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1 \rightarrow_c T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1 \leftrightarrow_c T_2) = (T_1\,{}_c{\rightarrow}_a\,T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) = (T_1 \rightarrow_c T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_c{\mapsto}_b\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_a{\mapsto}_b\,T_2) =$$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_2

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_b{\mapsto}_c\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1\,{}_b{\mapsto}_a\,T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1 \mapsto_b T_2) = (T_1 — T_2)$$

$$(T_1\,{}_c{\rightarrow}_a\,T_2) \odot (T_1 \leftrightarrow_b T_2) = (T_1 — T_2)$$

D.2.14 $((T_1\ {}_{b}{\leftarrow}_{c}\ T_2)) \odot (T_1 \rightarrow_{*} T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1 \rightarrow_{c} T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1 \leftrightarrow_{c} T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_2

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{a}{\rightarrow}_{c}\ T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_2

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{c}{\rightarrow}_{a}\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{c}{\mapsto}_{b}\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{a}{\mapsto}_{b}\ T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

T_2

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{b}{\mapsto}_{c}\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{b}{\mapsto}_{a}\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1 \mapsto_{b} T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{c}{\mapsto}_{b}\ T_1) \odot (T_1 \leftrightarrow_{b} T_2) \quad = \quad (T_1 — T_2)$

D.2.15 $((T_1\ {}_{b}{\leftarrow}_{a}\ T_2)) \odot (T_1 \rightarrow_{*} T_2)$

$(T_2\ {}_{a}{\mapsto}_{b}\ T_1) \odot (T_1 \rightarrow_{c} T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_2

$(T_2\ {}_{a}{\mapsto}_{b}\ T_1) \odot (T_1 \leftrightarrow_{c} T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_2

$(T_2\ {}_{a}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{a}{\rightarrow}_{c}\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_{a}{\mapsto}_{b}\ T_1) \odot (T_1\ {}_{c}{\rightarrow}_{a}\ T_2) \quad =$

T_1

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_2

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1\ {}_c{\mapsto}_b\ T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

$T_2 \Big)$

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1\ {}_a{\mapsto}_b\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1\ {}_b{\mapsto}_c\ T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

$T_2 \Big)$

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1\ {}_b{\mapsto}_a\ T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$T_2 \Big)$

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1 \mapsto_b T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	@	*	*	n

$T_2 \Big)$

$(T_2\ {}_a{\mapsto}_b\ T_1) \odot (T_1\ \leftrightarrow_b T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

$T_2 \Big)$

D.2.16 $((T_1\ {}_c{\leftarrow}_b\ T_2)) \odot (T_1 \rightarrow_* T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1 \rightarrow_c T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1 \leftrightarrow_c T_2) \quad = \quad (T_2 \rightarrow_c T_1)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_a{\rightarrow}_c\ T_2) \quad = \quad (T_1\ {}_a{\rightarrow}_c\ T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_c{\rightarrow}_a\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_c{\mapsto}_b\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_a{\mapsto}_b\ T_2) \quad = \quad \Big(T_1$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

$T_2 \Big)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_b{\mapsto}_c\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ {}_b{\mapsto}_a\ T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1 \mapsto_b T_2) \quad = \quad (T_1 — T_2)$

$(T_2\ {}_b{\mapsto}_c\ T_1) \odot (T_1\ \leftrightarrow_b T_2) \quad = \quad (T_2\ {}_b{\mapsto}_c\ T_1)$

D.2.17 $((T_1\,{}_a{\leftarrow}_b\;T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1 \rightarrow_c T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1 \leftrightarrow_c T_2) = (T_1\,{}_c{\rightarrow}_a\,T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) = (T_1 \rightarrow_c T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_c{\mapsto}_b\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_a{\mapsto}_b\,T_2) =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_b{\mapsto}_c\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1\,{}_b{\mapsto}_a\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1 \mapsto_b T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2\,{}_b{\mapsto}_a\,T_1) \odot (T_1 \leftrightarrow_b T_2) = (T_2\,{}_b{\mapsto}_a\,T_1)$$

D.2.18 $((T_1\,{}_b{\leftarrow}\;T_2)) \odot (T_1 \rightarrow_* T_2)$

$$(T_2 \mapsto_b T_1) \odot (T_1 \rightarrow_c T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1 \leftrightarrow_c T_2) =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	*	−	+	*

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_a{\rightarrow}_c\,T_2) =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	+	−	−

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	+	*	*	−
d	+	*	*	−

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_c{\rightarrow}_a\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_c{\mapsto}_b\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_a{\mapsto}_b\,T_2) =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	@	*	*	n

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	+	*	*	−

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_b{\mapsto}_c\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1\,{}_b{\mapsto}_a\,T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1 \mapsto_b T_2) = (T_1 \,\text{—}\, T_2)$$

$$(T_2 \mapsto_b T_1) \odot (T_1 \leftrightarrow_b T_2) =$$

T_1 \ T_2	b	c	a	d
b	*	*	*	*
c	*	*	*	*
a	*	*	*	*
d	−	−	+	+

T_1 \ T_2	b	c	a	d
b	+	*	*	−
c	+	*	*	−
a	*	*	*	*
d	*	*	*	*

Literaturverzeichnis

[AgJa90] Agrawal, Rakesh und Jagadish, H. V.: *Hybrid Transitive Closure Algorithms*, 16th VLDB Conference, Brisbane, 1990

[Burkert] Burkert, Holger: *Systemarchitektur und Betriebsmodell für das verteilte Transaktionssystem APRICOTS*, IPVR[2], Diplomarbeit (in Vorbereitung)

[ChRa90] Chrysanthis, P. K. und Ramamritham, K.: *ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behaviour*, Proc. ACM SIGMOD 1990

[GaSa87] Garcia-Molina, H. und Salem, K.: *Sagas*, Proc. ACM SIGMOD 1987

[HU] Hopcroft, J. E und Ullman, J. D.: *Introduction to Automata Theory, languages and computation*, Addison-Wesley 1979

[LauDun] Laube, Dieter und Dung, Tran Van: *Entwurf und Spezifikation eines prototypischen ConTract–Managers*, IPVR[2], Doppelstudienarbeit Nr. 875

[Lutz] Lutz, Rainer: *Verteilte Transaktionsverarbeitung mit Transaktionsbeziehungsnetzen*, IPVR[2], Diplomarbeit (in Vorbereitung)

[Moss81] Moss, J. E. B.: *Nested Transactions: An Approach to Reliable Computing*, M.I.T, Report MIT-LCS-TR-260, 1981

[Reuter89] Reuter, Andreas: *ConTracts: A Means for Extending Control Beyond Transaction Boundaries*, Proc. 2nd. Int. Workshop on High Performance Transaction Systems, Asilomar, Sept. 1989

[ReutWae90] Reuter, Andreas und Wächter, Helmut: *Neue Transaktionskonzepte*, Techn. Bericht, IPVR[2], Mai 1990

[ReutWae91] Reuter, Andreas und Wächter, Helmut: *The ConTractModel*, IEEE Data Engineering Bulletin, März 1991

[Waech90] Wächter, Helmut: *ConTracts: A Means for Improving Reliability in Distributed Computing*, IEEE Proc. Spring CompCon, San Francisco, 1991

[WaeReut90] Wächter, Helmut und Reuter, Andreas: *Grundkonzepte und Realisierungsstrategien des ConTract–Modells*, In: Informatik – Forschung und Entwicklung, Springer Verlag 1990

[Wahl] Wahl, Peter: *Interpretative Abwicklung von Prädikat–Transitionsnetzen*, IPVR[2], Studienarbeit Nr. 1021

[2] Institut für Parallele und Verteilte Höchstleistungsrechner, Universität Stuttgart