Institut fuer Informatik
Humboldt-Universitaet zu Berlin
Maerz 1998

Einsatz von ILF und DAWN zur Verifikation verteilter Algorithmen – Eine Vorstudie –

Thomas Baar Ekkart Kindler

13. März 1998

1 Einführung

Es ist meist sehr aufwendig, Beweise so zu führen, daß sie rein syntaktisch auf ihre Korrektheit hin überprüft werden können. Nahezu unmöglich ist es, derartige Beweise auch noch intuitiv verständlich zu führen, da viele für das Verständnis unwesentliche Details den Blick auf das Wesentliche verstellen.

DAWN[1] [11] wurde mit dem Ziel entwickelt, verteilte Algorithmen verständlich zu modellieren und deren Korrektheit intuitiv verständlich zu beweisen; trotzdem ist es möglich, Beweise soweit auszuführen, daß sie rein syntaktisch auf ihre Korrektheit überprüft werden können. Aber auch hier ist das Erstellen derartig detaillierter Beweise sehr aufwendig und niemand möchte derartig langatmige Beweise wirklich lesen.

ILF[2] [3] wurde mit dem Ziel entwickelt, Beweise, die von einem Theorembeweiser automatisch erzeugt wurden, aufzubereiten und anschaulich darzustellen. Darüber hinaus erlaubt ILF die Nutzung verschiedener Theorembeweiser — insbesondere können so die Stärken der verschiedenen Theorembeweiser kombiniert werden.

In dieser Vorstudie werden wir anhand eines einfachen Beispiels andeuten, wie ILF und DAWN kombiniert eingesetzt werden können, um die sich scheinbar widersprechenden Anforderungen der intuitiven Verständlichkeit und der syntaktisch überprüfbaren Korrektheit eines Beweises zu kombinieren. Die Beweise werden in der temporalen Logik von DAWN auf relativ hoher Abstraktionsebene geführt; die fehlenden Details werden anschließend ergänzt und mit Hilfe von ILF automatisch bewiesen. Wir mußten diese Details noch von Hand ergänzen — zukünftig soll dies aber automatisch geschehen.

In dieser Vorstudie zeigen wir die Realisierbarkeit eines interaktiven Werkzeuges zum Beweisen von verteilten Algorithmen mit Hilfe von automatischen Theorembeweisern. Insbesondere deuten wir eine mögliche Schnittstelle zwischen DAWN und ILF an, die durch die generierten Beweisaufgaben bestimmt wird. Diese Aufgaben lassen sich nach der Wahl von geeigneten Theorien automatisch beweisen. Da die Beweisaufgaben auch in anderen Beispielen ähnlich einfach sind, ist zu erwarten, daß sie sich im allgemeinen automatisch beweisen lassen. Dazu müssen

[1] DAWN ist ein Akronym für *Distributed Algorithms' Working Notation.*
[2] ILF ist ein Akronym für *Integrating Logical Functions.*

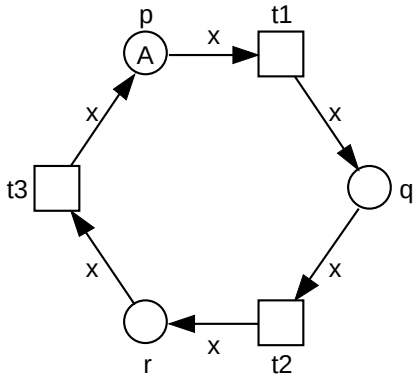
jedoch geeignete Theorien und Strategien entwickelt werden, was einem längerfristigen Projekt vorbehalten bleibt.

2 DAWN: Ein Beispiel

In diesem Abschnitt stellen wir DAWN anhand eines Beispiels vor. Wir werden zeigen, wie verteilte Algorithmen modelliert und ihre Eigenschaften beschrieben werden. Wir haben bewußt ein sehr einfaches Beispiel ausgewählt, damit wir uns auf die Beschreibung des Zusammenspiels von ILF und DAWN konzentrieren können, und die für ILF generierten Beweisaufgaben überschaubar bleiben. Für „echte“ Beispiele von verteilten Algorithmen und eine Einführung in DAWN verweisen wir auf [11, 10].

2.1 Modellierung

Ein verteilter Algorithmus wird in DAWN als Petrinetz[3] modelliert. Abbildung 1 zeigt einen Algorithmus, in dem jeder Agent aus der Menge A die Zustände p, q und r durchläuft; dabei beeinflußen sich die Agenten gegenseitig nicht. Initial befindet sich jeder Agent im Zustand p. Ein möglicher Zustand ist in einem Petrinetz durch eine *Stelle* repräsentiert, die graphisch als Kreis dargestellt ist; ein Agent x ist im Zustand p, wenn sich eine *Marke* x auf der Stelle p befindet. Es können sich jedoch mehrere Marken (sogar mehrmals die gleiche Marke) auf einer Stelle befinden; in unserem Beispiel kommt jeder Agent aus A auf der Stelle p genau einmal vor. Die Ansammlung von Marken auf den verschiedenen Stellen nennen wir eine *Markierung* des Petrinetzes; technisch können wir eine Stelle als eine Variable der Sorte Multimenge über den Agenten interpretieren. Mögliche Übergänge von einer Markierungen zu einer anderen werden



sorts AGENT
constants A: set of AGENT
variables x: AGENT

Abbildung 1: Ein einfacher Algorithmus

durch *Transitionen* repräsentiert, die graphisch als Quadrate dargestellt sind. Welche Veränderung das *Schalten* einer Transition bewirkt, wird durch die Inschriften der Kanten zwischen den Stellen und der entsprechenden Transition beschrieben — abhängig von der Belegung der freien

[3]Unter einem Petrinetz verstehen wir in DAWN immer ein algebraisches Petrinetz wie es in [6] definiert ist.

Variablen. Beispielsweise kann Transition $t1$ mit der Belegung $x = a$ schalten, wenn sich der Agent a auf der Stelle p befindet; durch das Schalten wird Agent a von der Stelle p entfernt und der Stelle q hinzugefügt. Für eine formale Definition der *Schaltregel* verweisen wir auf [6, 11].

2.2 Spezifikation

Die gewünschten Eigenschaften eines Algorithmus werden in DAWN durch Formeln einer temporalen Logik formuliert. In diesen Formeln kommen die Stellen als multimengen-wertige Variablen vor, die den Bezug zur Markierung des Petrinetzes herstellen. In unserem Beispiel betrachten wir zwei Eigenschaften:

1. Immer wenn sich ein Agent x im Zustand p befindet, dann ist er nicht zugleich im Zustand q. Dies wird durch die *Invariante*

$$\Box\, p(x) \Rightarrow \neg q(x)$$

   ausgedrückt. Dabei beziehen sich p und q auf die Stellen des Petrinetzes, und x ist eine (implizit allquantifizierte) Variable der Sorte $AGENT$. Der Ausdruck $p(x)$ ist wahr, wenn das Element x mindestens einmal in der Multimenge p vorkommt.

2. Wenn sich ein Agent x im Zustand p befindet, wird er sich irgendwann zu einem späteren Zeitpunkt auch in dem Zustand r befinden. Dies wird durch die *Leadsto*-Aussage

$$p(x) \rhd r(x)$$

   ausgedrückt.

2.3 Verifikation

Die Gültigkeit der beiden obigen Eigenschaften können wir in DAWN durch einfache Beweisregeln nachweisen. Der Beweis für $\Box\, p(x) \Rightarrow \neg q(x)$ sieht wie folgt aus:

(1)	$\Box\, p + q + r = A$	S-Invariant: $p + q + r$
(2)	$\Box\, p(x) \Rightarrow \neg q(x)$	Weakening (1)

Der Beweis ist aus numerierten Zeilen aufgebaut; nach der Nummer folgt die Eigenschaft die in diesem Beweisschritt gezeigt wird; dann folgt das Argument für die Gültigkeit der Eigenschaft.

Die erste Zeile besagt, daß $p + q + r$ eine sogenannte *S-Invariante* ist, d.h. die Summe[4] aller Marken auf den Stellen p, q und r wird von keiner Transition durch das Schalten verändert. Da die Summe der Marken initial A ergibt, ist der Wert von $p + q + r$ immer A. Jeder, der etwas Erfahrung mit Petrinetzen besitzt, wird sofort zustimmen, daß $p + q + r$ eine S-Invariante des Petrinetzes aus Abb. 1 ist — deshalb wird ein detaillierter Beweis nicht angegeben. Wir werden aber im nächsten Abschnitt zeigen, wie mit Hilfe eines Theorembeweisers das Beweisargument automatisch formal nachgeprüft werden kann.

Die zweite Zeile des Beweises besagt, daß $p(x) \Rightarrow \neg q(x)$ eine logische Konsequenz (eine Abschwächung) von $p + q + r = A$ ist. Denn wir wissen, daß jeder Agent aus A auf den Stellen p, q und r insgesamt genau einmal vorkommt; wenn sich Agent x also auf der Stelle p befindet (d.h.

[4] Das Symbol + bezeichnet hier die Summe von Multimengen.

wenn $p(x)$ gilt), kann x nicht außerdem auf der Stelle q vorkommen (d.h. es gilt $\neg q(x)$). Auch diese Argumentation ist intuitiv nachvollziehbar — deshalb wird dieses Argument nicht im Detail ausgeführt. Wieder werden wir im nächsten Abschnitt zeigen, wie man diese Argumentation automatisch mit Hilfe eines Theorembeweisers formal überprüfen kann.

Der Beweis für $p(x) \rhd r(x)$ sieht wie folgt aus (damit wir uns in Abschnitt 3 auf die einzelnen Beweiszeilen beziehen können, numerieren wir sie fortlaufend):

(3)	$p(x) \rhd q(x)$	Progress: $t1$
(4)	$q(x) \rhd r(x)$	Progress: $t2$
(5)	$p(x) \rhd r(x)$	Transitivity (3), (4)

Die Zeile (3) dieses Beweises sagt, daß wir die Eigenschaft $p(x) \rhd q(x)$ unmittelbar am Netz ablesen können; denn wenn $p(x)$ gilt, kann Transition $t1$ schalten; nach dem Schalten gilt $q(x)$. Außerdem sehen wir sofort, daß das Schalten von $t1$ durch keine andere (in Konflikt stehende Transition) verhindert werden kann. Auch hier sparen wir uns zunächst weitere formale Argumente. Die Zeile (4) ist analog zu Zeile (3).

Die Zeile (5) kombiniert die in Zeilen (3) und (4) bewiesenen Leadsto-Eigenschaften transitiv. Die Korrektheit dieser Regelanwendung kann rein syntaktisch überprüft werden.

3 Beweisaufgaben

Die im vorangegangenen Abschnitt dargestellten Beweise sind mit etwas Übung unmittelbar nachvollziehbar; in diesem Beispiel besteht auch kaum die Gefahr, daß wesentliche Beweisargumente fehlen. Bei längeren und ggf. komplizierteren Beweisen ist diese Gefahr allerdings gegeben — im schlimmsten Falle stimmen einige der einfachen Argumente nicht. Für diese Fälle ist es sinnvoll, die Korrektheit der Beweisargumente formal nachzuprüfen.

Wir zeigen nun anhand des Beweises aus Abschnitt 2.3, wie eine formale Überprüfung der einzelnen Beweisschritte aussieht. Wir haben diese Überprüfungen alle mit ILF automatisch durchgeführt; in Abschnitt 4 werden wir ein automatisch mit ILF bewiesenes Argument ausführlicher darstellen. Bei den anderen Argumenten geben wir nur an, welcher Theorembeweiser beim Beweisen erfolgreich war und wie lange er dafür gebraucht hat (auf einer Sun Sparc 2).

Wir gehen nun die einzelnen Beweisschritte nochmals durch und geben jeweils die für die formale Korrektheit zu überprüfenden Beweisaufgaben an.

3.1 S-Invarianten

S-Invarianten sind ein klassisches Konzept zur Verifikation von Eigenschaften von Petrinetzen. Üblicherweise werden S-Invarianten mit linear-algebraischen Techniken formuliert und verifiziert. Für den Einsatz in DAWN ist jedoch die Formulierung mit Hilfe von Termen zweckmäßiger [8, 11]. Um zu überprüfen, ob ein Term eine S-Invariante eines Petrinetzes ist, wird für jede Transition des Petrinetzes eine Gleichung generiert, deren Gültigkeit überprüft werden muß. Die Gleichung für einen Term u und eine Transition t ist $u_1 = u_2$, wobei u_1 aus u dadurch entsteht, daß für jedes Vorkommen eines Stellensymbols s in u die Inschrift der Kante von der Stelle s zur Transition t eingesetzt wird (und [] für die leere Multimenge, wenn keine solche Kante existiert); u_2 entsteht aus u durch entsprechende Ersetzung durch die Inschriften der ausgehenden

Kanten. Für den Beweisschritt (1) ergeben sich also die drei folgenden Beweisaufgaben (für jede Transition des Petrinetzes eine):

$$\begin{array}{lrcl} (1.1) & [x] + [] + [] & = & [] + [x] + [] \\ (1.2) & [] + [x] + [] & = & [] + [] + [x] \\ (1.3) & [] + [] + [x] & = & [x] + [] + [] \end{array}$$

Wenn diese drei Gleichungen gültig sind, ist der Term $p + q + r$ eine S-Invariante des Petrinetzes aus Abb. 1. Für die Überprüfung des Beweisschrittes (1) bleibt zu zeigen, daß der Wert von $p+q+r$ in der initialen Markierung tatsächlich A ist. Dies wird durch eine weitere Beweisaufgabe formalisiert:

$$(1.4) \quad A + [] + [] \quad = \quad A$$

Auf der linken Seite dieser Gleichung wird in $p + q + r$ jedes Stellensymbol durch die initiale Markierung der entsprechenden Stelle ersetzt; auf der rechten Seite steht der erwartete Wert.

Die Gleichungen (1.1)–(1.4) sind offensichtlich gültig; wir haben sie dennoch mit Hilfe von ILF automatisch überprüft, um den Aufwand dafür abschätzen zu können. Für alle Beweisaufgaben wurde ein Beweis von dem in ILF integrierten Gleichheitsbeweiser DISCOUNT [4] gefunden: (1.1) in 1.46 Sekunden, (1.2) in 0.77 Sekunden, (1.3) in 0.9 Sekunden und (1.4) in 1.24 Sekunden.

3.2 Abschwächung

Die Abschwächungs-Regel erlaubt es, eine bereits bewiesene Invariante $\Box\,\varphi$ abzuschwächen; d.h. durch $\Box\,\psi$ zu ersetzen, wenn ψ aus φ logisch folgt. Dazu muß die Gültigkeit der Implikation $\varphi \Rightarrow \psi$ überprüft werden.

Die Beweisaufgabe, die sich aus Zeile (2) des Beweises ergibt, ist also:

$$(2.1) \quad (p + q + r = A) \Rightarrow (p(x) \Rightarrow \neg q(x))$$

Bei der Überprüfung dieser Beweisaufgabe muß natürlich ausgenutzt werden, daß A eine Menge ist (d.h. daß jedes Element in A genau einmal vorkommt). Im allgemeinen müssen Annahmen über den zugrundeliegenden Datentyp, die im Petrinetzmodell des Algorithmus formuliert sind, dem automatischen Theorembeweisern in Form von Theorien zur Verfügung gestellt werden.

Ein Beweis für diese Beweisaufgabe wurde durch den in ILF integrierten Theorembeweiser SPASS [12] in 7.89 Sekunden gefunden.

3.3 Progress

In den Beweisschritten (3) und (4) haben wir ausgenutzt, daß eine Leadsto-Aussage unmittelbar durch das Schalten einer Transition t bewiesen werden kann. Die Regel dafür nennen wir *Progress-Regel*. Für die korrekte Anwendung der Regel (3) müssen die folgenden Beweisaufgaben übeprüft werden, wobei x, y und z Variablen der Sorte $AGENT$ sind:

$$\begin{array}{ll} (3.1) & (q + [x])(x) \\ (3.2) & p(x) \Rightarrow p \geq [x] \\ (3.3) & \neg \left(\begin{array}{l} p \geq [x] \wedge \\ p \geq [y] \wedge \\ \exists\, z : z \in [x] \wedge z \in [y] \wedge \\ \neg([x] = [y] \wedge [x] = [y]) \end{array} \right) \end{array}$$

Eine detaillierte Motivation für diese Beweisaufgaben können wir hier nicht geben; sie entstehen rein syntaktisch aus der Progress-Regel [11], wenn die Abkürzungen für *Zusicherungen*, *Aktiviertheit* und *Konflikte* aufgelöst werden (mit $\varphi = p(x)$, $\varphi' = q(x)$, $\chi_t = \mathit{false}$). Wir geben hier nur kurz die wesentliche Idee der einzelnen Beweisaufgaben an. Die Aussage (3.1) besagt, daß nach dem Schalten von $t1$ die Aussage $q(x)$ gilt; dies entspricht der Zusicherung[5] $\{\mathit{true}\}t1\{q(x)\}$. Die Aussage (3.2) besagt, daß die Transition $t1$ aktiviert ist, wenn $p(x)$ gilt; dies entspricht der Aussage $p(x) \Rightarrow \mathrm{ENABLED}(t1)$. Die Aussage (3.3) besagt, daß die Transition $t1$ mit keiner anderen Transition in Konflikt steht (auch nicht mit sich selbst in einem anderen Modus) — deshalb schaltet die Transition irgendwann.

Ein Beweis für die Aufgabe (3.1) wurde durch den in ILF integrierten Beweiser SETHEO [5] in 0.29 Sekunden gefunden. Der zugehörige von ILF aufbereitete Beweis wird in Abschnitt 4 diskutiert. Der Beweis für Aufgabe (3.2) wurde von SPASS in 11.95 Sekunden und der Beweis für Aufgabe (3.3) wurde in 1.49 Sekunden gefunden.

Die Beweisaufgaben für den Beweisschritt (4) sind ganz analog zu denen aus Schritt (3). Wir werden sie hier deshalb nicht betrachten.

3.4 Transitivität

Aus dem Beweisschritt (5) entstehen keine weiteren Beweisaufgaben, die mit Hilfe eines Theorembeweisers überprüft werden müßten. Es ist lediglich die syntaktisch korrekte Anwendung der Regel für die Transitivität von Leadsto-Aussagen zu überprüfen:

$$\frac{\varphi \rhd \psi \qquad \psi \rhd \chi}{\varphi \rhd \chi}$$

Für $\varphi = p(x)$, $\psi = q(x)$ und $\chi = r(x)$ entsprechen die als Voraussetzung angegebenen Zeilen (4) und (5) genau den Voraussetzungen der Regeln und die Zeile (5) der Schlußfolgerung der Regel.

3.5 Zusammenfassung

In diesem Abschnitt haben wir gezeigt, wie man die korrekte Anwendung der Beweisregeln von DAWN auf das Überprüfen von Zustandsaussagen zurückführen (bzw. rein syntaktisch überprüfen) kann; wie diese Beweisaufgaben mit Hilfe von ILF automatisch überprüft werden können, werden wir im folgenden Abschnitt anhand der Aufgabe (3.1) zeigen.

In DAWN gibt es natürlich wesentlich mehr Regeln als in unserem Beispiel vorkommen; die zugehörigen Beweisaufgaben besitzen aber immer eine ähnliche Charakteristik — auch wenn manche Regeln deutlich komplizierter sind als die hier dargestellten Regeln. Die hier beschriebenen Beweisaufgaben stellen damit eine gute Schnittstelle zwischen DAWN und ILF dar.

Im Hinblick auf ein Werkzeug zum interaktiven Beweisen von verteilten Algorithmen ist noch ein weiterer Frage wichtig: Was passiert, wenn der Theorembeweiser (bzw. ILF) die Korrektheit einer Beweisaufgabe nicht in vertretbarer Zeit nachweisen kann — bzw. die Beweisaufgabe falsifiziert? In diesem Falle muß der Benutzer darauf hingewiesen werden, daß er möglicherweise einen Fehler gemacht hat. Da jede der generierten Beweisaufgaben eine intuitive Interpretation besitzt, können wir dem Benutzer gezielte Hinweise zur Korrektur des Beweises — bzw. des

[5] Streng genommen müßte hier sogar $\{\mathit{true}\}t1, \mathit{id}\{q(x)\}$ stehen, wobei id den Modus beschreibt, in dem die Transition schaltet. Dies ist aber in dieser Vorstudie nicht von Bedeutung.

Algorithmus' — geben. Wenn beispielsweise Aufgabe (1.1) nicht bewiesen werden kann, ist klar, daß die Transition $t1$ die S-Invariante verletzt. Wenn Aufgabe (3.2) nicht bewiesen werden kann, ist die Transition $t1$ nicht in allen Fällen aktiviert und kann deshalb nicht in allen gewünschten Fällen schalten. Diese Möglichkeit der „Rückinterpretation" der Beweisaufgaben ist für einen interaktiven Beweiser sehr wichtig.

4 Nachweis der Teilaufgaben mit Hilfe von ILF

Die in Abschnitt 3 generierten Teilaufgaben (1.1)–(3.3) behaupten Eigenschaften über allgemein bekannte, mathematische Objekte wie Mengen, Multimengen, natürliche Zahlen. Der Nachweis solcher Behauptungen verlangt lediglich die Kenntnis mathematischer Zusammenhänge, der DAWN-Kalkül mit seinen Schlußregeln spielt hierfür keine Rolle.

Als Grundlage für die Arbeit der automatischen Theorembeweiser kann deshalb eine allgemeine Theorembibliothek aus der Mathematik dienen. Wir haben die mathematische Bibliothek der Spezifikationssprache Z, Mathematical Tool-kit, zur formalen Grundlage unserer Beweise von (1.1)–(3.3) gemacht. Alle in den automatisch gefundenen Beweisen als Axiom genutzten Voraussetzungen entstammen dem Tool-kit.

Wir präsentieren zunächst exemplarisch den von SETHEO gefundenen Beweis von (3.1), wie er in aufbereiteter Form von ILF an den Nutzer geliefert wird. Danach gehen wir auf offene sowie schon gelöste Probleme ein, die beim Umgang mit automatischen Beweisern auftreten können.

Axiom 4.1 $(def(\uplus))$ $\forall Par\ \forall Bag\ \forall Bag_1\ \forall Z_1\ Bag \uplus Bag_1 \sharp Z_1 = Bag \sharp Z_1 + Bag_1 \sharp Z_1$.

Axiom 4.2 $(prop(=))$ $\forall S_1\ \forall S_2\ S_1 = S_2 \rightarrow S_2 = S_1$.

Axiom 4.3 $(refl(\leq))$ $\forall N\ N \leq N$.

Axiom 4.4 $(def([\ldots]))$ $\forall N_1\ \forall N_2\ \forall N_3\ [N_1, \ldots, N_2]\sharp N_3 = 1 \leftrightarrow N_1 \leq N_3 \wedge N_3 \leq N_2$.

Axiom 4.5 $(one\_greater\_zero)$ $1 > 0$.

Axiom 4.6 $(prop(=))$ $\forall Z_1\ \forall Z_2\ \forall N_1\ Z_1 = Z_2 \rightarrow (Z_1 > N_1 \rightarrow Z_2 > N_1)$.

Axiom 4.7 $(prop(+))$ $\forall N\ \forall N_1\ N_1 > 0 \rightarrow N + N_1 > 0$.

Axiom 4.8 $(def(bagelement))$ $\forall Par\ \forall Bag\ \forall Z_1\ (Bag)(Z_1) \leftrightarrow Bag \sharp Z_1 > 0$.

Theorem 4.1 $(task3.1)$ $\forall Z_1\ (q \uplus [Z_1])(Z_1)$.

Proof[6]. We show directly that

$$(q \uplus [c_1])(c_1). \tag{1}$$

Because of $def(\uplus)$ and by $prop(=)$ $q\sharp c_1 + [c_1]\sharp c_1 = q \uplus [c_1]\sharp c_1$. Because of $refl(\leq)$ and by $def([\ldots])$ $[c_1]\sharp c_1 = 1$. Hence by $prop(=)$ $1 = [c_1]\sharp c_1$. Hence by $prop(=)$ and since $1 > 0$ $[c_1]\sharp c_1 > 0$. Hence by $prop(+)$ $q\sharp c_1 + [c_1]\sharp c_1 > 0$. Therefore by $prop(=)$ $q \uplus [c_1]\sharp c_1 > 0$. Hence by $def(bagelement)$ $(q \uplus [c_1])(c_1)$. Thus we have completed the proof of (1).

[6] SETHEO and ILF

Prinzipiell sind folgende Etappen von der gestellten Aufgabe bis zum aufbereiteten Beweis bei der maschinellen Verifikation zu bewältigen:

Überführung der Aufgaben in das Format der benutzten Beweiser

Die im Tool-kit in Z-Syntax aufgeführten Axiome müssen zunächst in ungetypte Formeln der Prädikatenlogik 1. Stufe übersetzt werden, da momentan automatische Theorembeweiser mit getypter Logik nicht umgehen können. Dieser Transformationsprozeß geschieht in zwei Stufen. Relativ problemlos ist eine Transformation benötigter Z-Ausdrücke in eine getypte prädikatenlogische Sprache zu bewerkstelligen; bei uns fungiert die ILF-Eingabesprache als Zielsprache. Eine nähere Betrachtung zu diesem Thema findet sich in [1]. Für die noch offene Überführung von getypter in ungetypte Prädikatenlogik stehen innerhalb von ILF mehrere Routinen zur Verfügung, die auf Termcodierungs- bzw. Relativierungsverfahren [9] beruhen.

Die Umsetzung der so erhaltenden ungetypten Theorie in Eingabefiles der einzelnen Beweiser kann danach vollautomatisch von ILF vorgenommen werden. Zu beachten ist hier, daß ILF dabei noch weitere Transformationsschritte (Klausifizierung, Ergänzung logischer Axiome) vornimmt, um den durchaus vielfältigen Anforderungen der einzelnen Beweiser an das Format ihrer Eingabefiles gerecht zu werden.

Auswahl einer relevanten Teiltheorie

Ein großes, noch ungelöstes Problem in der Deduktion ist die Einschränkung der relevanten Theorie. Bei unseren Beispielen war eine manuelle Theorieauswahl nötig, damit der Suchraum der automatischen Beweiser nicht explodiert. Alle im Abschnitt 3 zitierten Zeiten für die Beweissuche beziehen sich auf Beweisversuche mit optimaler Theorieeinschränkung.

Aufbereitung des Beweises

Eine nutzergerechte Aufbereitung der Beweise ist nach unserer Erfahrung sehr wichtig, nicht zuletzt wegen der Möglichkeit, auf diese Weise eventuell vorhandene Inkonsistenzen innerhalb des DAWN-Beweises aufzuspüren.

Der ILF-Nutzer kann durch Angabe sogenannter Tex-Ops die Darstellung von Beweisstrukturen und Symbolen der Theorie seinen Vorstellungen anpassen; für eine eingehende Beschreibung der dabei angewandten Techniken verweisen wir auf [2]. Die hier dargebotene Aufbereitung orientiert sich an der Symbolwahl, wie sie im Tool-kit der Sprache Z üblich ist. Insbesondere wurde hier als Symbol zur Multimengenvereinigung $\uplus$ anstatt $+$ gewählt.

Da es sich bei dem aufbereiteten Beweis um einen maschinell gefundenen und formalen handelt, wird es trotz guter Tex-Ops dem Nutzer allgemein schwerfallen, diesen Beweis genau zu verfolgen und wirklich zu verstehen.

Der Nutzer kann sich aber sehr schnell und ohne nennenswerten Aufwand davon überzeugen, welche Axiome für einen formalen Beweis denn wirklich benötigt werden. Sollte beispielsweise die Axiomenmenge leer sein, so ist dies ein ernstzunehmender Hinweis darauf, daß die Behauptung widersprüchlich ist und sich Fehler in die Spezifikation eingeschlichen haben.

5 Ausblick

In dieser Vorstudie haben wir anhand eines einfachen Beispiels gezeigt, wie man mit Hilfe von ILF und DAWN die Korrektheit eines verteilten Algorithmus' intuitiv und dennoch vollständig formal nachweisen kann. Alle in den Abschnitten 3 und 4 beschriebenen Aufgaben können dabei von einem Werkzeug automatisch durchgeführt werden. Da jede einzelne Beweisaufgabe eine intuitive Interpretation besitzt, ist eine Interaktion mit dem Anwender auf der intuitiven Ebene möglich (beispielsweise, beim Hinweis auf Fehler in DAWN-Beweisen). Ein solches interaktives Werkzeug werden wir in einem längerfristigen Projekt entwickeln — das Ziel dieser Vorstudie war der Nachweis der Realisierbarkeit dieses Werkzeugs.

Bei der Realisierung dieses Werkzeugs müssen die folgenden Aufgaben bearbeitet werden:

- Wahl der richtigen Abstraktionsebene für die Beweisregeln von DAWN, so daß alle daraus resultierenden Beweisaufgaben von einem Theorembeweiser automatisch bewiesen werden können.
- Automatisierung der Generierung von Beweisaufgaben aus einem Beweisschritt und dem zugehörigen Petrinetz und Zuordnung einer intuitiven Interpretation zu jeder Beweisaufgabe.
- Zusammenstellung von Theorien (insbes. für Mengen und Multimengen), die das vollautomatische Beweisen der Beweisaufgaben erlauben.
- Entwicklung von geeigneten Strategien für die Theorembeweiser und ggf. Auswahl der am besten geeigneten Theorembeweiser für bestimmte Klassen von Beweisaufgaben.

Diese Aufgaben müssen zunächst anhand einer etwas größeren Fallstudie (z.B. des Echo-Algorithmus' [7]) von Hand durchgeführt werden und dann verallgemeinert werden. Dann kann die Generierung der Beweisaufgaben automatisiert werden. Später können die Beweisschritte in DAWN in dem Maße vergrößert werden (und damit die Kompliziertheit der zugehörigen Beweisaufgaben), wie die Theorien und Strategien für die Theorembeweiser verbessert und an die spezifischen Erfordernisse in diesem Werkzeug angepaßt werden.

Mit dem so entstandenen Werkzeug können verteilte Algorithmen interaktiv bewiesen werden; das Finden der zentralen Beweisidee und ihre Formulierung in der temporalen Logik von DAWN bleibt aber weiterhin dem Benutzer vorbehalten.

Literatur

[1] T. Baar Formaler Beweis der Korrektheitseigenschaften einer Z - Spezifikation des Alternating-Bit-Protokolls unter Benutzung des Werkzeuges ILF. Diplomarbeit, Humboldt-Universität zu Berlin, 1997.

[2] B. I. Dahn, A. Wolf: Natural Language Presentation and Combination of Automatically Generated Proofs. In Proc. FroCoS'96, pp. 175- -192, Kluwer, 1996.

[3] B. I. Dahn, J. Gehne, Th. Honigmann, A. Wolf: Integration of Automated and Interactive Theorem Proving in ILF. In Proc. CADE-14, pp. 57-60, Springer, 1997.

[4] J. Denzinger, M. Kronenburg, S. Schulz: DISCOUNT. A Distributed and Learning Equational Prover. In Journal of Automated Reasoning 18(2), pp. 189-198, 1997.

[5] C. Goller, R. Letz, K. Mayr, J. Schumann: SETHEO V3.2: Recent Developments (System Abstract). In Proc. CADE-12, pp. 778–782, Springer, 1994.

[6] Ekkart Kindler and Wolfgang Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, 51:16–31, December 1996.

[7] Ekkart Kindler, Wolfgang Reisig, Hagen Völzer, and Rolf Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Computing*, 9:409–424, 1997.

[8] Ekkart Kindler and Hagen Völzer. Flexibility in algebraic nets. Informatik-Bericht 89, Institut für Informatik, Humboldt-Universität zu Berlin, November 1997. Erscheint in Proceedings von ICATPN '98, Springer-Verlag, Juni 1998.

[9] Mellish, C. S.: Implementing Systemic Classification by Unification. Comp. Ling. 14, 1988, pp. 40–51

[10] Wolfgang Reisig. *Elements of Distributed Algorithms — Modeling and Analysis with Petri Nets*. Springer-Verlag, 1998.

[11] M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker, E. Kindler, J. Freiheit, and J. Desel. DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 88, Humboldt-Universität zu Berlin, December 1997.

[12] C. Weidenbach, B. Gaede, G. Rock: Spass & Flotter, Version 0.42. In Proc. CADE-13, pp. 141–145, Springer, 1996.