Abschlußbericht

eXpander — Eine graphische Benutzeroberfläche für Expander für das X Window System

Markus Heller
heller@ls5.cs.uni-dortmund.de

22. Oktober 1998

Lehrstuhl Informatik 5
Fachbereich Informatik
Universität Dortmund
Baroper Straße 301
44221 Dortmund
http://www.ls5.cs.uni-dortmund.de

Zusammenfassung

Dieser Abschlußbericht beschreibt die Ergebnisse und den Verlauf der Arbeit bei der Entwicklung einer graphischen Benutzeroberfläche für die Expander-Umgebung sowie die Erweiterungsmöglichkeiten für das System, die im Laufe der Zeit sichtbar wurden..

Expander ist eine Beweis- und Test-Umgebung für algebraische Spezifikationen und funktional-logische Programme. Programm-Design, -Test und -Verifikation werden unterstützt durch spezielle Deduktions-Regeln für das Beweisen von Gentzen-Klauseln. Diese Formeln werden am häufigsten benutzt, um Anforderungen und Anfragen an Datentypen zu formulieren. Induktive Beweise werden aufgebaut aus Vereinfachungen, Folgerungen, Resolutionen und sog. Narrowing-Schritten [Pad91].

Für das in der Programmiersprache StandardML implementierte Expander-System wurde zunächst eine Benutzeroberfläche für das X Window System unter Unix realisiert und in weiteren Schritten darauf aufbauend Funktionserweiterungen ergänzt. Für die Realsierung der Benutzeroberfläche wurde die Sprache Tcl/Tk gewählt.

Danke!

An dieser Stelle möchte ich mich bei Peter Padawitz, Andreas Hallmann, Oliver Niese und Michael Rexhäuser noch einmal herzlich bedanken für die interessante und schöne Zeit am Lehrstuhl. Ich werde neben den anregenden Diskussionen und der gemeinsamen Arbeit auch den Spaß vermissen, den wir hatten!

Markus Heller

Inhaltsverzeichnis

Einleitung

In diesem Bericht wird ein Überblick über das Projekt "eXpander" gegeben, in dessen Rahmen eine graphische Benutzeroberfläche für das bestehende Beweis- und Test-Umgebung "Expander" entwickelt wurde.

Zunächst werden in Kapitel 1 und 2 die Aufgabenstellung und der Verlauf der Arbeit beschrieben. Im dritten Kapitel wird erläutert, welches Modell für den Dialog mit dem Benutzer der Oberfläche benutzt wird. Das Kapitel 4 stellt den Entwurf des Systems dar, hier wird auch auf einige Besonderheiten des Entwurfs eingegangen. Ergänzend wird hier auch die Fixpunktinduktion beschrieben.

Kapitel 5 und 6 beschreiben die Implementierung des entworfenen Systems und die Installationshinweise, um das eXpander-System zu starten.

Im Verlauf der Arbeit enstand eine Vielzahl von Möglichkeiten der Erweiterung von eXpander. Diese Ideen werden in Kapitel 7 zusammengestellt.

Im Anhang werden der Quellcode des eXpander-Systems und zusätzliche Programmtexte angegeben.

Um an einigen Stellen auf Detailfragen eingehen zu können, wird dort mitunter vorausgesetzt, daß der Leser im Umgang mit den Kommandos des Expander-Beweissystems vertraut ist (vgl. [Pad96]).

1 Aufgabenstellung

Am Lehrstuhl Informatik 5 der Universität Dortmund wurde durch Prof. Dr. Peter Padawitz die Beweis- und Test-Umgebung "Expander" für algebraische Spezifikationen und funktional-logische Programme entwickelt. Ebenfalls wurde eine Benutzeroberfläche für das Betriebssystem NeXtStep der Firma Next entworfen.

Es sollte in einem ersten Schritt für das Expander-System eine graphische Benutzeroberfläche "eXpander" für das X Windows System für das Betriebssystem Unix (Linux) entwickelt werden. In diesem Rahmen sollte zunächst ein geeignetes Modell für den Dialog mit dem Benutzer gefunden werden und anschließend implementiert werden.

2 Verlauf der Arbeit

Um diese Anforderungen zu erfüllen, wurden zunächst mehrere Möglichkeiten diskutiert, den Benutzer-Dialog abzuwickeln. Nachdem ein Modell ausgewählt worden war, wurden ein Prototyp und zwei aufeinanderfolgende Versionen des eXpander-Systems erstellt.

Der Prototyp sollte dazu dienen, die Belastbarkeit der Kommunikationsverbindung von dem zugrundeliegenden Beweissystem zu der Oberfläche zu testen und die grundsätzliche Machbarkeit des Modells für den Dialog mit dem Benutzer zu prüfen.

Nach der Version 1.0 wurde in der darauffolgenden Version das Dialog-Modell vereinfacht, schnellere Algorithmen implementiert (insbesondere für die Verwaltung der Eintrags-Assoziationen), Fixpunktinduktion erarbeitet und ein Hilfe-System integriert. Darüberhinaus wurde auch ein neues Modell für die Menü-Steuerung durch Oktagonale Pop-Up-Menüs entworfen und getestet. Diese Idee wurde jedoch wieder verworfen, da diese Steuerung sich als sehr gewöhnungsbedürftig herausstellte.

3 Benutzer-Sicht auf das eXpander-System

Innerhalb des Expander-Beweissystem wird versucht, Beweise ausgehend von Anfangs- und Zielformel gleichzeitig aufeinander hin zu führen. Dazu stehen ihm eine Menge von Kommandos uir Verfügung, mit dessen Hilfe er auf einer der beiden Beweisteile arbeiten kann unter Zuhilfenahme von Lemmata, deren Gültigkeit vorausgesetzt, schon beweisen oder vermutet wird. Ziel ist es, beide Beweisteile miteinander zu verbinden. Die Kommandos sind vorwärts und rückwarts anwendbar, falls es also gelingt, beide Beweisteile zusammenzuführen, hat man einen Beweis gefunden, daß die Zielformel eine Implikation für die Anfangsformel ist.

3.1 Gliederung der Oberfläche

Die Gliederung der eXpander-Oberfläche folgt dieser Idee, sie gliedert sich grob in Menüzeile ganz oben, den darunterliegenden Beweisteil, das Kommando-Toolkit, Meldungsfenster und Spezifikations-Bereich.

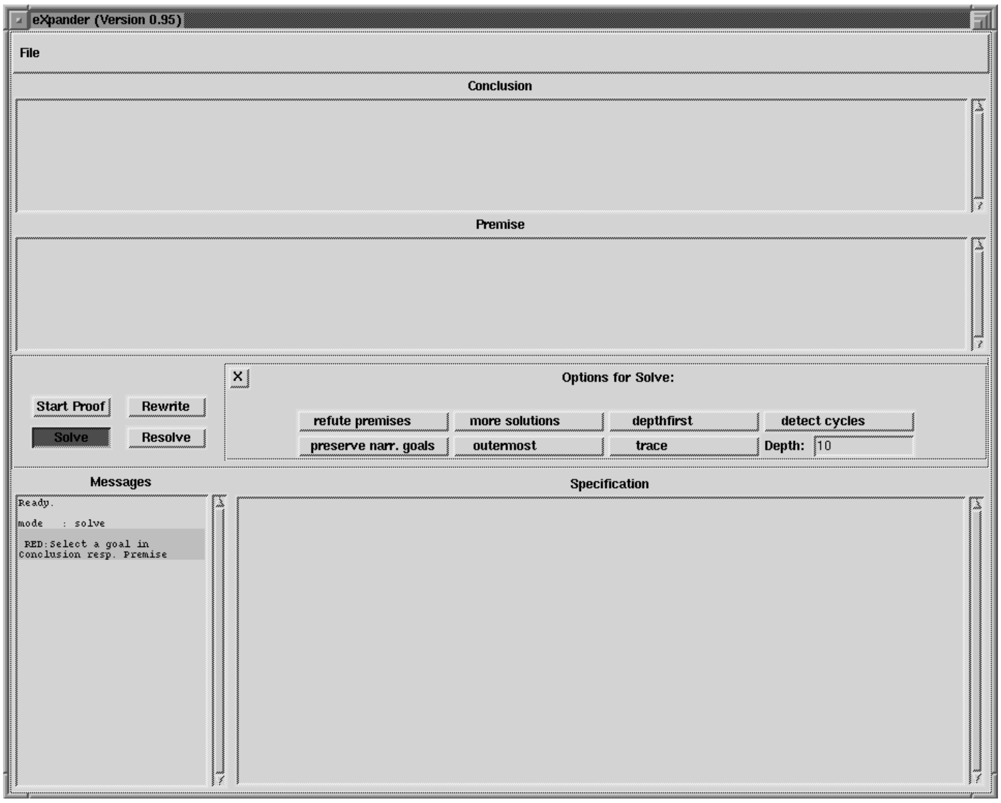

Figure 1. Die eXpander-Umgebung

In der Menüzeile werden Basisfunktionen angeboten, mit denen eine Spezifikation geladen werden oder mehrere Arten von Ergebnissen eines Beweises erstellt werden können.

Im Beweisteil sind Bereiche für die Prämisse und die Konklusion des aktuellen Beweises zu sehen. Hier werden die Teile des Beweises ausgewählt, die Argument für ein Expander-Kommando darstellen sollen.

Im Kommando-Toolkit werden die Befehle von Expander zur Verfügung gestellt. Grundsätzlich gibt es einfache und komplexere Kommandos, die unterschiedlich ausgeführt werden. Zum einen gibt es Kommandos, die gar keine Argumente erfodern (z.B. Backtrack-Befehle oder der Befehl "proved"). Diese werden einfach durch Druck auf den jeweiligen Button ausgeführt. Zum anderen gibt es auch Befehle, die eigene Argumente erfordern, z.B. "Rewrite" oder "Resolve" In diesem Fall wird zunächst das gewünschte Kommando aktiviert und falls dafür zusätzlich einstellbare Optionen verfügbar sind, werden diese zeitweilig in diesem Bereich eingeblendet. Nach Einstellung der Optionen und Markierung der gewünschten Argumente in Beweisteil und/oder Spezifikationsteil wird mit erneutem Druck auf denselben Button das aktivierte Kommando ausgeführt. Wie Argumentmarkierungen durchgeführt werden wird weiter unten beschrieben.

Über das Meldungsfenster kommuniziert das System mit dem Benutzer, u.a. die Hilfestellungen für die Auswahl von Beweisteilen bei der Kommando-Ausführung erscheinen hier ebenso wie Fehlermeldungen, die unverändert vom zugrundeliegenden Expander-Beweissystem stammen.

Im Spezifikations-Bereich wird die aktuell geladene Spezifikation dargestellt, in der die Axiome, Theoreme und Vermutungen enthalten sind.

3.2 Markierung von Argumenten eines Kommandos

Die Mehrzahl der Kommandos von Expander erfordern die Markierung von Objekten, dies können Terme, Atome, Goals oder ganze Klauseln sein. Der Benutzer unterscheidet diese Argumente weiter in Objekte, die ersetzt werden, in Objekte, die etwas ersetzen bzw. für das Finden einer Ersetzung herangezogen werden und schließlich noch in Objekte, die sich in anderen Teilen des aktuellen Schrittes nocheinmal finden lassen.

3.2.1 Markieren von Argumenten in einem Schritt

Um alle unterschiedlichen Arten von Argumenten in einem einzigen Schritt zu markieren, können diese auf drei unterschiedliche Arten markiert werden (Dies wird dem Benutzer am Bildschirm durch verschiedene Cursor-Typen bzw. Farben symbolisiert).

Meist werden die Objekte mit dem Pfeil-Cursor (in roter Farbe) markiert.

Zusätzlich können Objekte in grüner Farbe (Cursor "Sprühdose") markiert werden, dies sind dann meist zu ersetzende Terme. Darüberhinaus werden Paarungen von Objekten vorgenommen, wenn der Cursor in blauer Farbe ein Doppelpfeil ist, um die Korrespondenz anzuzeigen.

Dieser Ansatz wird in der aktuellen eXpander-Version verfolgt. Die genaue Art und Weise, wie zwischen den Cursortypen gewechselt werden kann und wie Paarungen von Objekten im detail vor sich gehen, sind dem eXpander-Manual zu entnehmen (vgl. [Hel97]).

3.2.2 Markierungen von Argumenten in mehreren Schritten

Falls für ein Kommando mehrere unterschiedliche Arten von Argumenten zu markieren sind, so können diese Argumente auch in mehreren Teildialogen festgelegt werden (Dies gilt für die Funktionen *Resolve*, *Rewrite* und *replace*).

Für die Funktion *Replace* ergibt sich:

- Dialog 1: "Ersetzungen"
  Hier werden bis zu vier Vorkommen eines Terms, der ersetzt werden soll, in einem Goal der Konklusion/Prämisse markiert.
- Dialog 2: "Ersetzer"
  In denselbem Goal wird eine Gleichung markiert, eine ihrer Seiten muß den zu ersetzenden Term matchen.

Für die Funktion *Rewrite* käme noch ein weiterer Dialog hinzu:

- Dialog 1: "Ersetzungen"(immer erforderlich)
  Hier werden ein oder mehrere Vorkommen eines Terms, der ersetzt werden soll, in einem Goal der Konklusion/Prämisse markiert.
- Dialog 2: "Ersetzer" (nicht bei *RewriteFunction* erforderlich)
  In einem Lemma, das ebensoviele Goals in der Konklusion enthält wie in Dialog 1 markiert wurden, wird in jedem dieser Konklusion-Goals eine Gleichung markiert. Eine Seite muß bei allen Gleichungen identisch sein, die andere muß jeweils mit einem zu ersetzenden Term aus Dialog 1 unifizierbar sein.
- Dialog 3: "Voraussetzungen" (nur bei Unterfunktion *RewritePremise* erforderlich)
  Für alle Prämissen-Atome aus Dialog 1 wird im benutzen Lemma wird nun ein Atom der Prämisse markiert, welches unifizierbar ist.

Die Funktion *Rewrite* läßt sich auf ähnliche Weise in zwei Dialogen realisieren (“Ersetzungen”, “Konnektor-Atom”)

4 Entwurf

In diesem Kapitel wird zunächst ein grafischer Überblick über die Struktur des entworfenen eXpander-Systems gegeben.

Exemplarisch werden dann Kernstücke des Entwurfs beschrieben, dies sind *Entry*-Datentyp, *Entry Association* und *TagSet*.

4.1 Klassenstruktur für das eXpander-System

Das eXpander-System gliedert sich zunächst grob in twei Teile, von denen einer in SML implementiert wurde und eer andere in TCl/Tk.

Der Teil der eXpander-Anwendung, der in TCl/Tk geschrieben ist, wird grafisch als Sammlung von Klassendiagrammen dargestellt, aus denen sich die Struktur ablesen läßt.

Für den SML-Teil der Anwendung werden ebenfalls Diagramme dargestellt, hierbei ist insbesondere zu beachten, daß es sich bei den dargestellten Moduln um Strukturen von SML und *nicht* um Klassen im objektorientierten Sinne handelt.

4.1.1 Tcl/Tk-Teil

Der Tcl/Tk-Teil ist in diesem Überblick objektorientiert in Klassen eingeteilt, um einen Überblick über das eXpander-System zu bieten. Da Tcl/Tk nicht objektorientiert ist, kann diese Struktur nur lose direkt in die Implementierung übertragen werden.

Neben dem Hauptprogramm gliedert sich dieser Teil in drei Pakete, das *Modell* mit der Anbindung an das Expander-Beweissystem und aktuellem Zustand des Beweises, den *View*-Teil, der die grafische Benutzeroberfläche bereitstellt und schließlich den *Control*-Teil, der hier jedoch lediglich Hilfsfunktionen bereitstellt wie das Initialisieren der Kommunikationsverbindung zum Expander-Beweissystem und weiter die Funktionalität, um Benutzer-Präferenzen einstellen zu können. Im traditionellen MVC-Ansatz der Anwendungserstellung sind hier die Event-Kontrollfunktionen bereits im View-Teil integriert, um die Übersichtlichkeit im TCl/Tk-Code nicht künstlich zu zerstören.

Interessant im *View*-Teil ist die Klasse *window*. Die Fenster, in denen Beweisteile und Spezifikationsteil enthalten sind, werden durch *hyper.hyperText* bereitgestellt (mitsamt Hilfsfunktionen *hyper.** zum Löschen, Markieren, Konfigurieren dieser Fenster.

Zentrale Klasse im View ist *window*, die das Hauptfenster zusammen mit der angebundenen Funktionalität enthält (Methode *window.main*.

Marken in Fenstern werden durch den Datentyp *Marke* realisiert, in der Klasse *highlight* sind alle Funktionen untergebracht, die zum Markieren, für den Wechsel von Cursorfarbe/-form und die Fehlermarkierung benötigt werden.

4.1.2 SML-Teil

Da auch SML nicht objektorientiert ist, dienen diese Diagramme nur dazu, einen Überblick über das eXpander-System zu bekommen.

Neben dem Code, der vom eigentlichen Beweissystem "Expander" von Prof. Dr. Padawitz stammt und der im *Modell*-Teil enthalten ist, sind dort auch die Ausführungscodes für die von der Oberfläche angeforderten Funktionsaufrufe angesiedelt. Wenn die Oberfläche einen Expander-Befehl aufruft, müssen noch besondere zusätzliche Schritte abgearbeitet werden, um insbesondere automatische Fehlermarkierung und Aktualisierungen der beteiligten Bildschirmfenster zu erledigen. Dies geschieht in der Struktur *Ecommands*. Die Struktur *actualproof* enthält die für die Oberfläche wichtigen Statusvariablen des Expander-Systems.

Im *View* sind die Strukturen untergebracht, die für die Verwaltung der Assoziation "Syntaktisches Objekt $\leftrightarrow$ Bildschirmmarkierung" benötigt werden. Benutzt wird hierbei u.a. der Funktor *association*, der eine Datenstruktur zur Verfügung stellt, um Assoziationen zu verwalten. Weiterhin sind im View auch Strukturen für die Verwaltung der in den verschiedenen Markierungsmodi ausgewählten Objekte enthalten.

Im *Control*-Teil stehen die Anbindung zur grafischen Oberfläche (*tk*), einfache Hilfsfunktionen (*EComAux*) (sowie eine Debug-Struktur, die bei der Fehlersuche benutzt wird).

4.2 Entries

Während der Festlegung der Argumente eines Expander-Kommandos markiert der Benutzer im Beweisteil und im Spezifikationsteil einzelne Objekte (Terme, Atome, Goals, Klauseln). Da der Oberfläche vom Expander-System nur Text-Ströme übermittelt werden, ist es notwendig, einzelnen Stücken im Textstrom, z.B. für eine Spezifikation, markierbare Objekte im entsprechenden Bildschirmfenster zuzuordnen. Der Datentyp *Entry* realisiert dies im eXpander-System.

Er enthält Anfangs-/Endkoordinaten im Bildschirmfenster und die Angaben, welches syntaktische Objekt im Expander-System sich dieser markierte Bildschirmbereich darstellt. Dies sind im Wesentlichen die sog. Ebene (LEVEL: Klausel, Goal, Atom, Term), die Menge (SET: Axiome, Theoreme, Vermutungen, aktuellerBeweisteil), die Seite (SIDE: Konklusion, Prämisse,...) und die konkreten Nummern der Klausel, des Goals, des Atoms und die Nummernliste für die Bestimmung des Termes innerhalb eines Atoms.

```
datatype SIDE = CONCL | PREM (..)
datatype SET  = AXM | THM | COJ | PROOF
datatype LEVEL= CLAUSE | GOAL | ATOM | TERM
datatype CURSORCOLOR= RED | GREEN | BLUE

type Entry= (SelectionStart*SelectionEnd)
                 * LEVEL * SET * SIDE
                 * ClauseNo * GoalNo * AtomNo * TermNo list
                 (..)
```

4.2.1 Abstraktes Datenobjekt *Entry Assoziation*

Für Beweisteil und Spezifiaktionsteil wird eine gemeinsame Assoziation gepflegt, die eine Zuordnung von syntaktischem Objekt und Bildschirmbereich verwaltet. Bei jeder Veränderung im Komklusions-, Prämissen- oder Spezifikationsfenster wird der zugehörige Teil in dieser Assoziationsmenge aktualisiert. Hier werden auch Operationen *findBestHint* zum Finden des syntaktischen Objektes, daß mit dem markierten Bildschirminhalt am besten übereinstimmt oder die Operation *findPossibleHints* zum Finden aller möglichen passenden syntaktischen Objekte.

4.2.2 Abstrakte Datenobjekte *TagSet*

Der Benutzer kann in bis zu drei unterschiedlichen Farben Objekte am Bildschirm markieren. Für die Verwaltung dieser markierten Objekte werden drei sog. *TagSets* benutzt, die eine Menge von Objektmarkierungen (tags) vom Datentyp Entry mit Operationen *insert, clear, read* und *write* bearbeiten können.

4.3 Fixpunktinduktion

Um Beweise über Fixpunktinduktion zu vereinfachen, wurde eine neue Funktion *Induce.fixpoint* entworfen. Damit sollen für gegebenes logisches Programm und eine Korrektheitsbedingung dem Benutzer die kanonischen Schritte der Einführung eines Hilfsprädikats und der Formulierung einer neuen Korrektheitsbedingung abgenommen werden.

Die Fixpunktinduktion kann angewendet werden, um ein gegebenes logisches Programm auf Korrektheit zu prüfen.

Um die Anwendung zu veranschaulichen und an dieser Anwendung die notwendigen Schritte besser zu erkennen, wird die Fixpunktinduktion am Beispiel der Berechnung aller Partitionen einer Liste, die in der Spezifikation PARTITION beschrieben ist, durchgeführt.

4.3.1 Beispiel: Spezifikation PARTITION

```
PARTITION

functs   flatten 4 5  >> 6
preds part 1 2 3
vars     x y L L' P P'

theorems
(1) { x::L = [x]@L }
(2) { x::(L@L') = (x::L)@L' }


axioms
(1) { part([x]    , [[x]]     ) }
(2) { part(x::y::L, [x]::P    ) }   <== { part(y::L,P) }
(3) { part(x::y::L, (x::L')::P) }   <== { part(y::L,L'::P) }

(4) { flatten(nil)  = nil         }
(5) { flatten(L::P) = L@flatten(P) }

(6) { (x::L,P) >> (L,P') = true }

conjects
(1) { L = flatten(P) } <== { part(L,P) }
(2) { part([1,2,3,4],P) }
```

4.3.2 Voraussetzungen

Um Fixpunktinduktion anwenden zu können, darf in den Axiomen der betrachteten Funktion *(hier: "part")* nur der Aufruf eines einzigen Prädikates vorhanden sein.

4.3.3 Anwendung

1. Die Korrektheitsbedingung: $L \equiv flatten(P) \Longleftarrow part(L, P)$

   Man führt ein Hilfsprädikat ein, um die Übersichtlichkeit zu bewahren, welches folgendermaßen beschrieben ist:

   ```
   { HELP(L,P) } <== { L = flatten(P) }
   ```

Die neu formulierte Korrektheitsbedingung lautet also:

```
{ HELP(L,P) } <== { part(L,P) }
```

2. Die Axiome für HELP sind leicht ableitbar, sie entstehen aus den Axiomen für "part" durch Ersetzen aller Vorkommen von "part" durch "HELP":

```
(1) { HELP ([x]    , [[x]]     ) }
(2) { HELP (x::y::L, [x]::P    ) }  <== { HELP (y::L, P    ) }
(3) { HELP (x::y::L, (x::L')::P) }  <== { HELP (y::L, L'::P) }
```

3. Ergänzen der Spezifikation PARTITION in conjectures:
   (a) ergänze Hilfsprädikats-definition
   (b) ergänze neue Korrektheitsbedingung
   (c) ergänze die Axiome für HELP

4.3.4 Funktionsdefinition

Um zukünftige Induktionsregeln vorzubereiten, wird eine **neue Strukur** mit Namen **"Induce"** . Erweiterungen für Induktions-Beweise werden sämtlich hier abgelegt.

Die Funktion wird wie folgt definiert[1]:

Name: Induce.fixpoint

Eingabe: Die Nummer der Conjecture, welche die Korrektheitsbedingung darstellt.

Ergebnis: Conjectures wird ergänzt um

1. die Definition des Hilfsprädikats
2. die **neue** Korrektheitsbedingung
3. die Axiome für das Hilfsprädikat um Axiome für HELP

[1] vgl [Pad97] S.59f

4.3.5 Beispiel-Sitzung mit EXPANDER

```
- parse "PARTITION";

PARTITION is syntactically correct
val it = () : unit

- start 1;

initial conclusion:
 (1) { L = flatten(P) }
initial premise:
 (1) { part(L,P) }
val it = () : unit

- induceFix 1;
the conjectures
      (3) { partHELP(L,P) }  <== { L = flatten(P) }
      (4) { partHELP(L,P) }  <== { part(L,P) }

      (5) { partHELP([x],[[x]]) }
      (6) { partHELP(x::(y::L),([x])::P) }       <== { partHELP(y::L,P) }
      (7) { partHELP(x::(y::L),(x::L')::P) }    <== { partHELP(y::L,L'::P) }
are added to the specification

val it = () : unit
```

4.3.6 Zukünftige Erweiterungen

Die Fixpunktinduktion kann erweitert werden, um Mengen von Klauseln bearbeiten zu können anstelle einer einzigen Klausel.

4.4 Oktagon

Im Verlauf der Arbeit entstand insbesondere aus dem Ziel, möglichst viel Bildschirmfläche für den Beweisteil und den Spezifikationbereich zu erhalten, die Suche nach einer alternativen Möglichkeit für die Kommando-Wahl im eXpander-System. Eine Möglichkeit wurde mit den oktagonalen Menüs umgesetzt und getestet.

Oktagonale Menüs erscheinen bei Knopfdruck an der Stelle, wo der Cursor sich befindet, können geschachtelt sein und bieten in einem oktagonalen Befehls-Feld die Wahl unter bis zu 8 Befehlen bzw. Untermenü-Aufrufen, die in den Teilbereichen um ein zentrales Abbruch-Feld angeordnet sind.

Vorteilhaft hierbei ist es, daß der Benutzer den Ort, an dem er mit dem Cursor gerade arbeitet, nicht verlassen muß, um ein Kommando auszuwählen, daß sich vielleicht in einer Menüzeile am oberen Bildschirmrand befindet. Weiterhin ist es nun möglich, daß der Benutzer den Ort innerhalb der oktagonalen Menü-Hierarchie schnell eintrainieren kann und sich dauerhaft merken kann. Im Test zeigte sich, daß sehr schnell die Art gelernt ist, wie ein bestimmter Befehl im Menü mit dem Cursor "angefahren" wird. Dadurch ist eine sehr schnelle Kommando-Wahl ermöglicht.

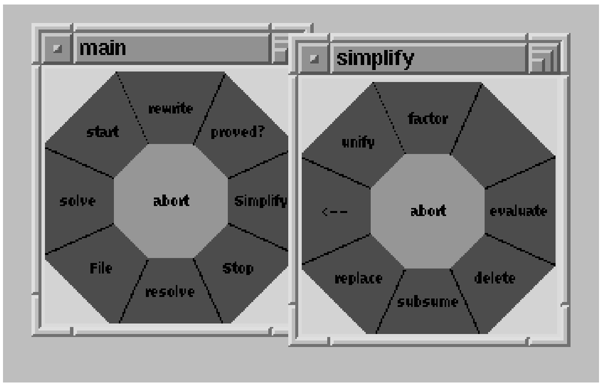

Figure 2. Das Hauptmenü-Oktagon von eXpander mit Untermenü Simplify

Hauptnachteil ist jedoch die starke Gewöhnungsbedürftigkeit an die oktagonalen Pop-Up-Menüs. Der Benutzer ist bereits gewöhnt an die Pull-Down-Menüs am oberen Bildschirmrand und Knopfleisten zur Kommando-Wahl auf dem Bildschirm.

Mit dem Hauptziel vor Augen, eXpander leicht z ugänglich zu machen für Interessierte, wurde darum die Idee der oktagonalen Menüs wieder verworfen.

5 Implementierung

Die Implementierung erfolgte in der Sprache TCl/Tk (Tool Command Language/ Tool kit) in den Versionen Tcl 7.6/Tk 4.2. Optional kann die Tcl/Tk-Erweiterung BLT benutzt werden in der Version 2.0 [2].

Der SML-Teil der Anwendung, der auch den Expander-Beweissystem umfaßt, ist in SML (Standard Meta Language) in der Version 0.93 implementiert [3].

5.1 Kommunikations-Kanal zwischen Expander und Oberfläche

Der SML-Interpreter wird als Unter-Prozeß vom Tcl/Tk-Interpreter "wish" aufgerufen.

Tcl/Tk öffnet dazu einen Kommunikationskanal für Schreiben/Lesen zum SML-Interpreter, dieser Kanal ist mit den Standard-Eingang *stdin* und Standard-Ausgang *stdout* des SML-Interpreters verbunden. Die SML-Seite hat keine Kenntnis davon, daß sie von der eXpander-Oberfläche (Tcl/Tk) angesteuert wird. Der SML-Interpreter verarbeiter auch beim gewohnten Benutzerbetrieb mit Tastatur und Bildschirm seine Ein-/Aufgaben über *stdin bzw. stdout*.

6 Beschaffung und Installation von eXpander

Das eXpander-System ist unter der URL http://www.ls5.cs.uni-dortmund.de erhältlich. Eine ausführliche Anleitung für die Installation ist enthalten.

[2] erhältlich unter URL http://sunscript.sun.com/TclTkCore/8.0.html
[3] erhältlich unter URL http://cm.bell-labs.com/cm/cs/what/smlnj/index.html

7 Die Zukunft für Expander

Hier werden Erweiterungsvorschläge für zukünftige eXpander-Versionen sein können.

7.1 Hierarchische Spezifikationen

Spezifikationen können eine große Anzahl an Axiomen, Theoremen und Vermutungen beinhalten, wobei oft mehrere Lemmata zusammengehören und eine Einheit bilden. So kann beispielsweise eine Funktion in mehreren Lemmata beschrieben werden, die zusammengehören. Wenn man diese Lemmata-Blöcke als Mini-Spezifikationen betrachtet, lassen sich daraus komplexere Spezifikationen erstellen.

Diese Hierarchie läßt sich -ähnlich einem Datei-Manager wie z.B. NeXtStep-Browser- am Bildschirm darstellen, man kann wahlweise einzelne Spezifikations-Teile ausblenden oder einblenden. Es ist leicht möglich, die Lesbarkeit und die Übersichtlichkeit eines Beweises zu verbessern, indem die Teile einer Spezifikation ausgeblendet werden, die für den aktuellen Beweis nicht relevant sind.

Es ist interessant, zu untersuchen, in welchem Maße Spezifikationsteile auch in mehreren Spezifikationen verwendet werden können. Beispielsweise könnte man versuchen, ob der Lemmata-Block, der Kommutativität natürlicher Zahlen beschreibt, nicht auch in Spezifikationen bzgl Ganzer Zahlen, etc. benutzt werden kann, ohne daß die datei dupliziert werden muß.

Expander legt alle Spezifiaktionen in einem Verzeichnis ab. Wenn aber eine hierarchische Spezifikation über mehrere Dateien verteilt wird, kann dies zu Problemen führen. Das System muß dafür Sorge tragen, daß die Dateinamen eindeutig einer Spezifikation zugeordnet werden. In obigem Beispiel der Kommutativität kann es problematisch sein, einen Spezifikations-Teil, der die Kommutativität natürlicher Zahlen beschreibt, als Datei mit Namen "Kommutativität" abzupeichern. Denn in anderen Spezifikation könnte ein solches Gesetz ganz anders lauten (Für dieses Beispiel läßt sich sicherlich noch eine Lösung finden, die mit vertretbarem Aufwand realisiert werden kann. Hier sollte lediglich die Problematik aufgezeigt werden).

Eine mögliche Lösung besteht darin, die Idee von Expander-Projekte einzuführen, die weiter unten in diesem Kapitel beschrieben wird.

7.2 Generierung von Beweis-Report-Dokumenten

Nachdem am Bildschirm ein Beweis erstellt worden ist, liegt es nahe, diesen Beweis auszudrucken. Beispielsweise werden bei dem Programm "InterACT" von Robert Geisler und Marcus Klar (Technische Universität Berlin)[4] LaTex-Dateien generiert werden, die einen anprechende Beweis-Darstellung enthalten. Ebenfalls mit Latex-Dateien können Beweis-Bäume ausgedruckt werden, um einen Überblick über den Beweisgang zu erlauben.

7.3 Experten- und Laien-Modus

Mit dem Expander-Beweissystem kann ein geübter Benutzer schnell und effizient in wenigen Schritten Beweise führen, wohingegen es ratsam sein kann, einem Anfänger bestimmte Regelanwendungen im System auf eine einfachere und didaktisch geeignet aufbereitete Art

[4]geisler@cs.tu-berlin.de bzw. klar@cs.tu-berlin.de

zu ermöglichen, damit er sich schrittweise an das System gewöhnen kann. Um beiden Interessen gerecht werden, sollte es einen Experten-Modus und einen Anfänger-Modus geben, zwischen denen der Benutzer jederzeit wechseln kann.

In beiden Modi könnten kontextsensitive Hilfestellungen gegeben werden, Abläufe an Beispielen erläutert und auch der Zugriff auf weiterführende Hilfe gegeben werden.

7.4 Makro-Funktion für Beweis-Schrittsequenzen

Innerhalb von Beweisen kann es sein, daß eine Sequenz von Beweis-Schritten zusammengefaßt wird zu einer Einheit. Man kann dann diese Makro-Schritte wiederholt auszuführen, auch in anderen Beweisen wiederverwenden und sie können innerhalb eines Beweises erheblich zur Übersichtlichkeit beitragen.

Dazu werden einzelne Makros als Sequenz von hintereinander auszuführenden Expander-Kommandos und ggf. anderen Befehlen abgespeichert. Beim Senden dieser Sequenz an Expander werden diese hintereinander abgearbeitet.

In der aktuellen Implementierung von eXpander gilt es dabei insbesondere zu beachten, daß innerhalb eines Schrittes die Goals umnumeriert werden, dadurch können nachfolgende Kommandos betroffen sein. Eine hierarchische Numerierung (z.B. Goal (1.2.5)) kann hier Abhilfe schaffen, dabei wird der Index 1.2.5 so interpretiert, daß dieses Goal das fünfte Goal ist in der Ergebnismenge einer Operation auf dem Goal 1.2. Auf diese Weise werden neu entstandene Goals eindeutig numeriert.

7.5 Simultane Operationen auf mehreren Goals

Diese Erweiterung betrifft die Theorie des zugrundeliegenden Beweis-Systems. Im Augenblick werden Beweis-Schritte auf einem Goal ausgeführt, dies kann zu einer Goal-Menge als Ergebnis führen. Wenn nun gleichzeitig mehrere Goals dieser Menge bearbeitet werden können, verkürzt dies den Beweis, vereinfacht ihn und trägt zur Übersichtlichkeit bei.

Da diese Erweiterung in der Zukunft geplant ist, ist beim Entwurf der Benutzer-Sicht auf eXpander darauf zu achten, daß auf geeignete Weise mehrere Goals ausgewählt werden können, um eine Operation damit durchzuführen. Wenn eine solche Operation komplexe Eingaben erfordert, kann dies zusätzliche Schwierigkeiten entstehen lassen insbesondere bei der Darstellung am Bildschirm.

7.6 Beweis-Bäume

Ein Beweis läßt sich darstellen als Baum, beginnend bei der Eingabe-Formel werden -mehrfach verzweigend- Zwischenschritte erreicht. Da Expander von Ziel- und Anfangs-Formel beginnend aufeinander zu gleichzeitig den Beweis aufbaut, wird dieser Beweis-Graph komplizierter sein. Er besteht aus den beiden Bäumen für Anfangs- und End-Formel, dessen Äste aufeinander zu wachsen.

Die Idee besteht nun darin, innerhalb eines Beweises in diesem Baum zu navigieren (vgl. Programm "InterACT" von Robert Geisler und Markus Klar), um einzelne Beweis-Schritte auf Baumknoten operieren zu lassen. Dabei tritt unter anderem das Problem auf, wie die Darstellung der in den Knoten stehen Goal-Information realisiert werden kann.

Eine Lösung kann darin bestehen, in einem großen Fenster zu navigieren, mit der Maus einzelne Knoten (oder auch Mengen von Knoten bei Operationen auf mehreren Goals) auszuwählen und die dort enthaltene Information spearat in einem zweiten Fenster anzeigen zu lassen (Beispiel: Programm Rose98 von Fa. Rational [5]).

Eine weitere Idee wäre, die Knoteninformation vergrößert über einem Knoten anzeigen, während sich der Cursor darüber befindet (ähnlich dem Hilfe-Anzeige beim Netscape-Browser, wenn der Cursor längere Zeit über einem Button stehen bleibt).

7.7 Java-eXpander

Mit dem Ziel, Expander leicht zugänglich zu machen (insbesondere über das Internet), liegt es nahe, eine grafische Benutzer-Schnittstelle bereitzustellen, die insbesondere die Darstellung in Browsern ermöglicht (ohne dabei jedoch auf zusätzliche Tcl/Tk-Plugins angewiesen zu sein).

Java bietet alle bekannten Vorteile einer objektorientierten Sprache wie u.a. leichtere Wartbarkeit und die Kapselung von Anwendungsteilen. Java empfiehlt sich auch wegen den umfangreichen Klassenbibliotheken AWT und SWING, die einfache und überzeugende Oberflächengestaltung ermöglichen.

7.8 Einführung von Expander-Projekten

Im Augenblick werden drei Verzeichnisse gepflegt, in denen Spezifikatione, erzeugte Graphen und erzeugte Beweise abgelegt werden. Damit ist jedes Projekt (z.B. Arbeiten mit Natürliche Zahlen über der Spezifikation NAT) über diese Verzeichnisse verteilt.

Es liegt nahe, alle Informationen eines Projektes zusammen zu halten. Dazu kann in einem allgemeinen Verzeichnis "Projekte" für jedes gegebene Projekt dort ein Unterverzeichnis erstellt werden, das alle Informationen für dieses projekt aufnimmt. Dort werden also u.a. die Spezifikationen, die Beweise, die Beweisbäume, die Graphen abgelegt.

Wenn beispielsweise mit hierarchischen Spezifikationen eine gegebene Spezifikation NAT über natürliche Zahlen (siehe obiges Beispiel bei hierarchischen Spezifikationen)sich über mehrere Dateien verteilen, ist es auf diese Weise möglich, Teil-Spezifikationen mit Namen "Kommuntativität" abzulegen, welche die Lemmata für die Kommutativität natürlicher Zahlen enthalten. Ein mögliches Problem kann dann die gleichzeitige Verwendung der Lemmata für die Kommutativität dieser Datei in anderen Spezifikationen sein, die in separaten Projekt-Verzeichnissen liegen. Dies kann jedoch mit symbolischen Links gelöst werden auf Dateisystemebene.

Abschließende Bemerkungen

Das Expander-Beweissystem wird in naher Zukunft eine in der Programmiersprache Java Benutzeroberfläche bekommen, die zur Zeit durch Thomas Ströter und Najib Rachachi entwickelt wird.

Insbesondere die in Java bereits enthaltenen Möglichkeiten für die Gestaltung von grafischen benutzeroberflächen und die Möglichkeit entfernter Aufrufe machen gerade diese Sprache interessant.

Dies kann ein erster Schritt sein, um das Beweissystem "Expander" von Prof. Dr. Peter Padawitz auch direkt über das Internet ansprechbar zu machen. Damit steht Expander dann noch einfacher zur Verfügung.

[5] http://www.rational.com

A Programm-Code des eXpander-Systems

A.1 Programmteil SML

A.2 Programmteil TCl/Tk

A.3 Makefiles

Literatur

[Pad96] P. Padawitz, *Expander - A System for Testing and Verifying Functional-Logic Programs*,
Report, FB Informatik, Universität Dortmund 1996

[Gei95] Robert Geisler, Marcus Klar, *InterACT*,http://tfs.cs.tu-berlin.de/interact/html/interact.html, 1995

[Pad97] P. Padawitz, *Theorie der Programmierung*, Vorlesungs-Skript WS 96/97, FB Informatik, Universität Dortmund 1997

[Hel97] M. Heller, *eXpander: A Graphical User Interface to Expander for the X Window System*, Manual, FB Informatik, Universität Dortmund 1997