Automatisierte Logik und Programmierung

Einheit 15

Assistenzsysteme zur Implementierung formalisierter Mathematik

1. Formalisierung mathematischer Theorien
2. Verwaltung formalisierten Wissens
3. Gestaltung der Benutzerinteraktion

Theorembeweisen ist mehr als Beweissuche

- **“Echte Beweisführung” ist niemals isoliert**
  - Beweise werden nicht “von scratch” geführt, sondern in einem Kontext
    - · Mathematik: Algebra, Analysis, Logik, Kategorientheorie, ...
    - · Programmierung: Verifikation, Synthese, Optimierung, Security, ...
    - · Physik, Elektrotechnik, Mechanik, ...
  - Kontext bestimmt Begriffswelt, Erkenntnisse, Methoden, ...
  - Beweisführung stützt sich massiv auf bekanntes Wissen
- **Logisches Schließen benötigt formales Wissen**
  - Präzisierung der Grundkonzepte der zugrundeliegenden “Theorie”
  - Formulierung der fundamentalen Einsichten der Theorie (Axiome)
  - Beweise für “ableitbare” Erkenntnisse (Sätze)
  - Formalisierung theoriespezifischer Methoden als Beweisstrategien
- **Wissen muß formal abgestützt werden**
  - Grundkonzepte werden über formale Definitionen auf CTT abgestützt
  - Axiome, Theoreme und Strategien werden aus heraus abgeleitet

Methodik des Aufbaus formaler Theorien

- **Formalisiere Grundkonzepte der Theorie**
  - Formale Notation für Datentyp, kanonische & nichtkanonische Elemente
  - Formulierung von Inferenzregeln/Axiomen für Elemente und Datentyp
- **Implementiere Grundkonzepte der Theorie**
  - Stütze Begriffe durch Abstraktionen/Display Formen auf CTT Terme ab
  - Beschreibe neue Inferenzregeln durch Taktiken
    - Bei echten Erweiterungen verwende explizite Regelobjekte
- **Erstelle erweiterte Objekttheorie**
  - Formalisiere wichtige Begriffe nur auf Basis der Grundkonzepte
  - Beweise mathematische Gesetze zu Eigenschaften abgeleiteter Konzepte insbesondere Rewrite-Lemmata zu Kombinationen von Operationen

Systematik wurde bisher nur wenig benutzt

Beispiel: Theorie endlicher Mengen

- **Theorie kann auf vier Grundbegriffe gestützt werden**

Datentyp: $\texttt{Set}(\alpha)$

Operationen: $\emptyset\texttt{:}\ \texttt{Set}(\alpha)$

$\texttt{+:}\ \texttt{Set}(\alpha)\times\alpha \longrightarrow \texttt{Set}(\alpha)$

$\in\texttt{:}\ \alpha\times\texttt{Set}(\alpha) \longrightarrow \texttt{Bool}$

Gesetze: $\texttt{a}\notin\emptyset$

$\texttt{x}\in\texttt{(S+a)} \;\Leftrightarrow\; \texttt{(x=a}\ \vee\ \texttt{x}\in\texttt{S)}$

$\texttt{(S+a)+x} = \texttt{(S+x)+a}$

$\texttt{(S+a)+a} = \texttt{S+a}$

$\texttt{P}(\emptyset) \wedge (\forall\texttt{S:Set}(\alpha)\texttt{.P(S)}\Rightarrow\forall\texttt{a:}\alpha\texttt{.P(S+a))} \;\Rightarrow\; \forall\texttt{S:Set}(\alpha)\texttt{.P(S)}$

- **Implementierung durch Restklassen von Listen**

$$
\begin{array}{ll}
\emptyset & \equiv\ \texttt{nil} \\
+ & \equiv\ \lambda\texttt{a,S. a.S} \\
\in & \equiv\ \lambda\texttt{a,S.}\ \exists\texttt{x}\in\texttt{S.x=}_b\texttt{a} \\
=_{Set} & \equiv\ \lambda\texttt{S,T.}\ (\forall\texttt{a}\in\texttt{S. a}\in\texttt{T)} \wedge (\forall\texttt{a'}\in\texttt{T. a'}\in\texttt{S)} \\
\texttt{Set}(\alpha) & \equiv\ \texttt{(S,T):}\alpha\,\texttt{list//S=}_{Set}\texttt{T}
\end{array}
$$

Alternative Implementierungen möglich (Bitvektoren, sortierte Listen,. . .)

Theorie endlicher Mengen
Formalisierung wichtiger abgeleiteter Begriffe

`empty?`	$\equiv$	λ`S. if S=`$\emptyset$` then tt else ff`
$\subseteq$	$\equiv$	λ`S,S'. `$\forall$`x`$\in$`S.x`$\in$`S'`
$\{$*list-exp*$\}$	$\equiv$	*list-exp*`.nil`
$\{i..j\}$	$\equiv$	`ind(`j`-`i`; _,_.`$\emptyset$`; `$\{j\}$`; diff,j-set.j-set+(`j`-diff))`
$\{f_x \mid x \in S \wedge p_x\}$	$\equiv$	`list_ind(`S`; `$\emptyset$`; a,_,GSF. if `p_x`[a/`x`] then GSF+`f_x`[a/`x`] else GSF`
$\lvert S \rvert$	$\equiv$	`list_ind(`S`; 0; a,S',card. if a`$\in$`S' then card else card+1)`
$-$	$\equiv$	λ`S,a. `$\{$`x`$\mid$`x`$\in$`S`$\wedge$`x`$\neq$`a`$\}$
$\cup$	$\equiv$	λ`S,S'. list_ind(S'; S; a,_,union.union+a)`
$\cap$	$\equiv$	λ`S,S'. `$\{$`x`$\mid$`x`$\in$`S`$\wedge$`x`$\in$`S'`$\}$
$\setminus$	$\equiv$	λ`S,S'. `$\{$`x`$\mid$`x`$\in$`S`$\wedge$`x`$\notin$`S'`$\}$
$\bigcup$	$\equiv$	λ`FAMILY. list_ind(FAMILY; `$\emptyset$`; S,FAM,Union.Union`$\cup$`S)`
$\bigcap$	$\equiv$	λ`FAMILY. list_ind(FAMILY; fail;` `S,FAM,inter. if empty?(FAM) then S else inter`$\cap$`S)`
`map`	$\equiv$	λ`f,S. `$\{$`f(x)`$\mid$`x`$\in$`S`$\}$

`reduce`	$\equiv$	λ`op,S. list_ind(S; fail;`
		`a,S',redS'. if empty?(S') then a`
		`else if a`$\in$`S' then redS' else op(redS',a))`

$T =_{Set} S \uplus S'$	$\equiv$	$T =_{Set} S \cup S'$ $\wedge$ `empty?(`$S \cap S'$`)`

Theorie endlicher Mengen (Auszug)
Fundamentale Gesetze abgeleiteter Konzepte

Lemmata zu "$\in$"

1. $a' \in \{a\} \Leftrightarrow a' = a$
2. $k \in \{i..j\} \Leftrightarrow (i \le k \wedge k \le j)$
3. $b \in \{f(x) \mid x \in S \wedge p(x)\}$
   $\Leftrightarrow \exists x \in S.\, p(x) \wedge b = f(x)$
4. $a \in \{x \mid x \in S \wedge p(x)\} \Leftrightarrow a \in S \wedge p(a)$
5. $a \in S - a' \Leftrightarrow a \in S \wedge a \neq a'$
6. $a \in S \cup S' \Leftrightarrow a \in S \vee a \in S$
7. $a \in S \cap S' \Leftrightarrow a \in S \wedge a \in S'$
8. $a \in S \setminus S' \Leftrightarrow a \in S \wedge a \notin S'$
9. $a \in \bigcup FAM \Leftrightarrow \exists S \in FAM.\, a \in S$
10. $a \in \bigcap FAM \Leftrightarrow \forall S \in FAM.\, a \in S$
11. $arb(S) \in S$
12. $b \in map(f,S) \Leftrightarrow \exists x \in S.\, b = f(x)$

Lemma zu "empty?"

1. $empty?(S) \Leftrightarrow S = \emptyset$
2. $empty?(S) \Leftrightarrow |S| = 0$
3. $S \subseteq S' \Rightarrow empty?(S') \Rightarrow empty?(S)$
4. $\neg empty?(S+a)$
5. $\neg empty?(\{a\})$
6. $empty?(\{f(x) \mid x \in S \wedge p(x)\})$
   $\Leftrightarrow empty?(S) \vee \forall x \in S.\, \neg p(x))$
7. $empty?(S-a) \Leftrightarrow empty?(S) \vee S = \{a\}$
8. $empty?(S \cup S')$
   $\Leftrightarrow empty?(S) \wedge empty?(S')$
9. $empty?(S) \vee empty?(S')$
   $\Rightarrow empty?(S \cap S')$
10. $empty?(S) \Rightarrow empty?(S \setminus S')$
11. $empty?(\bigcup FAM)$
    $\Leftrightarrow \forall S \in FAM.\, empty?(S)$
12. $\exists S \in FAM.\, empty?(S)$
11. $empty?(\bigcup FAM) \Rightarrow empty?(\bigcap FAM)$

Assistenz für Entwicklung formaler Theorien

- **Beweisystem muß Wissensverarbeitung unterstützen**
  - Erzeugung formaler Definitionen, Sätze, Beweise, Methoden, Texte
  - Strukturierung formalen Wissens in Theorien und Sub-Theorien
  - Umbenennen, Verschieben, Verlinken, Entfernen von Wissen
  - Verwendung formalen Wissens in Beweisen, Methoden und Texten
  - Durchsuchen gespeicherten Wissens, suche nach "relevantem" Wissen
- **Wissensverarbeitung ist mehr als "Sammeln"**
  - Neue Erkenntnisse kommen hinzu, andere werden entfernt
  - Spezifische Beweise und Beweismethoden ändern sich
  - Konsistenz des gespeicherten Wissens muß sichergestellt sein
  - Vorhandensein gespeicherten Wissens benötigt eine Rechtfertigung
    z.B. durch Verweise auf Inferenzregeln oder externe "Autoritäten"
- **Unterstützung für dezentrales Arbeiten**
  - Export, Import, Mischen und Prüfung von (Teil-)Theorien
  - Einschränkung von Schreib- und Zugriffsrechten

Aufbau formaler Wissensbanken

- **Textorientierte Gestaltung** (Isabelle, Coq, MetaPRL, SpecWare)
  – Objekte in konventioneller Textform, strukturiert durch Schlüsselworte
  – Dateien werden wie Programmcode sequentiell gelesen und compiliert
  + Konventionell editierbar, Informationen leicht austauschbar, einfach
  − Konsistenz nur durch strikt lineare Verarbeitung gesichert
  − Nur ein Benutzer, nur ein aktuell sichtbares Objekt
  − Keine Zugriffskontrolle: jeder kann alles überschreiben / löschen

- **Abstrakte Datenbank** (Nuprl)
  – Zugriffe auf Wissensbank über Datenbankmanagementsystem
  – DBMS verwaltet Namensgebung, Strukturierung und Zugriffsrechte
  − Komplexeres System, kein einfaches Editieren von Text möglich
  − Synchronisatzion, Import/Export von Theorien nur über das DBMS
  + Multiuser-Kooperationen möglich, Viele Objekte gleichzeitig sichtbar
  + Zugriffskontrolle und Transaktionskonzept sichert Konsistenz
  und erhöht Sicherheit gegenüber Mißbrauch, Irrtümern und Fehlern

DIE NUPRL LIBRARY ALS FORMALE WISSENSBANK

- **Externe Sicht: formales mathematisches Lehrbuch**
  – Definitionen, Sätze, Beweise, Methoden, Anmerkungen, Regeln, ...
  – Ermöglicht zusätzliche Inferenzregeln: `lemma`, `extract`,...
- **Interne Sicht: Kollektion von Bibliotheksobjekten**
  – Keine vorgegebene Strukturierung der Wissensbank
  – Externe Strukturen (Theorien, Directories, Links,...) sind aufgesetzt
- **Interne Struktur von Bibliotheksobjekten**

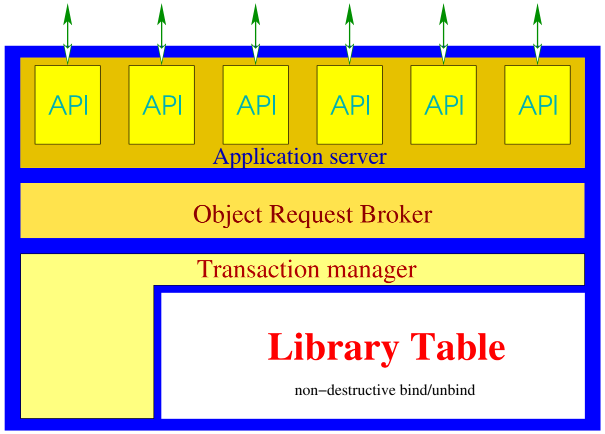
  Tupel bestehend aus Inhalt und Verwaltungsinformation

Inhalt:	Abstraktion, Display Form, Beweis, ML code, Text, ...
Art:	`ABS, DISP, STM, CODE, COM, RULE, DIR`,...
Eigenschaft:	Status, Name, Aktiv?, Referenzumgebung, ...
Extra:	Abhängige Objekten, interne Id, sichtbare Position, ...

  Repräsentation als abstrakte Nuprl-Terme ermöglicht Selbstreflektion
  – (z.B. Darstellung von Browser & Sequenzen modifizierbar durch Display-Objekte)

Aufgaben des Wissensbankmanagementsystems

- **Bereitstellung von Operationen zur Verwaltung von Objekten**
  - Erzeugung, Löschen, Umbenennen, Verschieben, (De)Aktivieren, Drucken,
  - Strukturierung in Theorien und Directories, Browsen, Suchen, . . .
- **Wissensarchivierung**
  - Zertifikate: Rechtfertigung für gespeicherte Inferenzen
  - Explizite Links und logische Abhängigkeiten zwischen Objekten
- **Anbindung anderer Komponenten**
  - Refiner, Editor, externe Systeme als Klienten
  - Mehrfache Instanzen desselben Klienten möglich



- **Datenbankoperationen**
  - Dauerhafter Objektspeicher, Konsistenzsicherung
  - Backup alter Zustände, Undo, Versionskontrolle
  - Transaktionsgesteuerter simultaner Zugriff mehrerer Klienten
  - Selektive Sichten auf Teile der Bibliothek

Implementierung der Objektsprache in der Library

- **Abstraktionsobjekte erweitern Operatorentabelle**
  - Basisterme der Theorie werden als Primitve deklariert

- ABS: function def

```
function{}(.A;x,.B[x];) == !primitive
```

  - Benutzerdefinierte Objekte werden durch andere Terme erklärt

- ABS: exists_uni

```
∃!x:T. P[x] == ∃x:T. P[x] ∧ (∀y:T. P[y] ⇒ y=x∈T)
```

- **Display-Objekte erweitern die Displaytabelle**

- DISP: exists_unique_df

```
Parens::Prec(exists)::∃!<x:var>:<T:type:*>{\?.} <P:prop:E>
 == exists_unique(<T>;<x>.<P>)
Parens::Prec(exists)::∃!<x:var>{\?.} <P:prop:E>
 == exists_unique(ℕ;<x>.<P>)
```

- **Code-Objekte enthalten ML Programme**
  - Konstruktoren, Destruktoren für objektsprachliche Terme, . . .

Unterstützt schnelle, flexible Implementierung beliebiger Theorien

Datenstrukturen für formale Definitionen

- **Struktur einer Abstraktion:** $lhs \equiv rhs$
  - lhs: Abstrakter Term, dessen Unterterme Variablen sind
  - rhs: Term, dessen freie Variablen auch in lhs frei sind

  Neuer Term auf linker Seite wird durch Term der rechten Seite definiert
- **Einfache Repräsentation als Datenstruktur**
  - Datentyp: `abstype` **abstraction** `= term # term`
  - Konstruktor `mk_abstraction` testet Zusatzbedingungen
- **Abstraktionsanwendung ist aufwendiger** (Manual §7.1)
  - Pattern Matching und Instantiierung von Variablen
  - Variablen zweiter Stufe beschreiben Terme mit gebundenen Variablen
- **Separate Behandlung der Display-Form** (Manual §7.2)
  - Formale Beschreibung einer vertrauten und verständlichen Notation
    - Textliche Darstellung, Formatierung, Klammerung, Abkürzungen, ...
  - Display-Formen werden im Layout-Algorithmus des Editors verwendet

Werkzeuge zur Erzeugung formaler Theorien

- **Direkte Konstruktion von Theoriebestandteilen**
  - `AddDef*` Explizite Definition neuer Begriffe
    - Benutzer beschreibt abstrakten Term und seine CTT-Definition
    - System generiert Abstraktion & Standard Display Form
    - System generiert wf-Theorem zur Unterstützung taktischer Typechecker
    - Benutzer modifiziert Display Form
    - Benutzer formuliert und beweist wf-Theorem
  - `AddRecDef*` Definition rekursiver Konzepte
  - `MkThm*`, `MkML*` Explizite Erzeugung von Theorem- und Code-Objekten
    - System erzeugt nur das Objekt
- `AddRecMod*` **Erzeugung von Theorie Modulen**
  - Erzeugung der Basistheorie als Σ-Type der CTT — Manual §4.3.2.3
    - Benutzer spezifiziert Grundtheorie als abstrakten Datentyp
    - System erzeugt "Spezifikationstheoreme" und zugehörige Definitionen
  - Erzeugung der erweiterten Objekttheorie wie oben.

Zugriff auf Objekte

- **Browsen von Theorien**
  - Interaktives Durchsuchen von Theorien mit dem Navigator
  - Benutzer sieht Namen und Anordnung der Objekte
  - Benutzer öffnet Objekte, um Inhalt zu sehen
- **Hyperlinks**
  - Terme und ML-Code enthalten Hyperlinks
  - Abstraktion / Display Form von Termen werden durch Mausklick sichtbar
  - Definition von ML Objekten werden durch Mausklick geöffnet
- **Suche**
  - `NameSearch*`: Suche nach Objekten, deren Name ein Pattern matcht
    - · Systematische Namensgebung macht relevante Objekte leicht aufspürbar
  - In Erprobung: Suche nach Objekten, die bestimmte Terme enthalten
- **Obid-Collectors**
  - Aufsammeln von Objekten mit bestimmten Gemeinsamkeiten
  - Auch möglich für Objekte, auf die kein Link mehr zeigt.

Methoden zur Organisation von Wissen

- **Strukturierung der Wissensbank in Theorien**
  - Theorieobjekte sind linear geordnet zur Vermeidung von Zyklen
  - Theorien sind i.w. baumartig angeordnet
  - Theorien können von anderen Theorien abhängig gemacht werden
- **Theorieobjekte haben Referenzumgebungen**
  - Abstraktionen, Lemmata, Taktiken, etc. das Objekt benutzen darf
    - Alle Objekte, auf die das Initialobjekt der Theorie verweist
    - Alle Objekte der Theorie, die vor dem Objekt erscheinen
  - Referenzumgebung wird nach Einfügen von Objekten aktualisiert
- **Taktiken können theorieabhängig sein**
  - Autotaktik verwendet wf-Theoreme aller Abstraktionen und theorieabhängige Ergänzungstaktiken
  - Autotaktik verwendet Proof-Caching für wiederholte Beweisteile

Wichtige Objekt-Theorien in Nuprl

Basiswissen Mathematik & Programmierung

- **Arithmetik & Zahlentheorie**
  - Definition von Zahlenbereichen, Intervallen, Teilbarkeit, ...
  - Theorie der Restklassen modulo n
  - Derzeit nur als Sammlung der Arbeit verschiedener Benutzer

    (Standard Theorie: `int_1`, `int_2`, `num_thy_1`, Obvious `nat_extra`, ...)
- **Standard-Datentypen der Programmierung**
  - Endliche Listen, Mengen, Bags, Stacks, Graphen, Bäume, Arrays, ...
    - Beispiel: radhika presentation: `lists`
  - Quelle: Inhalt von Lehrmaterial, -büchern und Forschungsergebnissen
- **Je nach Bedarf: Anwendungsnahe Objekttheorien**

  (Algebra, Robbins,...)
  - Definition neuer Konzepte einer Anwendung
  - Theoreme über Eigenschaften dieser Konzepte

Begriffe für Anwendungen in der Programmsynthese

`𝔹, true, false`	Data type of boolean expressions, explicit truth values
$\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$	Boolean connectives
$\forall$`x`$\in$`S.p`, $\exists$`x`$\in$`S.p`	Limited boolean quantifiers (on finite sets and sequences)
`if p then a else b`	Conditional
`Seq(`α`)`	Data type of finite sequences over members of α
`null?`, $\in$, $\sqsubseteq$	Decision procedures: emptiness, membership, prefix
`[]`, `[a]`, `[i..j]`, `[`a_1`...`a_n`]`	Empty/ singleton sequence, subrange, sequence former
`a.L`, `L·a`	prepend `a`, append `a` to `L`
`[f(x)`$\mid$`x`$\in$`L`$\wedge$`p(x)]`, `|L|`, `L[i]`	General sequence former, length of `L`, `i`-th element,
`domain(L)`, `range(L)`	The sets `{1..|L|}` and `{L[i]	i`$\in$`domain(L)}`
`nodups(L)`	Decision procedure: all `L[i]` are distinct (no duplicates)
`Set(`α`)`	Data type of *finite* sets over members of α
`empty?`, $\in$, $\subseteq$	Decision procedures: emptiness, membership, subset
$\emptyset$, `{a}`, `{i..j}`, `{`a_1`...`a_n`}`	Empty set, singleton set, integer subset, set former
`S+a`, `S-a`	element addition, element deletion
`{f(x)`$\mid$`x`$\in$`S`$\wedge$`p(x)}`, `|S|`	General set former, cardinality
`S`$\cup$`T`, `S`$\cap$`T`, `S\T`	Union, intersection, set difference
$\bigcup$`FAMILY`, $\bigcap$`FAMILY`	Union, intersection of a family of sets

Benutzerinterface (Editor)

Visuelle Unterstützung zur Bearbeitung von Wissen

- **Skript-/kommandorientierte Gestaltung** (Isabelle, Coq, MetaPRL)
  - Definitionen, Sätze, Beweise entstehen durch Eingabe von Kommandos
  - System zeigt jeweiliges (Teil-)Ergebnis an
  - Aufgesetztes Interface (e.g. ProofGeneral) unterstützt serielle Verarbeitung von Beweisskripten und mathematische Zeichensätze
  - \+ Geringer Aufwand, leicht zu lernen, Einsatz vertrauter Editoren möglich
  - − Textbasiertes Vorgehen, nur aktuelles Beweisziel sichtbar
  - − Syntax eingeschränkt durch Fähigkeiten des Parsers
- **Visuelle Interaktion** (Nuprl)
  - Benutzer navigiert visuell durch Bibliothek, Beweisbaum, ...
  - Struktureditoren unterstützen Manipulation verschiedenartiger Objekte
  - − Aufwendigere Implementierung, schwieriger für Anfänger
  - \+ Flexible Syntax, Trennung zwischen Notation und Bedeutung
  - \+ Parallele Bearbeitung mehrerer Beweisziele, mehr sichtbare Information

Bestandteile des Nuprl Editors

- **Navigator**
  – Navigation durch Bibliothek und Aufruf bereitgestellter Operationen
- **Beweiseditor**
  – Beweisführung und Navigation durch Beweisbäume
- **Termeditor**
  – Strukturelles Editieren von Termen in Präsentationsform
- **Objekteditoren**
  – Erstellung und Modifikation spezifischer Objekte
- **Kommandointerface**
  – Interpretation von ML-Programmen und metasprachlichen Befehlen
- **Unabhängiger Prozess**
  – Mehrere Editoren können gleichzeitig auf dieselbe Library zugreifen

Graphische Interaktion verbesserungsfähig (i.w. Textterminal)

GUI sollte sich an aktuellen Standards orientieren $\mapsto$ DA Maik Jorra

- **Visuelle Navigation durch Bibliothek**
  - Keyboard- oder Maus-gesteuertes Durchlaufen
  - Patterngesteuerte Namenssuche
  - Springen zu gespeicherten Positionen
- **Ausführung von Bibliothekskommandos**
  - Vorbereitete "Buttons" für die wichtigsten Operationen
    - Erzeugung von Objekten, Theorien, Definitionen, Modulen
    - Löschen, Kopieren, Verschieben, Umbenennen, Drucken, ...
    - Import, Export, Drucken und Dokumentation von Theorien
  - Aufruf der Operationen öffnet Kommandomenü
- **Undo und Redo für jede Operation**
- **Anpassbar**
  - Buttons und Erscheinungsbild durch Bibliotheksobjekte definiert

Browsen der Bibliothek mit Nuprls Navigator

- TERM: Navigator

```
MkTHY*  OpenThy*  CloseThy*  ExportThy*  ChkThy*  ChkAllThys*  ChkOpenThy*
CheckMinTHY*  MinTHY*  EphTHY*  ExTHY*

Mill*  ObidCollector*  NameSearch*  PathStack*  RaiseTopLoops*
PrintObjTerm*  PrintObj*  MkThyDocObj*  ProofHelp*  FixRefEnvs*
CpObj*  reNameObj*  EditProperty*  SaveObj*  RmLink*  MkLink*  RmGroup*

ShowRefenv*  SetRefenvSibling*  SetRefenvUsing*  SetRefenv*  SetInOBJ*
MkTHM*  MkML*  AddDef*  AddRecDef*  AddRecMod*  AddDefDisp*  AbReduce*
Act*  DeAct*  MkThyDir*  RmThyObj*  MvThyObj*

↑↑↑↑  ↑↑↑  ↓↓↓↓  ↓↓↓  <>  ><

Navigator: [num_thy_1; standard; theories]

List Scroll : Total 159,  Point 5,  Visible : 10
 ----------------------------------------------
    CODE  TTF  RE_init_num_thy_1
    COM   TTF  num_thy_1_begin
    COM   TTF  num_thy_1_summary
    COM   TTF  num_thy_1_intro
    DISP  TTF  divides_df
 -> ABS   TTF  divides
    STM   TTF  divides_wf
    STM   TTF  comb_for_divides_wf
    STM   TTF  zero_divs_only_zero
    STM   TTF  one_divs_any
 ----------------------------------------------
```

- Bewegung des Nav Points durch Keyboard, Maus, oder Arrow-buttons
- Öffnen von Objekten durch “rechtsgehen” (oder Mittel-Click)
- Sichtbarkeitsbereich kann vergrößert oder verkleinert werden

Editieren von Termen

Manual §5

- **Mathematische Notation erlaubt keine Parser**
  - Zu reichhaltig (nicht kontextfrei) und nicht einheitlich geregelt
  - Notation ist keine gute Repräsentationform für logische Konzepte
- **Typentheorie trennt Notation von Struktur**
  - Logische Struktur leichter zu verarbeiten
  - Separate Darstellungsform sorgt für verständliche Notation
- **Editiere logische Struktur von Termen**
  - bei gleichzeitiger Präsentation der Darstellungsform auf dem Bildschirm
- **Struktureditor**
  - Erzeugung des Termbaums durch Eintrag in Slots der Darstellungsform
  - Kenntnis der genauen Syntax nicht erforderlich
  - Umdenken erforderlich: keine lineare Eingabe von Text

Benutzer kann mit verständlicher Notation arbeiten

- **Sichtbare Entwicklung von Beweisen**
  - Navigation durch Beweisbaum mit Maus und Keyboard
  - Arbeiten im einzelnen Beweisknoten
  - Kontrolliertes Interface zum Refiner (via Library)
  - Graphische Interaktion verbesserungsfähig (i.w. Textterminal)

- **Operationen auf Beweisen**
  - Erzeugung von Beweiszielen mit Term-Editor
  - Synchrone oder asynchrone Ausführung von Taktiken
  - Komprimierung und Expansion bis zu elementaren Schritten
  - Verarbeitung von Backup-Beweisen und ‘Schmierblatt’-Beweisen
  - Erzeugung von Extrakt-Termen

Typischer Beweisknoten

- PRF: intsqrt

(1) # top 1
(2)
(3) 1. $x{:}\mathbb{N}$
$\vdash \exists y{:}\mathbb{N}.\ y^2 \le x \ \wedge\ x<(y+1)^2$

(4) BY NatInd 1

(5) # 1 1
.....basecase.....
$\vdash \exists y{:}\mathbb{N}.\ y^2 \le 0 \ \wedge\ 0<(y+1)^2$

(6) BY exR ⌜0⌝
There is 1 hidden subgoal

(5) # 1 2
.....upcase.....
1. $x{:}\mathbb{Z}$
2. $0<x$
3. $\exists y{:}\mathbb{N}.\ y^2 \le x-1 \ \wedge\ x-1<(y+1)^2$
$\vdash \exists y{:}\mathbb{N}.\ y^2 \le x \ \wedge\ x<(y+1)^2$

(5) BY

- PRF: intsqrt

(1) # top 1 2
(2) .....upcase.....
(3) 1. $x{:}\mathbb{Z}$
2. $0<x$
3. $\exists y{:}\mathbb{N}.\ y^2 \le x-1 \ \wedge\ x-1<(y+1)^2$
$\vdash \exists y{:}\mathbb{N}.\ y^2 \le x \ \wedge\ x<(y+1)^2$

(4) BY

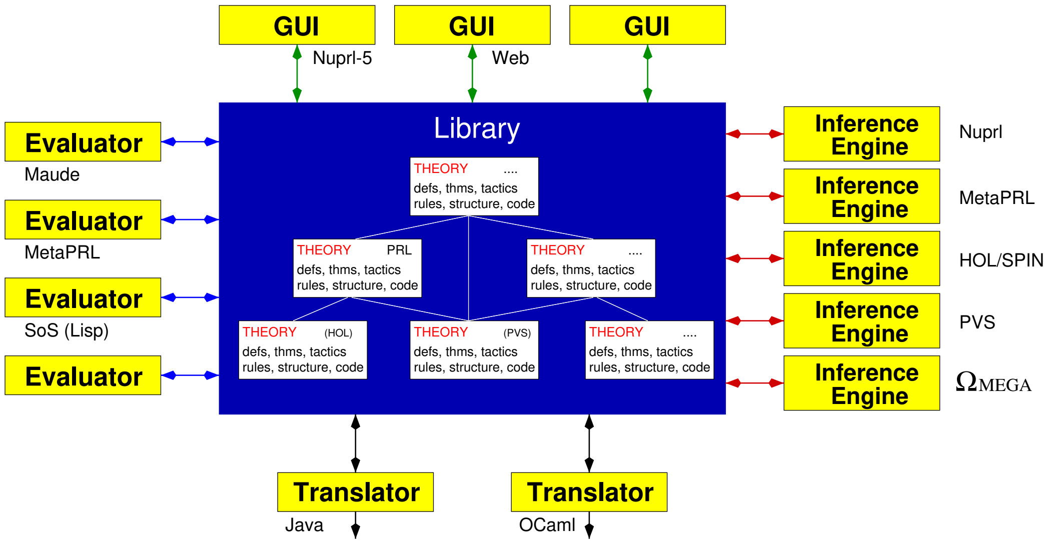
(1) Status und Adresse im Beweisbaum
(2) Annotation des Beweisknotens
(3) Beweisziel (Sequenz)
(4) Angewandte Beweistaktik
(5) Teilziele mit Status, Adresse, Sequenz (neue Hypothesen)
(6) Beweise der Teilziele, sofern vorhanden

Verarbeitung von Taktiken

- **Benutzer gibt Taktik als Inferenzschritt**
- **Beweiseditor ergänzt notwendige Daten**
  – Aktuelle Beweissequenz wird zum Beweisziel
  – Beweiseditor übergibt Beweisziel und Taktik an Refiner
  – Beweisbaum (falls vorhanden) unterhalb des Knotens wird ignoriert
- **Refiner wendet Taktik auf Beweisziel an**
  – Ergibt ungelöste Teilziele und Validierung
  – Anwendung der Validierung auf Teilziele erzeugt Beweisbaum
  – Fehlermeldung, falls Taktik nicht anwendbar
- **Library speichert Beweisbaum**
  – Rechtfertigung für den durchgeführten Inferenzschritt
  – Beweiseditor zeigt Taktik und offene Teilziele
  – Beweisbaum wird nur auf expliziten Wunsch sichtbar gemacht

Taktik wirkt wie abgeleitete Inferenzregel

NUPRL: GESAMTARCHITEKTUR



- **Kooperierende Prozesse**
  - Library im Zentrum
  - "Beliebig viele" Refiner, Editoren und externe Systeme als Klienten
  - Angebundene externe Klienten: MetaPRL, JProver
- **Kooperierende Inferenzmaschinen**
  - Asynchrones und verteiltes Theorembeweisen (In Erprobung)
- **Reflexive Systemstruktur**
  - Systemdesign in Library enthalten (und veränderbar)