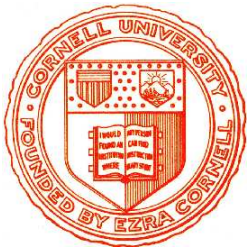
Automatisierte Logik und Programmierung

Einheit 16



Assistenzsysteme für die Implementierung formaler 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

- **Formuliere Grundkonzepte der Theorie**
  - Formale Notation für Datentyp, kanonische & nichtkanonische Elemente
  - Formulierung von Inferenzregeln/Axiomen für Elemente und Datentyp
- **Implementiere Grundkonzepte der Theorie in CTT**
  - 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 weitere 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: $\emptyset\texttt{: Set}(\alpha)$

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

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

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

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

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

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

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

- **Implementierung durch Restklassen von Listen**

  $\emptyset \equiv \texttt{nil}$

  $\texttt{+} \equiv \lambda \texttt{a,S. a.S}$

  $\in \equiv \lambda \texttt{a,S. } \exists \texttt{x} \in \texttt{S.x=}_b\texttt{a}$

  $=_{Set} \equiv \lambda \texttt{S,T. } (\forall \texttt{a} \in \texttt{S. a} \in \texttt{T}) \wedge (\forall \texttt{a'} \in \texttt{T. a'} \in \texttt{S})$

  $\texttt{Set}(\alpha) \equiv \texttt{(S,T)}:\alpha\,\texttt{list // S=}_{Set}\texttt{T}$

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

Endliche Mengen – erweiterte Objekttheorie

- **Formale Definition abgeleiteter Begriffe**

  $\texttt{empty?} \equiv \lambda \texttt{S}.\ \texttt{if S=}\emptyset\ \texttt{then tt else ff}$

  $\subseteq \equiv \lambda \texttt{S,S'}.\ \forall \texttt{x} \in \texttt{S}.\texttt{x} \in \texttt{S'}$

  Weitere Konzepte: $\{\textit{list-exp}\}$, $\{f_x \mid x \in S \wedge p_x\}$, $|S|$, $\cup$, $\cap$, $\setminus$, map... (Anhang, Folie 27)

- **Fundamentale Gesetze abgeleiteter Konzepte**

Lemmata zu "$\in$"

- $\texttt{a'} \in \{\texttt{a}\} \Leftrightarrow \texttt{a'=a}$
- $\texttt{b} \in \{\texttt{f(x)} \mid \texttt{x} \in \texttt{S} \wedge \texttt{p(x)}\}$
  $\Leftrightarrow \exists \texttt{x} \in \texttt{S}.\ \texttt{p(x)} \wedge \texttt{b=f(x)}$
- $\texttt{a} \in \texttt{S} \cup \texttt{S'} \Leftrightarrow \texttt{a} \in \texttt{S} \vee \texttt{a} \in \texttt{S}$
- $\texttt{a} \in \texttt{S} \cap \texttt{S'} \Leftrightarrow \texttt{a} \in \texttt{S} \wedge \texttt{a} \in \texttt{S'}$
- $\texttt{a} \in \texttt{S} \setminus \texttt{S'} \Leftrightarrow \texttt{a} \in \texttt{S} \wedge \texttt{a} \notin \texttt{S'}$
- $\texttt{b} \in \texttt{map(f,S)} \Leftrightarrow \exists \texttt{x} \in \texttt{S}.\texttt{b=f(x)}$

$\vdots$

Lemma zu "empty?"

- $\texttt{empty?(S)} \Leftrightarrow \texttt{S=}\emptyset$
- $\texttt{empty?}(\{\texttt{f(x)} \mid \texttt{x} \in \texttt{S} \wedge \texttt{p(x)}\})$
  $\Leftrightarrow \texttt{empty?(S)} \vee \forall \texttt{x} \in \texttt{S}.\neg\texttt{p(x)})$
- $\texttt{empty?(S} \cup \texttt{S')} \Leftrightarrow \texttt{empty?(S)} \wedge \texttt{empty?(S')}$
- $\texttt{empty?(S)} \vee \texttt{empty?(S')} \Rightarrow \texttt{empty?(S} \cap \texttt{S')}$
- $\texttt{empty?(S)} \Rightarrow \texttt{empty?(S} \setminus \texttt{S')}$
- $\texttt{empty?(map(f,S))} \Leftrightarrow \texttt{empty?(S)}$

$\vdots$

Für endliche Mengen sind mehr als 150 sinnvolle Lemmata herleitbar

Wichtige Theorien für praktische Anwendungen

- **Basiswissen Mathematik und Programmierung**
  - Arithmetik & Zahlentheorie, Intervalle, Teilbarkeit, Restklassen ...
  - Algebraische Strukturen: Gruppen, Ringe, Körper, ...
  - Rationale und reelle Zahlen, konstruktive Analysis, ...
  - Boole'sche Algebra, Strings, Graphen, ...
  - Endliche Listen, Mengen, Bags, Stacks, Maps, Bäume, Arrays, ...
- **Anwendungsnahe Objekttheorien** ↦ Einheit 17ff
  - Formuliere Definition spezifischer Konzepte einer Anwendung
    (Sortierprobleme, Scheduler, Costas Arrays, ...)
  - Beweise Theoreme über Eigenschaften dieser Konzepte
- **Theorien oft bedarfsabhängig von Benutzern aufgebaut**
  - Erweiterung & Systematisierung lückenhafter Theorien wäre sinnvoll
  - Quelle: Inhalt von Lehrmaterial, -büchern und Forschungsergebnissen

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, vertraute Textsuche mit grep oder Emacs
  - \+ Informationen leicht austauschbar, geringer Platzbedarf
  - − Konsistenz nur durch strikt lineare Verarbeitung gesichert
  - − Nur ein Benutzer, nur ein aktuell sichtbares Objekt
  - − Keine Zugriffskontrolle: jeder kann alles überschreiben / löschen
  - − Lokale Bibliotheken verschiedener Nutzer nicht leicht zu integrieren
  - − Systemupdates können existierende Nutzerbibliotheken ungültig machen
- **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**

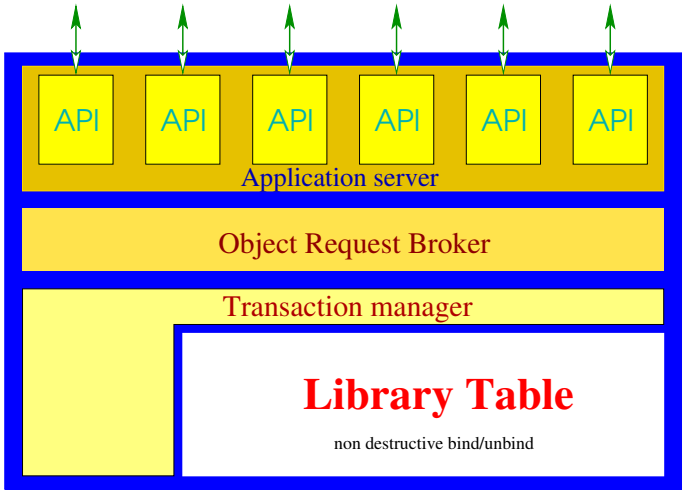
  Tupel bestehend aus Inhalt und Verwaltungsinformation

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 (vgl. §12, Folie 13)

```
- 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** (vgl. §12, Folie 14)

```
- 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, ...
  - Taktiken, Library-Programme, ...

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**
  - Pattern Matching und Instantiierung von Variablen
  - Variablen zweiter Stufe beschreiben Terme mit gebundenen Variablen
- **Separate Behandlung der Display-Form**
  - 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
    - 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 sichtbarer 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

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, ...
  - · Strukturediteren 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 (aufwendig)

Nuprl's Navigator

- **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

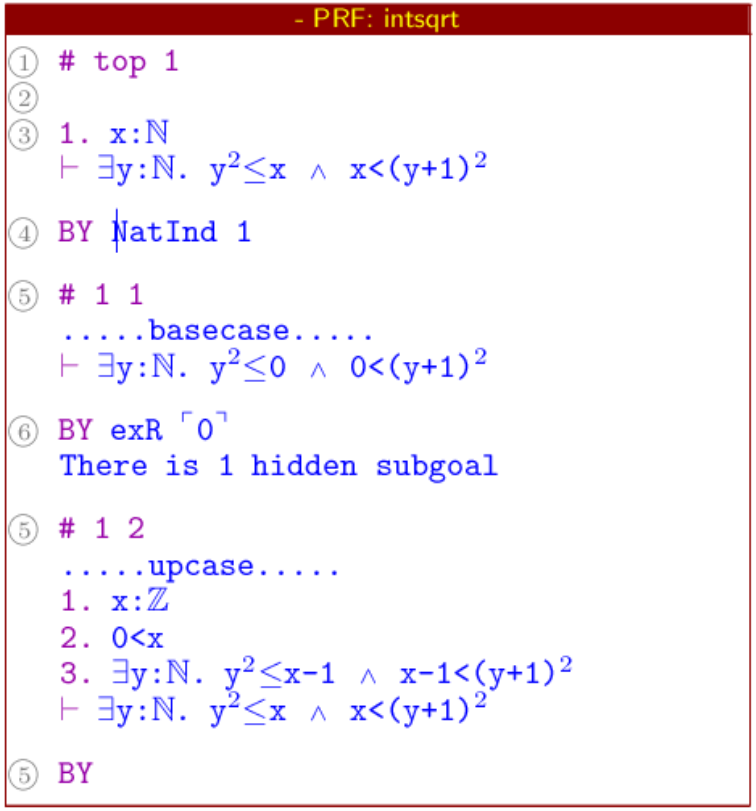
- **Mathematische Notation erlaubt keine Parser**
  - Zu reichhaltig (nicht kontextfrei) und nicht einheitlich geregelt
  - Notation ist keine gute Repräsentationform für logische Konzepte
- **Typentheorie trennt Notation von Struktur**
  - Logische Struktur leichter zu verarbeiten
  - Separate Darstellungsform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

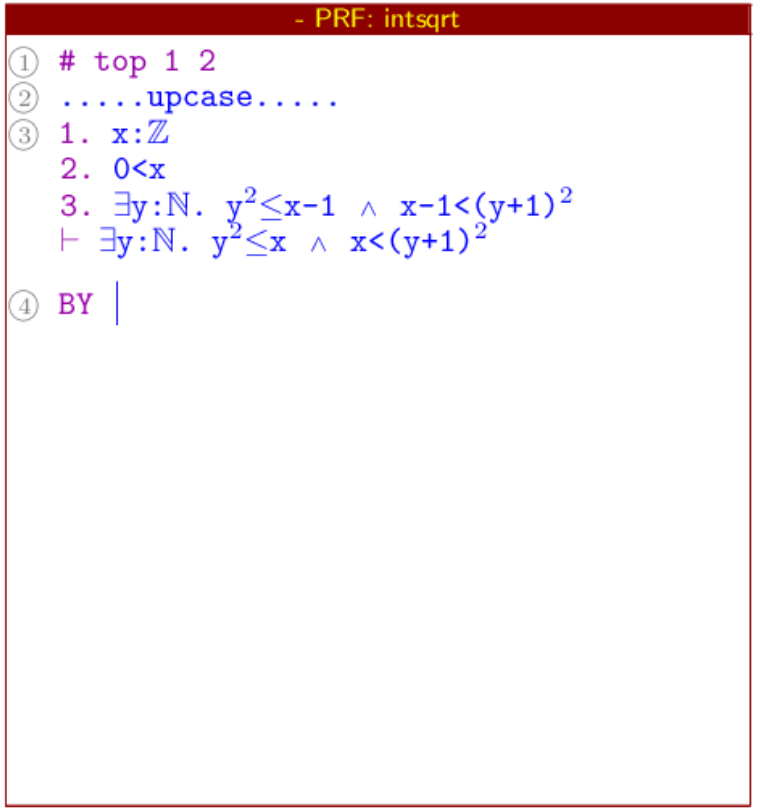
Benutzer kann mit verständlicher Notation arbeiten

Bearbeiten von Beweisen

- **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





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

Transformationstaktiken

- **Benutzer gibt Taktik als Kommando**
- **Beweiseditor ergänzt notwendige Daten**
  - Gesamter aktueller Beweisbaum wird zum Beweisziel
  - Beweiseditor übergibt Beweisziel und Taktik an Refiner
- **Refiner wendet Taktik auf Beweisziel an**
  - Ergibt ungelöste Teilziele und Validierung
  - Anwendung der Validierung auf Teilziele erzeugt Beweisbaum
  - Beweisziel bleibt unverändert, falls Taktik nicht anwendbar
- **Library speichert Beweisbaum**
  - Rechtfertigung für die durchgeführten Inferenzschritte
  - Beweiseditor zeigt ausgeführte Einzelschritte und offene Teilziele
  - Taktikname wird nicht gespeichert
- **Ziel: Modifikation existierender Beweise**
  - Kopieren, Expandieren, Komprimieren, Analogie, ...
  - In Nuprl 5 nur noch als vordefinierte Operationen des Beweiseditors

Effekte der Anwendung der Taktik `Cases`

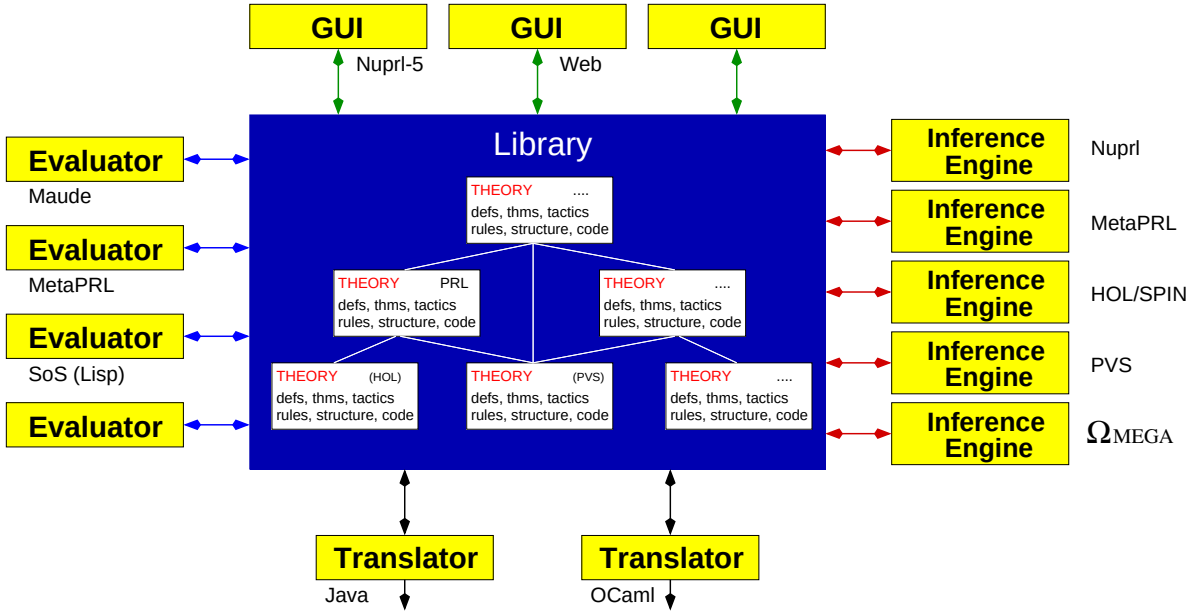
THM cases

```
# top
⊢ T

BY Cases [A;B]

# 1
⊢ A∨B

# 2
1. A
⊢ T

# 3
1. B
⊢ T
```

THM cases in steps

```
# top
⊢ T

BY cut 1 A∨B

# 1
⊢ A∨B

# 2
1. A∨B
⊢ T

BY orE 1

# 2 1
1. A
⊢ T

# 2 2
1. B
⊢ T
```

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)

ANHANG

Theorie endlicher Mengen
Formalisierung wichtiger abgeleiteter Begriffe

$$
\begin{array}{lcl}
\texttt{empty?} & \equiv & \lambda\texttt{S. if S=}\emptyset\texttt{ then tt else ff} \\
\subseteq & \equiv & \lambda\texttt{S,S'. }\forall\texttt{x}\in\texttt{S.x}\in\texttt{S'} \\
\{\textit{list-exp}\} & \equiv & \textit{list-exp}\texttt{.nil} \\
\{i\,..\,j\} & \equiv & \texttt{ind}(j\text{-}i;\ \_,\_.\emptyset;\ \{j\};\ \texttt{diff,j-set. j-set+}(j\text{-}\texttt{diff})) \\
\{f_x\,|\,x\in S\wedge p_x\} & \equiv & \texttt{list\_ind}(S;\ \emptyset;\ \texttt{a,\_,GSF. if }p_x[\texttt{a}/x]\texttt{ then GSF+}f_x[\texttt{a}/x]\texttt{ else GSF} \\
|S| & \equiv & \texttt{list\_ind}(S;\ \texttt{0};\ \texttt{a,S',card. if a}\in\texttt{S'}\ \texttt{ then card }\ \texttt{else card+1}) \\
\texttt{-} & \equiv & \lambda\texttt{S,a. }\{\texttt{x}\,|\,\texttt{x}\in\texttt{S}\wedge\texttt{x}\neq\texttt{a}\} \\
\cup & \equiv & \lambda\texttt{S,S'. list\_ind(S'; S; a,\_,union.union+a)} \\
\cap & \equiv & \lambda\texttt{S,S'. }\{\texttt{x}\,|\,\texttt{x}\in\texttt{S}\wedge\texttt{x}\in\texttt{S'}\} \\
\setminus & \equiv & \lambda\texttt{S,S'. }\{\texttt{x}\,|\,\texttt{x}\in\texttt{S}\wedge\texttt{x}\notin\texttt{S'}\} \\
\bigcup & \equiv & \lambda\texttt{FAMILY. list\_ind(FAMILY; }\emptyset\texttt{; S,FAM,Union.Union}\cup\texttt{S)} \\
\bigcap & \equiv & \lambda\texttt{FAMILY. list\_ind(FAMILY; fail;} \\
 & & \qquad\texttt{S,FAM,inter. if empty?(FAM) then S else inter}\cap\texttt{S)} \\
\texttt{map} & \equiv & \lambda\texttt{f,S. }\{\texttt{f(x)}\,|\,\texttt{x}\in\texttt{S}\} \\
\texttt{reduce} & \equiv & \lambda\texttt{op,S. list\_ind(S; fail;} \\
 & & \qquad\texttt{a,S',redS'. if empty?(S') then a} \\
 & & \qquad\qquad\texttt{else if a}\in\texttt{S' then redS' else op(redS',a))} \\
T =_{Set} S \uplus S' & \equiv & T =_{Set} S\cup S'\ \wedge\ \texttt{empty?}(S\cap S')
\end{array}
$$

Theorie endlicher Mengen (Auszug)
Fundamentale Gesetze abgeleiteter Konzepte

Lemmata zu "$\in$"

1. $a' \in \{a\} \Leftrightarrow a'=a$
2. $k \in \{i..j\} \Leftrightarrow (i \leq k \wedge k \leq j)$
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. $empty?(\bigcup FAM) \Rightarrow empty?(\bigcap FAM)$

Begriffe für Anwendungen in der Programmsynthese

$\mathbb{B}$, 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], [$a_1 \ldots a_n$]	Empty/ singleton sequence, subrange, sequence former
a.L, L·a	prepend a, append a to L
[f(x)|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}, {$a_1 \ldots a_n$}	Empty set, singleton set, integer subset, set former
S+a, S-a	element addition, element deletion
{f(x)|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