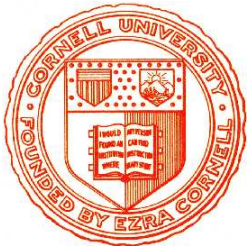


# Automatisierte Logik und Programmierung

## 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

- **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:  $\text{Set}(\alpha)$

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

$+: \text{Set}(\alpha) \times \alpha \rightarrow \text{Set}(\alpha)$

$\in: \alpha \times \text{Set}(\alpha) \rightarrow \text{Bool}$

Gesetze:  $a \notin \emptyset$

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

$(S+a)+x = (S+x)+a$

$(S+a)+a = S+a$

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

- **Implementierung durch Restklassen von Listen**

$\emptyset \equiv \text{nil}$

$+$   $\equiv \lambda a, S. a.S$

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

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

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

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

## ● Formale Definition abgeleiteter Begriffe

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

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

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

## ● Fundamentale Gesetze abgeleiteter Konzepte

### Lemmata zu “ $\in$ ”

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

### Lemma zu “ $\text{empty?}$ ”

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

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

- **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 kompiliert
  - + 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
  - Synchronisation, 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



- **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, ...

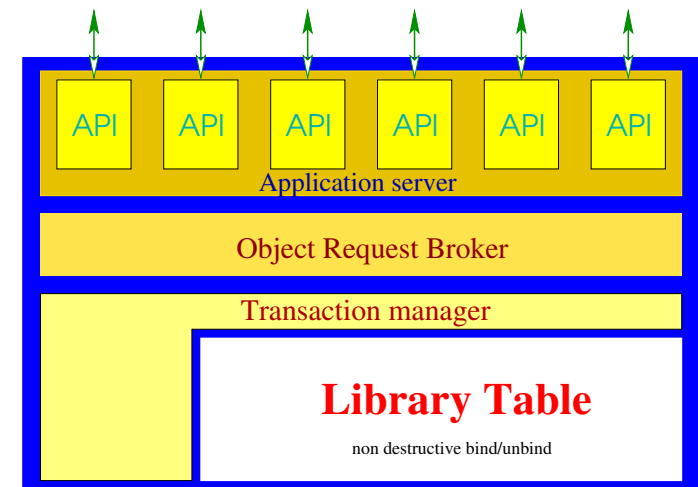
**Extra:** Abhängige Objekten, interne Id, sichtbare Position, ...

**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 Operatortabelle**

- Basisterme der Theorie werden als Primitive deklariert (vgl. §12, Folie 13)

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

- Benutzerdefinierte Objekte werden durch andere Terme erklärt

```
- ABS: exists_uni
 $\exists !x:T. P[x] == \exists x:T. P[x] \wedge (\forall y:T. P[y] \Rightarrow y=x \in T)$ 
```

## ● **Display-Objekte erweitern die Displaytabelle** (vgl. §12, Folie 14)

```
- DISP: exists_unique_df
Parens::Prec(exists):: $\exists !\langle x:\text{var} \rangle : \langle T:\text{type} \rangle * \{ \backslash ? . \} \langle P:\text{prop} : E \rangle$ 
== exists_unique(  $\langle T \rangle ; \langle x \rangle . \langle P \rangle$  )
Parens::Prec(exists):: $\exists !\langle x:\text{var} \rangle \{ \backslash ? . \} \langle P:\text{prop} : E \rangle$ 
== exists_unique(  $\mathbb{N} ; \langle x \rangle . \langle P \rangle$  )
```

## ● **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 sindNeuer 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

- **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  $\Sigma$ -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.

- **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, ...
  - 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 (aufwendig)

# NUPRL'S NAVIGATOR

- **Visuelle Navigation durch Bibliothek**
  - Keyboard- oder Maus-gesteuertes **Durchlaufen**
  - Patterngesteuerte **Namensuche**
  - **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 Repräsentationsform 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**

# 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

```
- PRF: intsqrt
① # top 1
②
③ 1. x:N
   ⊢ ∃y:N. y2≤x ∧ x<(y+1)2
④ BY NatInd 1
⑤ # 1 1
   .....basecase.....
   ⊢ ∃y:N. y2≤0 ∧ 0<(y+1)2
⑥ BY exR 「0」
   There is 1 hidden subgoal
⑤ # 1 2
   .....upcase.....
   1. x:ℤ
   2. 0<x
   3. ∃y:N. y2≤x-1 ∧ x-1<(y+1)2
   ⊢ ∃y:N. y2≤x ∧ x<(y+1)2
⑤ BY
```

```
- PRF: intsqrt
① # top 1 2
② .....upcase.....
③ 1. x:ℤ
   2. 0<x
   3. ∃y:N. y2≤x-1 ∧ x-1<(y+1)2
   ⊢ ∃y:N. y2≤x ∧ 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**

# 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



# VERFEINERUNGS- VS. TRANSFORMATIONSTAKTIK

## 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
```

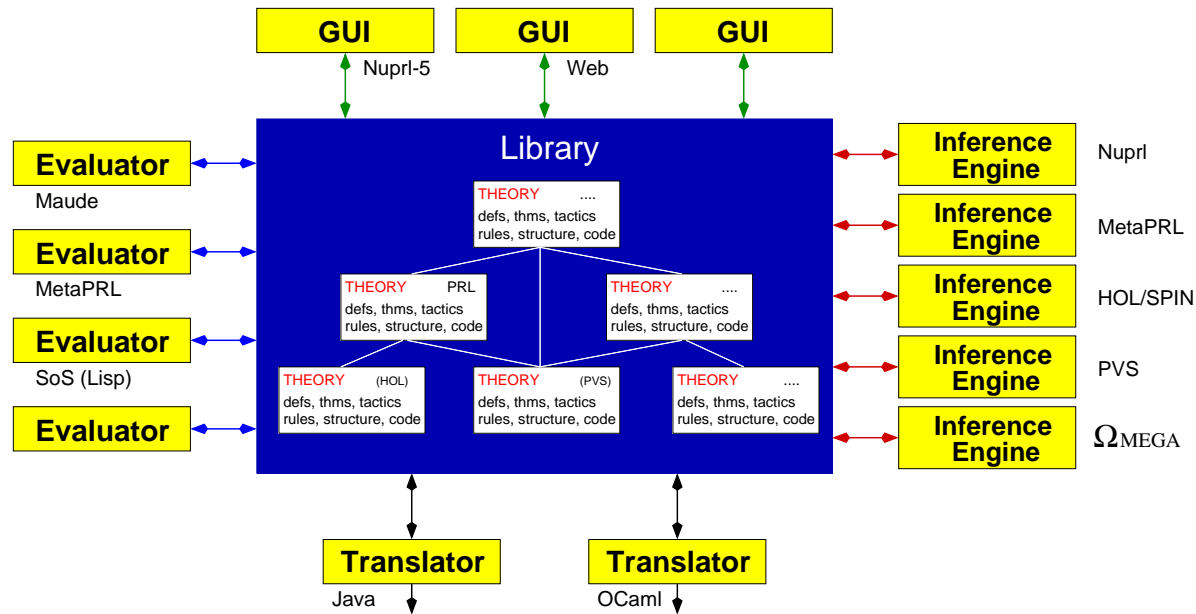
```
# 2  
1. A ∨ B  
┆ T
```

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

<b>empty?</b>	$\equiv \lambda S. \text{ if } S = \emptyset \text{ then tt else ff}$
$\subseteq$	$\equiv \lambda S, S'. \forall x \in S. x \in S'$
<b>{list-exp}</b>	$\equiv \text{list-exp.nil}$
<b>{i..j}</b>	$\equiv \text{ind}(j-i; \_, \_. \emptyset; \{j\}; \text{diff}, j\text{-set}. j\text{-set} + (j\text{-diff}))$
<b>{f<sub>x</sub>   x ∈ S ∧ p<sub>x</sub>}</b>	$\equiv \text{list\_ind}(S; \emptyset; a, \_, \text{GSF}. \text{ if } p_x[a/x] \text{ then GSF} + f_x[a/x] \text{ else GSF}$
<b> S </b>	$\equiv \text{list\_ind}(S; 0; a, S', \text{card}. \text{ if } a \in S' \text{ then card else card} + 1)$
-	$\equiv \lambda S, a. \{x   x \in S \wedge x \neq a\}$
∪	$\equiv \lambda S, S'. \text{list\_ind}(S'; S; a, \_, \text{union}. \text{union} + a)$
∩	$\equiv \lambda S, S'. \{x   x \in S \wedge x \in S'\}$
\	$\equiv \lambda S, S'. \{x   x \in S \wedge x \notin S'\}$
∪	$\equiv \lambda \text{FAMILY}. \text{list\_ind}(\text{FAMILY}; \emptyset; S, \text{FAM}, \text{Union}. \text{Union} \cup S)$
∩	$\equiv \lambda \text{FAMILY}. \text{list\_ind}(\text{FAMILY}; \text{fail};$ $S, \text{FAM}, \text{inter}. \text{ if empty?}(\text{FAM}) \text{ then } S \text{ else inter} \cap S)$
<b>map</b>	$\equiv \lambda f, S. \{f(x)   x \in S\}$
<b>reduce</b>	$\equiv \lambda \text{op}, S. \text{list\_ind}(S; \text{fail};$ $a, S', \text{redS}'. \text{ if empty?}(S') \text{ then } a$ $\text{ else if } a \in S' \text{ then redS}' \text{ else op}(\text{redS}', a))$
<b>T =<sub>Set</sub> S ⊕ S'</b>	$\equiv T =_{\text{Set}} S \cup S' \wedge \text{empty?}(S \cap S')$

# THEORIE ENDLICHER MENGEN (AUSZUG)

## FUNDAMENTALE GESETZE ABGELEITETER KONZEPTE

### Lemmata zu “ $\in$ ”

1.  $a' \in \{a\} \Leftrightarrow a' = a$
2.  $k \in \{i..j\} \Leftrightarrow (i \leq k \wedge k \leq j)$
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.  $a \in \bigcup \text{FAM} \Leftrightarrow \exists S \in \text{FAM}. a \in S$
10.  $a \in \bigcap \text{FAM} \Leftrightarrow \forall S \in \text{FAM}. a \in S$
11.  $\text{arb}(S) \in S$
12.  $b \in \text{map}(f, S) \Leftrightarrow \exists x \in S. b = f(x)$

### Lemma zu “empty?”

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

# 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
$\text{Seq}(\alpha)$	Data type of finite sequences over members of $\alpha$
null?, $\in$ , $\subseteq$	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 $\{L[i] \mid i \in \text{domain}(L)\}$
nodups(L)	Decision procedure: all $L[i]$ are distinct (no duplicates)
$\text{Set}(\alpha)$	Data type of <i>finite</i> sets over members of $\alpha$
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
$S \cup T$ , $S \cap T$ , $S \setminus T$	Union, intersection, set difference
$\bigcup \text{FAMILY}$ , $\bigcap \text{FAMILY}$	Union, intersection of a family of sets