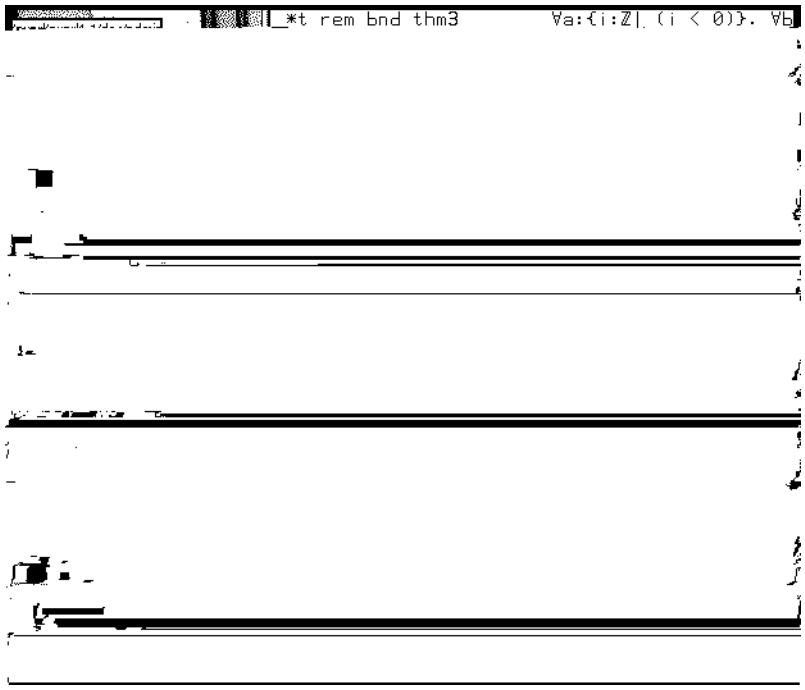
Nuprl Tutorial

Jason Hickey
Cornell University
February 2, 1997

Outline

- What are:
  - the goals
  - the philosophy
  - the domain
  - *of NuPRL*
- Interactive Theorem Proving
- System
  - Architecture
  - Logic “engine”
  - User interface



Philosophy

- Formalize & implement *mathematics* and *computation*
- *Assist* the implementor
- Logic/type theory is the formal language of choice
  - Possible to derive algorithms without ever writing a program
  - Programming comes as a by-product
- Truth of statements is undecidable
  - Proofs are by interaction
  - A great deal of *assistance* is available (checking, prompting, documenting)
  - Some *automation* available
  - System goals:
    - develop automation as well as math domains
    - make the user interface comfortable

Logic & Type Theory

- How can mathematics be formalized?
- Use a logic with statements, assertions, and inference rules
- Constructive type theory: higher order logic + computation
  - Types provide specifications for programs
  - Types are assertions
  - Programs can be derived from proofs, or can be shown to inhabit types
- Types (specifications/assertions/propositions):
  - Void, Int, * list
  - Equality
  - Functions ($x{:}A \rightarrow B$), Products ($x{:}A \times B$), Disjoint Unions ($A + B$)
  - Type universes (Ui is all types at level i, $Ui \in U\{i + 1\}$)

Types I

- Function space
  - $\forall i{:}N.\ i \geq 0$
  - $i{:}N \rightarrow i \geq 0$
  -

Types II

- Type universes

  Ui is all types/propositions to level i

  Z, Void U1

  – ∈ Ui x:A → B ∈

  1 P:N → Ui. P(0) ∀ i) P(i + 1)) ⇒ (∀

  ∈

  1 car:Ui × χαΘ 0.549 0 TΔ 6 μεμβερ: Z Y{ι + 1}

- Equality types

  a = b

  – T is a well-formed type

  21 a and b are well-formed elements of T

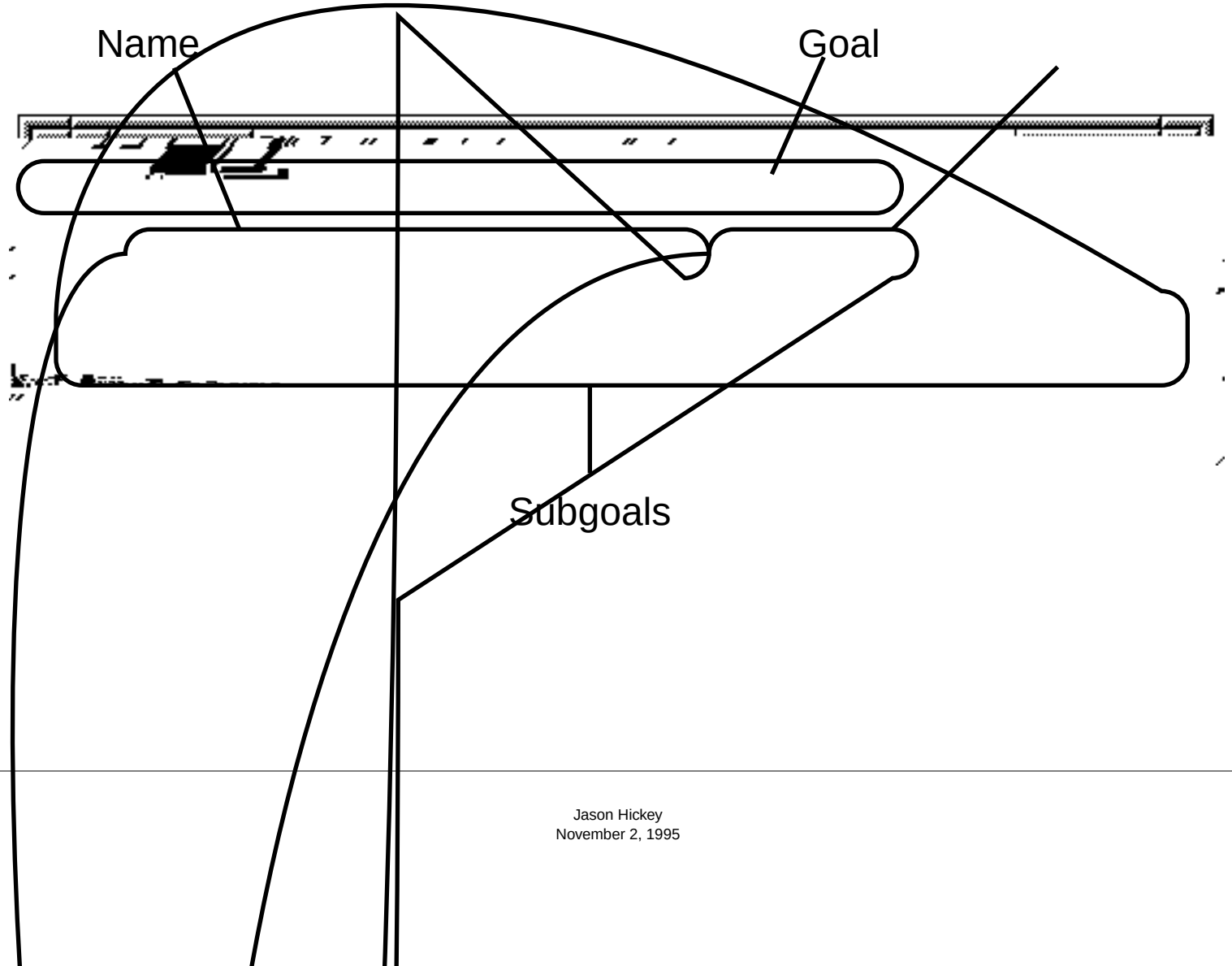
  membership: a = a

Sequents

- Sequent: $a_1{:}H_1, a_2{:}H_2, \ldots, a_n{:}H_n \vdash G$
  - H_1 is a type
  - for any $a_1 \in H_1$, $H_2[a_1]$ is a type
  - •••
  - for any $a_1 \in H_1, \ldots, a_{n-1} \in H_{n-1}[a_1, \ldots, a_{n-2}]$, $H_n[a_1,\ldots,a_{n-1}]$ is a type
  - for an a $_1 \in H_1, \ldots, a_n \in H_n[a_1, \ldots, a_{n-1}]$, $G[a_1, \ldots, a_n]$ is a type, and it is true
- Functionality
  - for any $a_1, b_1 \in H_1$, $H_2[a_1]$ and $H_2[b_1]$ are equal types
  - •••

Rules

- A rule is an implication on sequents:



Syntax Examples

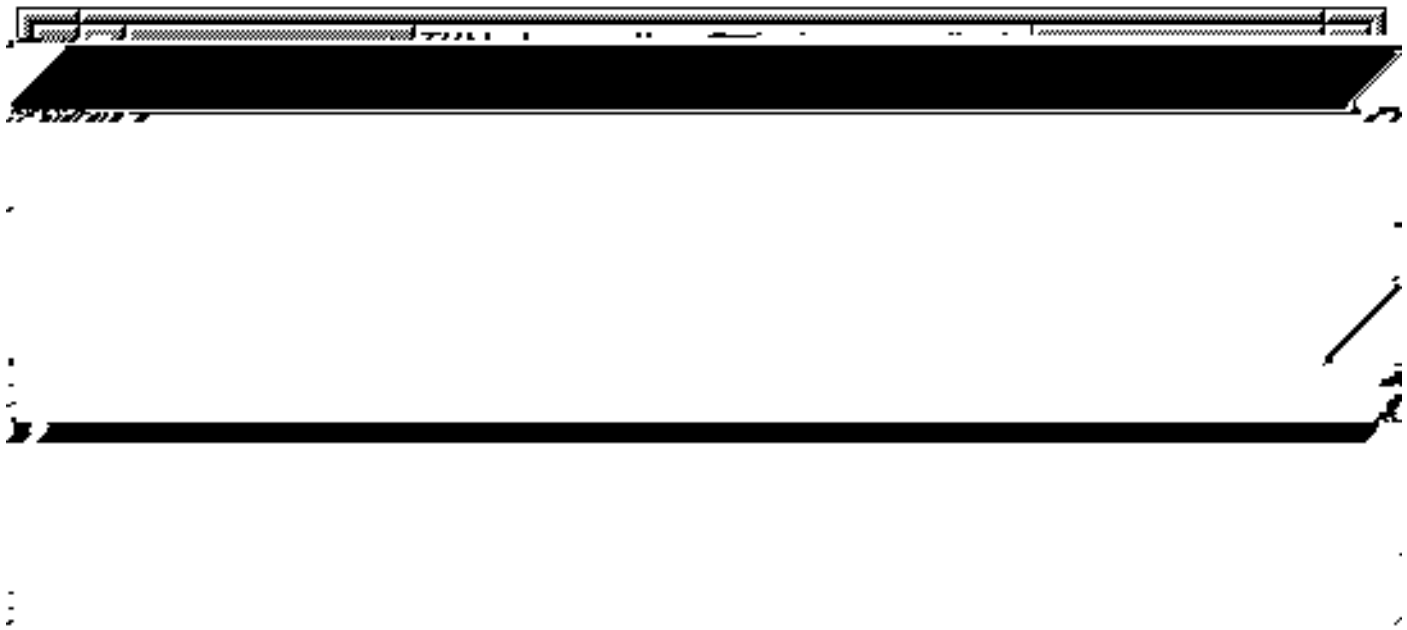
- Number: `1` ≡ `natural_number{1:n}`
- Variable: `x` ≡ `variable{x:v}`
- Summation: `i + j` ≡ `add{}(.variable{i:v}; .variable{j:v})`
- Abstraction: λx.x + 1 ≡ `lambda{}(x.x + 1)`

Jason Hickey
November 2, 1995

User Model

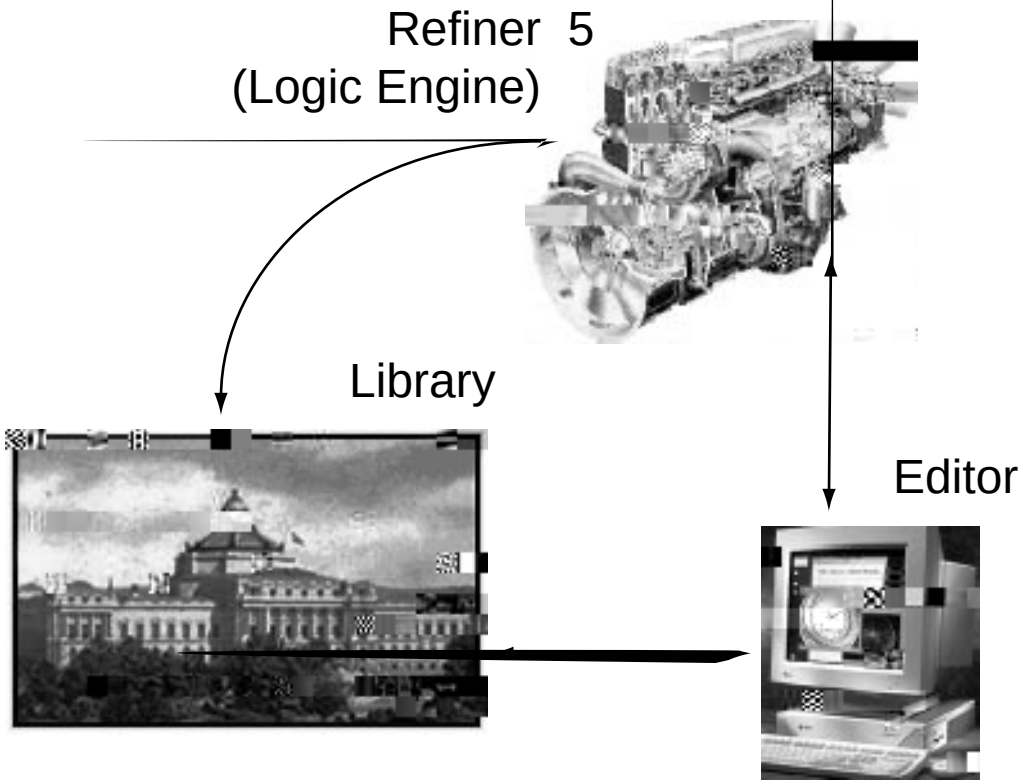
- The user builds domains by:
  - creating definitions of:
    -
    - properties
  - verifying properties in *theorems*
- The interface is interactive
  -
  -
  -
    - Supply the goal of a theorem
    - Refine it with a rule or a tactic to generate subgoals
    - Prove the subgoals

Example Proof Step



System Architecture

- Refiner enforces the logic
- Library mantains database of definitions and inference steps
- Editor provides a user interface

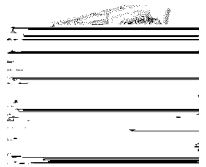


Refiner

-

Tactics

- Refiner may also provide a *tactic* language
- In NuPRL, the language is ML
- Tactics can be programmed to analyze goals, and perform more intelligent tasks



Styles of reasoning in NuPRL

- Proofs are interactive, and refined
- Prove something is true (inhabited by some program)
- Prove something is well-formed (that it is a sensible assertion)
  - Well-formedness is a major component of proving
  - The well-formedness a cking is delayed
  - A theorem must be verified to be well-formed as it is proved
  - *Example:* to prove A $\Rightarrow$ B, prove A is a proposition, then assume A and prove B
  - This style differs from other major type theories
  - Well-formedness is undecidable, but the type system is quite expressive
  - Not true that if a term is well-formed, then so is every subterm
    - ♦ Void $\rightarrow$ (1 + Z)

Provided Tactics

- `Auto : tactic`
  - Performs default well-formedness reasoning (gets most cases)
  - Performs simple logical reasoning
  - Tries to limit search so that every step gets closer to a proof
  - “Decompose” a clause
  - on 0, performs an *introduction*
  - on a hyp, performs an *elimination*, case anlysis or induction
- `NatInd : int  tactic`
  - better form of elimination on natural numbers
  - Perform reasoning about arithmetic
  - Linear systems of equations

Jason Hickey
November 2, 1995

Provided Tacticals

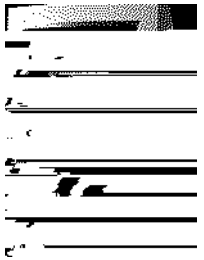
- A tactical is a function taking a tactic as an argument
- `Tac1 THEN Tac2`
  - Run Tac1 on goal, then run Tac2 on all subgoals
- `Tac1 THENW Tac2`
  - Run Tac1 on goal, then run Tac2 on all well-formedness subgoals
- `Tac1 ORELSE Tac2`
  - Run Tac1 on goal. If it fails, run Tac2
- Also have:
  - functions to examine terms
  -

Editor

- What are the objects of the system?
  - Definitions
  - Rules

    Theorems
  - Comments

    Code



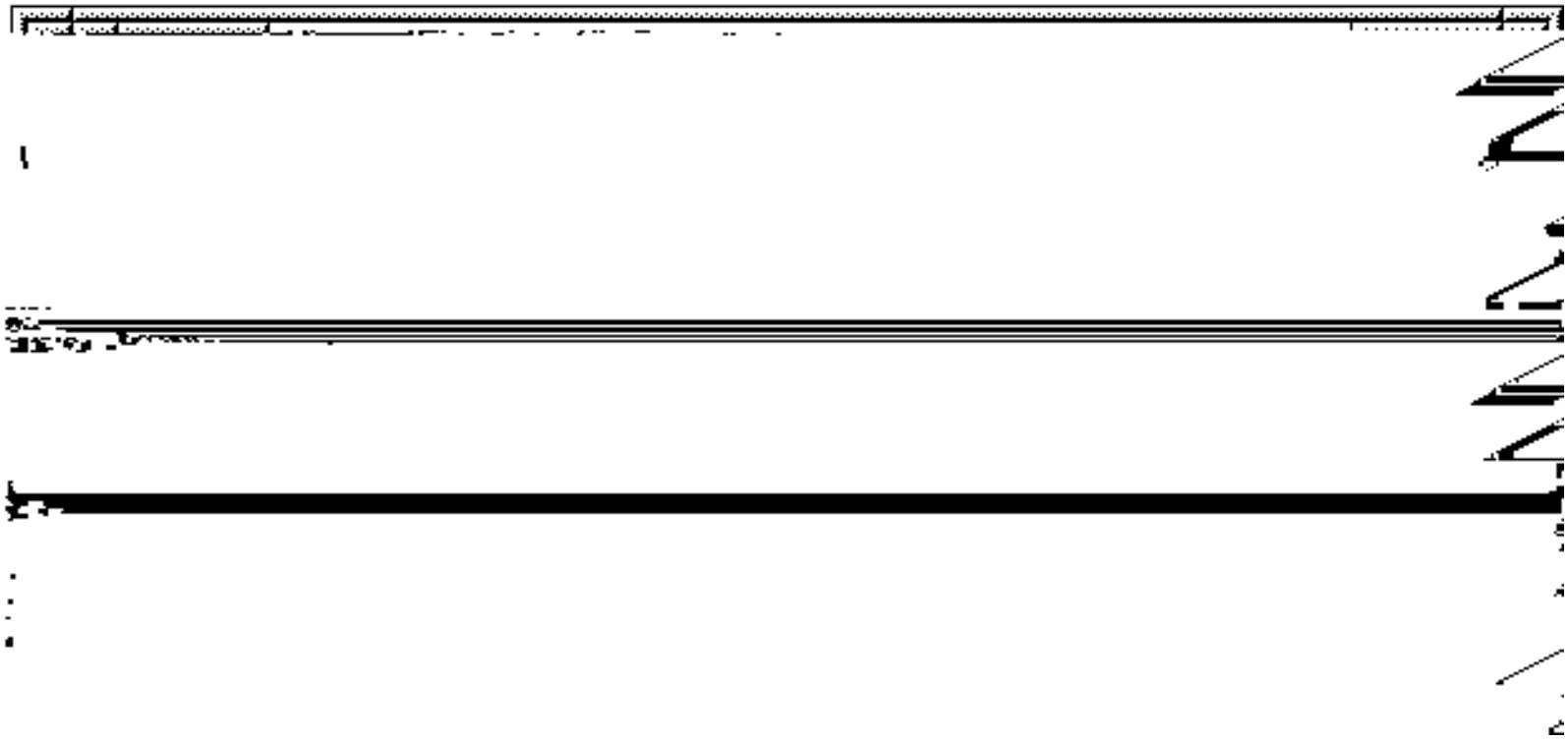
Library

- Library window displays loaded objects
- Organized into *theories*
  - Linear list of items

    Begins with comment object called **name**_`begin`
  - Ends with comment object cal.67d**name**_`end`
- One liner for each item
  - **Status** *: correct and complete, #: partial, -: inco 0ect–

    Type
    - C: comment, D: display form, T: theorem, A: abstraction, R: rule, M: ML code
    - lower case if object has not been checked
  - **Synopsis** first line of contents

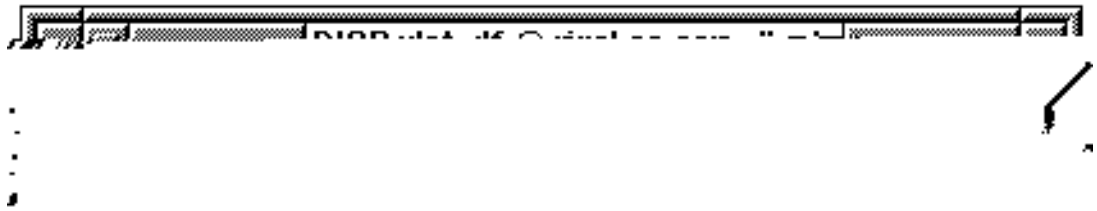
Library Window



Jason Hickey
November 2, 1995

Display Forms

- Terms may be:
  - **primitive**, like λv.b (`lambda{}(v.b)`), or *Z* (`int{}()`)
  - **defined**, like `let v = e in b`
    `let{}(.e; v.b)` → `(λv.b)(e)`
- A display form may be defined for any term
- Specifies how the term should be printed



LHS contains printing directives

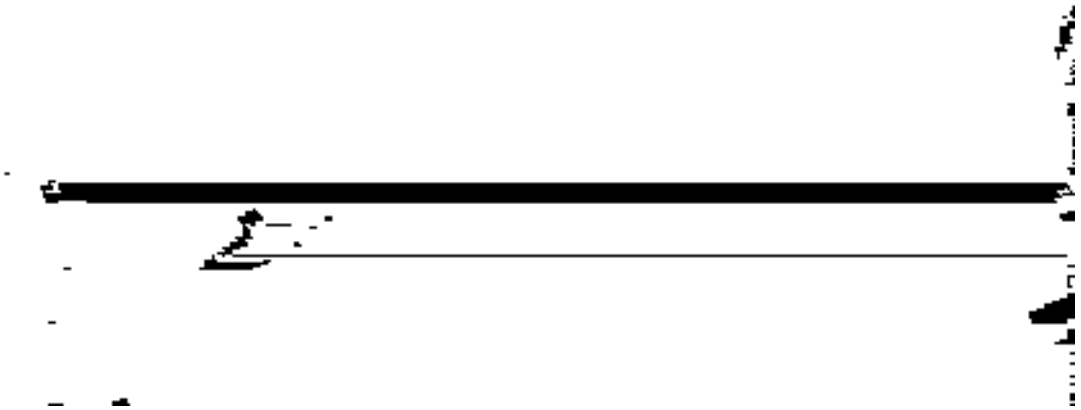
- Slots describe areas for terms
- When the term is constructed, the user is prompted for input in the slots
- Special instructions for line breaking, parenthesizathe sn, etc.

Abstractions

- Definitions are made through *a24 Qactions*

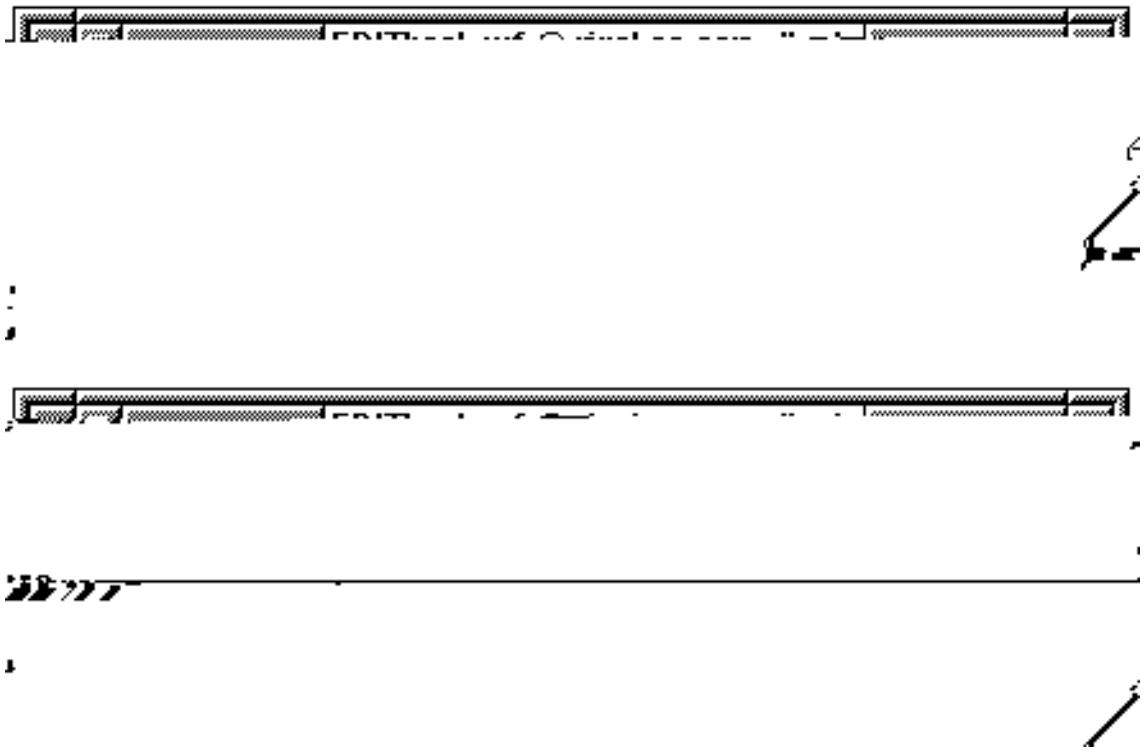
the defined/definition; the redex/contractum

- Often used to define more complex types:

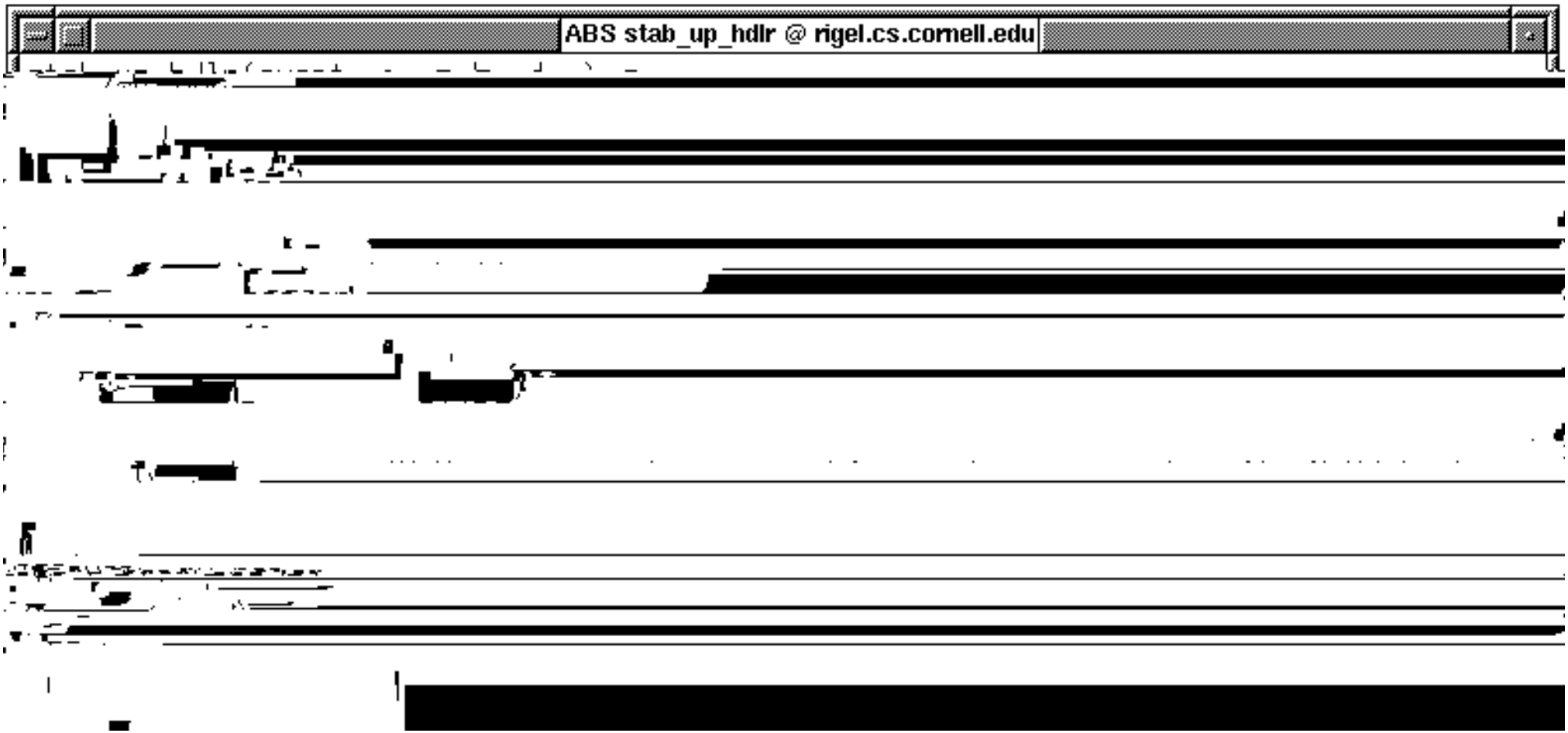


Abstraction Well-formedness

- Abstractions usually have a well-formedness theorem

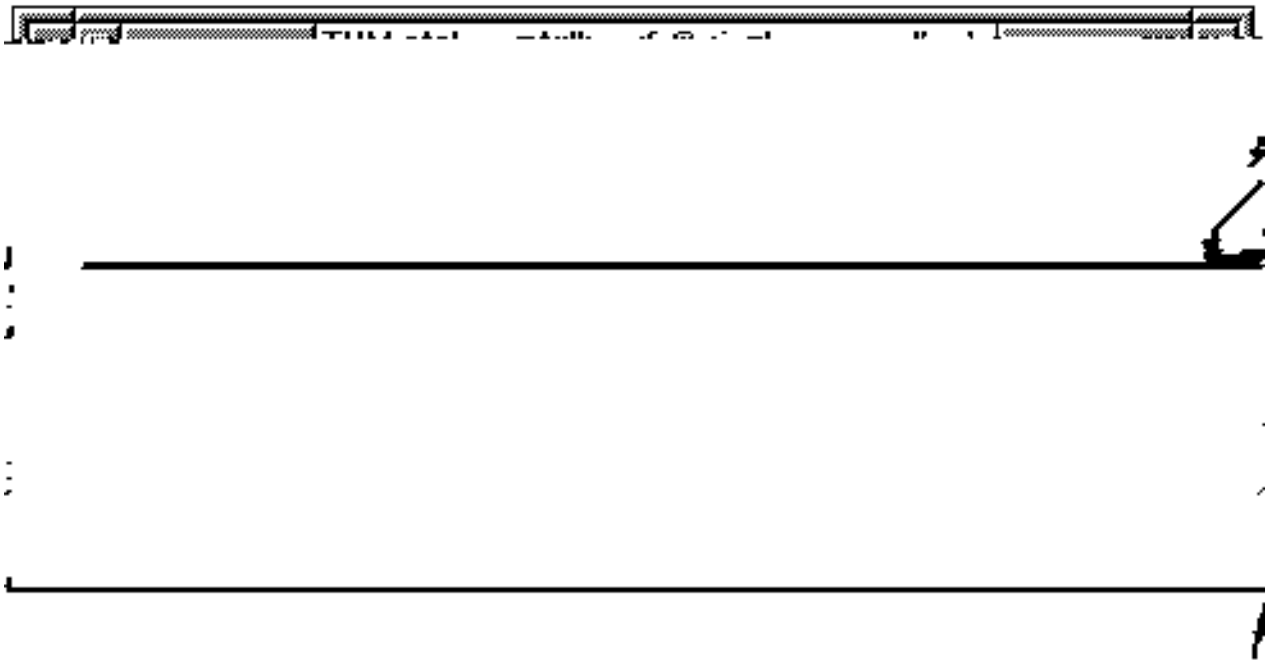


Larger Example



Large Well-formedness

- Well-formedness is quantified by types of arguments

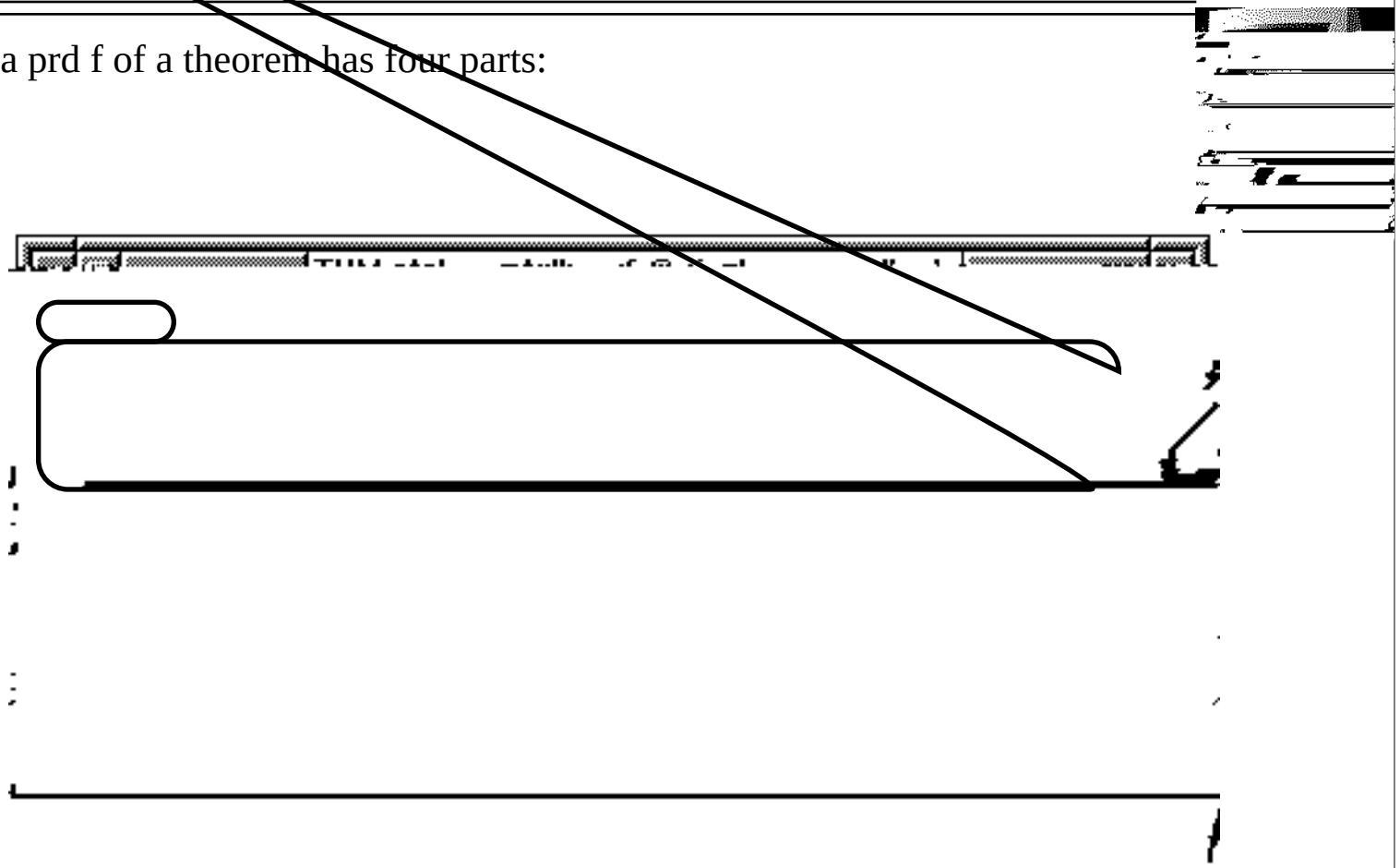


Term Editor Commands

- Large assortment of commands for editing terms
- Use either keyboard or mouse
- To get a new term, type the name of its display form (usually the operator name)
  - ^F moves *forward*, ^B moves *back*
  - ^P moves *up* the term tree
  - ^O *opens*

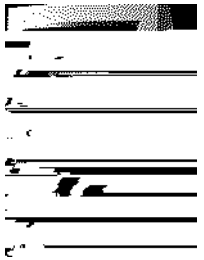
Refinement (Prd) Editor

- A window into a prd f of a theorem has four parts:



Refinement

- ^O opens goal or subgoal term or rule box
- ^O edits the main goal of a theorem
- Type tactic into rule box
- ^Z closes and *checks* a term or rule box
  - Set the goal
  - Runs the tactic to produce subgoals



Summary

- Nuprl is a system for developing formal mathematics
- Formalism is based on *type theory* (logic + computation)
- System is designed to assist and automate the formalization
- Three parts:
  - Refiner (for checking and inferring proofs)
  - Library (for storing rules, definitions, and proofs)
  - Editor (human interface for editing object in the library)
- Refiner uses ML as a *tactic* language
- Editor works directrly on the term tree
  - Structured editing
  -

Where to go from here

- Documentation
  - Nuprl book
  - Web site (http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html)
    - Nuprl book
    - Nuprl 4.2 reference manual
    - Nupme 4.2 tutorial
    - Library browser
- To use Nupml
  - ExNo ecute `~nuprl/bin/run-nuprl`
  - Use large SunOS machine, like **gemini** or **virgo**
  - `xhost` the machine you are running from
  - It does run on Solaris, just not officially supported