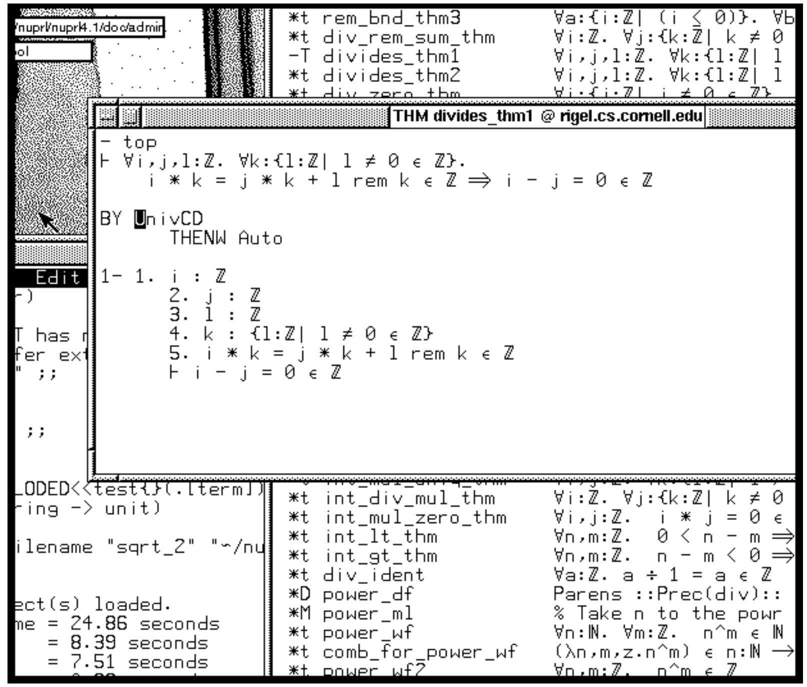
Nuprl Tutorial

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



Jason Hickey
November 2, 1995

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

Jason Hickey
November 2, 1995

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, $Ui \in U\{i + 1\}$)

Jason Hickey
November 2, 1995

Types I

- Function space
  - $\forall i{:}N.\ i \geq 0$
  - $i{:}N \rightarrow i \geq 0$
  - $i{:}N \rightarrow \{i\ldots\}$
- Product space
  - $\exists i{:}N.$ TM(i) halts
  - $\forall i{:}N.\ \exists j{:}N.\ j * j \leq i \wedge (j + 1) * (j + 1) > i$
  - $i{:}N \rightarrow j{:}N \times (j * j \leq i) \times ((j + 1) * (j + 1) > i)$
- Disjoint union
  - $A + B$
  - $\forall i{:}N.$ TM(i) halts $\vee \neg$TM(i) halts

Jason Hickey
November 2, 1995

Types II

- Type universes
  - Ui is all types/propositions to level i
  - Z, Void $\in$ U1
  - A, B $\in$ Ui $\Rightarrow$ x:A $\rightarrow$ B $\in$ Ui, etc
  - $\forall$P:N $\rightarrow$ Ui. P(0) $\Rightarrow$ ($\forall$i:N.P(i) $\Rightarrow$ P(i + 1)) $\Rightarrow$ ($\forall$i:N. P(i))
  - Ui $\in$ U{i + 1}
- Set
  - car:Ui $\times$ insert:Z $\rightarrow$ car $\times$ member: Z $\rightarrow$ car $\rightarrow$ Ui $\times$ … $\in$ U{i + 1}
- Equality types
  - a = b $\in$ T
  - T is a well-formed type
  - a and b are well-formed elements of T
  - a and b are equal elements in T
  - membership: a = a $\in$ T means a $\in$ T

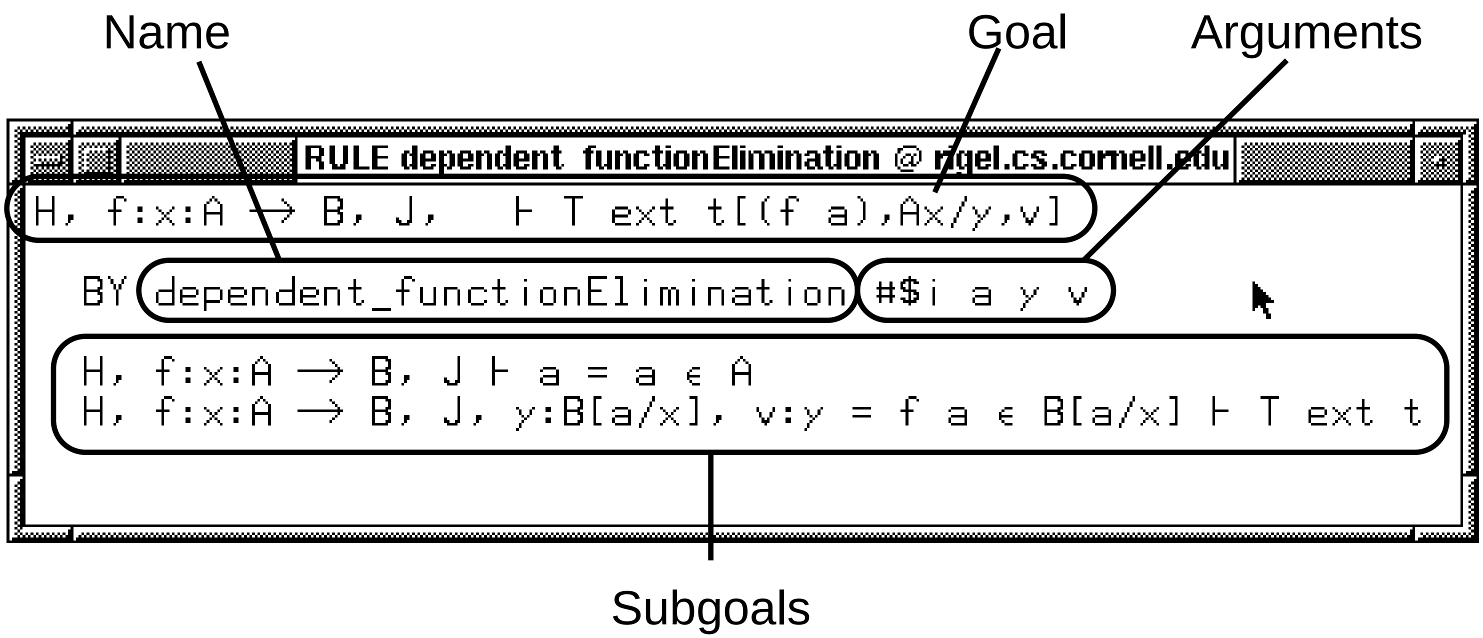
Sequents

- Sequent: a_1:H_1, a_2:H_2, …, a_n:H_n |— G
  - H_1 is a type
  - for any $a_1 \in H_1$, $H_2[a_1]$ is a type
  - •••
  - for any $a_1 \in H_1$, …, $a_{n-1} \in H_{n-1}[a_1, \ldots, a_{n-2}]$, $H_n[a_1,\ldots,a_{n-1}]$ is a type
  - for any $a_1 \in H_1$, …, $a_n \in H_n[a_1, \ldots, a_{n-1}]$, $G[a_1, \ldots, a_n]$ 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
  - •••

Jason Hickey
November 2, 1995

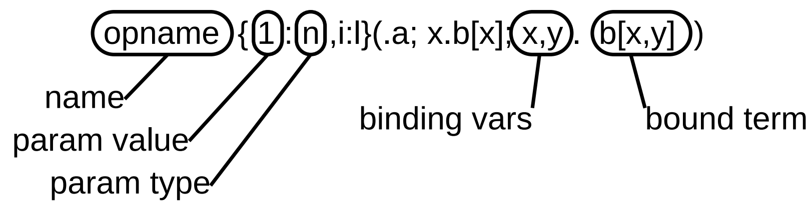
Rules

- A rule is an implication on sequents:



Syntax

- Everything is a *term*:
  - term ::= opname{params}(bterms)
  - opname ::= <string>
  - params ::= ε | paramlist
  - paramlist ::= param | paramlist , param
  - param ::= <number> : n | <string> : s | level-exp : l | …
  - bterms ::= ε | btermlist
  - btermlist ::= bterm | btermlist ; bterm
  - bterm ::= vars . term
  - vars ::= ε | varlist
  - varlist ::= var | varlist , var
  - var ::= <string>



Syntax Examples

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

Jason Hickey
November 2, 1995

User Model

- The user builds domains by:
  - creating definitions of:
    - types
    - methods
    - properties
  - verifying properties in *theorems*
- The interface is interactive
  - As opposed to batch AI first order theorem provers
  - Premises are complex
  - Backward chaining (goal oriented)
    - Supply the goal of a theorem
    - *Refine* it with a rule or a tactic to generate subgoals
    - Prove the subgoals

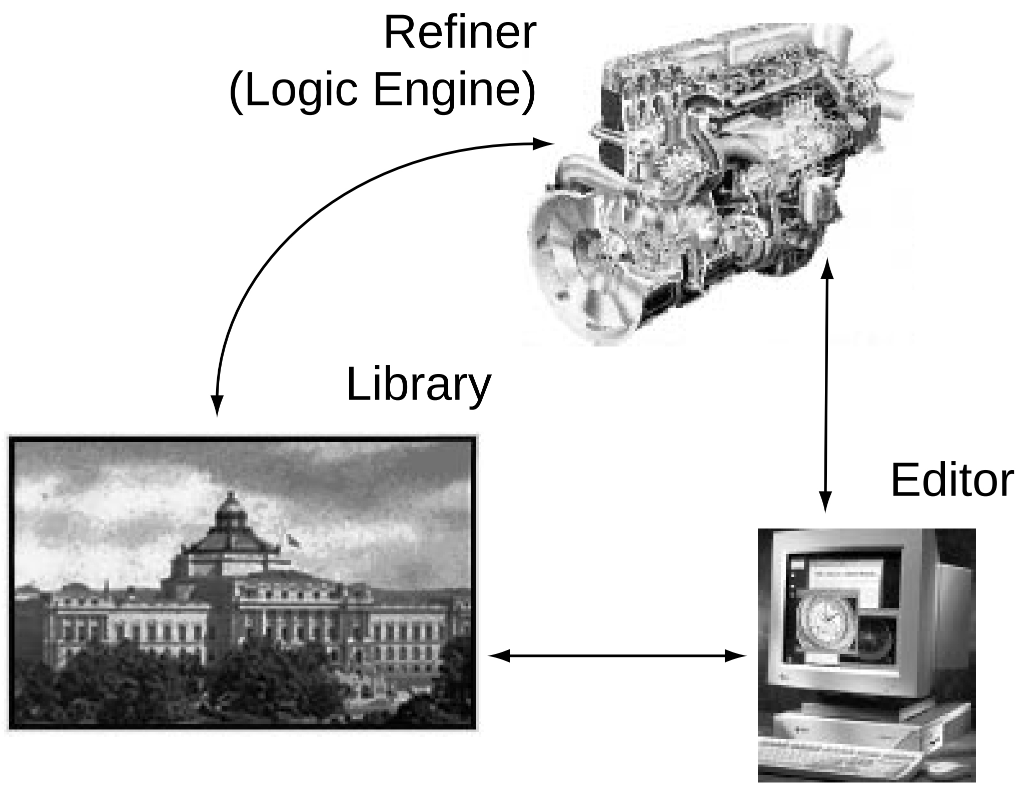
Example Proof Step

THM stamps_thm @ rigel.cs.cornell.edu

```
# top 1 2 1
1. postage : ℤ
2. 0 < postage
3. postage - 1 ≥ 8
   ⇒ (∃i,j:ℕ. 3 * i + 5 * j = postage - 1 ∈ ℤ)
4. postage ≥ 8
⊢ ∃i,j:ℕ. 3 * i + 5 * j = postage ∈ ℤ

BY Decide ⌜postage = 8 ∈ ℤ⌝
      THENW Auto

1# 5. postage = 8 ∈ ℤ
      ⊢ ∃i,j:ℕ. 3 * i + 5 * j = postage ∈ ℤ

2# 5. ¬(postage = 8 ∈ ℤ)
      ⊢ ∃i,j:ℕ. 3 * i + 5 * j = postage ∈ ℤ
```

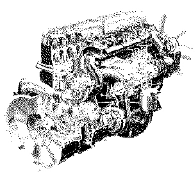
System Architecture

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



Refiner

- Performs refinement according to rule collection



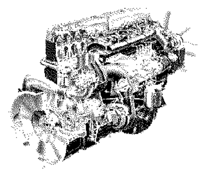
THM excluded_middle_thm @ rigel.cs.cornell.edu

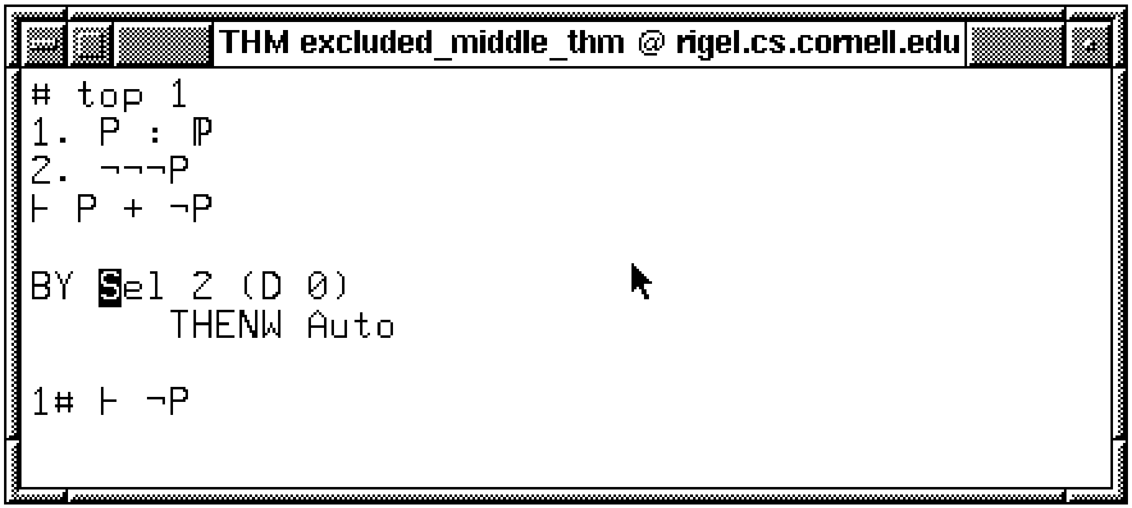
```
# top 1
1. P : ℙ
2. ¬¬¬P
⊢ P + ¬P

BY Refine `inrFormation`
          [mk_level_exp_arg (mk_level_exp [`i`, 0])]

1# ⊢ ¬P

2# ⊢ P = P ∈ 𝕌
```


Jason Hickey
November 2, 1995

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



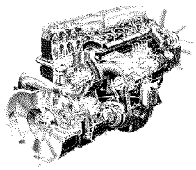


Jason Hickey
November 2, 1995

Styles of reasoning in NuPRL

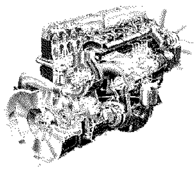
- Proofs are interactive, and refined
- Prove something is true (inhabited by some program)
- Prove something is well-formed (that it is a sensible assertion)
  - Well-formedness is a major component of proving
  - The well-formedness checking is delayed
  - A theorem must be verified to be well-formed 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
- `D : int` → `tactic`
  - “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
- `Arith : tactic` and `SupInf : tactic`
  - Perform reasoning about arithmetic
  - Linear systems of equations
- `RW : conv` → `int` → `tactic`
  - Rewrites using equivalences or implications

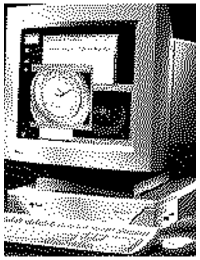
Provided Tacticals



- 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
  – functions to build terms
  – all standard functions in ML
- Tactics are secure because there is no way to construct a tactic, except from other tactics.

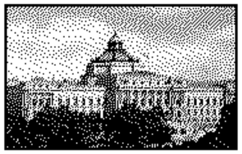
Jason Hickey
November 2, 1995

Editor



- What are the objects of the system?
  - Definitions
  - Rules
  - Theorems
  - Comments
  - Code
- All of them contain terms
- Structured editing of terms
- Each term may have a *display form*

Library



- Library window displays loaded objects
- Organized into *theories*
  - Linear list of items
  - Begins with comment object called **name**_`begin`
  - Ends with comment object called **name**_`end`
- One liner for each item
  - **Status** *: correct and complete, #: partial, -: incorrect
  - **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

Library (File: ~/nuprl/theories/horus/event_1.thy) @ rigel.cs.cornell.edu

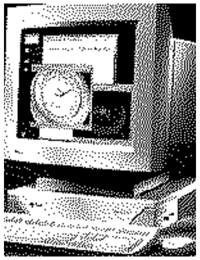
```
 *C udp_eid_start     ------ begin udp_eid ------
 *D udp_eid_type_df  UdpEidType== udp_eid_type{}
 *A udp_eid_type  UdpEidType == addr:INetAddr × port:Port × Mux
 *t udp_eid_type_wf  UdpEidType ∈ 𝕌1
 *D udp_addr_df   <t:udp:E>.addr== udp_addr{}(<t>)
 *A udp_addr      t.addr == t.1
 *t udp_addr_wf   ∀t:UdpEidType. t.addr ∈ INetAddr
|*D udp_port_df   <t:udp:E>.port== udp_port{}(<t>)
 *A udp_port      t.port == t.2.1
 *t udp_port_wf   ∀t:UdpEidType. t.port ∈ Port
 *D udp_mux_df    <t:udp:E>.mux== udp_mux{}(<t>)
```

ML top loop @ rigel.cs.cornell.edu

```
M> jump "state_1_begin"| ;;
```


Jason Hickey
November 2, 1995

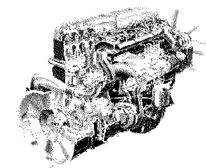
Display Forms

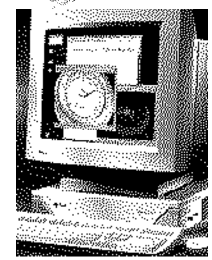


- Terms may be:
  - **primitive**, like `λv.b` (`lambda{}(v.b)`), or Z (`int{}()`)
  - **defined**, like `let v = e in b`
    `let{}(.e; v.b) → (λv.b)(e)`
- A display form may be defined for any term
- Specifies how the term should be printed

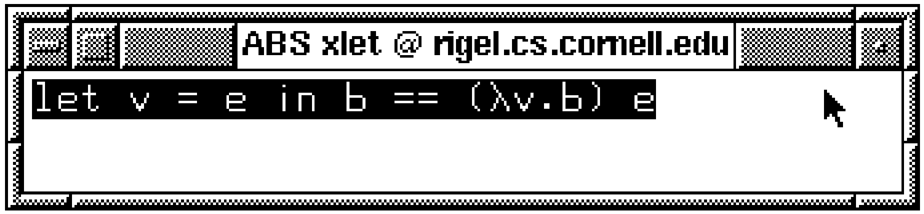
- LHS contains printing directives
- Slots describe areas for terms
- When the term is constructed, the user is prompted for input in the slots
- Special instructions for line breaking, parenthesization, etc.

Abstractions

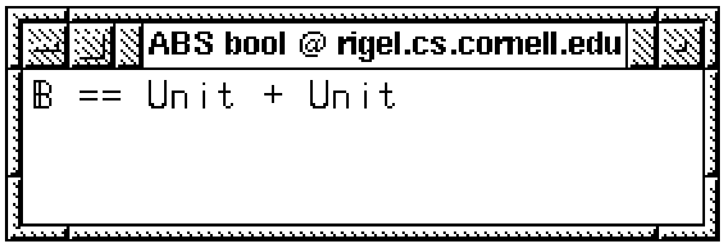


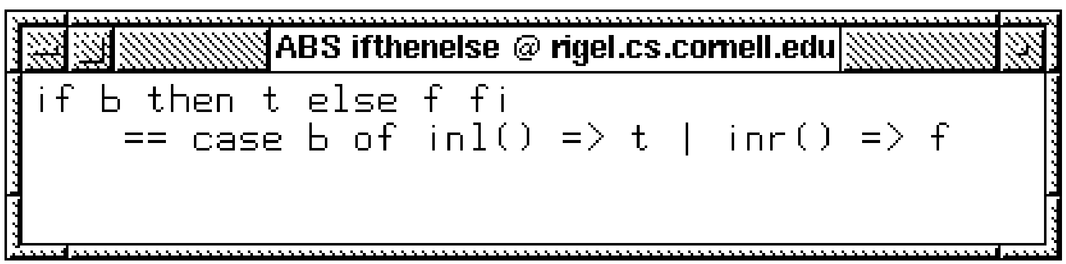


- Definitions are made through *abstractions*
- An abstraction defines a pair of terms that are:
  - the defined/definition; the redex/contractum



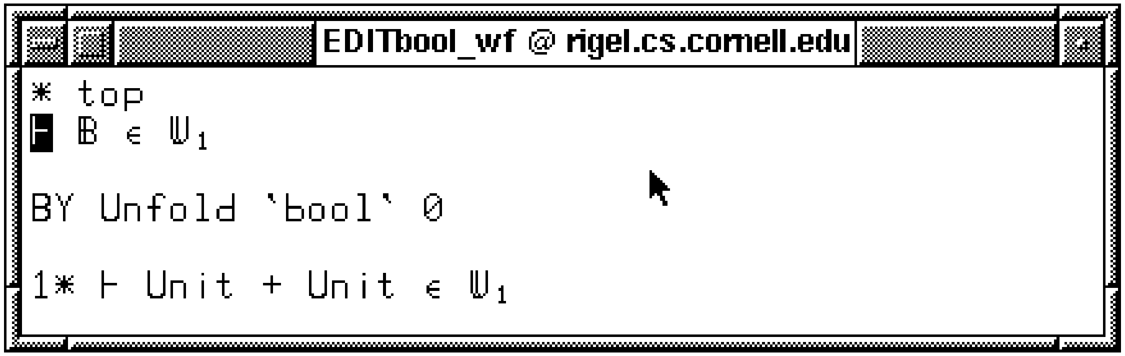
- Often used to define more complex types:

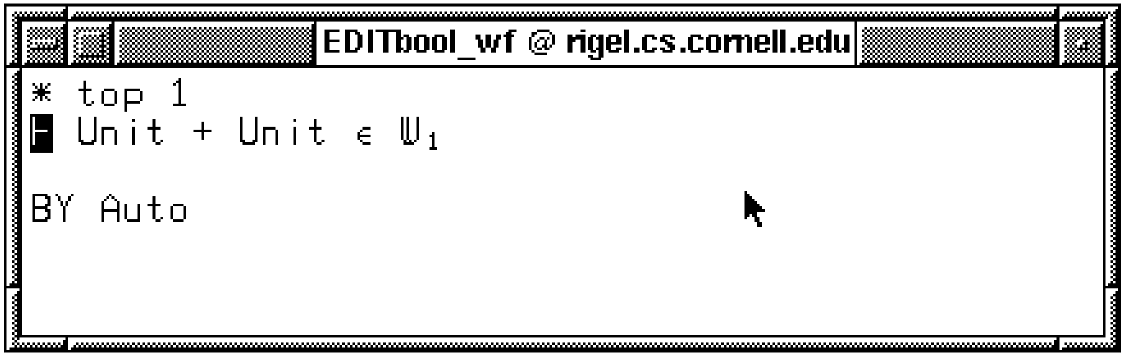




Abstraction Well-formedness

- Abstractions usually have a well-formedness theorem





Jason Hickey
November 2, 1995

Larger Example

ABS stab_up_hdlr @ rigel.cs.cornell.edu

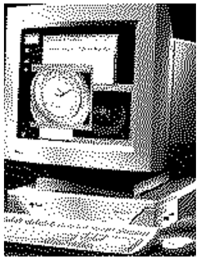
```
stab_up_hdlr(event, msg, header)
    == Case(<event, header>)
       Case <up_event(UpView, my_rank = my_rank', nmembers = nmembers'), NoHdr> =>
           my_rank ← my_rank';
           nmembers ← nmembers';
           acks ← make_vect2(nmembers, nmembers, 0);
           stable ← make_vect(nmembers, 0);
           ncasts ← make_vect(nmembers, 0);
           flushing ← ff;
           queue.append(stab_up(<event, msg>))
       Case <up_event(UpCast, origin = origin), NoHdr> =>
           /* Update #casts entry and set ack field */
           ncasts[origin]++;
           queue.append(stab_up(<UpSet(StabilityName,
                                 [upAck(Ack(<origin, ncasts[origin]>))],
                                 event), msg>))
       Case <·, NoHdr> =>
           queue.append(stab_up(<event, msg>))
       Default => /* We need a way to fail */ skip
```

Large Well-formedness

- Well-formedness is quantified by types of arguments

Jason Hickey
November 2, 1995

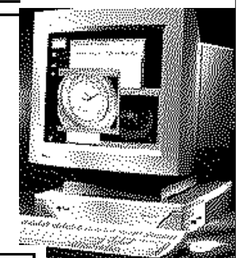
Term Editor Commands



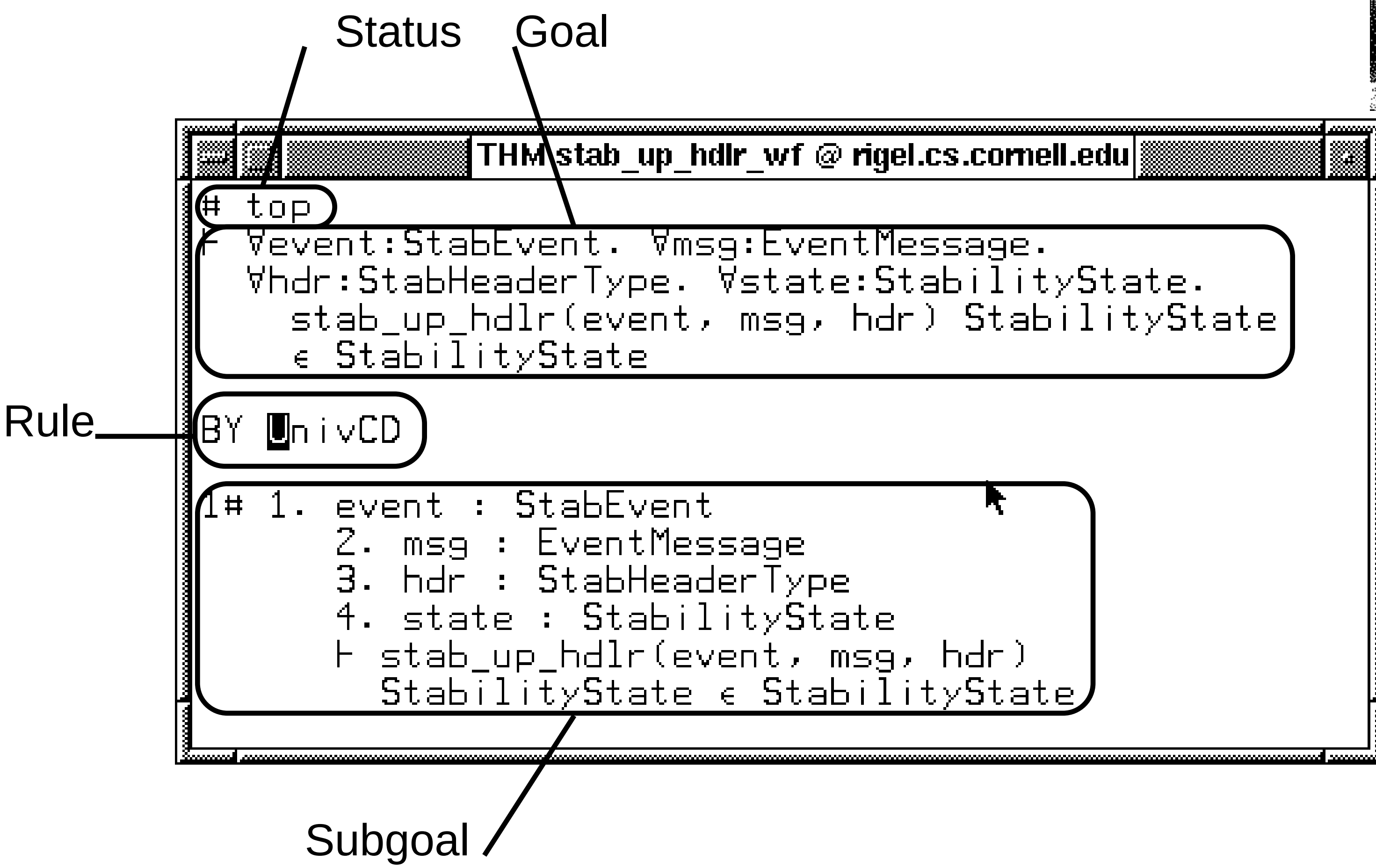
- Large assortment of commands for editing terms
- Use either keyboard or mouse
- To get a new term, type the name of its display form (usually the operator name)
  - ^F moves *forward*, ^B moves *back*
  - ^P moves *up* the term tree
  - ^O *opens* a term slot
    - in the middle to text (like tactic text)
    - in the middle of a term (to add a subterm)
  - ^Xex *explodes* a term (displays it in canonical form)
  - ^Xim *implodes* it
  - ^K *kills* a term, ^Y *yanks* it back from the kill ring

Jason Hickey
November 2, 1995

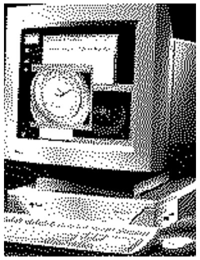
Refinement (Proof) Editor



- A window into a proof 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

Jason Hickey
November 2, 1995

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
  - Display forms for terms

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
    - ♦ Nuprl 4.2 tutorial
    - ♦ Library browser
- To use Nuprl
  - Execute `~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

Jason Hickey
November 2, 1995