



Integrating Proof Assistants as Reasoning and Verification Tools into a Scientific WYSIWYG Editor

Serge Autexier, Christoph Benzmüller, Armin Fiedler, Henri Lesourd

`serge@ags.uni-sb.de`

DFKI GmbH & CS Department, Saarland University, Saarbrücken, Germany

UITP'05, April 9th 2005

Edinburgh, Scotland

Motivation

Why have mathematical proof assistance systems not yet achieved recognition/relevance in mathematical practise (unlike CAS)?

1. No support for the language the mathematician is used to.
 - Most systems impose their own formal language on the user.
 - Require a machine-oriented formalization of the mathematical content to allow for powerful automatic inference capabilities.
 - As a result, the line of reasoning is often unnatural and obscured.
2. The proofs are at a level of excruciating detail
 - They spell out many logically necessary steps, which a human would nevertheless consider trivial or obvious.
 - Thus, the proofs are often illegible and incomprehensible.
3. Acceptance of mathematical assistant systems would be increased by integrating them with scientific WYSIWYG text editors.

Goals



- Writing a paper and using a theorem prover will be one combined task.
- Instead of developing a new user interface for Omega, adapt Omega to serve as a mathematical service provider for the scientific editor
- Analogy to spell- and grammar-checkers used in word processors, use PA to check derivations, typings, etc.
- In this talk:
 - ▶ The general approach (state based)
 - ▶ The formal representation language for math docs (with abstract proof representation language)
 - ▶ Synchronisation between Editor and PA
 - ▶ Example

Our Approach



- Preparation of math. documents in $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ in *direct* interaction with ΩMEGA
- Requires
 1. make semantic content of the document accessible for formal analysis (currently semantic annotations in the document, provided by author)
 2. interactions in either direction should be localised and aware of the surrounding context

Representation of Semantic Content



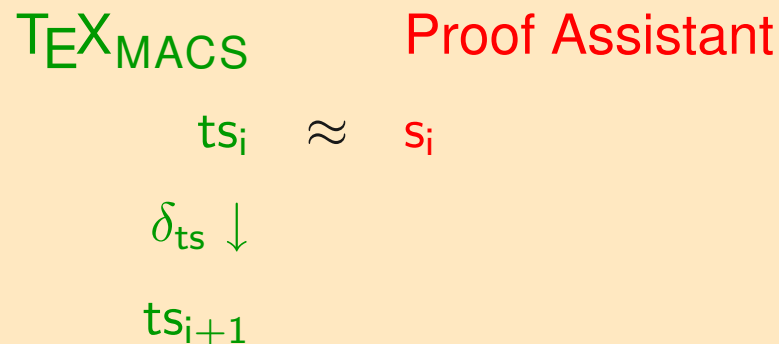
- *Both* the user and Ω MEGA shall be able to *change the document*
- Need for a *common representation format S* for the pure math. content
- Includes many notions known from *specification languages* (terms, formulas, declarations, . . .)
- means to indicate *logical context adjusting to the textual structure*; less rigid than the usual structured theories used in specification languages
- *Accommodate* various aspects of *under-specification* (details purposely omitted by the author)
- Includes a language for proofs *(CORE assertion level)*

Realisation: Augment the TexMacs document format (T) by this semantic annotations (S) (conservative)

Interaction



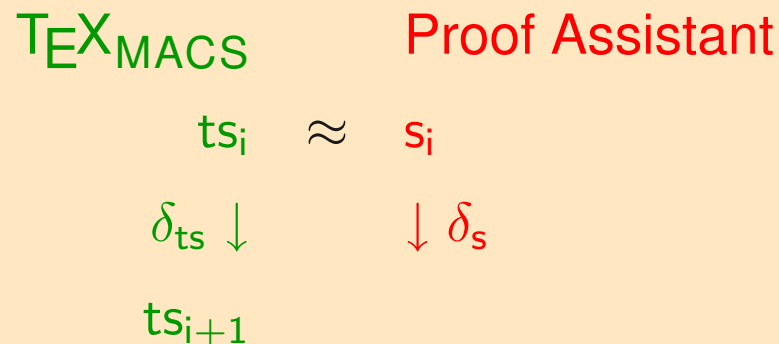
- The mathematical content of the document is represented both in $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ (in the language $T + S$) and in the proof assistant (in the language S).
- Both representations (namely in $T + S$ and in S) must be synchronised (both the user and the PA can do changes)
- This synchronisation is done using a diff/patch mechanism tailored to the tree structure of the $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ documents.
- Need to identify parts in $T + S$ with parts in S and vice-versa



Interaction



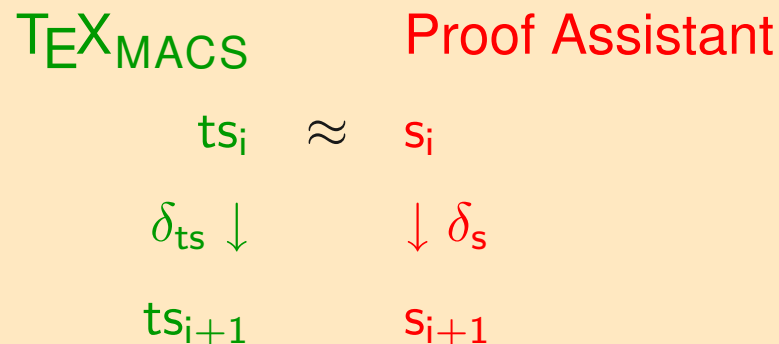
- The mathematical content of the document is represented both in $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ (in the language $T + S$) and in the proof assistant (in the language S).
- Both representations (namely in $T + S$ and in S) must be synchronised (both the user and the PA can do changes)
- This synchronisation is done using a diff/patch mechanism tailored to the tree structure of the $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ documents.
- Need to identify parts in $T + S$ with parts in S and vice-versa



Interaction



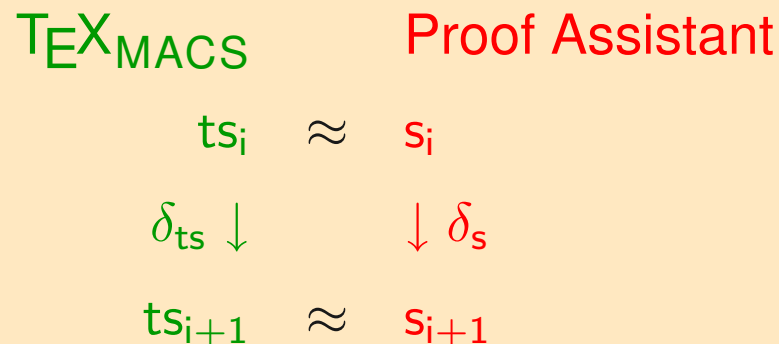
- The mathematical content of the document is represented both in $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ (in the language $T + S$) and in the proof assistant (in the language S).
- Both representations (namely in $T + S$ and in S) must be synchronised (both the user and the PA can do changes)
- This synchronisation is done using a diff/patch mechanism tailored to the tree structure of the $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ documents.
- Need to identify parts in $T + S$ with parts in S and vice-versa



Interaction



- The mathematical content of the document is represented both in $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ (in the language $T + S$) and in the proof assistant (in the language S).
- Both representations (namely in $T + S$ and in S) must be synchronised (both the user and the PA can do changes)
- This synchronisation is done using a diff/patch mechanism tailored to the tree structure of the $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ documents.
- Need to identify parts in $T + S$ with parts in S and vice-versa



Keys and Menus



- Use key based mechanism to identify corresponding parts and propagate patches
- Further include a language M for structured menus and actions
- Supports the description of specific interactions between $\text{T}_{\text{EX}}\text{MACS}$ and the PA
- Has an evaluation semantics that allows the user interface to flexibly compute
 - ▶ the necessary parameters/arguments for the commands
 - ▶ in interaction with the proof assistants.
- Augment $T + S$ from to $T + S + M$, where the menus can be attached to arbitrary parts of a document
- Changes of the documents are propagated between $T + S + M$ and $S + M$ via the diff/patch mechanism
- This allows adaptation of the menus (support context-sensitive menus and actions)



Introduction to TexMacs

- TEX_{MACS} is a WYSIWYG editor that allows for easy authoring of mathematical documents (www.texmacs.org)
- TEX_{MACS} documents can be stored as ASCII files annotated with *markup*.
- Example:

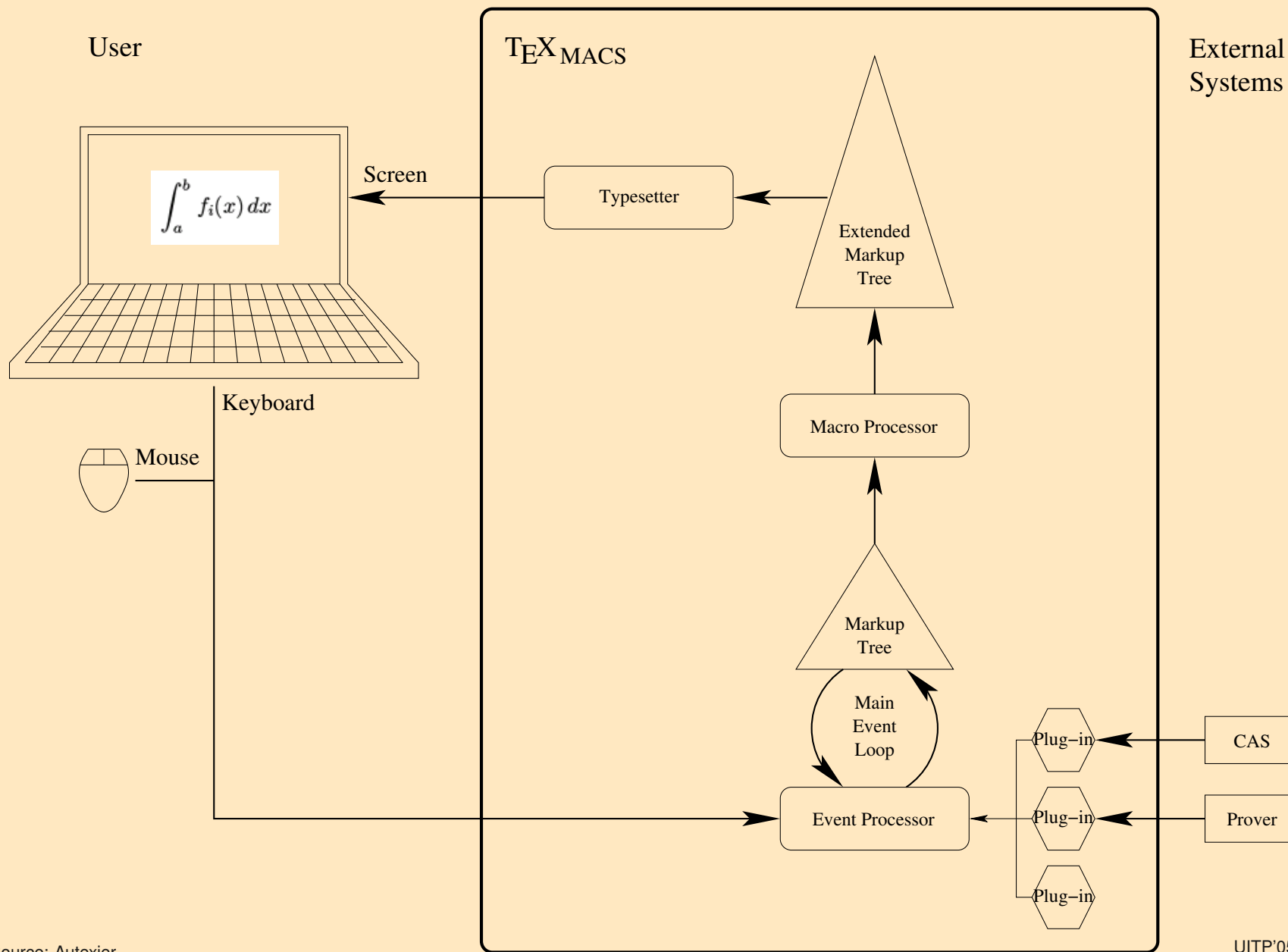
$$2 \sum_{i=1}^n a_i \int_a^b f_i(x) g_i(x) dx$$

is encoded in TEX_{MACS} markup as

```
<\equation*>
2<big|sum><rsup|n><rsub|i=1>a<rsub|i><big|int><rsup|b><rsub|a>
f<rsub|i>(x)g<rsub|i>(x) d<no-break>x
</equation*>
```

- Internal TEX_{MACS}-representation of a document is a *tree*.
- The tree can be displayed in different ways, such as in the *markup language*, in *LaTeX syntax* or as a *Lisp/Scheme s-expression*.

TEX_{MACS} Architecture



TeXmacs Components



- The *macro processor* expands the user-defined macro markup tags into primitive, macro-free T_EX_{MACS} markup.

`(underlined-italics x) ⇒ (with font-shape italic (underline (arg x)))`

is first rewritten by the macro processor as

`(with font-shape italic (underline "This is italics underlined text."))`

and then displayed in T_EX_{MACS} as: *This is italics underlined text.*

- The *event processor* detects user input events and updates the current markup tree accordingly.
- One can define new *user inputs* (e.g., keyboard shortcuts, mouse events), and write a T_EX_{MACS} *plug-in* to define the corresponding new *actions*
- Such plug-ins are extensions to the event processor, and can be written using the *Scheme* scripting language

(allows external systems to modify the current markup tree)



Formal Representation of Math. Document

Mathematical Objects/Fragments



We currently have the following mathematical fragments

- Symbols, Declarations and Terms
- Definitions and Theorems
- Proofs and Proof steps
- Keys and Links
- Contexts and Environments

Symbols, Declarations and Terms



- Symbol: A macro-name corresponding to some formal symbol

Example: `set-equal`, `set-union`, `set-intersection`, `set-complement`

```
(set-equal a b)           ⇒ (with mode math (arg a) '=' (arg b))
(set-union a b)          ⇒ (with mode math (arg a) '<cup>' (arg b))
(set-intersection a b)  ⇒ (with mode math (arg a) '<cap>' (arg b))
(set-complement a)     ⇒ (with mode math (overline (arg a)))
```

- Declarations and Terms

```
NAME           ::= SYMBOL | _
DECLARATION    ::= (type-constant NAME) | (type-variable NAME) | NAME
TERM           ::= SYMBOL | (SYMBOL TERM+) | (BINDER NAME TYPE TERM)
BINDER         ::= forall | exists | lambda
```

Example:

```
(set-equal (set-complement (set-union A B))
            (set-intersection (set-complement A) (set-complement B)))
```

is displayed as $\overline{A \cup B} = \overline{A} \cap \overline{B}$

Definitions and Theorems



```
DEFINITION ::= (definition DECLARATION* CONCEPT TERM)
CONCEPT  ::= (type-constant NAME) | (constant NAME TYPE)
THEOREM    ::= (theorem      NAME DECLARATION* ASSUMPTION* CONCLUSION)
CONCLUSION ::= (conclusion NAME TERM)
ASSUMPTION ::= (assumption NAME TERM)
```

Then the deMorgan theorem $\overline{A \cup B} = \bar{A} \cap \bar{B}$ can be encoded by

```
(theorem deMorgan
 (conclusion -
  (forall A set
   (forall B set
    (set-equal
     (set-complement (set-union A B))
     (set-intersection (set-complement A) (set-complement B)))))))
```

and is displayed in the $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ editor as

Theorem deMorgan:

Conclusion:

$\forall A : \text{set. } \forall B : \text{set. } \overline{A \cup B} = \bar{A} \cap \bar{B}$

Proofs and Proof steps



```
PROOF ::= (proof @THEOREM PROOF-STEP)
PROOF-STEP ::= (proof-step CONTENT RULE-DESCRIPTION
                (PROOF-STEP*) MENU)
RULE-DESCRIPTION ::= open | closed | (apply TERM TERM)
                  | (focus TERM+) | (unfocus) | ...
CONTENT ::= (GOAL AVAILABLE-ASSUMPTIONS
            ALTERNATIVE-GOALS)
ALTERNATIVE-GOALS ::= (TERM*)
AVAILABLE-ASSUMPTIONS ::= (TERM*)
GOAL ::= TERM
```

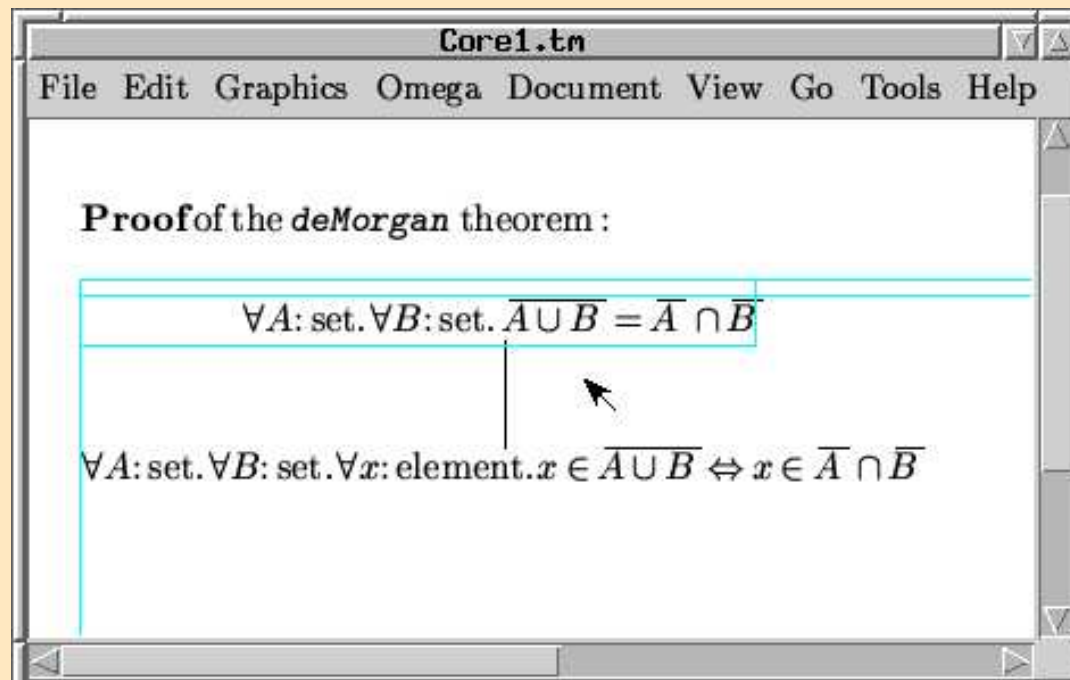
where @THEOREM is a link to a fragment of the type THEOREM.

Example: Initial proof steps of deMorgan theorem; application of

$$\forall A : \text{set}. \forall B : \text{set}. A = B \Leftrightarrow \forall x : \text{element}. x \in A \Leftrightarrow x \in B$$

Example Proof

```
(proof @THEO1
  (proof-step (  $\forall A : \text{set} . \forall B : \text{set} . \overline{A \cup B} = \overline{A} \cap \overline{B}$  ) (...))
  (apply (  $A = B \Rightarrow \forall x : \text{element} . x \in A \Leftrightarrow x \in B$  ))
  ((proof-step (  $\forall A : \text{set} . \forall B : \text{set} . \forall x : \text{element} .$ 
     $x \in \overline{A \cup B} \Leftrightarrow x \in \overline{A} \cap \overline{B}$  ) (...))
  open () (menu ...)))
(menu ...))
```



Keys and Links

- Introduce a system of symbolic *keys* and *links*
 - ▶ to denote parts of the document
 - ▶ to allow actions in menus to refer to parts of a T_EX_{MACS} document
 - ▶ to design localised patch applications

- Based on a general mean to attach *attributes* to the fragments

The key of a fragment is the value of its attribute (if so) of name *key*

```
KEYED-ELEMENT ::= (keyed (ATTR*) ELEMENT)
ATTR          ::= (SYMBOL ELEMENT)
```

- A *link* refers (1) to a different fragment with the respective key or (2) to one of its sub-elements (indicated by a path):

```
LINK ::= (link KEY (NAT*))
```

- *Example:* Formula $1 + 2 \times 3$, key FORM1 and further attributes

```
(keyed ((key FORM1) (depth 2) (leaves 3))
  (expr + 1 (expr * 2 3)))
```

Display $1 + 2 \times 3$

$(\text{keyed } a \ b) \implies (\text{arg } b)$

Contexts and Environments



- Mathematical fragments may use symbols not defined in the current document.
- They “live” in packages that we call *theories*
(defined in other $\text{T}_{\text{E}}\text{X}_{\text{M}}\text{A}^{\text{C}}\text{S}$ documents or in a knowledge base)
- Need to indicate the theory for a fragment *(Context)*

```
CONTEXT ::= (context THEORY BODY)
THEORY  ::= (theory-by-name NAME) | (theory-union THEORY+)
```

Assuming a theory `set-theory` in a knowledge base:

```
(context (theory-by-name set-theory)
  (keyed ((key THEO1)
    (theorem deMorgan
      (conclusion _  $\forall A : \text{set} . \forall B : \text{set} . \overline{A \cup B} = \overline{A} \cap \overline{B}$  )))))
```

Context & Environments II



- Allows to build nested context structures
- Inside the `BODY` one has access to the joint name spaces and assertions of all theories indicated by surrounding `CONTEXTS` (*Environment*)
- Given a theory stored in some mathematical repository, we need the set of `TEXMACS` macros corresponding to the symbols defined in it
(to allow display of symbols and terms in their common mathematical form)
- We implemented such a mechanism between `TEXMACS` and the mathematical database `MBASE`(via Ω MEGA)

Synchronisation

Menus

- Basic menu actions are
 - ▶ either to apply a function
(provided by the PA)
 - ▶ or to chose a specific number of arguments
- **Submenus** are realised via nested CHOICE and APPLY
- Labels of submenus are those from CHOICE and APPLY
- Actual **computation from values** are FUNCTION-NAME and SELECTED

```

MENU ::= (menu EVALUATE ::= BOOLEAN
          ACTION)
ACTION ::= APPLY | CHOICE
APPLY ::= (apply LABEL ::= STRING
          FOLDED ::= BOOLEAN
          FUNCTION-NAME ::= SYMBOL
          PARAMETERS ::= ( EXPR*))
CHOICE ::= (choice LABEL ::= STRING
            FOLDED ::= BOOLEAN
            MIN ::= NAT
            MAX ::= NAT
            SET ::= ( EXPR*)
            SELECTED ::= (LINK*))
EXPR ::= ACTION | S-EXPRESSION

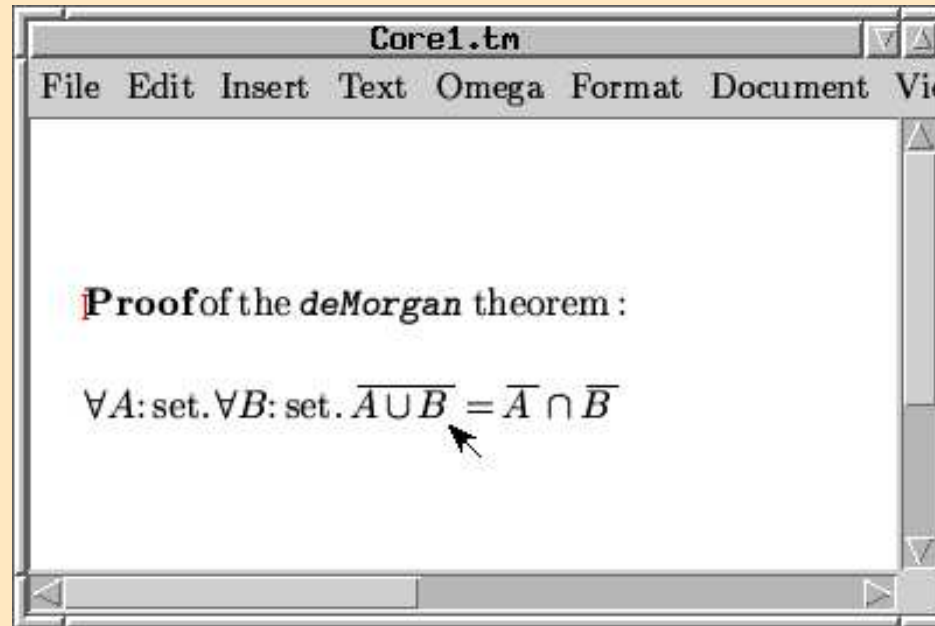
```

Example



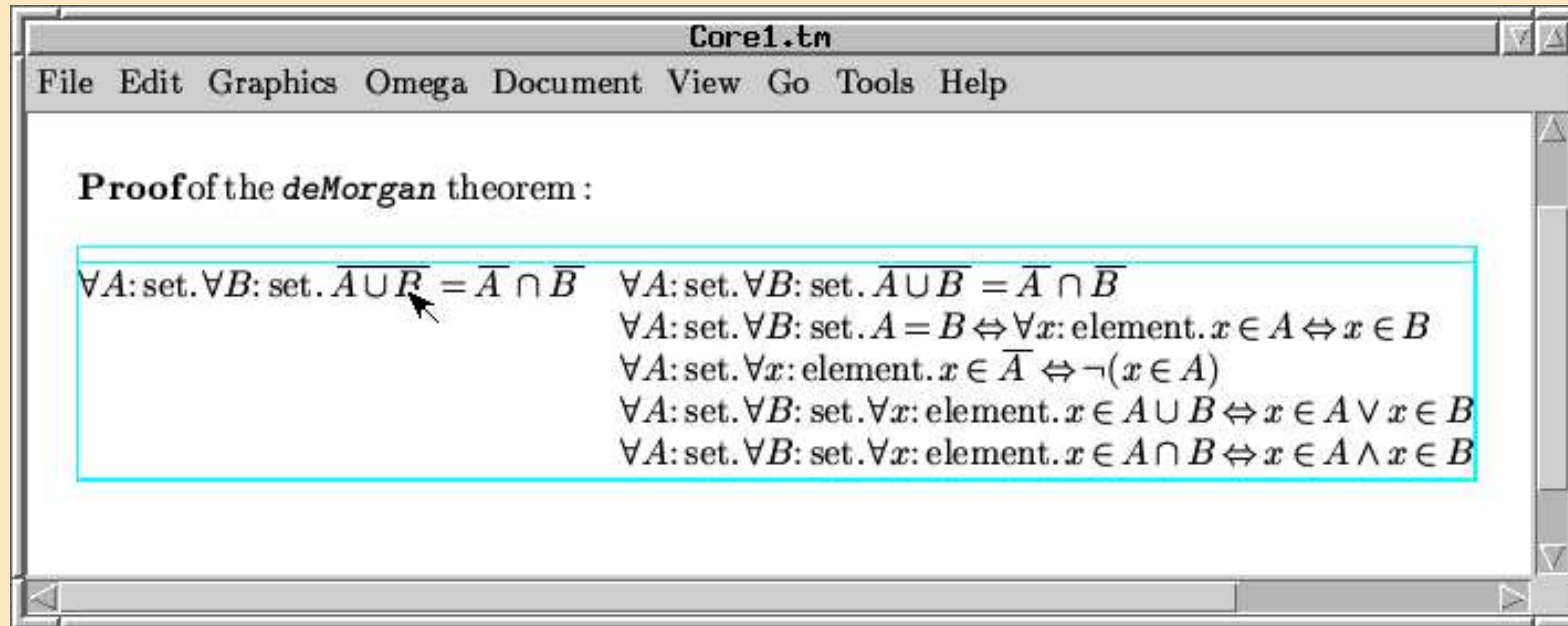
```
(keyed ((key THEO1))
 (theorem deMorgan (conclusion _ (keyed ((key GOAL1))  $\forall A : \text{set. } \forall B : \text{set. } \overline{A \cup B} = \overline{A} \cap \overline{B}$ 
 [...])
 (with prog-language "core" prog-session "default"
 (keyed ((key PROOF1))
 (proof @THEO1
 (proof-step
 (@GOAL1 (keyed ((key HYP1))  $\forall A : \text{set. } \forall B : \text{set. } A = B \Leftrightarrow \forall x \in E, x \in A \Leftrightarrow x \in B$  )
 (keyed ((key HYP2))  $\forall A : \text{set. } \forall x : \text{element. } x \in \overline{A} \Leftrightarrow \neg(x \in A)$  )
 (keyed ((key HYP3))  $\forall A : \text{set. } \forall B : \text{set. } \forall x : \text{element. } x \in A \cup B \Leftrightarrow x \in A \vee x \in B$  )
 (keyed ((key HYP4))  $\forall A : \text{set. } \forall B : \text{set. } \forall x : \text{element. } x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$  )
 open ()
 (keyed ((key MENU1))
 (menu false
 (keyed ((key APPLY1))
 (apply _ true apply-rule
 ((keyed ((key CHOICE1))
 (choice _ false 2 2 (@GOAL1 @HYP1 @HYP2 @HYP3 @HYP4) ())))))))))
```

Display



Menu Not Activated

Display



Menu Activated

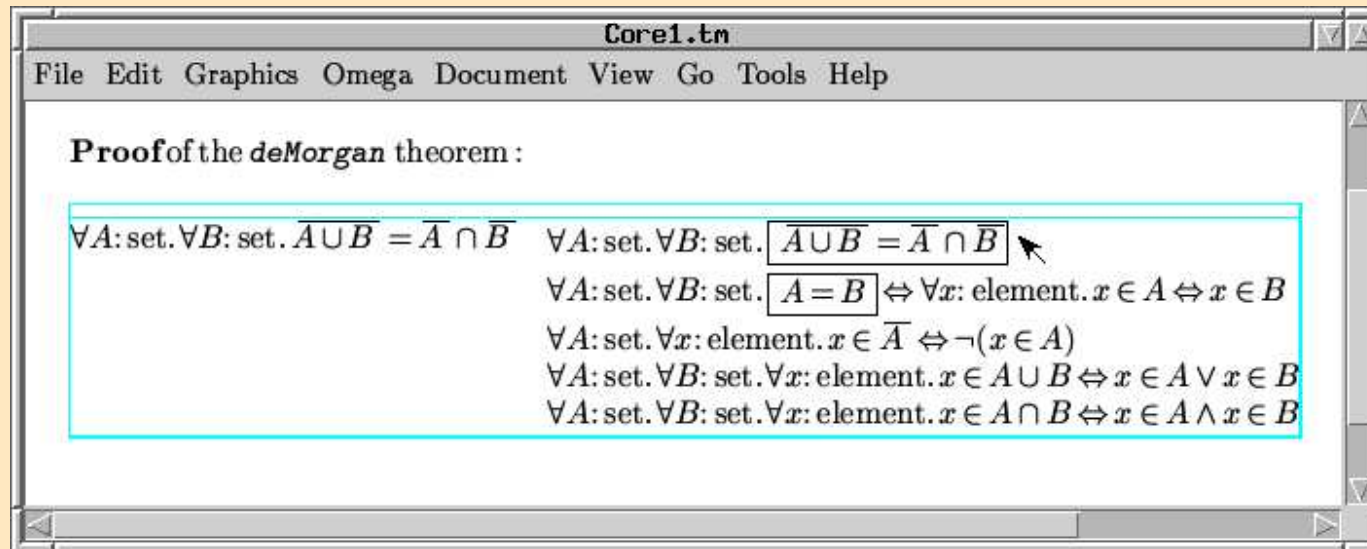
Evaluation



- An `APPLY` node is reducible iff all the `EXPR` objects contained in its `PARAMETERS` slot are reducible.
- A `CHOICE` node is reducible iff n objects are selected from `SET` and $n \in [\text{MIN} \dots \text{MAX}]$
- Any other kind of s-expression is always reducible.

```
MENU ::= (menu EVALUATE ::= BOOLEAN
          ACTION)
ACTION ::= APPLY | CHOICE
APPLY ::= (apply LABEL ::= STRING
          FOLDED ::= BOOLEAN
          FUNCTION-NAME ::= SYMBOL
          PARAMETERS ::= ( EXPR*))
CHOICE ::= (choice LABEL ::= STRING
            FOLDED ::= BOOLEAN
            MIN ::= NAT
            MAX ::= NAT
            SET ::= ( EXPR*)
            SELECTED ::= (LINK*))
EXPR ::= ACTION | S-EXPRESSION
```

Example



```
(menu true
  (keyed ((key APPLY1))
    (apply - false
      apply-rule
        ((keyed ((key CHOICE1))
          (choice - false 2 2 (@GOAL1 @HYP1 @HYP2 @HYP3 @HYP4)
            ( @GOAL1(2 2) @HYP1(2 2 1))))
```

Evaluation

- A reducible APPLY node evaluates to the result of the call (eval-apply @APPLY FUNCTION-NAME PARAMETERS)
 - ▶ Result can include *patches* Kept until whole menu is reducible
- User selection or deselection of an item from SET immediately updates the SELECTED-slot
A reducible CHOICE node evaluates to its SELECTED slot.
- A reducible MENU node evaluates to the evaluation of its ACTION slot. If the evaluation is successful, the stored patches are applied.

```

MENU ::= (menu EVALUATE ::= BOOLEAN
          ACTION)
ACTION ::= APPLY | CHOICE
APPLY ::= (apply [...]
           FUNCTION-NAME ::= SYMBOL
           PARAMETERS ::= (EXPR*))
CHOICE ::= (choice [...]
            MIN ::= NAT
            MAX ::= NAT
            SET ::= (EXPR*)
            SELECTED ::= (LINK*))
EXPR ::= ACTION | S-EXPRESSION

```

```

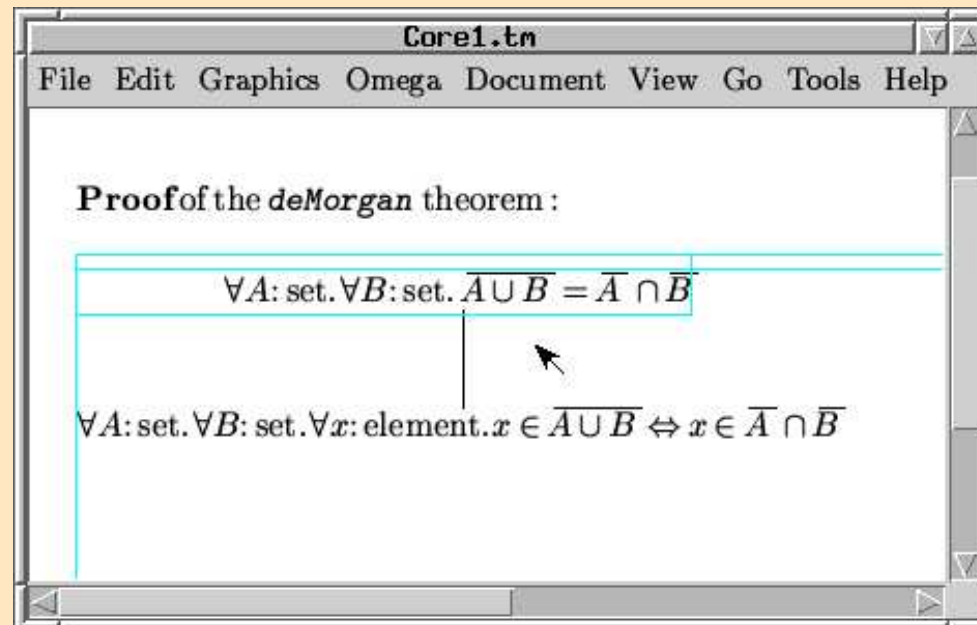
REMOTE-CALL ::=
  (eval-apply @APPLY FUNCTION-NAME PARAMETERS)
REMOTE-RESULT ::= (result RESULT (PATCH*))
PATCH ::= (patch LINK KEYED-ELEMENT)
RESULT ::= BOOLEAN | (error ...)
          | S-EXPRESSION

```

Example



```
(result true ((patch @PROOF1(2 2)
  (apply (A = B ⇒ ∀x: element.x ∈ A ⇔ x ∈ B))
  (patch @PROOF1(2 3)
    ((proof-step
      (keyed ((key GOAL2))
        (∀A: set.∀B: set.∀x: element.x ∈  $\overline{A \cup B} \Leftrightarrow x \in \overline{A} \cap \overline{B}$ 
        (...)) (...))
      open () (keyed ((key MENU2)) (menu ...))))))
```



Related Work



- TmCoq: aims at the full integration of $\text{T}_{\text{EX}}^{\text{MACS}}$ with the theorem prover Coq.
 - ▶ So far is only a shallow integration via $\text{T}_{\text{EX}}^{\text{MACS}}$ -shell plugins
 - ▶ Observe that we do want a format independent from a specific PA
- Proof General and Pcoq:
 - ▶ both made significant contributions such as proof-by-pointing, proof script management, structural editing of formulae and commands in combination with mouse-based navigation, etc.
 - ▶ However: Aim at being UIs for a PA supporting the interaction means of the PA
 - ▶ While our aim with $\text{T}_{\text{EX}}^{\text{MACS}}$ is to get independent from the PA.

Conclusion



- Ongoing integration of $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ with ΩMEGA (state based)
- We try to not impose the formal language of the PA to the author
- Defined a semantic language for
 - ▶ mathematics documents and
 - ▶ menus for interactions

which conservatively extends the $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ document format

- Synchronisation mechanism based on key-based diff and patch mechanism tailored to the $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ document format
 - ▶ Uniform evaluation semantics
 - ▶ Allows for context-sensitive menus
- We illustrated proofs using CORE-style, but a more abstract, textbook-style language for proofs is under the way