Hypertext Navigation of ACL2 Proofs with XMLEye

Antonio García Domínguez   Francisco Palomo Lozano
Inmaculada Medina Bulo

Departamento de Lenguajes y Sistemas Informáticos

May 2009

UCA
Universidad de Cádiz

Summary

XML

Summary

Output in ACL2

1. ACL2 produces linear text output
   - Usually long
   - Slightly formatted
2. Although there is an underlying structure
   - Scripts are, mainly, sequences of events and commands
   - Some events produce proofs as their outputs
   - Some commands may expand to several events
   - Both can produce a variety of outputs or even nothing
3. But representing structure in a linear way poses problems
   - Difficult to read
   - Not easy to process

Output in ACL2

1. ACL2 produces linear text output
   - Usually long
   - Slightly formatted
2. Although there is an underlying structure
   - Scripts are, mainly, sequences of events and commands
   - Some events produce proofs as their outputs
   - Some commands may expand to several events
   - Both can produce a variety of outputs or even nothing
3. But representing structure in a linear way poses problems
   - Difficult to read
   - Not easy to process

Output in ACL2

1. ACL2 produces linear text output
   - Usually long
   - Slightly formatted
2. Although there is an underlying structure
   - Scripts are, mainly, sequences of events and commands
   - Some events produce proofs as their outputs
   - Some commands may expand to several events
   - Both can produce a variety of outputs or even nothing
3. But representing structure in a linear way poses problems
   - Difficult to read
   - Not easy to process

Output in ACL2

1. ACL2 produces linear text output
   - Usually long
   - Slightly formatted
2. Although there is an underlying structure
   - Scripts are, mainly, sequences of events and commands
   - Some events produce proofs as their outputs
   - Some commands may expand to several events
   - Both can produce a variety of outputs or even nothing
3. But representing structure in a linear way poses problems
   - Difficult to read
   - Not easy to process

We lack a formal description of ACL2 output

What is the point in reading proofs?

1. Past and recent efforts in ACL2 to reduce output
   - Inhibition
   - Evisceration
   - Gag-mode
2. Nothing bad about reducing output, if ...
   - You are trying to abbreviate huge terms
   - You are interested in completing proofs as quick as possible
   - You are not interested in proof details
   - You are not trying to learn ACL2
3. However, there are contexts where reading proofs is useful
   - Script improvement
   - Proof communication
   - Proof presentation

What is the point in reading proofs?

1. Past and recent efforts in ACL2 to reduce output
   - Inhibition
   - Evisceration
   - Gag-mode
2. Nothing bad about reducing output, if ...
   - You are trying to abbreviate huge terms
   - You are interested in completing proofs as quick as possible
   - You are not interested in proof details
   - You are not trying to learn ACL2
3. However, there are contexts where reading proofs is useful
   - Script improvement
   - Proof communication
   - Proof presentation

What is the point in reading proofs?

1. Past and recent efforts in ACL2 to reduce output
   - Inhibition
   - Evisceration
   - Gag-mode
2. Nothing bad about reducing output, if ...
   - You are trying to abbreviate huge terms
   - You are interested in completing proofs as quick as possible
   - You are not interested in proof details
   - You are not trying to learn ACL2
3. However, there are contexts where reading proofs is useful
   - Script improvement
   - Proof communication
   - Proof presentation

Why proof presentation is important

Q: How do you convince someone about an ACL2 proof?

- This is easier for other systems based on, say, LCF
- We do not have yet proof objects for ACL2
- ACL2 is not yet formally verified
- We can envision large portions of ACL2 being verified by itself

A: You convince her with ACL2's output

- ACL2 proofs are more readable because of natural language
- Some systems can translate proof objects to natural language
- This translation often produces too low-level or poor results

XML

Why proof presentation is important

Q: How do you convince someone about an ACL2 proof?

- This is easier for other systems based on, say, LCF
- We do not have yet proof objects for ACL2
- ACL2 is not yet formally verified
- We can envision large portions of ACL2 being verified by itself

A: You convince her with ACL2's output

- ACL2 proofs are more readable because of natural language
- Some systems can translate proof objects to natural language
- This translation often produces too low-level or poor results

Summary

What is not XMLEye?

1. An ACL2 flavor
2. An IDE for ACL2
3. A tool for completing proofs quicker
4. A Holy Grail dealing with all of ACL2 output complexities

What is XMLEye?

Definition

XMLEye is a generic framework for creating viewers for complex structured document formats

1. Free, GNU GPL, software
2. Platform-independent (Java)
3. Truly generic
4. Data-driven
5. Independent of ACL2

What is XMLEye?

Definition

XMLEye is a generic framework for creating viewers for complex structured document formats

1. Free, GNU GPL, software
2. Platform-independent (Java)
3. Truly generic
4. Data-driven
5. Independent of ACL2

Why using XMLEye with ACL2?

1. ACL2 input and output can be transformed into XML
   - We have designed a specific XML vocabulary for ACL2 output
   - This covers a major subset of possible output
   - This output can be validated against its specification
2. You can add XSLT stylesheets for different purposes
   - Mining information thru XML to XML transformations
   - Presentation markup thru XML to XHTML transformations
   - Other transformations are conceivable
3. You can reorganize the output and infer new information
4. You can present and navigate the result

Why using XMLEye with ACL2?

1. ACL2 input and output can be transformed into XML
   - We have designed a specific XML vocabulary for ACL2 output
   - This covers a major subset of possible output
   - This output can be validated against its specification
2. You can add XSLT stylesheets for different purposes
   - Mining information thru XML to XML transformations
   - Presentation markup thru XML to XHTML transformations
   - Other transformations are conceivable
3. You can reorganize the output and infer new information
4. You can present and navigate the result

Why using XMLEye with ACL2?

1. ACL2 input and output can be transformed into XML
    - We have designed a specific XML vocabulary for ACL2 output
    - This covers a major subset of possible output
    - This output can be validated against its specification
2. You can add XSLT stylesheets for different purposes
    - Mining information thru XML to XML transformations
    - Presentation markup thru XML to XHTML transformations
    - Other transformations are conceivable
3. You can reorganize the output and infer new information
4. You can present and navigate the result

Why using XMLEye with ACL2?

1. ACL2 input and output can be transformed into XML
   - We have designed a specific XML vocabulary for ACL2 output
   - This covers a major subset of possible output
   - This output can be validated against its specification
2. You can add XSLT stylesheets for different purposes
   - Mining information thru XML to XML transformations
   - Presentation markup thru XML to XHTML transformations
   - Other transformations are conceivable
3. You can reorganize the output and infer new information
4. You can present and navigate the result

Interfacing XMLEye with an external processor

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/accepted-doc">
  <name>ACL2 Proof Script</name>
  <name language="es">Guión de ACL2</name>
  <edit>emacs %s</edit>
  <import>perl pprocACL2.perl %s</import>
  <extensions>
    <extension>lsp</extension>
    <extension>lisp</extension>
    <extension>acl2</extension>
  </extensions>
</format>
```

Interfacing XMLEye with an external processor

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/accepted-doc">
  <name>ACL2 Proof Script</name>
  <name language="es">Guión de ACL2</name>
  <edit>emacs %s</edit>
  <import>perl pprocACL2.perl %s</import>
  <extensions>
    <extension>lsp</extension>
    <extension>lisp</extension>
    <extension>acl2</extension>
  </extensions>
</format>
```

Interfacing XMLEye with an external processor

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/accepted-doc">
  <name>ACL2 Proof Script</name>
  <name language="es">Guión de ACL2</name>
  <edit>emacs %s</edit>
  <import>perl pprocACL2.perl %s</import>
  <extensions>
    <extension>lsp</extension>
    <extension>lisp</extension>
    <extension>acl2</extension>
  </extensions>
</format>
```

Interfacing XMLEye with an external processor

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/accepted-doc">
  <name>ACL2 Proof Script</name>
  <name language="es">Guión de ACL2</name>
  <edit>emacs %s</edit>
  <import>perl pprocACL2.perl %s</import>
  <extensions>
    <extension>lsp</extension>
    <extension>lisp</extension>
    <extension>acl2</extension>
  </extensions>
</format>
```

Interfacing XMLEye with an external processor

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/accepted-doc">
  <name>ACL2 Proof Script</name>
  <name language="es">Guión de ACL2</name>
  <edit>emacs %s</edit>
  <import>perl pprocACL2.perl %s</import>
  <extensions>
    <extension>lsp</extension>
    <extension>lisp</extension>
    <extension>acl2</extension>
  </extensions>
</format>
```

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
    - Dependencies are computed on the fly
    - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
    - Where a theorem is used
    - Unused events

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

XML

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
   - Dependencies are computed on the fly
   - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
   - Where a theorem is used
   - Unused events

XML

Features of XMLEye with ACL2 postprocessor

1. Accepts an important subset of ACL2 output
2. Establishes links between its elements
3. Presents the output in hypertext form
4. Highlights its different parts
5. Provides different levels of detail
6. Processes book dependencies like *make*
    - Dependencies are computed on the fly
    - *Makefiles* are not required
7. Allows invoking the editor and watching for changes
8. Infers new script-local information for the end user
    - Where a theorem is used
    - Unused events

A fragment of the XML ACL2 specification

```
<!ELEMENT certifybook  (warning*, observation*, error*, checksum,
                        verification, includechk, write_cert,
                        compilation, summary)>
<!ELEMENT comment      (#PCDATA)>
<!ELEMENT defpkg       (warning*, observation*, error*, summary)>
<!ELEMENT defthm       (warning*, observation*, error*,
                        (rule_classes, subgoal, deps_storage?)?,
                        summary)>
<!ELEMENT deps_storage (output, rules)>
<!ELEMENT output       (sexpr|paragraph)*>
<!ELEMENT paragraph    (text|sexpr)*>
<!ELEMENT sexpr        (#PCDATA)>
<!ELEMENT simp         (rules, output, subgoal*)>
<!ELEMENT subgoal      (simp|abandon|elim|hypothesis|induction|
                        generalize|discard|preinduction|
                        nottheorem|crossfert|failure)>
<!ELEMENT unknown ANY>
```

A fragment of the XML ACL2 specification

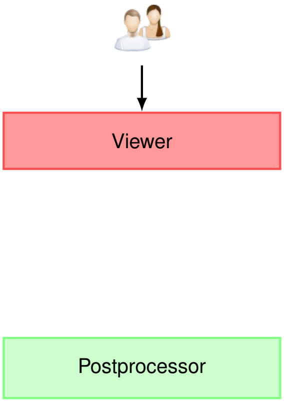
```
<!ELEMENT certifybook  (warning*, observation*, error*, checksum,
                        verification, includechk, write_cert,
                        compilation, summary)>
<!ELEMENT comment      (#PCDATA)>
<!ELEMENT defpkg       (warning*, observation*, error*, summary)>
<!ELEMENT defthm       (warning*, observation*, error*,
                        (rule_classes, subgoal, deps_storage?)?,
                        summary)>
<!ELEMENT deps_storage (output, rules)>
<!ELEMENT output       (sexpr|paragraph)*>
<!ELEMENT paragraph    (text|sexpr)*>
<!ELEMENT sexpr        (#PCDATA)>
<!ELEMENT simp         (rules, output, subgoal*)>
<!ELEMENT subgoal      (simp|abandon|elim|hypothesis|induction|
                        generalize|discard|preinduction|
                        nottheorem|crossfert|failure)>
<!ELEMENT unknown ANY>
```

A fragment of the XML ACL2 specification

```
<!ELEMENT certifybook  (warning*, observation*, error*, checksum,
                        verification, includechk, write_cert,
                        compilation, summary)>
<!ELEMENT comment      (#PCDATA)>
<!ELEMENT defpkg       (warning*, observation*, error*, summary)>
<!ELEMENT defthm       (warning*, observation*, error*,
                        (rule_classes, subgoal, deps_storage?)?,
                        summary)>
<!ELEMENT deps_storage (output, rules)>
<!ELEMENT output       (sexpr|paragraph)*>
<!ELEMENT paragraph    (text|sexpr)*>
<!ELEMENT sexpr        (#PCDATA)>
<!ELEMENT simp         (rules, output, subgoal*)>
<!ELEMENT subgoal      (simp|abandon|elim|hypothesis|induction|
                        generalize|discard|preinduction|
                        nottheorem|crossfert|failure)>
<!ELEMENT unknown ANY>
```

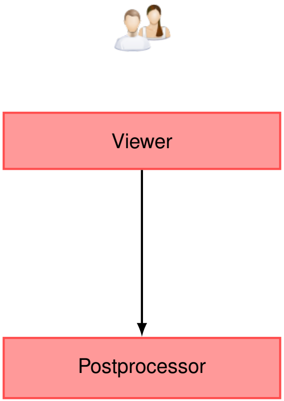
Summary

Workflow



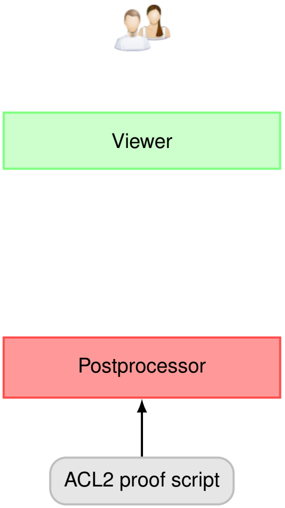
1. The user opens a file in the viewer

Workflow



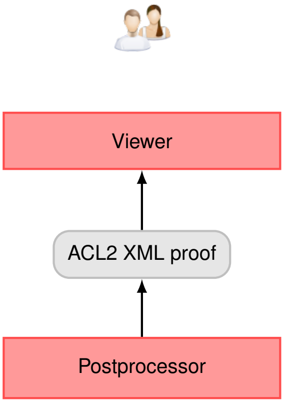
2. The viewer tries to import it

Workflow



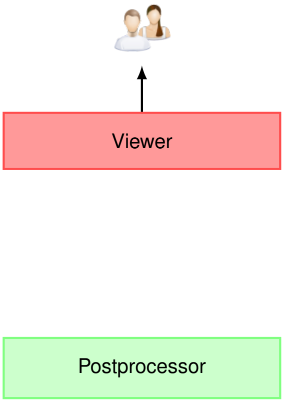
3. The postprocessor produces the XML

Workflow



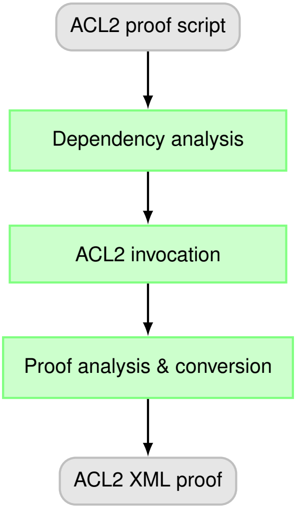
4. The viewer transforms the XML

Workflow

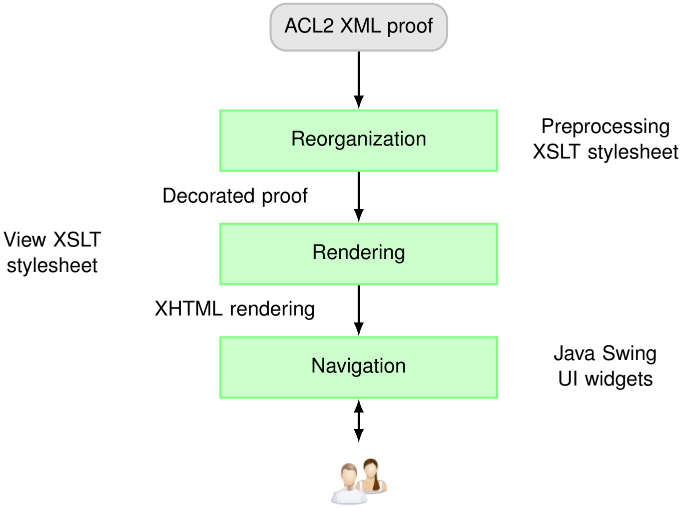


5. The viewer presents the result

Postprocessor architecture



Viewer architecture



Summary

Mathematical Knowledge Management

HELM *http://helm.cs.unibo.it*

- Hypertextual Electronic Library of Mathematics
- Coq online
- NuPRL online

Summary

Contributions

1. Tools for postprocessing and navigating ACL2 proofs
2. Valuable for learning and communicating
3. General and extensible approach
4. Users can extend the system with their own stylesheets
5. No recompilation needed

Contributions

1. Tools for postprocessing and navigating ACL2 proofs
2. Valuable for learning and communicating
3. General and extensible approach
4. Users can extend the system with their own stylesheets
5. No recompilation needed

Contributions

1. Tools for postprocessing and navigating ACL2 proofs
2. Valuable for learning and communicating
3. General and extensible approach
4. Users can extend the system with their own stylesheets
5. No recompilation needed

XML

Contributions

1. Tools for postprocessing and navigating ACL2 proofs
2. Valuable for learning and communicating
3. General and extensible approach
4. Users can extend the system with their own stylesheets
5. No recompilation needed

XML

Contributions

1. Tools for postprocessing and navigating ACL2 proofs
2. Valuable for learning and communicating
3. General and extensible approach
4. Users can extend the system with their own stylesheets
5. No recompilation needed

Future work

1. Statistics, proof metrics, effort metrics
2. Full-project event dependency analysis
3. Analysis thru connected components
4. Useful for proof script improvement
   - Reorganization
   - Detecting events that should be local
   - Detecting events that could be eliminated

A suggestion

Regarding function dependencies, this is not straightforward because the definition summaries do not include this info. New, `(:TERMINATION ...)` rules, could be added in function definition summaries to mark the functions they use.

Future work

1. Statistics, proof metrics, effort metrics
2. Full-project event dependency analysis
3. Analysis thru connected components
4. Useful for proof script improvement
   - Reorganization
   - Detecting events that should be local
   - Detecting events that could be eliminated

A suggestion

Regarding function dependencies, this is not straightforward because the definition summaries do not include this info. New, `(:TERMINATION ...)` rules, could be added in function definition summaries to mark the functions they use.

Future work

1. Statistics, proof metrics, effort metrics
2. Full-project event dependency analysis
3. Analysis thru connected components
4. Useful for proof script improvement
   - Reorganization
   - Detecting events that should be local
   - Detecting events that could be eliminated

A suggestion

Regarding function dependencies, this is not straightforward because the definition summaries do not include this info. New, `(:TERMINATION ...)` rules, could be added in function definition summaries to mark the functions they use.

XML

Future work

1. Statistics, proof metrics, effort metrics
2. Full-project event dependency analysis
3. Analysis thru connected components
4. Useful for proof script improvement
   - Reorganization
   - Detecting events that should be local
   - Detecting events that could be eliminated

A suggestion

Regarding function dependencies, this is not straightforward because the definition summaries do not include this info. New, `(:TERMINATION ...)` rules, could be added in function definition summaries to mark the functions they use.

Future work

1. Statistics, proof metrics, effort metrics
2. Full-project event dependency analysis
3. Analysis thru connected components
4. Useful for proof script improvement
   - Reorganization
   - Detecting events that should be local
   - Detecting events that could be eliminated

A suggestion

Regarding function dependencies, this is not straightforward because the definition summaries do not include this info. New, `(:TERMINATION ...)` rules, could be added in function definition summaries to mark the functions they use.

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

XML

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

Final comments

1. It would be nice if we had a formal description of ACL2 output
2. This formal description could be used in a variety of ways
3. This does not imply a complete markup of ACL2 output
4. Even a light markup would be profitable
5. This could foster new tools for ACL2
6. We are still far from knowledge management

A final reflexion

XML is the standard for content markup and it is here to stay

References

A. Asperti, L. Padovani, C. Sacerdoti Coen, F. Guidi, and I. Schena.
Mathematical Knowledge Management in HELM.
Ann. Math. Artif. Intell., 38(1–3):27–46, 2003.

A. García Domínguez.
XMLEye Wiki.
http://wiki.shoyusauce.org, Oct. 2008.

A. García Domínguez.
XMLEye Forge at RedIris.
https://forja.rediris.es/projects/csl2-xmleye, Mar. 2009.

Thanks for your attention

Questions?

Never answer the question that is asked of you
Answer the question that you wish had been asked of you

— Robert McNamara (2003)