Tool Interoperability in the Maude Formal Environment

Francisco Durán[1] Camilo Rocha[2] José M. Álvarez[1]

[1]Universidad de Málaga
[2]University of Illinois at Urbana-Champaign

4th Conference on Algebra and Coalgebra in Computer Science
August 31, 2011
Winchester, UK

Main Contribution

The Maude Formal Environment (MFE) is an executable formal specification in Maude within which a user can interact with tools to mechanically verify properties of Maude specifications

Main Contribution

The Maude Formal Environment (MFE) is an executable formal specification in Maude within which a user can interact with tools to mechanically verify properties of Maude specifications

- it has been designed to be easily extended with tools having heterogeneous designs
  - it currently offers five tools

Main Contribution

The Maude Formal Environment (MFE) is an executable formal specification in Maude within which a user can interact with tools to mechanically verify properties of Maude specifications

- it has been designed to be easily extended with tools having heterogeneous designs
  - it currently offers five tools
- it implements a mechanism to keep track of pending proof obligations

Main Contribution

The Maude Formal Environment (MFE) is an executable formal specification in Maude within which a user can interact with tools to mechanically verify properties of Maude specifications

- it has been designed to be easily extended with tools having heterogeneous designs
  - it currently offers five tools
- it implements a mechanism to keep track of pending proof obligations
- its tool interoperability allows for discharging proof obligations of different nature without switching between different tool environments and presents the user with a consistent user interface

Main Contribution

The Maude Formal Environment (MFE) is an executable formal specification in Maude within which a user can interact with tools to mechanically verify properties of Maude specifications

- it has been designed to be easily extended with tools having heterogeneous designs
  - it currently offers five tools
- it implements a mechanism to keep track of pending proof obligations
- its tool interoperability allows for discharging proof obligations of different nature without switching between different tool environments and presents the user with a consistent user interface
- it allows the execution of several instances of each tool

Motivation

The Example of Readers and Writers

We want to check in the `R+W` system that it is never the case that more than (i) one writer or (ii) writers and readers share a critical resource at the same time. A state is represented by a term

$$\langle r, w \rangle$$

where r and w are the number of readers and writers accessing the critical resource.

Motivation

The Example of Readers and Writers

We want to check in the `R+W` system that it is never the case that more than (i) one writer or (ii) writers and readers share a critical resource at the same time. A state is represented by a term

$$\langle r, w \rangle$$

where r and w are the number of readers and writers accessing the critical resource.

- `R+W` needs to be executable, i.e., its equations ground Church-Rosser and terminating, and its rewrite rules ground coherent with respect the equations

Motivation

The Example of Readers and Writers

We want to check in the `R+W` system that it is never the case that more than (i) one writer or (ii) writers and readers share a critical resource at the same time. A state is represented by a term

$$\langle r, w \rangle$$

where r and w are the number of readers and writers accessing the critical resource.

- `R+W` needs to be executable, i.e., its equations ground Church-Rosser and terminating, and its rewrite rules ground coherent with respect the equations
- for initial state $\langle 0, 0 \rangle$, the set of initial states is infinite, so we apply a state abstraction in `R+W-ABS` which needs to be checked executable

Outline

1. Tools in the Environment
2. Design and Main Features
3. Demo

Outline

Tool Overview

In the current version of MFE one can interact with the following tools:

Tool Overview

In the current version of MFE one can interact with the following tools:

MTT *Maude Termination Tool*
termination of equational and rewrite specifications

Tool Overview

In the current version of MFE one can interact with the following tools:

MTT *Maude Termination Tool*
termination of equational and rewrite specifications

SCC *Sufficient Completeness Checker*
sufficient completeness and freeness of equational specifications, and deadlock of rewrite specifications

Tool Overview

In the current version of MFE one can interact with the following tools:

MTT *Maude Termination Tool*
: termination of equational and rewrite specifications

SCC *Sufficient Completeness Checker*
: sufficient completeness and freeness of equational specifications, and deadlock of rewrite specifications

CRC *Church-Rosser Checker*
: ground confluence and sort-decreasingness of equational specifications

Tool Overview

In the current version of MFE one can interact with the following tools:

- **MTT** *Maude Termination Tool*
  termination of equational and rewrite specifications
- **SCC** *Sufficient Completeness Checker*
  sufficient completeness and freeness of equational specifications, and deadlock of rewrite specifications
- **CRC** *Church-Rosser Checker*
  ground confluence and sort-decreasingness of equational specifications
- **ChC** *Maude Coherence Checker*
  ground coherence of rewrite specifications

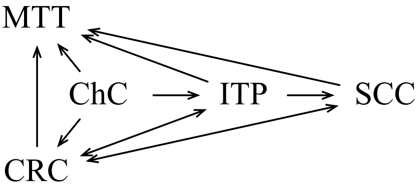
Tool Overview

In the current version of MFE one can interact with the following tools:

- MTT *Maude Termination Tool*
  termination of equational and rewrite specifications
- SCC *Sufficient Completeness Checker*
  sufficient completeness and freeness of equational specifications, and deadlock of rewrite specifications
- CRC *Church-Rosser Checker*
  ground confluence and sort-decreasingness of equational specifications
- ChC *Maude Coherence Checker*
  ground coherence of rewrite specifications
- ITP *Inductive Theorem Prover*
  inductive properties of equational specifications

Tool-dependency Graph in MFE

One important aspect in the integration task is the interaction complexity due to the nontrivial dependencies among tools



Outline

MFE Design Overview

- MFE is modeled in Maude as an interactive object-based system where tools are objects, the communication mechanism is message passing, and user interaction is available through Full Maude

MFE Design Overview

- MFE is modeled in Maude as an interactive object-based system where tools are objects, the communication mechanism is message passing, and user interaction is available through Full Maude

- integration and interoperation of tools within MFE is module-centric given that its main purpose is to support formal analysis of Maude modules

MFE Design Overview

- MFE is modeled in Maude as an interactive object-based system where tools are objects, the communication mechanism is message passing, and user interaction is available through Full Maude
- integration and interoperation of tools within MFE is module-centric given that its main purpose is to support formal analysis of Maude modules
- although some classes and functionality are provided in MFE, it imposes no constraint on how each tool should model its particular domain or maintains its internal state

Main Classes

The object-oriented model of MFE consists of three main classes

`Proof` class of proof objects that keep the state of specific proof requests

`Tool` class of tool objects that keep the life-cycle of proof objects

`Controller` inherits from Full Maude's `DatabaseClass` and provides a centralized entry point for handling user request

User Interaction

The user interacts with the environment via commands

User Interaction

The user interacts with the environment via commands

- each command is encapsulated as a message in the object configuration

User Interaction

The user interacts with the environment via commands

- each command is encapsulated as a message in the object configuration
- each tool object and the controller object have a module defining the signature of commands it can handle
  - the controller handles any command it can parse
  - if the controller receives a command it cannot parse, then it delegates the message to the *active* tool
  - if the tool can parse the delegated command, then it notifies the controller and handles the command
  - otherwise, it will notify the failure to the controller, which in turn will output an error message to the user

Commands in MFE

MFE provides the following user commands:

`(select tool <tool-name> .)` sets `<tool-name>` as the *active* tool

`(MFE help .)` shows MFE's help information

`(show global state .)` shows the state of the environment

Commands in MFE

MFE provides the following user commands:

`(select tool <tool-name> .)` sets `<tool-name>` as the *active* tool

`(MFE help .)` shows MFE's help information

`(show global state .)` shows the state of the environment

The tools available in MFE's current release provide at least the following commands:

`(<tool-name> help .)` shows the help information of tool `<tool-name>`

`(show state .)` shows the state of the tool

Proof Obligations

A tool in MFE keeps track of both its pending and discharged proof obligations

Proof Obligations

A tool in MFE keeps track of both its pending and discharged proof obligations

- a user can submit proof obligations to other tools by means of the following command and then be notified when these are discharged

```
(submit .)
```

Proof Obligations

A tool in MFE keeps track of both its pending and discharged proof obligations

- a user can submit proof obligations to other tools by means of the following command and then be notified when these are discharged

  ```
  (submit .)
  ```

- when all proof obligations in the verification task of a module's property are discharged, the corresponding tool notifies the success result to the user or to the tool originating the verification task

Trusting Proof Obligations

Tools in general can impose constraints on its inputs

Trusting Proof Obligations

Tools in general can impose constraints on its inputs

- for instance, SCC does not support parametric modules but proofs for such modules could be obtained by hand or using another tool

Trusting Proof Obligations

Tools in general can impose constraints on its inputs

- for instance, SCC does not support parametric modules but proofs for such modules could be obtained by hand or using another tool
- MFE offers the following command for keeping track of proofs obtained outside the environment

  ```
  (trust .)
  ```

External Utilities

For tools which depend on external utilities not directly available from Maude such as MTT and SCC, we have extended the latest release of the Maude system with *built-in* operators associated with appropriate C++ code that interacts with the external tools

Outline

Obtaining and Using MFE

The tool, the pimped version of Maude, and more examples are available at

http://maude.lcc.uma.es/MFE

Thank you!