# Maude Summer School: Lecture 3-I

José Meseguer

University of Illinois at Urbana-Champaign and
Leverhulme visiting professor at King's College, London

# Executability Conditions

What properties should a Maude functional module fmod $(\Sigma, E)$ endfm with $\Omega \subseteq \Sigma$ its data constructors (the [ctor] attribute) have for correct execution?

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$.

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**.

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \rightarrow v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \rightarrow_{\vec{E}} t_1 \rightarrow_{\vec{E}} t_2 \ldots t_n \rightarrow_{\vec{E}} t_{n+1} \ldots$$

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

   (ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

(ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$ (computed by Maude's `red` $t$ command).

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

   (ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$ (computed by Maude's `red` $t$ command).

2. **Sufficient Completenes**:

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

   (ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$ (computed by Maude's `red` $t$ command).

2. **Sufficient Completenes**: for any $\Sigma$-term $t$ without variables (called a ground term), $t!_E$ is an $\Omega$-term (constructor term).

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

   (ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$ (computed by Maude's `red` $t$ command).
2. **Sufficient Completenes**: for any $\Sigma$-term $t$ without variables (called a ground term), $t!_E$ is an $\Omega$-term (constructor term).
3. **Sort Preservation** means that if $t$ has sort $s$ and $t \to_{\vec{E}} t'$, then $t'$ also has sort $s$.

# Executability Conditions

What properties should a Maude functional module `fmod` $(\Sigma, E)$ `endfm` with $\Omega \subseteq \Sigma$ its data constructors (the `[ctor]` attribute) have for correct execution? First, for each oriented equation $u \to v$ we must have $vars(v) \subseteq vars(u)$. Furthermore,

1. **Unique termination**. (i) Termination, i.e., no infinite sequences:

$$t_0 \to_{\vec{E}} t_1 \to_{\vec{E}} t_2 \ldots t_n \to_{\vec{E}} t_{n+1} \ldots$$

   (ii) All such sequences from $t$ terminate in a unique result, denoted $t!_E$ (computed by Maude's `red` $t$ command).
2. **Sufficient Completenes**: for any $\Sigma$-term $t$ without variables (called a ground term), $t!_E$ is an $\Omega$-term (constructor term).
3. **Sort Preservation** means that if $t$ has sort $s$ and $t \to_{\vec{E}} t'$, then $t'$ also has sort $s$.

These three properties have a straightforward generalization modulo axioms $B$, replacing the relation $\to_{\vec{E}}$ by the relation $\to_{\vec{E}/B}$.

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem);

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem); but it can often be checked with the MTT and MTA tools.

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem); but it can often be checked with the MTT and MTA tools.
- The unique result property —that is, determinism— is checked by the CRC tool assuming termination.

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem); but it can often be checked with the MTT and MTA tools.
- The unique result property —that is, determinism— is checked by the CRC tool assuming termination.
- The sufficient completeness property can be checked automatically by the SCC tool under mild assumptions.

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem); but it can often be checked with the MTT and MTA tools.
- The unique result property —that is, determinism— is checked by the CRC tool assuming termination.
- The sufficient completeness property can be checked automatically by the SCC tool under mild assumptions.
- Sort preservation is an easy syntactic check also supported by the CRC tool.

# Checking Executability Conditions

The Maude formal environment (MFE) has tools supporting the checking of conditions (1)–(3):

- The termination property is undecidable (the halting problem); but it can often be checked with the MTT and MTA tools.
- The unique result property —that is, determinism— is checked by the CRC tool assuming termination.
- The sufficient completeness property can be checked automatically by the SCC tool under mild assumptions.
- Sort preservation is an easy syntactic check also supported by the CRC tool.

# Determinism = Confluence

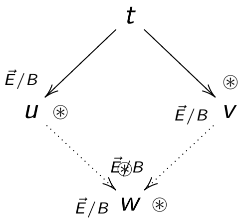
Determinism is captured by confluence. The rules $\vec{E}$ of $(\Sigma, B, \vec{E})$ are confluent modulo $B$ iff for each $\Sigma$-term $t$, whenever $t \rightarrow^{\circledast}_{\vec{E}/B} u$ and $t \rightarrow^{\circledast}_{\vec{E}/B} v$, there is a $w$ such that $u \rightarrow^{\circledast}_{\vec{E}/B} w$ and $v \rightarrow^{\circledast}_{\vec{E}/B} w$.

# Determinism = Confluence

Determinism is captured by confluence. The rules $\vec{E}$ of $(\Sigma, B, \vec{E})$ are confluent modulo $B$ iff for each $\Sigma$-term $t$, whenever $t \rightarrow^{\circledast}_{\vec{E}/B} u$ and $t \rightarrow^{\circledast}_{\vec{E}/B} v$, there is a $w$ such that $u \rightarrow^{\circledast}_{\vec{E}/B} w$ and $v \rightarrow^{\circledast}_{\vec{E}/B} w$. This can be described diagrammatically (dashed arrows denote existential quantification):

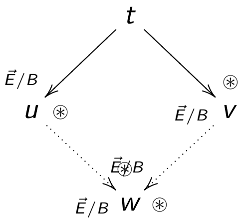
# Determinism = Confluence

Determinism is captured by confluence. The rules $\vec{E}$ of $(\Sigma, B, \vec{E})$ are confluent modulo $B$ iff for each $\Sigma$-term $t$, whenever $t \rightarrow^{\circledast}_{\vec{E}/B} u$ and $t \rightarrow^{\circledast}_{\vec{E}/B} v$, there is a $w$ such that $u \rightarrow^{\circledast}_{\vec{E}/B} w$ and $v \rightarrow^{\circledast}_{\vec{E}/B} w$. This can be described diagrammatically (dashed arrows denote existential quantification):



# Determinism = Confluence

Determinism is captured by confluence. The rules $\vec{E}$ of $(\Sigma, B, \vec{E})$ are confluent modulo $B$ iff for each $\Sigma$-term $t$, whenever $t \to^{\circledast}_{\vec{E}/B} u$ and $t \to^{\circledast}_{\vec{E}/B} v$, there is a $w$ such that $u \to^{\circledast}_{\vec{E}/B} w$ and $v \to^{\circledast}_{\vec{E}/B} w$. This can be described diagrammatically (dashed arrows denote existential quantification):




Under the termination assumption, confluence is decidable and checkable by Maude's Church-Rosser Checker (CRC) tool.

# The Semantics of Functional Modules

In Lecture 1, I stated that:

# The Semantics of Functional Modules

In Lecture 1, I stated that:

"The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

# The Semantics of Functional Modules

In Lecture 1, I stated that:

> "The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

# The Semantics of Functional Modules

In Lecture 1, I stated that:

> "The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It means that there is a first-order interpretation of $(\Sigma, E)$, namely, a $(\Sigma, E)$-algebra $\mathbb{C}_{\Sigma/E}$, called its canonical term algebra, that defines the mathematical meaning of `fmod` $(\Sigma, E)$ `endfm`.

# The Semantics of Functional Modules

In Lecture 1, I stated that:

"The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It means that there is a first-order interpretation of $(\Sigma, E)$, namely, a $(\Sigma, E)$-algebra $\mathbb{C}_{\Sigma/E}$, called its canonical term algebra, that defines the mathematical meaning of `fmod` $(\Sigma, E)$ `endfm`.

In the algebra $\mathbb{C}_{\Sigma/E}$:

# The Semantics of Functional Modules

In Lecture 1, I stated that:

"The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It means that there is a first-order interpretation of $(\Sigma, E)$, namely, a $(\Sigma, E)$-algebra $\mathbb{C}_{\Sigma/E}$, called its canonical term algebra, that defines the mathematical meaning of `fmod` $(\Sigma, E)$ `endfm`.

In the algebra $\mathbb{C}_{\Sigma/E}$: (i) each sort $s \in S$ is interpreted as the set $T_{\Omega,s}$ of (ground) constructor terms of sort $s$; and

# The Semantics of Functional Modules

In Lecture 1, I stated that:

"The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It means that there is a first-order interpretation of $(\Sigma, E)$, namely, a $(\Sigma, E)$-algebra $\mathbb{C}_{\Sigma/E}$, called its canonical term algebra, that defines the mathematical meaning of `fmod` $(\Sigma, E)$ `endfm`.

In the algebra $\mathbb{C}_{\Sigma/E}$: (i) each sort $s \in S$ is interpreted as the set $T_{\Omega,s}$ of (ground) constructor terms of sort $s$; and (i) each function symbol $f : s_1 \dots s_n \to s$ in $\Sigma$ is interpreted as the function:

# The Semantics of Functional Modules

In Lecture 1, I stated that:

"The meaning of a program $P$ is a mathematical model $\mathbb{C}_P$ in first-order logic, called its canonical model."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It means that there is a first-order interpretation of $(\Sigma, E)$, namely, a $(\Sigma, E)$-algebra $\mathbb{C}_{\Sigma/E}$, called its canonical term algebra, that defines the mathematical meaning of `fmod` $(\Sigma, E)$ `endfm`.

In the algebra $\mathbb{C}_{\Sigma/E}$: (i) each sort $s \in S$ is interpreted as the set $T_{\Omega,s}$ of (ground) constructor terms of sort $s$; and (i) each function symbol $f : s_1 \ldots s_n \to s$ in $\Sigma$ is interpreted as the function: $f_{\mathbb{C}_{\Sigma/E}} : T_{\Omega,s_1} \times \ldots \times T_{\Omega,s_n} \ni (t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)!_E \in T_{\Omega,s}$.

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It exactly means what is says: that, for $\varphi$ any first-order $\Sigma$-formula, `fmod` $(\Sigma, E)$ `endfm` satisfies $\varphi$ iff

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It exactly means what is says: that, for $\varphi$ any first-order $\Sigma$-formula, `fmod` $(\Sigma, E)$ `endfm` satisfies $\varphi$ iff

$$\mathbb{C}_{\Sigma/E} \models \varphi.$$

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It exactly means what is says: that, for $\varphi$ any first-order $\Sigma$-formula, `fmod` $(\Sigma, E)$ `endfm` satisfies $\varphi$ iff

$$\mathbb{C}_{\Sigma/E} \models \varphi.$$

The way we prove that $\mathbb{C}_{\Sigma/E} \models \varphi$ is by induction on the constructors $\Omega$ of $\mathbb{C}_{\Sigma/E}$.

# Formal Verification of Functional Modules

In Lecture 1 I also stated that:

> "Saying that program $P$ satisfies a formal property $\varphi$ exactly means that $\mathbb{C}_P \models \varphi$ in the first-order logic sense."

What does this mean for a functional module `fmod` $(\Sigma, E)$ `endfm` satisfying executability conditions (1)–(3) and with $\Omega \subseteq \Sigma$?

It exactly means what is says: that, for $\varphi$ any first-order $\Sigma$-formula, `fmod` $(\Sigma, E)$ `endfm` satisfies $\varphi$ iff

$$\mathbb{C}_{\Sigma/E} \models \varphi.$$

The way we prove that $\mathbb{C}_{\Sigma/E} \models \varphi$ is by induction on the constructors $\Omega$ of $\mathbb{C}_{\Sigma/E}$. This can be done with Maude's ITP tool.

# Unit Testing for Maude Programs

Adrián Riesco at Madrid's Complutense University has developed the **MUnit** tool for unit testing of Maude programs.

# Unit Testing for Maude Programs

Adrián Riesco at Madrid's Complutense University has developed the **MUnit** tool for unit testing of Maude programs.

A description of **MUnit** as well as pointer to its `github` repository can be found in the paper:

# Unit Testing for Maude Programs

Adrián Riesco at Madrid's Complutense University has developed the **MUnit** tool for unit testing of Maude programs.

A description of **MUnit** as well as pointer to its `github` repository can be found in the paper:

A. Riesco, "MUnit: A Unit Framework for Maude," Proc. WRLA 2018, LNCS 11152, pp. 45–58, 2018.