Program Verification: Lecture 24

José Meseguer

University of Illinois
at Urbana-Champaign
Case Analysis Rule

\{u_1, \ldots, u_k\} \subseteq T_{\Omega}(X) \text{ as a pattern set for } s \iff T_{\Omega}, s = \bigcup_{1 \leq l \leq k} \{u_l \varphi \mid \varphi \in \mathcal{X} \rightarrow T_{\Omega}\}.

Example. \{0, s(x)\} and \{0, s(0), s(s(y))\} are pattern sets for Nat.

The following auxiliary rule allows reasoning by cases:

\[\begin{align*}
&\text{Case Analysis} \\
&\bigwedge_{1 \leq l \leq k} [A, C] \vdash T(u | \varphi) \{x : s \mapsto \rightarrow u_l\} \rightarrow \ast \{x : s \mapsto \rightarrow u_l\} [A, C] \vdash T(u | \varphi) \rightarrow \ast[A, C]
\end{align*}\]

where \(x : s \in \text{vars}(u)\) and \(\{u_1, \ldots, u_k\}\) is a pattern set for \(s\).
Call \( \{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s \) a pattern set for sort \( s \) iff
\[
T_{\Omega,s} = \bigcup_{1 \leq l \leq k}\{u_l \rho \mid \rho \in [X \to T_\Omega]\}.
\]
Case Analysis Rule

Call \( \{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s \) a \textit{pattern set} for sort \( s \) iff
\[
T_{\Omega,s} = \bigcup_{1 \leq l \leq k} \{u_l \rho \mid \rho \in [X \to T_\Omega]\}.
\]

\textbf{Example.} \( \{0, s(x)\} \) and \( \{0, s(0), s(s(y))\} \) are pattern sets for \( Nat \).
Case Analysis Rule

Call \( \{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s \) a pattern set for sort \( s \) iff
\[
T_{\Omega,s} = \bigcup_{1 \leq l \leq k}\{u_l \rho \mid \rho \in [X \to T_\Omega]\}.
\]

**Example.** \( \{0, s(x)\} \) and \( \{0, s(0), s(s(y))\} \) are pattern sets for \( Nat \).

The following auxiliary rule allows reasoning by cases:
Case Analysis Rule

Call \( \{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s \) a **pattern set** for sort \( s \) iff
\[
T_{\Omega,s} = \bigcup_{1 \leq l \leq k} \{ u_l \rho \mid \rho \in [X \to T_\Omega] \}.
\]

**Example.** \( \{0, s(x)\} \) and \( \{0, s(0), s(s(y))\} \) are pattern sets for \( Nat \).

The following auxiliary rule allows reasoning by cases:

**Case Analysis**

\[
\bigwedge_{1 \leq l \leq k} [A, C] \vdash_T (u \mid \varphi)\{x:s \mapsto u_l\} \xrightarrow{\ast} A\{x:s \mapsto u_l\}
\]

\[
[A, C] \vdash_T u \mid \varphi \xrightarrow{\ast} A
\]
Call \( \{u_1, \ldots, u_k\} \subseteq T_\Omega(X)_s \) a **pattern set** for sort \( s \) iff
\[
T_{\Omega,s} = \bigcup_{1 \leq l \leq k} \{u_l \rho \mid \rho \in [X \to T_\Omega]\}.
\]

**Example.** \( \{0, s(x)\} \) and \( \{0, s(0), s(s(y))\} \) are pattern sets for \( Nat \).

The following auxiliary rule allows reasoning by cases:

**Case Analysis**

\[
\begin{align*}
\bigwedge_{1 \leq l \leq k} \left[ A, C \right] & \vdash_T (u \mid \varphi)\{x : s \mapsto u_l\} \rightarrow^* A\{x : s \mapsto u_l\} \\
\left[ A, C \right] & \vdash_T u \mid \varphi \rightarrow^* A
\end{align*}
\]

where \( x : s \in \text{vars}(u) \) and \( \{u_1, \ldots, u_k\} \) is a pattern set for \( s \).
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \rightarrow^{\ast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow^{\ast} B$. 

How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude Reachability Logic Prover. To use this tool to prove properties of a rewrite theory specified as a system module `FOO` you:

1. Load `FOO` into Maude
2. Give to Maude the command `load rltool`
3. Form now on, all your commands are given to the tool, and not really to Maude. They should be enclosed in parentheses and ended by a period right before the closing parenthesis (as for Full Maude).

   The first such command should be: `(select FOO .)`
Suppose we want to prove that a rewrite theory \( \mathcal{R} = (\Sigma, B, R) \) satisfies a reachability formula \( A \rightarrow^\ast B \), denoted \( \mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B \). How can we do it?
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \rightarrow^\ast B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B$. How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool:
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \xrightarrow{\circ} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \xrightarrow{\circ} B$. How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. 
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \rightarrow^\star B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\star B$. How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module `FOO` you:

1. Load `FOO` into Maude
2. Give to Maude the command `load rltool`
3. Form now on, all your commands are given to the tool, and not really to Maude. They should be enclosed in parentheses and ended by a period right before the closing parenthesis (as for Full Maude). The first such command should be: `(select FOO .)`
Suppose we want to prove that a rewrite theory \( R = (\Sigma, B, R) \) satisfies a reachability formula \( A \xrightarrow{\star} B \), denoted \( R = (\Sigma, B, R) \vdash A \xrightarrow{\star} B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude Reachability Logic Prover. To use this tool to prove properties of a rewrite theory specified as a system module F00 you:

1. load F00 into Maude
Suppose we want to prove that a rewrite theory \( R = (\Sigma, B, R) \) satisfies a reachability formula \( A \xrightarrow{\star} B \), denoted \( R = (\Sigma, B, R) \models A \xrightarrow{\star} B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module FOO you:

1. load FOO into Maude
2. give to Maude the command
Suppose we want to prove that a rewrite theory \( \mathcal{R} = (\Sigma, B, R) \) satisfies a reachability formula \( A \rightarrow^\ast B \), denoted \( \mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module F00 you:

1. load F00 into Maude
2. give to Maude the command
   load rltool
Proving Formulas in the Reachability Logic Tool

Suppose we want to prove that a rewrite theory \( \mathcal{R} = (\Sigma, B, R) \) satisfies a reachability formula \( A \rightarrow^\ast B \), denoted \( \mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude \textit{Reachability Logic Prover}. To use this tool to prove properties of a rewrite theory specified as a system module \texttt{FOO} you:

1. load \texttt{FOO} into Maude
2. give to Maude the command
   
   \texttt{load rltool}
3. Form now on, all your commands are given to the tool, and not really to Maude.
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \rightarrow^\ast B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B$. How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module `FOO` you:

1. load `FOO` into Maude
2. give to Maude the command  
   ```
   load rltool
   ```
3. Form now on, all your commands are given to the tool, and not really to Maude. They should be enclosed in parentheses and ended by a period right before the closing parenthesis (as for Full Maude).
Suppose we want to prove that a rewrite theory \( R = (\Sigma, B, R) \) satisfies a reachability formula \( A \rightarrow^\star B \), denoted \( R = (\Sigma, B, R) \models A \rightarrow^\star B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module `FOO` you:

1. load `FOO` into Maude
2. give to Maude the command
   `load rltool`
3. Form now on, all your commands are given to the tool, and not really to Maude. They should be enclosed in parentheses and ended by a period right before the closing parenthesis (as for Full Maude). The first such command should be:
Suppose we want to prove that a rewrite theory \( R = (\Sigma, B, R) \)
satisfies a reachability formula \( A \xrightarrow{\text{\star}} B \), denoted 
\( R = (\Sigma, B, R) \models A \xrightarrow{\text{\star}} B \). How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool: the Maude *Reachability Logic Prover*. To use this tool to prove properties of a rewrite theory specified as a system module `FOO` you:

1. load `FOO` into Maude  
2. give to Maude the command  
   
   ```maude
   load rltool
   ```  
3. Form now on, all your commands are given to the tool, and not really to Maude. They should be enclosed in parentheses and ended by a period right before the closing parenthesis (as for Full Maude). The first such command should be:  
   ```maude
   (select FOO .)
   ```
After this you will be ready to give commands to the tool to:

Reachability Logic Tool Commands

After this you will be ready to give commands to the tool to:
After this you will be ready to give commands to the tool to: (i) enter goals, and (ii) prove such goals.
After this you will be ready to give commands to the tool to: (i) enter goals, and (ii) prove such goals. As for other Maude tools, there is a grammar for all such commands.
After this you will be ready to give commands to the tool to: (i) enter goals, and (ii) prove such goals. As for other Maude tools, there is a grammar for all such commands. A first fragment is:
After this you will be ready to give commands to the tool to: (i) enter goals, and (ii) prove such goals. As for other Maude tools, there is a grammar for all such commands. A first fragment is:

\[
\begin{align*}
\text{VariableName} & \ ::= \ <\text{Special}> \\
\text{ModuleName} & \ ::= \ <\text{Special}> \\
\text{Term} & \ ::= \ <\text{Special}> \\
\text{Atom} & \ ::= \ (\text{Term})=(\text{Term}) \\
& \quad | \ (\text{Term})=/=(\text{Term}) \\
\text{Conjunction} & \ ::= \ true \\
& \quad | \ \text{Atom} \\
& \quad | \ \text{Conjunction} \ \text{\textbackslash\textbackslash} \ \text{Conjunction} \\
\text{Pattern} & \ ::= \ (\text{Term}) \ "|" \ \text{Conjunction} \\
\text{PatternFormula} & \ ::= \ \text{Pattern} \\
& \quad | \ \text{PatternFormula} \ \text{\textbackslash\textbackslash} \ \text{PatternFormula} \\
\text{RFormula} & \ ::= \ \text{Pattern} \ =>\text{A PatternFormula}
\end{align*}
\]
For example, for CHOICE, the reachability formula
For example, for CHOICE, the reachability formula

\[ \{M\} | \top \xrightarrow{\ast} \{M'\} | M' \subseteq M = tt \]
For example, for CHOICE, the reachability formula

\[ \{ M \} \mid \top \xrightarrow{\ast} \{ M' \} \mid M' \subseteq M = tt \]

is expressed in this grammar as:

\((\{M:\text{MSet}\}) \mid \text{true} \Rightarrow A \)

\((\{M':\text{MSet}\}) \mid (M':\text{MSet} = C M:\text{MSet}) = (tt)\)
For example, for CHOICE, the reachability formula

\[
\{M\} \mid \top \xrightarrow{\ast} \{M'\} \mid M' \subseteq M = tt
\]

is expressed in this grammar as:

\[
(\{M:\text{MSet}\}) \mid \text{true} \Rightarrow A
\]

\[
(\{M':\text{MSet}\}) \mid (M':\text{MSet} = C M:\text{MSet}) = (tt)
\]

We can now give commands according to the following grammar:
Reachability Logic Tool Commands (III)

Nat ::= <Special>

GoalName ::= Nat | Nat GoalName

TermSet ::= {Term} | TermSet U TermSet

Command ::= (select ModuleName .)
| (subsumed Pattern =< Pattern .)
| (add-goal RFormula .)
| (def-term-set PatternFormula .)
| (start-proof .)
| (step .)
| (step Nat .)
| (step* .)
| (case GoalName on VariableName by TermSet .)
| (quit .)
Nat ::= <Special>
GoalName ::= Nat | Nat GoalName
TermSet ::= {Term} | TermSet U TermSet
Command ::= (select ModuleName .)
| (subsumed Pattern =< Pattern .)
| (add-goal RFormula .)
| (def-term-set PatternFormula .)
| (start-proof .)
| (step .)
| (step Nat .)
| (step* .)
| (case GoalName on VariableName by TermSet .)
| (quit .)

Let us illustrate each of these commands.
Nat ::= <Special>
GoalName ::= Nat | Nat GoalName
TermSet ::= \{Term\} | TermSet U TermSet
Command ::= (select ModuleName .)
| (subsumed Pattern =< Pattern .)
| (add-goal RFormula .)
| (def-term-set PatternFormula .)
| (start-proof .)
| (step .)
| (step Nat .)
| (step* .)
| (case GoalName on VariableName by TermSet .)
| (quit .)

Let us illustrate each of these commands.
Many of the properties we will prove are *invariants* using:

\[
\text{Corollary}
\]

If \( J S_0 \subseteq J B K \) and \( B \leftarrow \top \) holds in R \( \text{stop} \), then \( B \) is an invariant for \( R \) from initial states \( S_0 \).

To discharge the proof obligation \( J \langle 0,0 \rangle | \top K \subseteq J \text{Mutex} 1 K \) we use the command \((\text{subsumed Pattern} = \langle \text{Pattern} . \rangle)\) For example, in \( \text{READERS-WRITERS-stop} \), proving the invariant \( \text{Mutex} = \langle R,W \rangle | W = 0 \lor (W = 1 \land R = 0) \) requires that we first check \( J \langle 0,0 \rangle | \top K \subseteq J \text{Mutex} 1 K \) by giving the command:

\[
(\text{subsumed} (< 0,0 >) | \text{true} = < (< R:Nat,W:Nat >) | (W:Nat) = (0) .)
\]

because in the current tool syntax the condition in a pattern must be a conjunction so that \( \text{Mutex} \) is decomposed as:

\[
\text{Mutex} 1 = \langle R,W \rangle | W = 0 \quad \text{and} \quad \text{Mutex} 2 = \langle R,W \rangle | W = 1 \land R = 0.
\]
Reachability Logic Tool Commands (IV)

Many of the properties we will prove are *invariants* using:

**Corollary**

\[
\text{If } [S_0] \subseteq [B] \text{ and } B \rightarrow^\otimes [B] \text{ holds in } \mathcal{R}_{\text{stop}}, \text{ then } B \text{ is an invariant for } \mathcal{R} \text{ from initial states } S_0. 
\]
Many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\otimes} [B]$ holds in $R_{stop}$, then $B$ is an invariant for $R$ from initial states $S_0$.

To discharge the proof obligation $[S_0] \subseteq [B]$
Many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\circledast} [B]$ holds in $R_{stop}$, then $B$ is an invariant for $R$ from initial states $S_0$.

To discharge the proof obligation $[S_0] \subseteq [B]$ we use the command (subsumed Pattern $\xleftarrow{\text{Pattern}}$).

Reachability Logic Tool Commands (IV)
Many of the properties we will prove are invariants using:

**Corollary**

If \( [S_0] \subseteq [B] \) and \( B \xrightarrow{\text{stop}} [B] \) holds in \( \mathcal{R}_{\text{stop}} \), then \( B \) is an invariant for \( \mathcal{R} \) from initial states \( S_0 \).

To discharge the proof obligation \( [S_0] \subseteq [B] \) we use the command (subsumed Pattern =< Pattern .)

For example, in READERS-WRITERS-stop, proving the invariant \( \text{Mutex} = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0) \)
Many of the properties we will prove are *invariants* using:

**Corollary**

*If* \([ S_0 ] \subseteq [ B ] \) *and* \( B \xrightarrow{\text{stop}} [ B ] \) *holds in* \( \mathcal{R}_{\text{stop}} \), *then* \( B \) *is an invariant for* \( \mathcal{R} \) *from initial states* \( S_0 \).

To discharge the proof obligation \([ S_0 ] \subseteq [ B ] \) we use the command (subsumed Pattern =< Pattern .)

For example, in `READERS-WITERS-stop`, proving the invariant

\[ \text{Mutex} = \langle R, W \rangle | W = 0 \lor (W = 1 \land R = 0) \]

requires that we first check \([\langle 0, 0 \rangle | \top ] \subseteq [\text{Mutex}_1] \) by giving the command:
Many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\text{stop}} [B]$ holds in $\mathcal{R}_{\text{stop}}$, then $B$ is an invariant for $\mathcal{R}$ from initial states $S_0$.

To discharge the proof obligation $[S_0] \subseteq [B]$ we use the command (subsumed Pattern $\Rightarrow$ Pattern).

For example, in READERS–WRITERS–stop, proving the invariant $\text{Mutex} = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0)$ requires that we first check $[\langle 0, 0 \rangle \mid \top] \subseteq [\text{Mutex}_1]$ by giving the command:

(subsumed ($< 0,0 >$) $\mid$ true $\Rightarrow$ ($< R:\text{Nat},W:\text{Nat} >$) $\mid$ (W: Nat) = (0).)
Reachability Logic Tool Commands (IV)

Many of the properties we will prove are *invariants* using:

**Corollary**

*If* $[S_0] \subseteq [B]$ *and* $B \overset{\circ}{\rightarrow} [B]$ *holds in* $\mathcal{R}_{\text{stop}}$, *then* $B$ *is an invariant for* $\mathcal{R}$ *from initial states* $S_0$.

To discharge the proof obligation $[S_0] \subseteq [B]$ *we use the command* (subsumed Pattern =< Pattern .)

For example, in READERS–WRITERS–stop, proving the invariant $Mutex = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0)$ requires that we first check $[\langle 0, 0 \rangle \mid \top] \subseteq [Mutex_1]$ by giving the command:

(subsumed (< 0,0 >) | true =< (< R: Nat,W: Nat >) | (W: Nat) = (0) .)

because in the current tool syntax the condition in a pattern must be a *conjunction* so that $Mutex$ is decomposed as:
Reachability Logic Tool Commands (IV)

Many of the properties we will prove are *invariants* using:

**Corollary**

If \([S_0] \subseteq [B]\) and \(B \rightarrow^\circ [B]\) holds in \(R_{stop}\), then \(B\) is an invariant for \(R\) from initial states \(S_0\).

To discharge the proof obligation \([S_0] \subseteq [B]\) we use the command (subsumed Pattern =< Pattern .)

For example, in READERS-WRITERS-stop, proving the invariant \(Mutex = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0)\) requires that we first check \([\langle 0, 0 \rangle \mid \top] \subseteq [Mutex_1]\) by giving the command:

(subsumed (< 0,0 >) | true =< (< R: Nat, W: Nat >) | (W: Nat) = (0) .)

because in the current tool syntax the condition in a pattern must be a *conjunction* so that \(Mutex\) is decomposed as:

\(Mutex_1 = \langle R, W \rangle \mid W = 0\) and
Many of the properties we will prove are \textit{invariants} using:

\textbf{Corollary}

\textit{If \([S_0] \subseteq [B]\) and \(B \xrightarrow{\mathcal{R}_{\text{stop}}} [B]\) holds in \(\mathcal{R}_{\text{stop}}\), then \(B\) is an invariant for \(\mathcal{R}\) from initial states \(S_0\).}

To discharge the proof obligation \([S_0] \subseteq [B]\) we use the command (subsumed Pattern \(=\langle\) Pattern \(\rangle\).)

For example, in READERS–WRITERS–stop, proving the invariant \(\text{Mutex} = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0)\) requires that we first check \([\langle 0, 0 \rangle \mid \top] \subseteq [\text{Mutex}_1]\) by giving the command:

\[
\text{(subsumed} (< 0, 0 >) \mid \text{true} =\langle (< R:\text{Nat}, W:\text{Nat} >) \mid (W:\text{Nat}) = (0) \rangle)
\]

because in the current tool syntax the condition in a pattern must be a \textit{conjunction} so that \(\text{Mutex}\) is decomposed as:

\[
\text{Mutex}_1 = \langle R, W \rangle \mid W = 0 \quad \text{and} \quad \text{Mutex}_2 = \langle R, W \rangle \mid W = 1 \land R = 0.
\]
The set $[T]$ of terminating states should also be specified as a pattern formula $T$. 
The set \([T]\) of *terminating states* should also be specified as a *pattern formula* \(T\). We only require \([T]\) to be *contained* in, or equal to, the set of *all* terminating states.
The set $[T]$ of \textit{terminating states} should also be specified as a \textit{pattern formula} $T$. We only require $[T]$ to be \textit{contained} in, or equal to, the set of \textit{all} terminating states. This allows more detailed reasoning about $T$-terminating sequences to localize the reasoning to $T$ by the inference relation $\vdash_T$ (see inference rules).
The set \([T]\) of *terminating states* should also be specified as a *pattern formula* \(T\). We only require \([T]\) to be *contained* in, or equal to, the set of *all* terminating states. This allows more detailed reasoning about \(T\)-terminating sequences to localize the reasoning to \(T\) by the inference relation \(\vdash_T\) (see inference rules).

In this way we can prove invariants for *any* rewrite theory \(\mathcal{R}\), terminating, non-terminating, or never-terminating, by defining:
The set \([T]\) of \textit{terminating states} should also be specified as a \textit{pattern formula} \(T\). We only require \([T]\) to be \textit{contained} in, or equal to, the set of \textit{all} terminating states. This allows more detailed reasoning about \(T\)-terminating sequences to localize the reasoning to \(T\) by the inference relation \(\vdash_T\) (see inference rules).

In this way we can prove invariants for \textit{any} rewrite theory \(\mathcal{R}\), terminating, non-terminating, or never-terminating, by defining:

\[
T = [x_1, \ldots, x_n] \mid \top \quad \text{as terminating states in } \mathcal{R}_{\text{stop}}.
\]
The set \([T]\) of terminating states should also be specified as a pattern formula \(T\). We only require \([T]\) to be contained in, or equal to, the set of all terminating states. This allows more detailed reasoning about \(T\)-terminating sequences to localize the reasoning to \(T\) by the inference relation \(\vdash_T\) (see inference rules).

In this way we can prove invariants for any rewrite theory \(\mathcal{R}\), terminating, non-terminating, or never-terminating, by defining: \(T = [x_1, \ldots, x_n] \mid \top\) as terminating states in \(\mathcal{R}_{\text{stop}}\).

For example for READERS–WRITERS–stop, we specify \(T\) by giving the command:
The set $[T]$ of terminating states should also be specified as a pattern formula $T$. We only require $[T]$ to be contained in, or equal to, the set of all terminating states. This allows more detailed reasoning about $T$-terminating sequences to localize the reasoning to $T$ by the inference relation $\vdash_T$ (see inference rules).

In this way we can prove invariants for any rewrite theory $\mathcal{R}$, terminating, non-terminating, or never-terminating, by defining: $T = [x_1, \ldots, x_n] \mid \top$ as terminating states in $\mathcal{R}_{stop}$.

For example for READERS–WRITERS–stop, we specify $T$ by giving the command:

(def-term-set ([R:Nat,W:Nat]) | true .)
Recall that in general we need to prove a set $C$ of reachability formulas,

```
(add-goal RFormula .)
```

For example, in CHOICE, to enter the formula

```
{M} | ⊤ \rightarrow ⊛ {M'} | M' \subseteq M = tt
```

we give the command:

```
(add-goal (({M : MSet}) | true => A (({M' : MSet}) | (M' : MSet =C M : MSet)) = (tt) .)
```

The tool gives each entered goal a number. It will later generate subgoals named by number sequences $n_1$...$n_k$, naming goal $n_1$•...•$n_k$, such as 2 3 1 as the first child of child 3 of goal 2.
Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \rightarrow^{\circ} B$ and perhaps some auxiliary lemmas.
Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \xrightarrow{\circ} B$ and perhaps some auxiliary lemmas. To enter to the tool each formula in $C$ we give the command:

$$(\text{add-goal} \ RFormula .)$$

For example, in CHOICE, to enter the formula

$${\{M\}} | \top \xrightarrow{\circ} {\{M'\}} | M' \subseteq M = \text{tt}$$

we give the command:

$$(\text{add-goal} \ (\{M: MSet\}) | \text{true} \Rightarrow A \ (\{M': MSet\}) | (M': MSet =_C M: MSet) = (\text{tt}) .)$$

The tool gives each entered goal a number. It will later generate subgoals named by number sequences $n_1 \ldots n_k$, naming goal $n_1 \ldots n_k$, such as $2 3 1$ as the first child of child 3 of goal 2.

Recall that in general we need to prove a set $C$ of reachability formulas, including the *main formula* $A \rightarrow^{\circledast} B$ and perhaps some *auxiliary lemmas*. To enter to the tool each formula in $C$ we give the command: (add-goal RFormula .)

For example, in CHOICE, to enter the formula$
\{M\}|\top \rightarrow^{\circledast} \{M'\}|M' \subseteq M = \texttt{tt}$
we give the command:

```
(add-goal (\{M:MSet\}) | true =>\(\{M':MSet\}\) | (M':MSet =C M:MSet) = (tt) .)
```

The tool gives each entered goal a number. It will later generate subgoals named by number sequences $n_1 \ldots n_k$, naming goal $n_1 \ldots n_k$, such as $2 3 1$ as the first child of child 3 of goal 2.
Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \rightarrow^{\odot} B$ and perhaps some auxiliary lemmas. To enter to the tool each formula in $C$ we give the command: (add-goal RFormula .)

For example, in CHOICE, to enter the formula
Reachability Logic Tool Commands (VI)

Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \rightarrow^\star B$ and perhaps some auxiliary lemmas. To enter to the tool each formula in $C$ we give the command: 

$$(\text{add-goal } \text{RFormula } ).$$

For example, in CHOICE, to enter the formula

$$\{M\} | \top \rightarrow^\star \{M'\} \mid M' \subseteq M = tt$$
Recall that in general we need to prove a set \( C \) of reachability formulas, including the main formula \( A \longrightarrow^{\circ} B \) and perhaps some auxiliary lemmas. To enter to the tool each formula in \( C \) we give the command: \((\text{add-goal} \text{ RFormula .})\)

For example, in \textsc{Choice}, to enter the formula

\[
\{ M \} \mid \top \longrightarrow^{\circ} \{ M' \} \mid M' \subseteq M = tt
\]

we give the command:
Reachability Logic Tool Commands (VI)

Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \rightarrow^{\ast} B$ and perhaps some auxiliary lemmas. To enter to the tool each formula in $C$ we give the command: (add-goal RFormula .)

For example, in CHOICE, to enter the formula

$$\{M\} | \top \rightarrow^{\ast} \{M'\} | M' \subseteq M = \text{tt}$$

we give the command:

(\text{add-goal (}}\{M:\text{MSet}\}) | \text{true} \Rightarrow A
(\{M':\text{MSet}\}) | (M':\text{MSet} =_{C} M:\text{MSet}) = (\text{tt}) .)
Recall that in general we need to prove a set $C$ of reachability formulas, including the *main formula* $A \xrightarrow{\circ} B$ and perhaps some *auxiliary lemmas*. To enter to the tool each formula in $C$ we give the command: $(\text{add-goal RFormula .})$

For example, in CHOICE, to enter the formula

$$\{M\} | \top \xrightarrow{\circ} \{M'\} | M' \subseteq M = \text{tt}$$

we give the command:

$$(\text{add-goal (\{M:MSet\}) | true =>A (\{M':MSet\}) | (M':MSet =C M:MSet) = (\text{tt}) .})$$

The tool gives each entered goal a number.
Recall that in general we need to prove a set $C$ of reachability formulas, including the *main formula* $A \rightarrow^\circ B$ and perhaps some *auxiliary lemmas*. To enter to the tool each formula in $C$ we give the command: (add-goal RFormula .)

For example, in CHOICE, to enter the formula

$$\{M\} \mid \top \rightarrow^\circ \{M'\} \mid M' \subseteq M = tt$$

we give the command:

```
(add-goal (\{M:MSet\}) \mid \text{true} \Rightarrow \text{A}
\quad (\{M':\text{MSet}\}) \mid (M':\text{MSet} =C M:\text{MSet}) = (tt) .)
```

The tool gives each entered goal a number. It will later generate *subgoals* named by *number sequences* $n_1 \ldots n_k$. 
Recall that in general we need to prove a set $C$ of reachability formulas, including the \textit{main formula} $A \xrightarrow{\star} B$ and perhaps some \textit{auxiliary lemmas}. To enter to the tool each formula in $C$ we give the command: $(\text{add-goal } \text{RFormula } )$

For example, in CHOICE, to enter the formula

$$\{M\} \mid \top \xrightarrow{\star} \{M'\} \mid M' \subseteq M = \text{tt}$$

we give the command:

$$(\text{add-goal } (\{M:\text{MSet}\}) \mid \text{true } \Rightarrow A
\quad (\{M':\text{MSet}\}) \mid (M':\text{MSet} =_{\text{C}} M:\text{MSet}) = (\text{tt}) ).$$

The tool gives each entered goal a number. It will later generate \textit{subgoals} named by \textit{number sequences} $n_1 \ldots n_k$, naming goal $n_1 \bullet \ldots \bullet n_k$, such as
Recall that in general we need to prove a set $C$ of reachability formulas, including the *main formula* $A \rightarrow^\circ B$ and perhaps some *auxiliary lemmas*. To enter to the tool each formula in $C$ we give the command: (add-goal RFormula .)

For example, in CHOICE, to enter the formula

$$\{M\} \mid \top \rightarrow^\circ \{M'\} \mid M' \subseteq M = tt$$

we give the command:

(add-goal (\{M:MSet\}) | true =A

(\{M':MSet\}) | (M':MSet =C M:MSet) = (tt) .)

The tool gives each entered goal a number. It will later generate *subgoals* named by *number sequences* $n_1 \ldots n_k$, naming goal $n_1 \bullet \ldots \bullet n_k$, such as 2 3 1 as the first child of child 3 of goal 2.
After:

(i) checking containments of the form $J_S^0 \subseteq J_B^K$ with the (subsumed Pattern =< Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal $RFormula$ .) command, we can start the proof process by giving the (start-proof .) command.

If we want to see which goals are obtained by one (resp. $n$) step(s) of applying some rule of inference to each of current goals we give the command: (step .) (resp. (step $n$ .)). Instead, if we want to go to the end of the proof process in the hope that it will terminate we give the (step* .) command. And at any time we can quit giving the (quit .) command.
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern =< Pattern .) command
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\preceq$ Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal RFormula .) command,
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\implies$ Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal RFormula .) command, we can start the proof process by giving the (start-proof .) command.
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern =< Pattern .) command and (ii) adding all goals in $\mathcal{C}$ to the tool with the (add-goal RFormula .) command, we can start the proof process by giving the (start-proof .) command.

If we want to see which goals are obtained by one (resp. $n$) step(s) of applying some rule of inference to each of current goals we give the command:
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern =$\preceq$ Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal RFormula .) command, we can start the proof process by giving the (start-proof .) command.

If we want to see which goals are obtained by one (resp. $n$) step(s) of applying some rule of inference to each of current goals we give the command: (step .) (resp. (step $n$ .)).
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\leftarrow$ Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal $R$Formula .) command, we can start the proof process by giving the (start-proof .) command.

If we want to see which goals are obtained by one (resp. $n$) step(s) of applying some rule of inference to each of current goals we give the command: (step .) (resp. (step $n$ .)).

Instead, if we want to go to the end of the proof process in the hope that it will terminate we give the (step* .) command.
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\Rightarrow$ Pattern .) command and (ii) adding all goals in $C$ to the tool with the (add-goal RFormula .) command, we can start the proof process by giving the (start-proof .) command.

If we want to see which goals are obtained by one (resp. $n$) step(s) of applying some rule of inference to each of current goals we give the command: (step .) (resp. (step $n$ .)).

Instead, if we want to go to the end of the proof process in the hope that it will terminate we give the (step* .) command. And at any time we can quit giving the (quit .) command.
At any time in the proof process we can apply the Case Analysis rule to a goal named with a number list \( l \) to decompose it into several subgoals by giving the command:
At any time in the proof process we can apply the **Case Analysis** rule to a goal named with a number list \( l \) to decompose it into several subgoals by giving the command:

\[
\text{(case GoalName on VariableName by TermSet .)}
\]

For example, if we want to do case analysis on the goal

\[
\begin{align*}
\{\text{M}:\text{MSet}\} & \mid \text{true} \Rightarrow \{\text{M'}:\text{MSet}\} \mid (\text{M'}:\text{MSet} = \text{C} \text{M}:\text{MSet}) = (\text{tt})
\end{align*}
\]

which was named, say, as goal 1 by the tool, using the pattern set \( \{\text{N}:\text{Nat}, \text{M}_1:\text{MSet}, \text{M}_2:\text{MSet}\} \), we will give the command:

\[
\text{(case 1 on M:MSet by }\{\text{N}:\text{Nat}\} \cup \{\text{M}_1:MSet, \text{M}_2:MSet\}.)
\]
At any time in the proof process we can apply the **Case Analysis** rule to a goal named with a number list $l$ to decompose it into several subgoals by giving the command:

$$(\text{case GoalName on VariableName by TermSet .})$$

For example, if we want to do case analysis on the goal

$$(\{M:\text{MSet}\} \mid \text{true} \Rightarrow A(\{M':\text{MSet}\}) \mid (M':\text{MSet} = \text{C} M:\text{MSet}) = (\text{tt})$$

which was named, say, as goal 1 by the tool, using the pattern set $\{N:\text{Nat}, M_1:\text{MSet}, M_2:\text{MSet}\}$, we will give the command:

$$(\text{case 1 on M:\text{MSet} by } \{N:\text{Nat}\} U \{M_1:\text{MSet}, M_2:\text{MSet}\} .)$$
At any time in the proof process we can apply the **Case Analysis** rule to a goal named with a number list $l$ to decompose it into several subgoals by giving the command:

$$(\text{case } \text{GoalName on } \text{VariableName by } \text{TermSet} \, .)$$

For example, if we want to do case analysis on the goal

$$(\{M: \text{MSet}\}) \mid \text{true} \Rightarrow A (\{M': \text{MSet}\}) \mid (M': \text{MSet} = C M: \text{MSet}) = (tt)$$
At any time in the proof process we can apply the **Case Analysis** rule to a goal named with a number list \( l \) to decompose it into several subgoals by giving the command:

\[
\text{case GoalName on VariableName by TermSet .}
\]

For example, if we want to do case analysis on the goal

\[
(\{M\}:\text{MSet}\}) \mid \text{true} \Rightarrow A (\{M’\}:\text{MSet}\}) \mid (M’:\text{MSet} =\langle M\!\!\:\text{MSet} ) = (tt)
\]

which was named, say, as goal 1 by the tool, using the pattern set \( \{N:\text{Nat}, M_1:\text{MSet} M_2:\text{MSet}\}\), we will give the command:
At any time in the proof process we can apply the **Case Analysis** rule to a goal named with a number list \( l \) to decompose it into several subgoals by giving the command:

\[
\text{(case GoalName on VariableName by TermSet .)}
\]

For example, if we want to do case analysis on the goal

\[
({M: MSet}) \mid \text{true} \Rightarrow A ({M’: MSet}) \mid (M’: MSet =_C M: MSet) = (tt)
\]

which was named, say, as goal 1 by the tool, using the pattern set \( \{N: Nat, M_1: MSet, M_2: MSet\} \), we will give the command:

\[
\text{(case 1 on M: MSet by \{N: Nat\} U \{M1: MSet, M2: MSet\} .)}
\]
We first recall the CHOICE module from Lecture 23

mod CHOICE is
    protecting NAT.
    sorts MSet State Pred.
    subsorts Nat < MSet.
    op __ : MSet MSet -> MSet [ctor assoc comm].
    op {_} : MSet -> State.
    op tt : -> Pred [ctor].
    op _=C_ : MSet MSet -> Pred [ctor].
    vars U V : MSet. var N : Nat.
    eq U =C U = tt.
    eq U =C U V = tt.
    rl [choice] : {U V} => {U}.
endm
Example Proofs (II)

Also recall the Hoare Triple from Lecture 23:

\[\{M \mid \top\} \_CHOICE\ \{N \mid N \subseteq M = tt\}\]

In the tool notation, we can write this as the reachability formula:

\[\langle\{M : \text{MSet}\}\rangle \mid \text{true} \Rightarrow A\]
\[\langle\{N : \text{Nat}\}\rangle \mid (N : \text{Nat} =_C M : \text{MSet}) = (tt)\]

Sometimes, we cannot prove a goal as-is and must analyze cases; this formula is one such example
Example Proofs (III)

\[
\{M : MSet\} \mid \text{true} \implies A \\
\{N : \text{Nat}\} \mid (N : \text{Nat} = C M : MSet) = (tt)
\]

The case analysis occurs on variable \(M : MSet\);
Two cases: \(M : MSet \mapsto N : \text{Nat}\) (or) \(M : MSet \mapsto M1 : MSet\ M2 : MSet\)

Recall any terminating state in this theory has the form \(\{N : \text{Nat}\}\)

Now we are ready to prove this example in the tool
The full proof script is given below:

```plaintext
load choice.maude
load rltool.maude
(select module CHOICE .)
(def-term-set ({N:Nat}) | true .)
(add-goal ({M:MSet}) | true =>A
   ({N:Nat}) | (N:Nat =C M:MSet) = (tt) .)
(start-proof .)
(case 1 on M:MSet by {K:Nat} U {M1:MSet M2:MSet} .)
(step* .)

Note: 3 proof rules sufficient to prove triple for all multisets
```
Q: Does the system handle general reachability formulas as nicely?

A: Let us illustrate by example...

Recall the CHOICE reachability formula from Lecture 23:

\[
\{ M \} \mid \top \rightarrow^\ast \{ M' \} \mid M' \subseteq M = \text{tt}
\]

Expressible in the tool notation as:

\[
(\{ M : \text{MSet} \}) \mid \text{true} \Rightarrow A
\]

\[
(\{ M' : \text{Nat} \}) \mid (M' : \text{Nat} =_{C} M : \text{MSet}) = (\text{tt})
\]

We expect the proof will be similar to its Hoare Triple cousin...
The proof script confirms our suspicions:

load choice.maude
load rltool.maude
(select module CHOICE .)
(def-term-set ({N:Nat}) | true .)
(add-goal ({M:MSet}) | true =>A
          ({M’:MSet}) | (M’:MSet =C M:MSet) = (tt) .)
(start-proof .)
(case 1 on M:MSet by {K:Nat} U {M1:MSet M2:MSet} .)
(step* .)

Except for N:Nat \mapsto M’:MSet, the two proofs are identical
We already saw READERS–WRITERS–stop in Lecture 23

mod READERS–WRITERS–stop is
  sorts Nat State .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  sort State .
  vars R W : Nat .
  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm

Recall the *mutual exclusion* proof we were working on earlier...
In READERS–WRITERS, by our corollary, to prove the invariant

\[ \text{Mutex} = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0) \]

holds from state \( \langle 0, 0 \rangle \), we must check:

1. \( [[\langle 0, 0 \rangle \mid \top]] \subseteq [[\text{Mutex}_1]] \)
2. \( \text{Mutex}_1 \rightarrow^\ast [[\text{Mutex}]] \)
3. \( \text{Mutex}_2 \rightarrow^\ast [[\text{Mutex}]] \)

where:

\( \text{Mutex}_1 = \langle R, W \rangle \mid W = 0 \) and
\( \text{Mutex}_2 = \langle R, W \rangle \mid W = 1 \land R = 0. \)

Now we can write our proof script
Example Proofs (IX)

load r&w.maude
load rltool.maude
(select module READERS-WRITERS-stop .)
(subsumed (< 0,0 >) | true =<
  (< R: Nat, W: Nat >) | (W: Nat) = (0) .)
(def-term-set ([R: Nat, W: Nat]) | true .)
(add-goal (< R: Nat, W: Nat >) | (W: Nat) = (0)
  =>A ([ R': Nat, W': Nat ]) | (W’: Nat) = (0) /
  ([ R': Nat, W': Nat ]) | (W’: Nat) = (s(0)) /
  (R’: Nat) = (0) .)
(add-goal (< R: Nat, W: Nat >) | (W: Nat) = (s(0)) /
  (R: Nat) = (0)
  =>A ([ R’: Nat, W’: Nat ]) | (W’: Nat) = (0) /
  ([ R’: Nat, W’: Nat ]) | (W’: Nat) = (s(0)) /
  (R’: Nat) = (0) .)
(start-proof .)
(step* .)