Program Verification: Lecture 24

José Meseguer

University of Illinois
at Urbana-Champaign
Case Analysis Rule

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

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

The following auxiliary rule allows reasoning by cases:

\[ \frac{A, C \vdash T(u | \phi) \{ x : s \mapsto u_l \} \rightarrow \star \ A \{ x : s \mapsto u_l \} \mid A, C \vdash T \ u | \phi \rightarrow \star \ A}{\text{where } x : s \in \text{vars}(u) \text{ and } \{ u_1, \ldots, u_k \} \text{ is a pattern set for } s.} \]
Case Analysis Rule

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

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

\[
[A, C] \vdash_T u \mid \varphi \rightarrow^\ast A
\]
Case Analysis Rule

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

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

The following auxiliary rule allows reasoning by cases:

**Case Analysis**

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

\[
[A, C] \vdash_T u \mid \varphi \longrightarrow^{\star} A
\]

where \( x:s \in vars(u) \) 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 $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow^\ast B$.
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \xrightarrow{\otimes} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \xrightarrow{\otimes} B$. 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 $A \xrightarrow{\ast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \xrightarrow{\ast} B$. 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*. 
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 $A \rightarrow \bowtie B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \rightarrow \bowtie B$. 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 \( A \longrightarrow^{\ast} B \), denoted \( R = (\Sigma, B, R) \models A \longrightarrow^{\ast} B \). 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
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
   ```clojure
   load rltool
   ```
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
   ```
   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 \( \mathcal{R} = (\Sigma, B, R) \) satisfies a reachability formula \( A \longrightarrow^\ast B \), denoted \( \mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^\ast B \). 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:
Suppose we want to prove that a rewrite theory $\mathcal{R} = (\Sigma, B, R)$ satisfies a reachability formula $A \longrightarrow^* B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^* B$. 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
   ```
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 F00 .)
   ```
After this you will be ready to give commands to the tool to:

1. enter goals, and
2. prove such goals.

As for other Maude tools, there is a grammar for all such commands. A first fragment is:

- **VariableName** ::= \texttt{<Special>}
- **ModuleName** ::= \texttt{<Special>}
- **Term** ::= \texttt{<Special>}
- **Atom** ::= (\texttt{Term})=(\texttt{Term}) | (\texttt{Term})=/=(\texttt{Term})
- **Conjunction** ::= true | Atom | Conjunction \& Conjunction
- **Pattern** ::= (\texttt{Term}) | Conjunction
- **PatternFormula** ::= Pattern | PatternFormula \lor PatternFormula
- **RFormula** ::= Pattern \Rightarrow A PatternFormula
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.
Reachability Logic Tool 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:

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

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

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

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 \texttt{CHOICE}, the reachability formula

\[
\{M\} \models \top \xrightarrow{*} \{M'\} \models M' \subseteq M = \text{tt}
\]

is expressed in this grammar as:

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

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

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 .)
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 .)

Let us illustrate each of these commands.
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 .)

Let us illustrate each of these commands.
Since many of the properties we will prove are *invariants* using:
Since many of the properties we will prove are \textit{invariants} using:

\textbf{Corollary}

\textit{If }$[S_0] \subseteq [B] \text{ and } B \xrightarrow{\circ} [B]$ \textit{holds in }$\mathcal{R}_{\text{stop}}$, \textit{then }$B$ \textit{is an invariant for }$\mathcal{R}$ \textit{from initial states }$S_0$.\textit{.}
Since many of the properties we will prove are \textit{invariants} using:

\textbf{Corollary}

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

to discharge the proof obligation $[S_0] \subseteq [B]$
Since 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 =< Pattern .)
Since many of the properties we will prove are *invariants* using:

**Corollary**

If \([S_0] \subseteq [B]\) and \(B \rightarrow^\ast [B]\) holds in \(\mathcal{R}_{stop}\), 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

\[ Mutex = \langle R, W \rangle \mid W = 0 \lor (W = 1 \land R = 0) \]
Since many of the properties we will prove are invariants using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\circ} [B]$ holds in $R_{\text{stop}}$, 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:
Since many of the properties we will prove are *invariants* using:

**Corollary**

If \([ S_0 ] \subseteq [ B ]\) and \(B \rightarrow^\star [ B ]\) 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) .)
Since many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\ast} [B]$ 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 $Mutex = \langle R, W \rangle | W = 0 \lor (W = 1 \land R = 0)$ requires that we first check $[[\langle 0, 0 \rangle | \top]] \subseteq [[Mutex_1]]$ 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:
Since many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\circledast} [B]$ holds in $R_{\text{stop}}$, 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 $\leftarrow$ 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 $\langle 0, 0 \rangle \mid \text{true} \leftarrow (\langle R:\text{Nat}, W:\text{Nat} \rangle) \mid (W:\text{Nat}) = (0)$.)

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
Since many of the properties we will prove are *invariants* using:

**Corollary**

If $[S_0] \subseteq [B]$ and $B \xrightarrow{\circ} [B]$ holds in $R_{\text{stop}}$, 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 $Mutex = \langle R, W \rangle | W = 0 \lor (W = 1 \land R = 0)$ requires that we first check $[[\langle 0, 0 \rangle | \top]] \subseteq [[Mutex_1]]$ 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:

$Mutex_1 = \langle R, W \rangle | W = 0$ and

$Mutex_2 = \langle R, W \rangle | W = 1 \land R = 0$. 
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 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).
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:
The set $\llbracket T \rrbracket$ of terminating states should also be specified as a pattern formula $T$. We only require $\llbracket T \rrbracket$ 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: $T = [x_1, \ldots, x_n] | \top$ as terminating states in $\mathcal{R}_{stop}$.
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: $T = [x_1, \ldots, x_n] \mid \top$ as terminating states in $\mathcal{R}_{\text{stop}}$.

For example for \textsc{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:

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

\( \rightarrow \ast \rightarrow \)
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** $A \xrightarrow{\mathcal{R}} B$ and perhaps some **auxiliary lemmas**. To enter to the tool each formula in $C$ we give the command:
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

\[
\{ M \}:|v\rightarrow^\ast \{ M' \} | M' \subseteq M = tt
\]

we give the command:

(\texttt{(add-goal (\{ M:MSet \}) | true ⇒ A (\{ M':MSet \}) | (M':MSet =C M:MSet) = (tt) .)})
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
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: (add-goal RFormula .)

For example, in CHOICE, to enter the formula

$$\{M\} | \top \rightarrow{}^\star \{M'\} | M' \subseteq M = tt$$
Recall that in general we need to prove a set $C$ of reachability formulas, including the main formula $A \rightarrow^{\circlearrowright} B$ 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

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

we give the command:
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

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

we give the command:

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

For example, in CHOICE, to enter the formula

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

we give the command:

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

The tool gives each entered goal a number.
Recall that in general we need to prove a set $C$ of reachability formulas, including the \textit{main formula} $A \rightarrow^\otimes B$ and perhaps some \textit{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

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

we give the command:

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

The tool gives each entered goal a number. It will later generate \textit{subgoals} named by \textit{number sequences} $n_1 \ldots n_k$. 
Recall that in general we need to prove a set $C$ of reachability formulas, including the *main formula* $A \xrightarrow{\circledast} B$ 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

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

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* $n_1 \ldots n_k$, 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 $A \longrightarrow^\ast B$ 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

$$\{M\} \mid \top \longrightarrow^\ast \{M'\} \mid M' \subseteq M = \text{tt}$$

we give the command:

$$(\text{add-goal } (\{M:MSet\}) \mid \text{true} \Rightarrow A \hspace{1cm} (\{M':MSet\}) \mid (M':MSet =C M:MSet) = (\text{tt}) .)$$

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

(i) checking containments of the form $J \subseteq J$ with the \texttt{(subsumed Pattern =< Pattern .)} command and (ii) adding all goals in $C$ to the tool with the \texttt{(add-goal RFormula .)} command, we can start the proof process by giving the \texttt{(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:\texttt{(step .)} (resp. \texttt{(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 \texttt{(step* .)} command. And at any time we can quit giving the \texttt{(quit .)} command.
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\preceq$ Pattern .) 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,
After: (i) checking containments of the form $[S_0] \subseteq [B]$ with the (subsumed Pattern $\subseteq$ 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 $[\mathcal{S}_0] \subseteq [\mathcal{B}]$ with the \texttt{(subsumed Pattern = Pattern .)} command and (ii) adding all goals in $\mathcal{C}$ to the tool with the \texttt{(add-goal RFormula .)} command, we can start the proof process by giving the \texttt{(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 $<=$ 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 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.
After: (i) checking containments of the form $[S_0] \subseteq [B]$ 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.
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:

```
(case GoalName on VariableName by TermSet .)
```
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:

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

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

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

\[
\text{(case 1 on M : \text{MSet} by } \{ \text{N} : \text{Nat} \} \cup \{ M_1 : \text{MSet}, M_2 : \text{MSet} \} .)
\]
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:

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

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

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

\[
(\{M:\text{MSet}\} \mid \text{true} \Rightarrow A (\{M^' : \text{MSet}\}) \mid (M^' : \text{MSet} =_{C} M : \text{MSet}) = (\text{tt})
\]

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:
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

```plaintext
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] .
vary U V : MSet . var N : Nat .
eq U =C U = tt .
eq U =C U V = tt .
rl [choice] : {U V} => {U} .
endm
```
Also recall the Hoare Triple from Lecture 23:

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

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

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

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

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

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

The case analysis occurs on variable \( M : MSet \);
Two cases: \( M : MSet \mapsto N : \text{Nat} \) (or) \( M : MSet \mapsto M_1 : MSet \) \( M_2 : MSet \)

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:

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:

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

Expressible in the tool notation as:

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

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

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
  protecting NAT .
  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

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

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

1. \( \llbracket \langle 0, 0 \rangle \mid \top \rrbracket \subseteq \llbracket Mutex_1 \rrbracket \)
2. \( Mutex_1 \xrightarrow{\ast} [Mutex] \)
3. \( Mutex_2 \xrightarrow{\ast} [Mutex] \)

where:

\( Mutex_1 = \langle R, W \rangle \mid W = 0 \) and
\( Mutex_2 = \langle R, W \rangle \mid W = 1 \land R = 0. \)

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) = (0)
  => A ([ R:Nat,W:Nat ]) | (W) = (0) \/
  ([ R:Nat,W:Nat ]) | (W) = (1) \/ (R) = (0) .)
(add-goal (< R:Nat,W:Nat >) | (W) = (1) \/ (R) = (0)
  => A ([ R:Nat,W:Nat ]) | (W) = (0) \/
  ([ R:Nat,W:Nat ]) | (W) = (1) \/ (R) = (0) .)
(start-proof .)
(step* .)