Program Verification: Lecture 24

José Meseguer

University of Illinois
at Urbana-Champaign

Case Analysis Rule

Case Analysis Rule

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

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.

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

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

Case Analysis Rule

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

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

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

where $x{:}s \in vars(u)$ and $\{u_1, \ldots, u_k\}$ is a pattern set for s.

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$.

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. How can we do it?

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. How can we do it?

The inference rules of reachability logic have been implemented in Maude as a new tool:

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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*.

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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:

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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`

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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.

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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).

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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:

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 $A \longrightarrow^{\circledast} B$, denoted $\mathcal{R} = (\Sigma, B, R) \models A \longrightarrow^{\circledast} B$. 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 .)`

Reachability Logic Tool Commands

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: (i) enter goals, and (ii) prove such goals.

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.

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:

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:

```
VariableName   ::= <Special>
ModuleName     ::= <Special>
Term           ::= <Special>
Atom           ::= (Term)=(Term)
                 | (Term)=/=(Term)
Conjunction    ::= true
                 | Atom
                 | Conjunction /\ Conjunction
Pattern        ::= (Term) "|" Conjunction
PatternFormula ::= Pattern
                 | PatternFormula \/ PatternFormula
RFormula       ::= Pattern =>A PatternFormula
```

Reachability Logic Tool Commands (II)

For example, for `CHOICE`, the reachability formula

Reachability Logic Tool Commands (II)

For example, for `CHOICE`, the reachability formula

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

Reachability Logic Tool Commands (II)

For example, for `CHOICE`, the reachability formula

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

is expressed in this grammar as:

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

Reachability Logic Tool Commands (II)

For example, for `CHOICE`, the reachability formula

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

is expressed in this grammar as:

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

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.

Reachability Logic Tool Commands (IV)

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

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *holds in* $\mathcal{R}_{stop}$*, then* B *is an invariant for* $\mathcal{R}$ *from initial states* S_0*.*

Reachability Logic Tool Commands (IV)

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

Corollary

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

To discharge the proof obligation $[\![S_0]\!] \subseteq [\![B]\!]$

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *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 .)`

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *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 $Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *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 $Mutex = \langle R, W\rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$ requires that we first check $[\![\langle 0, 0\rangle \mid \top]\!] \subseteq [\![Mutex_1]\!]$ by giving the command:

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ and $B \longrightarrow^{\circledast} [B]$ 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 $Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$ 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) .)
```

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ and $B \longrightarrow^{\circledast} [B]$ 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 $Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$ 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 $B \longrightarrow^{\circledast} [B]$ 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 $Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$ 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

Reachability Logic Tool Commands (IV)

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

Corollary

If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *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 $Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$ 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
$Mutex_2 = \langle R, W \rangle \mid W = 1 \wedge R = 0$.

Reachability Logic Tool Commands (V)

The set $[\![T]\!]$ of *terminating states* should also be specified as a *pattern formula* T.

Reachability Logic Tool Commands (V)

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.

Reachability Logic Tool Commands (V)

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).

Reachability Logic Tool Commands (V)

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:

Reachability Logic Tool Commands (V)

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 $\mathcal{R}_{stop}$.

Reachability Logic Tool Commands (V)

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:

Reachability Logic Tool Commands (V)

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 .)
```

Reachability Logic Tool Commands (VI)

Recall that in general we need to prove a set $\mathcal{C}$ of reachability formulas,

Reachability Logic Tool Commands (VI)

Recall that in general we need to prove a set $\mathcal{C}$ of reachability formulas, including the *main formula* $A \longrightarrow^{\circledast} B$ and perhaps some *auxiliary lemmas*.

Reachability Logic Tool Commands (VI)

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

Reachability Logic Tool Commands (VI)

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

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

we give the command:

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

we give the command:

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

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

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.

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

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* $n_1 \dots n_k$,

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

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

Reachability Logic Tool Commands (VI)

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

For example, in `CHOICE`, to enter the formula

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

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.

Reachability Logic Tool Commands (VII)

After:

Reachability Logic Tool Commands (VII)

After: (i) checking containments of the form $[\![S_0]\!] \subseteq [\![B]\!]$ with the `(subsumed Pattern =< Pattern .)` command

Reachability Logic Tool Commands (VII)

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,

Reachability Logic Tool Commands (VII)

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.

Reachability Logic Tool Commands (VII)

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:

Reachability Logic Tool Commands (VII)

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: `(step .)` (resp. `(step n .)`).

Reachability Logic Tool Commands (VII)

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: `(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.

Reachability Logic Tool Commands (VII)

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: `(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.

Reachability Logic Tool Commands (VIII)

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:

Reachability Logic Tool Commands (VIII)

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 .)
```

Reachability Logic Tool Commands (VIII)

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 .)
```

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

Reachability Logic Tool Commands (VIII)

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 .)
```

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

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

Reachability Logic Tool Commands (VIII)

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 .)
```

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

```
({M:MSet}) | true =>A ({M':MSet}) | (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:

Reachability Logic Tool Commands (VIII)

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 .)
```

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

```
({M:MSet}) | true =>A ({M':MSet}) | (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:

```
(case 1 on M:MSet by {N:Nat} U {M1:MSet M2:MSet} .)
```

Example Proofs (I)

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:

$$\{\{M\} \mid \top\} \ \texttt{CHOICE} \ \{\{N\} \mid N \subseteq M = tt\}$$

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

```
({M:MSet}) | true =>A
       ({N:Nat}) | (N:Nat =C M:MSet) = (tt)
```

Sometimes, we *cannot* prove a goal as-is and must analyze cases; this formula is one such example

Example Proops (III)

```
({M:MSet}) | true =>A
        ({N:Nat}) | (N:Nat =C M:MSet) = (tt)
```

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

Recall any terminating state in this theory has the form `{N:Nat}`

Now we are ready to prove this example in the tool

Example Proofs (IV)

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*

Example Proofs (V)

Q: Does the system handle general reachability formulas as nicely?

A: Let us illustrate by example...

Recall the `CHOICE` reachability formula from Lecture 23:

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

Expressible in the tool notation as:

```
({M:MSet}) | true =>A
     ({M':Nat}) | (M':Nat =C M:MSet) = (tt)
```

We expect the proof will be similar to its Hoare Triple cousin...

Example Proofs (VI)

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*

Example Proofs (VII)

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 .
  op <_,_> : Nat Nat -> State [ctor] .
  op [_,_] : Nat Nat -> State [ctor] .
  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 > .
  rl < R, W >    => [R,W] .
endm
```

Recall the *mutual exclusion* proof we were working on earlier...

Example Proofs (VIII)

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. $[\![\langle 0, 0 \rangle \mid \top]\!] \subseteq [\![Mutex_1]\!]$
2. $Mutex_1 \longrightarrow^{\circledast} [Mutex]$
3. $Mutex_2 \longrightarrow^{\circledast} [Mutex]$

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: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* .)
```