

Program Verification: Lecture 22

José Meseguer and Stephen Skeirik

University of Illinois
at Urbana-Champaign

Case Analysis Rule

Call $\{u_1, \dots, u_k\} \subseteq T_\Omega(X)_s$ 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

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

where $x:s \in vars(u)$ and $\{u_1, \dots, u_k\}$ 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 $A \longrightarrow^* B$, 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 From 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 .)`

Reachability Logic Tool Commands

Reachability Formula Syntax

Now you will be ready to give commands to: (i) enter goals and (ii) prove goals. As with all Maude tools, there is an associated command grammar. Here is the syntax for reachability formulas:

```
Atom                ::= (Term)=(Term)
                   | (Term)≠(Term)
Conjunction          ::= true
                   | Atom
                   | Conjunction /\ Conjunction
Pattern              ::= (Term) "|" Conjunction
PatternFormula       ::= Pattern
                   | PatternFormula \/ PatternFormula
RFormula             ::= Pattern => PatternFormula
```

Reachability Logic Tool Commands (II)

Reachability Formula Syntax

For example, for CHOICE, the reachability formula

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

is expressed in this grammar as:

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

We can now give commands according to the following grammar:

Reachability Logic Tool Commands (III)

Command Syntax

```
TermSet ::= (Term) | TermSet U TermSet
VariableSet ::= (Variable) | VariableSet U VariableSet
Command ::= (select ModuleName .)
| (use tool ToolName for ProblemName on Modulename .)
| (def-term-set PatternFormula .)
| (subsumed Pattern =< Pattern .)
| (add-goal Label : RFormula .)
| (inv Label to Operator with VariableSet on PatternFormula .)
| (inv Label to Operator on PatternFormula)
| (start-proof .)
| (auto .)
| (auto Nat .)
| (auto* .)
| (case Nat on VariableName by TermSet .)
| (on Nat use strat Labellist .)
| (quit .)
```

Q: How do we use these commands in a proof?

Reachability Logic Tool Commands (IV)

Command Syntax

Q: How do we use these commands in a proof?

A: The general process has the following steps:

- 1 Select the appropriate backend solver (for SMT reasoning)
- 2 Define the set of terminating states to be used (using extended theory \mathcal{R}_{stop} when reasoning about invariants)
- 3 Perform any subsumption checks (only needed for invariants)
- 4 Add goals, auxiliary lemmas, and/or invariants to be proved
- 5 Start the proof
- 6 Apply tactics to complete the proof

We will illustrate the process above through two examples.

Reachability Proof Example (I)

Suppose we will *not* prove any invariants.

```
mod CHOICE is --- from Lecture 21
  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
```


Reachability Proof Example (II)

According to the procedure outlined above, the first step is to select our SMT solver backend by using the `use tool` command. We need to provide a:

- 1 ToolName – use magic constant: `varsat`
- 2 ProblemName – use magic constant: `validity`
- 3 ModuleName – replace with module name, i.e. `CHOICE`

Thus for the module `CHOICE`, we give command:

```
(use tool varsat for validity on CHOICE .)
```

Reachability Proof Example (III)

Then, the next step is to define our set of terminating states $\llbracket T \rrbracket$ as *pattern formula* T .

For the theory CHOICE, we can specify T by giving the command:

```
(def-term-set ({N:Nat}) | true .)
```

Q: How do we know we selected the correct set $\llbracket T \rrbracket$?

A: Currently, this property must be manually checked by the user. Here we see the rule [choice] non-deterministically removes elements from the state whenever there are two or more elements.

Reachability Proof Example (IV)

Next we need to enter our goals into tool including the *main formula* $A \longrightarrow^* B$ and perhaps some *auxiliary lemmas*. To enter to the tool each formula in \mathcal{C} we give the command: (add-goal Label : RFormula .)

For example, in CHOICE, to enter the formula

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

we give the command:

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

The tool gives each generated goal a unique number by sequentially incrementing a counter.

Reachability Proof Example (V)

At this point, 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: `(auto .)` (resp. `(auto n .)`).

Instead, if we want to go to the end of the proof process in the hope that it will terminate we give the `(auto* .)` command. And at any time we can quit giving the `(quit .)` command.

Reachability Proof Example (VI)

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 Nat on VariableName by TermSet .)
```

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

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

Reachability Proof Example (VII)

Putting it all together, we can complete the proof using the following script:

```
load choice.mauve
load rltool.mauve
(select CHOICE .)
(use tool varsat for validity on CHOICE .)
(def-term-set ({N:Nat}) | true .)
(add-goal reaches-subset : ({M:MSet}) | true =>
    ({M':MSet}) | (M':MSet =C M:MSet) = (tt) .)
(start-proof .)
(case 1 on M:MSet by (N:Nat) U (M1:MSet M2:MSet) .)
(auto* .)
```

Reachability Proof Example: Alternative Proof

Note $\llbracket \{M\} \mid \top \rrbracket \subseteq \llbracket \{M'\} \mid M' \subseteq M = tt \rrbracket$ holds!

Because with substitution $\{M' \mapsto M\}$ we get $M \subseteq M = tt$, which is equivalent to \top .

Thus we can use the simpler alternative proof script below.

```
load choice.maude
load rltool.maude
(select CHOICE .)
(use tool varsat for validity on CHOICE .)
(def-term-set ({N:Nat}) | true .)
(add-goal reaches-subset : ({M:MSet}) | true =>
    ({M':MSet}) | (M':MSet =C M:MSet) = (tt) .)
(start-proof .)
(auto .)
```

Invariant Proof Example (I)

Recall when proving *invariants*, we need the following result:

Corollary

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

This introduces three new requirements we need to handle:

- we need to define the theory \mathcal{R}_{stop}
- we need to relativize our proof to use terminating states defined by the new operator $[-]$ in \mathcal{R}_{stop}
- we need to perform a subsumption check $\llbracket S_0 \rrbracket \subseteq \llbracket B \rrbracket$

Invariant Proof Example (II)

We already saw READERS-WRITERS in Lecture 21.

```
mod READERS-WRITERS is
  protecting NAT .
  sort State .
  op <_,_> : Nat Nat -> State [ctor] . --- readers/writers
  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
```

Invariant Proof Example (III)

To prove invariants over a non-terminating theory like READERS-WRITERS, we first define its extension READERS-WRITERS-stop:

```
mod READERS-WRITERS-stop is protecting READERS-WRITERS .  
  op [_,_] : Nat Nat -> State .  
  var R W : Nat .  
  rl [stop] : < R, W > => [ R, W ] .  
endm
```

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

Invariant Example Proof (IV)

In READERS-WRITERS, by our corollary, to prove the invariant

$$Mutex = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$$

holds from state $\langle 0, 0 \rangle$, we must check in READERS-WRITERS-stop:

$$1 \quad \llbracket \langle 0, 0 \rangle \mid \top \rrbracket \subseteq \llbracket Mutex_1 \rrbracket$$

$$2 \quad Mutex_1 \longrightarrow^* [Mutex]\sigma$$

$$3 \quad Mutex_2 \longrightarrow^* [Mutex]\sigma$$

where:

$$Mutex_1 = \langle R, W \rangle \mid W = 0$$

$$Mutex_2 = \langle R, W \rangle \mid W = 1 \wedge R = 0$$

Substitution σ is a renaming of $Mutex$.

Invariant Proof Example (V)

Next the set $\llbracket T \rrbracket$ of *terminating states* should also be specified as a *pattern formula* T defined using the new operator $\llbracket _ \rrbracket$.

This is allowed because $\llbracket T \rrbracket$ need only be *contained* in, or equal to, the set of *all* terminating states. Thus, we perform 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:
 $T = [x_1, \dots, x_n] \mid \top$ as terminating states in \mathcal{R}_{stop} .

For example for READERS-WRITERS-stop, we specify T by:

```
(def-term-set ([R:Nat,W:Nat]) | true .)
```

Invariant Proof Example (VI)

We next need to perform a subsumption check.

If our invariant $B = \textit{Mutex}$ where:

$$\textit{Mutex} = \langle R, W \rangle \mid W = 0 \vee (W = 1 \wedge R = 0)$$

and our initial state $S_0 = \langle 0, 0 \rangle \mid \top$

We discharge the proof obligation $\llbracket S_0 \rrbracket \subseteq \llbracket B \rrbracket$ by using the command (subsumed Pattern =< Pattern .)

For example, in READERS-WRITERS-stop, we could write:

```
(subsumed (< 0,0 >) | true =<
  (< R:Nat,W:Nat >) | (W:Nat) = (0) \/  
  (< R:Nat,W:Nat >) | (W:Nat) = (1) /\ (R:Nat)=(0) .)
```

Invariant Proof Example (VII)

After: (i) checking containments of the form $\llbracket S_0 \rrbracket \subseteq \llbracket B \rrbracket$ with the `(subsumed Pattern =< Pattern .)` command and (ii) adding all goals in \mathcal{C} to the tool with the `(add-goal Label : RFormula .)` command, we can start the proof process by giving the `(start-proof .)` command.

At this point, we can make use of the `auto`, `case`, and `quit` commands exactly as shown before in the CHOICE example.

Let's put it all together.

(Don't forget to select our backend SMT solver!)

Invariant Proof Example (VIII)

```
load rw.mauve
load rltool.mauve
(select module READERS-WRITERS-stop .)
(use tool varsat for validity on READERS-WRITERS-stop .)
(subsumed (< 0,0 >) | true =<
  (< R:Nat,W:Nat >) | (W:Nat) = (0) \ /
  (< R:Nat,W:Nat >) | (W:Nat) = (1) /\ (R:Nat) = (0) .)
(def-term-set ([R:Nat,W:Nat]) | true .)
(add-goal mutex#1 : (< R:Nat,W:Nat >) | (W:Nat) = (0)
=> ([ R':Nat,W':Nat ]) | (W':Nat) = (0) \ /
  ([ R':Nat,W':Nat ]) | (W':Nat) = (1) /\ (R':Nat) = (0) .)
(add-goal mutex#2 : (< R:Nat,W:Nat >) | (W:Nat) = (1) /\ (R:Nat) = (0)
=> ([ R':Nat,W':Nat ]) | (W':Nat) = (0) \ /
  ([ R':Nat,W':Nat ]) | (W':Nat) = (1) /\ (R':Nat) = (0) .)
(start-proof .)
(on 1 use strat mutex#1 mutex#2 .)
(on 2 use strat mutex#1 mutex#2 .)
(auto* .)
```

Invariant Proof Example (IX)

Q: We saw these extra commands above. What do they mean?

```
(on 1 use strat mutex#1 mutex#2 .)
(on 2 use strat mutex#1 mutex#2 .)
```

A: These commands instruct the tool, on all descendants from some goal named n , use the labelled initial goals as potential axioms. By default, given an `add-goal` command with label L , only L is used as an axiom to prove goals that are descendants of the goal labelled L corresponding to the `add-goal` command.

Q: The proof above is a little verbose. Can we do better?

A: Yes, by using the `inv` command as shown below.

Invariant Proof Example (X)

```
load rw.maude
load rltool.maude
(select module READERS-WRITERS-stop .)
(use tool varsat for validity on READERS-WRITERS-stop .)
(subsumed (< 0,0 >) | true =<
  (< R:Nat,W:Nat > | (W:Nat) = (0) \\/
  (< R:Nat,W:Nat > | (W:Nat) = (1) /\ (R:Nat) = (0) .)
(def-term-set ([R:Nat,W:Nat]) | true .)
(inv mutex to '[_'] on
  (< R:Nat,W:Nat > | (W:Nat) = (0) \\/
  (< R:Nat,W:Nat > | (W:Nat) = (1) /\ (R:Nat) = (0) .)
(on 1 use strat mutex#1 mutex#2 .)
(on 2 use strat mutex#1 mutex#2 .)
(auto* .)
```

Here, we have to tell inv:

- a which operator is our “stopwatch”;
- b the pattern formula representing our invariant;
- c all variables in the pattern which are *parameters* (if any is a parameter) using the optional with `VariableSet` argument (not used here: unparametric).

Invariant Proof Example (XI)

Advantages of the invariant command:

- 1 we do not need to perform easy-to-mess-up variable renaming;
- 2 we do not need to repeat common pieces of goals;
- 3 generated goal labels are automatically made unique by appending # n for the n^{th} generated goal.