Program Verification: Lecture 24

José Meseguer and Stephen Skeirik

University of Illinois
at Urbana-Champaign

Case Analysis Rule

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

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

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

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

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 =>A PatternFormula
```

Reachability Logic Tool Commands (II)

Reachability Formula Syntax

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)

Command Syntax

```
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 .)
```

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

We will illustrate the process above through two examples.

Reachability Proof Example (I)

Recall the CHOICE module from Lecture 23, and suppose we do *not* wish to prove an invariant.

```
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
```

Reachability Proof Example (II)

According to the procedure outlined above, the first step is define our set of terminating states $[\![T]\!]$ 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 $[\![T]\!]$?

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 (III)

Next we need to enter our goals into tool 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 Proof Example (IV)

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: `(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 Proof Example (V)

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} .)
```

Reachability Proof Example (VI)

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

```
load choice.maude
load rltool.maude
(select 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 {N:Nat} U {M1:MSet M2:MSet} .)
(step* .)
```

Invariant Proof Example (I)

Recall 2TOKEN which we covered when discussing model checking:

```
mod 2TOKEN is
  sorts Name Proc Token Conf State .
  subsorts Proc Token < Conf .
  op {_}       : Conf      -> State [ctor] .
  op none      :           -> Conf  [ctor] .
  ops * $      :           -> Token [ctor] .
  ops a b      :           -> Name  [ctor] .
  op __        : Conf Conf -> Conf  [assoc comm id: none] .
  op [_,wait] : Name      -> Proc  [ctor] .
  op [_,crit] : Name      -> Proc  [ctor] .
  var C : Conf .
  rl [a-enter] : { $ [a,wait] C } => { [a,crit]   C } .
  rl [b-enter] : { * [b,wait] C } => { [b,crit]   C } .
  rl [a-exit]  : {   [a,crit] C } => { [a,wait] * C } .
  rl [b-exit]  : {   [b,crit] C } => { [b,wait] $ C } .
endm
```

Invariant Proof Example (II)

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

Corollary

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

Invariant Proof Example (III)

To prove invariants over a non-terminating theory like `2TOKEN`, we first define its extension `2TOKEN-stop`:

```
mod 2TOKEN-stop is protecting 2TOKEN .
  op [_] : Conf -> State .
  var C  : Conf .
  rl [stop] : { C } => [ C ] .
endm
```

One invariant we might like to prove is *mutual exclusion*, i.e. that only one process is critical at any moment. This can be specified by:

$$\begin{aligned} Mutex = & \{[a, wait]\ [b, wait]\ T{:}Token\} \mid \top \\ & \vee \{[a, wait]\ [b, crit]\} \mid \top \\ & \vee \{[a, crit]\ [b, wait]\} \mid \top \end{aligned}$$

Invariant Proof Example (IV)

We next need to perform a subsumption check.

If our invariant $B = Mutex$ where:

$$\begin{aligned} Mutex = &\{[a, wait]\ [b, wait]\ T{:}Token\} \mid \top \\ &\vee \{[a, wait]\ [b, crit]\} \mid \top \\ &\vee \{[a, crit]\ [b, wait]\} \mid \top \end{aligned}$$

and our initial state $S_0 = \{[a, wait]\ [b, wait]\ T{:}Token\} \mid \top$

We discharge the proof obligation $[\![S_0]\!] \subseteq [\![B]\!]$ by using the command `(subsumed Pattern =< Pattern .)`

For example, in `2TOKEN-stop`, we would write:

```
(subsumed ({ [a,wait] [b,wait] T:Token  }) | true =<
  ({ [a,wait] [b,wait] T':Token }) | true \/
  ({ [a,crit] [b,wait]          }) | true \/
  ({ [a,wait] [b,crit]          }) | true .)
```

Invariant Proof Example (III)

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

This is allowed because $[\![T]\!]$ 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: $T = [x_1, \ldots, x_n] \mid \top$ as terminating states in $\mathcal{R}_{stop}$.

For example for `2TOKEN-stop`, we specify T by:

```
(def-term-set ([C:Conf]) | true .)
```

Invariant Proof Example (V)

The next step in our procedure is to add any goals to be proved.
In the case of invariants, our goals are given by the corollary above.
In the case of `2TOKEN-stop`, our invariant generates three goals:

```
(add-goal ({ [a,wait] [b,wait] T:Token  }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
(add-goal ({ [a,crit] [b,wait] }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
(add-goal ({ [a,wait] [b,crit] }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
```

Invariant Proof Example (VI)

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.

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

Let's put it all together.

Invariant Proof Example (VII)

```
load token.maude
load rltool.maude
(select 2TOKEN-stop .)
(def-term-set ([C:Conf]) | true .)
(subsumed ({ [a,wait] [b,wait] T:Token  }) | true =<
  ({ [a,wait] [b,wait] T':Token }) | true \/
  ({ [a,crit] [b,wait]          }) | true \/
  ({ [a,wait] [b,crit]          }) | true .)
(add-goal ({ [a,wait] [b,wait] T:Token  }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
(add-goal ({ [a,crit] [b,wait] }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
(add-goal ({ [a,wait] [b,crit] }) =>A
  ([ [a,wait] [b,wait] T':Token ]) | true \/
  ([ [a,crit] [b,wait]          ]) | true \/
  ([ [a,wait] [b,crit]          ]) | true .)
(start-proof .)
(step* .)
```