Lecture 2: Deductive Verification of Reactive Systems:

- Deductive Invariance Proofs

- Prototype Verification System (PVS)

Cristina Seceleanu
MRTC, MdH

E-mail: cristina.seceleanu@mdh.se
Deductive Invariance Proofs

• Safety property:
  • property that can be specified by: [] p (LTL)

• The properties above: invariance properties (of model Γ)

• Given a transition system (discrete or timed), we denote:

  • \( V \) – A finite set of typed state variables. A \( V \)-state \( s \) is an interpretation of \( V \). \( \Sigma_V \) – the set of all \( V \)-states.

  • \( \emptyset \subseteq V \) – A set of observable variables.

  • \( \Theta \) – An initial condition. A satisfiable assertion that characterizes the initial states.

  • \( \rho \) – A transition relation. An assertion \( \rho(V, V') \), referring to both unprimed (current) and primed (next) versions of the state variables. For example, \( x' = x + 1 \) corresponds to the assignment \( x := x + 1 \).
Basic Invariance Proofs

• The simplest rule for proving invariance properties: Basic INV
  • establishes \( p \) as an invariant of some model \( P \).

Rule Basic INV

I1. \( \Theta \Rightarrow p \)
I2. \( p \land \rho_\tau \Rightarrow p' \quad (\{p\} \tau \{p\}) \)

\[ \square p \]

An assertion/predicate \( p \) satisfying I1 and I2 is called inductive.

• Premise I1 (proof obligation/verification condition):
  requires the initial condition \( \Theta \) to imply property \( p \).

• Premise I2 (proof obligation/verification condition):
  requires that all transitions in \( P \) preserve \( p \).
GRC example: Modeling the Trains

- Trains component: Timed Automaton with pre-post conditions
- r: variables ranging over trains

Diagram:

- **notHere**
  - \{Pre_Enter \( P(r) \)\}
  - \{Post_Exit (r)\}
- **P**
  - \{Post_Enter P(r)\}
  - \{Pre_Enter I(r)\}
- **I**
  - \{Post_Enter I(r)\}
  - \{Pre_Exit (r)\}

Arrows indicate transitions:
- **Enter \( P(r) \)**
- **Exit \( r \)**
GRC example: Modeling the Trains

State:

\[ \text{now}, \text{ a nonnegative real, initially } 0 \]

for each train \( r \):

\[ r.\text{status} \in \{ \text{not-here, P, I} \}, \text{ initially not-here} \]

\[ \text{first}(\text{enterI}(r)), \text{ a nonnegative real, initially } 0 \]

\[ \text{last}(\text{enterI}(r)), \text{ a nonnegative real or } \infty, \text{ initially } \infty \]

Transitions:

\[ \text{enterI}(r) \rightarrow \text{P}(r) \]

Precondition:

\[ s. r.\text{status} = \text{not-here} \]

Effect:

\[ s'. r.\text{status} = P \]

\[ s'. \text{first}(\text{enterI}(r)) = \text{now} + \epsilon_1 \]

\[ s'. \text{last}(\text{enterI}(r)) = \text{now} + \epsilon_2 \]

\[ \text{exit}(r) \]

Precondition:

\[ s. r.\text{status} = I \]

Effect:

\[ s'. r.\text{status} = \text{not-here} \]

\[ \nu(\Delta t) \]

Precondition:

for all \( r \), \( s.\text{now} + \Delta t \leq s.\text{last}(\text{enterI}(r)) \)

Effect:

\[ s'. \text{now} = s.\text{now} + \Delta t \]
Example: Invariance Proof

• Inv = ( \( \forall r . r.status = P \Rightarrow ( \text{first(Enter I(r))} + \varepsilon_2 - \varepsilon_1 = \text{last(Enter I(r))} ) \) )

• Apply rule \((\Rightarrow 2 \lor)\)

\[ \forall r. \]
\[ r.status=P \Rightarrow \text{first(enter I(r))} + \varepsilon_1 - \varepsilon_2 = \text{last(enter I(r))} \]

\[ \Leftrightarrow \{ \text{rule } "\Rightarrow 2 \lor" \} \]

\[ \neg (r.status=P) \lor (\text{first(enter I(r))} + \varepsilon_2 - \varepsilon_1 = \text{last(enter I(r))}) \]

• Apply Basic INV
Example: Invariance Proof

Prove I1: initial condition

At $t = 0$:

$$\forall r. \quad \text{now} = 0 \land r.\text{status} = \text{not-here} \land \text{first(enter I(r))} = 0 \land \text{last(enter I(r))}$$

$$\Rightarrow \{ \land \text{elimination, } \lor \text{ introduction} \}$$

$$\neg (r.\text{status}=P) \lor (\text{first(enter I(r))} + \epsilon 2 - \epsilon 1 = \text{last(enter I(r))})$$

$T$

Next, to prove:

$$p \land p_\tau \rightarrow p'$$
• We assume the claim is true at $t = n$, for all $s$:

$$p = \neg (s.r.status = P) \lor (s.first(enter I(r)) + \varepsilon_2 - \varepsilon_1 = s.last(enter I(r)))$$

and we have to prove it for $t = n+1$:

$$p' = \neg (s'.r.status = P) \lor (s'.first(enter I(r)) + \varepsilon_2 - \varepsilon_1 = s'.last(enter I(r)))$$

• For transition $\text{enter P}(r)$ - we assume true:

$$\rho_\tau =
\begin{align*}
& s'.r.status = P \\
& s'.first(enter I(r)) = now + \varepsilon_1 \\
& s'.last(enter I(r)) = now + \varepsilon_2
\end{align*}$$

Out claim becomes:

$$\neg(P = P) \lor (now + \varepsilon_1 + \varepsilon_2 - \varepsilon_1 = now + \varepsilon_2)$$

$$\iff \begin{cases} \text{simplification} \\ \neg T \lor T \\ \iff \begin{cases} \text{LEM} \\ \text{TRUE} \end{cases} \end{cases}$$
• For transition enter \( I(r) \):

\[
\begin{align*}
& s'.r\text{.status} = I \\
& s'.\text{first}(\text{enter } I(r)) = 0 \\
& s'.\text{last}(\text{enter } I(r)) = \infty
\end{align*}
\]

Out claim becomes:

\[
\neg (I = P) \lor (0 + \varepsilon 2 - \varepsilon 1 = \infty)
\]

\[
\iff \text{[simplification]}
\]

\[
\neg F \lor (\ldots)
\]

\[
\iff \text{[negation, dominance of T]}
\]

\[
\text{TRUE.}
\]
More General Invariance Proofs

• Not all invariants are inductive

For example, the claim

The sum $1 + 3 + 5 + \cdots + (2k - 1)$ is a perfect square

or, more mathematically

$$p : \exists u : 1 + 3 + 5 + \cdots + (2k - 1) = u^2$$

cannot be proven by induction, using $p$ as the induction hypothesis.

To overcome this difficulty, one often has to come up with a strengthening of $p$, being an assertion $\varphi$ which implies $p$ and is inductive. For the above example, this can be

$$\varphi : 1 + 3 + 5 + \cdots + (2k - 1) = k^2$$
More General Invariance Proofs

Rule More General INV

I1. \( \Theta \Rightarrow \varphi \)
I2. \( \varphi \land \rho_\tau \Rightarrow \varphi' \) \( (\{\varphi\} \tau \{\varphi\}) \)
I3. \( \varphi \Rightarrow \rho \)

\[ \begin{align*}
\varnothing \rho & \\
\end{align*} \]

By rule Basic INV, \( \varphi \) is an invariant of the system.

All reachable states satisfy \( \varphi \).

Since, by I3, \( \varphi \Rightarrow \rho \), it follows that \( \rho \) is also an invariant.
A Brief Intro to PVS: Basic concepts

• PVS: Prototype Verification system: theorem-prover
  (http://pvs.csl.sri.com/)

• a specification language, a theorem prover, and much more ...

• the PVS prover is an interactive theorem prover with built-in semi-decision procedures

• relatively easy to plug in new proof strategies and decision procedures
• written in LISP

• PVS system guide

• PVS language

• PVS prover guide
PVS Specification Language

- PVS specification – built in HOL

- Variables and constants have types
  - Inbuilt: nat, int, real, bool
  - User defined, also uninterpreted (TYPE)
  - x: nat (constant)
  - y: VAR nat (variable)

- Declaring types.
  - Tuples: T1: TYPE [nat, nat]
  - Records: T2: TYPE [# first, second : nat #]
  - Function type: A1: TYPE [upto(10) -> T1]
PVS Specification Language

- Basic logical constructs:
  - OR (also written as \( \lor \) )
  - AND (also written & , \( \land \) )
  - NOT ( \( \neg \) )
  - EQUALITY ( = ) , INEQUALITY ( /= )
  - IMPLIES ( => )
  - IFF ( <=> )

- Universal quantifier: FORALL
- Existential quantifier: EXISTS
Proofs in PVS

• user interacts with PVS to construct a proof tree.

• each node of the tree is a proof goal.

• parent goal follows from the children by means of a proof step.

• tree is complete when all leaves are true.
Proofs in PVS (cont’d)

- A proof goal is a sequent (a sequence of formulas)
- A sequent S is represented as

  \{\text{-1}\} A1
  \{\text{-2}\} A2
  [\text{-3}] A3 ...

  |-- -- --

  \{\text{1}\} B1
  \{\text{2}\} B2
  \{\text{3}\} B3 ....

- A1, A2, A3 ... are called antecedents; B1,B2,B3... are consequents.
- Interpretation: A1 \land A2 \land A3 \land ... \Rightarrow B1 \lor B2 \lor B3 \lor ...
Proofs in Sequent Calculus

Proofs are done by transforming the sequent until one of the following forms is obtained:

\[
\begin{align*}
\vdots \\
\phi \\
\phi \\
\vdots \\
\end{align*}
\]

i.e. \( \Gamma, \phi \vdash \phi \lor \cdots \)

which is a case of Rule Premise and \( \lor \text{i} \)

\[
\begin{align*}
\vdots \\
\vdots \\
\vdash \\
\vdots \\
\end{align*}
\]

i.e. \( \Gamma \vdash T \lor \cdots \)

which is a case of Dominance of \( T \)

\[
\begin{align*}
\vdots \\
\bot \\
\vdots \\
\end{align*}
\]

i.e. \( \Gamma, \bot \vdash \cdots \)

Which is a case of \( \bot \text{i} \).
Check Validity of Arguments in PVS

• Check if the sequent is a valid argument – prove the theorem:

\[ V1: \text{THEOREM } A_1 \land \ldots \land A_n \text{ IMPLIES } B_1 \lor \ldots \lor B_n \]

• Check consistency of premises – prove the theorem:

\[ V2: \text{THEOREM } A_1 \land \ldots \land A_n \text{ IMPLIES } \text{FALSE} \]
\[ \iff \{ \text{why?} \} \]

\[ V2: \text{THEOREM } \neg A_1 \land \ldots \land A_n \]
Basics notions

Specification files: text files containing theories. Include system definitions and lemmas. Extension .pvs.

Proof files save proofs that have been composed. Extension .prf.

Context: Set of specification and proof files in one directory.

Interface: Emacs editor.
PVS prover commands

- primitive rules
  - propositional rules
  - quantifier rules
  - equality rules
  - structural rules
  - control rules
  - others: using lemmas, induction, extensionality, decision procedures
PVS prover commands

- primitive rules
  - propositional rules
  - quantifier rules
  - equality rules
  - structural rules
  - control rules
  - others: using lemmas, induction, extensionality, decision procedures
- commands and keywords for combining primitive rules into strategies (not covered in this lecture)
Propositional rules

\[
\frac{\Gamma \vdash \Delta, p \lor q}{\Gamma \vdash \Delta, p, q} \quad \frac{\Gamma, p \land q \vdash \Delta}{\Gamma, p, q \vdash \Delta}
\]

\[
\frac{\Gamma, \neg p \vdash \Delta}{\Gamma \vdash \Delta, p} \quad \frac{\Gamma \vdash \Delta, \neg p}{\Gamma, p \vdash \Delta}
\]

Correspond to the PVS flatten command.

\[
\frac{\Gamma, p \lor q \vdash \Delta}{\Gamma, p \vdash \Delta} \quad \frac{\Gamma \vdash \Delta, p \land q}{\Gamma \vdash \Delta, p} \quad \frac{\Gamma \vdash \Delta, p \land q}{\Gamma \vdash \Delta, q}
\]

These expanding rules correspond to the PVS split command.
Quantifier Rules

**Skolemization** (skolem!, skosimp*)

Requires that $t$ be a **new constant** that does not occur in the sequent

$$
\Gamma, (\exists x : p) \vdash \Delta \\
\frac{\Gamma, p\{x \leftarrow t\} \vdash \Delta}{\Gamma, p\{x \leftarrow t\} \vdash \Delta}
$$

$$
\Gamma \vdash \Delta, (\forall x : p) \\
\frac{\Gamma \vdash \Delta, (\forall x : p)\{x \leftarrow t\}}{\Gamma \vdash \Delta, p\{x \leftarrow t\}}
$$

**Instantiation** (inst, inst-cp)

$$
\Gamma, (\forall x : p) \vdash \Delta \\
\frac{\Gamma, (\forall x : p), p\{x \leftarrow t\} \vdash \Delta}{\Gamma, (\forall x : p), p\{x \leftarrow t\} \vdash \Delta}
$$

$$
\Gamma \vdash \Delta, (\exists x : p) \\
\frac{\Gamma \vdash \Delta, (\exists x : p), p\{x \leftarrow t\}}{\Gamma \vdash \Delta, p\{x \leftarrow t\}}
$$

**Strengthening Rules**

Allow a **stronger sequent** to be derived from a **weaker one** by **removing formulas**

$$
\Gamma, p \vdash \Delta \\
\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta}
$$

$$
\Gamma \vdash \Delta, p \\
\frac{\Gamma \vdash \Delta, p}{\Gamma \vdash \Delta}
$$
propositional rules: flatten

performs disjunctive simplification

\[
\{ -1 \} A1 \\
\{ -2 \} \text{not } A2 \\
\vdash \\
\{ 1 \} B1
\]

Rule ? (flatten)
propositional rules: flatten

performs disjunctive simplification

\[
\begin{align*}
\{\text{-1}\} & \ A1 \\
\{\text{-2}\} & \ \text{not} \ A2 \\
\text{ infer } & \ \text{- - -}
\end{align*}
\]

\[
\{\text{1}\} \ B1
\]

Rule ? (flatten)

\[
\begin{align*}
\{-1\} & \ A1 \\
\text{ infer } & \ \text{- - -}
\end{align*}
\]

\[
\{\text{1}\} \ B1
\]

\[
\{\text{2}\} \ A2
\]
propositional rules: flatten

performs disjunctive simplification

\[\begin{align*}
\{-1\} & \quad A1 \\
\{-2\} & \quad \textbf{not} \quad A2 \\
\{1\} & \quad B1 \\
\{-1\} & \quad A1 \quad \text{and} \quad A2 \\
\{1\} & \quad B1 \quad \textbf{implies} \quad B2 \\
\end{align*}\]

Rule ? (flatten)

\[\begin{align*}
\{-1\} & \quad A1 \\
\{1\} & \quad B1 \\
\{2\} & \quad A2 \\
\end{align*}\]

Rule ? (flatten)
propositional rules: flatten

performs disjunctive simplification

\[
\begin{align*}
\{ -1 \} & \quad A1 \\
\{ -2 \} & \quad \textbf{not} \ A2 \\
\{ 1 \} & \quad B1 \\
\{ -1 \} \quad & \quad A1 \text{ and } A2 \\
\{ 1 \} & \quad B1 \text{ implies } B2
\end{align*}
\]

Rule ? (flatten)

\[
\begin{align*}
\{ -1 \} & \quad A1 \\
\{ 1 \} & \quad B1 \\
\{ 2 \} & \quad A2
\end{align*}
\]

\[
\begin{align*}
\{ -1 \} \quad & \quad A1 \\
\{ -2 \} & \quad A2 \\
\{ -3 \} & \quad B1 \\
\{ 1 \} & \quad B2
\end{align*}
\]

Rule ? (flatten)
propositional rules: split

splits a conjunctive formula in the current goal and collects the resulting subgoal(s)

\{-1\} A1
\[\vdash\]
\{1\} B1 and B2

Rule ? (split 1)
propositional rules: split
splits a conjunctive formula in the current goal and collects the resulting subgoal(s)

\{-1\} A1
\vdash --
\{1\} B1 \text{ and } B2

Rule ? (split 1)

Subgoal.1
\{-1\} A1
\vdash --
\{1\} B1

Subgoal.2
\{-1\} A1
\vdash --
\{1\} B2
propositional rules: split

splits a conjunctive formula in the current goal and collects the resulting subgoal(s)

\[
\begin{align*}
\{\text{-1}\} & \quad A_1 \\
\vdash & \quad - - \\
\{1\} & \quad B_1 \text{ and } B_2
\end{align*}
\]

Rule ? (split 1)

Subgoal.1

\[
\begin{align*}
\{\text{-1}\} & \quad A_1 \\
\vdash & \quad - - \\
\{1\} & \quad B_1
\end{align*}
\]

Subgoal.2

\[
\begin{align*}
\{\text{-1}\} & \quad A_1 \\
\vdash & \quad - - \\
\{1\} & \quad B_2
\end{align*}
\]

\[
\begin{align*}
\vdash & \quad - - \\
[1] & \quad A_1 \iff A_2
\end{align*}
\]

Rule ? (split)
propositional rules: split
splits a conjunctive formula in the current goal and collects the resulting subgoal(s)

\{\text{-1}\} A1
\vdash - -
\{1\} B1 \text{ and } B2

Rule ? (split 1)

Subgoal.1
\[-1\] A1
\vdash - -
\{1\} B1

Subgoal.2
\[-1\] A1
\vdash - -
\{1\} B2

\vdash - -

Rule ? (split)

Subgoal.1
\vdash - -
\{1\} A1 implies A2
propositional rules: split
splits a conjunctive formula in the current goal and collects the resulting subgoal(s)

\[
\{\neg 1\} A1 \\
\vdash \quad \vdash \\
\{1\} B1 \text{ and } B2
\]

Rule ? (split 1)

Subgoal.1
\[
\neg 1 \quad A1 \\
\vdash \quad \vdash \\
\{1\} B1
\]

Subgoal.2
\[
\neg 1 \quad A1 \\
\vdash \quad \vdash \\
\{1\} B2
\]

\[
\vdash \quad \vdash \\
[1] A1 \iff A2
\]

Rule ? (split)

Subgoal.1
\[
\vdash \quad \vdash \\
\{1\} A1 \text{ implies } A2
\]

Subgoal.2
\[
\vdash \quad \vdash \\
\{1\} A2 \text{ implies } A1
\]
propositional rules: lift-if

lifts branching structure to the top level

\[ \vdash \quad \{1\} \ foo([\text{IF}(A,B,C)]) \]

Rule \( ? \) (lift-if)
propositional rules: lift-if

lifts branching structure to the top level

\[ \vdash - - \]
\[ \{1\} \ foo(\text{IF}(A,B,C)) \]

Rule \( ? \) (lift-if)

\[ \vdash - - \]
\[ [1] \ \text{IF}(A, \ foo(B), \ foo(C)) \]

Rule \( ? \) (split)
propositional rules: lift-if
lifts branching structure to the top level

\[ \vdash - - \]
\[ \{1\} \ foo(IF(A,B,C)) \]

Rule ? (lift-if)

\[ \vdash - - \]
\[ [1] \ IF(A, foo(B), foo(C)) \]

Rule ? (split)

Subgoal.1

\[ \vdash - - \]
\[ \{1\} \ A \ implies \ foo(B) \]
propositional rules: lift-if
lifts branching structure to the top level

\[ \vdash \quad \{1\} \ foo(\text{IF}(A,B,C)) \]

Rule ? (lift-if)

\[ \vdash \quad \{1\} \ A \implies foo(B) \]

Subgoal.1

\[ \vdash \quad \{1\} \ \text{not A} \implies foo(C) \]

Subgoal.2

\[ \vdash \quad \{1\} \ \text{IF}(A, foo(B), foo(C)) \]

Rule ? (split)
propositional rules: lift-if
lifts branching structure to the top level

\[
\begin{align*}
\Gamma & \vdash \varphi \\
\{1\} & \text{foo(\text{IF}(A,B,C))} \\
\end{align*}
\]

Rule ? (lift-if)

\[
\begin{align*}
\Gamma & \vdash \varphi \\
\{1\} & A \implies \text{foo}(B) \\
\end{align*}
\]

Subgoal.1

\[
\begin{align*}
\Gamma & \vdash \varphi \\
\{1\} & \neg A \implies \text{foo}(C) \\
\end{align*}
\]

Subgoal.2

\[
\begin{align*}
\Gamma & \vdash \varphi \\
\{-1\} & A \\
\{1\} & \text{foo}(B) \\
\end{align*}
\]

Subgoal.1
propositional rules: lift-if
lifts branching structure to the top level

\[ \vdash \quad \{1\} \ foo(\text{IF}(A,B,C)) \]

Rule ? (lift-if)

\[ \vdash \quad \{1\} \not \ A \ implies \ foo(C) \]

Subgoal.1
\[ \vdash \quad \{1\} \ A \ implies \ foo(B) \]

Subgoal.2

\[ \vdash \quad \{1\} \ foo(B) \]

Subgoal.1
\[ \vdash \quad \{1\} \ A \]

Subgoal.2
\[ \vdash \quad \{1\} \ A \]
\[ \{2\} \ foo(C) \]

Rule ? (split)
quantifier rules: skolem, skolem! , and typepred
replace universally quantified variables with constants

\[ \{-1\} A1 \]
\[ \vdash \_{-} \_{-} \]
\[ \{1\} \text{Forall} (s:\text{Start}): B1(s) \]

Rule ? (skolem ("s1"))
quantifier rules: skolem, skolem!, and typepred
replace universally quantified variables with constants

\[\begin{array}{c}
\{\text{-1}\} \ A1 \\
\vdash \quad \quad \\
\{\text{1}\} \ \text{Forall} \ (s:\text{Start}): \ B1(s)
\end{array}\]

Rule ? (skolem ("s1"))

\[\begin{array}{c}
\{\text{-1}\} \ A1 \\
\vdash \quad \quad \\
\{\text{1}\} \ B1(s1)
\end{array}\]
quantifier rules: skolem, skolem!, and typepred
replace universally quantified variables with constants

\[ \{-1\} A1 \]
\[ \vdash - - \]
\[ \{1\} \text{Forall (s:Start): } B1(s) \]

Rule ? (skolem ("s1"))

\[ [-1] A1 \]
\[ \vdash - - \]
\[ \{1\} B1(s1) \]

Rule ? (typepred "s1")

\[ \{-1\} \text{Start(s1)} \]
\[ [-2] A1 \]
\[ \vdash - - \]
\[ [1] B1(s1) \]
quantifier rules: skolem, skolem! , and typepred
replace universally quantified variables with constants

{-1} A1
|-  -  -
{1} Forall (s:Start): B1(s)

{-1} Exists (s:Start): A1(s)
|-  -  -
{1} B1

Rule ? (skolem "s1")

[-1] A1
|-  -  -
{1} B1(s1)

Rule ? (skolem "s0")

[-1] A1
|-  -  -
{1} B1(s1)

Rule ? (typepred "s1")

{-1} Start(s1)
[-2] A1
|-  -  -
[1] B1(s1)
quantifier rules: skolem, skolem!, and typepred
replace universally quantified variables with constants

\{-1\} A1
\implies \quad \implies
\{1\} \textbf{Forall} (s: \texttt{Start}): B1(s)
\{1\} B1

\textit{Rule ? (skolem "s1")}

\{-1\} A1
\implies \quad \implies
\{1\} B1(s1)
\{1\} B1

\textit{Rule ? (typepred "s1")}

\{-1\} \texttt{Start}(s1)
\{-2\} A1
\implies \quad \implies
\{1\} B1(s1)
quantifier rules and introducing lemmas

\{-1\} A1
\-
\{1\} Exists (n:nat): B1(n)
Rule ? (inst 1 (n "5"))
quantifier rules and introducing lemmas

\{-1\} A1
\[\begin{array}{c}
\end{array}\]
\{1\} \textbf{Exists} (n:nat): B1(n)
Rule ? (inst \textbf{1} (n "5"))

\{-1\} A1
\[\begin{array}{c}
\end{array}\]
\{1\} B1(5)
quantifier rules and introducing lemmas

\[
\begin{align*}
\{\neg 1\} & \ A1 \\
\text{\vdash} & \ \text{--} \\
\{1\} & \ \textbf{Exists} \ (n:\text{nat}) : \ B1(n) \\
\text{Rule } ? (\textbf{inst} \ 1 \ (n \ "5")) \\
\{\neg 1\} & \ A1 \\
\text{\vdash} & \ \text{--} \\
\{1\} & \ B1(5)
\end{align*}
\]
quantifier rules and introducing lemmas

\[
\begin{align*}
\{ -1 \} & \quad A1 \\
\quad & \quad \vdash \quad \vdash \\
\{ 1 \} & \quad \exists n : \text{nat} \; : \; B1(n) \\
& \quad \text{Rule \ ? (inst 1 (n "5"))}
\end{align*}
\]

\[
\begin{align*}
\{ -1 \} & \quad A1 \\
\quad & \quad \vdash \quad \vdash \\
\{ 1 \} & \quad B1(5) \\
\end{align*}
\]

Suppose we have:

Fact: Lemma \( \exists n : \neg \neg P(n) \)
quantifier rules and introducing lemmas

\[
\{ -1 \} \ A1 \\
\vdash \quad \quad \\
\{ 1 \} \ \textbf{Exists} \ (n: \text{nat}) : \ B1(n) \\
\text{Rule} \ ? \ (\text{inst} \ 1 \ (n \ "5"))
\]

\[
\{ -1 \} \ A1 \\
\vdash \quad \quad \\
\{ 1 \} \ B1(5)
\]

Suppose we have:

\textit{Fact: Lemma Exists} \ (n) : \ P(n)

ongoing proof sequent...

\[
\{ -1 \} \ \textbf{Forall} \ (n) : \ P(n) \Rightarrow Q(n) \\
\vdash \quad \quad \\
\{ 1 \} \ \textbf{Exists} \ (n) : \ Q(n)
\]
quantifier rules and introducing lemmas

\[\{1\} \text{Exists} \ (n: \text{nat}) : B1(n)\]
\[\text{Rule ? (inst 1 (n "5"))}\]

\[\{1\} \ B1(5)\]

Suppose we have:

\text{Fact: Lemma } \text{Exists}(n) : P(n)\]

ongoing proof sequent...

\[\{1\} \text{Forall}(n) : P(n) \Rightarrow Q(n)\]
\[\text{Rule ? (lemma "Fact")}\]
quantifier rules and introducing lemmas

\[
\begin{align*}
\{1\} A1 \\
\vdash \quad - \\
\{1\}\text{ Exists (n:nat): } B1(n) \\
\text{Rule ? (inst 1 (n "5"))} \\
\end{align*}
\]

\[
\begin{align*}
\{1\} B1(5) \\
\end{align*}
\]

Suppose we have:

Fact: Lemma \text{ Exists(n): } P(n)

ongoing proof sequent...

\[
\begin{align*}
\{1\} \text{ Forall(n): } P(n) \Rightarrow Q(n) \\
\vdash \quad - \\
\{1\} \text{ Exists(n): } Q(n) \\
\end{align*}
\]

Rule ? (lemma "Fact")
quantifier rules and introducing lemmas

\[-1\] \( A \)
\[1\] \( \textbf{Exists} (n: \text{nat}) : B1(n) \)
\( \text{Rule ? (inst 1 (n "5"))} \)

\[-1\] \( A \)
\[1\] \( B1(5) \)

Suppose we have:

\textbf{Fact: Lemma} \( \text{Exists}(n): P(n) \)

ongoing proof sequent...

\[-1\] \( \textbf{Forall}(n): P(n) \Rightarrow Q(n) \)
\[1\] \( \text{Exists}(n): Q(n) \)

\( \text{Rule ? (lemma "Fact")} \)
quantifier rules and introducing lemmas

\{-1\} A1

\{-2\} \exists(n) : P(n)

\{-1\} \forall(n) : P(n) \Rightarrow Q(n)

\begin{align*}
\{1\} & \exists(n: \text{nat}) : B1(n) \\
\text{Rule } \Omega & (\text{inst } 1\ (n \ "5"))
\end{align*}

\{-2\} \forall(n) : P(n) \Rightarrow Q(n)

\begin{align*}
\{1\} & \exists(n) : Q(n) \\
\text{Rule } \Omega & (\text{skolem } -1\ "n1")
\end{align*}

\[-1\] A1

\{-1\} P(n1)

\begin{align*}
\{1\} & B1(5) \\
\{1\} & \exists(n) : Q(n)
\end{align*}

Suppose we have:

Fact: Lemma \exists(n) : P(n) \quad \{1\} \exists(n) : Q(n)

ongoing proof sequent...

\{-1\} \forall(n) : P(n) \Rightarrow Q(n)

\begin{align*}
\{1\} & \exists(n) : Q(n) \\
\text{Rule } \Omega & (\text{lemma } "\text{Fact}")
\end{align*}
quantifier rules and introducing lemmas

{-1} A1
- - -
{1} Exists (n:nat): B1(n)
Rule ? (inst 1 (n "5"))

{-1} Exists(n): P(n)
[-2] Forall(n): P(n) \Rightarrow Q(n)
- - -
[1] Exists(n): Q(n)

Rule ? (skolem -1 "n1")

{-1} A1
- - -
{1} B1(5)

Suppose we have:

Fact: Lemma Exists(n): P(n)

ongoing proof sequent...

{-1} Forall(n): P(n) \Rightarrow Q(n)
- - -
{1} Exists(n): Q(n)

Rule ? (inst -2 "n1")

{-1} P(n1)
[-2] Forall(n): P(n) \Rightarrow Q(n)
- - -
[1] Exists(n): Q(n)

Rule ? (lemma "Fact")
quantifier rules and introducing lemmas

\[\begin{align*}
\{\neg 1\} \ A1 \\
\end{align*}\]

\[\begin{align*}
\{1\} \ \text{Exists} \ (n:\text{nat}) : B1(n) \\
\text{Rule '? (inst 1 (n "5"))}'
\end{align*}\]

\[\begin{align*}
\{\neg 1\} \ A1 \\
\end{align*}\]

\[\begin{align*}
\{1\} \ B1(5) \\
\text{Suppose we have:}
\end{align*}\]

\[\begin{align*}
\text{Fact: Lemma Exists} (n) : P(n) \\
\text{ongoing proof sequent...}
\end{align*}\]

\[\begin{align*}
\{\neg 1\} \ \text{Forall} (n) : P(n) \Rightarrow Q(n) \\
\end{align*}\]

\[\begin{align*}
\{1\} \ \text{Exists} (n) : Q(n) \\
\text{Rule '? (inst -2 "n1")'}
\end{align*}\]

\[\begin{align*}
\{\neg 1\} \ P(n1) \\
\end{align*}\]

\[\begin{align*}
\{2\} \ P(n1) \Rightarrow Q(n1) \\
\end{align*}\]

\[\begin{align*}
\{1\} \ \text{Exists} (n:\text{nat}) : Q(n) \\
\text{Rule '? (lemma "Fact")'}
\end{align*}\]
control rules

1. \textit{(undo} k\text{)} undoes proof back to \textit{k}^{th} level ancestor
2. \textit{(postpone)} mark current goal as pending and move focus to next unproved goal in proof tree
3. \textit{(quit)} terminate current proof attempt
control rules

1. *(undo)* $k$ undoes proof back to $k^{th}$ level ancestor
2. *(postpone)* mark current goal as pending and move focus to next unproved goal in proof tree
3. *(quit)* terminate current proof attempt

```
          main goal
            |
          cmd1   goal
            |
        cmd2   |
          goal.1   goal.2
            |
          cmd3   cmd4
            |
  goal2.1  goal2.2
```
more prover commands

- (expand "foo"): expands the definition of "foo" in the sequent
more prover commands

- **(expand "foo"):** expands the definition of "foo" in the sequent
- **(induct "n"):** for a universally quantified formula over natural numbers this invokes the standard induction schema
more prover commands

- (expand "foo"): expands the definition of "foo" in the sequent
- (induct "\(n\)\): for a universally quantified formula over natural numbers this invokes the standard induction schema
- (induct "\(x\)\): does the same for any well-founded set with an associated induction schema
more prover commands

- **(expand "foo")**: expands the definition of "foo" in the sequent
- **(induct "n")**: for a universally quantified formula over natural numbers this invokes the standard induction schema
- **(induct "x")**: does the same for any well-founded set with an associated induction schema
- **(apply-extensionality)**: deduce $f = g$ from $f(a) = g(a), f(b) = g(b)$, for $f, g : \{a, b\} \rightarrow T$
Example

$$\varphi : (\forall x : P(x) \lor \neg Q(x)) \rightarrow (\exists y : P(y)) \lor (\forall z : \neg Q(z))$$

$$\vdash \neg(\forall x : P(x) \lor \neg Q(x)) \lor (\exists y : P(y)) \lor (\forall z : \neg Q(z))$$

**flatten**

$$\vdash \neg(\forall x : P(x) \lor \neg Q(x)), (\exists y : P(y)), (\forall z : \neg Q(z))$$

**flatten**

$$(\forall x : P(x) \lor \neg Q(x)) \vdash (\exists y : P(y)), (\forall z : \neg Q(z))$$

**skolemize**

$$(\forall x : P(x) \lor \neg Q(x)) \vdash (\exists y : P(y)), \neg Q(a)$$

**flatten**

$$(\forall x : P(x) \lor \neg Q(x)), Q(a) \vdash (\exists y : P(y))$$

**instantiate x with a**

$$P(a) \lor \neg Q(a), Q(a) \vdash (\exists y : P(y))$$

**split**

$$P(a), Q(a) \vdash (\exists y : P(y))$$

$$\neg Q(a), Q(a) \vdash (\exists y : P(y))$$

**instantiate y with a**

**flatten**

$$P(a), Q(a) \vdash P(a)$$

$$Q(a) \vdash (\exists y : P(y)), Q(a)$$
A *clock of period* $K$, is a set of “sample instances”:

$$
\text{clock}_K := \{t_0, t_1, t_2, \ldots, t_n, \ldots\} \\
= \{0, K, 2K, \ldots, nK, \ldots\}
$$

E.g., for a period $K = 5$, the clock of period 5 is simply

$$
\text{clock}_5 := \{0, 5, 10, 15, \ldots\}
$$

Can define $\text{pre}$, $\text{next}$ and $\text{init}$ operators on clock values:

$$
\text{pre}_K(t_n) := \begin{cases} 
    t_{n-1}, & n \geq 1 \\
    \text{undefined}, & \text{otherwise}
\end{cases}
$$

$$
\text{next}_K(t_n) := t_{n+1}
$$

$$
\text{init}(t_n) := \begin{cases} 
    \text{TRUE}, & n = 0 \\
    \text{FALSE}, & \text{otherwise}
\end{cases}
$$
Specifying Timing in PVS: Clocks Theory

Clocks[ K: posreal ]: THEORY
BEGIN
non_neg: TYPE = { x: real | x>=0 }
time: TYPE = non_neg
t: VAR time

clock: TYPE = { t: time | EXISTS(n:nat): t=n*K }

x: VAR clock

init(x): bool = (x=0)
noninit_elem: TYPE = { x | not init(x) }
y: VAR noninit_elem

pre(y): clock = y - K
next(x): noninit_elem = x + K
rank(x): nat = x/K

clock_induction: PROPOSITION
  FORALL (P: pred[clock]):
    (FORALL (x: clock): init(x)
      IMPLIES P(x)) AND
    (FORALL (y: noninit_elem): P(pre(y))
      IMPLIES P(y))
    IMPLIES (FORALL (x: clock): P(x))

END Clocks
HELD\_FOR Operator

\[ \text{HELD\_FOR} : \text{pred}(\text{clock}_K) \times \mathbb{R}^+ \rightarrow \text{pred}(\text{clock}_K) \]

For \( P : \text{clock}_K \rightarrow \{\text{TRUE, FALSE}\} \),

\[ P \ \text{HELD\_FOR}(\text{duration})(t_n) = \text{TRUE} \]

iff \( (\exists t_j \in \text{clock}_K) \) such that

\[ (t_n - t_j \geq \text{duration}) \land \]
\[ (\forall t_i \in \text{clock}_K)(t_j \leq t_i \leq t_n \Rightarrow P(t_i)) \]

Example 1: Let \( K = 150 \), \( \text{duration} = 295 \), and \( \text{Sensor}(t) \) be a clock predicate:

\[
\begin{array}{c|cccc}
\text{Sensor} & T & F & \_ & \_ \\
\text{F} & 0 & 1 & 2 & 3 \\
\text{time} & \_ & \_ & \_ & \_ \\
\end{array}
\]

\[
\begin{array}{c|cccc}
st & 0 & 150 & 300 & 450 \\
f & F & F & F & T \\
\end{array}
\]

\[ f = (\text{Sensor})\ \text{HELD\_FOR}(295) \] example

**NOTE:** We ignore intersample behavior of \( \text{Sensor} \).
Held_For [K:posreal] : THEORY
BEGIN
IMPORTING Clocks[K]

\[\text{t, t_now, t_n, t_j: VAR clock}\]
\[\text{duration: VAR time}\]
\[\text{P: VAR pred[clock]}\]

\[\text{Held_For(P, duration): pred[clock] = (LAMBDA (t_n): (LAMBDA (t_j): (t_n - t_j) \geq \text{duration}) \text{ and FORALL}(t: clock | t \geq t_j \& t \leq t_n): P(t)})}\]

END Held_For
A Small Example: Sensor-lock system

simple : THEORY
BEGIN

K: posreal = 50
IMPORTING Held_For[K]

t: VAR clock

Sensor(t):bool = IF (t<1000) THEN FALSE
ELSE TRUE ENDS

duration:time = 295

good: THEOREM (t>=1000+duration) IMPLIES
    Held_For(Sensor,duration)(t)

bad: THEOREM (t>=1000+duration-K)
    IMPLIES Held_For(Sensor,duration)(t)

END simple
A Small Example: Sensor-lock system (cont’d)

Prove theorem ‘good’ of the Sensor-lock example

\[ \forall t_n \in \text{clock}_{50} \cdot (t_n \geq 1295 \Rightarrow \text{Held\_For}(\text{Sensor}, \text{duration})(t_n)) \]

\[ \equiv \{\text{Definition of Held\_For}\} \]
\[ \forall t_n \in \text{clock}_{50} \cdot (t_n \geq 1295 \Rightarrow (\exists t_j \in \text{clock}_{50} \cdot (t_n - t_j \geq 295) \land (\forall t_i \in \text{clock}_{50} \cdot t_j \leq t_i \leq t_n \Rightarrow \text{Sensor}(t_i)))) \]

\[ \equiv \{\text{rule } \Rightarrow 2 \lor\} \]
\[ \forall t_n \in \text{clock}_{50} \cdot (t_n < 1295 \lor (\exists t_j \in \text{clock}_{50} \cdot (t_n - t_j \geq 295) \land (\forall t_i \in \text{clock}_{50} \cdot t_j \leq t_i \leq t_n \Rightarrow \text{Sensor}(t_i)))) \]

\[ \equiv \{\exists \text{ elimination: } t_j = 1000\} \]
\[ \forall t_n \in \text{clock}_{50} \cdot (t_n < 1295 \lor (t_n \geq 1295 \land (\forall t_i \in \text{clock}_{50} \cdot 1000 \leq t_i \leq t_n \Rightarrow \text{Sensor}(t_i)))) \]

\[ \equiv \{\text{Definition of Sensor}(t)\} \]
\[ \forall t_n \in \text{clock}_{50} \cdot (t_n < 1295 \lor t_n \geq 1295) \]

\[ \equiv \{\text{LEM}\} \]
\[ True \]
Example: Reactor Shutdown System (SDS)

What is an SDS?

- watchdog system that monitors system parameters
- shuts down (trips) reactor if it observes "bad" behavior
- process control is performed a separate Digital Control computer (DCC) - not as critical

Consider simple subsystem: Power Conditioning

- Many sensors have a Power threshold below (or above) which readings are unreliable so it’s “conditioned out” for certain Power levels.
- A deadband is used to eliminate sensor “chatter”

Idea: Use code reuse - write one general routine and pass in sensor parameters for different sensors
General Power Conditioning Function

\[
PwrCond(\text{Prev:bool, Power, Kin, Kout:posreal}):bool =
\]

<table>
<thead>
<tr>
<th>Power $\leq$ Kout</th>
<th>Kout $&lt;$ Power $&lt;$ Kin</th>
<th>Power $\geq$ Kin</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>(\text{Prev} )</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

PVS (Prototype Verification System), a “proof assistant” can automatically check for completeness (coverage) and determinism (disjointness) of tables.

i.e. PVS checks that a table defines a total function.
Power Conditioning: Description

When Power:

- drops below $K_{out}$, sensor is unreliable so it’s “conditioned out” ($\text{PwrCond} = \text{FALSE}$).
- exceeds $K_{in}$, the sensor is “conditioned in” and is used to evaluate the system.
- is between $K_{out}$ and $K_{in}$, the value of $\text{PwrCond}$ is left unchanged by setting it to its previous value, $\text{Prev}$.

E.g. For the graph of Power above, $\text{PwrCond}$ would start out FALSE, then become TRUE at time $t_1$ and remain TRUE.
PVS Specification of a General $PwrCond$ Function

\[
PwrCond(\text{Prev}: \text{bool}, \text{Power}, \text{Kin}, \text{Kout}: \text{posreal}): \text{bool} = \text{TABLE}
\]
\[
\begin{array}{|c|c|}
\hline
\text{Prev} & \\text{Prev} & \text{TRUE} \\
\hline
\end{array}
\]

\[
\text{ENDTABLE}
\]

The above PVS specification of the $PwrCond$ table produces the following proof obligations or "TCCs".

\[
\% \text{Disjointness TCC generated (at line 14, column 55) for unfinished}
\]

PwrCond_TCC1: OBLIGATION

\[
\text{FORALL (Kin, Kout: posreal, Power):}
\]
\[
\begin{array}{c}
\text{NOT (Power } \leq \text{ Kout AND Power } > \text{ Kout & Power } < \text{ Kin) AND}
\end{array}
\]
\[
\begin{array}{c}
\text{NOT (Power } \leq \text{ Kout AND Power } \geq \text{ Kin) AND}
\end{array}
\]
\[
\begin{array}{c}
\text{NOT ((Power } > \text{ Kout & Power } < \text{ Kin) AND Power } \geq \text{ Kin});}
\end{array}
\]

\[
\% \text{Coverage TCC generated (at line 14, column 55) for proved - complete}
\]

PwrCond_TCC2: OBLIGATION

\[
\text{FORALL (Kin, Kout: posreal, Power):}
\]
\[
\begin{array}{c}
\text{(Power } \leq \text{ Kout OR} \% \text{ Column1}
\end{array}
\]
\[
\begin{array}{c}
\text{(Power } > \text{ Kout & Power } < \text{ Kin) \% Column2}
\end{array}
\]
\[
\begin{array}{c}
\text{OR Power } \geq \text{ Kin) \% Column3}
\end{array}
\]
Type-checking PwrCond

The coverage TCC is easily proved by PVS. Thus we conclude that at least one column is always satisfied for every input.

But attempting the Disjointness TCC fails, indicating that the columns overlap. The resulting unprovable sequent for the disjointness TCC is:

\[ \text{PwrCond\_TCC1} : \]

\[-1\] \( \text{Kin!1} > 0 \)
\[-2\] \( \text{Kout!1} > 0 \)
\[-3\] \( \text{Power!1} > 0 \)
\[-4\] \( \text{Power!1} \leq \text{Kout!1} \)
\[-5\] \( (\text{Kin!1} \leq \text{Power!1}) \)
|--------
\[ \text{[1]} \quad \text{FALSE} \]

Rule?
Step 1: Characteristic Equation

Writing down the characteristic formula for the unprovable sequent.

\[ K_{in1} > 0 \land K_{out1} > 0 \land Power_1 > 0 \land \\
Power_1 \leq K_{out1} \land K_{in1} \leq Power_1 \rightarrow \bot \]

which is equivalent to:

\[ \neg(Power > 0 \land K_{in} > 0 \land K_{out} > 0 \land \\
Power \leq K_{out} \land K_{in} \leq Power) \]

We know that an interpretation structure (i.e., a program) will satisfy this formula iff it satisfies the formula’s universal closure:

\[ (\forall Power, K_{in}, K_{out} : \text{posreal}) \]

\[ \neg(Power > 0 \land K_{in} > 0 \land K_{out} > 0 \land \\
Power \leq K_{out} \land K_{in} \leq Power) \quad (1) \]
Step 2: Find a Counter Example

Find a counter example that makes the characteristic formula false.

NOTE: The counter examples values for *Kin*, *Kout* and *Power* must be of the type

\[ \text{posreal} = \{ x : \text{real} | x > 0 \} \]

So, to make (1) false, we make \( \neg(1) \), if equivalently the following true:

\[ (\exists \text{Power, Kin, Kout} : \text{posreal}) \]
\[ (\text{Power} > 0 \land \text{Kin} > 0 \land \text{Kout} > 0 \land \text{Power} \leq \text{Kout} \land \text{Kin} \leq \text{Power}) \]

With a slight abuse of notation this simplifies to:

\[ (\exists \text{Power, Kin, Kout} : \text{posreal}) \]
\[ (0 < \text{Kin} \leq \text{Power} \leq \text{Kout}) \]

e.g. Take *Kin*₁ = 1, *Kout*₁ = 3, *Power*₁ = 2
Step 3: Verify Counter Example

Verify that the counter example satisfies the conditions of two or more columns of the table for PwrCond.

In the case when $K_{in} = 1$, $K_{out} = 3$, $Power = 2$ we have the condition on column 1 and condition 3 satisfied since:

i) $Power \leq K_{out}$ because $2 \leq 3$, and

ii) $Power \geq K_{in}$ because $1 \leq 2$. 
Step 4: Find the Error Source

What implicit assumption did the designers make regarding input arguments Kin and Kout that led them to omit the counter example case from the table?

They assumed that $K_{out} < K_{in}$.

Why is such an undocumented assumption dangerous in a setting where code may be reused by other developers?

When someone other than the code developer reuses the code, they may not know about any implicit assumption and may use the code in a way that it was not intended for.

E.g. Suppose a sensor was only valid at low values of Power. Someone may think that they could just use function PwrCond with $K_{out} > K_{in}$. In this case it is not specified what the function will do!
Step 5: Correct the Error

Problem: Determinism check fails when

\[ Kin \leq Kout \]

Why? Implicit (undocumented) assumption from diagram that \( Kin > Kout \)

Fix: Make assumption explicit.

How? Use dependent typing to create a new version of the PwrCond table that makes the assumed relation between Kin and Kout explicit and thereby rules out any counter examples like those above.

\[
PwrCond(Prev:bool, Power, Kin:posreal, 
    Kout:\{x:posreal| x<Kin\}):bool = TABLE 
\]

\[
| [Power<=Kout | Power>Kout & Power<Kin | Power>=Kin] |
|------------------------------|
| FALSE | Prev | TRUE |
|------------------------------|
\]

ENDTABLE
Exercise

A. Prove or disprove the two claims below, by using the inference rules given in the course. In case the argument is not valid, find a counter-example:

1. $\vdash ((p \Rightarrow q) \Rightarrow q) \Rightarrow ((q \Rightarrow p) \Rightarrow p)$

2. $\vdash ((q \Rightarrow (m \lor r)) \land m \land (r \Rightarrow q)) \Rightarrow q$