What is PVS

- **PVS** is an extensive higher-order-logic **deductive verification system** based on sequent calculus.

- **Specification language** for writing **theorems**

- **Interactive prover**

- **Full documentation** at the course’s web page.
Logic of PVS

The prover maintains a proof tree.

- **Root** is the premise to be proved
- **Offspring** follow from a node by a proof step
- **Tree** is complete when all leaves are true
- Each node is proof goal
- Each proof goal is a sequent
Sequents

A sequent is comprised of sequent formulas: antecedents followed by consequents.

Represented in the form

\[ p_1, p_2, \ldots, p_n \vdash q_1, q_2, \ldots, q_m, \]

where \( p_i \) are the antecedents, \( q_i \) are the consequents.

Interpretation:

\[ \forall \text{free} : (p_1 \land p_2 \land \ldots \land p_n) \rightarrow (q_1 \lor q_2 \lor \ldots \lor q_m) \]

where free denotes the free (unbound) variables.
Sequent Axioms

Used to prove that the leaf sequents are true

\[ A1 : \Gamma, p \vdash \Delta, p \]
\[ A2 : \Gamma \vdash \Delta, 1 \]
\[ A3 : \Gamma, 0 \vdash \Delta \]

I.e. \[ A1 : \Gamma \land p \rightarrow \Delta \lor p \]

\[ A2 : \Gamma \rightarrow \Delta \lor 1 \]

\[ A3 : \Gamma \land 0 \rightarrow \Delta \quad \leftrightarrow \]
\[ \neg (\Gamma \land 0) \lor \Delta \quad \leftrightarrow \]
\[ \neg \Gamma \lor \neg 0 \lor \Delta \quad \leftrightarrow \]
\[ \neg \Gamma \lor 1 \lor \Delta \quad \leftrightarrow \]
\[ 1 \]

Corresponds to PVS assert command.
Propositional Rules

The inference rule \( \Gamma \vdash \Delta \xrightarrow{\Gamma_1 \vdash \Delta_1} \) means:

In order to prove \( \Gamma \vdash \Delta \), it is sufficient to prove \( \Gamma_1 \vdash \Delta_1 \).

Following are some useful (propositional) inference rules:

\[
\begin{align*}
\Gamma \vdash \Delta, p \lor q & \quad \Gamma, p \land q \vdash \Delta \\
\Gamma \vdash \Delta, p, q & \quad \Gamma, p, q \vdash \Delta \\
\Gamma, \neg p \vdash \Delta & \quad \Gamma \vdash \Delta, \neg p \\
\Gamma \vdash \Delta, p & \quad \Gamma, p \vdash \Delta \\
\end{align*}
\]

Correspond to the PVS flatten command.

\[
\begin{align*}
\Gamma, p \lor q \vdash \Delta & \quad \Gamma \vdash \Delta, p \land q \\
\Gamma, p \vdash \Delta & \quad \Gamma, q \vdash \Delta \\
\end{align*}
\]

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

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

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

Instantiation (inst, inst-cp)

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

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

Strengthening Rules

Up to now, all rules replaced a sequent by a validity-equivalent sequent. That is, no information has been lost.

The following rule allows the replacement of a sequent by a stronger sequent, obtained by removing formulas:

Hiding (delete, hide)

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

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

\[ P(a), Q(a) \vdash P(a) \]

\[ Q(a) \vdash (\exists y : P(y)), Q(a) \]

1 - Axiom A1

1 - Axiom A1
Basic Definitions in PVS

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.
Example - reservations

reservation: THEORY
BEGIN
  room: TYPE
  date: TYPE
  name: TYPE
  free: name
  reservations: TYPE = [room, date → name]
  reserve(r:room, d:date, n:name, reg:reservations):
    reservations = reg WITH [(r, d) := n]
  cancel(r:room, d:date, reg:reservations):
    reservations = reg WITH [(r, d) := free]
  reserved(r:room, d:date, reg:reservations): bool =
    reg(r, d) ≠ free
END reservation

- room, date, name are uninterpreted types
- free is a constant
- reservations is an uninterpreted function
- reserve, reserved, cancel are interpreted functions
Proving Lemmas

\texttt{reserved}(r, d, \texttt{reg}): \ \texttt{bool} = \texttt{reg}(r, d) \neq \texttt{free}
\texttt{cancel}(r, d, \texttt{reg}): \ \texttt{reservations} =
\begin{align*}
\texttt{reg} \ \texttt{WITH} \ [(r, d) := \texttt{free}]
\end{align*}

\texttt{canceled\_not\_reserved}: \ \textbf{LEMMA}
\begin{align*}
\forall \ r, d, \texttt{reg}: \ \neg \ \texttt{reserved}(r, d, \texttt{cancel}(r, d, \texttt{reg}))
\end{align*}
\begin{align*}
\{1\} \ \ \text{FORALL} \ r, d, \texttt{reg}: \ \text{NOT} \ \texttt{reserved}(r, d, \texttt{cancel}(r, d, \texttt{reg}))
\end{align*}
Rule? (skosimp*)

\begin{align*}
\{1\} \ \ \texttt{reserved}(r!1, d!1, \texttt{cancel}(r!1, d!1, \texttt{reg}!1))
\end{align*}
\begin{align*}
\{1\} \ \ \text{cancel}(r!1, d!1, \texttt{reg}!1)(r!1, d!1) \neq \texttt{free}
\end{align*}
Rule? (expand "\texttt{reserved}")

\begin{align*}
\{1\} \ \ \text{FALSE}
\end{align*}
Rule? (expand "\texttt{cancel}")

\begin{align*}
\{1\} \ \ \text{FALSE}
\end{align*}
\begin{align*}
\text{which is trivially true.}
\end{align*}
Q.E.D.
Proving Lemmas - ctd

Alternatively, the **grind** command would have **proved** this lemma.

**grind** is a **strategy** that **expands definitions, skolemizes, instantiates, simplifies** ...  

It can often be used to **complete a proof**.
Another lemma

\begin{verbatim}
reserved(r, d, reg):  bool = reg(r, d) \neq free
reserve(r, d, n, reg):  reservations =
  reg WITH [(r, d) := n]

is_reserved:  LEMMA
  \forall r, d, n, reg:
  reserved(r, d, reserve(r, d, n, reg))

is_reserved:

  |------
  {1}  FORALL r, d, n, reg: reserved(r, d, reserve(r, d, n, reg))
Rule? (skosimp*)

  |------
  {1}  reserved(r!1, d!1, reserve(r!1, d!1, n!1, reg!1))
Rule? (expand "reserved")

  |------
  {1}  reserve(r!1, d!1, n!1, reg!1)(r!1, d!1) /= free
Rule? (expand "reserve")

{-1}  n!1 = free
  |------
Rule?
\end{verbatim}
The LTL framework

A set of PVS theories and strategies defining basic LTL constructs, and proof rules like BINV.

Example MUX-SEM

\[
\begin{align*}
\text{in} & \quad N : \text{integer where } N > 1 \\
\text{local} & \quad y : \{0, 1\} \text{ where } y = 1
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
N \\
p=1
\end{array} \\
\quad P[p] :: \\
\quad \begin{cases}
\quad \ell_0 : \text{loop forever do} \\
& \begin{cases}
\quad \ell_1 : \text{noncritical} \\
\quad \ell_2 : \text{request } y \\
\quad \ell_3 : \text{critical} \\
\quad \ell_4 : \text{release } y
\end{cases}
\end{cases}
\end{align*}
\]

Figure 1: Parameterized MUX-SEM
Proving Mutual Exclusion

reachable: ASSERTION =

...(∀ (i: PROC_ID): ∀ (j: PROC_ID):
(loc(i) > 2 → y = 0) ∧
(i = j ∨ loc(i) < 3 ∨ loc(j) < 3))

Hints:

• Existential quantification is expensive, often requiring manual instantiation: avoid when possible

• Try to take universal quantifiers to the top level

• Disjunction (∨) is more difficult to work with than conjunction (∧), often requiring manual splitting.
The proof

\[
\begin{array}{l}
\text{\{1\}} \quad \text{is\_P\_reachable\_valid(G(reachable)), pfs)}
\end{array}
\]

Rule? (BINV "reachable")

\[
\begin{array}{l}
\text{Rule BINV}
\end{array}
\]

\[
\begin{array}{l}
\text{l1. } \Theta \rightarrow \varphi
\end{array}
\]

\[
\begin{array}{l}
\text{l2. } \varphi(V) \land \rho(V, V') \rightarrow \varphi(V')
\end{array}
\]

\[
\begin{array}{l}
\hline
G\varphi
\end{array}
\]
The inductive step of the BINV rule

{−1,(rho)}

loc(current!1)(p!1) = 0 AND y(next!1) = y(current!1) AND
loc(next!1) = loc(current!1) WITH [(p!1) := 1]
OR loc(current!1)(p!1) = 1 AND y(next!1) = y(current!1) AND
loc(next!1) = loc(current!1) WITH [(p!1) := 2]
OR loc(current!1)(p!1) = 2 AND y(current!1) = 1 AND
y(next!1) = 0 AND loc(next!1) = loc(current!1) WITH [(p!1) := 3]
OR loc(current!1)(p!1) = 3 AND y(next!1) = y(current!1) AND
loc(next!1) = loc(current!1) WITH [(p!1) := 4]
OR loc(current!1)(p!1) = 4 AND y(next!1) = 1 AND
loc(next!1) = loc(current!1) WITH [(p!1) := 0]

{−2,(reachable invariant)}

FORALL (i: PROC_ID): FORALL (j: PROC_ID):
(loc(current!1)(i) > 2 IMPLIES y(current!1) = 0) AND
(i = j OR loc(current!1)(i) < 3 OR loc(current!1)(j) < 3)

{1,(rtp)}

(loc(next!1)(i!1) > 2 IMPLIES y(next!1) = 0) AND
(i!1 = j!1 OR loc(next!1)(i!1) < 3 OR loc(next!1)(j!1) < 3)
Rule? (split-rho)

this yields 5 subgoals:

reachableInv.1 :
{-1,(rho)} loc(current!1)(p!1) = 0
{-2,(rho)} y(next!1) = y(current!1)
{-3,(rho)}
    loc(next!1) = loc(current!1) WITH [(p!1) := 1]
{-4,(reachable invariant)}
    FORALL (i: PROC_ID): FORALL (j: PROC_ID):
        IF loc(current!1)(i) > 2 THEN y(current!1) = 0 ELSE TRUE ENDIF
        AND (i = j OR loc(current!1)(i) < 3 OR loc(current!1)(j) < 3)
|-------
{1,(rtp)}
    IF loc(current!1) WITH [(p!1) := 1](i!1) > 2
        THEN y(current!1) = 0
    ELSE TRUE ENDIF
    AND
    (i!1 = j!1 OR
        loc(current!1) WITH [(p!1) := 1](i!1) < 3 OR
        loc(current!1) WITH [(p!1) := 1](j!1) < 3)
Rule? (inst = "i!1" "j!1")

[-1,(rho)] loc(current!1)(p!1) = 0
[-2,(rho)] y(next!1) = y(current!1)
[-3,(rho)] loc(next!1) = loc(current!1) WITH [(p!1) := 1]
{-4,(reachable invariant)}
  IF loc(current!1)(i!1)>2 THEN y(current!1)=0 ELSE TRUE ENDIF AND
  (i!1 = j!1 OR loc(current!1)(i!1)<3 OR loc(current!1)(j!1)<3)
  |------
[1,(rtp)]
  IF loc(current!1) WITH [(p!1) := 1] (i!1) > 2
    THEN y(current!1) = 0
  ELSE TRUE
  ENDIF
  AND
  (i!1 = j!1 OR
   loc(current!1) WITH [(p!1) := 1] (i!1) < 3 OR
   loc(current!1) WITH [(p!1) := 1] (j!1) < 3)

Rule? (grind)
This completes the proof of reachableInv.1.
reachableInv.5 :

{-1, (rho)} loc(current!1)(p!1) = 4
{-2, (rho)} y(next!1) = 1
{-3, (rho)} loc(next!1) = loc(current!1) WITH [(p!1) := 0]
{-4, (reachable invariant)}
   FORALL (i: PROC_ID): FORALL (j: PROC_ID):
      IF loc(current!1)(i) > 2 THEN y(current!1) = 0 ELSE TRUE ENDP
      AND (i = j OR loc(current!1)(i) < 3 OR loc(current!1)(j) < 3)
|--------
{1, (rtp)}
   IF loc(current!1) WITH [(p!1) := 0](i!1) > 2
      THEN FALSE
      ELSE TRUE
   ENDP
   AND
   (i!1 = j!1 OR
    loc(current!1) WITH [(p!1) := 0](i!1) < 3 OR
    loc(current!1) WITH [(p!1) := 0](j!1) < 3)
Rule? (inst = "i!1" "j!1")

[-1,(rho)]
   loc(current!1)(p!1) = 4
[-2,(rho)]
   y(next!1) = 1
[-3,(rho)]
   loc(next!1) = loc(current!1) WITH [(p!1) := 0]
{-4,(reachable invariant)}
   IF loc(current!1)(i!1)>2 THEN y(current!1)=0 ELSE TRUE ENDF AND
   (i!1 = j!1 OR loc(current!1)(i!1)<3 OR loc(current!1)(j!1)<3)
|--------
[1,(rtp)]
   IF loc(current!1) WITH [(p!1) := 0](i!1) > 2
      THEN FALSE
   ELSE TRUE
   ENDF
   AND
   (i!1 = j!1 OR
    loc(current!1) WITH [(p!1) := 0](i!1) < 3 OR
    loc(current!1) WITH [(p!1) := 0](j!1) < 3)
Rule? (grind)
this yields 2 subgoals:
reachableInv.5.1:

[-1, (rho)]
   \text{loc(current!1)(p!1) = 4}

[-2, (rho)]
   \text{y(next!1) = 1}

[-3, (rho)]
   \text{loc(next!1) = loc(current!1) WITH [(p!1) := 0]}

{-4} \text{ y(current!1) = 0}

{-5} \text{ i!1 = j!1}

{-6} \text{ loc(current!1)(j!1) > 2}
   |-------
{1} \text{ p!1 = j!1}

Rule?

next!1 violates reachable as \text{y(next!1) = 1 and loc(next!1)(j!1) > 2}.

current!1 violates reachable:
   \text{loc(current!1)(p!1) = 4 and loc(current!1)(j!1) > 2}.
Rule? (undo inst)

reachableInv.5 :
{-1,(rho)} \text{loc(current!1)(p!1)} = 4
{-2,(rho)} \text{y(next!1)} = 1
{-3,(rho)} \text{loc(next!1)} = \text{loc(current!1)} \text{WITH } [(p!1) := 0]
{-4,(reachable invariant)}
\quad \text{FORALL } (i: \text{PROC_ID}): \text{FORALL } (j: \text{PROC_ID}):
\quad \quad \text{IF } \text{loc(current!1)(i)} > 2 \text{ THEN } \text{y(current!1)} = 0 \text{ ELSE TRUE ENDIF}
\quad \quad \text{AND } (i = j \text{ OR } \text{loc(current!1)(i)} < 3 \text{ OR } \text{loc(current!1)(j)} < 3)
\quad \quad \text{|------}
{1,(rtp)}
\quad \text{IF } \text{loc(current!1)} \text{ WITH } [(p!1) := 0](i!1) > 2
\quad \quad \quad \text{THEN FALSE}
\quad \quad \text{ELSE TRUE}
\quad \quad \text{ENDIF}
\quad \quad \text{AND}
\quad \quad \quad (i!1 = j!1 \text{ OR}
\quad \quad \quad \quad \text{loc(current!1)} \text{ WITH } [(p!1) := 0](i!1) < 3 \text{ OR}
\quad \quad \quad \quad \quad \text{loc(current!1)} \text{ WITH } [(p!1) := 0](j!1) < 3)
Rule? (inst-cp - "i!1" "j!1")

[-1,(rho)] loc(current!1)(p!1) = 4
[-2,(rho)] y(next!1) = 1
[-3,(rho)] loc(next!1) = loc(current!1) WITH [(p!1) := 0]
[-4,(reachable invariant)]
    FORALL (i: PROC_ID): FORALL (j: PROC_ID): 
        IF loc(current!1)(i)>2 THEN y(current!1)=0 ELSE TRUE ENDIF
        AND (i = j OR loc(current!1)(i)<3 OR loc(current!1)(j)<3)
[-5,(reachable invariant)]
    IF loc(current!1)(i!1)>2 THEN y(current!1)=0 ELSE TRUE ENDIF AND
    (i!1 = j!1 OR loc(current!1)(i!1)< 3 OR loc(current!1)(j!1) < 3)
|--------
[1,(rtp)]
    IF loc(current!1) WITH [(p!1) := 0](i!1) > 2
        THEN FALSE ELSE TRUE
    ENDIF
    AND (i!1 = j!1 OR
        loc(current!1) WITH [(p!1) := 0](i!1) < 3 OR
        loc(current!1) WITH [(p!1) := 0](j!1) < 3)
Rule? (grind :if-match nil)

this yields 2 subgoals:

reachableInv.5.1 :

[-1,(rho)]
   \text{loc}(\text{current!1})(\text{p!1}) = 4

[-2,(rho)]
   \text{y}(\text{next!1}) = 1

[-3,(rho)]
   \text{loc}(\text{next!1}) = \text{loc}(\text{current!1}) \text{ WITH } [(\text{p!1}) := 0]

{-4,(reachable invariant)}
   \begin{align*}
   & \text{FORALL (i: PROC_ID):} \\
   & \quad \text{FORALL (j: PROC_ID):} \\
   & \qquad (i = j \text{ OR } \text{loc}(\text{current!1})(i) < 3 \text{ OR } \text{loc}(\text{current!1})(j) < 3)
   \end{align*}

{-5} \quad \text{y}(\text{current!1}) = 0

{-6} \quad \text{i!1} = \text{j!1}

{-7} \quad \text{loc}(\text{current!1})(\text{j!1}) > 2
   \quad |---

{1} \quad \text{p!1} = \text{j!1}
Rule? (inst = "p!1" "j!1")

[-1, (rho)]
    \text{loc(current!1)}(p!1) = 4

[-2, (rho)]
    y(next!1) = 1

[-3, (rho)]
    \text{loc(next!1) = loc(current!1) WITH [(p!1) := 0]}

{-4, (reachable invariant)}
    (p!1 = j!1 OR \text{loc(current!1)}(p!1) < 3 OR \text{loc(current!1)}(j!1) < 3)

[-5]  y(current!1) = 0

[-6]  i!1 = j!1

[-7]  \text{loc(current!1)}(j!1) > 2
    |--------
[1]  p!1 = j!1

Rule? (assert)

This completes the proof of reachableInv.5.1.
LTL framework

- A prelude library of LTL theories
- Includes definitions of state sequences, temporal operators, proofs of LTL proof rules
- Strategies manipulating LTL structures (e.g. \texttt{split-rho}) or applying proof rules (e.g. \texttt{binv})
State Sequences

- **State** is a type-consistent interpretation of the system variables \( V \)

- **State sequence** is an infinite sequence of states, represented as a mapping from time (\( \mathbb{N} \)) to states:

\[
\text{STATE_SEQ: TYPE = [TIME} \rightarrow \text{STATE]}
\]

(Recall: \( \sigma : s_0, s_1, s_2, \ldots \))

- **Assertions** are properties defined on individual states, without reference to their position in the state sequence.

\[
\text{ASSERTION: TYPE = [STATE} \rightarrow \text{bool]}
\]

Disjunction, conjunction, negation and implication over assertions are defined in the natural manner.
Example

\[ V = \{ a, b : \text{boolean} \} \]

Four distinct states, \( s^{00} : \langle a : 0, b : 0 \rangle \)
\( s^{01} : \langle a : 0, b : 1 \rangle \)
\( s^{10} : \langle a : 1, b : 0 \rangle \)
\( s^{11} : \langle a : 1, b : 1 \rangle \)

**STATE** type for this system is \( \{ s^{00}, s^{01}, s^{10}, s^{11} \} \)

State sequence \( S_{\text{opp}} \) defined as

\[ S_{\text{opp}} : [\text{TIME} \mapsto \text{STATE}] = \begin{array}{c|c}
0 & s^{01} \\
1 & s^{10} \\
2 & s^{01} \\
3 & s^{10} \\
\vdots \\
\end{array} \]

Assertion \( a_{\text{implies}} b \) is defined to be true in every state \( s \) in which \( a \rightarrow b \).
I.e. it is true of state \( s \) iff \( s \neq s^{10} \).

\( a_{\text{implies}} b \) is true at states \( S_{\text{opp}}(0), S_{\text{opp}}(2), \ldots \)
Lambda expression

Lambda (λ) expression denote unnamed functions. For example, the function which adds 3 to an integer may be written as

\[ \lambda(x: \text{int}): x + 3 \]

and defines a function of type \([\text{int} \mapsto \text{int}]\).

So, more formally,

\[
\text{S\_opp: STATE\_SEQ =}
\begin{align*}
(\lambda (t: \text{TIME}):
  \text{IF } &\exists (j: \text{TIME}): t = 2 \times j \\
  \text{THEN } &\# a := \text{FALSE}, b := \text{TRUE} \#
  \text{ELSE } &\# a := \text{TRUE}, b := \text{FALSE} \#
  \text{ENDIF}
\end{align*}
\]

The assertion \texttt{a\_implies\_b} is defined as

\[
\text{a\_implies\_b: ASSERTION =}
(\lambda (s: \text{STATE}): s'a \rightarrow s'b)
\]
Similarly, we can define

\[ a_{\text{and\_b}}: \text{ASSERTION} = (\lambda (s: \text{STATE}): s' a \land s' b) \]

\[ a_{\text{or\_b}}: \text{ASSERTION} = (\lambda (s: \text{STATE}): s' a \lor s' b) \]

Using conjunction and negation over assertions,

\[ a_{\text{xor\_b}}: \text{ASSERTION} = (\lambda (s: \text{STATE}): a_{\text{or\_b}}(s) \land \neg (a_{\text{and\_b}}(s))) \]
Temporal Properties

Temporal properties are interpreted over state sequences.

TP: TYPE = [STATE_SEQ, TIME → boolean]

E.g., the henceforth operator, $G$ (◻), is defined as

$$G: [TP → TP] =
(\lambda (a: TP):
  (\lambda (seq: STATE_SEQ), (j: TIME):
    \forall (t: TIME): t \geq j \rightarrow a(seq, t)))$$

That is, $G(a)$ holds at every position $j$ in $seq$ s.t. for all $t \geq j$, $a$ holds at state $seq(t)$.

There is automatic conversion from assertions to temporal properties. The temporal property is derived by evaluating the assertion at every state in the sequence:

$$\text{assertion\_to\_TP}(p: \text{ASSERTION}): \text{TP} =
(\lambda (seq: \text{STATE\_SEQ}), (t: \text{TIME}):
  p(seq(t)))$$

I.e. $p(seq, t) = p(seq(t))$
• \( \text{a\_or\_b}(S\_opp(t)) \)
  evaluates an assertion on state \( S\_opp(t) \).

\( \text{a\_or\_b}(S\_opp, t) \)
evaluates a temporal property at position \( t \) of \( S\_opp \)
\( \text{a\_or\_b} \) is converted into a temporal property

Both return the same value.

• Consider \( G(\text{a\_or\_b})(S\_opp, 0) \)

\( G(\text{a\_or\_b}) \) is a temporal property
It is evaluated at position \( 0 \) of \( S\_opp \).

\( G(\text{a\_or\_b})(S\_opp(0)) \)
is incorrectly typed: a temporal property cannot be converted to
an assertion, nor can it be evaluated at an individual state.

• Which of the following are true?
  
  - \( \text{a\_implies\_b}(S\_opp(0)) \)
  - \( \text{a\_implies\_b}(S\_opp, 0) \)
  - \( G(\text{a\_implies\_b})(S\_opp, 0) \)
  - \( \text{a\_implies\_b}(S\_opp, 1) \)
  - \( G(\text{a\_or\_b})(S\_opp, 1) \)
  - \( G(\text{not}(\text{a\_and\_b}))(S\_opp, 1) \)