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. `split-rho`) or applying proof rules (e.g. `binv`)

State Sequences

- A state is a type-consistent interpretation of the system variables V
- A state sequence is an infinite sequence of states, represented as a mapping from time ($\mathbb{N}$) to states:

  ```
  STATE_SEQ: TYPE = [TIME → 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.

  ```
  ASSERTION: TYPE = [STATE → bool]
  ```

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

Example

$V = \{a, b : \text{boolean}\}$

Four distinct states,

$$\begin{array}{ll} s^{00}: & \langle a : \mathrm{F},\ b : \mathrm{F}\rangle \\ s^{01}: & \langle a : \mathrm{F},\ b : \mathrm{T}\rangle \\ s^{10}: & \langle a : \mathrm{T},\ b : \mathrm{F}\rangle \\ s^{11}: & \langle a : \mathrm{T},\ b : \mathrm{T}\rangle \end{array}$$

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

State sequence S_opp defined as

$$S\_opp : [\mathtt{TIME} \mapsto \mathtt{STATE}] = \begin{array}{lll} 0 & \mapsto & s^{01} \\ 1 & \mapsto & s^{10} \\ 2 & \mapsto & s^{01} \\ 3 & \mapsto & s^{10} \\ & \cdots & \end{array}$$

Assertion a_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_implies_b is true at states $S\_opp(0), S\_opp(2), \ldots$

Lambda expression

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

$$\lambda(x : int) : x + 3$$

and defines a function of type [int $\mapsto$ int].

So, more formally,

```
S_opp: STATE_SEQ =
   (λ (t: TIME):
      IF ∃ (j: TIME): t = 2 × j
        THEN (# a := FALSE, b := TRUE #)
      ELSE (# a := TRUE, b := FALSE #)
      ENDIF)
```

The assertion a_implies_b is defined as

```
a_implies_b: ASSERTION =
   (λ (s: STATE): s‘a → s‘b)
```

Similarly, we can define

```
a_and_b: ASSERTION =
   (λ (s: STATE): s`a ∧ s`b)

a_or_b: ASSERTION =
   (λ (s: STATE): s`a ∨ s`b)
```

Using conjunction and negation over assertions,

```
a_xor_b: ASSERTION =
   (λ (s: STATE): a_or_b(s) AND NOT(a_and_b(s)))
```


Temporal Properties

Temporal properties are interpreted over state sequences.

```
TP: TYPE = [STATE_SEQ, TIME → boolean]
```

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

```
G: [TP → TP] =
   (λ (a: TP):
      (λ (seq: STATE_SEQ), (j: TIME):
         ∀ (t: TIME): t ≥ j → 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:

```
assertion_to_TP(p: ASSERTION): TP =
   (λ (seq: STATE_SEQ), (t: TIME):
      p(seq(t)))
```

I.e. $p(\text{seq}, t) = p(\text{seq}(t))$

- `a_or_b`(`S_opp`(t))
  evaluates an assertion on state `S_opp`(t).

  `a_or_b`(`S_opp`, t)
  evaluates a temporal property at position t of `S_opp`
  `a_or_b` is converted into a temporal property

  Both return the same value.

- Consider G(`a_or_b`)(`S_opp`, 0)

  G(`a_or_b`) is a temporal property
  It is evaluated at position 0 of `S_opp`.

  G(`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?

  - `a_implies_b`(`S_opp`(0))
  - `a_implies_b`(`S_opp`, 0)
  - G(`a_implies_b`)(`S_opp`, 0)
  - `a_implies_b`(`S_opp`, 1)
  - G(`a_or_b`)(`S_opp`, 1)
  - G(`not`(`a_and_b`))(`S_opp`, 1)