Semantics of Lustre programs

- A Lustre program is in essence a set of constraints between its input streams and output streams.
- These constraints operate in an algebra of streams.
- But they can also seen as Boolean and arithmetic constraints over instantaneous configurations of the program.

**Important observation:** A stream $x$ containing values of type $T$ is essentially a function $x : \mathbb{N} \rightarrow T$.
For each $n \in \mathbb{N}$,

$$x(n)$$

is the value of $x$ at position (or, *instant*) $n$. 

---

Let $L$ be a Lustre program. Let
\[ x_1, \ldots, x_p \]
be streams given in input to $L$, and
\[ x_{p+1}, \ldots, x_{p+q} \]
be the non-input (i.e., local and output) streams computed by $P$.

For each $n \in \mathbb{N}$, the tuple of values
\[ \langle x_1(n), x_2(n), \ldots, x_{p+q}(n) \rangle \]
is a configuration (of $L$ at instant $n$).
Instantaneous Configuration: Example

node counter (R:bool, X:int) returns (Y:bool);
var C: int;
let
    C = X -> if R then X else pre(C) + 1;
    Y = (C = 5);
tel;

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & \ldots \\
R &=& \text{false, false, false, true, false, false, \ldots} \\
X &=& \text{0, 4, 5, 1, 0, 11, \ldots} \\
C &=& \text{0, 1, 2, 1, 2, 3, \ldots} \\
Y &=& \text{false, false, false, false, false, false, \ldots} \\
\end{array}
\]

\[\langle R(3), X(3), C(3), Y(3)\rangle = \langle \text{true, 1, 1, false} \rangle\] is the configuration at instant 3.
node counter (R:bool, X:int) returns (Y:bool);
var C: int;
let
  C = X -> if R then X else pre(C) + 1;
  Y = (C = 5);
tel;

The program above can be seen as the following set of constraints for all $n \in \mathbb{N}$.

$$C(n) = \begin{cases} 
  X(n) & \text{if } n = 0 \\
  \text{else if } R(n) \text{ then } X(n) \\
  \text{else } C(n-1) + 1
\end{cases}$$

$$Y(n) = C(n) = 5$$
node test( X: bool ) returns ( P : bool );
var A, B : bool;
let
   A = X -> pre A;
   B = not (not X -> pre (not B));

--- A and B are identical streams
   P = A = B;
tel;

Conjecture: test always returns that constantly true stream.

How do we prove that?
Mathematically, the program test expresses, for all \( n \in \mathbb{N} \), the constraints set \( \Delta_n \):

\[
A(n) = \begin{cases} 
    \text{if } n = 0 \text{ then } X(n) \text{ else } A(n - 1) \\
    \text{else} 
\end{cases} \\
B(n) = \neg \left( \text{if } n = 0 \text{ then } \neg X(n) \text{ else } \neg B(n - 1) \right) \\
P(n) = A(n) = B(n)
\]

We want to show that \( P(n) = \text{true} \) for all \( n \in \mathbb{N} \).

To do that, we can reason by induction on \( n \):

1. First, we prove that \( P(0) \)’s value is always true.
2. Then, we prove that whenever \( P(n) \) is true for an arbitrary \( n \) then \( P(n + 1) \) is also true.
Induction proof:

Base case) Prove that $\Delta_0 \Rightarrow P(n)$
Induction Step) Prove that $\Delta_n \land \Delta_{n+1} \land P(n) \Rightarrow P(n+1)$

We have 3 possibilities.

1. Both the base case and the induction step hold. Then, we can conclude that $P$ is always true.
2. The base case does not hold. Then, clearly, $P$ is sometimes false.
3. The base case holds but the induction step does not. Then, we cannot conclude anything about $P$. 
Induction Proof: Example

\[ A(n) = \begin{cases} X(n) & \text{if } n = 0 \\ A(n - 1) & \text{else} \end{cases} \]

\[ \Delta_n: \quad B(n) = \begin{cases} \neg \big( \text{if } n = 0 \text{ then } \neg X(n) \text{ else } \neg B(n - 1) \big) & \end{cases} \]

\[ P(n) = A(n) = B(n) \]

**Base case** \( \Delta_0 \) is equivalent to:

\[ A(0) = X(0) \]
\[ B(0) = \neg \neg X(0) \]
\[ P(0) = A(0) = B(0) \]

Clearly, \( P(0) = \text{true} \)
Induction Proof: Example

\[ A(n) = \begin{cases} X(n) & \text{if } n = 0 \\ A(n-1) & \text{else} \end{cases} \]

\[ \Delta_n: \quad B(n) = \neg (\neg X(n) \text{ if } n = 0 \neg B(n-1)) \]

\[ P(n) = A(n) = B(n) \]

**Induction Step:** Assume that \( A(n), B(n), C(n) \) are defined as in \( \Delta_n \). \( \Delta_{n+1} \) is equivalent to:

\[ A(n+1) = A(n) \]

\[ B(n+1) = \neg (\neg B(n)) \]

\[ P(n+1) = A(n+1) = B(n+1) \]

If we assume that \( P(n) \) is true, it must be that \( A(n) = B(n) \).

But then, we can conclude that \( P(n+1) \) is true.
Limits of Simple Induction

node counter (R: bool) returns (P: bool);
  var C: int;
  let
    C = 0 -> if (R or pre(C) = 2) then 0
        else pre(C) + 1;

    P = C <= 4;
  tel;

Observe:

- C is never more than 2, so P is constantly true.
- However, simple induction is unable to prove that.
- The problem is that the induction step does not hold.
Why the induction step does not hold

\[
C(n) = \begin{cases} 
0 & \text{if } n = 0 \\
\text{else} & \text{if } R(n) \text{ or } C(n-1) = 2 \text{ then } 0 \text{ else } C(n-1) + 1 
\end{cases}
\]

\[
\Delta_n: \quad \text{if } R(n) \text{ or } C(n-1) = 2 \text{ then } 0 \text{ else } C(n-1) + 1
\]

\[
P(n) = C(n) \leq 4
\]

\[
\Delta_{n+1}: \quad C(n + 1) = \begin{cases} 
0 & \text{if } R(n+1) \text{ or } C(n) = 2 \text{ then } 0 \text{ else } C(n) + 1 \\
\text{else} & \text{if } R(n+1) \text{ or } R(n) \text{ or } R(n+1) \text{ to } false, \text{ we can satisfy } \Delta_n \wedge \Delta_{n+1} \wedge P(n) \text{ and falsify } P(n + 1).
\]

We need to show that the following implication holds:

\[
\Delta_n \wedge \Delta_{n+1} \wedge P(n) \Rightarrow P(n + 1) \quad (\ast)
\]

However, if we set, e.g., \( n \) to 10, \( C(n - 1) \) to 3, and \( R(n) \) and \( R(n + 1) \) to false, we can satisfy \( \Delta_n \wedge \Delta_{n+1} \wedge P(n) \) and falsify \( P(n + 1) \).
Why the induction step does not hold

\[
C(n) = \begin{cases} 
0 & \text{if } n = 0 \\
\text{else} & \begin{cases} 
0 & \text{if } R(n) \text{ or } C(n - 1) = 2 \\
C(n - 1) + 1 & \text{else}
\end{cases}
\end{cases}
\]

\[
\Delta_n: \begin{cases} 
0 & \text{if } R(n) \text{ or } C(n - 1) = 2 \\
\text{else} & C(n - 1) + 1
\end{cases}
\]

\[
P(n) = C(n) \leq 4
\]

\[
C(n + 1) = \begin{cases} 
0 & \text{if } R(n + 1) \text{ or } C(n) = 2 \\
\text{else} & C(n) + 1
\end{cases}
\]

\[
\Delta_{n+1}: \begin{cases} 
0 & \text{if } R(n + 1) \text{ or } C(n) = 2 \\
\text{else} & C(n + 1) \leq 4
\end{cases}
\]

Problem:

- a value of 3 for \( C(n - 1) \) is impossible in the program
- but the premise of (\( \ast \)) is not strong enough to rule it out
Why the induction step does not hold

\[ C(n) = \begin{cases} \text{if } n = 0 \text{ then } 0 \text{ else} \\ \Delta_n: \quad \text{if } R(n) \text{ or } C(n - 1) = 2 \text{ then } 0 \text{ else } C(n - 1) + 1 \\ P(n) = C(n) \leq 4 \end{cases} \]

\[ \Delta_{n+1}: \quad C(n+1) = \begin{cases} \text{if } R(n+1) \text{ or } C(n) = 2 \text{ then } 0 \text{ else } C(n) + 1 \\ P(n+1) = C(n+1) \leq 4 \end{cases} \]

**Problem:**
- a value of 3 for \( C(n - 1) \) is impossible in the program
- but the premise of (\( \ast \)) is not strong enough to rule it out

**Solution:**
- look at a few more preceding configurations
**$k$-induction: Induction with Depth**

Fix some $k \geq 0$

**Base case)** Prove that

$$\Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$$

**Induction Step)** Prove that

$$\Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$$

We have again 3 possibilities:
$k$-induction: Induction with Depth

Fix some $k \geq 0$

**Base case** Prove that

$$\Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$$

**Induction Step** Prove that

$$\Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$$

We have again 3 possibilities:

1. Both the base case and the induction step hold. Then, we can conclude that $P$ is always true.
\textit{k}-\textit{induction: Induction with Depth}

Fix some $k \geq 0$

**Base case**) Prove that

$$\Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$$

**Induction Step**) Prove that

$$\Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$$

We have again 3 possibilities:

2. The base case does not hold.
   Then, $P$ is false for some $m \in \{0, \ldots, k\}$. 

$k$-induction: Induction with Depth

Fix some $k \geq 0$

**Base case)** Prove that

$$\Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$$

**Induction Step)** Prove that

$$\Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$$

We have again 3 possibilities:

3. The base case holds but the induction step does not. Then, we cannot conclude anything about $P$.

But we can increase $k$ and start again.
\[
C(n) = \begin{cases} 
0 & \text{if } n = 0 \\
\text{else} & \end{cases}
\]

\[
\Delta_n: \quad \begin{cases} 
0 & \text{if } R(n) \text{ or } C(n - 1) = 2 \\
\text{else} & C(n - 1) + 1
\end{cases}
\]

\[
P(n) = C(n) \leq 4
\]

\[
\Delta_{n+1}: \quad \begin{cases} 
0 & \text{if } R(n + 1) \text{ or } C(n) = 2 \\
\text{else} & C(n) + 1
\end{cases}
\]

\[
P(n + 1) = C(n + 1) \leq 4
\]

With $k$-induction we can prove that $P$ is always true.

**Exercise:** Find the smallest value of $k$ that will do.
The $k$-induction Procedure

1: $k := 0$;
2: while true do
3: check validity of
   $\Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$;
4: if counter-example found then
5:    return counter-example
6: end if;
7: check validity of
   $\Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$;
8: if valid then
9:    return "Property holds"
10: end if;
11: $k := k + 1$;
12: end while
Features of the $k$-induction Procedure

- When $\Delta$ contains no multiplications, the validity tests in lines 3 and 7 can be performed completely automatically.

- The induction procedure is **sound**: if it says that the property holds, then the property does hold.

- However, the procedure is still **incomplete**: for some properties that do hold it may loop forever.

- The procedure can be made **complete** for some (large) classes of Lustre programs, including **finite state** ones.

- However, it is **impossible** to make the procedure complete (and still automatic) for all Lustre programs.
node counter2 (R, X: bool) returns (P: bool);
var C: int; let
  C = 0 -> if (R or pre(C) = 2) then 0
       else pre(C) + 1;
  P = X or (C <= 4);
tel;

Observe:

- Similar to counter but now \( P \) is \( X \) or \( C \leq 4 \) instead of \( C \leq 4 \), with \( X \) an additional input stream.

- \( P \) is always \textit{true} but \( k \)-induction is \textit{unable} to prove that for any \( k \).

- For each \( k \), there is a counter-example for the induction step, e.g., \( n = 10 \), \( C(n - 1) = 4 \), \( X(n) = \text{true} \), \ldots, \( X(n + k) = \text{true} \), and \( X(n + k + 1) = \text{false} \).
A Simplifying Assumption

Let us consider only Lustre programs where $\text{pre}$ applies only to variables.

**Note:** This is with no loss of generality. For example, the first program below can be rewritten equivalently into the second:

```plaintext
code node Foo (X,Y: int) returns (Z:int);
let
    Z = 0 -> pre (X + Y);
tel;

code node FooNorm (X,Y: int) returns (Z:int);
var U: int
let
    U = X + Y;
    Z = 0 -> pre(U);
tel;
```
If \( L \) is a Lustre program, let \( S \) be the tuple of \( L \)'s state variables, non-input variables that occur within a pre.

**Example.** \( S = \langle A, C \rangle \) for this program:

```plaintext
node test( X: bool ) returns ( P : bool );
var A, B, C : bool;
let
  A = X -> pre A;
  B = not (not X -> pre(C));
  C = not B;
  P = A = B;
tel;
```

The value \( S_n \) that the tuple \( S \) has at some instant \( n \) is the *state of \( L \) at instant \( n \).*
We can make \( k \)-induction less incomplete, by considering only configurations with distinct states.

Let \( D_{0,k} \) be the formula stating that the states \( S_0, \ldots, S_k \) are pairwise distinct. (And similarly for \( D_{n,n+k+1} \)).

We can use

**Base case)**
\[
D_{0,k} \land \Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)
\]

**Induction step)**
\[
D_{n,n+k+1} \land \Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)
\]
The $k$-induction Procedure with Distinct States

1: $k := 0$;
2: while true do
3:   check validity of $D_{0,k} \land \Delta_0 \land \cdots \land \Delta_k \Rightarrow P(0) \land \cdots \land P(k)$;
4:   if counter-example found then
5:     return counter-example
6:   end if;
7:   check validity of $D_{n,n+k+1} \land \Delta_n \land \cdots \land \Delta_{n+k+1} \land P(n) \land \cdots \land P(n+k) \Rightarrow P(n+k+1)$;
8:   if valid then
9:     return "Property holds"
10: end if;
11: $k := k + 1$;
12: check validity of $\Delta_0 \land \cdots \land \Delta_k \Rightarrow \neg D_{0,k}$;
13: if valid then
14:   return "Property holds"
15: end if;
16: end while
Adding the distinct states restriction to \( k \)-induction preserves its soundness.

It makes it complete for programs where every legal execution sequence with pairwise distinct states is shorter than some positive integer \( d \).

This is the case, for instance, for \textit{finite state programs}, programs whose state variables can take only finitely many values.

But it is also the case for some infinite state programs like \texttt{counter2}. 