Synchronous Dataflow Programming

CS684: Embedded Systems
Topic 2

Paritosh Pandya

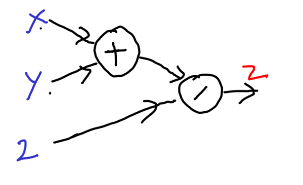
Indian Institute of Technology, Bombay

January 31, 2021

Synchronous Dataflow Programs in Lustre/Heptagon

A network of Operators connected by named wires.

Example

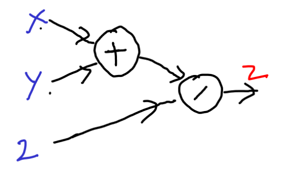



```
node MEAN(X,Y: int)
        returns (Z:int)
  let
    Z =  (X + Y) / 2 ;
  tel
```

Synchronous Dataflow Programs in Lustre/Heptagon

A network of Operators connected by named wires.

Example




```
node MEAN(X,Y: int)
        returns (Z:int)
  let
    Z =  (X + Y) / 2 ;
  tel
```

Semantics

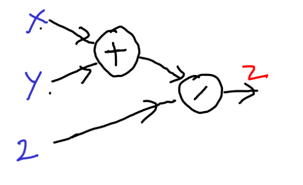
Semantics: X, Y, Z are discrete flows.

- Time is discrete. Given by $\mathbb{N}$. (e.g. clock cycles in circuits).
- Flow $X : \mathbb{N} \rightarrow Val_X$. (almost)
- $\forall t \in \mathbb{N} : \quad Z_t \;=\; (X_t + Y_t)/2$ (pointwise application)

Synchronous Dataflow Programs in Lustre/Heptagon

A network of Operators connected by named wires.

Example




```
node MEAN(X,Y: int)
        returns (Z:int)
  let
    Z =  (X + Y) / 2 ;
  tel
```

Simulation

X	1	2	1	$\cdots$
Y	3	4	1	$\cdots$
Z	2	3	2	$\cdots$
$Clock$	0	1	2	$\cdots$

$$Z_i \;=\; (X_i + Y_i)/2, \quad \forall i \in Clock$$

Set of Equations Defining Variables

Equivalent Programs

```
node MEAN(X,Y: int)
        returns (Z:int)
  let
    Z =  (X + Y) / 2 ;
  tel
```

```
node MEAN(X,Y: int)
        returns (Z:int)
  var W : int;
  let
    Z =  W / 2 ;
    W =  X + Y ;
  tel
```

Structure of Lustre Node

```
node <NodeName> ( <Inputflows with Types> )
             returns ( <Outputflows with Types> )
 var <Internalflows with types>;
 let
    V1 = EXPR1;
    ...
    Vn = EXPRn;
 tel
```

- Set of equations, one for each output or local flow (variable).
- Declarative – order not important.
- Semantics: Order the equations in data dependancy order and then compute at each reaction.
- All operators are applied pointwise.
- Causality ensures deterministic behaviour – unique output for each input.

Language Elements

- Primitive Data types bool, int, real
- Expressions and equations
  - constants 2 gives the flow 2,2,2,2
  - combinational operators arithmetic, logical, ...
    $Z = X + Y$ applied pointwise $Z_i = X_i + Y_i, \ \forall i$.
  - `if < bexpr> then <expr> else <expr>`

Memory/Delay

pre X

X	x_0	x_1	x_2	$\cdots$
$pre(X)$	nil	x_0	x_1	$\cdots$
$Clock$	0	1	2	$\cdots$
$pre(pre(X))$	nil	nil	x_0	$\cdots$

$$\begin{aligned} (pre(X))_0 &= nil \\ (pre(X))_i &= X_{i-1}, \quad \forall i > 0 \end{aligned}$$

Initialization $X -\rangle Y$

X Y	x_0 y_0	x_1 y_1	x_2 y_2	$\cdots$ $\cdots$
$X\text{->}Y$	x_0	y_1	y_2	$\cdots$
$Clock$	0	1	2	$\cdots$
$X\text{->}pre(Y)$	x_0	y_0	y_1	$\cdots$

$$\begin{aligned} (X\text{->}Y)_0 &= X_0 \\ (X\text{->}Y)_i &= Y_i, \quad \forall i > 0 \end{aligned}$$

Examples of Equations

- Counter `X = 0 -> (pre(X) + 1)`
- Fibonnachi `Z = 1 -> pre(Z -> (Z + pre(Z)))`
- Edge

```
node EDGE(X:bool) returns (Y:bool)
    let
      Y = false -> X and not pre X ;
    tel
```

- Counter with Reset Counts no of occurrences of X. Resets when R occurs.

```
node COUNTER(X,R:bool) returns (N:int)
let
  N = 0 -> if R then 0
           else if X then (pre N)+1
                else (pre N);
tel
```

Modularity: Node as Operator

```
node MINMAX(X:int)
          returns (min,max:int)
let
  min = X -> if (pre(min) < X) then
                pre(min) else X;
  max = X -> if (X < pre(max)) then
             (pre max) else X;
tel
```

Tools

Compilation:
lustre file.lus nodename

Simulation
luciole file.lus nodename

```
node AvgMINMAX(X:int)
             returns (Z:real)
var U,V:int;
let
   Z = Average(U,V);
   U,V = MINMAX(X);
tel
```

```
Node Average(X,Y:int
        returns (Z:real)
let
 Z = (real(X)+real(Y))/2.
tel
```

Causality

Causally incorrect Programs

- `X = X` Circular definition.
- `X = Y; Y = X` Indirect Circular Definition
- `X = pre(X)` Causality ok but failur of initialization.
- `X = 0 -> pre(X)` Correct
- Syntactic causality Failure.

```
X = if C then A else Y;
Y = if C then X else B
```

Equivalent causally correct program.

```
X = if C then A else B;
Y = if C then A else B
```

More Examples

Inverse Z transform?
Stopwatch?

Heptagon

- Extension of Lustre. Similar to commercial language SCADE.
- New data types: enumeration, structures, array iterators, Generic nodes, Automata.
- Enumeration `type Tlight = Red | Yellow | Green`
- Structured Records `type complex = { re: real; im: real}`
- Arrays `type Req: bool^5`

See Heptagon Manual for how to read elements of strucutured types and how to modify its value.

Arrays

- Array type *BaseType^Indextype* E.g. `Req : bool^5`
  Static index access `Req[0]`, `Req[4]`.
  Dynamic index access `Req.[x] default false`.
- Array modification `[ Req with [x] = false ]`.
- Array constructor `[1,x,3,y,5]`
- Array slice `Req[1:3]` gives array of size 3.
- Array slice catenation: `A1@A2`

Array Slices Example

```
node rotate() returns (y0,y1,y2,y3:int)
var z: int^4;
let
  z = ([5,6,7,8]) fby ([z[3]]@z[0..2]);
  (y0,y1,y2,y3) = (z[0],z[1],z[2],z[3]);
tel
```

Global Types and Constants

```
const n : int = 3
const v1 : int^n = [2,3,5]

node examplearrayslice()   returns (z : int^n)
let
   z = ( (v1) fby ([z[n-1]]@z[0..n-2]));
tel

node display() returns (y0,y1,y2:int)
var z: int^n;
let
  z = examplearrayslice();
  (y0,y1,y2) = (z[0],z[1],z[2]);
tel
```

Parameterized Nodes and Static Genericity

We can pass static (compile time) parameters to nodes.

```
const n : int = 10
const t0 : float^n = 1.0^n

node TRANSVEC<<m:int; t1: int^n>>(a:int^m) = (b:int^m)
let
o = map<<m>> (+)(a, t1);
tel

node SHIFT(a:int^n) = (o:int^n)
let
o = f<<n, t0>>(a);
tel
```

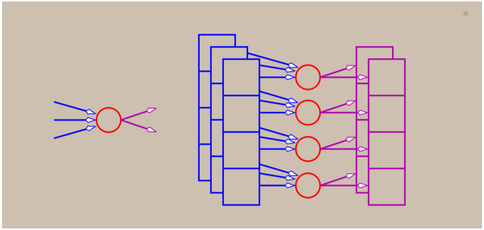
Array Iterators

Objectives: To manipulate arrays iteratively.

- Given vectors $A : int^n$ and $B : int^n$, add them up to give vector $C : int^n$. Use map.
- Given array $A : int^n$, find sum of its elements. Use fold.
  We can also compute $\Sigma(A[i]^2)$ and use this to find standard deviation.
- Mapfold combines the map and the fold.

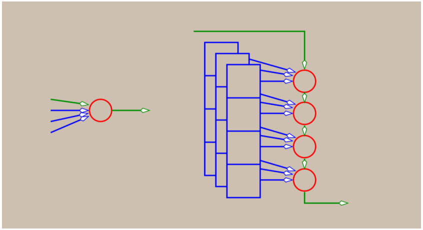
Map

- Example: Adding two 3-dimentional vectors `a,b:real^3` to get `c:real^3`.
  Method: Use $+$ pointwise on every index accumulating the sum.
- `c = map<<3>>(+) ([1,3,5],[4,3,-2])` gives `[5,6,3]`
- In general `map<<n>>(F)` $(x_1, ..., x_m)$ returns $(y_1, ..., y_k)$
  Here $F : (t_1 \times \ldots \times t_m) \rightarrow t'_1 \times \ldots \times t'_k$.
  Also, $x_i : t_i\hat{\ }n$ for $1 \leq i \leq m$ and $y_j : t'_j\hat{\ }n$ for $1 \leq j \leq k$.



Fold

- Example: Finding sum of array of 4 elements `a:int^4` to get `c:int`.
  Method: Use $+$ pointwise on every index accumulating the sum.
- `c = fold<<4>>(+) ([1,3,5,7],0)` gives 16
- In general `fold<<n>>(F)` $(x_1, \ldots, x_m, z)$ returns y
  Here $F : (t_1 \times \ldots \times t_m \times t) \rightarrow t$.
  Also, $x_i : t_i\hat{}n$ for $1 \leq i \leq m$ and $z, y : t$.



Mapfold

- Example: Adding two 3-dimentional vectors `a,b:real^3` AND getting their dot-product `c:real^3; dot:real`
- `node F(a,b,c:real) returns (d,e:real)  let d=a+b; e=c + (`
- `c = mapfold<<3>>(F) ([1,3,5],[4,3,-2],0)` gives `[5,6,3], 3`
- In general `map<<n>>(F)` $(x_1, ..., x_m)$ returns $(y_1, ..., y_k)$
  Here $F : (t_1 \times \ldots \times t_m) \rightarrow t'_1 \times \ldots \times t'_k$.
  Also, $x_i : t_i\hat{\ }n$ for $1 \leq i \leq m$ and $y_j : t'_j\hat{\ }n$ for $1 \leq j \leq k$.