1-synchronous clocks, underspecified clocks and non-determinism

Guillaume Iooss, Albert Cohen, Marc Pouzet

ENS - PARKAS

April 29, 2019
Context of the presentation

- Link with the previous presentation:
  - Front-end in the previously presented compilation chain
  - Based on the synchronous compiler *Heptagon*
  - Orthogonal to the architecture used
Context of the presentation

- Link with the previous presentation:
  - Front-end in the previously presented compilation chain
  - Based on the synchronous compiler *Heptagon*
  - Orthogonal to the architecture used

- In relation to Lopht:
  - Manage the harmonic multi-periodic aspect
  - Normalization of the input Lustre program + annotations

- Other motivations:
  - Make specification easier to write manually in Lustre
  - Using more information which could be specified
Background - Synchronous language

- Manipulate infinite flow of values
- Global tick synchronize the production of values
- Point-to-point operators
- Accessing past values possible ("fby" ≈ memory)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>4</td>
<td>−2</td>
<td>1</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>...</td>
</tr>
<tr>
<td>x + y</td>
<td>4</td>
<td>−1</td>
<td>2</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>42 fby y</td>
<td>42</td>
<td>4</td>
<td>−2</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
**Background - Clocks**

- A stream might have no value on a tick
- **Clock**: $x :: clk$
  - Encode the presence of a value
  - Can be an arbitrary boolean stream
- Temporal operators: sub-sampling (when) and fusion (merge)
- Clocking analysis: check coherency of clocks

<table>
<thead>
<tr>
<th>$x$ :: $c$</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ :: $c$</td>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>...</td>
</tr>
<tr>
<td>$z = x$ when $b$ :: $c$ on $b$</td>
<td>0</td>
<td>$-$</td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>$y$ :: $c$ on not $b$</td>
<td>$-$</td>
<td>42</td>
<td>$-$</td>
<td>$-$</td>
<td>...</td>
</tr>
<tr>
<td>merge $b z y$ :: $c$</td>
<td>0</td>
<td>42</td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>
Background - Lustre

- Equational language for synchronous programs (similar languages: Scade, Heptagon, ...)

```
node accumulator(i : int) returns (o : int)
var x : int
let
    x = 0 fby o;
    o = x + i;
tel
```
Background - Lustre

- Equational language for synchronous programs
  (similar languages: Scade, Heptagon, ...)

```lustre
node accumulator(i : int) returns (o : int)
var x : int
let
  x = 0 fby o;
  o = x + i;
tel
```

- **Code generation:**
  - "reset" and "step" functions
  - Infinite "while" loop (1 iteration = 1 base tick)
  - Clocks: encoded using "if" conditions
Background - N-synchronous model

- **N-synchronous model:**
  - Ultimately periodic clocks
  - Example: 101(1001)
  - Strictly periodic: no initialization phase

⇒ Clocking analysis becomes more predictable
Background - N-synchronous model

- **N-synchronous model:**
  - Ultimately periodic clocks
  - Example: 101(1001)
  - Strictly periodic: no initialization phase

⇒ Clocking analysis becomes more predictable

- **buffer:** Communication between variables on two different clocks
  - Clocks must be compatible (adaptability relation: <:)
  ⇒ Able to compute the size of a buffer
1-synchronous clocks

Consider integration program:
Top-level node, orchestrating all tasks of an application
- Multiple harmonic periods (ex: 5 ms / 10 ms / 20 ms / …)
- Tasks are present only once per period
1-synchronous clocks

- Consider integration program:
  - Top-level node, orchestrating all tasks of an application
    - Multiple harmonic periods (ex: 5 ms / 10 ms / 20 ms / …)
    - Tasks are present only once per period

- 1-synchronous clocks: "\(0^k10^{n-k-1}\)" (or "\(0^k(10^{n-1})\)"
  with \(0 \leq k < n\), \(n = \text{period}\) and \(k = \text{phase}\)
1-synchronous clocks

- Consider integration program:
  Top-level node, orchestrating all tasks of an application
  - Multiple harmonic periods (ex: 5 ms / 10 ms / 20 ms / ...)
  - Tasks are present only once per period

- 1-synchronous clocks: 
  
  \[(0^k 10^{n-k-1})\] (or \[0^k(10^{n-1})\])

  with \(0 \leq k < n\), \(n = \) period and \(k = \) phase

- Integration program: only 1-synchronous clocks are used
  \(\leadsto\) Can use that condition to do more inside a compiler
In this talk

Three incremental modifications on top of Lustre:

1. Restriction of the clock calculus to 1-synchronous clocks
   - Specialization of the N-synchronous clocks
   - Associated specialized clocking rules
   - Code generation possibilities (Hyperperiod Expansion)
In this talk

Three incremental modifications on top of Lustre:

1. **Restriction of the clock calculus to 1-synchronous clocks**
   - Specialization of the N-synchronous clocks
   - Associated specialized clocking rules
   - Code generation possibilities (Hyperperiod Expansion)

2. **Phases of the clock of some variables are not specified**
   - Kahn semantic satisfied, dataflow semantic not
   - Constraints on phases obtained from clocking rules
   - Solution used to go back to fully-specified Lustre program
In this talk

Three incremental modifications on top of Lustre:

1. Restriction of the clock calculus to 1-synchronous clocks
   - Specialization of the N-synchronous clocks
   - Associated specialized clocking rules
   - Code generation possibilities (Hyperperiod Expansion)

2. Phases of the clock of some variables are not specified
   - Kahn semantic satisfied, dataflow semantic not
   - Constraints on phases obtained from clocking rules
   - Solution used to go back to fully-specified Lustre program

3. Non-deterministic computation
   - Don’t mind which instance of a value used
   - Neither semantics are satisfied
   - More freedom for phase selection
   - Go back to deterministic program
1-synchronous clock calculus - Same period

- Clock calculus restricted to 1-synchronous clocks.
  - What happens to temporal operators?
1-synchronous clock calculus - Same period

- Clock calculus restricted to 1-synchronous clocks.  
  \(\leadsto\) What happens to temporal operators?

- **buffer**: phase not specified \(\leadsto\) not yet
- **delay**: increment the phase of the clock / \(\text{delay}(d) = \text{delay}^d\)
  - Should not cross the period (no initialization)
    \[
    H \vdash a :: (0^k10^{n-k-1}) \quad 0 \leq d < n - k
    \]
    \[
    H \vdash \text{delay}(d) a :: (0^{k+d}10^{n-(k+d)-1})
    \]
Clock calculus restricted to 1-synchronous clocks.

What happens to temporal operators?

(buffer: phase not specified \(\leadsto\) not yet)

delay: increment the phase of the clock / \(\text{delay}(d) = \text{delay}^d\)

Should not cross the period (no initialization)

\[ H \vdash a :: (0^k10^{n-k-1}) \quad 0 \leq d < n - k \]

\[ H \vdash \text{delay}(d) a :: (0^{k+d}10^{n-(k+d)-1}) \]

delayfby(d): (initialization required / \(\approx\) "short fby")

\[ H \vdash a :: (0^k10^{n-k-1}) \quad H \vdash i :: (0^{k+d-n}10^{n-(k+d-n)-1}) \quad 0 \leq k + d - n < n \]

\[ H \vdash i \text{ delayfby}(d) a :: (0^{k+d-n}10^{n-(k+d-n)-1}) \]
Toward slower periods (when)

Clocks must be 1-synchronous + subclock condition:

⇒ Harmonicity condition

⇒ Argument of the when must be of the form \((F^k TF^{n-k-1})\)
Toward slower periods (when)

Clocks must be 1-synchronous + subclock condition:

⇒ Harmonicity condition

⇒ Argument of the when must be of the form "\((F^k TF^{n-k-1})\)"

\[
\begin{align*}
q \times n + k & \quad m = pn \quad l = qn + k \\
H \vdash a :: (0^k 10^{n-k-1}) \quad H \vdash a \text{ when } (F^q TF^{p-1-q}) :: (0^l 10^{m-l-1})
\end{align*}
\]
Toward faster periods (merge/current)

Clocks must be 1-synchronous + subclock condition:

⇒ Harmonicity condition

- **merge**: one branch per instance of fast period
- **current** (repetition of a value, with eventual updates)
  - Argument (when the update occurs) must be \( (F^k TF^{n-k-1}) \)
  - Initialization needed ("i")
Toward faster periods (merge/current)

Clocks must be 1-synchronous + subclock condition:

\[ \Rightarrow \text{Harmonicity condition} \]

- **merge**: one branch per instance of fast period
- **current** (repetition of a value, with eventual updates)
  - Argument (when the update occurs) must be "\((F^k \cdot TF^{n-k-1})\)"
  - Initialization needed ("i")

\[
H \vdash a :: (0^k 10^{n-k-1}) \\
H \vdash i :: (0^l 10^{m-l-1}) \\
n = pm \\
l = k - mq
\]

\[
H \vdash \text{current}((F^q \cdot TF^{p-1-q}), i, a) :: (0^l 10^{m-l-1})
\]
Code generation

- Use 1-synchronous restriction to generate efficient code
  - Know exactly when the activation will happen
  - All "buffer" are of size 1 → memory cell
Code generation

- Use 1-synchronous restriction to generate efficient code
  - Know exactly when the activation will happen
  - All "buffer" are of size 1 \(\sim\) memory cell

- Three code generation schemes:
  - Classical step function (base clock)
    - If conditions
  - One step function per phase (base clock)
    - No if conditions / while loop looping on them in order
  - One step function for the whole period (slowest clock)
    \(\Rightarrow\) Hyperperiod expansion transformation
Hyperperiod expansion - Example

**Idea:** change base period to a slower one (ex: scm of all periods)  
⇒ (duplicate fast computation)
**Hyperperiod expansion - Example**

**Idea:** change base period to a slower one (ex: scm of all periods)  
⇒ (duplicate fast computation)

**Example:**

<table>
<thead>
<tr>
<th>Input:</th>
<th>x :: (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local:</td>
<td>a :: (1), b :: (10)</td>
</tr>
<tr>
<td></td>
<td>a = f(x);       // f stateless</td>
</tr>
<tr>
<td></td>
<td>b = g(a when (10)); // g stateless</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input:</th>
<th>x₀, x₁ :: (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local:</td>
<td>a₀, a₁, b :: (1)</td>
</tr>
<tr>
<td></td>
<td>a₀ = f(x₀);</td>
</tr>
<tr>
<td></td>
<td>a₁ = f(x₁);</td>
</tr>
<tr>
<td></td>
<td>b = g(a₀);</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Hyperperiod expansion - More details

Transformed equation gives a set of equations. Intuitions:

- \( r(\text{Var}) \in \mathbb{N}^* \): ratio between \( \text{Var} \)'s period and slowest period
- Variable duplication: \( \text{Var} \sim \text{Var}_0, \ldots, \text{Var}_{r(\text{Var})-1} \)
- Applied on a normalized program
- Each equation is duplicated \( r(\text{lhsVar}) \) times
Hyperperiod expansion - More details

Transformed equation gives a set of equations. Intuitions:

- $r(Var) \in \mathbb{N}^*$: ratio between Var’s period and slowest period
- Variable duplication: $Var \sim Var_0, \ldots, Var_{r(Var)-1}$
- Applied on a normalized program
- Each equation is duplicated $r(lhsVar)$ times

Some interesting rules (informally written):

- $a = op(b1, \ldots, bm) \Rightarrow a_i = op(b1_i, \ldots, bm_i)$ for $0 \leq i < r$
Hyperperiod expansion - More details

Transformed equation gives a set of equations. Intuitions:

- \( r(Var) \in \mathbb{N}^* \): ratio between \( Var \)'s period and slowest period
- Variable duplication: \( Var \sim Var_0, \ldots, Var_{r(Var)-1} \)
- Applied on a normalized program
- Each equation is duplicated \( r(lhsVar) \) times

Some interesting rules (informally written):

- \( a = op(b1, \ldots, bm) \Rightarrow a_i = op(b1_i, \ldots, bm_i) \) for \( 0 \leq i < r \)
- \( a = i \ fby \ b \Rightarrow a_0 = i \ fby \ b_{r-1} \mid a_i = b_{i-1} \) for \( 1 \leq i < r \)
Hyperperiod expansion - More details

Transformed equation gives a set of equations. Intuitions:

- \( r(\text{Var}) \in \mathbb{N}^* \): ratio between Var's period and slowest period
- Variable duplication: \( \text{Var} \sim \text{Var}_0, \ldots, \text{Var}_{r(\text{Var})-1} \)
- Applied on a normalized program
- Each equation is duplicated \( r(\text{lhsVar}) \) times

Some interesting rules (informally written):

- \( a = \text{op}(b_1, \ldots, b_m) \Rightarrow a_i = \text{op}(b_{1i}, \ldots, b_{mi}) \) for \( 0 \leq i < r \)
- \( a = i \text{ fby } b \Rightarrow a_0 = i \text{ fby } b_{r-1} \mid a_i = b_{i-1} \) for \( 1 \leq i < r \)
- \( a = b \text{ when } (F^p \top F^{n-p-1}) \Rightarrow a_i = b_{p+i\times n} \) for \( 0 \leq i < r(a) \)
Hyperperiod expansion - More details

Transformed equation gives a set of equations. Intuitions:

- \( r(Var) \in \mathbb{N}^* \): ratio between Var’s period and slowest period
- Variable duplication: \( Var \leadsto Var_0, \ldots, Var_{r(Var) - 1} \)
- Applied on a normalized program
- Each equation is duplicated \( r(lhsVar) \) times

Some interesting rules (informally written):

- \( a = op(b_1, \ldots, bm) \Rightarrow a_i = op(b_{1i}, \ldots, b_{mi}) \) for \( 0 \leq i < r \)
- \( a = i \ fby \ b \Rightarrow a_0 = i \ fby \ b_{r-1} \mid a_i = b_{i-1} \) for \( 1 \leq i < r \)
- \( a = b \) when \( (F^p \ T \ F^{n-p-1}) \Rightarrow a_i = b_{p+i\times n} \) for \( 0 \leq i < r(a) \)
- \( a = \text{current}((F^p \ T \ F^{n-p-1}), \ \text{init}, \ b) \)

\[
\Rightarrow \begin{cases} 
    a_i = \text{init}_i \ fby \ b_{r(b)-1} & \text{for } 0 \leq i < p \\
    a_i = b_{\left\lfloor \frac{i-p}{n} \right\rfloor} & \text{for } p \leq i < r(a)
\end{cases}
\]
Hyperperiod expansion - Discussion

- Positive points:
  - Get rid of the multi-periodic aspect
  - Natural way to manage long tasks (with no cutting)
  - Decouple the phases of different instances of a variable
Hyperperiod expansion - Discussion

- **Positive points:**
  - Get rid of the multi-periodic aspect
  - Natural way to manage long tasks (with no cutting)
  - Decouple the phases of different instances of a variable

- **Negative points:**
  - Stateless functions needed
    (If stateful, need to expose the internal state and pass it
     + reset function to get initial state
     + at annotation to reuse the memory of states)
  - Additional real-time constraints needed on inputs/outputs
    (release/deadline)
The problem with phases

- Phases = large-grain schedule across the periods
  - "Good" choice of phases is architecture dependent
    (sequential: WCET balancing / parallel: ...more complicated)
The problem with phases

- Phases = large-grain schedule across the periods
  - "Good" choice of phases is architecture dependent
    (sequential: WCET balancing / parallel: ...more complicated)
- Phase computation is tedious to write and modify:
  - One phase modification impacts many equations
  - Humanly impossible for large applications
The problem with phases

- Phases = large-grain schedule across the periods
  - "Good" choice of phases is architecture dependent
    (sequential: WCET balancing / parallel: ...more complicated)
- Phase computation is tedious to write and modify:
  - One phase modification impacts many equations
  - Humanly impossible for large applications

⇒ Choice of phases should be separated from the computation
The problem with phases

- Phases = large-grain schedule across the periods
  → "Good" choice of phases is architecture dependent
    (sequential: WCET balancing / parallel: ...more complicated)
- Phase computation is tedious to write and modify:
  - One phase modification impacts many equations
  - Humanly impossible for large applications
  ⇒ Choice of phases should be separated from the computation

**Modification proposed:**
- Option to only define the period of some local variables
- Implicit buffers operator (clock of rhs <: clock of lhs)

**Compilation flow:**
- Clocking analysis gathers the constraints on phase
- Solver finds a solution (given cost function)
- Use this solution to explicit phases and buffer (→ delay)
Extracting constraints from clocking rules

- **buffer**: delay of an unknown length
  - $(0^k10^{n-k-1}) <: (0^l10^{m-l-1})$ iff $m = n$ and $k \leq l$

  \[
  H \vdash a : (0^k10^{n-k-1}) \quad 0 \leq k \leq l < n
  \]

  \[
  H \vdash \text{buffer} \ a : (0^l10^{n-l-1})
  \]
Extracting constraints from clocking rules

- **buffer**: delay of an unknown length
  \[ (0^k10^{n-k-1}) <: (0^l10^{m-l-1}) \text{ iff } m = n \text{ and } k \leq l \]

\[
H \vdash a :: (0^k10^{n-k-1}) \quad 0 \leq k \leq l < n
\]

**buffer** by:
additional initialization (period crossed)

Variations of buffer with other constraints:
  - buffer which fixes its phase (ex: \( p \leq 3 \))
  - buffer which constraint the latency (ex: \( p_B - p_A \leq 3 \))
Example of clock extraction

a, e :: period(1);
b, d :: period(2);
c :: period(6);
b = buffer f_1(a when (FT));
c = buffer f_2(b when (TFF));
d = buffer f_3(current((FFT), 0, c))
e = buffer f_4(current((TF), 0, d))
Example of clock extraction

\[
\begin{align*}
\text{a, e} &:: \text{period}(1); \\
\text{b, d} &:: \text{period}(2); \\
\text{c} &:: \text{period}(6); \\
\text{b} &= \text{buffer } f_1(\text{a when (FT)}); \\
\text{c} &= \text{buffer } f_2(\text{b when (TFF)}); \\
\text{d} &= \text{buffer } f_3(\text{current( (FFT), 0, c)}); \\
\text{e} &= \text{buffer } f_4(\text{current( (TF), 0, d)}); \\
\end{align*}
\]

- **Bounds from variable declaration:**
  \[0 \leq p_a, p_e < 1 / 0 \leq p_b, p_d < 2 / 0 \leq p_c < 6\]

- **Constraints from buffer:**
  \[p_a + 1 \leq p_b / p_b \leq p_c / p_c - 4 \leq p_d / p_d \leq p_e\]

- **Solutions:**
  \[p_a = p_e = 0 / p_b = 1 / p_d = 0 / 1 \leq p_c \leq 4\]
Solving the constraints (1)

- **Solving:**
  - Constraint form allows efficient solving
  - Issue: Constraints for the cost function have a different form
Solving the constraints (1)

- **Solving:**
  - Constraint form allows efficient solving
  - Issue: Constraints for the cost function have a different form

- **Use case:** flight control application
  (6k nodes, 30k data, 4 harmonic periods)
  - Sequential case: load balancing across phases
    (task weight = its WCET)
  - Direct ILP formulation of the problem tricky possible
    (Introduce boolean variable $\delta_{T,k}$ for the phases)
    $\Rightarrow$ Does not scale...
Solving the constraints (1)

- **Solving:**
  - Constraint form allows efficient solving
  - Issue: Constraints for the cost function have a different form

- **Use case:** flight control application
  (6k nodes, 30k data, 4 harmonic periods)
  - Sequential case: load balancing across phases
    (task weight = its WCET)
  - Direct ILP formulation of the problem tricky possible
    (Introduce boolean variable $\delta_{T,k}$ for the phases)
    \[ \Rightarrow \text{Does not scale...} \]
  - ILP formulation with only boolean variable
    \[ \Rightarrow \text{First integral solution found after 40 mins} \]
    - Good solution, non-optimal, but takes too much time
Solving the constraints (2)

- Using an ILP is an overkill
  - In this context, no need for an optimal solution
  - A "good enough" solution is enough
Solving the constraints (2)

- Using an ILP is an overkill
  - In this context, no need for an optimal solution
  - A "good enough" solution is enough

- **Heuristic:**
  - Initial solution: smallest valid phases for all nodes
  - Decrease toward local minimum:
    - Soft push (moving a phase without moving the rest)
    - Intermediate data structure $\rightarrow$ quick evaluation of solution
Solving the constraints (2)

- Using an ILP is an overkill
  - In this context, no need for an optimal solution
  - A "good enough" solution is enough

- **Heuristic:**
  - Initial solution: smallest valid phases for all nodes
  - Decrease toward local minimum:
    - Soft push (moving a phase without moving the rest)
    - Intermediate data structure → quick evaluation of solution

⇒ **Result:** decreasing takes less than a second
  0,6% above the rational average
Solving the constraints (2)

- Using an ILP is an overkill
  - In this context, no need for an optimal solution
  - A "good enough" solution is enough

- **Heuristic:**
  - Initial solution: smallest valid phases for all nodes
  - Decrease toward local minimum:
    - Soft push (moving a phase without moving the rest)
    - Intermediate data structure → quick evaluation of solution

⇒ **Result:** decreasing takes less than a second
  0,6% above the rational average

- **Reinjection step:**
  - Complete the clocks of local variables
  - Replace all buffer with delay (or remove them)
Non-deterministic computation

- Physical values with low temporal variability
  - Ex: outside temperature
  - Want last value, but not strict requirement (older one ok)
  - Constraint on phase can be relaxed

⇒ Express and use ND to give more freedom to the compiler

Wanted constraint: $p_a + 2 \leq p_b$
(instead of $p_a + 4 \leq p_b$)
Non-deterministic computation

- Physical values with low temporal variability
  - Ex: outside temperature
  - Want last value, but not strict requirement (older one ok)
  - Constraint on phase can be relaxed

⇒ Express and use ND to give more freedom to the compiler

| ? |
|---|---|---|---|
|   |   |   |   |

Wanted constraint: \( p_a + 2 \leq p_b \)
(instead of \( p_a + 4 \leq p_b \))

- How to express notion in a minimal way in the language?
Non-deterministic operator: fby?

- **Proposition:** operator "fby?" to control non-determinism
Non-deterministic operator: fby?

- **Proposition:** operator "fby?" to control non-determinism
- Value of (i fby? expr) can be:
  - expr
  - or (i fby expr)
- **Analysis:**
  - Clocking: same rule than fby
  - Initialization: no issue
  - Causality: conservatively assume no fby
Non-deterministic operator: \texttt{fby}?

- **Proposition:** operator "\texttt{fby}?" to control non-determinism
- Value of \((i \ \texttt{fby}? \ expr)\) can be:
  - expr
  - or \((i \ \texttt{fby} \ expr)\)

- **Analysis:**
  - Clocking: same rule than \texttt{fby}
  - Initialization: no issue
  - Causality: conservatively assume no \texttt{fby}

- Value of \((i \ \texttt{fby}?^n \ expr)\) can be:
  - expr
  - or \((i \ \texttt{fby}^k \ expr)\) (with \(0 \leq k \leq n\))
Non-deterministic operator: $fby?$

- **Proposition:** operator "$fby?$" to control non-determinism
- Value of $(i \ fby? \ expr)$ can be:
  - $expr$
  - or $(i \ fby \ expr)$

- **Analysis:**
  - Clocking: same rule than $fby$
  - Initialization: no issue
  - Causality: conservatively assume no $fby$

- Value of $(i \ fby?^n \ expr)$ can be:
  - $expr$
  - or $(i \ fby^k \ expr)$ (with $0 \leq k \leq n$)

- **Determinization pass:** Replace all $fby?$ by a possible value (in our case: fix that depending on its phase)
Constraint extraction with non-determinism

\[ y = (i \ fby?^2 \ x) \ \text{when (FFT)} \]

\[ y = i \ fby?^2 \ \text{current}((\text{TFF}), 0, x) \]
Constraint extraction with non-determinism

\[ y = (i \ fby^2 \ x) \text{ when } (\text{FFT}) \quad y = i \ fby^2 \ \text{current}((\text{TFF}), 0, x) \]

- Typing analysis: rule for \( fby? \) doesn’t give any constraint
  - Recognize \( fby? \) under a \( \text{when} \) & above a \( \text{current} \)
    - Typing rules for these specific situations
- Other option: defining \( \text{when}? \) and \( \text{current}? \) operators
In summary...

- 3 incremental extensions:
  - 1-synchronous clocks
  - ...with unknown phases
  - ...with non-deterministic computation

- Hyperperiod expansion transformation

- Constraints on phase can be inferred from the clocking rules

- Non-deterministic operator & adaptation of constraints
In summary...

- 3 incremental extensions:
  - 1-synchronous clocks
  - ... with unknown phases
  - ... with non-deterministic computation

- Hyperperiod expansion transformation

- Constraints on phase can be inferred from the clocking rules

- Non-deterministic operator & adaptation of constraints

- Thank you for listening, ...
  - ... Do you have any questions?