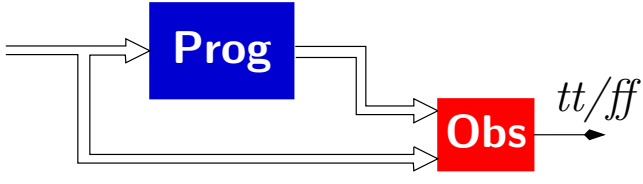
From Discrete Duration Calculus to Synchronous Observers

Laure Gonnord
Nicolas Halbwachs
Pascal Raymond

Vérimag, Grenoble

Specifying (safety) properties with synchronous observers



Well-known advantages:

- **maximal power**
- **same language for programs and properties**
- **specifications are "executable"**

Higher-level formalisms sometimes needed

Specification formalisms (e.g., temporal logics)

- **simulation needed**
- **is decidability an issue?**

Decision procedures for TL

$$\varphi \longrightarrow \mathcal{A}_\varphi$$

$$\sigma \in \mathcal{L}(\mathcal{A}_\varphi) \iff \sigma \models \varphi$$

$$\varphi \text{ satisfiable} \iff \mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset$$

Easy if $\mathcal{A}_\varphi$ rational

Model-checking of TL

$$\boldsymbol{P} \models \varphi \Longleftrightarrow \boldsymbol{\mathcal{L}}(\boldsymbol{P} \times \boldsymbol{\mathcal{A}}_{\neg\varphi}) = \emptyset$$

Model-checking of TL

$$P \models \varphi \Longleftrightarrow \mathcal{L}(P \times \mathcal{A}_{\neg\varphi}) = \emptyset$$

Rationality of $\mathcal{A}_\varphi$ is of little practical interest, since P has generally an infinite number of states

Model-checking of TL

$$P \models \varphi \Longleftrightarrow \mathcal{L}(P \times \mathcal{A}_{\neg\varphi}) = \emptyset$$

Rationality of $\mathcal{A}_\varphi$ is of little practical interest, since P has generally an infinite number of states

Even in the finite-state case, the size of the automaton can be prohibitive.

Model-checking of TL

$$P \models \varphi \iff \mathcal{L}(P \times \mathcal{A}_{\neg\varphi}) = \emptyset$$

Rationality of $\mathcal{A}_\varphi$ is of little practical interest, since P has generally an infinite number of states

Even in the finite-state case, the size of the automaton can be prohibitive.

$\longrightarrow$ symbolic automata, possibly extended with variables (counters)

Translating high level specification languages into symbolic automata

(synchronous observers, safety properties)

$\sigma \models \varphi$ **iff** 

Translating high level specification languages into symbolic automata

(synchronous observers, safety properties)

$\sigma \models \varphi$ **iff**

- **executable**
- **use in verification, runtime verification, testing...**
- **no explosion**
- **not necessarily finite state (counters...)**

A well-known case: REGLO [Raymond96]
Translation of regular expressions into symbolic acceptors (Lustre observers)
Strictly linear size

A new experience: QDDC [Pandya 2000-03]

$\longrightarrow$ one conclusion: there are common (safety) properties which are surprizingly hard to express with observers, i.e., hard to check "on the fly"

QDDC: Quantified Discrete Duration Calculus

Examples

"Whenever p has been true continuously during at least n steps, q holds"

$$\Box\left((\lceil\!\lceil p \rceil\!\rceil \wedge \eta \geq n) \Rightarrow true^{\frown}\lceil q \rceil^{0}\right)$$

or $p \xrightarrow{n} q$

"In any interval of duration d, p holds at least k times"

$$\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$$

QDDC: Syntax

Propositions (on states)

$$P ::= 0 \mid 1 \mid p \mid \neg P \mid P \wedge P \mid P \vee P$$

Formulas (on traces and windows)

$$\varphi ::= \lceil P \rceil^0 \mid \lceil\lceil P \rceil\rceil \mid \eta \; op \; c \mid \Sigma P \; op \; c \mid$$

$$\varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists p \; \varphi \mid \varphi_1 \frown \varphi_2$$

$$(op \in \{\leq, <, =, \neq, >, \geq\})$$

QDDC: Semantics

States: sets of basic propositions (or Boolean valuations of propositional symbols) — models of propositions: $s \models P$

Traces: finite sequences of states

$$\sigma = \sigma_1\sigma_2 \ldots \sigma_n \qquad |\sigma| = n$$

Windows: intervals in a trace

$$\sigma[b, e] = \sigma_b\sigma_{b+1} \ldots \sigma_e \quad 1 \leq b \leq e \leq |\sigma|$$

QDDC: Semantics (cont.)

$$
\begin{array}{lll}
\boldsymbol{\sigma}[b,e] \models \lceil P \rceil^0 & \textbf{iff} & b = e \textbf{ and } \boldsymbol{\sigma}_b \models P \\
\boldsymbol{\sigma}[b,e] \models \lceil\lceil P \rceil & \textbf{iff} & b < e \textbf{ and } \forall i, b \leq i < e, \boldsymbol{\sigma}_i \models P \\
\boldsymbol{\sigma}[b,e] \models \eta \; op \; c & \textbf{iff} & (e-b) \; op \; c \\
\boldsymbol{\sigma}[b,e] \models \Sigma P \; op \; c & \textbf{iff} & Card\,\{i | b \leq i < e, \boldsymbol{\sigma}_i \models P\} \; op \; c \\
\boldsymbol{\sigma}[b,e] \models \exists p \, \varphi & \textbf{iff} & \exists \boldsymbol{\sigma}', \; \boldsymbol{\sigma}'[b,e] \models \varphi, \\
 & & \textbf{and } \forall i, \forall q \neq p, \boldsymbol{\sigma}_i(q) = \boldsymbol{\sigma}'_i(q) \\
\boldsymbol{\sigma}[b,e] \models \varphi_1 \frown \varphi_2 & \textbf{iff} & \exists i, b \leq i \leq e, \\
 & & \boldsymbol{\sigma}[b,i] \models \varphi_1 \textbf{ and } \boldsymbol{\sigma}[i,e] \models \varphi_2
\end{array}
$$

$\boldsymbol{\sigma} \models \varphi$ **iff** $\boldsymbol{\sigma}[1, |\boldsymbol{\sigma}|] \models \varphi$

Decidable, as long as constants are known

(no parameters)

QDDC: Derived operators

$\lceil\!\lceil P \rceil\!\rceil = \lceil\!\lceil P \rceil\!\rceil^\frown \lceil P \rceil^0$ **(right-closed interval)**

$\Diamond\varphi = true^\frown \varphi^\frown true$ **(eventually φ)**

$\Box\varphi = \neg\Diamond\neg\varphi$ **(always φ)**

$P_1 \xrightarrow{c} P_2 = \neg\Diamond\left((\lceil\!\lceil P_1 \rceil\!\rceil \wedge \eta \geq c)^\frown \lceil \neg P_2 \rceil^0\right)$

(whenever P_1 has been continuously true during c steps, P_2 is true)

Our extensions: **(forget decidability)**

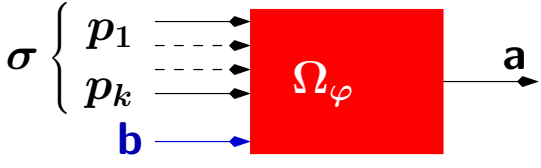
- **Constants c (in $\eta \; op \; c$, $\Sigma P \; op \; c$, $P_1 \xrightarrow{c} P_2$) can be symbolic (parameters)**
- **The atomic propositions can be conditions on parameters (e.g., $c_1 \geq c_2$)**

Observers of formulas (in Lustre)

inputs:

- σ **(sequence of values for the propositions)**
- **a "starter"** b **(a Boolean, true only once)**

output a=true at t **iff** $\sigma, [t_b, t] \models \varphi$



Example 1: $\lceil\lceil p \rceil\rceil$

True of each interval $[t_b, t]$ where p always holds

The observer output should

- be true before the (unique) occurrence of b

- take the value of p when b

- remain true as long as p is true

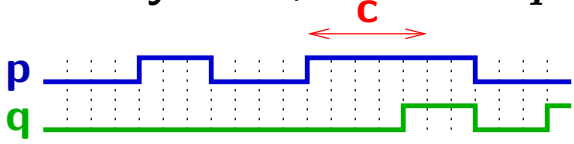
a = before(b) or (if b then p else p and pre(a))

where

before(b) = not b -> (not b and pre(before(b)))

Example 2: $p \xrightarrow{c} q$

True on $[t_b, t]$ iff all subinterval of length c where p is always true, end with q true



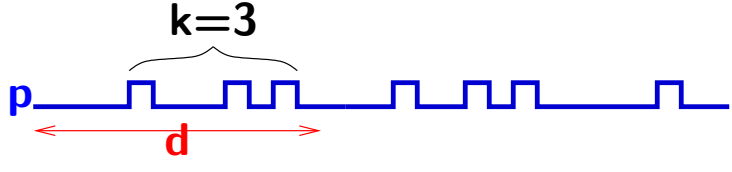

```
a = before(b) or
        ((true -> pre(a)) and (age(p)>=c => q))
```

where

```
age(p) = if p then (0-> pre(age(p))+1 else 0
```

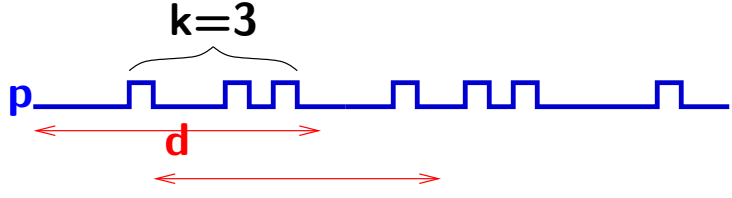
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



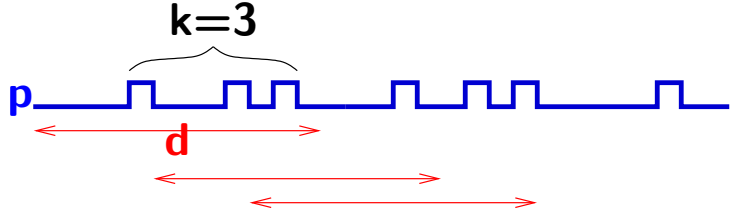
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



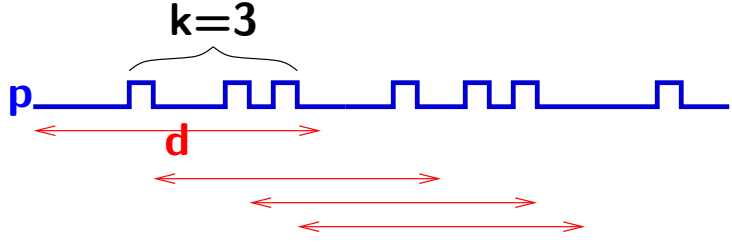
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



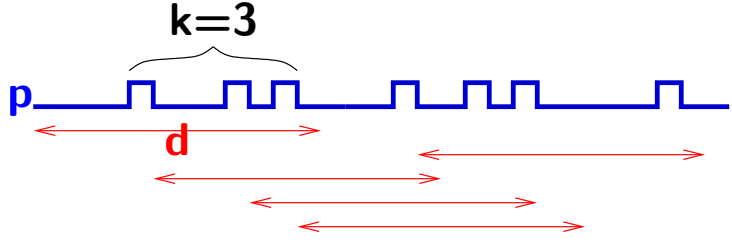
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



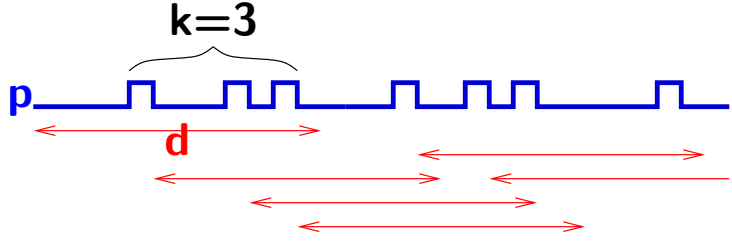
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



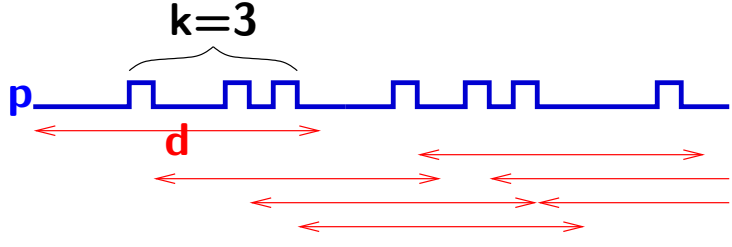
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



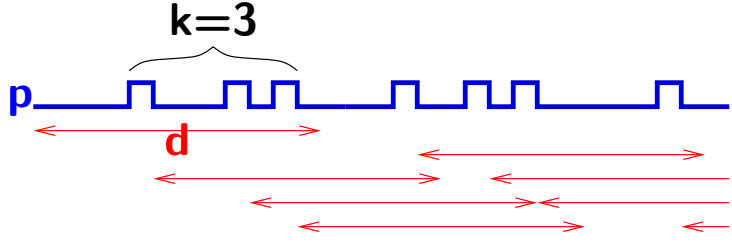
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



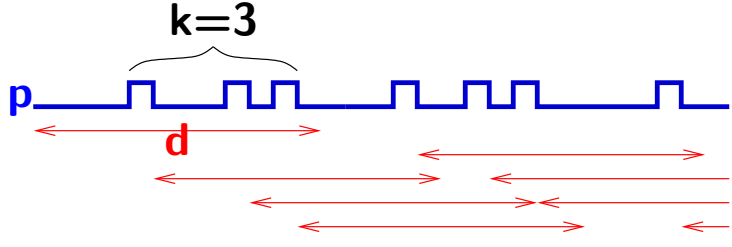
Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



Example 3: $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

Each interval longer than d contains at least k occurrences of p



k (or d) counters needed!

Problem if d and k are parameters...

(for testing, one could use generic arrays. Out of reach of verification tools...)

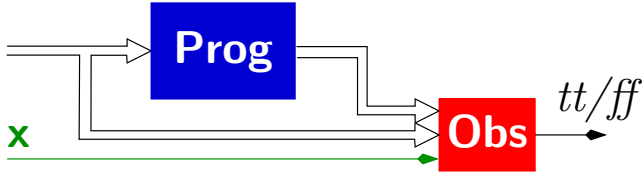
A solution: non-deterministic observers

Idea: non deterministically start a counter at each instant — by means of an additional input, say x (oracle).

A solution: non-deterministic observers

Idea: non deterministically start a counter at each instant — by means of an additional input, say x (oracle).

Since a model-checker verifies that the output is always true whatever be the inputs, it will verify whatever be x, so, for all intervals.



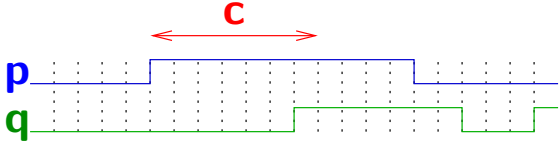
Example 3 (revisited): $\Box(\eta \geq d \Rightarrow \Sigma p \geq k)$

```
a = before(b) or
      ((true -> pre(a)) and length<d or nb_p ≥ k)
length = nb_since(true,x)
nb_p = nb_since(p,x)

where
 nb_since(b1,b2) =
      if before(b2) then 0
      else if b2 then if b1 then 1 else 0
      else pre(nb_since(b1,b2)) +
           if b1 then 1 else 0
```

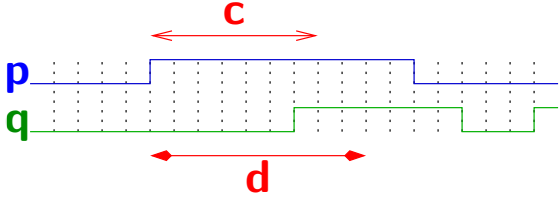
An example of verification

Prove that $\left(p \xrightarrow{c} q \wedge d \geq c\right) \Longrightarrow (p \xrightarrow{d} q)$



An example of verification

Prove that $\left(p \xrightarrow{c} q \wedge d \geq c\right) \Longrightarrow (p \xrightarrow{d} q)$



An example of verification (cont.)

Prove that, whatever be p, q, c, d and x, the output a is always true

```
a1 = before(x) or                    -- p -c-> q
     ((true -> pre(a1)) and (nb_since(p,x)<c or q)
a2 = before(x) or                    -- p -d-> q
     ((true -> pre(a2)) and (nb_since(p,x)<d or q)
-- (p -c-> q / d ≥ c) => p -d-> q
a = (a1 and d ≥ c) => a2;
```

An example of verification (cont.)

Prove that, whatever be p, q, c, d and x, the output a is always true

```
a1 = before(x) or                    -- p -c-> q
     ((true -> pre(a1)) and (nb_since(p,x)<c or q)
a2 = before(x) or                    -- p -d-> q
     ((true -> pre(a2)) and (nb_since(p,x)<d or q)
-- (p -c-> q / d ≥ c) => p -d-> q
a = (a1 and d ≥ c) => a2;
```

Instantly proved, both by Lesar and Nbac

A fragment of QDDC
recognizable by non-deterministic acceptors

Why only a fragment?

- **Liveness properties expressible in QDDC**
- **Oracles can only be universally quantified**
- **Formulas $\exists p\, \varphi$ and $\varphi_1 \frown \varphi_2$ need existentially quantified oracles**

3-levels syntax

$$\varphi ::= \neg\psi$$

$$\psi ::= \xi \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \exists \boldsymbol{p}\ \psi \mid \psi_1 \frown \psi_2$$

$$\xi ::= \lceil \boldsymbol{P} \rceil^0 \mid \lceil\lceil \boldsymbol{P} \rceil \mid \boldsymbol{\eta}\ op\ \boldsymbol{c} \mid \boldsymbol{\Sigma P}\ op\ \boldsymbol{c} \mid \neg\xi$$

Example: $\Box(\eta > c \Rightarrow \Sigma p \geq d)$

Translation into basic QDDC:

$$\neg\,(true^\frown(\eta > c \wedge \Sigma p < d)^\frown true)$$

Translation into Lustre:

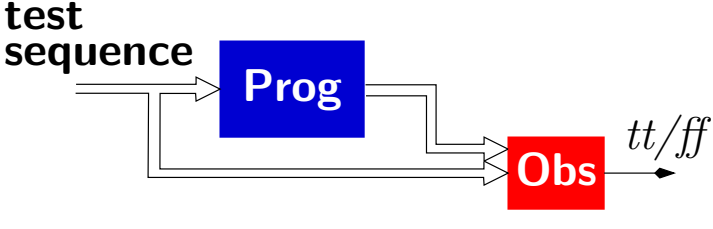
```
a = not (   nb_since(true,x) > c and
            nb_since(p,x) < d )
```

Ok for verification (since the verification tools universally quantify over oracles)

Not suitable for property simulation, nor for testing

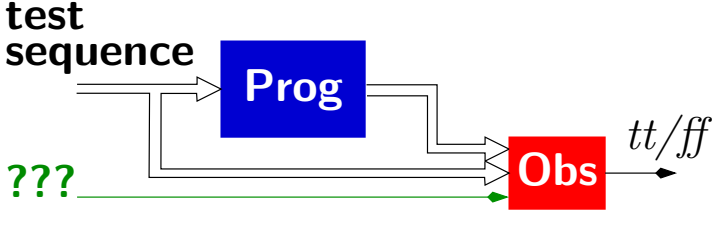
Ok for verification (since the verification tools universally quantify over oracles)

Not suitable for property simulation, nor for testing



Ok for verification (since the verification tools universally quantify over oracles)

Not suitable for property simulation, nor for testing



A deterministic fragment

Sources of non-determinism:

- $\exists p\ \varphi$: **completely non-deterministic, hopeless**
- $\varphi_1 \frown \varphi_2$: **often used in a deterministic way**
  ex. $\lceil\!\lceil P \rceil\!\rceil \frown \lceil\!\lceil Q \rceil\!\rceil^0$

$\longrightarrow$ **replace** $\varphi_1 \frown \varphi_2$ **by** $\varphi_1 \; then \; \varphi_2$**:**

the interval consists of a maximal interval satisfying φ_1, followed by an interval satisfying φ_2

φ_1 restricted to formulas that can only change from $true$ to $false$.

Two-levels syntax

$$\varphi \ ::= \ \psi \mid \mathit{end}(\boldsymbol{P}) \mid \psi \ \mathit{then} \ \varphi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi$$

$$\psi \ ::= \ \mathit{begin}(\boldsymbol{P}) \mid \lceil\!\lceil \boldsymbol{P} \rceil\!\rceil \mid \boldsymbol{\eta} \leq \boldsymbol{c} \mid \boldsymbol{\Sigma P} \leq \boldsymbol{c} \mid \mathit{age}(\boldsymbol{P}) \leq \boldsymbol{c} \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2$$

Examples

$$\lceil\!\lceil \boldsymbol{P} \rceil\!\rceil^\frown \lceil \boldsymbol{Q} \rceil^0 \ \equiv \ \lceil\!\lceil \boldsymbol{P} \rceil\!\rceil \wedge \mathit{end}(\boldsymbol{Q})$$

$$\boldsymbol{p} \xrightarrow{\boldsymbol{c}} \boldsymbol{q} \ \equiv \ \mathit{age}(\boldsymbol{p}) \leq \boldsymbol{c} \vee \boldsymbol{q}$$

$$\lceil\!\lceil \boldsymbol{P} \rceil\!\rceil^\frown \lceil\!\lceil \mathit{begin}(\neg \boldsymbol{P}) \wedge \boldsymbol{Q} \rceil\!\rceil \ \equiv \ \lceil\!\lceil \boldsymbol{P} \rceil\!\rceil \ \mathit{then} \ \lceil\!\lceil \boldsymbol{Q} \rceil\!\rceil$$

Example of translation

$$\lceil\!\lceil P \rceil\!\rceil^\frown \lceil Q \rceil^0 \equiv \lceil\!\lceil P \rceil\!\rceil \wedge end(Q)$$

Obs$_{\lceil\!\lceil P \rceil\!\rceil}$ **= after(b) and pre(always_from(Obs**$_P$**,b))**

Obs $_{end(Q)}$ **= after(b) and Obs**$_Q$

Obs$_{\lceil\!\lceil P \rceil\!\rceil^\frown \lceil Q \rceil^0}$ **= Obs**$_{\lceil\!\lceil P \rceil\!\rceil}$ **and Obs** $_{end(Q)}$

Conclusion

- **There are common safety properties which are not obvious to translate into symbolic acceptors**
- **Non-deterministic acceptors increase the descriptive power**
- **What about really complex formalisms (SUGAR...)?**