# Contracts for modular discrete controller synthesis

Gwenaël Delaval[1] Hervé Marchand[1] Éric Rutten[2]

[1]INRIA Rennes, VerTeCs project

[2]INRIA Rhône-Alpes, Pop-Art project

December 4th, 2008 — Aussois

# Introduction

Motivation: introduction of discrete controller synthesis into a modular compilation process

Modularity motivations:

- Easier usability from a programmer's point of view
- Scalability (critical for methods implying state space exploration)
- Dealing with abstract components/IP blocks/...

# Outline

# Discrete controller synthesis: principle

## Goal

Enforcing a temporal property $\Phi$ on a system (on which $P$ does not a priori hold)

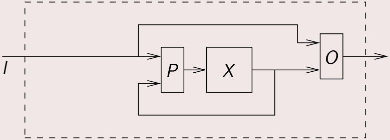
# Discrete controller synthesis: principle

## Goal

Enforcing a temporal property $\Phi$ on a system (on which $P$ does not a priori hold)

## Principle (on implicit equational representation)

| | |
|---|---|
| $X$ | memory |
| $P$ | transition function |
| $O$ | output function |



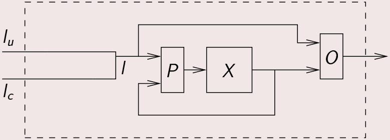
# Discrete controller synthesis: principle

## Goal

Enforcing a temporal property $\Phi$ on a system (on which $P$ does not a priori hold)

## Principle (on implicit equational representation)

| | |
|---|---|
| $X$ | memory |
| $P$ | transition function |
| $O$ | output function |



- Partition of inputs into controllable ($I_c$) and uncontrollable ($I_u$) inputs

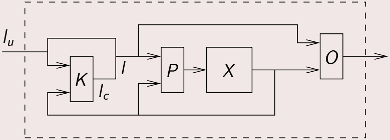
# Discrete controller synthesis: principle

## Goal

Enforcing a temporal property $\Phi$ on a system (on which $P$ does not a priori hold)

## Principle (on implicit equational representation)

| | |
|---|---|
| $X$ | memory |
| $P$ | transition function |
| $O$ | output function |



- Partition of inputs into controllable ($I_c$) and uncontrollable ($I_u$) inputs
- Computation of a controller $K(X, I, I_c)$ such as the system controlled by $K$ satisfies $\Phi$

# DCS tool: Sigali

Use of an existing tool, Sigali (INRIA Rennes, VerTeCs and Espresso)

From:

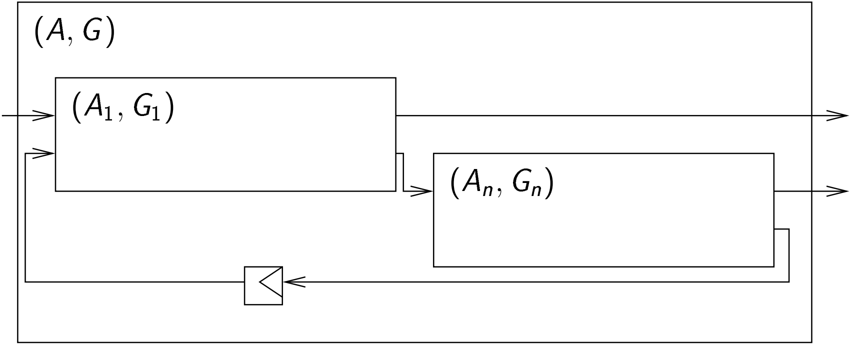
- a polynomial dynamic system (PDS) $S$, with

$$S(X, I) = \begin{cases} X' = P(X, I) \\ Q_0(X) \end{cases}$$

- a partition $I = I_u \uplus I_c$
- an invariance property $\Phi = \forall\Box G$

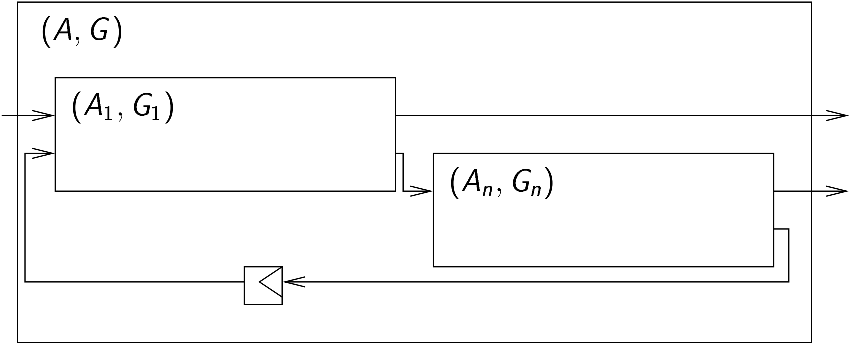
Sigali will compute $K = \mathrm{DCS}(S, I_c, \Phi)$, $K(X, I_u, I_c)$ being the most permissive controller for $S$ satisfying $\Phi$

# Contracts and validation

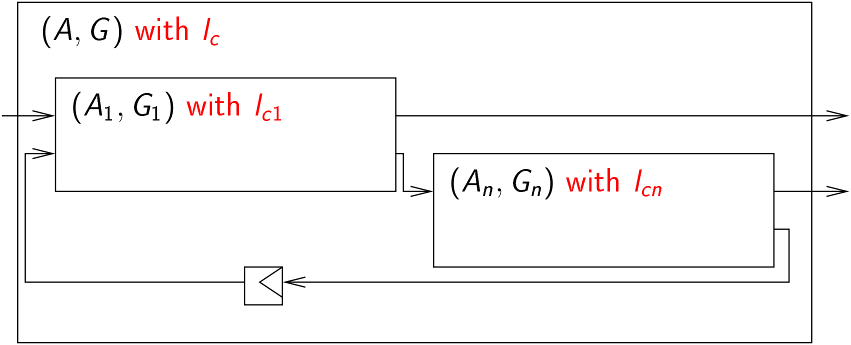


From the environment hypothesis $\Box A_i \Rightarrow \Box G_i$, $i \in \{1, \ldots, n\}$,
check that $\Box A \Rightarrow \Box G$

# Proposal: contracts and DCS

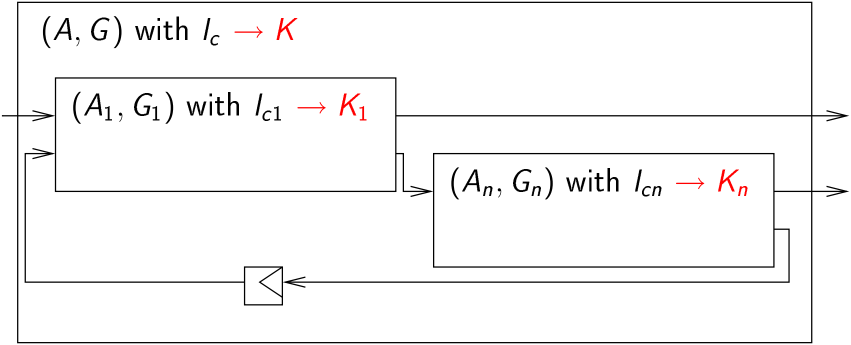


# Proposal: contracts and DCS



- To each contract, associate controllable additional variables, local to the component

# Proposal: contracts and DCS



- To each contract, associate controllable additional variables, local to the component
- Compute a local controller for each component

# Language extension

Extension of the Heptagon/MiniLustre language (LRI, Demons) with a contract syntax:

```
node f($x_1$,...,$x_n$) returns ($y_1$,...,$y_p$)
  contract
    let
      $v_i$ = $e_i$($x_1$,...,$x_n$,$y_1$,...,$y_p$);
      ⋮
    tel
    assume $e_a$($x_1$,...,$x_n$,$y_1$,...,$y_p$,$v_i$)
    enforce $e_g$($x_1$,...,$x_n$,$y_1$,...,$y_p$,$v_i$) with ($c_1$,...,$c_m$)

let
  $y_i$ = $f_i$($x_1$,...,$x_n$,$y_1$,...,$y_p$,$c_1$,...,$c_m$);
  ⋮
tel
```

# Translation into PDS

Computation of two PDS for each node $f$: $S_f$ for the body, $S_f^c$ for the contract.

$$D \longrightarrow (S, I_u, I_c, \mathcal{C})$$

- Translation of equations $D$ into PDS $S$
- Additional uncontrollable inputs $I_u$ from non-inlined applications
- Additional controllable inputs $I_c$ from inlined applications
- Set of contracts $\mathcal{C}$ to be enforced

# Translation: inlined applications

$$\texttt{node}\ f(x_1, \ldots, x_n)\ \texttt{returns}\ (y_1, \ldots, y_p)$$
$$\texttt{contract}\ (A, G)\ \texttt{with}\ I_c$$
$$\downarrow$$
$$\frac{S_f(X, \{x_1, \ldots, x_n\} \uplus I_c) = \begin{cases} X' = P_f(X, x_1, \ldots, x_n, I_c) \\ Q_{0f}(X) \end{cases}}{(z_1, \ldots, z_p) = \texttt{inlined}\ f(e_1, \ldots, e_n)}$$
$$\downarrow$$
$$(S_f[e_i/x_i, z_i/y_i], \emptyset, \hat{I}_c, \{(A[e_i/x_i, z_i/y_i], G[e_i/x_i, z_i/y_i])\})$$

- Inlining by renaming variables of PDS $S_f$: no “textual” inlining
- No uncontrollable inputs added
- “Phantom” controllable inputs from $f$'s controller

# Translation: applications

$$\dfrac{\begin{array}{c}\texttt{node}\ f(x_1,\dots,x_n)\ \texttt{returns}\ (y_1,\dots,y_p)\\ \texttt{contract}\ (A,G)\ \texttt{with}\ I_c\\ \downarrow\\ S_f^c(X,\{x_1,\dots,x_n,y_1,\dots,y_p\}) = \left\{\begin{array}{l} X' = P_f(X,x_1,\dots,x_n,I_c)\\ Q_{0f}(X)\end{array}\right.\end{array}}{\begin{array}{c}(z_1,\dots,z_p) = f(e_1,\dots,e_n)\\ \downarrow\\ (S_f^c[e_i/x_i,z_i/y_i],\{z_1,\dots,z_p\},\emptyset,\{(A[e_i/x_i,z_i/y_i],G[e_i/x_i,z_i/y_i])\})\end{array}}$$

- Inlining of the contract $S_f^c$
- Outputs $z_1,\dots,z_p$ are added as uncontrollable inputs
- No additional controllable inputs

# Synthesis objective

For the node:

```
node f(x_1, ..., x_n) returns (y_1, ..., y_p)
  contract (A, G) with I_c
let D tel
```

- Translate the equations:

$$D \longrightarrow (S', I'_u, I'_c, \{C_1, \dots, C_n\})$$

- With PDS $S = S'(X, \{x_1, \dots, x_n\} \uplus I'_u \uplus I_c \uplus I'_c)$: compute $K = \mathrm{DCS}(S, I_c \uplus I'_c, \Phi)$ with:

$$\Phi = \forall\Box\Big((A_1 \Rightarrow G_1) \wedge \dots \wedge (A_n \Rightarrow G_n) \Rightarrow \big(A \Rightarrow (G \wedge A_1 \wedge \dots \wedge A_n)\big)\Big)$$

# Controller triangulation

For the execution of the controller, we need to compute from $K$ a set of equations $D_c$, to be "weaven" into the initial node by parallel composition.

The result $K$ of the DCS is the most permissive controller: relation $K(X, I_u, c_1, \dots, c_n)$.

From $K$, compute $\text{Triang}(K) = \{K_1, \dots, K_n\}$, such as:

$$\begin{aligned}
c_1 &= K_1(X, I_u, \hat{c}_1) \\
c_2 &= K_2(X, I_u, c_1, \hat{c}_2) \\
&\vdots \\
c_n &= K_n(X, I_u, c_1, \dots, c_{n-1}, \hat{c}_2)
\end{aligned}$$

# Causality issues

Some inputs in $I_u$, added to represent non-inlined applications outputs, can depend on some controllable variables.

```
node f(x1,x2:bool) returns (y1,y2)
  contract
  enforce ... with (c1,c2:bool)
let
  y1 = g(x1,c1);
  y2 = g(x2,c2);
tel
```

We have here $c_1 \prec y_1$ and $c_2 \prec y_2$: $y_1$ and $y_2$ must be quantified while performing triangulation.

$$\begin{aligned} c_1 &= \forall y_1, y_2, K_1(x_1, x_2, y_1, y_2, \hat{c}_1) \\ c_2 &= \forall y_2, K_2(x_1, x_2, y_1, y_2, c_1, \hat{c}_2) \end{aligned}$$

# Example: delayable tasks

```
node delayable(r,c,e:bool) returns (act:bool)
let
  automaton
    state Idle
      do act = false
      until r & c then Active
      until a & not c then Wait
    state Wait
      do act = false
      until c then Active
    state Active
      do act = true
      until e then Idle
  end
tel
```

# Example (cont'd)

Set of $n$ exlusive delayable tasks

```
node ntasks($r_1$,...,$r_n$,$e_1$,...,$e_n$) returns ($a_1$,...,$a_n$:bool)
  contract
  let
    ca$_1$ = a$_1$ & (a$_2$ or ... or a$_n$);
    ⋮
    ca$_{n-1}$ = a$_{n-1}$ & a$_n$;
  tel
  enforce not (ca$_1$ or ... or ca$_{n-1}$) with ($c_1$,...,$c_n$)
let
  $a_1$ = inlined delayable($r_1, c_1, e_1$);
  ⋮
  $a_n$ = inlined delayable($r_n, c_n, e_n$);
tel
```

# Example: composition

```
node main(r_1,...,r_2n,e_1,...,e_2n) returns (a_1,...,a_2n:bool)
  contract
  let
    ca_1 = a_1 & (a_2 or ... or a_2n);
    ⋮
    ca_2n−1 = a_2n−1 & a_2n;
  tel
  enforce not (ca_1 or ... or ca_2n−1) with ()
let
  (a_1,...,a_n) = ntasks(r_1,...,r_n,e_1,...,e_n);
  (a_n+1,...,a_2n) = ntasks(r_n+1,...,r_2n,e_n+1,...,e_2n);
tel
```

⟶ the contract of `ntasks` is not controllable enough to enforce the main contract

# Example (refinement, naive version)

Contract refinement for composition of several `ntasks` components:

```
node ntasks(c,$r_1$,...,$r_n$,$e_1$,...,$e_n$) returns ($a_1$,...,$a_n$:bool)
  contract
  let
    ca$_1$ = a$_1$ & (a$_2$ or ... or a$_n$);...
    ca$_{n-1}$ = a$_{n-1}$ & a$_n$;
    one = $a_1$ or ... or $a_n$;
  tel
  enforce not (ca$_1$ or ... or ca$_{n-1}$) & (c or not one)
  with ($c_1$,...,$c_n$)
let
  $a_1$ $=$ inlined delayable($r_1, c_1, e_1$); ...
  $a_n$ $=$ inlined delayable($r_n, c_n, e_n$);
tel
```

# Example: composition, 2nd try

```
node main($r_1,\dots,r_{2n}$,$e_1,\dots,e_{2n}$) returns ($a_1,\dots,a_{2n}$:bool)
  contract
  let
    $ca_1$ = $a_1$ & ($a_2$ or ... or $a_{2n}$);
    $\vdots$
    $ca_{2n-1}$ = $a_{2n-1}$ & $a_{2n}$;
  tel
  enforce not ($ca_1$ or ... or $ca_{2n-1}$) with (c:bool )
let
  ($a_1,\dots,a_n$) = ntasks(c,$r_1,\dots,r_n$,$e_1,\dots,e_n$);
  ($a_{n+1},\dots,a_{2n}$) = ntasks(not c,$r_{n+1},\dots,r_{2n}$,$e_{n+1},\dots,e_{2n}$);
tel
```

⟶ Synthesis succeed, but the controllers of `ntasks` cannot allow the tasks to go into the active state !

# Example (refinement, correct version)

Use of environment hypothesis to allow more permissive behaviours:

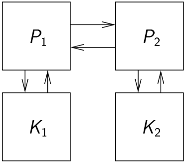
```
node ntasks(c,r_1,...,r_n,e_1,...,e_n) returns (a_1,...,a_n:bool)
  contract
  let
    ca_1 = a_1 & (a_2 or ... or a_n);...
    ca_{n-1} = a_{n-1} & a_n;
    one = a_1 or ... or a_n;
    pone = false fby one;
  tel
  assume (not pone or c)
  enforce not (ca_1 or ... or ca_{n-1}) & (c or not one)
  with (c_1,...,c_n)
let
  a_1 = inlined delayable(r_1,c_1,e_1); ...
  a_n = inlined delayable(r_n,c_n,e_n);
tel
```

# Contribution

- Method for use of contracts for modular controller synthesis
- Integration of an existing controller synthesis tool into a modular compilation process
- Implementation into an existing modular compiler: method accessible through a programming language

# Prospects

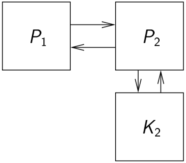
- Diagnosis issues:
  - Synthesis can fail: path of uncontrollable events leading to error states
  - The controller computed can be too strong, e.g., restrict the system or some part of it to stay in its initial state
  - During controller triangulation, quantification can fail
- Decentralized control and program distribution



- Interaction with non-boolean parts/other program transformation or validation methods

# Prospects

- Diagnosis issues:
  - Synthesis can fail: path of uncontrollable events leading to error states
  - The controller computed can be too strong, e.g., restrict the system or some part of it to stay in its initial state
  - During controller triangulation, quantification can fail
- Decentralized control and program distribution



- Interaction with non-boolean parts/other program transformation or validation methods