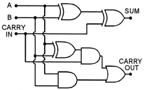
Revisiting the Glue of BIP

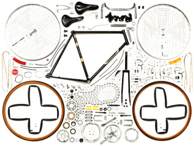
Jacques Combaz

MeTRiD 2019

Component-Based Design



- Complex systems built by assembling components:
  - digital circuits from gates
  - mechanical object from smaller parts (car, ...)
  - backhoe loader from Lego blocks
  - ETAPS from conferences, workshops, tutorials, ...
  - ...

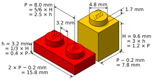


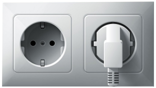
- Advantages:
  - managing complexity: divide-and-conquer
  - modularity: replace / upgrade components
  - reuse of existing components.

ETAPS
EUROPEAN JOINT CONFERENCES ON THEORY & PRACTICE OF SOFTWARE

Assembling Components

- Components interfaces
  - assembly options

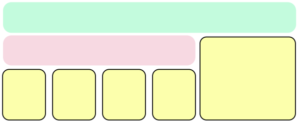




- Systems:
  - what are the components?

  - how are they assembled (architecture)?

Glues for Software Components



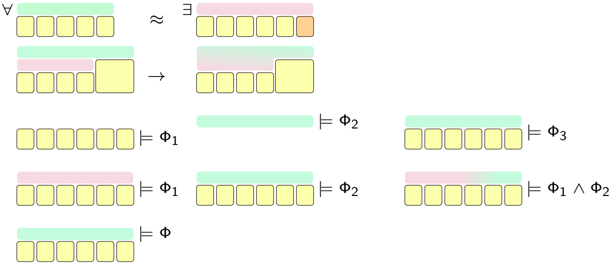
"*Glues*" assemble *software components* based on their interfaces:

- coordination / synchronization: joint actions, barriers, precedence, . . .
- communication: exchange of data
- hierarchy: construction of software components from simpler ones.

Interesting Properties of Glues

Interesting notions for glues:

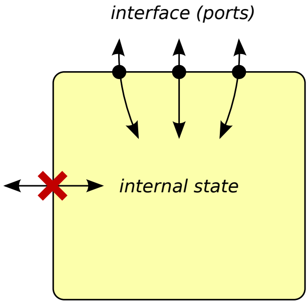
- expressivity[1]
- flattening[2]
- compositionality[3]
- composability[4]
- synthesis[5]



Studied in the **BIP** framework (Sifakis et al.):

[1]: *A Notion of Glue Expressiveness for Component-Based Systems*, CONCUR 2008
[2]: *Source-to-Source Architecture Transformation for Performance Optimization in BIP*, IEEE Trans. Indus. Info. 2010
[3]: *D-Finder: A Tool for Compositional Deadlock Detection and Verification*, CAV 2009
[4]: *A General Framework for Architecture Composability*, SEFM 2014
[5]: *Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems*, Soft. Comp. 2011

(Atomic) Components



Components = private data (internal state) + interface (set of ports):

- ports convey data (in and out)
- transformation of internal state = execution on a port.

(Atomic) Components

(Atomic) Component

A component is an LTS $B = (Q, P, \rightarrow)$ having:

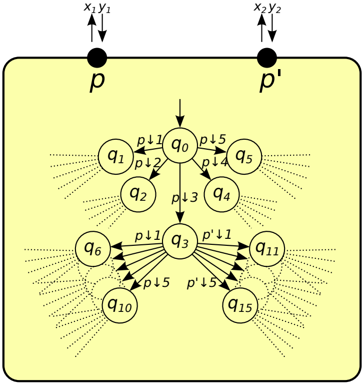
- (internal) states Q (possibly infinite)
- an interface P, i.e. finite set of ports exposing values in domain $\mathcal{D}$
- interface state (at state q) given by $x : P \rightarrow (\mathbb{B} \times \mathcal{D})$
- transitions $\rightarrow \subseteq Q \times P \times \mathcal{D} \times Q$:
  - notation: $q \xrightarrow{p\downarrow y} q'$ for $(q, p, y, q') \in \rightarrow$
  - $y \in \mathcal{D}$: value received by B through p
  - p is enabled at q iff $q \xrightarrow{p\downarrow y} q'$.

Assumptions:

- $x(p) = (e, v)$ at state q if p exposes v and if p enabled $\Leftrightarrow e = \mathtt{true}$
- *completeness*: $\exists y \in \mathcal{D} \; . \; q \xrightarrow{p\downarrow y} q' \Rightarrow \forall y' \in \mathcal{D} \; . \; \exists q'' \in Q \; . \; q \xrightarrow{p\downarrow y'} q''$.

(Atomic) Components

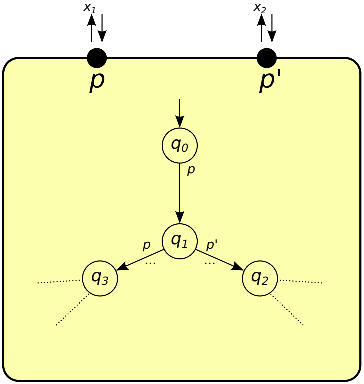
$\mathcal{D} = \{1, 2, 3, 4, 5\}$



Assumptions:

- $x(p) = (e, v)$ at state q if p exposes v and if p enabled $\Leftrightarrow e = \texttt{true}$
- *completeness*: $\exists y \in \mathcal{D} \;.\; q \xrightarrow{p\downarrow y} q' \Rightarrow \forall y' \in \mathcal{D} \;.\; \exists q'' \in Q \;.\; q \xrightarrow{p\downarrow y'} q''$.

(Atomic) Components without Data

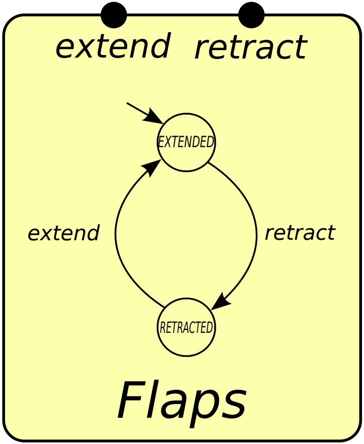


Special case: ports without data:

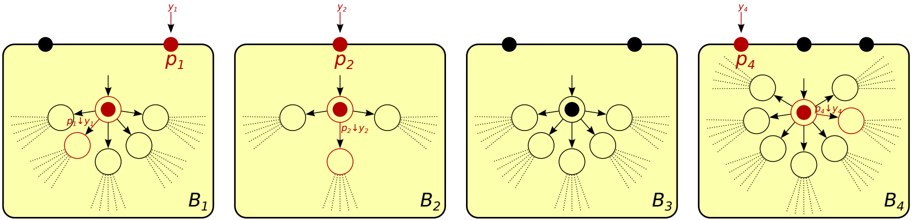
- interface state: $x : P \to \mathbb{B}$
- transitions are of the form $q \xrightarrow{p} q'$.

Example of (Atomic) Component: Aircraft Flaps

```
port type Port()

atom type Flaps()
  export port Port extend(), retract()

  place EXTENDED, RETRACTED

  initial to EXTENDED

  on retract from EXTENDED  to RETRACTED

  on extend  from RETRACTED to EXTENDED
end
```



Composition of Components



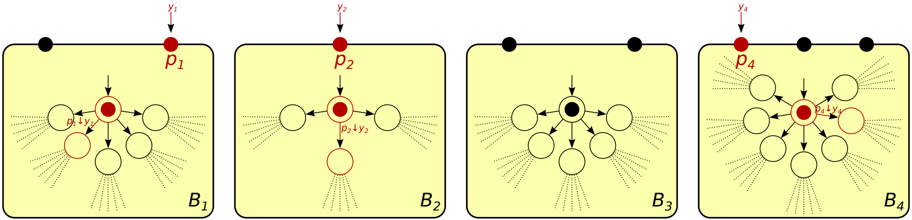
(Parallel) Composition of Components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

- States: $(q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n$
- Execution: subsets of component transitions $\{q_i \xrightarrow{p_i \downarrow y_i}_i q_i'\}_{i \in I}$:

$$(q_1, \ldots, q_n) \xrightarrow{\{p_i \downarrow y_i\}_{i \in I}} (q_1', \ldots, q_n')$$

  - for $i \in I$ we have $q_i \xrightarrow{p_i \downarrow y_i}_i q_i'$ (execution in B_i)
  - for $i \notin I$ we have $q_i' = q_i$ (no execution in B_i).

Composition of Components



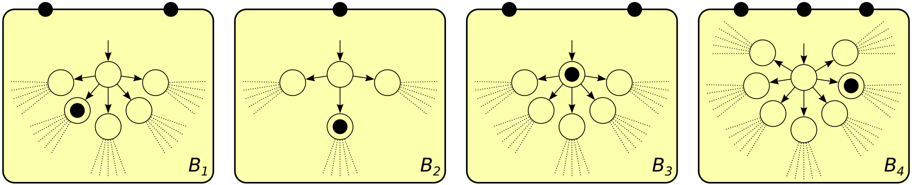
(Parallel) Composition of Components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

- States: $(q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n$
- Execution: partial function $y : P_1 \cup \ldots \cup P_n \nrightarrow \mathcal{D}$, $dom(y) \neq \emptyset$:

$$(q_1, \ldots, q_n) \xrightarrow{\downarrow y} (q'_1, \ldots, q'_n)$$

  - either $dom(y) \cap P_i = \{p_i\}$ and $q_i \xrightarrow{p_i \downarrow y(p_i)}_i q'_i$ (execution in B_i)
  - or $dom(y) \cap P_i = \emptyset$ and $q'_i = q_i$ (no execution in B_i).

Composition of Components



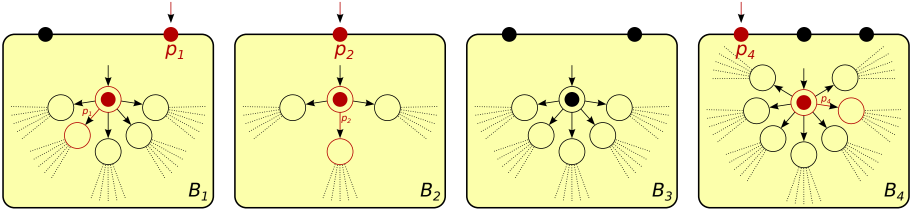
(Parallel) Composition of Components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

- States: $(q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n$
- Execution: partial function $y : P_1 \cup \ldots \cup P_n \nrightarrow \mathcal{D}$, $dom(y) \neq \emptyset$:

$$(q_1, \ldots, q_n) \xrightarrow{\downarrow y} (q'_1, \ldots, q'_n)$$

  - either $dom(y) \cap P_i = \{p_i\}$ and $q_i \xrightarrow{p_i \downarrow y(p_i)}_i q'_i$ (execution in B_i)
  - or $dom(y) \cap P_i = \emptyset$ and $q'_i = q_i$ (no execution in B_i).

Composition of Components without Data



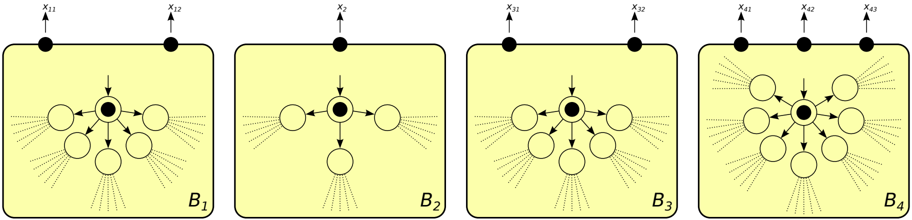
(Parallel) Composition of Components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

- States: $(q_1, \ldots, q_n) \in Q_1 \times \ldots \times Q_n$
- Execution: subset of ports $y \subseteq P_1 \cup \ldots \cup P_n$, $y \neq \emptyset$:

$$(q_1, \ldots, q_n) \xrightarrow{\downarrow y} (q'_1, \ldots, q'_n)$$

  - either $y \cap P_i = \{p_i\}$ and $q_i \xrightarrow{p_i}_i q'_i$ (execution in B_i)
  - or $y \cap P_i = \emptyset$ and $q'_i = q_i$ (no execution in B_i).

Notations

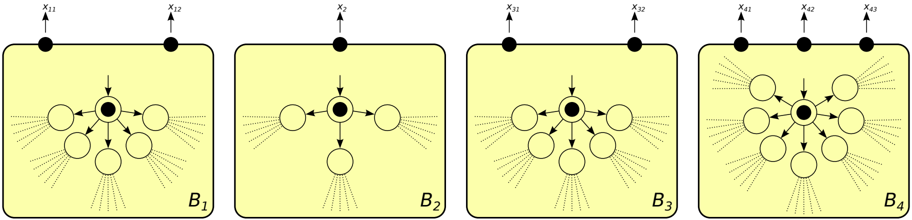


Notations (components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1...n}$)

Given a subset of ports $P \subseteq P_1 \cup \ldots \cup P_n$:

- $[P]^{\uparrow}$: *states of interface* P, i.e. $x : P \rightarrow (\mathbb{B} \times \mathcal{D})$
- $[P]^{\wr}$: *executions* over P, i.e. $y : P \nrightarrow \mathcal{D}$ s.t. $\forall i \,.\, |dom(y) \cap P_i| \leq 1$
- $[P]^{\downarrow}$: *executions* with domain P, i.e. $y \in [P]^{\wr}$ s.t. $dom(y) = P$.

Notations without Data

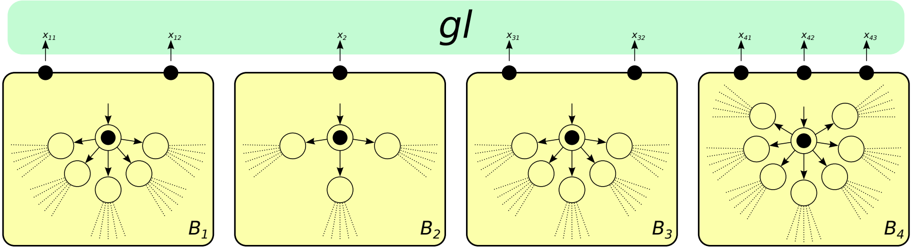


Notations (components $\{B_i = (Q_i, P_i, \to_i)\}_{i=1\dots n}$ without data)

Given a subset of ports $P \subseteq P_1 \cup \ldots \cup P_n$:

- $[P]^{\uparrow}$: *states of interface* P, i.e. $x : P \to \mathbb{B}$
- $[P]^{\wr}$: *executions* over P, i.e. $y \subseteq P$ s.t. $\forall i \, . \, |y \cap P_i| \leq 1$
- $[P]^{\downarrow}$: *executions* with domain P, i.e. $[P]^{\downarrow} = \{P\}$ if $\forall i \, . \, |P \cap P_i| \leq 1$, $\emptyset$ otherwise.

Glues



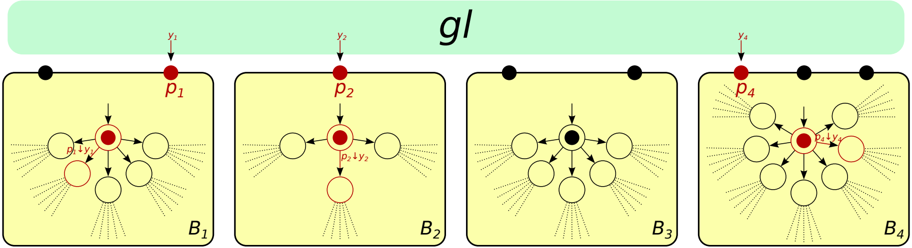
Glue for a composition of components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

A glue gl restricts executions $\xrightarrow{\downarrow y}$ in a composition, based (only) on interface state $x \in [P]^{\uparrow}$ of ports $P = P_1 \cup \ldots \cup P_n$:

$$gl : [P]^{\uparrow} \rightarrow 2^{[P]^{\wr}}$$

s.t. $\forall x \,.\, \forall y \in gl(x) \,.\, \forall p \in dom(y) \,.\, x(p) \in \{\texttt{true}\} \times \mathcal{D}$ (i.e. all ports executed by y are enabled in x).

Glues



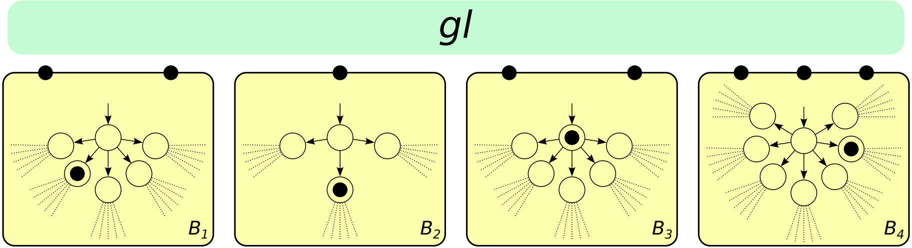
Glue for a composition of components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$

A glue gl restricts executions $\xrightarrow{\downarrow y}$ in a composition, based (only) on interface state $x \in [P]^{\uparrow}$ of ports $P = P_1 \cup \ldots \cup P_n$:

$$gl : [P]^{\uparrow} \rightarrow 2^{[P]^{\wr}}$$

s.t. $\forall x \,.\, \forall y \in gl(x) \,.\, \forall p \in dom(y) \,.\, x(p) \in \{\mathtt{true}\} \times \mathcal{D}$ (i.e. all ports executed by y are enabled in x).

Glues



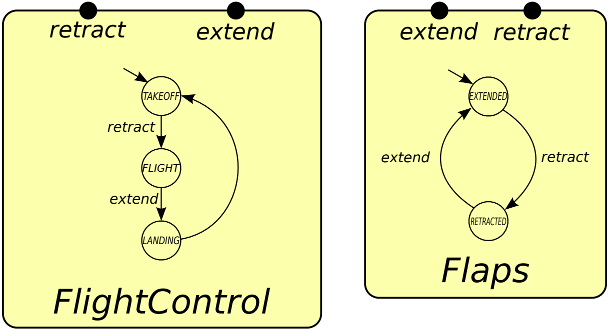
Glue for a composition of components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\dots n}$

A glue gl restricts executions $\xrightarrow{\downarrow y}$ in a composition, based (only) on interface state $x \in [P]^{\uparrow}$ of ports $P = P_1 \cup \dots \cup P_n$:

$$gl : [P]^{\uparrow} \rightarrow 2^{[P]^{\wr}}$$

s.t. $\forall x \,.\, \forall y \in gl(x) \,.\, \forall p \in dom(y) \,.\, x(p) \in \{\texttt{true}\} \times \mathcal{D}$ (i.e. all ports executed by y are enabled in x).

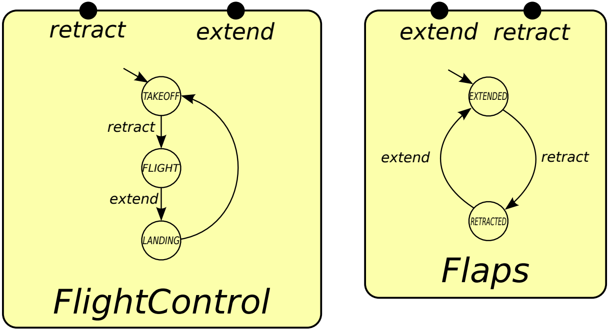
Example of Glue: Flight Control (FC) + Flaps (F)



- Synchronize both ports *retract* (resp. *extend*) if they are enabled.

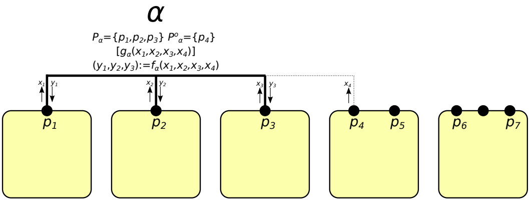
Example of Glue: Flight Control (FC) + Flaps (F)

$$gl(x) \subseteq \{y, y'\}$$
$$y = \{FC.retract, F.retract\}$$
$$y' = \{FC.extend, F.extend\}$$
$$y \in gl(x) \text{ iff } x(FC.retract) \wedge x(FC.retract)$$
$$y' \in gl(x) \text{ iff } x(FC.extend) \wedge x(FC.extend)$$



- Synchronize both ports *retract* (resp. *extend*) if they are enabled.

Interactions



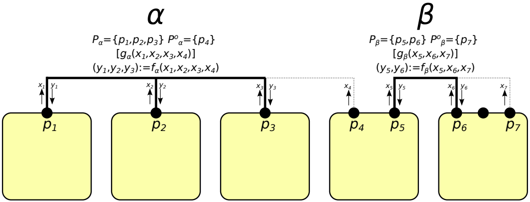
Proposal for Interaction

An interaction $\alpha = (g_\alpha, f_\alpha)$ is a synchronization between a subset of ports $P_\alpha \neq \emptyset$ of $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1 \dots n}$ s.t. $|P_\alpha \cap P_i| \leq 1$ and

- the guard $g_\alpha : [P_\alpha \cup P^o_\alpha]^\uparrow \rightarrow \mathbb{B}$ sat. $g_\alpha(x) \Rightarrow x(P_\alpha) \subseteq \{\mathtt{true}\} \times X$
- $f_\alpha : [P_\alpha \cup P^o_\alpha]^\uparrow \rightarrow [P_\alpha]^\downarrow$ is the transfer function

P^o are the ports observed by interaction α, possibly $P_\alpha \cup P^o_\alpha = P$.

Semantics of Interactions

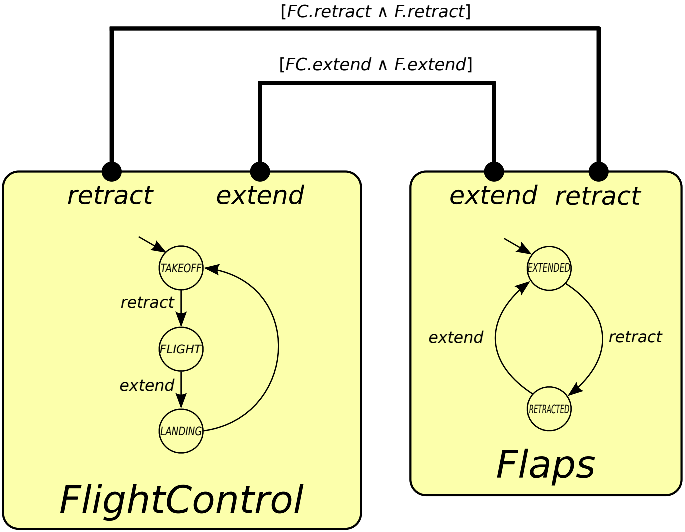


Glue for Interactions

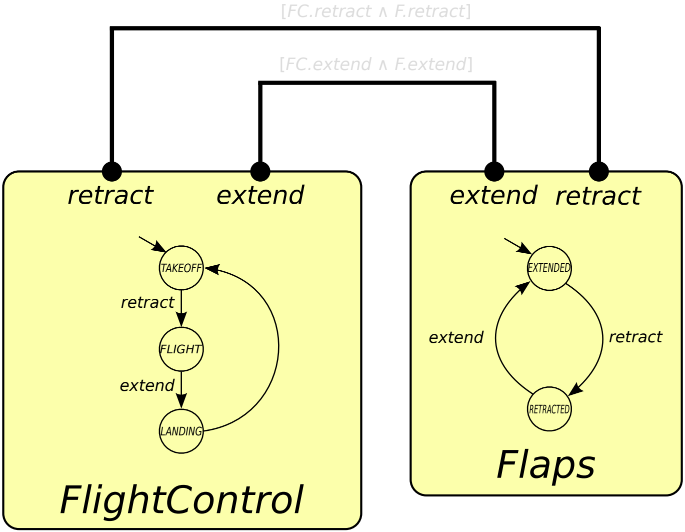
A set of interactions Γ corresponds to the glue gl satisfying:

$$gl(x) = \{\ f_\alpha(x) \mid \alpha \in \Gamma\ \wedge\ g_\alpha(x)\ \}.$$

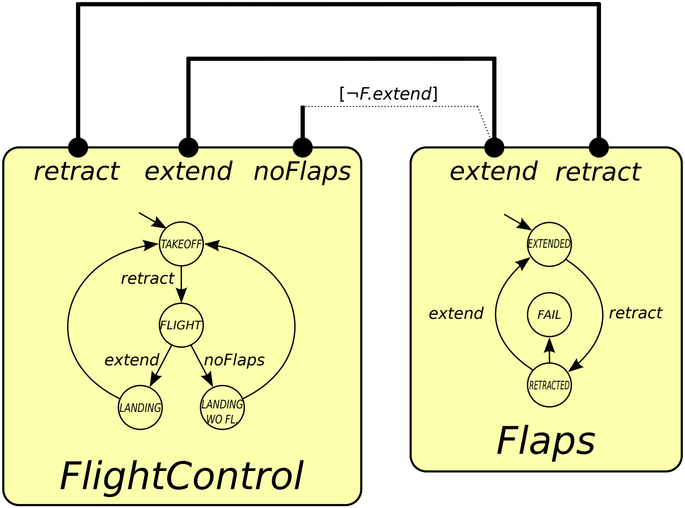
Example: Flight Control + Flaps



Example: Flight Control + Flaps



Example: Flight Control + Flaps (cont'd)



Universality of Interactions

Theorem (Universality)

For any glue gl on a composition $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1\ldots n}$ their exists an equivalent set of interactions Γ s.t.:

$$|\Gamma| \leq 2^{|P|-1}|gl|$$

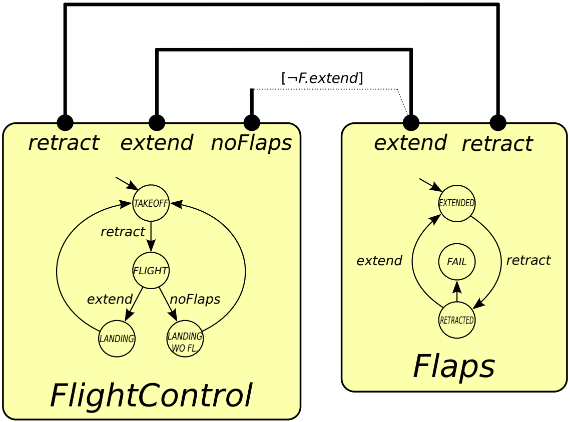
where $|gl| = \max_{x \in [P]^{\uparrow}} |gl(x)|$.

Sketch of proof for $|gl| = K < +\infty$:

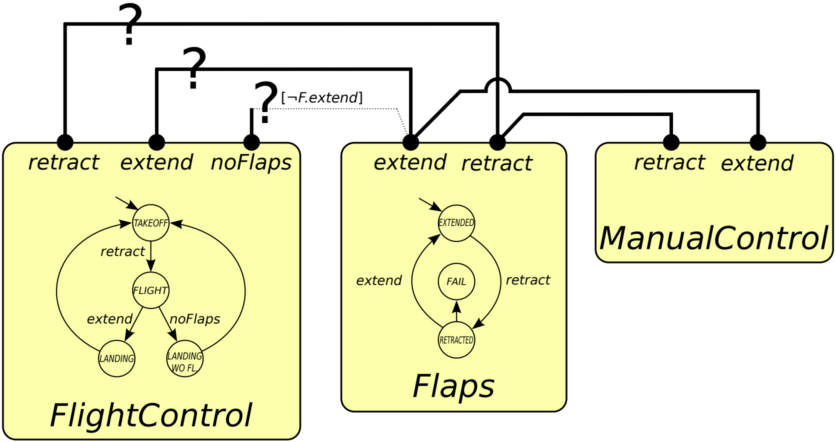
For $P' \subseteq P$ s.t. $\forall i \ |P' \cap P_i| \leq 1$ we build interactions $\{\alpha_j\}_{j=1..K}$ s.t.:

- $\forall j$ we have $P_{\alpha_j} = P'$ and $P^o_{\alpha_j} = P_1 \cup \ldots \cup P_n$
- $\forall x$ we have $\{ \ y \in gl(x) \mid dom(y) = P' \ \} = \{y_j\}_{j=1..L}$, $L \leq K$, and:
  - $g_{\alpha_j}(x) \Leftrightarrow 1 \leq j \leq L$
  - $f_{\alpha_j}(x) = y_j$ for $j = 1..L$, any value otherwise.

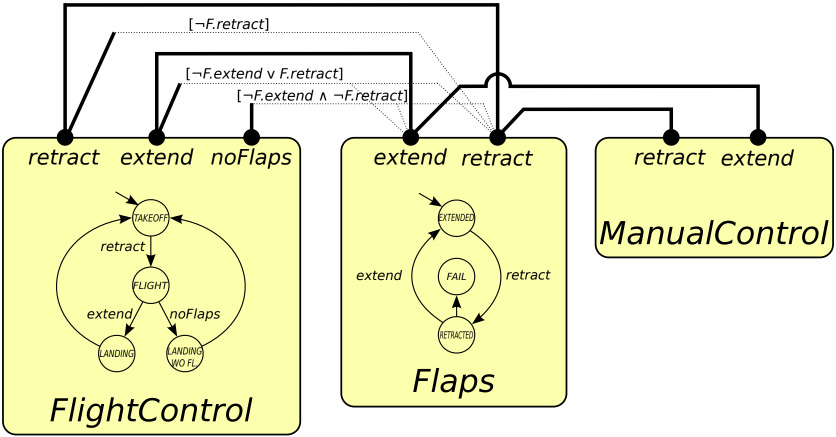
Universality but not Minimalism



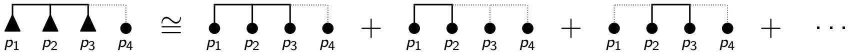
Universality but not Minimalism



Universality but not Minimalism



A Notion of Connectors



connector interactions

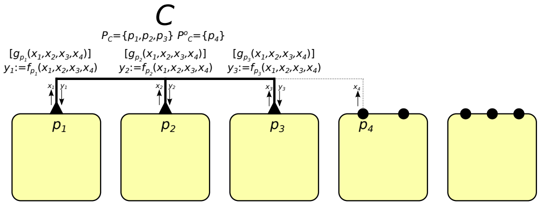
Connectors: compact representations for subsets interactions Γ s.t.:

- guards are mutually exclusive (max. 1 enabled interaction at a state):
$$\forall \alpha \neq \beta \in \Gamma \,.\, \forall x \,.\, \neg(g_\alpha(x) \wedge g_\beta(x))$$
- at most one participating port per component: $\left| P_i \cap \bigcup_{\alpha\in\Gamma} P_\alpha \right| \leq 1$.

Remark: for such sets Γ, for any $p \in \bigcup_{\alpha\in\Gamma} P_\alpha$ we can define:

- $g_p : \left[\bigcup_{\alpha\in\Gamma} P_\alpha \cup P_\alpha^o \right]^\uparrow \to \mathbb{B}$ s.t. $g_p = \bigvee_{\alpha\in\Gamma} g_\alpha \wedge p \in P_\alpha$
- $f_p : \left[\bigcup_{\alpha\in\Gamma} P_\alpha \cup P_\alpha^o \right]^\uparrow \to \mathcal{D}$ s.t. $f_p(x) = f_\alpha(x)(p)$ if $p \in P_\alpha$ and $g_\alpha(x)$.

Connectors

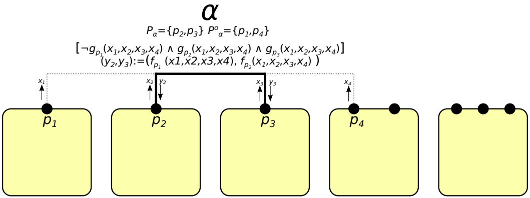


Proposal for Connectors

A connector C is defined by

- a set of potentially participating ports P_C s.t. $\forall i \,.\, |P_C \cap P_i| \leq 1$
- a set of observed ports P^o_C
- guards $\left\{g_p : [P_C \cup P^o_C]^\uparrow \to \mathbb{B}\right\}_{p \in P_C}$ s.t. $g_p(x) \Rightarrow p$ is enabled in x
- transfer functions $\left\{f_p : [P_C \cup P^o_C]^\uparrow \to \mathcal{D}\right\}_{p \in P_C}$.

Connectors

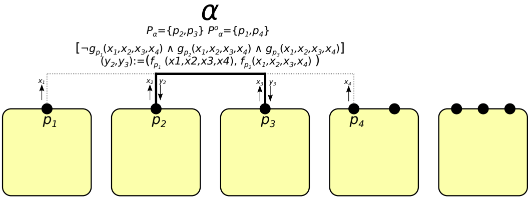


Interactions of a Connector

A connector C corresponds to interactions $\{\ \alpha \mid P_\alpha \neq \emptyset\ \wedge\ P_\alpha \subseteq P_C\ \}$ and:

- $P^o_\alpha = (P_C \setminus P_\alpha) \cup P^o_C$
- $\forall x\ .\ g_\alpha(x) = \bigwedge_{p \in P_\alpha} g_p(x)\ \wedge\ \bigwedge_{p \in P_c \setminus P_\alpha} \neg g_p(x)$
- $\forall x\ .\ f_\alpha(x) : p \mapsto f_p(x)$.

Glue for Connectors



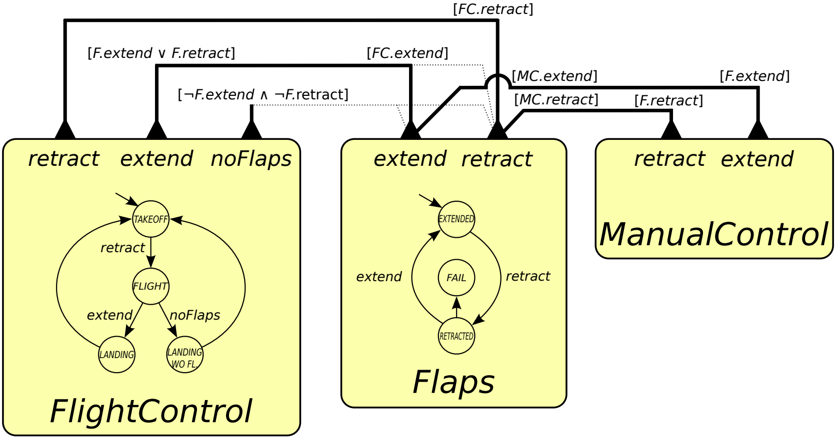
Glue for Connectors

A set of connectors $\mathcal{C}$ corresponds to the glue gl satisfying
$y \in gl(x)$ iff $\exists C \in \mathcal{C}$ s.t.:

- $dom(y) = \{ \ p \in P_C \mid g_p(x) \ \} \neq \emptyset$
- $\forall p \in dom(y) \ . \ y(p) = f_p(x)$.

$\rightarrow$ computing gl is linear in number of connectors and part. ports.

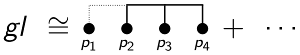
Example: Flight Control + Flaps + Manual Control



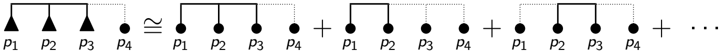
Conclusion

Contributions to the BIP framework:

- formalization of glues taking into account data
- proposed interactions: expressive enough to encode any glue gl



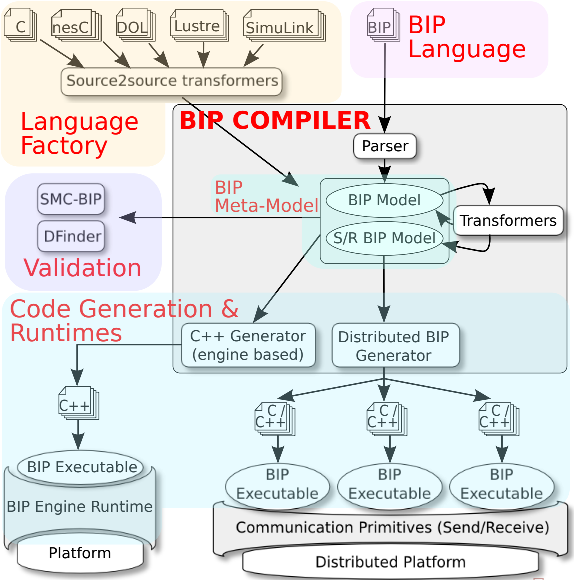
- proposed connectors:
  - compact representations of interactions (incl. guards / data transfers)



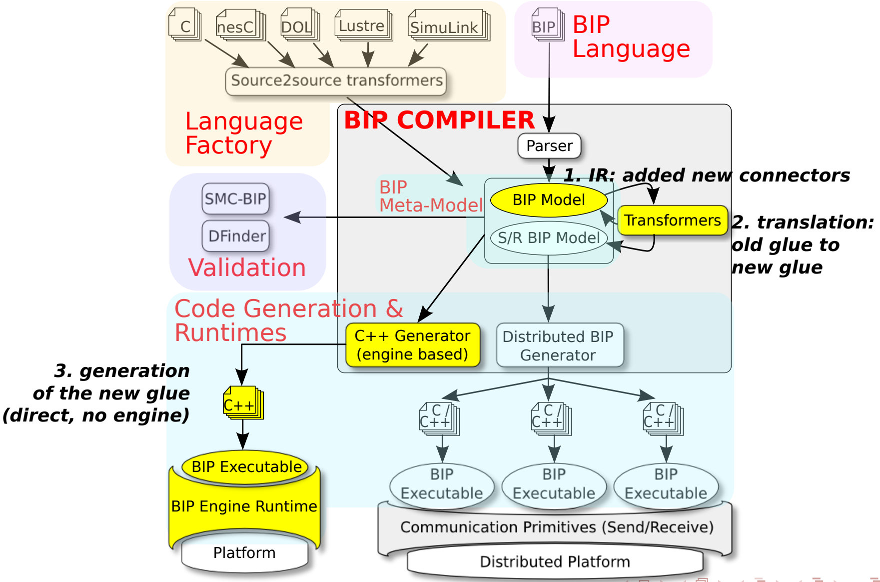
  - runtime evaluation of linear complexity.

Thank you.

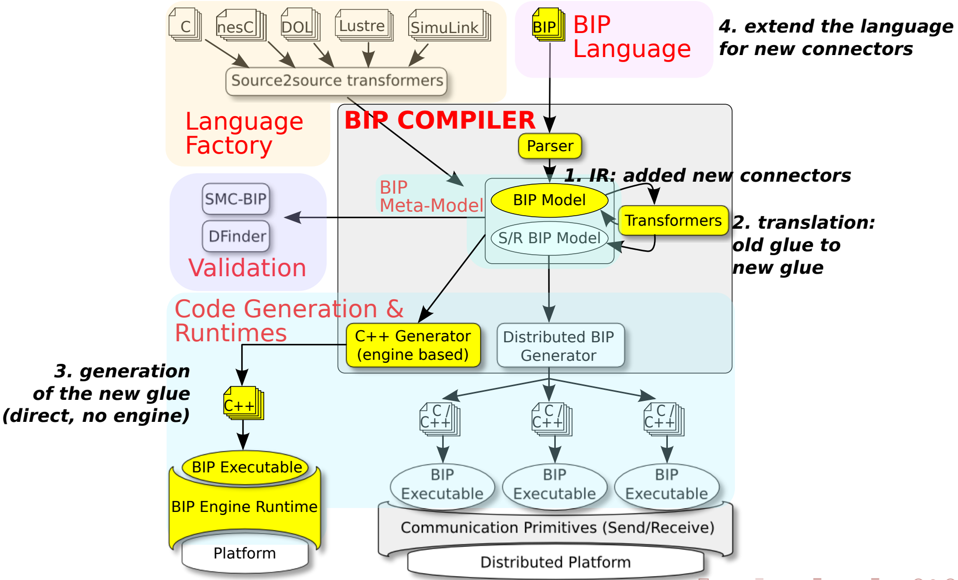
Implementation Steps



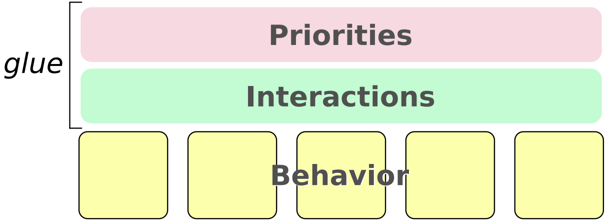
Implementation Steps



Implementation Steps



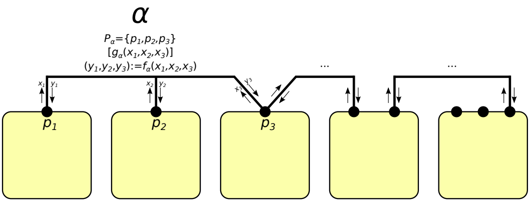
The Glue of BIP



BIP offers a rich language for assembling components using:

- interactions = multiparty rendez-vous + data transfer
- connectors = compact representation of sets of interactions: *rendez-vous, broadcast, atomic broadcast, causal chain, ...*
- priorities = partial order on interactions

Interactions



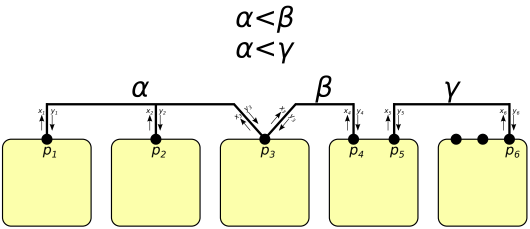
Interaction

An interaction $\alpha = (g_\alpha, f_\alpha)$ is a synchronization between a subset of ports $P_\alpha \neq \emptyset$ of $\{B_i = (Q_i, P_i, \to_i)\}_{i=1\dots n}$ s.t. $|P_\alpha \cap P_i| \leq 1$ and

- the guard $g_\alpha : [P_\alpha]^\uparrow \to \mathbb{B}$ satisfies $g_\alpha(x) \Rightarrow \forall p \in P_\alpha \ . \ \mathcal{E}_p(x)$
- $f_\alpha : [P_\alpha]^\uparrow \to [P_\alpha]^\uparrow$ is the transfer function.

The above functions can be applied to any $x \in [P]^\uparrow$ by taking $x_{|P_\alpha}$.

Priorities



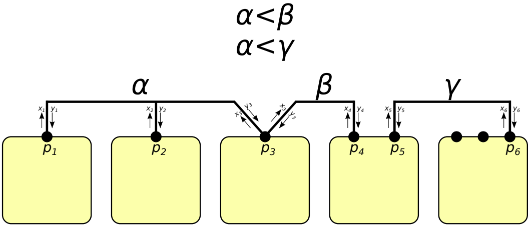
Priorities

Priorities is a set of rules that define a strict partial $\prec$ order over a set of interactions Γ.

A special case is maximal progress over a subset Γ':

$$\alpha \subsetneq_{\Gamma'} \beta \iff \alpha, \beta \in \Gamma' \ \wedge \ P_\alpha \subsetneq P_\beta.$$

Semantics of Interactions + Priorities

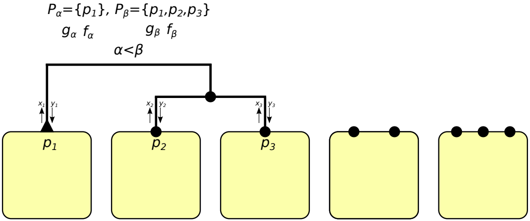


Glue for Interactions + Priorities

Interactions Γ under priorities $\prec$ corresponds to the glue gl satisfying:

$$gl(x) = \{\ f_\alpha(x) \mid \alpha \in \Gamma\ \wedge\ g_\alpha(x)\ \wedge\ \forall \alpha \prec \beta\ .\ \neg g_\beta(x)\ \}.$$

Connectors

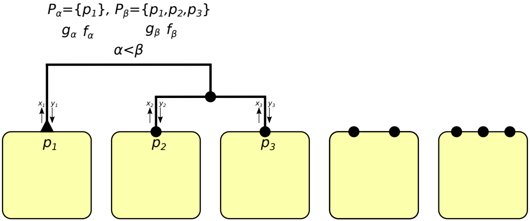


Connector

Connectors = subsets of interactions + priorities, defined using trigger ports (▲) and synchron ports (•) (possibly hierarchically):

- ports P_α of interactions contain at least one ▲ or all the ports
- guards g_α and transfer functions f_α: given by enumeration
- priorities = maximal progress ($\subsetneq$).

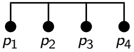
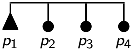
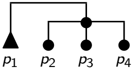
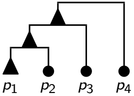
Connectors



Connectors: Recursive Characterization of Subsets of Ports P_α

- $P_\alpha \models C_1 \; C_2 \; C_3 \; C_4 \; \dots \; C_n$ iff $P_\alpha = \bigcup_{i=1\dots n} P_i$ and $\forall i \;.\; P_i \models C_i$
- $P_\alpha \models C_1 \; C_2 \; \dots \; C_j \; \dots \; C_n$ iff $P_\alpha = \bigcup_{i \in \{j\} \cup I} P_i$ and $\forall i \;.\; P_i \models C_i$
- $P_i \models p_i$ iff $P_i = \{p_i\}$ for a (component) port p_i.

Example of Connectors

Pattern	Connector	Interactions (Priorities $= \subsetneq$)
rendez-vous		$\{p_1, p_2, p_3, p_4\}$
broadcast		$\{p_1\}$ $\{p_1, p_2\}$ $\{p_1, p_3\}$ $\{p_1, p_4\}$ $\{p_1, p_2, p_3\}$ $\{p_1, p_2, p_4\}$ $\{p_1, p_3, p_4\}$ $\{p_1, p_2, p_3, p_4\}$
atomic broadcast		$\{p_1\}$ $\{p_1, p_2, p_3, p_4\}$
causality chain		$\{p_1\}$ $\{p_1, p_2\}$ $\{p_1, p_2, p_3\}$ $\{p_1, p_2, p_3, p_4\}$

Example of Connectors: Concrete Syntax

rendez-vous	connector type RendezVous(IPort p1, IPort p2, IPort p3, IPort p4)
	  define p1 p2 p3 p4
	  on p1 p2 p3 p4 provided (/* guard */) down { /* transfer function */ }
	end

broadcast	connector type Broadcast(IPort p1, IPort p2, IPort p3, IPort p4)
	  define p1' p2 p3 p4
	  on p1 provided (...) down { ... }
	  on p1 p2 provided (...) down { p2.x = p1.x; }
	  on p1 p3 provided (...) down { p3.x = p1.x; }
	  on p1 p4 provided (...) down { p3.x = p1.x; }
	  on p1 p2 p3 provided (...) down { p2.x = p1.x; p3.x = p1.x; }
	  on p1 p2 p4 provided (...) down { p2.x = p1.x; p4.x = p1.x; }
	  on p1 p3 p4 provided (...) down { p3.x = p1.x; p4.x = p1.x; }
	  on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }
	end

atomic broadcast	connector type AtomicBroadcast(IPort p1, IPort p2, IPort p3, IPort p4)
	  define p1' (p2 p3 p4)
	  on p1 provided (...) down { ... }
	  on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }
	end

causality chain	connector type CausalityChain(IPort p1, IPort p2, IPort p3, IPort p4)
	  define ((p1' p2)' p3)' p4
	  on p1 provided (...) down { ... }
	  on p1 p2 provided (...) down { p2.x = p1.x; }
	  on p1 p2 p3 provided (...) down { p2.x = p1.x; p3.x = p1.x; }
	  on p1 p2 p3 p4 provided (...) down { p2.x = p1.x; p3.x = p1.x; p4.x = p1.x; }
	end

Example of Connectors: Concrete Syntax (No Data)

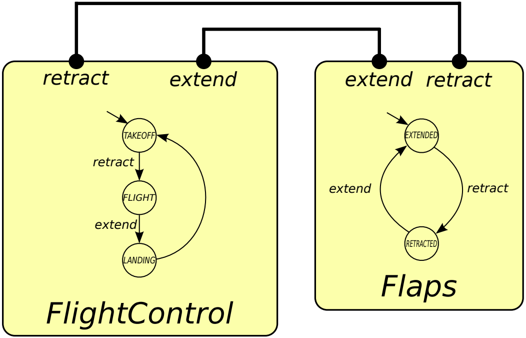
rendez-vous	`connector type RendezVous(Port p1, Port p2, Port p3, Port p4)`
	`  define p1 p2 p3 p4`
	`end`

broadcast	`connector type Broadcast(Port p1, Port p2, Port p3, Port p4)`
	`  define p1' p2 p3 p4`
	`end`

atomic broadcast	`connector type AtomicBroadcast(Port p1, Port p2, Port p3, Port p4)`
	`  define p1' (p2 p3 p4)`
	`end`

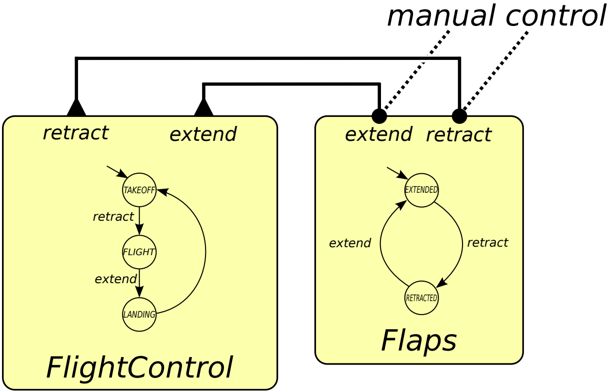
causality chain	`connector type CausalityChain(Port p1, Port p2, Port p3, Port p4)`
	`  define ((p1' p2)' p3)' p4`
	`end`

Example: Flight Control + Flaps



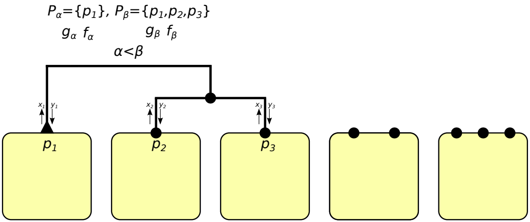
- Synchronize ports *extend* (resp. *retract*) of *FlightControl* and *Flaps*

Example: Flight Control + Flaps + Manual Control



- Can always execute port *extend* / *retract* in *FlightControl*

Glue in BIP = Connectors + Priorities



Glue = Connectors + Priorities

In BIP, the glue gl is defined by connectors $C_1, \ldots, C_m$ and priorities $\prec_{user}$ and corresponds to interactions Γ and priorities $\prec$ s.t.:

- $\Gamma = \bigcup_{i=1 \ldots m} \Gamma_j$ where Γ_j is the set of interactions of C_j
- $\prec = \prec_{user} \oplus \bigoplus_{j=1 \ldots m} \subsetneq_{\Gamma_j}$ where $\prec_{user}$ must be compatible with $\subsetneq_{\Gamma_j}$.

Implementation of Connectors/Interactions + Priorities

Implementation

For interactions Γ and priorities $\prec$, an implementation evaluate $gl(x)$ by searching for one/all interaction(s) $\alpha = (g_\alpha, f_\alpha) \in \Gamma$ s.t.:

$$g_\alpha(x) \wedge \bigwedge_{\alpha \prec \beta} \alpha \prec \beta \Rightarrow \neg g_\beta(x).$$

Optimization #1: filter by g_α, then apply $\prec$

1. Compute $\Delta = \{ \alpha \mid g_\alpha(x) \}$.
2. Remove any α from Δ if $\exists \beta \in \Delta$ s.t. $\alpha \prec \beta$.

Optimization #2: follow priority order $\prec$

Compute $search(\Gamma)$ where $\top(\Delta) = \{ \alpha \in \Delta \mid \forall \beta \in \Delta \,.\, \alpha \not\prec \beta \}$ and:

- $search(\emptyset) = \emptyset$
- $search(\Delta) = \{\alpha \in \top(\Delta) \mid g_\alpha(x)\} \cup search(\{ \alpha \in \top(\Delta) \mid \neg g_\alpha(x)\})$.

Limitations of Connectors/Interactions + Priorities

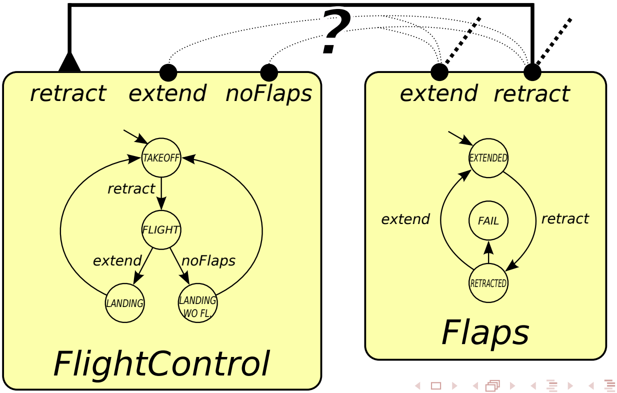
Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal "glue"!

Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

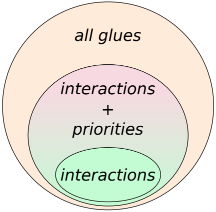
- expressivity: not the universal “glue”!
    - execute *extend* if *extend* or *retract* is enabled in *Flaps*
    - execute *noFlaps* if both *extend* and *retract* are disabled in *Flaps*



Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

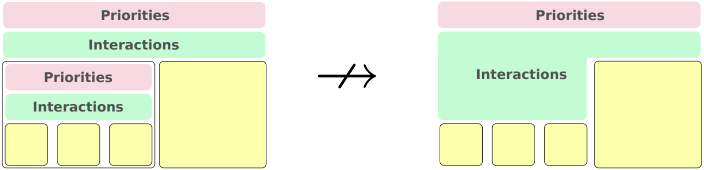
- expressivity: not the universal "glue"!



Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal "glue"!
- flattening of hierarchical structures: not always possible



Limitations of Connectors/Interactions + Priorities

Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal “glue”!
- flattening of hierarchical structures: not always possible
- not compact w.r.t. data (exponential enumeration)

```
connector type Broadcast(IPort p1, IPort p2, IPort p3, ..., IPort pn)
  define p1' p2 p3 ... pn
  on p1             provided (...) down { ... }                               // 1
  on p1 p2          provided (...) down { p2.x = p1.x; }                      // 2
  on p1 p3          provided (...) down { p3.x = p1.x; }                      // 3
  ...
  on p1 p2 p3 ... pn provided (...) down { p2 = p1.x; p2 = p1.x; ... pn = p1.x; }  // 2^n-1
end
```

Limitations of Connectors/Interactions + Priorities

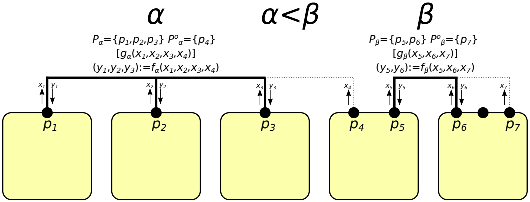
Connectors + Priorities = elegant for synchronization but:

- expressivity: not the universal "glue"!
- flattening of hierarchical structures: not always possible
- not compact w.r.t. data (exponential enumeration)
- runtime complexity

WC complexity	searching	memory allocation
rendez-vous	1	1
flat + (1)	$O(n)$	1
flat	$O(2^n)$	$O(\binom{n}{n/2})$
hierarchy + (1) + (2)	$O(n)$	1
hierarchy + (2)	$O(2^n)$	$O(\binom{n}{n/2})$
hierarchy	$O(2^n)$	$O(2^n)$

(1): guards s.t. $g_\alpha \wedge g_\beta \Rightarrow g_\gamma$ if $P_\alpha \cup P_\beta \subseteq P_\gamma$
(2): guards are independent from lower levels

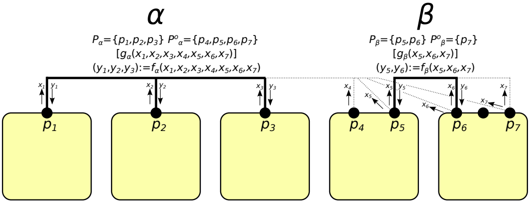
Encoding Priorities in Interactions



Priorities $\prec$ corresponds to modifying interactions α of a set Γ s.t.:

- $P'_\alpha = P_\alpha$
- $P'^o_\alpha = P^o_\alpha \cup \bigcup_{\alpha \prec \beta} P_\beta \cup P^o_\beta$
- $g'_\alpha : [P_\alpha \cup P'^o_\alpha]^\uparrow \rightarrow \mathbb{B}$ s.t. $g'_\alpha(x) = g_\alpha(x) \wedge \bigwedge_{\alpha \prec \beta} \neg g_\beta(x)$
- $f'_\alpha : [P_\alpha \cup P'^o_\alpha]^\uparrow \rightarrow [P_\alpha]^\uparrow$ s.t. $f'_\alpha(x) = f_\alpha(x)$.

Encoding Priorities in Interactions



Priorities $\prec$ corresponds to modifying interactions α of a set Γ s.t.:

- $P'_\alpha = P_\alpha$
- $P'^{o}_\alpha = P^{o}_\alpha \cup \bigcup_{\alpha \prec \beta} P_\beta \cup P^{o}_\beta$
- $g'_\alpha : [P_\alpha \cup P'^{o}_\alpha]^\uparrow \to \mathbb{B}$ s.t. $g'_\alpha(x) = g_\alpha(x) \wedge \bigwedge_{\alpha \prec \beta} \neg g_\beta(x)$
- $f'_\alpha : [P_\alpha \cup P'^{o}_\alpha]^\uparrow \to [P_\alpha]^\uparrow$ s.t. $f'_\alpha(x) = f_\alpha(x)$.

Example of (Atomic) Component: Aircraft Flaps

```
port type Port(const int n)

atom type Flaps(int MAX_CYCLES)
  data int n
  export port Port extend(nbCycles),
                  retract(nbCycles)

  place EXTENDED, RETRACTED

  initial to EXTENDED
    do { nbCycles = 0; }

  on retract from EXTENDED  to RETRACTED
    provided (nbCycles < MAX_CYCLES)

  on extend  from RETRACTED to EXTENDED
    do { nbCycles++; }
end
```