A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
with Jean-Pierre Talpin
Paul Le Guernic    Thierry Gautier

1 INRIA, Resarch Unit of Rennes-Bretagne-Atlantique,
  Rennes, France
  Team ESPRESSO

December 1, 2008
1 Introduction
   - Context
   - Motivations
   - Goals

2 A Model for Contracts
   - Process
   - Process-filter
   - Contract

3 Use Case

4 Conclusion

5 Further work

A Boolean algebra of contracts for assume-guarantee reasoning
Yann Glouche
1. **Introduction**
   - Context
   - Motivations
   - Goals

2. **A Model for Contracts**
   - Process
   - Process-filter
   - Contract

3. Use Case

4. Conclusion

5. Further work

---

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
1 Introduction
   - Context
   - Motivations
   - Goals

2 A Model for Contracts
   - Process
   - Process-filter
   - Contract

3 Use Case

4 Conclusion

5 Further work

A Boolean algebra of contracts for assume-guarantee reasoning
Yann Glouche
1 Introduction
   - Context
   - Motivations
   - Goals

2 A Model for Contracts
   - Process
   - Process-filter
   - Contract

3 Use Case

4 Conclusion

5 Further work

A Boolean algebra of contracts for assume-guarantee reasoning
Yann Glouche
1. Introduction
   - Context
   - Motivations
   - Goals

2. A Model for Contracts
   - Process
   - Process-filter
   - Contract

3. Use Case

4. Conclusion

5. Further work

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
1 Introduction
   - Context
   - Motivations
   - Goals

2 A Model for Contracts
   - Process
   - Process-filter
   - Contract

3 Use Case

4 Conclusion

5 Further work
Polychyrony

1. Tool used for embedded systems design
2. Developed by the team ESPRESSO
3. Design of concurrent systems
   architecture exploration
   simulation and checking
SIGNAL hypotheses:

1. **Abstraction** of the real time
2. Communications and calculus are **instantaneous**
3. The set of tags is equipped with a **partial order relation**

Abstract the components by their interface
Abstract description of distributed architectures
Use a formal concept for integrating a system in Polychrony for:

1. Testing the compatibility between the implementation of a component and its interface
2. Checking the substituability between two components in a system
3. Checking the adequation between an application and its environment execution
4. Finding the errors at all steps of the system design
Use a formal concept for integrating a system in *Polychrony* for:

1. **Testing the compatibility** between the implementation of a component and its interface
2. **Checking the substituability** between two components in a system
3. **Checking the adequation** between an application and its environment execution
4. **Finding the errors at all steps** of the system design
Use a formal concept for integrating a system in *Polychrony* for:

1. Testing the **compatibility** between the implementation of a component and its interface
2. Checking the **substituability** between two components in a system
3. Checking the adequation between an application and its environment execution
4. Finding the **errors** at all steps of the system design
Use a formal concept for integrating a system in Polychrony for:

1. Testing the **compatibility** between the implementation of a component and its interface
2. Checking the **substituability** between two components in a system
3. Checking the **adequation** between an application and its environment execution
4. Finding the errors at all steps of the system design
Goals

1. Use the concept of *assume/guarantee* for designing the SIGNAL processes

2. Extend the SIGNAL language for operating with type system based on the assume/guarantee reasoning
Goals

1. Use the concept of *assume/guarantee* for designing the **SIGNAL** processes

2. Extend the **SIGNAL** language for operating with type system based on the *assume/guarantee* reasoning
A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Processes

Definition: Behavior

- \( \mathcal{V} \) be an infinite, countable set of variables,
- \( \mathcal{D} \) a set of values;
- for \( \mathcal{Y} \), a finite set of variables included in \( \mathcal{V} \), \( \mathcal{Y} \) nonempty,
- a \( \mathcal{Y} \)-behavior is a function \( c : \mathcal{Y} \rightarrow \mathcal{D} \); the set of \( \mathcal{Y} \)-behaviors is \( \mathcal{B}_\mathcal{Y} \).

\[
\mathcal{B}_\mathcal{Y} = \Delta \mathcal{Y} \rightarrow \mathcal{D} , \quad \mathcal{B}_\emptyset = \Delta \emptyset
\] (1)
Processes

Definition: Behavior restriction

\[ c|_X = \Delta \{ (x, c(x)) / x \in X \} \] (2)
Processes

**Definition: Process**

For \( X \), a **finite** set of variables \((X \subset \mathcal{V})\), a \( X \)-process \( p \) is a nonempty set of \( X \)-behaviors; \( \mathbb{P}_X \) is the set of \( X \)-processes;

\( b_1 \in \mathbb{B}_{\{x, y\}} \)

\( b_2 \in \mathbb{B}_{\{x, y\}} \)

\( b_3 \in \mathbb{B}_{\{x, y\}} \)

\( p \)
Processes

**Definition: Process**

For \( X \), a finite set of variables \((X \subseteq \mathcal{V})\), a \( X \)-process \( p \) is a nonempty set of \( X \)-behaviors; \( \mathbb{P}_X \) is the set of \( X \)-processes;

\[ \Omega = \Delta \{ \emptyset \}, \quad \mathcal{U} = \Delta \emptyset \]  

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Processes

Definition: Process complement

For $X$, a finite set of variables ($X \subset \mathcal{V}$), the complement $\tilde{p}$ of a process $p \in \mathbb{P}_X$ is defined by:

$$p \in \mathbb{P}_X \implies \tilde{p} = \Delta (\mathbb{B}_X \setminus p) = \{b \in \mathbb{B}_X / b \notin p\} \quad (4)$$
Example

Let $p$ a process (with $\text{var}(p) = \{x, y\}$, and $x, y \in \mathbb{N}$), defined by the set of behaviors such that $x > 0 \land y$ is odd

then $\tilde{p}$ is the set of behaviors such that $x \leq 0 \lor y$ is even.
Processes

Definition: Process restriction and extension

When \( X, Y \) are finite sets of variables such that \( X \subseteq Y \subset \mathcal{V} \), \( Y \) nonempty,

\[
q|_X = \Delta \{ c|_X / c \in q \}
\]

\[
p|_Y = \Delta \{ c \in B_Y / c|_X \in p \}
\]
Processes

Example

Let \( p \) a process (with \( \text{var}(p) = \{x, y, z\} \), and \( x, y, z \in \mathbb{N} \)), defines by the set of behaviors such that \( x > 0 \land y \) is odd \( \land z < 2 \) then \( p|\{x, y\} \) is the set of behaviors such that \( x > 0 \land y \) is odd.

Example

Let \( p \) a process (with \( \text{var}(p) = \{x, y\} \), and \( x, y, z \in \mathbb{N} \)), defines by the set of behaviors such that \( x > 0 \land y \) is odd \( \land z \in \mathbb{N} \).

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Processes

Definition: Strict processes extension

For \( X, Y \) nonempty, finite sets of variables such that \( X \subseteq Y \subseteq \mathcal{V} \) and \( p \in \mathbb{P}_X \), \( p \preceq q \) states that \( q \) is a full extension of \( p \) to \( Y \): a variable in \( Y \setminus X \) may hold any legal value; thus

\[
(p \preceq q) \iff ((\text{var}(p) \subseteq \text{var}(q)) \land (p|_{\text{var}(q)} = q))
\]  

(7)

Corollary.

\((\mathbb{P}, \preceq)\) is a poset.
Definition: Strict processes extension

For $X, Y$ nonempty, finite sets of variables such that $X \subseteq Y \subset V$ and $p \in P_X$, $p \preceq q$ states that $q$ is a full extension of $p$ to $Y$: a variable in $Y \setminus X$ may hold any legal value; thus

$$\quad (p \preceq q) \iff ((\text{var}(p) \subseteq \text{var}(q)) \land (p|_{\text{var}(q)} = q)) \quad (7)$$

Corollary.

$(P, \preceq)$ is a poset.
Processes

The upper set of a process is the set of processes that contain its behaviors.

\[ [\leq \uparrow p] = \Delta \{ q \in \mathbb{P} / p \leq q \} \quad (8) \]
A Model for Contracts

Processes

Definition: Variable control

A process $q$ controls a variable $y$, denoted by $(q \triangleright y)$ iff

$$(y \in \text{var}(q)) \land q \subset ((q \upharpoonright (\text{var}(q) \setminus \{y\})) \upharpoonright \text{var}(q))$$

(9)
Processes

**Definition: Reduced process**

A process $p$ is *reduced* iff it controls all of its variables:

$$p \text{ is reduced } \iff p \triangleright var(p)$$  \hspace{1cm} (10)

We denote by $\downarrow q$, called *reduction of* $q$, the (minimal) process such that $q \preceq \downarrow q$
Processes

Definition: Reduced process

A process $p$ is reduced iff it controls all of its variables:

$$p \text{ is reduced } \iff p \triangleright \text{var}(p)$$

(10)

We denote by $\triangledown q$, called reduction of $q$, the (minimal) process such that $q \triangledown q$
A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
We define the *inclusion lower set* of a process to capture all the subsets of its behaviors. Let $\mathcal{P}^* = \Delta \mathcal{P} \cup \{\emptyset\}$, $\mathcal{R} \subseteq \mathcal{P}^*$, $[\mathcal{R} \downarrow \subseteq]$ is the lower set of $\mathcal{R}$ for $\subseteq$:

$$[\mathcal{R} \downarrow \subseteq] = \Delta \{ p \in \mathcal{P}^*/(\exists q \in \mathcal{R})(p \subseteq q)\} \quad (11)$$
**Process-filter**

**Definition: Process-filter**

Formally a set of processes $R$ is a *process-filter* iff ($\exists r \in P^*$)

$$(((r = \nabla r) \land (R = \{\nabla r\}) \subseteq R))$$

the process $r$ is a *generator* of $R$ and that $R$ is generated by $r$.

$\nabla R$ denotes the generator of $R$.

$\Phi$ is the set of process-filters.
Process-filter

Example

Let \( r \) a process (with \( \text{var}(r) = \{ x, y, z \} \), and \( x, y, z \in \mathbb{N} \)), defines by the set of behaviors such that \( x > 10 \land y \text{ is odd} \land z \in \mathbb{N} \) (\( z \) is a free variable).

Then process-filter \([\lceil\lceil \triangleleft r \rceil \rceil] \subseteq\]
defines the set of processes which satisfy \( x > 10 \land y \text{ is odd} \).

Let \( s \) a process (with \( \text{var}(s) = \{ x, y, u \} \), and \( x, y, u \in \mathbb{N} \)), defines by the set of behaviors such that \( x > 10 \land y \text{ is odd} \land u > 100 \). Then \( u \in \lceil\lceil \triangleleft r \rceil \rceil \).
The filtered variable set of $\mathbf{R}$ is $\text{var}(\mathbf{R})$ defined by:

$$\text{var}(\mathbf{R}) \triangleq \var(\mathbf{R})$$

(12)
Definition: Process-filter relaxation

For $R$ and $S$, two process-filters, the relation $R$ is less wide than $S$, written $R \sqsubseteq S$ is defined by:

$$Z = \text{var}(R) \cup \text{var}(S) \implies (R \sqsubseteq S \iff \nabla|Z \subseteq \nabla|Z)$$

Example
Let $x, y, z \in \mathbb{N}$ three variables. The process-filter $R$ defines the set of processes which satisfy $x > 10 \land y$ is odd $\land z$ is even, the process-filter $S$ defines the set of processes which satisfy $x > 0 \land y$ is odd, then we have $R \sqsubseteq S$. 
Process-filter

Definition: Process-filter relaxation

For \( R \) and \( S \), two process-filters, the relation \( R \) is less wide than \( S \), written \( R \sqsubseteq S \) is defined by:

\[
Z = \text{var}(R) \cup \text{var}(S) \quad \Rightarrow \quad (R \sqsubseteq S \iff \downarrow|Z \subseteq \downarrow|Z) \quad (13)
\]

Example

Let \( x, y, z \in \mathbb{N} \) three variables.
The process-filter \( R \) defines the set of processes which satisfy \( x > 10 \land y \text{ is odd} \land z \text{ is even} \),
The process-filter \( S \) defines the set of processes which satisfy \( x > 0 \land y \text{ is odd} \),
then we have \( R \sqsubseteq S \).
Corollary

\((\Phi, \sqsubseteq)\) is a poset.
A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Let $x, y, z \in \mathbb{N}$ three variables.

**Example**

The process-filter $R$ defines the set of processes which satisfy
$x > 10 \land y$ is odd $\land z$ is even,

The process-filter $S$ defines the set of processes which satisfy
$x > 10 \land y$ is even $\land z$ is odd,

then $R \sqcup S$ satisfies $x > 10$ ($y, z$ are free variables)
Process-filter

Let $x, y, z \in \mathbb{N}$ three variables.

**Example**

The process-filter $R$ defines the set of processes which satisfy $x > 10 \land y$ is odd $\land z$ is even,

The process-filter $S$ defines the set of processes which satisfy $x > 10 \land y$ is even $\land z$ is odd,

then $R \sqcup S$ satisfies $x > 10$ ($y, z$ are free variables)

**Example**

The process-filter $R$ defines the set of processes which satisfy $(x > 10 \land y \in \mathbb{N} \land z$ is even) $\lor (x = 3 \land y = 1 \land z = 1),$

The process-filter $S$ defines the set of processes which satisfy $x > 10 \land y \in \mathbb{N},$

then $R \sqcap S$ satisfies $x > 10 \land z$ is even ($y$ is a free variable)
Definition: Process-filter complement

The complement $\tilde{R}$ of a process-filter $R$ is defined by:

$$ (\tilde{R} = \Delta [\tilde{\nabla} R \subseteq \subseteq]) $$

(16)

$\text{var}(R) = \text{var}(\tilde{R})$

Theorem: Process-filter Boolean algebra

$(\Phi, \sqsubseteq)$ is a Boolean algebra with $\mathbb{P}^*$ as 1, $\{\emptyset\}$ as 0 and the complement $\tilde{R}$.
Definition: Process-filter complement

The complement $\tilde{R}$ of a process-filter $R$ is defined by:

$$(\tilde{R} = \Delta [\tilde{\triangledown} R] \subseteq \triangleleft)$$

(16)

$\text{var}(R) = \text{var}(\tilde{R})$

Theorem: Process-filter Boolean algebra

$(\Phi, \sqsubseteq)$ is a Boolean algebra with $\mathbb{P}^*$ as 1, $\{\emptyset\}$ as 0 and the complement $\tilde{R}$.
**Definition: Contract**

A *contract* $C = (A,G)$ is a *pair* of process-filters. $\text{var}(C)$, the *variable* set of $C = (A,G)$, is defined by $\text{var}(C) = \text{var}(A) \cup \text{var}(G)$. $\mathcal{C} = \Phi \times \Phi$ is the *set* of contracts.
**Defintion: Satisfaction**

Let $\mathbf{C} = (\mathbf{A}, \mathbf{G})$ a contract, $p$ a process:

$p \vdash \mathbf{C} \iff ([\hat{p}] \cap \mathbf{A}) \subseteq \mathbf{G}$

where $[\hat{p}]$ denote $[[\preceq \uparrow \hat{p}] \downarrow \subseteq]$. 

A Boolean algebra of contracts for assume-guarantee reasoning
Yann Glouche
Definition: Satisfaction preorder

A contract \((A_1, G_1)\) is \textit{finer} than a contract \((A_2, G_2)\), written \((A_1, G_1) \leadsto (A_2, G_2)\), iff all processes that satisfy the contract \((A_1, G_1)\) also satisfy the contract \((A_2, G_2)\):

\[
(A_1, G_1) \leadsto (A_2, G_2) \iff (\forall p \in \mathcal{P})( (p \models (A_1, G_1)) \implies (p \models (A_2, G_2)))
\]

(17)

The relation \textit{finer} on contracts satisfies the following property:

\[
(A_1, G_1) \leadsto (A_2, G_2) \iff \langle \tilde{A}_1 \sqcup G_1 \rangle \sqsubseteq \langle \tilde{A}_2 \sqcup G_2 \rangle
\]

(18)

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Definition: Satisfaction preorder

A contract \((A_1, G_1)\) is \textit{finer} than a contract \((A_2, G_2)\), written \((A_1, G_1) \sim (A_2, G_2)\), iff all processes that satisfy the contract \((A_1, G_1)\) also satisfy the contract \((A_2, G_2)\):

\[
(A_1, G_1) \sim (A_2, G_2) \iff (\forall p \in P)((p \models (A_1, G_1)) \implies (p \models (A_2, G_2)))
\]  

(17)

The relation \textit{finer} on contracts satisfies the following property:

\[
(A_1, G_1) \sim (A_2, G_2) \iff (\widetilde{A_1} \cup G_1) \sqsubseteq (\widetilde{A_2} \cup G_2)
\]

(18)
Definition: Filtering equivalence of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ are *filtering equivalent*, denoted $(A_1, G_1) \leftrightarrow (A_2, G_2)$, if and only if:

$((A_1, G_1) \rightarrow (A_2, G_2)) \land ((A_2, G_2) \rightarrow (A_1, G_1))$

Corollary

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ are *filtering equivalent* if and only if $(\sim A_1 \sqcup G_1) = (\sim A_2 \sqcup G_2)$. 
Definition: Filtering equivalence of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ are *filtering equivalent*, denoted $(A_1, G_1) \leftrightarrow (A_2, G_2)$, if and only if: $(A_1, G_1) \Rightarrow (A_2, G_2) \land (A_2, G_2) \Rightarrow (A_1, G_1)$

Corollary

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ are *filtering equivalent* if and only if $(\sim A_1 \sqcup G_1) = (\sim A_2 \sqcup G_2)$. 
Definition: Refinement of contracts

Let \( C_1 = (A_1, G_1) \) and \( C_2 = (A_2, G_2) \) two contracts. The contract \( C_1 \) refines the contract \( C_2 \), written \( C_1 \preceq C_2 \), if and only if the three following properties are satisfied:

(a) \( (A_1, G_1) \sim (A_2, G_2) \)

(b) \( A_2 \sqsubseteq A_1 \)

(c) \( G_1 \sqsubseteq A_1 \sqcup G_2 \)
A Boolean algebra of contracts for assume-guarantee reasoning

Corollary

$(C, \preceq)$ is a poset.
If we suppose $G_1$ is an abstraction of $A_1$: $(G_1 \sqsubseteq A_1)$ then $C_1 \preccurlyeq C_2$, if and only if the three following properties are satisfied:

(a) $(A_2 \sqcap G_1) \sqsubseteq G_2$
(b) $(A_2 \sqsubseteq A_1)$
(c) $(G_1 \sqsubseteq A_1)$
Lemma: Greatest lower bound of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ have a greatest lower bound $C = (A, G)$ defined by:

$$A = A_1 \sqcup A_2 \quad (19)$$

$$G = ((A_1 \cap \tilde{A}_2 \cap G_1) \sqcup (\tilde{A}_1 \cap A_2 \cap G_2) \sqcup (G_1 \cap G_2)) \quad (20)$$

Lemma: Least upper bound of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ have a least upper bound $C = (A, G)$ defined by:

$$A = A_1 \sqcap A_2 \quad (21)$$

$$G = (\tilde{A}_1 \cap G_1) \sqcup (\tilde{A}_2 \cap G_2) \sqcup (A_1 \cap G_2) \sqcup (A_2 \cap G_1) \quad (22)$$
Lemma: Greatest lower bound of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ have a greatest lower bound $C = (A, G)$ defined by:

$$A = A_1 \sqcup A_2$$ (19)

$$G = ((A_1 \cap \neg A_2 \cap G_1) \sqcup (\neg A_1 \cap A_2 \cap G_2) \sqcup (G_1 \cap G_2))$$ (20)

Lemma: Least upper bound of contracts

Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ have a least upper bound $C = (A, G)$ defined by:

$$A = A_1 \sqcap A_2$$ (21)

$$G = (\neg A_1 \cap G_1) \sqcup (\neg A_2 \cap G_2) \sqcup (A_1 \cap G_2) \sqcup (A_2 \cap G_1)$$ (22)
Corollary

$(C, \preceq)$ is a distributive lattice with $(\{\emptyset\}, \mathbb{P}^\ast)$ as supremum and $(\mathbb{P}^\ast, \{\emptyset\})$ as infimum.
The lattice of contracts *filtering equivalent* to \((A,G)\) is presented using the following notations for filters:

<table>
<thead>
<tr>
<th></th>
<th>({\cup})</th>
<th>(A \sqcap \tilde{G})</th>
<th>(A \sqcap G)</th>
<th>(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>({\cup})</td>
<td>(A \sqcap \tilde{G})</td>
<td>(A \sqcap G)</td>
<td>(A)</td>
</tr>
<tr>
<td>1</td>
<td>(\tilde{A} \sqcap \tilde{G})</td>
<td>(\tilde{G})</td>
<td>((A \sqcap G) \cup (\tilde{A} \sqcap \tilde{G}))</td>
<td>(A \sqcup \tilde{G})</td>
</tr>
<tr>
<td>2</td>
<td>(\tilde{A} \sqcap G)</td>
<td>((A \sqcap \tilde{G}) \cup (\tilde{A} \sqcap G))</td>
<td>(G)</td>
<td>(A \sqcup \tilde{G})</td>
</tr>
<tr>
<td>3</td>
<td>(\tilde{A})</td>
<td>(\tilde{A} \sqcup \tilde{G})</td>
<td>(4,11)</td>
<td>(4,3)</td>
</tr>
</tbody>
</table>

A Boolean algebra of contracts for assume-guarantee reasoning

Yann Glouche
Introduction
- Context
- Motivations
- Goals

A Model for Contracts
- Process
- Process-filter
- Contract

Use Case

Conclusion

Further work
We illustrate the distinctive features of our contract algebra by considering the specification of a four-stroke engine and its translation into observers in the synchronous language Signal.
The **successive** operation modes of a 4-stroke engine: **Intake, Compression, Combustion and Exhaust**. They are driven by the **camshaft** whose position is measured in degrees.
We wish to define a contract to stipulate that intake always takes place in the first quarter on the camshaft revolution.

\[ A_{\text{Intake}} = \text{cam modulo } 360^\circ < 90 \quad G_{\text{Intake}} = \text{Intake} \]

The complementary is simply defined by \( \tilde{A}_{\text{Intake}} = \text{cam modulo } 360^\circ \geq 90 \).

The generic structure of processes in contracts finds a direct instance and compositional translation into the synchronous multi-clocked model of computation of Signal.

\[ A_{\text{intake}} = \text{true when } (\text{cam modulo } 360 < 90) \]
\[ G_{\text{intake}} = \text{true when intake default false} \]
We wish to define a contract to stipulate that \textit{intake} always takes place in the first quarter on the \textit{camshaft} revolution. 

\[ A_{\text{Intake}} = \text{cam modulo } 360^\circ < 90 \quad G_{\text{Intake}} = \text{Intake} \]

The \textit{complementary} is simply defined by \[ \sim A_{\text{Intake}} = \text{cam modulo } 360^\circ \geq 90. \]

The generic structure of processes in contracts finds a \textit{direct} instance and compositional translation into the synchronous multi-clocked model of computation of \textit{Signal}. 

\[ A_{\text{intake}} = \text{true when } (\text{cam modulo } 360^\circ < 90) \]
\[ G_{\text{intake}} = \text{true when intake default false} \]
We wish to define a contract to stipulate that intake always takes place in the first quarter on the camshaft revolution.

\[ A_{\text{Intake}} = \text{cam modulo } 360^\circ < 90 \quad G_{\text{Intake}} = \text{Intake} \]

The complementary is simply defined by \( \overline{A}_{\text{Intake}} = \text{cam modulo } 360^\circ \geq 90 \).

The generic structure of processes in contracts finds a direct instance and compositional translation into the synchronous multi-clocked model of computation of Signal.

\[ A_{\text{intake}} = \text{true when } (\text{cam modulo } 360^\circ < 90) \]
\[ G_{\text{intake}} = \text{true when intake default false} \]
1. Introduction
   - Context
   - Motivations
   - Goals

2. A Model for Contracts
   - Process
   - Process-filter
   - Contract

3. Use Case

4. Conclusion

5. Further work

A Boolean algebra of contracts for assume-guarantee reasoning
Yann Glouche
We introduced the notion of process-filters.
We introduced the notion of process-filters.

A main result is that the structure of process-filters is a Boolean algebra.
We introduced the notion of process-filters.

A main result is that the structure of process-filters is a Boolean algebra.

and applied it to the specification of a component-based design process.
1. Introduction
   - Context
   - Motivations
   - Goals

2. A Model for Contracts
   - Process
   - Process-filter
   - Contract

3. Use Case

4. Conclusion

5. Further work
Further work

1. Develop a module system based on the paradigm of contract for a synchronous multi-clocked formalism, SIGNAL,
Further work

1. Develop a module system based on the paradigm of contract for a synchronous multi-clocked formalism, SIGNAL.

2. Develop a prototype of compiler for the language based on typing by contracts.