The Regorous Approach to Business Process Compliance

Guido Governatori

Last JD, 20 October 2013
Outline

- Definition of (Business Process) Compliance
- Process specifications
- Normative requirements
  - Deontic Concepts
  - Logic of Violation
  - Defeasibility
- Compliance-by-design
- The Regorous approach
Ensuring that business operations, processes, and practices are in accordance with a given prescriptive (often legal) document
What is Compliance?

Ensuring that business operations, processes, and practices are in accordance with a given prescriptive (often legal) document

<table>
<thead>
<tr>
<th>Regulatory</th>
<th>Standards</th>
<th>Contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basel II</td>
<td>Best practice models</td>
<td>Service Agreement</td>
</tr>
<tr>
<td>Sarbanes-Oxley</td>
<td>SAP solution maps</td>
<td>Customer Contract</td>
</tr>
<tr>
<td>OFAC (USA Patriot Act)</td>
<td>ISO 9000</td>
<td>Warranty</td>
</tr>
<tr>
<td>OSFI “blocked entity” lists</td>
<td>Medical guidelines</td>
<td>Insurance Policy</td>
</tr>
<tr>
<td>HIPAA</td>
<td></td>
<td>Business Partnership</td>
</tr>
<tr>
<td>Graham-Leach-Bliley</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to ensure compliance?

Compliance is a relationship between two sets of specifications:

- Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.

   - Conceptually sound representation of processes
   - Conceptually sound representation of and reasoning with norms

We can identify formal loopholes, deadlocks and inconsistencies in normative systems, and make hidden conditions explicit.
How to ensure compliance?

Compliance is a relationship between two sets of specifications

- Conceptually sound representation of processes
- Conceptually sound representation of and reasoning with norms
- We can identify formal loopholes, deadlocks and inconsistencies in normative systems,
- We can make hidden conditions explicit
How to ensure compliance?

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.
How to ensure compliance?

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.

- Conceptually sound representation of processes
How to ensure compliance?

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.

- Conceptually sound representation of processes
- Conceptually sound representation of and reasoning with norms
How to ensure compliance?

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.

- Conceptually sound representation of processes
- Conceptually sound representation of and reasoning with norms
  - we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and
How to ensure compliance?

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive (legal) documents.

- Conceptually sound representation of processes
- Conceptually sound representation of and reasoning with norms
  - we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and
  - we can make hidden conditions explicit
Part I

Business Process Models
Business Process Model

Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)
Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:
Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:

- Tasks are activities to be performed
Business Process Model

Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:

- Tasks are activities to be performed
- Connectors consist of:
  - sequence (a task is performed after another task),
  - parallel—and-split and and-join—(tasks are to be executed in parallel),
  - choice—(x)or-split and (x)or-join—(at least (most) one task in a set of tasks must be executed).
Business Process Model

Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:

- Tasks are activities to be performed
- Connectors consist of
  - sequence (a task is performed after another task),
Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:

- Tasks are activities to be performed
- Connectors consist of
  - parallel—and-split and and-join—(tasks are to be executed in parallel),
Self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. It describes:

- What needs be done and when (control flows)
- What we need to work on (data)
- Who is doing the work (human and system resources)

A language for BPM usually has two elements:

- Tasks are activities to be performed
- Connectors consist of
  - parallel—and-split and and-join—(tasks are to be executed in parallel),
  - choice—(x)or-split and (x)or-join—(at least (most) one task in a set of task must be executed).
Business Process Model Example
Execution Traces

$t_1 : A, B, C, D, E, F, H$

$t_2 : A, D, B, C, E, G, H$

$t_3 : A, D, B, C, E, F, H$

\ldots
Trace:
From sequence of tasks to sequence of states

---

Let $\text{Lit}$ be a set of literals, $T$ be the set of traces of a process and $\mathbb{N}$ be the set of natural numbers

$\text{State}: T \times \mathbb{N} \mapsto 2^{\text{Lit}}$

The function $\text{State}$ returns the set of literals describing “what’s going on in a trace $t$ after the execution of the $n$-th task in the process”.

Example

Tasks
• A: "turn the light on"
• B: "check if glass is empty"
• C: "fill glass with water"
• D: "turn glass upside-down"

Propositions
• p: "the light is on"
• q: "the glass is full"

Trace 1: ⟨A, B, D⟩
Trace 2: ⟨A, B, C, D⟩
Example

Tasks

- A: “turn the light on”
- B: “check if glass is empty”
- C: “fill glass with water”
- D: “turn glass upside-down”

Propositions

- \(p\): “the light is on”
- \(q\): “the glass is full”
Example

Tasks
- A: “turn the light on”
- B: “check if glass is empty”
- C: “fill glass with water”
- D: “turn glass upside-down”

Propositions
- p: “the light is on”
- q: “the glass is full”

Trace 1: \( \langle A, B, D \rangle \)
Trace 2: \( \langle A, B, C, D \rangle \)
Example

Tasks
- A: “turn the light on”
- B: “check if glass is empty”
- C: “fill glass with water”
- D: “turn glass upside-down”

Propositions
- p: “the light is on”
- q: “the glass is full”

Trace 1: $\langle A, B, D \rangle$
Trace 2: $\langle A, B, C, D \rangle$

- $State(i, 1) = \{ p \}, \ i \in \{ 1, 2 \}$
- $State(1, 2) = \{ p, q \}$
- $State(2, 2) = \{ p, \neg q \}$
- $State(2, 3) = \{ p, q \}$
- $State(1, 3) = \{ p, \neg q \}$
- $State(2, 4) = \{ p, \neg q \}$
Part II

Modelling Norms
Key components of Normative Systems

A normative system is a set of clauses (norms).
A normative system is a set of clauses (norms). Norms are modelled as **if . . . then** rules

\[ A_1, \ldots, A_n \implies C \]

- Definitional clauses (constitutive rules: defining terms used in a legal context)
- Prescriptive clauses (norms defining “normative effects”)
  - obligations
  - permissions
  - prohibitions
  - violations
A normative system is a set of clauses (norms). Norms are modelled as if ... then rules

\[ A_1, \ldots, A_n \Rightarrow C \]

- Definitional clauses (constitutive rules: defining terms used in a legal context)
- Prescriptive clauses (norms defining “normative effects”)
  - obligations
  - permissions
  - prohibitions
  - violations

Norms are defeasible (handling exceptions)
Normative Effects

Obligation  A situation, an act, or a course of action to which a bearer is legally bound, and if it is not achieved or performed results in a violation.

Prohibition  A situation, an act, or a course of action which a bearer should avoid, and if it is achieved results in a violation.

Permission  Something is permitted if the obligation or the prohibition to the contrary does not hold.
Example

Contract fragment

3.1 A “Premium Customer” is a customer who has spent more than $10000 in goods.

3.2 Services marked as “special order” are subject to a 5% surcharge. Premium customers are exempt from special order surcharge.

5.2 The (Supplier) shall on receipt of a purchase order for (Services) make them available within one day.

5.3 If for any reason the conditions stated in 4.1 or 4.2 are not met the (Purchaser) is entitled to charge the (Supplier) the rate of $100 for each hour the (Service) is not delivered.
Defeasibility:
Reasonable results with minimum effort
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[ PhD \rightarrow Uni \]
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[ \text{PhD} \rightarrow \text{Uni} \]
\[ \text{Weekend} \rightarrow \neg \text{Uni} \]
\[ \text{PublicHoliday} \rightarrow \neg \text{Uni} \]
\[ \text{Sick} \rightarrow \neg \text{Uni} \]
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[
\begin{align*}
\text{PhD} & \rightarrow \text{Uni} \\
\text{Weekend} & \rightarrow \neg \text{Uni} \\
\text{PublicHoliday} & \rightarrow \neg \text{Uni} \\
\text{Sick} & \rightarrow \neg \text{Uni} \\
\text{Weekend} \land \text{VICdeadline} & \rightarrow \text{Uni}
\end{align*}
\]
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[ \text{PhD} \rightarrow \text{Uni} \]
\[ \text{Weekend} \rightarrow \neg \text{Uni} \]
\[ \text{PublicHoliday} \rightarrow \neg \text{Uni} \]
\[ \text{Sick} \rightarrow \neg \text{Uni} \]
\[ \text{Weekend} \land \text{VICdeadline} \rightarrow \text{Uni} \]

VIC= Very Important Conference
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[
\begin{align*}
\text{PhD} & \rightarrow \text{Uni} \\
\text{Weekend} & \rightarrow \neg \text{Uni} \\
\text{PublicHoliday} & \rightarrow \neg \text{Uni} \\
\text{Sick} & \rightarrow \neg \text{Uni} \\
\text{Weekend} \land \text{VICdeadline} & \rightarrow \text{Uni} \\
\text{VICdeadline} \land \text{PartnerBirthday} & \rightarrow \neg \text{Uni}
\end{align*}
\]
Defeasibility:
Reasonable results with minimum effort

Factual omniscience and (non-)monotonic reasoning

\[ \text{PhD} \rightarrow \text{Uni} \]
\[ \text{Weekend} \rightarrow \neg \text{Uni} \]
\[ \text{PublicHoliday} \rightarrow \neg \text{Uni} \]
\[ \text{Sick} \rightarrow \neg \text{Uni} \]
\[ \text{Weekend} \land \text{VICdeadline} \rightarrow \text{Uni} \]
\[ \text{VICdeadline} \land \text{PartnerBirthday} \rightarrow \neg \text{Uni} \]

\[ \text{Phd} \land (\neg \text{Weekend} \lor (\text{Weekend} \land \text{VICdeadline} \land \neg \text{PartnerBirthday})) \land \neg \text{Sick} \ldots \rightarrow \text{Uni} \]
Defeasibility: Example 1

TELECOMMUNICATIONS CONSUMER PROTECTIONS CODE
(C628:2012)
Section 2.1. Definitions

**Complaint** means an expression of dissatisfaction made to a Supplier in relation to its Telecommunications Products or the complaints handling process itself, where a response or Resolution is explicitly or implicitly expected by the Consumer.

An initial call to a provider to request a service or information or to request support is not necessarily a Complaint. An initial call to report a fault or service difficulty is not a Complaint. However, if a Customer advises that they want this initial call treated as a Complaint, the Supplier will also treat this initial call as a Complaint.

If a Supplier is uncertain, a Supplier must ask a Customer if they wish to make a Complaint and must rely on the Customer’s response.
Defeasibility: Example 2

NATIONAL CONSUMER CREDIT PROTECTION ACT 2009 (Act No. 134 of 2009) Section 29

(1) A person must not engage in a credit activity if the person does not hold a licence authorising the person to engage in the credit activity.

(3) For the purposes of subsections (1) and (2), it is a defence if:

(a) the person engages in the credit activity on behalf of another person (the principal); and

(b) the person is:

(i) an employee or director of the principal or of a related body corporate of the principal; or

(ii) a credit representative of the principal; and . . .
Modelling Obligations: Deontic Logic

Extension of logic with the operators OBL and PER.

- \( \text{SpecialOrderPrice}(x) = \text{Price}(x) + 5\% \)
- \( \text{OBL}_{\text{Supplier}} \text{MakeGoodsAvailable1Day} \)
- \( \text{PER}_{\text{Purchaser}} \text{ChargeSupplier} \)
Standard Deontic Logic

Extension of classical logic with the modal operators OBL and PER.

- $\text{OBL}\alpha \equiv \neg \text{PER}\neg\alpha$, $\text{PER}\alpha \equiv \neg \text{OBL}\neg\alpha$
- $\text{OBL}(\alpha \rightarrow \beta) \rightarrow (\text{OBL}\alpha \rightarrow \text{OBL}\beta)$
- $\text{OBL}\alpha \rightarrow \text{PER}\alpha$ or $(\text{OBL}\alpha \rightarrow \neg \text{OBL}\neg\alpha)$
Standard Deontic Logic

Extension of classical logic with the modal operators OBL and PER.

- \( \text{OBL}\alpha \equiv \neg \text{PER}\neg\alpha \), \( \text{PER}\alpha \equiv \neg \text{OBL}\neg\alpha \)
- \( \text{OBL}(\alpha \rightarrow \beta) \rightarrow (\text{OBL}\alpha \rightarrow \text{OBL}\beta) \)
- \( \text{OBL}\alpha \rightarrow \text{PER}\alpha \) or \( \text{OBL}\alpha \rightarrow \neg \text{OBL}\neg\alpha \)

Standard Deontic Logic is not able to deal with violations
### Violation paradox

#### Rules for Last JD Lectures

- Guido should not tell lies in his presentation
- If Guido tells a lie then he has to explain why
- It ought to be the case that if Guido does not tell a lie then he does not explain why
- Guido tells lies in his presentation
Rules for Last JD Lectures

- Guido should not tell lies in his presentation
- If Guido tells a lie then he has to explain why
- It ought to be the case that if Guido does not tell a lie then he does not explain why
- Guido tells lies in his presentation

- \( \text{OBL} \neg \text{lie} \)
- \( \text{lie} \rightarrow \text{OBLexplain} \)
- \( \text{OBL}(\neg \text{lie} \rightarrow \neg \text{explain}) \)
- \( \text{lie} \)
Violation paradox

Rules for Last JD Lectures

- Guido should not tell lies in his presentation
- If Guido tells a lie then he has to explain why
- It ought to be the case that if Guido does not tell a lie then he does not explain why
- Guido tells lies in his presentation

OBL$\neg$\textit{lie}

$\textit{lie} \rightarrow$ OBL$\textit{explain}$

OBL($\neg\textit{lie} \rightarrow \neg\textit{explain}$)

$\textit{lie}$

OBL$\textit{explain}$ and OBL$\neg$\textit{explain}
Kobayashi Maru
What’s the problem?
What’s the problem?

\[ a \Rightarrow Ob \quad \neg b \Rightarrow Oc \quad \neg c \Rightarrow Od \]
What’s the problem?

\[
a \Rightarrow Ob \\
\neg b \Rightarrow Oc \\
\neg c \Rightarrow Od
\]

What about \( a, \neg b \) but \( d \)?
What’s the problem?

\[ a \Rightarrow Ob \]
\[ \neg b \Rightarrow Oc \]
\[ \neg c \Rightarrow Od \]

What about \( a \), \( \neg b \) but \( d \)?

A logic of violations
A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$. 
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.
2. A violation does not exist without an obligation it violates.
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.
2. A violation does not exist without an obligation it violates.
3. A reparation of a violation does not exist without a violation it repairs.
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.
2. A violation does not exist without an obligation it violates.
3. A reparation of a violation does not exist without a violation it repairs.
4. A reparation can be an obligation itself, and thus it can be violated.
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.
2. A violation does not exist without an obligation it violates.
3. A reparation of a violation does not exist without a violation it repairs.
4. A reparation can be an obligation itself, and thus it can be violated.
5. Permissions cannot be violated.
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.

2. A violation does not exist without an obligation it violates.

3. A reparation of a violation does not exist without a violation it repairs.

4. A reparation can be an obligation itself, and thus it can be violated.

5. Permissions cannot be violated.

- Prescriptive clauses cannot be taken in isolation.
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.
2. A violation does not exist without an obligation it violates.
3. A reparation of a violation does not exist without a violation it repairs.
4. A reparation can be an obligation itself, and thus it can be violated.
5. Permissions cannot be violated.

- Prescriptive clauses cannot be taken in isolation.
- It is possible to have chains of obligations/violations
Logic of Violations

1. A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.

2. A violation does not exist without an obligation it violates.

3. A reparation of a violation does not exist without a violation it repairs.

4. A reparation can be an obligation itself, and thus it can be violated.

5. Permissions cannot be violated.

- Prescriptive clauses cannot be taken in isolation.
- It is possible to have chains of obligations/violations
- New prescriptive clauses can be derived from the given prescriptive clauses
Modelling Norms

Norms are modelled as rules in FCL.

**Language**
- literals $p, q, \ldots$ (atomic proposition and their negation)
- deontic literals $Op$ (Obligatory $p$), $P$ (Permitted $p$), $Fp$ (Forbidden $p$, i.e., $O\neg p$)

**Rules**
- Normal rules
  $$A_1, \ldots, A_n \Rightarrow OB$$
  $A_1 \ldots, A_n$ trigger the obligation of $B$.
- Rules for violations
  $$A_1, \ldots, A_n \Rightarrow OB_1 \otimes OB_2 \otimes OB_3 \otimes \cdots \otimes OB_n$$
  $A_1 \ldots, A_n$ trigger the obligation of $B_1$ but if $B_1$ is violated then $B_2$ is obligatory and so on.
Modelling Obligations

Let $\text{Lit}$ be a set of literals, $T$ be the set of traces of a process and $\mathbb{N}$ be the set of natural numbers.

$$\text{Force}: T \times \mathbb{N} \rightarrow 2^{\text{Lit}}$$
Let $\text{Lit}$ be a set of literals, $T$ be the set of traces of a process and $\mathbb{N}$ be the set of natural numbers

$$\text{Force} : T \times \mathbb{N} \mapsto 2^{\text{Lit}}$$

The function $\text{Force}$ returns the set of literals describing what is obligatory for a particular task.
Persistent vs immediate obligations

- An **immediate** (or punctual or non-persistent) obligation must be satisfied in the task where it occurs.
  ‘complaints in person or by phone must be acknowledge immediately’

- A **persistent** obligation is activated and it remain in force in the future after it has been activated.
  ‘A service provider must not disclose personal information without the written consent of the customer’
Definition (Punctual Obligation)

An obligation \( o \) is a **punctual obligation** in \( t \) if and only if

\[
\exists n \in \mathbb{N} : o \notin \text{Force}(t, n-1), \quad o \notin \text{Force}(t, n+1), \quad o \in \text{Force}(t, n).
\]

A punctual obligation \( o \) is **violated** in \( t \) if and only if \( o \notin \text{State}(t, n) \).
Graphical Illustration of a Punctual Obligation

\[ o \in \text{Force}(t, n) \]

\[ o \notin \text{State}(t, n) \]

violation of \( o \)
Persistent Obligations: Achievement vs Maintenance

- For an **achievement obligation**, a certain condition must occur at least once before the deadline:
  ‘Customers must pay before the delivery of the good, after receiving the invoice’

- For **maintenance obligations**, a certain condition must obtain during all instants before the deadline:
  ‘After opening a bank account, customers must keep a positive balance until bank charges are taken out’
An obligation $o$ is a **maintenance obligation** in $t$ if and only if

$$\exists n, m \in \mathbb{N} : n < m,$$

$$o \notin \text{Force}(t, n-1),$$

$$o \notin \text{Force}(t, m+1),$$

$$\forall k : n \leq k \leq m, o \in \text{Force}(t, k)$$

A maintenance obligation $o$ is **violated** in $t$ if and only if

$$\exists k : n \leq k \leq m, o \notin \text{State}(t, k).$$
Modelling Maintenance Obligations

Definition (Maintenance Obligation)

An obligation $o$ is a **maintenance obligation** in $t$ if and only if

$$\exists n, m \in \mathbb{N} : n < m,$$

$$o \notin \text{Force}(t, n - 1),$$

$$o \notin \text{Force}(t, m + 1),$$

$$\forall k : n \leq k \leq m, o \in \text{Force}(t, k)$$

A maintenance obligation $o$ is **violated** in $t$ if and only if

$$\exists k : n \leq k \leq m, o \notin \text{State}(t, k).$$

It is possible to relax/strengthen the condition by dropping the conditions on $m$.
Modelling Maintenance Obligations

Definition (Maintenance Obligation)

An obligation $o$ is a **maintenance obligation** in $t$ if and only if

$$\exists n, m \in \mathbb{N}: n < m,$$

$$o \notin \text{Force}(t, n - 1),$$

$$o \notin \text{Force}(t, m + 1),$$

$$\forall k: n \leq k \leq m, o \in \text{Force}(t, k)$$

A maintenance obligation $o$ is **violated** in $t$ if and only if

$$\exists k: n \leq k \leq m, o \notin \text{State}(t, k).$$

It is possible to relax/strengthen the condition by dropping the conditions on $m$.

Maintenance obligations can be used to model prohibitions.
Graphical Illustration of a Maintenance Obligation

\[ o \in \text{State}(t, k) \]

violation of \( o \)
Achievement Obligations: Preemptive vs Non-preemptive

- **preemptive obligations**: the fulfillment of an obligation can happen before the obligation has been triggered.
  
  (1) A report under section 53 must be given:
  
  (a) if the movement of the physical currency is to be effected by a person bringing the physical currency into Australia with the person—at the time worked out under subsection (2); or
  
  [...]
  
  (d) in any other case—at any time before the movement of the physical currency takes place.’

- **non preemptive obligations**: the fulfillment of an obligation can happen only after the obligation has been triggered.
  ‘Executors and administrators of a decedent’s estate will be required to give notice to each beneficiary named in the Will within 60 days after the date when an order admitting a will to probate has been signed.’
Modelling Achievement Obligations

Definition (Achievement Obligation)

An obligation $o$ is an **achievement obligation** in $t$ if and only if

$$\exists n, m \in \mathbb{N}: n < m,$$

$$o \notin Force(t, n - 1),$$

$$o \notin Force(t, m + 1),$$

$$\forall k: n \leq k \leq m, o \in Force(t, k)$$

An achievement obligation $o$ is **violated** in $t$ if and only if

- $o$ is preemptive and $\forall k: k \leq m, o \notin State(t, k);$  
- $o$ is non-preemptive and $\forall k: n \leq k \leq m, o \notin State(t, k).$
Graphical Illustration of Achievement Obligations

Achievement preemptive

\[ t \quad - \quad n - 1 \quad \bullet \quad n \quad \bullet \quad m \quad m + 1 \quad z \]

\[ o \notin \text{State} \quad \text{violation of } o \]

Achievement non-preemptive

\[ t \quad - \quad n - 1 \quad \bullet \quad n \quad \bullet \quad m \quad m + 1 \quad z \]

\[ o \notin \text{State} \quad \text{violation of } o \]
Perdurant vs Non-Perdurant

- **perdurant obligation**: the violation of the obligation does not extinguish the obligation itself.
  ‘A billing error must be fixed in the next billing cycle’

- **non-perdurant obligation**: the violation of the obligation terminates the obligation.
  ‘The assignment must be submitted by the due date’
Modelling Perdurant Obligations

Definition (Perdurant Obligation)

An obligation \( o \) is a **perdurant obligation** in \( t \) if and only if

\[
\exists n, m \in \mathbb{N}: n < m, \\
\quad o \notin \text{Force}(t, n - 1), \\
\quad o \notin \text{Force}(t, m + 1), \\
\forall k : n \leq k \leq m, o \in \text{Force}(t, k)
\]

A perdurant obligation \( o \) is **violated** in \( t \) if and only if

\[
\exists k : n < k < m, \forall j, j \leq k, o \notin \text{State}(t, j)
\]
Graphical Illustration of Perdurant Obligations

\[ t \sim o \notin \text{Force} \sim n \rightarrow o \in \text{Force} \sim n \rightarrow o \notin \text{Force} \sim d \rightarrow m \rightarrow m+1 \rightarrow z \sim o \notin \text{State}(t, k) \sim \text{violation of } o \]
Compensable vs Non-compensable

- **compensable obligations**: fulfilling the penalty related to the violation of the obligation makes the process compliant. ‘Each complaint must be either resolved or escalated and reported to the Telecommunication Ombudsmen’

- **non-compensable obligations**: violating the obligation makes the process non-compliant. ‘To pass the course a student has to pass the final exam’
Compensations or Acceptable Alternatives?

TCPC 2012, Section 8.1.1
...implement, operate and comply with a Complaint handling process that:

(vii) requires all Complaints to be:

A. Resolved in an objective, efficient and fair manner; and
B. escalated and managed under the Supplier’s internal escalation process if requested by the Consumer or a former Customer.
TCPC 2012, Section 8.1.1
...implement, operate and comply with a Complaint handling process that:

(vii) requires all Complaints to be:

A. Resolved in an objective, efficient and fair manner; and
B. escalated and managed under the Supplier’s internal escalation process if requested by the Consumer or a former Customer.

YAWL Deed of Assignment, Clause 5.2.
Each Contributor indemnifies and will defend the Foundation against any claim, liability, loss, damages, cost and expenses suffered or incurred by the Foundation as a result of any breach of the warranties given by the Contributor under **clause 5.1**.
**Modelling Compensations**

**Definition (Compensation)**

A *compensation* is a function $\text{Comp}: \text{Lit} \mapsto 2^{\text{Lit}}$.

**Definition (Compensable Obligation)**

An obligation $o$ is *compensable* in $t$ if and only if $\text{Comp}(o) \neq \emptyset$ and $\forall o' \in \text{Comp}(o), \exists n \in \mathbb{N} : o' \in \text{Force}(t, n)$.

**Definition (Compensated Obligation)**

An obligation $o$ is compensated in $t$ if and only if it is violated and for every $o' \in \text{Comp}(o)$ either:

1. $o'$ is not violated in $t$, or
2. $o'$ is compensated in $t$. 
Why Defeasible Logic

Rule-based non-monotonic formalism

- Flexible
- Efficient (linear complexity)
- Directly skeptic semantics
- Argumentation semantics
- Constrictive proof theory
- Encompasses other non-monotonic formalisms used in AI and Law
- Applied in several fields/optimised implementations
- Extensible
Why Defeasible Logic

Rule-based non-monotonic formalism

- Flexible
- Efficient (linear complexity)
- Directly skeptic semantics
- Argumentation semantics
- Constructive proof theory
- Encompasses other non-monotonic formalisms used in AI and Law
- Applied in several fields/optimised implementations
- Extensible
Basics of Defeasible Logic

- Derive (plausible) conclusions with the minimum amount of information.
  - Definite conclusions
  - Defeasible conclusions
- Defeasible Theory
  - Facts
  - Strict rules ($A \rightarrow B$)
  - Defeasible rules ($A \Rightarrow B$)
  - Defeaters ($A \sim B$)
  - Superiority relation over rules
Proving Conclusions in Defeasible Logic

1. Give an argument for the conclusion you want to prove
Proving Conclusions in Defeasible Logic

1. Give an argument for the conclusion you want to prove
2. Consider all possible counterarguments to it
Proving Conclusions in Defeasible Logic

1. Give an argument for the conclusion you want to prove
2. Consider all possible counterarguments to it
3. Rebut all counterarguments
Give an argument for the conclusion you want to prove

Consider all possible counterarguments to it

Rebut all counterarguments
  • Defeat the argument by a stronger one
  • Undercut the argument by showing that some of the premises do not hold
Example

Facts: $A_1, A_2, B_1, B_2$

Rules: $r_1 : A_1 \Rightarrow C$
        $r_2 : A_2 \Rightarrow C$
        $r_3 : B_1 \Rightarrow \neg C$
        $r_4 : B_2 \Rightarrow \neg C$
        $r_5 : B_3 \Rightarrow \neg C$

Superiority relation:
    $r_1 > r_3$
    $r_2 > r_4$
    $r_5 > r_1$
Example

Facts: \( A_1, A_2, B_1, B_2 \)

Rules:
\[
\begin{align*}
  r_1 &: A_1 \Rightarrow C \\
  r_2 &: A_2 \Rightarrow C \\
  r_3 &: B_1 \Rightarrow \neg C \\
  r_4 &: B_2 \Rightarrow \neg C \\
  r_5 &: B_3 \Rightarrow \neg C
\end{align*}
\]

Superiority relation:
\[
\begin{align*}
  r_1 &> r_3 \\
  r_2 &> r_4 \\
  r_5 &> r_1
\end{align*}
\]

Phase 1: Argument for \( C \)

Phase 2: Possible counterarguments

Phase 3: Rebut the counterarguments
Example

Facts: \( A_1, A_2, B_1, B_2 \)

Rules:
\[
\begin{align*}
  r_1 &: A_1 \implies C \\
r_2 &: A_2 \implies C \\
r_3 &: B_1 \implies \neg C \\
r_4 &: B_2 \implies \neg C \\
r_5 &: B_3 \implies \neg C \\
\end{align*}
\]

Superiority relation:
\[
\begin{align*}
  r_1 &> r_3 \\
r_2 &> r_4 \\
r_5 &> r_1 \\
\end{align*}
\]

Phase 1: Argument for \( C \)
\( A_1 \) (Fact), \( r_1 : A_1 \implies C \)

Phase 2: Possible counterarguments
\( r_3 : B_1 \implies \neg C \)
\( r_4 : B_2 \implies \neg C \)
\( r_5 : B_3 \implies \neg C \)

Phase 3: Rebut the counterarguments
\( r_5 \) is not applicable
\( r_3 \) weaker than \( r_1 \)
\( r_2 \) weaker than \( r_4 \)
Example

Facts: $A_1, A_2, B_1, B_2$

Rules:
- $r_1: A_1 \Rightarrow C$
- $r_2: A_2 \Rightarrow C$
- $r_3: B_1 \Rightarrow \neg C$
- $r_4: B_2 \Rightarrow \neg C$
- $r_5: B_3 \Rightarrow \neg C$

Superiority relation:
- $r_1 > r_3$
- $r_2 > r_4$
- $r_5 > r_1$

Phase 1: Argument for $C$
- $A_1$ (Fact), $r_1: A_1 \Rightarrow C$

Phase 2: Possible counterarguments
- $r_3: B_1 \Rightarrow \neg C$
- $r_4: B_2 \Rightarrow \neg C$
- $r_5: B_3 \Rightarrow \neg C$

Phase 3: Rebut the counterarguments
- $r_3$ is weaker than $r_1$
- $r_4$ is weaker than $r_2$
- $r_5$ is not applicable
Example

Facts: $A_1, A_2, B_1, B_2$

Rules:
- $r_1: A_1 \Rightarrow C$
- $r_2: A_2 \Rightarrow C$
- $r_3: B_1 \Rightarrow \neg C$
- $r_4: B_2 \Rightarrow \neg C$
- $r_5: B_3 \Rightarrow \neg C$

Phase 1: Argument for $C$
- $A_1$ (Fact), $r_1: A_1 \Rightarrow C$

Phase 2: Possible counterarguments
- $r_3: B_1 \Rightarrow \neg C$
- $r_4: B_2 \Rightarrow \neg C$
- $r_5: B_3 \Rightarrow \neg C$

Superiority relation:
- $r_1 > r_3$
- $r_2 > r_4$
- $r_5 > r_1$
Example

Facts: \(A_1, A_2, B_1, B_2\)

Rules:
- \(r_1: A_1 \Rightarrow C\)
- \(r_2: A_2 \Rightarrow C\)
- \(r_3: B_1 \Rightarrow \neg C\)
- \(r_4: B_2 \Rightarrow \neg C\)
- \(r_5: B_3 \Rightarrow \neg C\)

Superiority relation:
- \(r_1 > r_3\)
- \(r_2 > r_4\)
- \(r_5 > r_1\)

Phase 1: Argument for \(C\)
- \(A_1\) (Fact), \(r_1: A_1 \Rightarrow C\)

Phase 2: Possible counterarguments
- \(r_3: B_1 \Rightarrow \neg C\)
- \(r_4: B_2 \Rightarrow \neg C\)
- \(r_5: B_3 \Rightarrow \neg C\)

Phase 3: Rebut the counterarguments
- \(r_1 > r_3\)
- \(r_2 > r_4\)
- \(r_5 > r_1\)
Example

Facts: $A_1, A_2, B_1, B_2$

Rules:
\[
\begin{align*}
  r_1 &: A_1 \Rightarrow C \\
  r_2 &: A_2 \Rightarrow C \\
  r_3 &: B_1 \Rightarrow \neg C \\
  r_4 &: B_2 \Rightarrow \neg C \\
  r_5 &: B_3 \Rightarrow \neg C
\end{align*}
\]

Superiority relation:
\[
\begin{align*}
  r_1 &> r_3 \\
  r_2 &> r_4 \\
  r_5 &> r_1
\end{align*}
\]

Phase 1: Argument for $C$
- $A_1$ (Fact), $r_1 : A_1 \Rightarrow C$

Phase 2: Possible counterarguments
- $r_3 : B_1 \Rightarrow \neg C$
- $r_4 : B_2 \Rightarrow \neg C$
- $r_5 : B_3 \Rightarrow \neg C$

Phase 3: Rebut the counterarguments
- $r_3$ weaker than $r_1$
- $r_4$ weaker than $r_2$
- $r_5$ is not applicable
Extending the language of Defeasible Logic

- extend the language with the deontic operators OBL and PER.
- extend the language with the reparation operator. Permitted only in the head/conclusion of rules.
To prove OBL$\rho_n$ from a rule $A \Rightarrow OBL\rho_1 \otimes \cdots \otimes OBL\rho_{n-1} \otimes OBL\rho_n$, we have to show that $\neg \rho_1, \ldots, \neg \rho_{n-1}$ are provable.

To disprove OBL$\rho_n$ from a rule $A \Rightarrow OBL\rho_1 \otimes \cdots \otimes OBL\rho_{n-1} \otimes OBL\rho_n$, that at least one among $\rho_1, \ldots, \rho_{n_1}$ are rejected.
Business Process Compliance Problem

Given a business process model

- identify what holds in the process
- identify what norms are valid for the process
  - determine what are the obligations, prohibitions, and permissions in force
  - determine when the obligations, prohibitions and permissions are in force in the process (for each trace)
Checking Compliance Recipe

1. Determine the effects of each task and propagate them to successive tasks
1. Determine the effects of each task and propagate them to successive tasks
   - Use the effects to trigger obligations. Run FCL with the effects of a task as input.
Checking Compliance Recipe

1. Determine the effects of each task and propagate them to successive tasks
   - Use the effects to trigger obligations. Run FCL with the effects of a task as input.
   - Check which obligations have been fulfilled, violated
Checking Compliance Recipe

1 Determine the effects of each task and propagate them to successive tasks
   • Use the effects to trigger obligations. Run FCL with the effects of a task as input.
   • Check which obligations have been fulfilled, violated
   • Shake well and serve!
The Journey to Compliance

1. Take or design a business process
2. Annotate the process
   - effects of the tasks (each task is annotated with the effects it produces)
   - rules encoding the norms relevant to the process
Adding Annotations

- A: Enter New Customer Information
- B: Identity Check
- C: Login for Existing Customer
- D: Approve Account Opening
- E: Open Account
- F: Apply Account Policy
- G: Accept initial Deposit
- H: Accept Empty Initial Balance
- I: Initiate Account
- J: Notify Customer and Close Case

<table>
<thead>
<tr>
<th>Task</th>
<th>Semantic Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>newCustomer(x)</td>
</tr>
<tr>
<td>B</td>
<td>checkIdentity(x)</td>
</tr>
<tr>
<td>C</td>
<td>checkIdentity(x), recordIdentity(x)</td>
</tr>
<tr>
<td>E</td>
<td>owner(x, y), account(y)</td>
</tr>
<tr>
<td>F</td>
<td>accountType(y, type)</td>
</tr>
<tr>
<td>G</td>
<td>positiveBalance(y)</td>
</tr>
<tr>
<td>H</td>
<td>¬positiveBalance(y)</td>
</tr>
<tr>
<td>I</td>
<td>accountActive(y)</td>
</tr>
<tr>
<td>J</td>
<td>notify(x, y)</td>
</tr>
</tbody>
</table>
Rules for the Process

- All new customers must be scanned against provided databases for identity checks.

\[ r_1 : newCustomer(x) \Rightarrow O\text{checkIdentity}(x) \]

- Retain history of identity checks performed.

\[ r_2 : checkIdentity(x) \Rightarrow O\text{recordIdentity}(x) \]

- Accounts must maintain a positive balance, unless approved by a bank manager, or for VIP customers.

\[ r_3 : account(x) \Rightarrow O\text{positiveBalance}(x) \otimes O\text{approveManager}(x) \]

\[ r_4 : account(x), accountType(x, VIP) \Rightarrow P\neg \text{positiveBalance}(x) \]
Ideal Semantics: compliance checking

- FCL constraints determine behavioural paths (generic)
  - behavioural paths special case business processes
  - currently expressed as event sequences
- Ideal situation
  - Execution traces do not violate NFCL
- Sub-ideal situation
  - There are violations, but they are repaired/compensated
- Non-ideal (non-compliant) situation
  - There are violations, but they are NOT repaired/compensated
- Irrelevant situation
  - No rule is applicable
Handling Obligations

An obligation chain $OA_1 \otimes \cdots \otimes OA_n$ is **active** given a set of literals $S$, if

- there is a rule $\Gamma \Rightarrow OA_1 \otimes \cdots \otimes OA_n$ such that $\Gamma \subseteq S$, i.e., the rule is triggered by the situation, and

- for all rule for conflicting chains, either
  - the chain is not triggered by the situation or
  - the negation of an element before the conflicting element is not in the situation.
The Algorithm

Input: \textit{Current} set of active obligation chains

\[ A_1 \otimes A_2 = C \in \text{Current} \]

For each \( C \in \text{Current} \)

if \( A_1 = \text{OB} \), then

if \( B \in S \), then

remove([\( T, R, A_1 \otimes A_2 \)], \text{Current}),
remove([\( T, R, A_1 \otimes A_2 \)], \text{Unfulfilled})

if \( [T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2] \in \text{Violated} \) then

add([\( T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2 \)], \text{Compensated})

if \( \neg B \in S \), then

add([\( T, R, A_1 \otimes A_2 \)], \text{Violated}), add([\( T, R, A_2 \)], \text{Current})

else

add([\( T, R, A_1 \otimes A_2 \)], \text{Unfulfilled}).
Finally Compliant!

Definition

- An execution trace is **compliant** iff for all \([T, R, A] \in \text{Current}\),
  \(A = OB \otimes C\), for every \([T, R, A, B] \in \text{Violated}\),
  \([T, R, A, B] \in \text{Compensated}\) and \(\text{Unfulfilled} = \emptyset\).

- An execution trace is **fully compliant** iff for all \([T, R, A] \in \text{Current}\),
  \(A = OB \otimes C\), \(\text{Violated} = \emptyset\) and \(\text{Unfulfilled} = \emptyset\).

- A process is (fully) **compliant** iff all its execution traces are (fully) compliant.
Checking Compliance: Example

Compute the current chains of obligations. Add them to Current

Current
[A, r₁, Ob₁ ⊗ Oc₁]
[A, r₂, Ob₂ ⊗ Oc₂]
[A, r₃, Ob₃ ⊗ Oc₃ ⊗ Od₃]

Violated
[r₁ : a₁ ⇒ Ob₁ ⊗ Oc₁]
[r₂ : a₂ ⇒ Ob₂ ⊗ Oc₂]
[r₃ : a₃ ⇒ Ob₃ ⊗ Oc₃ ⊗ Od₃]
[r₄ : a₄ ⇒ Ob₄ ⊗ Oc₄]
[r₅ : c₂ ⇒ Oc₃]

Unfulfilled

Compensated
Checking Compliance: Example

Examine the chain one by one against the effects.

Current

A - B - C

Violated

r₁ : a₁ ⇒ Ob₁ ⊗ Oc₁
r₂ : a₂ ⇒ Ob₂ ⊗ Oc₂
r₃ : a₃ ⇒ Ob₃ ⊗ Oc₃ ⊗ Od₃
r₄ : a₄ ⇒ Ob₄ ⊗ Oc₄
r₅ : c₂ ⇒ Oc₃

Current

[A, r₁, Ob₁ ⊗ Oc₁]
[A, r₂, Ob₂ ⊗ Oc₂]
[A, r₃, Ob₃ ⊗ Oc₃ ⊗ Od₃]

Unfulfilled

Compensated
Checking Compliance: Example

$\begin{align*}
a_1 & \quad b_1 & \quad c_2 \\
a_2 & \quad -b_2 & \\
a_3 & \\
\end{align*}$

$\begin{align*}
A & \xrightarrow{a_1} B & \xrightarrow{b_1} C \\
A & \xrightarrow{a_2} & \\
A & \xrightarrow{a_3} &
\end{align*}$

$r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1$
$r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2$
$r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3$
$r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4$
$r_5 : c_2 \Rightarrow Oc_3$

$Ob_1 \otimes Oc_1$ and $b_1$. Remove the chain from Current

Current

$[A, r_1, Ob_1 \otimes Oc_1]$
$[A, r_2, Ob_2 \otimes Oc_2]$
$[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$

Unfulfilled

Violated

Compensated
Checking Compliance: Example

\[ \begin{align*}
a_1 & \Rightarrow Ob_1 \otimes Oc_1 \\
a_2 & \Rightarrow Ob_2 \otimes Oc_2 \\
a_3 & \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \\
b_1 & \Rightarrow Ob_4 \otimes Oc_4 \\
c_2 & \Rightarrow Oc_3
\end{align*} \]

Ob\(_1\) \otimes Oc\(_1\) and \(b_1\). Remove the chain from Current

Current

\([A, r_2, Ob_2 \otimes Oc_2] \]

\([A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]

Unfulfilled

Compensated

The Regorous Approach to Business Process Compliance
Guido Governatori

Copyright NICTA 2013
Checking Compliance: Example

\[ r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \]
\[ r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \]
\[ r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \]
\[ r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \]
\[ r_5 : c_2 \Rightarrow Oc_3 \]

Ob_2 \otimes Oc_2 \text{ and } \neg b_2. \text{ Violation}

Current
\[ [A, r_2, Ob_2 \otimes Oc_2] \]
\[ [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]

Unfulfilled

Compensated
Checking Compliance: Example

\[ a_1 \rightarrow b_1 \rightarrow c_2 \]

\[ a_2 \rightarrow \neg b_2 \rightarrow c_3 \]

\[ a_3 \rightarrow \]

\[ r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \]
\[ r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \]
\[ r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \]
\[ r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \]
\[ r_5 : c_2 \Rightarrow Oc_3 \]

Add the chain to Violated, add the compensatory part to Current

Current

\[ [A, r_2, Ob_2 \otimes Oc_2] \]
\[ [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]
\[ [B, r_2, Oc_2] \]

Violated

\[ [B, r_2, Ob_2 \otimes Oc_2] \]

Unfulfilled

Compensated
Checking Compliance: Example

\[ r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \]
\[ r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \]
\[ r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \]
\[ r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \]
\[ r_5 : c_2 \Rightarrow Oc_3 \]

\( Ob_3 \otimes Oc_3 \otimes Od_3 \). No information about \( b_3 \)

Current
\[ [A, r_2, Ob_2 \otimes Oc_2] \]
\[ [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]
\[ [B, r_2, Oc_2] \]

Violated
\[ [B, r_2, Ob_2 \otimes Oc_2] \]

Unfulfilled
Compensated
Checking Compliance: Example

Remove $Ob_3 \otimes Oc_3 \otimes Od_3$ from Current, add it to Unfulfilled

Unfulfilled

Compensated

Current

Violated

Guido Governatori
Checking Compliance: Example

$\begin{align*}
a_1 & \quad b_1 & \quad c_2 \\
a_2 & \quad \neg b_2 & \quad c_3
\end{align*}

A \rightarrow B \rightarrow C \rightarrow \text{Black Circle}

$\begin{align*}
r_1 : a_1 & \Rightarrow Ob_1 \otimes Oc_1 \\
r_2 : a_2 & \Rightarrow Ob_2 \otimes Oc_2 \\
r_3 : a_3 & \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \\
r_4 : a_4 & \Rightarrow Ob_4 \otimes Oc_4 \\
r_5 : c_2 & \Rightarrow Oc_3
\end{align*}$

$Oc_2$. No information about $c_2$

Current
$[A, r_2, Ob_2 \otimes Oc_2]$  
$[B, r_2, Oc_2]$

Violated
$[B, r_2, Ob_2 \otimes Oc_2]$

Unfulfilled
$[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$

Compensated
Checking Compliance: Example

Remove $Oc_2$ from Current, add it to Unfulfilled

Current

$[A, r_2, Ob_2 \otimes Oc_2]$  

Violated

$[B, r_2, Ob_2 \otimes Oc_2]$  

Unfulfilled

$[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$  
$[B, r_2, Oc_2]$  

Compensated

$r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1$  
$r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2$  
$r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3$  
$r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4$  
$r_5 : c_2 \Rightarrow Oc_3$
Checking Compliance: Example

We have done with $B$ and we move to $C$

Current
$[A, r_2, Ob_2 \otimes Oc_2]$

Violated
$[B, r_2, Ob_2 \otimes Oc_2]$

Unfulfilled
$[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$
$[B, r_2, Oc_2]$

Compensated

$r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1$
$r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2$
$r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3$
$r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4$
$r_5 : c_2 \Rightarrow Oc_3$
Checking Compliance: Example

We move the elements of Unfulfilled to Current
Current  [A, r_2, Ob_2 \otimes Oc_2]
Violated   [B, r_2, Ob_2 \otimes Oc_2]

Unfulfilled  [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]
             [B, r_2, Oc_2]
Compensated

r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1
r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2
r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3
r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4
r_5 : c_2 \Rightarrow Oc_3
Checking Compliance: Example

We move the elements of Unfulfilled to Current

Current

[A, r2, Ob2 ⊗ Oc2]
[A, r3, Ob3 ⊗ Oc3 ⊗ Od3]
[B, r2, Oc2]

We move the elements of Unfulfilled to Current

Violated

[B, r2, Ob2 ⊗ Oc2]

Unfulfilled

Compensated

r1 : a1 ⇒ Ob1 ⊗ Oc1
r2 : a2 ⇒ Ob2 ⊗ Oc2
r3 : a3 ⇒ Ob3 ⊗ Oc3 ⊗ Od3
r4 : a4 ⇒ Ob4 ⊗ Oc4
r5 : c2 ⇒ Oc3
Checking Compliance: Example

We compute the set of chains for $C$

Current

- $[A, r_2, Ob_2 \otimes Oc_2]$
- $[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$
- $[B, r_2, Oc_2]$
- $[C, r_5, Oc_3]$

Unfulfilled

Violated

- $[B, r_2, Ob_2 \otimes Oc_2]$

Compensated
Checking Compliance: Example

$\begin{align*}
& \text{A} \quad \text{B} \quad \text{C} \\
& a_1 \quad b_1 \quad c_2 \\
& a_2 \quad \neg b_2 \\
& a_3 \quad c_3
\end{align*}$

$\begin{align*}
& r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \\
& r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \\
& r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \\
& r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \\
& r_5 : c_2 \Rightarrow Oc_3
\end{align*}$

$Oc_2$ and $c_2$, remove the chain from Current

Current

[$[A, r_2, Ob_2 \otimes Oc_2]$]

[$[A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]$]

[$[C, r_5, Oc_3]$]

Violated

[$[B, r_2, Ob_2 \otimes Oc_2]$]

Unfulfilled

Compensated
Checking Compliance: Example

\[ A \rightarrow B \rightarrow C \rightarrow \circ \]

- \( a_1 \)
- \( a_2 \)
- \( a_3 \)
- \( b_1 \)
- \( b_2 \)
- \( c_2 \)
- \( c_3 \)

\( r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \)
\( r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \)
\( r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \)
\( r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \)
\( r_5 : c_2 \Rightarrow Oc_3 \)

\( B, r_2, Ob_2 \otimes Oc_2 \) in Violated and \( c_2 \), add to Compensated

Current
- \([A, r_2, Ob_2 \otimes Oc_2]\)
- \([A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]\)
- \([C, r_5, Oc_3]\)

Unfulfilled

Violated
- \([B, r_2, Ob_2 \otimes Oc_2]\)

Compensated
- \([B, r_2, Ob_2 \otimes Oc_2]\)
Checking Compliance: Example

\[ a_1 \quad b_1 \quad c_2 \quad c_3 \]

\[ a_1 \Rightarrow Ob_1 \otimes Oc_1 \]
\[ a_2 \Rightarrow Ob_2 \otimes Oc_2 \]
\[ a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \]
\[ a_4 \Rightarrow Ob_4 \otimes Oc_4 \]
\[ c_2 \Rightarrow Oc_3 \]

\( Oc_3 \) and \( c_3 \), remove the chain from Current

Current
\[ [A, r_2, Ob_2 \otimes Oc_2] \]
\[ [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]
\[ [C, r_5, Oc_3] \]

Unfulfilled

Violated
\[ [B, r_2, Ob_2 \otimes Oc_2] \]

Compensated
\[ [B, r_2, Ob_2 \otimes Oc_2] \]
Checking Compliance: Example

\[ r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \]
\[ r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \]
\[ r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \]
\[ r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \]
\[ r_5 : c_2 \Rightarrow Oc_3 \]

Still no information about \( b_3 \). The chain to Unfulfilled

Current
\[ [A, r_2, Ob_2 \otimes Oc_2] \]
\[ [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \]

Violated
\[ [B, r_2, Ob_2 \otimes Oc_2] \]

Unfulfilled

Compensated
\[ [B, r_2, Ob_2 \otimes Oc_2] \]
http://www.regorous.com
Evaluation of Regorous

Formalised Chapter 8 (Complaints) of TCPC 2012. Modelled the compliant handling/management processes of an Australian telco

process removed.

41 tasks, 12 decision points (xor), 2 loops
shortest trace: 6 traces longest trace (loop): 33 tasks
longest trace (no loop): 22 tasks
over 1000 traces, over 25000 states
TCPC 2012 Chapter 8. Contains over 100 commas, plus 120 terms (in Terms and Definition Section).
Required 223 propositions, 176 rules.

<table>
<thead>
<tr>
<th>Obligation Type</th>
<th>Count</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Punctual Obligation</td>
<td>5</td>
<td>(5)</td>
</tr>
<tr>
<td>Achievement Obligation</td>
<td>90</td>
<td>(110)</td>
</tr>
<tr>
<td>Preemptive</td>
<td>41</td>
<td>(46)</td>
</tr>
<tr>
<td>Non preemptive</td>
<td>49</td>
<td>(64)</td>
</tr>
<tr>
<td>Non perdurant</td>
<td>5</td>
<td>(7)</td>
</tr>
<tr>
<td>Maintenance Obligation</td>
<td>11</td>
<td>(13)</td>
</tr>
<tr>
<td>Prohibition</td>
<td>7</td>
<td>(9)</td>
</tr>
<tr>
<td>Non perdurant</td>
<td>1</td>
<td>(4)</td>
</tr>
<tr>
<td>Permission</td>
<td>9</td>
<td>(16)</td>
</tr>
<tr>
<td>Compensation</td>
<td>2</td>
<td>(2)</td>
</tr>
</tbody>
</table>
Guido Governatori.
Representing business contracts in RuleML.

Guido Governatori.

Guido Governatori and Antonino Rotolo.
A conceptually rich model of business process compliance.

Guido Governatori and Antonino Rotolo.
Norm Compliance in Business Process Modeling.

Guido Governatori and Shazia Sadiq.
The journey to business process compliance.
Guido Governatori and Shazia ÂोShek.
Rule based business process compliance.
In Proc. of RuleML2012@ECAI Challenge, CEUR Workshop Proceedings 874, article 5, 2012.

Ruopeng Lu, Shazia Sadiq, and Guido Governatori.
Measurement of compliance distance in business processes.

Shazia Sadiq and Guido Governatori.
Managing Regulatory Compliance in Business Processes.

Shazia Sadiq, Guido Governatori, and Kioumars Naimiri.
Modelling of control objectives for business process compliance.