Formal Models of Business Process Compliance

Guido Governatori
Marta Indulska
Michael zur Muehlen

JURIX, Rotterdam, 16 December 2009
Overview

- What is compliance
- Deontic concepts
- Logic of violations
- Defeasible Deontic Logic of Violations
- BPM Compliance
Overview

- What is compliance
- Deontic concepts
- Logic of violations
- Defeasible Deontic Logic of Violations
- BPM Compliance
  - Business Process Models
Overview

- What is compliance
- Deontic concepts
- Logic of violations
- Defeasible Deontic Logic of Violations
- BPM Compliance
  - Business Process Models
  - Annotations in execution semantics
Overview

- What is compliance
- Deontic concepts
- Logic of violations
- Defeasible Deontic Logic of Violations
- BPM Compliance
  - Business Process Models
  - Annotations in execution semantics
  - Norms regulating the process
Overview

- What is compliance
- Deontic concepts
- Logic of violations
- Defeasible Deontic Logic of Violations
- BPM Compliance
  - Business Process Models
  - Annotations in execution semantics
  - Norms regulating the process
  - Checking compliance
Ensuring that business operations, processes, and practices are in accordance with a given prescriptive (often legal) document
What’s compliance?

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

**Regulatory**
- Basel II
- Sarbanes-Oxley
- OFAC (USA Patriot Act)
- OSFI “blocked entity” lists
- HIPAA
- Graham-Leach-Bliley

**Standards**
- Best practice models
- SAP solution maps
- ISO 9000
- Medical guidelines

**Contracts**
- Service Agreement
- Customer Contract
- Warranty
- Insurance Policy
- Business Partnership
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
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.
Ensuring that business processes are in accordance with given prescriptive rules

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive rules
Ensuring that business processes are in accordance with given prescriptive rules

Compliance is a relationship between two sets of specifications

Alignment of formal specifications for business processes and formal specifications for prescriptive rules

Ensuring that business processes are compliant requires a suitable language for expressing normative specifications in such a way as

\[\text{we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and we can make hidden conditions explicit.}\]
Ensuring Compliance

- Ensuring that business processes are in accordance with given prescriptive rules
- Compliance is a relationship between two sets of specifications
- Alignment of formal specifications for business processes and formal specifications for prescriptive rules
- Ensuring that business processes are compliant requires a suitable language for expressing normative specifications in such a way as
  - we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and
Ensuring Compliance

- Ensuring that business processes are in accordance with given prescriptive rules
- Compliance is a relationship between two sets of specifications
- Alignment of formal specifications for business processes and formal specifications for prescriptive rules
- Ensuring that business processes are compliant requires a suitable language for expressing normative specifications in such a way as
  - we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and
  - we can make hidden conditions explicit
Ensuring Compliance

- Ensuring that business processes are in accordance with given prescriptive rules
- Compliance is a relationship between two sets of specifications
- Alignment of formal specifications for business processes and formal specifications for prescriptive rules
- Ensuring that business processes are compliant requires a suitable language for expressing normative specifications in such a way as
  - we can identify formal loopholes, deadlocks and inconsistencies in normative systems, and
  - we can make hidden conditions explicit

Without this, we do not have any guarantee that a given business process is compliant, because we do not know if all relevant norms have been considered.
Internal (Governance) or External (Regulations/Standards/Contracts) generate compliance requirements.
Internal (Governance) or External (Regulations/Standards/Contracts) generate compliance requirements

Consultants define Control Objectives to fulfill compliance obligations
Compliance Pipeline

- Internal (Governance) or External (Regulations/Standards/Contracts) generate compliance requirements
- Consultants define Control Objectives to fulfill compliance obligations
- Control Objectives analysed in terms of risk and necessary internal controls required for “effective and efficient” implementation of control objectives
Compliance Pipeline

- Internal (Governance) or External (Regulations/Standards/Contracts) generate compliance requirements

- Consultants define *Control Objectives* to fulfill compliance obligations

- Control Objectives analysed in terms of *risk* and necessary *internal controls* required for “effective and efficient” implementation of control objectives
  - Proposed internal controls are built into business processes / transactions (*Compliance by design*)
Compliance Pipeline

- Internal (Governance) or External (Regulations/Standards/Contracts) generate compliance requirements
- Consultants define *Control Objectives* to fulfill compliance obligations
- Control Objectives analysed in terms of *risk* and necessary *internal controls* required for “effective and efficient” implementation of control objectives
  - Proposed internal controls are built into business processes / transactions (*Compliance by design*)
  - Operational practices are monitored for deviation from prescribed internal controls (*Detection of non-compliance*)
Compliance by design methodology

Controls Directory Management
Ontological Alignment mapping between the languages, common ontology.

Modeling Control Objectives Formal language based on deontic logic to represent and analyze normative positions stemming from compliance requirements

**Example**

The creation and approval of purchase requests must be undertaken by two separate purchase officers

\[ c1 : CreatePR(x, y), PurchaseOfficer(y) \Rightarrow O\neg ApprovedPR(x, y) \]

Process Model Enrichment attach semantic annotations to activities in a process model (Flow Tags, Data Tags, Resource Tags, Time Tags)

Event Monitoring Operational practices are monitored for deviation from prescribed internal controls
Key components of Normative Systems

A normative system is a set of clauses

- Definitional clauses (counts-as rules)
- Prescriptive clauses (norms)
  - obligations
  - permissions
  - prohibitions
  - violations
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.
Formalising Compliance

- Clausal forms (Sergot-Kowalski, Ghose-Koliadis, Hoffman-Weber-Governatori)
Formalising Compliance

- Clausal forms (Sergot-Kowalski, Ghose-Koliadis, Hoffman-Weber-Governatori)
- Temporal logic (van der Aalst, Rinderle-Ma)
Formalising Compliance

- Clausal forms (Sergot-Kowalski, Ghose-Koliadis, Hoffman-Weber-Governatori)
- Temporal logic (van der Aalst, Rinderle-Ma)
- Event Calculus (Goedertier, Singh, Farrell-Sergot)
Formalising Compliance

- Clausal forms (Sergot-Kowalski, Ghose-Koliadis, Hoffman-Weber-Governatori)
- Temporal logic (van der Aalst, Rinderle-Ma)
- Event Calculus (Goedertier, Singh, Farrell-Sergot)
- Deontic Logic (Governatori-Milosevic-Sadiq, Goedertier, Liu)
Clausal Approaches

\[ p \lor (q \land r) \lor s \]

- do not distinguish different normative positions
Clausal Approaches

\[ p \lor (q \land r) \lor s \]

- do not distinguish different normative positions
- directionality of norms (what are the premises and what is the conclusion)
Clausal Approaches

\[ p \lor (q \land r) \lor s \]

- do not distinguish different normative positions
- directionality of norms (what are the premises and what is the conclusion)
- semantic compliance (annotations)
Temporal Logic

$p \rightarrow A\Box q$

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
Temporal Logic

\[ p \rightarrow A\Box q \]

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
- Mature technology
$p \rightarrow A\Box q$

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
- Mature technology
- Structural Compliance only
$p \rightarrow A\Box q$

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
- Mature technology
- Structural Compliance only
- Does not distinguish normative positions
Temporal Logic

$p \rightarrow A□q$

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
- Mature technology
- Structural Compliance only
- Does not distinguish normative positions
- Standard Deontic Logic can be simulated in Temporal Logic
Temporal Logic

$p \rightarrow A \square q$

and model checking

- Temporal logic and model checking have been used to verification of software and hardware system
- Mature technology
- Structural Compliance only
- Does not distinguish normative positions
- Standard Deontic Logic can be simulated in Temporal Logic
- Permissions must always be instantiated
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.

- $OBL\alpha \equiv \neg PER\neg \alpha, \quad PER\alpha \equiv \neg OBL\neg \alpha$
- $OBL(\alpha \rightarrow \beta) \rightarrow (OBL\alpha \rightarrow OBL\beta)$
- $OBL\alpha \rightarrow PER\alpha$ or $(OBL\alpha \rightarrow \neg OBL\neg \alpha)$

Standard Deontic Logic is not able to deal with violations
### Rules for JURIX Tutorials

- 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
Violation paradox

Rules for JURIX Tutorials

- 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 $\neg lie$
- $lie \rightarrow OBL\neg explain$
- OBL($\neg lie \rightarrow \neg explain$)
- $lie$
Violation paradox

Rules for JURIX Tutorials

- 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

\[ \text{OBL} \neg \text{lie} \]
\[ \text{lie} \rightarrow \text{OBL explain} \]
\[ \neg \text{lie} \rightarrow \neg \text{explain} \]
\[ \text{lie} \]

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

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

\[ a \Rightarrow Ob \]
\[ \neg b \Rightarrow Oc \]
\[ \neg c \Rightarrow Od \]
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 $A_1, \ldots, A_n \vdash XB$. 
A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash XB$.

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

A violation does not exist without an obligation it violates.

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

A reparation can be an obligation itself, and thus it can be violated.
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.
A (normative) prescriptive clause is represented by a rule $A_1, \ldots, A_n \vdash X B$.

A violation does not exist without an obligation it violates.

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

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

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.
Making it explicit

Introducing the reparation operator $\otimes$

$\text{PrimeMinister} \Rightarrow O \neg \text{lie} \otimes O \text{explain} \otimes O \text{impeachment}$

Merging rules to obtain new rules

Removing redundancies

Detecting and solving conflicts
Introducing the reparation operator \(\otimes\)

\[\text{PrimeMinister} \Rightarrow O\neg \text{lie} \otimes O\text{explain} \otimes O\text{impeachment}\]
Making it explicit

- Introducing the reparation operator $\otimes$

$$PrimeMinister \Rightarrow O\neg\text{lie} \otimes O\text{explain} \otimes O\text{impeachment}$$

- Merging rules to obtain new rules
Making it explicit

- Introducing the reparation operator $\otimes$

  $$PrimeMinister \Rightarrow O\neg\text{lie} \otimes O\text{explain} \otimes O\text{impeachment}$$

- Merging rules to obtain new rules
- Removing redundancies
Making it explicit

- Introducing the reparation operator $\otimes$

$PrimeMinister \Rightarrow \neg\text{lie} \otimes \neg\text{explain} \otimes \text{Impeachment}$

- Merging rules to obtain new rules
- Removing redundancies
- Detecting and solving conflicts
Merging rules

\[
\frac{\Gamma \vdash \text{OBL}A \quad \Delta, \neg A \vdash \mathbf{X}B}{\Gamma, \Delta \vdash \text{OBL}A \otimes \mathbf{X}B}
\]

Example

From JURIX \(\vdash \text{OBL}\) guidon \neg lie

Marta Granted Lie

we obtain

JURIX, Martian Granted Lie \(\vdash \text{OBL}\) guidon \neg lie \(\otimes \) OBL guidon explain

\(\copyright\) NICTA 2009
Merging rules

\[ \Gamma \vdash \text{OBL}A \quad \Delta, \neg A \vdash \text{XB} \]
\[ \Gamma, \Delta \vdash \text{OBL}A \otimes \text{XB} \]

Example

From

\[ JURIX \vdash \text{OBL}_{\text{guido}} \neg \text{lie} \]
\[ \text{MartaGrantedLie}, \text{lie} \vdash \text{OBL} \text{explain} \]

we obtain

\[ JURIX, \text{MartaGrantedLie} \vdash \text{OBL}_{\text{guido}} \neg \text{lie} \otimes \text{OBL}_{\text{guido}} \text{explain} \]
Example 1

\[ r_1 : JURIX \vdash \text{OBL} \neg lie \]
\[ r_2 : JURIX \vdash \text{OBL} \neg lie \otimes \text{OBLexplain} \]
# Removing redundancies

## Example 1

\[ r_1 : JURIX \vdash OBL \neg lie \]
\[ r_2 : JURIX \vdash OBL \neg lie \otimes OBL \text{explain} \]

## Example 2

\[ r_1 : JURIX, lie \vdash OBL \text{explain} \]
\[ r_2 : JURIX \vdash OBL \neg lie \otimes OBL \text{explain} \]
Removing redundancies

Example 1

\[ r_1 : JURIX \vdash OBL \neg \text{lie} \]
\[ r_2 : JURIX \vdash OBL \neg \text{lie} \otimes \text{OBLexplain} \]

Example 2

\[ r_1 : JURIX, \\text{lie} \vdash \text{OBLexplain} \]
\[ r_2 : JURIX \vdash OBL \neg \text{lie} \otimes \text{OBLexplain} \]

The normative content of \( r_1 \) is included in \( r_2 \), and thus \( r_1 \) can be removed.
Detecting conflicts

\[ \Gamma \vdash A \quad \Delta \vdash \neg A \]
\[ \Gamma, \Delta \vdash false \]

if \( \Gamma \) and \( \Delta \) are mutually consistent and \( A \) and \( \neg A \) cannot be repaired
Detecting conflicts

\[ \frac{\Gamma \vdash A \quad \Delta \vdash \neg A}{\Gamma, \Delta \vdash \text{false}} \]

if \( \Gamma \) and \( \Delta \) are mutually consistent and \( A \) and \( \neg A \) cannot be repaired

**Example**

From

\[ JURIX \vdash OBL\text{lie} \]
\[ LongSeminar \vdash OBL\neg\text{lie} \]

we obtain

\[ JURIX, LongSeminar \vdash \text{false} \]
Normal FCL Forms

- A ‘cleaned-up’ version of the FCL specifications
- Related clauses are merged, in particular linking original clauses and reparation clauses
- Removing redundancies, in particular clauses that are subsumed by other clauses
- Detecting and resolving conflicts
- NFCL form is used for compliance checking. NFCL forms describe behavioural and state space of contract

---

Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

\[ PhD \rightarrow Uni \]
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

\[ \text{PhD} \rightarrow \text{Uni} \]
\[ \text{Weekend} \rightarrow \neg \text{Uni} \]
\[ \text{PublicHoliday} \rightarrow \neg \text{Uni} \]
\[ \text{Sick} \rightarrow \neg \text{Uni} \]
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

\[ PhD \rightarrow Uni \]
\[ Weekend \rightarrow \neg Uni \]
\[ PublicHoliday \rightarrow \neg Uni \]
\[ Sick \rightarrow \neg Uni \]
\[ Weekend \land VIC\text{deadline} \rightarrow Uni \]
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

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

\( \text{VIC} = \text{Very Important Conference} \) (e.g., Jurix, ICAIL, . . .)
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

\( PhD \rightarrow Uni \)

\( Weekend \rightarrow \neg Uni \)

\( PublicHoliday \rightarrow \neg Uni \)

\( Sick \rightarrow \neg Uni \)

\( Weekend \land VIC\text{deadline} \rightarrow Uni \)

\( VIC\text{deadline} \land PartnerBirthday \rightarrow \neg Uni \)

VIC= Very Important Conference (e.g., Jurix, ICAIL, . . .)
Why Non-monotonic Reasoning

Reasonable Results with Minimum Effort

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

\[ PhD \land (\neg Weekend \lor (Weekend \land VIC\text{deadline} \land \neg PartnerBirthday)) \land \neg Sick \ldots \rightarrow Uni \]

VIC = Very Important Conference (e.g., Jurix, ICAIL, \ldots)
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
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
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
1. Give an argument for the conclusion you want to prove
2. Consider all possible counterarguments to it
3. Rebut all counterarguments
1. Give an argument for the conclusion you want to prove
2. Consider all possible counterarguments to it
3. 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**
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), $r_1: A_1 \Rightarrow C$
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

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

Facts: $A_1$, $A_2$, $B_1$, $B_2$

Rules:
1. $r_1: A_1 \Rightarrow C$
2. $r_2: A_2 \Rightarrow C$
3. $r_3: B_1 \Rightarrow \neg C$
4. $r_4: B_2 \Rightarrow \neg C$
5. $r_5: B_3 \Rightarrow \neg C$

Phase 1: Argument for $C$

1. $A_1$ (Fact), $r_1: A_1 \Rightarrow C$

Phase 2: Possible counterarguments

1. $r_3: B_1 \Rightarrow \neg C$
2. $r_4: B_2 \Rightarrow \neg C$
3. $r_5: B_3 \Rightarrow \neg C$

Superiority relation:

1. $r_1 > r_3$
2. $r_2 > r_4$
3. $r_5 > r_1$
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 $C$
$A_1 \text{ (Fact), } r_1: A_1 \Rightarrow C$

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
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
        $r_3$ weaker than $r_1$
        $r_4$ weaker than $r_2$
        $r_5$ is not applicable
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 $\text{OBL}p_n$ from a rule 
$A \Rightarrow \text{OBL}p_1 \otimes \cdots \otimes \text{OBL}p_{n-1} \otimes \text{OBL}p_n$, we have to show that 
$\neg p_1, \ldots, \neg p_{n-1}$ are provable.

To disprove $\text{OBL}p_n$ from a rule 
$A \Rightarrow \text{OBL}p_1 \otimes \cdots \otimes \text{OBL}p_{n-1} \otimes \text{OBL}p_n$, that at least one 
among $p_1, \ldots, p_{n_1}$ are rejected.

For more details see

Guido Governatori, 2005. Representing Business Contracts in 
RuleML. *International Journal of Cooperative Information Systems*, 
Part I

BPM Compliance
A Business Process Model (BPM) describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives. A language for BPM usually has two elements: tasks are activities to be performed and 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).
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.
- A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.
- A language for BPM usually has two elements:
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.

A language for BPM usually has two elements:
- Tasks are activities to be performed
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.

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).
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.

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).
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives

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),
A BPM describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives.

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 task must be executed).
Example: Account Opening Process

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

© NICTA 2009
Execution Traces

A, B, C, D, E, F, H
A, D, B, C, E, G, H
A, D, B, C, E, F, H
Business Process Compliance

Recommendations

What-if analysis

Status report

Compliance checker

Obligations

Input

Annotated process model

Compliance rule base & checker

Logical state representation

Rule1
Rule2
Rule3
Rule4
Rule5
Rule6
Rule7
Rule8
Rule9
...

I*(e1)
I*(e2)
I*(e3)
I*(e4)

Recommendation sub-system

Legalese

Formalisation

Recommendations

T1
Post1

T2
Post2

T3
Post3

T4
Post4

T5
Post5

T6
Post6

T7
Post7

© NICTA 2009
Checking Compliance Recipe

1. Determine the effects of each task and propagate them to successive tasks.

Shake well and serve!
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.
Determining the effects of each task and propagating 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!
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

### Task Semantic Annotation

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

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

Retain history of identity checks performed.

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

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

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

\[ r_4 : \text{account}(x), \text{accountType}(x, \text{VIP}) \Rightarrow P\neg \text{positiveBalance}(x) \]
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.
**Input**: Current set of active obligation chains

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

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

if \( A_1 = OB \), 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 \([T, R, A] \in \text{Current}, \ A = OB \odot C\), 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 \([T, R, A] \in \text{Current}, \ A = OB \odot C\), \(\text{Violated} = \emptyset\) 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
r₄ : c₂ ⇒ Oc₃

Unfulfilled

Compensated
Examine the chain one by one against the effects.

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

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

Violated

Compensated
Checking Compliance: Example

\[ a_1 \rightarrow b_1 \rightarrow c_2 \]

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

\( Ob_1 \otimes Oc_1 \) 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

\( NICTA \ 2009 \)
Checking Compliance: Example

$Ob_1 \otimes Oc_1$ 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

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

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

\( Ob_2 \otimes Oc_2 \) and \( \neg b_2 \). 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

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

Current

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

Violated

[B, r2, Ob2 ⊗ Oc2]

Unfulfilled

Compensated

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

$O b_3 \otimes O c_3 \otimes O d_3$. No information about $b_3$

Current

$[A, r_2, O b_2 \otimes O c_2]$
$[A, r_3, O b_3 \otimes O c_3 \otimes O d_3]$
$[B, r_2, O c_2]$

Violated

$[B, r_2, O b_2 \otimes O c_2]$

Unfulfilled

Compensated

$r_1 : a_1 \Rightarrow O b_1 \otimes O c_1$
$r_2 : a_2 \Rightarrow O b_2 \otimes O c_2$
$r_3 : a_3 \Rightarrow O b_3 \otimes O c_3 \otimes O d_3$
$r_4 : a_4 \Rightarrow O b_4 \otimes O c_4$
$r_4 : c_2 \Rightarrow O c_3$
Checking Compliance: Example

Remove $Ob_3 \otimes Oc_3 \otimes Od_3$ from Current

Current

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

Unfulfilled

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

Compensated

$B, r_2, Ob_2 \otimes Oc_2$

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

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

\( Oc_2 \). No information about \( c_2 \)

Current
\[ \begin{align*}
    [A, r_2, Ob_2 \otimes Oc_2] \\
    [B, r_2, Oc_2]
\end{align*} \]

Violated
\[ \begin{align*}
    [B, r_2, Ob_2 \otimes Oc_2]
\end{align*} \]

Unfulfilled
\[ \begin{align*}
    [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3]
\end{align*} \]

Compensated
Checking Compliance: Example

\[ a_1, a_2, a_3, b_1, \neg b_2, c_2, c_3 \]

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

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
Checking Compliance: Example

\[
\begin{align*}
&\quad a_1
\quad a_2
\quad a_3 \\
\rightarrow &\quad b_1 \\
\rightarrow &\quad \neg b_2 \\
\rightarrow &\quad c_2 \\
\rightarrow &\quad c_3
\end{align*}
\]

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

\[
\begin{align*}
&\quad r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \\
&\quad r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \\
&\quad r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \\
&\quad r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \\
&\quad r_4 : c_2 \Rightarrow Oc_3
\end{align*}
\]
Checking Compliance: Example

We move the elements of Unfulfilled to Current
Current  
[A, r₂, Ob₂ ⊗ Oc₂]

Violated  
[B, r₂, Ob₂ ⊗ Oc₂]

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

Compensated  

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

We move the elements of Unfulfilled to Current

Current

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

Unfulfilled

Compensated

Violated

[B, r₂, Ob₂ ⊗ Oc₂]
Checking Compliance: Example

We compute the set of chains for \( C \)

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

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

**Unfulfilled**

**Compensated**

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

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

Current

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

Unfulfilled

Violated

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

Compensated

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

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

\[ a_2 \quad -b_2 \quad \]

\[ a_3 \]

\[ A \rightarrow B \rightarrow C \rightarrow \text{Circle} \]

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

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

\[ \begin{align*}
\text{Current} & : \quad [A, r_2, Ob_2 \otimes Oc_2] \\
& \quad [A, r_3, Ob_3 \otimes Oc_3 \otimes Od_3] \\
& \quad [C, r_5, Oc_3] \\
\text{Unfulfilled} & : \\
\text{Violated} & : \quad [B, r_2, Ob_2 \otimes Oc_2] \\
\text{Compensated} & : \quad [B, r_2, Ob_2 \otimes Oc_2] \\
\end{align*} \]

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

\[ \begin{align*}
& \quad r_1 : a_1 \Rightarrow Ob_1 \otimes Oc_1 \\
& \quad r_2 : a_2 \Rightarrow Ob_2 \otimes Oc_2 \\
& \quad r_3 : a_3 \Rightarrow Ob_3 \otimes Oc_3 \otimes Od_3 \\
& \quad r_4 : a_4 \Rightarrow Ob_4 \otimes Oc_4 \\
& \quad r_4 : c_2 \Rightarrow Oc_3
\end{align*} \]
Checking Compliance: Example

\[ O b_3 \otimes O c_3 \otimes O d_3. \text{ No information about } b_3 \text{.} \]

Current

\[ [A, r_2, O b_2 \otimes O c_2] \]
\[ [A, r_3, O b_3 \otimes O c_3 \otimes O d_3] \]

Unfulfilled

Violated

\[ [B, r_2, O b_2 \otimes O c_2] \]

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

Compensated

\[ [B, r_2, O b_2 \otimes O c_2] \]
Ontology of Norms

Persistent vs immediate obligations

- An *immediate* obligation must be satisfied as soon as it occurs. ‘When banks proceed with any wire transfer, they must transmit a message, via SWIFT, to the receiving bank requesting that the payment is made according to the instructions given.’

- 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.’

Immediate obligations can be used to check the ‘structural compliance of a BP’
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.
Preemptive or Non-preemptive Obligations

- Only for achievement obligations
- **Preemptive obligations**: the fulfillment of an obligation can happen before the obligation has been triggered.
- **Non preemptive obligations**: the fulfilment 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 X of an order admitting a will to probate has been signed.’
Structural Compliance: sequence

$t_1 \Rightarrow O^p t_2$
Structural Compliance: AND-Split

\[ t \Rightarrow O^p t_1 \]

\[ \vdots \]

\[ t \Rightarrow O^p t_n \]
Structural Compliance: AND-Join

$t_1, \ldots, t_n \Rightarrow O^P t$
Structural Compliance: OR-Split

\[ t \Rightarrow O^P(\text{ORsplitID}) \]

\[ t_1 \Rightarrow \text{ORsplitID} \]

\[ \vdots \]

\[ t_n \Rightarrow \text{ORsplitID} \]

for an XOR, in addition we need

\[ t_i \Rightarrow \neg t_j \quad i \neq j, 1 \leq i, j \leq n \]
Structural Compliance: OR-Joins

\[ t_1 \Rightarrow O^p t \]

\[ \vdots \]

\[ t_n \Rightarrow O^p t \]
Modifications of the basic algorithm

\( C^3 \) seems to be computationally hard (the number of execution paths is at least exponential); is it possible to identify computationally feasible cases

- punctual obligations: easy
- achievement obligations without-sanction: easy
- preemptive achievement obligations: easy
- non-preemptive obligations: hard (interleaving of parallel subprocess, EXPTIME-hard)
- maintenance obligations: hard (‘synchronisation of sequential tasks’ (EXPTIME-hard) + interleaving and synchronisation of parallel processes).
Non-preemptive Obligations

\[ O^{a, np} A \otimes OB \]

Diagram:

- \( T_1 : \neg A \)
- \( T_2 : B \)
Non-preemptive Obligations

\[ O^{a, np} A \otimes OB \]

\[ T_1 : \neg A \]

\[ T_2 : B \]

\[ T_1; T_2 \] Compliant
Non-preemptive Obligations

\[ O^{a, np} A \otimes OB \]

\[ T_1 : \neg A \]

\[ T_2 : B \]

\[ T_1 ; T_2 \quad \text{Compliant} \]

\[ T_2 ; T_1 \quad \text{Non-compliant} \]
Maintenance Obligations

\[ O^m_{\text{red}} \]
$O^m_{\text{red}}$
Summary and Past Work

- Presented a framework to check the compliance of a business process wrt a set of norms;
- The proposed framework is computationally feasible (all the operations are in polynomial time);
- checking compliance is computationally hard.
- Extend the framework with time and deadlines and with a rich ontology of obligations: persistent vs achievement obligations.
References

Nirmit Desai, Nanjangud C. Narendra, Munindar P. Singh.
Checking correctness of business contracts via commitments.

Using the event calculus for tracking the normative state of contracts.

Aditya Ghose, George Koliadis.
Auditing Business Process Compliance.

Stijn Goedertier and Jan Vanthienen.
Designing compliant business processes with obligations and permissions.

Guido Governatori.
Representing business contracts in RuleML.

Guido Governatori, Jörg Hoffmann, Shazia Sadiq, and Ingo Weber.
Detecting regulatory compliance for business process models through semantic annotations.
Guido Governatori and Zoran Milosevic. 
Dealing with contract violations: formalism and domain specific language. 

Guido Governatori, Zoran Milosevic, and Shazia Sadiq. 
Compliance checking between business processes and business contracts. 

Guido Governatori and Antonino Rotolo. 
Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. 

Guido Governatori and Antonino Rotolo. 
An algorithm for business process compliance. 

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.

Ying Liu, Samuel Müller, Ke Xu.
A static compliance-checking framework for business process models.

Ruopeng Lu, Shazia Sadiq, and Guido Governatori.
Compliance aware business process design.

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

Stefanie Rinderle-Ma, Linh Thao Ly, Peter Dadam.
Business Process Compliance.
Anne Rozinat, Wil M. P. van der Aalst.  
Conformance checking of processes based on monitoring real behavior.  

Shazia Sadiq and Guido Governatori.  
A methodological framework for aligning business processes and regulatory compliance.  

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

Ingo Weber, Guido Governatori, and Jörg Hoffmann.  
Approximate compliance checking for annotated process models.  