Towards the Specification of Natural Language Accountability Policies with AccLab: The Laptop Policy Use Case

Walid Benghabrit
Jean-Claude Royer
Anderson Santana De Oliveira

ASCOLA Group, IMT Atlantique - INRIA Rennes
firstname.lastname@imt-atlantique.fr
and
SAP Research Lab, Sophia Antipolis
Anderson.Santana.De.Oliveira@sap.com

Nice, CARe-MAS Workshop

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language
- Our tool: AccLab, a laboratory for accountability

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language
- Our tool: AccLab, a laboratory for accountability
- The policy: the shortest of seven policies from the Hope University in Liverpool

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language
- Our tool: AccLab, a laboratory for accountability
- The policy: the shortest of seven policies from the Hope University in Liverpool
- On going work ...

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language
- Our tool: AccLab, a laboratory for accountability
- The policy: the shortest of seven policies from the Hope University in Liverpool
- On going work ...
  - Laptop in a final state, but other specifications not yet complete

Objectives

- Experiment with our language and tool support to evaluate their suitability in formalizing some real policies
- Excluded: not possible to specify all situations (human interaction, judgment, ambiguity, missing information, ...)
- Our language: Abstract Accountability Language
- Our tool: AccLab, a laboratory for accountability
- The policy: the shortest of seven policies from the Hope University in Liverpool
- On going work ...
  - Laptop in a final state, but other specifications not yet complete
  - Tool support still under development

Accountability

- Accountability at design time: specification and verification

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification
- Usage (UE): Authorizations, obligations, behaviour, ...

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification
- Usage (UE): Authorizations, obligations, behaviour, ...
- Audit (audit): Who, when, what

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification
- Usage (UE): Authorizations, obligations, behaviour, ...
- Audit (audit): Who, when, what
- Rectification (RE): Remediation, sanction, compensation, explanations, ...

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification
- Usage (UE): Authorizations, obligations, behaviour, ...
- Audit (audit): Who, when, what
- Rectification (RE): Remediation, sanction, compensation, explanations, ...
- The simplest form is `(NOT UE) => (audit => RE)`

Accountability

- Accountability at design time: specification and verification
- It involved agents with dynamic behaviour and dynamic policies
- Focus on expressiveness and static verification
- Accountability = usage + audit + rectification
- Usage (UE): Authorizations, obligations, behaviour, ...
- Audit (audit): Who, when, what
- Rectification (RE): Remediation, sanction, compensation, explanations, ...
- The simplest form is `(NOT UE) => (audit => RE)`
- Quantifiers and modal operators makes it more complex

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`
- Authorizations

```
// Prohibition to process
FORALL d:Data (d.subject==Kim) => DENY Hospital.process(d)
```

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`
- Authorizations

```
// Prohibition to process
FORALL d:Data (d.subject==Kim) => DENY Hospital.process(d)
```

- Obligations

```
// Obligation to notify
FORALL d:Data ((d.subject==Kim) AND Hospital.process(d)) =>
         Hospital.notify[Kim]("processing")
```

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`
- Authorizations

```
// Prohibition to process
FORALL d:Data (d.subject==Kim) => DENY Hospital.process(d)
```

- Obligations

```
// Obligation to notify
FORALL d:Data ((d.subject==Kim) AND Hospital.process(d)) =>
         Hospital.notify[Kim]("processing")
```

- Linear temporal logic

```
ALWAYS FORALL d:Data (Kim.input[Hospital](d) =>
       SOMETIME EXISTS ack:Receipt Hospital.send[Kim](ack))
```

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`
- Authorizations

```
// Prohibition to process
FORALL d:Data (d.subject==Kim) => DENY Hospital.process(d)
```

- Obligations

```
// Obligation to notify
FORALL d:Data ((d.subject==Kim) AND Hospital.process(d)) =>
         Hospital.notify[Kim]("processing")
```

- Linear temporal logic

```
ALWAYS FORALL d:Data (Kim.input[Hospital](d) =>
       SOMETIME EXISTS ack:Receipt Hospital.send[Kim](ack))
```

- Privacy concerns, delegation, protocols ... [BGRS15, RSDO16]

AAL expressiveness

- Message/service: `sender.action[receiver](arguments)`
- Internal action: `sender.action(arguments)`
- Authorizations

```
// Prohibition to process
FORALL d:Data (d.subject==Kim) => DENY Hospital.process(d)
```

- Obligations

```
// Obligation to notify
FORALL d:Data ((d.subject==Kim) AND Hospital.process(d)) =>
         Hospital.notify[Kim]("processing")
```

- Linear temporal logic

```
ALWAYS FORALL d:Data (Kim.input[Hospital](d) =>
       SOMETIME EXISTS ack:Receipt Hospital.send[Kim](ack))
```

- Privacy concerns, delegation, protocols ... [BGRS15, RSDO16]
- Evaluate the linear temporal approach (only 3 duration in the seven policies)

AccLab

- A tool support to experiment with accountability policies

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture
- A syntax directed editor for AAL with semantic controls

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture
- A syntax directed editor for AAL with semantic controls
- Verification via a link to the TSPASS prover

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture
- A syntax directed editor for AAL with semantic controls
- Verification via a link to the TSPASS prover
- A simulation tool and a monitoring tool

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture
- A syntax directed editor for AAL with semantic controls
- Verification via a link to the TSPASS prover
- A simulation tool and a monitoring tool
- Templates and some type constructions not yet fully implemented

AccLab

- A tool support to experiment with accountability policies
- An end-to-end perspective: from specification until enforcement via monitoring
- A component GUI to design the system architecture
- A syntax directed editor for AAL with semantic controls
- Verification via a link to the TSPASS prover
- A simulation tool and a monitoring tool
- Templates and some type constructions not yet fully implemented
- `https://github.com/hkff/AccLab`

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession
2. I shall only use the laptop for University related purposes.

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession
2. I shall only use the laptop for University related purposes.
3. I shall keep the laptop in good working order and will notify I.T. Services of any defect or malfunction during my use.

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession
2. I shall only use the laptop for University related purposes.
3. I shall keep the laptop in good working order and will notify I.T. Services of any defect or malfunction during my use.
4. I shall not install and / or download any unauthorized software and / or applications

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession
2. I shall only use the laptop for University related purposes.
3. I shall keep the laptop in good working order and will notify I.T. Services of any defect or malfunction during my use.
4. I shall not install and / or download any unauthorized software and / or applications
5. I shall not allow the laptop to be used by an unknown or unauthorized person. I assume the responsibility for the actions of others while using the laptop

The laptop user agreement

In accepting the use of a University laptop, I agree to the following conditions:

1. I understand that I am solely responsible for the laptop whilst in my possession
2. I shall only use the laptop for University related purposes.
3. I shall keep the laptop in good working order and will notify I.T. Services of any defect or malfunction during my use.
4. I shall not install and / or download any unauthorized software and / or applications
5. I shall not allow the laptop to be used by an unknown or unauthorized person. I assume the responsibility for the actions of others while using the laptop
6. ...

Our working process

- First reading and analysis of most of the seven policies

Our working process

- First reading and analysis of most of the seven policies
- Extraction and construction of the information system (types, roles, actions, conditions, etc)

Our working process

- First reading and analysis of most of the seven policies
- Extraction and construction of the information system (types, roles, actions, conditions, etc)
- Choose accountability patterns for translating policy sentences

Our working process

- First reading and analysis of most of the seven policies
- Extraction and construction of the information system (types, roles, actions, conditions, etc)
- Choose accountability patterns for translating policy sentences
- Formalize sentences in AAL

Our working process

- First reading and analysis of most of the seven policies
- Extraction and construction of the information system (types, roles, actions, conditions, etc)
- Choose accountability patterns for translating policy sentences
- Formalize sentences in AAL
- Verify some correct usage and violation cases

Our working process

- First reading and analysis of most of the seven policies
- Extraction and construction of the information system (types, roles, actions, conditions, etc)
- Choose accountability patterns for translating policy sentences
- Formalize sentences in AAL
- Verify some correct usage and violation cases
- And iterate ...

Audit and rectification

- Nothing about the audit time and process

Audit and rectification

- Nothing about the audit time and process
- But sometime information about the need to monitor the user activity

Audit and rectification

- Nothing about the audit time and process
- But sometime information about the need to monitor the user activity
- `auditor.audit[LHU]()`

Audit and rectification

- Nothing about the audit time and process
- But sometime information about the need to monitor the user activity
- `auditor.audit[LHU]()`
- More precise descriptions are possible as soon as we get the details

Audit and rectification

- Nothing about the audit time and process
- But sometime information about the need to monitor the user activity
- `auditor.audit[LHU]()`
- More precise descriptions are possible as soon as we get the details
- Quite the same problem for rectification (clause 10)

Listing 5: Simple Rectification in AAL

```
TEMPLATE LHURectificationPolicy (resp:AllPerson)
       (@EligibleUser(resp) => LHU.disciplinaryAction[resp]())
```

The usage expression

- Clause 1 to 9 express permission, interdiction, obligation, behaviour

The usage expression

- Clause 1 to 9 express permission, interdiction, obligation, behaviour
- We show a part coming from clause 1 and clause 5

The usage expression

- Clause 1 to 9 express permission, interdiction, obligation, behaviour
- We show a part coming from clause 1 and clause 5
- We interpret that there is a laptop assigned and the user should return it to the university secretary

Listing 8: Laptop Policy Agreement in AAL

```
TEMPLATE laptopUA (resp:AllPerson)(
  (FORALL laptop:Laptop FORALL p:Purpose
     (PERMIT resp.use[laptop](p)) =>
      (@EligibleUser(resp) AND @assigned(resp, laptop)))
  AND
  (FORALL laptop:Laptop FORALL p:Purpose
     (@EligibleUser(resp) AND @assigned(resp, laptop))
     => (SOMETIME resp.bringBack[LHUsecretary]()))
  AND ...)
```

A First Global View

- The Laptop Accountability Policy in AAL

```
TEMPLATE LaptopAccountabilityPolicy (resp:AllPerson) (
  // from 1 to 9 but clause 7
  @template(ACCOUNT, @template(laptopUA, resp),
                     @template(LHURectificationPolicy, resp))
   AND
  // from clause 7 since it is another schema
  @template(ACCUNTIL, @template(condition7, resp),
     @template(achievement7, resp),
     @template(LHUWeakRectificationAndPay, resp))}))
```

A First Global View

- The Laptop Accountability Policy in AAL

```
TEMPLATE LaptopAccountabilityPolicy (resp:AllPerson) (
  // from 1 to 9 but clause 7
  @template(ACCOUNT, @template(laptopUA, resp),
                     @template(LHURectificationPolicy, resp))
   AND
  // from clause 7 since it is another schema
  @template(ACCUNTIL, @template(condition7, resp),
     @template(achievement7, resp),
     @template(LHUWeakRectificationAndPay, resp))}))
```

- The ACCOUNT Accountability Template in AAL

```
TEMPLATE ACCOUNT(UE:Template, RE:Template) (
 ALWAYS ((NOT UE) => (ALWAYS  (auditor.audit[LHU]() => RE))))
```

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

```
1: ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))
```

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

1: `ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))`

2: `FORALL resp:Any ALWAYS ((NOT UE(resp)) => RE(resp))`

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

1: `ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))`

2: `FORALL resp:Any ALWAYS ((NOT UE(resp)) => RE(resp))`

- The first formula implies the second but the opposite is false

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

1: `ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))`

2: `FORALL resp:Any ALWAYS ((NOT UE(resp)) => RE(resp))`

- The first formula implies the second but the opposite is false
- Principle of separated normal form to assist in writing

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

1: `ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))`

2: `FORALL resp:Any ALWAYS ((NOT UE(resp)) => RE(resp))`

- The first formula implies the second but the opposite is false
- Principle of separated normal form to assist in writing
- `ALWAYS (cond => conc)` with `cond` FOL formula and `conc` FOTL formulas with only `NEXT` and `SOMETIME`

Accountability Schemas

- Proposing schemas (patterns) to represent accountability practices
- Basic scheme: `(NOT UE) => (audit => RE)`
- Templates: `ACCOUNT`, `ACCUNTIL`, `ATNEXT`, ...

1: `ALWAYS FORALL resp:Any ((NOT UE(resp)) => RE(resp))`

2: `FORALL resp:Any ALWAYS ((NOT UE(resp)) => RE(resp))`

- The first formula implies the second but the opposite is false
- Principle of separated normal form to assist in writing
- `ALWAYS (cond => conc)` with `cond` FOL formula and `conc` FOTL formulas with only `NEXT` and `SOMETIME`
- There are more precise descriptions in the literature (see [DFK06, Fis11])

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an `UNTIL` which has a non readable separated form

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an `UNTIL` which has a non readable separated form
- Rather we propose the following one (`A`, `B` FOL expressions)

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an `UNTIL` which has a non readable separated form
- Rather we propose the following one (`A`, `B` FOL expressions)
- ACCUNTIL scheme:

```
(A UNTIL B) OR ((NOT B) UNTIL ((NOT (A OR B)) => RE))
```

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an `UNTIL` which has a non readable separated form
- Rather we propose the following one (`A`, `B` FOL expressions)
- ACCUNTIL scheme:

```
(A UNTIL B) OR ((NOT B) UNTIL ((NOT (A OR B)) => RE))
```

- We drop away an infinite behaviour in the negative part

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an UNTIL which has a non readable separated form
- Rather we propose the following one (A, B FOL expressions)
- ACCUNTIL scheme:

```
(A UNTIL B) OR ((NOT B) UNTIL ((NOT (A OR B)) => RE))
```

- We drop away an infinite behaviour in the negative part
- Clause 8 needs an ATNEXT scheme to linearly encode "within 24 hours"

Finite Trace

- Standard semantics is based on infinite traces while real monitoring is finite trace based
- Clause 7: "should delete saved work before to return the assigned laptop"
- Needs an UNTIL which has a non readable separated form
- Rather we propose the following one (A, B FOL expressions)
- ACCUNTIL scheme:

```
(A UNTIL B) OR ((NOT B) UNTIL ((NOT (A OR B)) => RE))
```

- We drop away an infinite behaviour in the negative part
- Clause 8 needs an ATNEXT scheme to linearly encode "within 24 hours"
- It means that the violation is observed "at next" state and then the rectification may occur at this state or later

Clause 5

- We interpret it as: If an eligible user permits another to use his assigned laptop he will be responsible in case of violation

```
(FORALL laptop:Laptop FORALL any:AllPerson FORALL  p:Purpose
   (PERMIT any.use[laptop](p)) =>
        (@EligibleUser(resp) AND @assigned(resp, laptop)
            AND @known(resp, any) AND (NOT @unauthorized(any)))
```

Clause 5

- We interpret it as: If an eligible user permits another to use his assigned laptop he will be responsible in case of violation

```
(FORALL laptop:Laptop FORALL any:AllPerson FORALL  p:Purpose
   (PERMIT any.use[laptop](p)) =>
        (@EligibleUser(resp) AND @assigned(resp, laptop)
            AND @known(resp, any) AND (NOT @unauthorized(any)))
```

- Violation then rectification of the responsible person

```
(ALWAYS  ((walid.sign[LaptopAccountabilityPolicy]()
              AND @EligibleUser(walid))
 => ALWAYS ((@assigned(walid, mylap) AND @Laptop(mylap)
              AND  @AllPerson(me) AND @Purpose(pur)
              AND ((NOT @known(walid, me)) OR @unauthorized(me))
              AND me.use[mylap](pur))
                 => ALWAYS ((auditor.audit[LHU]())
                           => LHU.disciplinaryAction[walid]())))
```

The Current Specification

- Several steps of improvements and corrections

The Current Specification

- Several steps of improvements and corrections
- We use two accountability templates and separate clause 7 and 8

The Current Specification

- Several steps of improvements and corrections
- We use two accountability templates and separate clause 7 and 8
- Laptop usage = clause [1 .. 5, 9] + 7 + 8

The Current Specification

- Several steps of improvements and corrections
- We use two accountability templates and separate clause 7 and 8
- Laptop usage = clause [1 .. 5, 9] + 7 + 8
- Clause 6 calls external policies (IT usage, data, etc)

The Current Specification

- Several steps of improvements and corrections
- We use two accountability templates and separate clause 7 and 8
- Laptop usage = clause [1 .. 5, 9] + 7 + 8
- Clause 6 calls external policies (IT usage, data, etc)
- Input: 1 page, 10 sentences, 83KO

The Current Specification

- Several steps of improvements and corrections
- We use two accountability templates and separate clause 7 and 8
- Laptop usage = clause [1 .. 5, 9] + 7 + 8
- Clause 6 calls external policies (IT usage, data, etc)
- Input: 1 page, 10 sentences, 83KO
- Output: 9 clauses, 5 templates, 4KO

```
TEMPLATE LaptopAccountabilityPolicy (any:AllPerson) (
        @template(ACCOUNT, @template(laptopUA, any),
                           @template(LHURectificationPolicy, any))
        AND
        @template(ATNEXT, @template(clause8, any),
                          @template(LHURectificationPolicy, any))
        AND
        @template(ACCOUNT, @template(clause7, any),
                           @template(PayToLHU, any)))
```

Verification

- Satisfiable with few behaviour, and the type declarations

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description
- We can check correct behaviour (should be satisfiable)

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description
- We can check correct behaviour (should be satisfiable)
- We check various formulas denoting a violation of the usage and a rectification of the responsible agent

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description
- We can check correct behaviour (should be satisfiable)
- We check various formulas denoting a violation of the usage and a rectification of the responsible agent
- We saw one example with the clause 5 but we prove at least one for each sentence of the policy and few variations

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description
- We can check correct behaviour (should be satisfiable)
- We check various formulas denoting a violation of the usage and a rectification of the responsible agent
- We saw one example with the clause 5 but we prove at least one for each sentence of the policy and few variations
- TSPASS generates between 2000 - 4000 CNF in less than 1 s

Verification

- Satisfiable with few behaviour, and the type declarations
- No property in the laptop policy description
- We can check correct behaviour (should be satisfiable)
- We check various formulas denoting a violation of the usage and a rectification of the responsible agent
- We saw one example with the clause 5 but we prove at least one for each sentence of the policy and few variations
- TSPASS generates between 2000 - 4000 CNF in less than 1 s
- However, our translator is less efficient nearly 4s

Lessons

- Classic problems in analyzing natural texts

Lessons

- Classic problems in analyzing natural texts
- Too many things lacking, we should invent most of the information system, the behaviours, ...

Lessons

- Classic problems in analyzing natural texts
- Too many things lacking, we should invent most of the information system, the behaviours, ...
- Audit and rectification are missing or not detailed

Lessons

- Classic problems in analyzing natural texts
- Too many things lacking, we should invent most of the information system, the behaviours, ...
- Audit and rectification are missing or not detailed
- Thus our specification is rather our interpretation of the laptop policy

Lessons

- Classic problems in analyzing natural texts
- Too many things lacking, we should invent most of the information system, the behaviours, ...
- Audit and rectification are missing or not detailed
- Thus our specification is rather our interpretation of the laptop policy
- Templates are useful for readability, structuration, reuse

Lessons

- Classic problems in analyzing natural texts
- Too many things lacking, we should invent most of the information system, the behaviours, ...
- Audit and rectification are missing or not detailed
- Thus our specification is rather our interpretation of the laptop policy
- Templates are useful for readability, structuration, reuse
- Our language strengths to distinguish the proper behaviour from the policy: Example with “the sign and then accept the policy” clause

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues
- Targeting FOL with sorts and interpreted theories

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues
- Targeting FOL with sorts and interpreted theories
- Will add more quantifiers and could also lead to decidability issues

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues
- Targeting FOL with sorts and interpreted theories
- Will add more quantifiers and could also lead to decidability issues
- But writing is more uniform

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues
- Targeting FOL with sorts and interpreted theories
- Will add more quantifiers and could also lead to decidability issues
- But writing is more uniform
- This logic is actually well-known (a map of the decidable fragments exists)

Pro and cons of the linear way

- Even LTL is subtle but with FOTL it is really tricky
- Behaviour and accountability clauses are restricted by the monodic constraint
- Equality, functions are lacking but are known to lead to decidability issues
- Targeting FOL with sorts and interpreted theories
- Will add more quantifiers and could also lead to decidability issues
- But writing is more uniform
- This logic is actually well-known (a map of the decidable fragments exists)
- There are numerous solvers with equality, functions, etc (SPASS, Z3, CVC4, YICES, ...)

Conclusion and future work

- AAL provides flexibility and expressiveness in policies

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover
- In progress: the case study

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover
- In progress: the case study
- Future work

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover
- In progress: the case study
- Future work
  - Improve the tool support

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover
- In progress: the case study
- Future work
  - Improve the tool support
  - Target an SMT solver

Conclusion and future work

- AAL provides flexibility and expressiveness in policies
- Templates are convenient
- Linear temporal logic and FOL: pro and cons
- The main problem is the weaknesses of the current prover
- In progress: the case study
- Future work
  - Improve the tool support
  - Target an SMT solver
  - Rationalize schemas

Related Work

- Accountability: A complex notion which crosscuts several domains

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)
- Several theories

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)
- Several theories
- Really few tools

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)
- Several theories
- Really few tools
- CLAN: dynamic deontic logic without quantifiers and trace compliance

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)
- Several theories
- Really few tools
- CLAN: dynamic deontic logic without quantifiers and trace compliance
- AIR: rule-based language for the semantic Web focusing on explanations

Related Work

- Accountability: A complex notion which crosscuts several domains
- Few surveys [WABL+08, LZW10, ZX12, GHI+12]
- Two main views: Software engineering or multi-agent (AI)
- Several theories
- Really few tools
- CLAN: dynamic deontic logic without quantifiers and trace compliance
- AIR: rule-based language for the semantic Web focusing on explanations
- Zou et al. [ZWL10]: formal service contract for accountable SaaS services

LHU Information Technology Policies

- www.hope.ac.uk/aboutus/itservices/policies/

LHU Information Technology Policies

- `www.hope.ac.uk/aboutus/itservices/policies/`
- Data protection aligned with DPA 1998 and guidance for staff (10 pages, 236KO, 30 clauses, 6KO)

LHU Information Technology Policies

- `www.hope.ac.uk/aboutus/itservices/policies/`
- Data protection aligned with DPA 1998 and guidance for staff (10 pages, 236KO, 30 clauses, 6KO)
- IT usage policy: general policy for University IT services (6 pages, 266KO, 18 clauses, 4KO)

LHU Information Technology Policies

- `www.hope.ac.uk/aboutus/itservices/policies/`
- Data protection aligned with DPA 1998 and guidance for staff (10 pages, 236KO, 30 clauses, 6KO)
- IT usage policy: general policy for University IT services (6 pages, 266KO, 18 clauses, 4KO)
- Portable data security policy for portable devices and sensitive data (4 pages, 191KO, 18 clauses, 3KO)

LHU Information Technology Policies

- `www.hope.ac.uk/aboutus/itservices/policies/`
- Data protection aligned with DPA 1998 and guidance for staff (10 pages, 236KO, 30 clauses, 6KO)
- IT usage policy: general policy for University IT services (6 pages, 266KO, 18 clauses, 4KO)
- Portable data security policy for portable devices and sensitive data (4 pages, 191KO, 18 clauses, 3KO)
- Information security policy: general security policy (12 pages, 264KO, 0)

LHU Information Technology Policies

- `www.hope.ac.uk/aboutus/itservices/policies/`
- Data protection aligned with DPA 1998 and guidance for staff (10 pages, 236KO, 30 clauses, 6KO)
- IT usage policy: general policy for University IT services (6 pages, 266KO, 18 clauses, 4KO)
- Portable data security policy for portable devices and sensitive data (4 pages, 191KO, 18 clauses, 3KO)
- Information security policy: general security policy (12 pages, 264KO, 0)
- Wireless policy and procedure: To manage wireless network (7 pages, 248KO, 0)


Walid Benghabrit, Hervé Grall, Jean-Claude Royer, and Mohamed Sellami.
Abstract accountability language: Translation, compliance and application.
In *APSEC*, New Delhi, India, December 2015. IEEE Computer Society.


Anatoli Degtyarev, Michael Fisher, and Boris Konev.
Monodic temporal resolution.
ACM Transactions on Computational Logic, 7(1):108–150, January 2006.


Michael Fisher.
An Introduction to Practical Formal Methods using Temporal Logic.
Wiley, 2011.


Daniel Guagnin, Leon Hempel, Carla Ilten, Inga Kroener, Daniel Neyland, and Hector Postigo, editors.
Managing Privacy through Accountability.
Palgrave Macmillan, 2012.


Kwei-Jay Lin, Joe Zou, and Yan Wang.
Accountability computing for e-society.
In *24th Advanced Information Networking and Applications Conference (AINA)*, pages 34–41. Ieee, 2010.


Jean-Claude Royer and Anderson Santana De Oliveira.
AAL and static conflict detection in policy.
In *CANS, 15th International Conference on Cryptology and Network Security*, LNCS, pages 367–382. Springer, November 2016.


Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman.

Information accountability.
Commun. ACM, 51(6):82–87, June 2008.


Joe Zou, Yan Wang, and Kwei-Jay Lin.
A formal service contract model for accountable saaS and cloud services.
In *International Conference on Services Computing*, pages 73–80. IEEE Computer Society, 2010.


Yang Xiao Zhifeng Xiao, Nandhakumar Kathiresshan.
A survey of accountability in computer networks and distributed systems.
Security and Communication Networks, 2012.