
A. Armando\textsuperscript{1,2}, E. Giunchiglia\textsuperscript{2}, M. Maratea\textsuperscript{2} and S. E. Ponta\textsuperscript{3}

\textsuperscript{1}Security & Trust Unit, Fondazione Bruno Kessler, Trento, Italy
\textsuperscript{2}DIBRIS, University of Genova
\textsuperscript{3}SAP Research Sophia-Antipolis, Mougins, France

ICAPS 2013
Rome, June 13th 2013
Business processes under authorization constraints are

- sets of coordinated activities (workflow),
- subject to an access control policy stating which agent can execute which activity.

Analysis of Business Processes

- Difficult to get right.
- Traditional verification techniques do not ensure the needed level of assurance.
Business processes under authorization constraints are
- sets of coordinated activities (workflow),
- subject to an access control policy stating which agent can execute which activity.

Analysis of Business Processes
- Difficult to get right.
- Traditional verification techniques do not ensure the needed level of assurance.
High level formalisms for expressing actions and how they affect the world, described by states, i.e. sets of atoms called fluents.

Expressive languages, e.g. non-determinism, indirect, and conditional effects.

Automated analysis support through existing tools.

In the context of business processes: Allow for the separate specification of workflow and security policy in a natural way.
Contributions

- Formal specification of business processes under authorization constraints based on the action language $C$ that allows for a natural and concise modeling.

- **Automatic analysis** of the formal specification by using the Causal Calculator (CCALC).

- **Effectiveness** of the approach assessed applying it against a Loan Origination Process (LOP), a business process from the banking domain.
1. Introduction

2. Business Process under Authorization Constraints

3. Action Language $C$

4. Formal Modeling of Business Processes with $C$

5. Automatic Analysis

6. Conclusions
Business Processes under authorization constraints are

- sets of coordinated activities (workflow), here of the LOP represented with a Petri net.

- subject to an access control policy stating which agent can execute which activity.
Business Processes under authorization constraints are

- sets of coordinated activities (workflow), here represented with a Petri net.
- subject to an access control policy stating which agent can execute which activity.

specified by an access control model, e.g. RBAC,
- delegation rules,
- additional security constraints, e.g. Separation of Duty (SoD).
LOP - RBAC Model

- **User Assignment** (Alice is a director)
- **Permission Assignment** (a director can execute the approval task)
- **Hierarchy of roles** (a director is more senior than a manager)

Examples of permission assignment for the LOP

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputCustData</td>
<td>preprocessor</td>
</tr>
<tr>
<td>prepareContract</td>
<td>postprocessor</td>
</tr>
<tr>
<td>intRating</td>
<td>if (isindustrial) then postprocessor else preprocessor</td>
</tr>
<tr>
<td>extRating</td>
<td>if (interrupted) then director else supervisor</td>
</tr>
<tr>
<td>approve</td>
<td>if (lowrisk) then manager else director</td>
</tr>
<tr>
<td>sign</td>
<td>if (isindustrial) then director else manager</td>
</tr>
</tbody>
</table>
User Assignment (Alice is a director)

Permission Assignment (a director can execute the approval task)

Hierarchy of roles (a director is more senior than a manager)

Examples of permission assignment for the LOP

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputCustData</td>
<td>preprocessor</td>
</tr>
<tr>
<td>prepareContract</td>
<td>postprocessor</td>
</tr>
<tr>
<td>intRating</td>
<td>if (isindustrial) then postprocessor else preprocessor</td>
</tr>
<tr>
<td>extRating</td>
<td>if (interrupted) then director else supervisor</td>
</tr>
<tr>
<td>approve</td>
<td>if (lowrisk) then manager else director</td>
</tr>
<tr>
<td>sign</td>
<td>if (isindustrial) then director else manager</td>
</tr>
</tbody>
</table>
LOP - Delegation of Tasks

A *delegation* represents a typical flexibility requirement. It can be expressed with

\[ \langle \text{PreCondition}, \text{ARole}, \text{DRole}, \text{Task} \rangle \]

- If *PreCondition* holds and
- *ARole* is authorized to perform *Task* according to the permission assignment relation,
- then *ARole* can delegate *DRole* to execute *Task*.

Example of delegation rule for the LOP

- **D**: \( \langle \text{intRatingOK}, \text{director}, \text{manager}, \text{approve} \rangle \).
Separation of Duty

Critical tasks must be executed by different agents.

Widely used in the financial setting to prevent frauds.

SoD Constraints of the LOP

<table>
<thead>
<tr>
<th>Name</th>
<th>Critical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>inputCustData, prepareContract, intRating, extRating</td>
</tr>
<tr>
<td>C2</td>
<td>intRating, approve</td>
</tr>
<tr>
<td>C3</td>
<td>extRating, approve</td>
</tr>
<tr>
<td>C4</td>
<td>prepareContract, approve, sign</td>
</tr>
</tbody>
</table>
Separation of Duty

Critical tasks must be executed by different agents.

Widely used in the financial setting to prevent frauds.

SoD Constraints of the LOP

<table>
<thead>
<tr>
<th>Name</th>
<th>Critical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>inputCustData, prepareContract, intRating, extRating</td>
</tr>
<tr>
<td>C2</td>
<td>intRating, approve</td>
</tr>
<tr>
<td>C3</td>
<td>extRating, approve</td>
</tr>
<tr>
<td>C4</td>
<td>prepareContract, approve, sign</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Business Process under Authorization Constraints
3. Action Language $\mathcal{C}$
4. Formal Modeling of Business Processes with $\mathcal{C}$
5. Automatic Analysis
6. Conclusions
The Action Language \( \mathcal{C} \)

Expressive propositional action language given by

- fluent and action symbols,
- causal laws, i.e. propositions
  - static laws \( \text{caused } F \text{ if } G \)
  - dynamic laws \( \text{caused } F \text{ if } G \text{ after } H \)

Some useful abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expanded Form</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonexecutable ( H \text{ if } F ).</td>
<td>( \text{caused } \bot \text{ after } H \land F ).</td>
<td>( \neg F ) is a precondition of ( H )</td>
</tr>
<tr>
<td>( H \text{ causes } F \text{ if } G ).</td>
<td>( \text{caused } F \text{ if } \top \text{ after } G \land H ).</td>
<td>( F ) is true after ( H ) is executed in a state in which ( G ) is true</td>
</tr>
<tr>
<td>( H \text{ may cause } F \text{ if } G ).</td>
<td>( \text{caused } F \text{ if } F \text{ after } H \land G ).</td>
<td>( F ) is true by default after ( H ) is executed in a state in which ( G ) is true</td>
</tr>
<tr>
<td>default ( F ).</td>
<td>( \text{caused } F \text{ if } F ).</td>
<td>( F ) is true by default</td>
</tr>
<tr>
<td>constraint ( F ).</td>
<td>( \text{caused } \bot \text{ if } \neg F ).</td>
<td>( F ) must be true</td>
</tr>
</tbody>
</table>

The CCALC tool supports automated reasoning of \( \mathcal{C} \) specifications.
The Action Language \(\mathcal{C}\)

Expressive propositional action language given by

- fluent and action symbols,
- causal laws, i.e. propositions
  
  - static laws  \(\text{caused } F \text{ if } G\)
  
  - dynamic laws  \(\text{caused } F \text{ if } G \text{ after } H\)

Some useful abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expanded Form</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonexecutable</td>
<td>(H \text{ if } F).</td>
<td>(\text{caused } \bot \text{ after } H \land F).</td>
</tr>
<tr>
<td>(H \text{ causes } F \text{ if } G.)</td>
<td>(\text{caused } F \text{ if } \top \text{ after } G \land H).</td>
<td>(F) is true after (H) is executed in a state in which (G) is true</td>
</tr>
<tr>
<td>(H \text{ may cause } F \text{ if } G.)</td>
<td>(\text{caused } F \text{ if } F \text{ after } H \land G).</td>
<td>(F) is true by default after (H) is executed in a state in which (G) is true</td>
</tr>
<tr>
<td>default (F.)</td>
<td>(\text{caused } F \text{ if } F).</td>
<td>(F) is true by default</td>
</tr>
<tr>
<td>constraint (F.)</td>
<td>(\text{caused } \bot \text{ if } \neg F).</td>
<td>(F) must be true</td>
</tr>
</tbody>
</table>

The CCALC tool supports automated reasoning of \(\mathcal{C}\) specifications.
The Action Language $\mathcal{C}$

Expressive propositional action language given by
- fluent and action symbols,
- causal laws, i.e. propositions
  - static laws $\text{caused } F \text{ if } G$
  - dynamic laws $\text{caused } F \text{ if } G \text{ after } H$

Some useful abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expanded Form</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonexecutable $H \text{ if } F.$</td>
<td>caused $\perp$ after $H \land F.$</td>
<td>$\neg F$ is a precondition of $H$</td>
</tr>
<tr>
<td>$H$ causes $F$ if $G.$</td>
<td>caused $F$ if $\top$ after $G \land H.$</td>
<td>$F$ is true after $H$ is executed in a state in which $G$ is true</td>
</tr>
<tr>
<td>$H$ may cause $F$ if $G.$</td>
<td>caused $F$ if $F$ after $H \land G.$</td>
<td>$F$ is true by default after $H$ is executed in a state in which $G$ is true</td>
</tr>
<tr>
<td>default $F.$</td>
<td>caused $F$ if $F.$</td>
<td>$F$ is true by default</td>
</tr>
<tr>
<td>constraint $F.$</td>
<td>caused $\perp$ if $\neg F.$</td>
<td>$F$ must be true</td>
</tr>
</tbody>
</table>

The **CCALC** tool supports automated reasoning of $\mathcal{C}$ specifications.
Outline

1. Introduction
2. Business Process under Authorization Constraints
3. Action Language $C$
4. Formal Modeling of Business Processes with $C$
5. Automatic Analysis
6. Conclusions
Expressiveness of the approach

In the context of business processes, with $C$ it is possible to model:

- execution of tasks,
- (deterministic, non-deterministic, conditional) effects of tasks,
- delegation rules,
- access control policy,
- SoD constraints,
- ...
We show how the LOP can be specified in $\mathcal{C}$.

<table>
<thead>
<tr>
<th>Fluent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>activated($a, r$)</td>
<td>agent $a$ is playing role $r$</td>
</tr>
<tr>
<td>ua($a, r$)</td>
<td>agent $a$ is assigned to role $r$</td>
</tr>
<tr>
<td>pa($r, t$)</td>
<td>role $r$ has the permission to perform task $t$</td>
</tr>
<tr>
<td>granted($a, r, t$)</td>
<td>agent $a$ obtained by means of role $r$ the permission to perform task $t$</td>
</tr>
<tr>
<td>delegated($a, r, t$)</td>
<td>agent $a$ is delegated by an agent in role $r$ to perform task $t$</td>
</tr>
<tr>
<td>executed($a, r, t$)</td>
<td>agent $a$ has executed task $t$ through role $r$</td>
</tr>
<tr>
<td>senior($r_1, r_2$)</td>
<td>role $r_1$ is more senior than or is as senior as $r_2$</td>
</tr>
<tr>
<td>lowrisk</td>
<td>the risk associated with the loan is low</td>
</tr>
<tr>
<td>interrupted</td>
<td>the execution of the process is interrupted</td>
</tr>
<tr>
<td>$p_1, \ldots, p_{18}$</td>
<td>the places of the extended Petri net</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>exec($a, r, t$)</td>
<td>execution of task $t$ by agent $a$ by means of role $r$</td>
</tr>
<tr>
<td>d($a_1, a_2$)</td>
<td>agent $a_1$ delegates agent $a_2$ by means of delegation rule $d$</td>
</tr>
<tr>
<td>disable($a_1, a_2, t$)</td>
<td>agent $a_1$ disables agent $a_2$ from performing task $t$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: task intRating
Execution of Tasks in $c$

Example: task intRating

**Preconditions**

\[ \text{nonexecutable } \text{exec}(a, r, \text{intRating}) \]

\[ \text{if } \neg(p_6 \land \text{granted}(a, r, \text{intRating})) \]

**Deterministic effects**

\[ \text{exec}(a, r, \text{intRating}) \]

\[ \text{causes } p_8 \land \neg p_6 \land \text{executed}(a, \text{intRating}) \]

**Non-deterministic effects**

\[ \text{exec}(a, r, \text{intRating}) \] may cause intRatingOK

**Conditional effects**

\[ \text{exec}(a, r, \text{intRating}) \] may cause highProfileIndCust if isindustrial
Delegation Rules in $\mathcal{C}$

Example: delegation rule $D$

$\langle \text{intRatingOK}, \text{director}, \text{manager}, \text{approve}\rangle$.

### Preconditions

**nonexecutable** $d(a_1, a_2)$ if $\neg (ua(a_1, \text{director}) \land ua(a_2, \text{manager}) \land pa(\text{director}, \text{approve}) \land \text{intRatingOK})$

### Deterministic effects

$d(a_1, a_2)$ **causes** $\text{granted}(a_1, \text{director}, \text{approve}) \land \text{delegated}(a_2, \text{director}, \text{approve})$

for all $a_1 \neq a_2$. 
Access Control Policy in $c$

RBAC policy establishes which agent can perform which task

\[
\text{caused } \text{granted}(a, r, t) \\
\text{if } (\text{ua}(a, r) \land \text{activated}(a, r) \land \text{pa}(r_1, t) \land \text{senior}(r, r_1))
\]

(and a corresponding law for “causing” $\neg\text{granted}(a, r, t)$).

A specific permission assignment

\[
\text{caused } \text{pa}(&\text{director, approve}) \text{ if } \neg\text{lowrisk} \\
\text{default } \neg\text{pa}(&\text{director, approve}) \\
\text{caused } \text{pa}(\text{manager, approve}) \text{ if } \text{lowrisk} \\
\text{default } \neg\text{pa}(\text{manager, approve})
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>approve</td>
<td>if (lowrisk) then manager else director</td>
</tr>
</tbody>
</table>
RBAC policy establishes which agent can perform which task

\[
\text{caused } \text{granted}(a, r, t) \\
\text{if } (\text{ua}(a, r) \land \text{activated}(a, r) \land \text{pa}(r_1, t) \land \text{senior}(r, r_1))
\]

(and a corresponding law for “causing” \(\neg \text{granted}(a, r, t)\)).

A specific permission assignment

\[
\begin{align*}
\text{caused } & \text{pa}(\text{director, approve}) \text{ if } \neg \text{lowrisk} \\
\text{default } & \neg \text{pa}(\text{director, approve}) \\
\text{caused } & \text{pa}(\text{manager, approve}) \text{ if } \text{lowrisk} \\
\text{default } & \neg \text{pa}(\text{manager, approve})
\end{align*}
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>approve</td>
<td>(\text{if (lowrisk) then manager else director})</td>
</tr>
</tbody>
</table>
Access Control Policy in $c$

RBAC policy establishes which agent can perform which task

\[
\text{caused } \text{granted}(a, r, t) \\
\text{if } (\text{ua}(a, r) \land \text{activated}(a, r) \land \text{pa}(r_1, t) \land \text{senior}(r, r_1))
\]

(and a corresponding law for “causing” $\neg \text{granted}(a, r, t)$).

A specific permission assignment

\[
\text{caused } \text{pa}(\text{director, approve}) \text{ if } \neg \text{lowrisk} \\
\text{default } \neg \text{pa}(\text{director, approve}) \\
\text{caused } \text{pa}(\text{manager, approve}) \text{ if } \text{lowrisk} \\
\text{default } \neg \text{pa}(\text{manager, approve})
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>approve</td>
<td>if (lowrisk) then manager else director</td>
</tr>
</tbody>
</table>
Example: SoD constraint C4

\[
\text{constraint } \neg (\text{executed}(a, r_1, \text{prepareContract}) \land \text{executed}(a, r_2, \text{approve}) \land \text{executed}(a, r_3, \text{sign}))
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Critical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>prepareContract, approve, sign</td>
</tr>
</tbody>
</table>
Expressiveness of the approach

With $\mathcal{C}$ it is also possible to model:

- internal dependencies within the process. (e.g. caused lowrisk if $\neg$highValue $\land$ intRatingOK)
- internal and external events, which are not explicitly represented in the model (e.g. activation of a role).
- invocation of entities, e.g. invocation of Credit Bureau,
- all remaining parts of the workflow, e.g. “dummy” activities to begin/end a flow,
- exceptions
Experiments

Motivation: Given the resulting specifications, and the interplay with the security policy, it may not be trivial to establish if other desirable security properties hold (e.g. because entailed by the already enforced security policy), or if it is necessary to revise the model and/or the security policy in order.

Scenario for the LOP: We defined a complete permission and user assignment, with a number of delegation rules and SoD constraints (comprised the ones seen before).

Problem (Verification of Security Properties)

If the process terminates successfully, then no single agent has performed all the tasks intRating, extRating if highValue, approve and sign.

We have

1. run CCALC on the related specifications of the problem,
2. to determine whether there exists a path leading to a state in which the same agent performs the tasks intRating, extRating if highValue, approve, and sign.
The path holds if the following property is satisfied

\[ p_{18} \land \text{productOK} \land \text{executed}(a, \text{intRating}) \land \\
(\neg \text{highValue} \lor \text{executed}(a, \text{extRating})) \land \\
\text{executed}(a, \text{approve}) \land \text{executed}(a, \text{sign}) \]  

(1)

CCALC found the trace, in few seconds.

The trace shows that (1) is satisfied, and from the trace it is possible to reconstruct the reasons for the violation, i.e. which agent has performed the violation performing which tasks.
Other contributions

- We experimented with other reasoning tasks:
  - Synthesis of the Permission Assignment: for a given number of agents, synthesize a security policy for the business process under given security requirements. In particular, we synthesize the permission assignment for an RBAC model for the LOP given some requirements for the user and the permission assignment.
  - Resource Allocation Plan: for a given number of agents and for all execution flows, find (if any) an assignment of activities to agents ensuring the completion of the business process according to the given security policy.
  - Comparison between C and SMV on this setting.
Outline

1. Introduction
2. Business Process under Authorization Constraints
3. Action Language $C$
4. Formal Modeling of Business Processes with $C$
5. Automatic Analysis
6. Conclusions
Conclusions

Action-based approach to the formal specification, and automatic analysis of business process under authorization constraints that

- is natural and concise, and
- allows to perform an automatic analysis.

Effectiveness shown by applying it against a business process taken from the banking domain.

Our experiments shows that the proposed approach can be used in a number of reasoning tasks.
More details on:

A. Armando, E. Giunchiglia, M. Maratea, S.E. Ponta
A state is an interpretation of the fluent symbols that satisfies $G \supset F$ for every static law in the action description $D$.

A transition is any triple $\langle s, a, s' \rangle$ where

- $s (s', \text{resp.})$ is the initial (resulting, resp.) state, and
- $a$ is an action;

A formula $F$ is caused in a transition $\langle s, a, s' \rangle$ if it is

- the head of a static law from $D$ such that $s'$ satisfies $G$, or
- the head of a dynamic law from $D$ such that $s'$ satisfies $G$ and $s \cup a$ satisfies $H$.

A transition $\langle s, a, s' \rangle$ is causally explained according to $D$ if its resulting state $s'$ is the only interpretation of the fluent symbols that satisfies all formulas caused in this transition.

The transition diagram represented by $D$ is the directed graph which has the states of $D$ as nodes and includes an edge from $s$ to $s'$ labeled $a$ for every transition $\langle s, a, s' \rangle$ that is causally explained according to $D$. 
### Abbreviations for Causal Laws

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expanded Form</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonexecutable $H$ if $F$.</td>
<td>caused ⊥ after $H \land F$.</td>
<td>$\neg F$ is a precondition of $H$</td>
</tr>
<tr>
<td>$H$ causes $F$ if $G$.</td>
<td>caused $F$ if $\top$ after $G \land H$.</td>
<td>$F$ is true after $H$ is executed in a state in which $G$ is true</td>
</tr>
<tr>
<td>$H$ may cause $F$ if $G$.</td>
<td>caused $F$ if $F$ after $H \land G$.</td>
<td>$F$ is true by default after $H$ is executed in a state in which $G$ is true</td>
</tr>
<tr>
<td>default $F$.</td>
<td>caused $F$ if $F$.</td>
<td>$F$ is true by default</td>
</tr>
<tr>
<td>inertial $F$.</td>
<td>caused $F$ if $F$ after $F$.</td>
<td>$F$ is an inertial fluent</td>
</tr>
<tr>
<td>constraint $F$.</td>
<td>caused ⊥ if $\neg F$.</td>
<td>$F$ must be true</td>
</tr>
</tbody>
</table>
Immediate Invocation of Entities in $C$

Actions executed as soon as their preconditions hold.

Example: invocation of the Credit Bureau

**nonexecutable** $\text{invoke}(a,r,\text{creditBureau})$
- if $\neg \text{granted}(a,r,\text{extRating}) \lor \neg p_6 \lor \text{lowrisk}$.

**caused** $\text{invoke}(a,r,\text{creditBureau})$
- if $\text{granted}(a,r,\text{extRating}) \land p_6 \land \neg \text{lowrisk}$.

$\text{invoke}(a,r,\text{creditBureau})$
- **causes** $\neg p_6 \land \text{invoked}(a,r,\text{creditBureau}) \land p_7$. 
The value of fluents can be determined by the interaction of other process features.

Example: the interruption of the process

\[
\text{caused } \text{interrupted if } p_7 \land \\
\text{invoked}(a_1, r, \text{creditBureau}) \land \text{accessed}(a_2).
\]

\[
\text{caused } \neg \text{interrupted if } p_7 \land \\
\text{invoked}(a, \text{director, creditBureau}).
\]

for all \( a_1 \neq a_2 \) and \( r \neq \text{director} \) and where \text{interrupted} is an inertial fluent
Events influencing the process whose causes often do not need to be modeled in the scope of the process.

Expressed by exogenous fluents (fluents that can arbitrarily change value in the transition (unless there is some other rule constraining their value).

Example: the non reversible access to the information exchanged with Credit Bureau by an unauthorized agent

\[
\text{caused } \text{accessed}(a) \text{ after } \text{accessed}(a).
\]

where \text{accessed}(a) is an exogenous fluent.
The Causal Calculator (CCALC)

Given a $C$ specification, an initial state and a desired state, CCALC automatically

- establishes if there exists an execution path starting for the initial state and reaching the desired state, and
- in case the execution path exists, CCALC returns a trace representing the execution path.
caused \(\neg \text{granted}(a, r, t)\) if \(\neg \text{ua}(a, r) \lor \neg \text{activated}(a, r)\lor\)

\[
\left( \bigwedge_{r_1 \in \mathcal{R}} \neg \text{pa}(r_1, t) \lor \neg \text{senior}(r, r_1) \right)
\]

where \(\mathcal{R}\) is the set of roles involved in the business process.
Some advantages of C vs NuSMV

- State-of-the-art model checkers—being geared to the analysis of hardware designs—require the system to be modeled as the composition of independent (yet interacting) sub-components.

- On the contrary, business processes and the associated security policy are best viewed as a collection of actions subject to a given workflow pattern and a set of independent access control rules.

- Moreover C provides modelers with the ability to specify the system incrementally: This is an important feature that is less supported by the specification languages of state-of-the-art model-checkers.