Verifying Soundness under Presence of Data

Nikola Trčka
Outline

• Classical soundness for *Workflow Nets*
• Consequences of the complete data abstraction
• Extended model: *Workflow Nets With Data*
  • *soundness*
  • data-flow errors
• Conclusions
Workflow nets

- **Standard model of control-flow in business processes**
- **Subclass of Petri nets**
  - one start place, one end place, all nodes on the way from start to end
- **Example – Loan approval process:**

![Loan approval process diagram]

- Start: Register request
- P1: Check client data
- P2: Approve
- P3: Inform client
- P4: Utilize loan
- P5: Sync
- P6: Send letter
- P7: Update client data
- End
Soundness for workflow nets

- Most widely used correctness criterion
- Captures errors like livelock or deadlock
- Two requirements:
  1. Proper termination possible from any state
  2. Every task can be executed

```
always completes but B is never enabled

does not complete if B is chosen
```
Soundness and data

- Classical soundness considers control-flow only.
- Data information is ignored even though data can influence routing.
- This can lead to
  - false positives (e.g. overlooked livelocks), and
  - false negatives (e.g. deadlocks falsely reported).
- Moreover, data-flow can itself be incorrect.
- Therefore,

  data information should be included in some form!
False positive – Overlooked livelock

- This workflow is sound (loop eventually exited).
- But now there is a **livelock**!
False negative – Deadlock falsely reported

- Soundness reports deadlock.
- But there is actually no deadlock!

produces document D

decision made w.r.t. D in exactly the same way
Data-flow issues not considered by soundness

- Data element can be missing for a task

- Data element can also be inconsistent, redundant, not deleted (on time), etc.
How to include data?

- **Proper way (theoretically):** Translate to a formalism that directly supports data, e.g. Colored Petri nets, and do the verification there.

- **Problem:**
  - Often large; analysis intractable.
  - Target framework might not have sufficient capabilities, or is not flexible enough.

- **Some abstraction from data is still necessary!**

- **Need for a complete, unifying and configurable analysis framework!**
Our approach

• **Model:**
  • *Workflow Nets with Data* model proposed
  • Data global (common store) and case related
  • Data elements included, but their values are abstracted away
  • Semantics given by (suitable) translation to regular workflow nets

• **Analysis:**
  • On the reachability graph (boundedness assumed)
  • By model checking (CTL*)
  • Pattern-based
Workflow nets with data

- Workflow nets where tasks:
  - can **read**, **write**, or **destroy** data elements.
  - have **guards** based on predicates (**P** and **not P**).

- **Weakness:** Predicate dependency not considered.
Loan approval workflow with data

- **Data elements:**
  - \(c\): client information
  - \(r\): loan request
  - \(d\): decision
  - \(a\): approval document
  - \(p\): final report
Translation to workflow net

- **Naive:** As workflow nets with data are just annotated workflow nets, simply take the underlying workflow net.

- **Problems:**
  - Guards are not taken into account
  - Cannot capture parallel execution of tasks

- **Better translation is needed.**
Proper translation

- Split task \( t \) into its start \( (t_s) \) and its end \( (t_e) \), connect them by new place \( (p_t) \).
- Add special place for every guard.
- Start task “guarded” buy the new guard place
- End task can change all depending guards in all possible ways

\[
\begin{align*}
\text{pred1}(c)_{\text{true}} & \quad \text{pred2}(b)_{\text{true}} & \quad \text{pred2}(b)_{\text{false}} \\
\end{align*}
\]
# Model checking soundness

<table>
<thead>
<tr>
<th>Property</th>
<th>Formalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-dead-transitions</td>
<td>$\text{EF exec}(t)$</td>
</tr>
<tr>
<td>Classical soundness</td>
<td>$\text{AGEF} \left[ \text{end} = 1 \land \bigwedge_{p \in P \setminus {\text{end}}} (p = 0) \right]$</td>
</tr>
<tr>
<td>Safe classical soundness</td>
<td>$\bigwedge_{p \in P} (p \leq 1) \land \text{AGEF} \left[ \text{end} = 1 \land \bigwedge_{p \in P \setminus {\text{end}}} (p = 0) \right]$</td>
</tr>
<tr>
<td>Easy soundness</td>
<td>$\text{EF} \left[ \text{end} = 1 \land \bigwedge_{p \in P \setminus {\text{end}}} (p = 0) \right]$</td>
</tr>
<tr>
<td>Relaxed soundness</td>
<td>$\text{E}[\text{exec}(t) \land \text{F} \left[ \text{end} = 1 \land \bigwedge_{p \in P \setminus {\text{end}}} (p = 0) \right]]$</td>
</tr>
<tr>
<td>Completion with fairness</td>
<td>$\text{A}\left[\forall t \in T \left[ \text{G} \left( \bigwedge_{p \in t} p \geq 1 \right) \Rightarrow \text{G} \text{F} \text{exec}(t) \right]\right]$</td>
</tr>
<tr>
<td>Lazy soundness</td>
<td>$\text{AG} \left[ \text{end} \leq 1 \land \text{exec}(t) \land \text{EF end} = 1 \right]$</td>
</tr>
</tbody>
</table>

- All properties either LTL or CTL
Model checking soundness

```java
1  Place en;
2  Nat x 5;
3  Place END p3;
4  Transition t2 t2;

6  public formula hasLessThanNTokens( Place p, Nat n ) =
7      |Checks whether the place p always has < n tokens|
8      p < n;
10 formula safenessCheck() =
11      |Safeness check|
12      forall p in Place:
13          hasLessThanNTokens(p, 2);
15 formula properTerm() =
16      |Proper termination check|
17          (END == 1)
19      \forall k in Place:
20          (k != END -> k == 0));

23 public formula classicalSoundness() =
24      |Checks classical soundness property|
25         _A _G _E _F properTerm();
27 public formula test(Place end) =
29      ()
31 public formula test1(Place end) =
33      _A _F end >= 0;
```
### Model checking data flow

- **Properties defined as data-flow anti-patterns:**

<table>
<thead>
<tr>
<th>Anti-pattern</th>
<th>Formalization</th>
</tr>
</thead>
</table>
| Missing Data                 | $E[(\neg w(d) \cup (r(d) \lor d(d))) \lor$   
|                              | $F[d(d) \land (\neg w(d) \cup (r(d) \lor d(d)))]$                         |
| Strongly Redundant Data      | $EF[w(d) \land AX [\neg r(d) \cup (\text{final} \lor (d(d) \land \neg r(d)))]$ |
| Weakly Redundant Data        | $EF[w(d) \land AX [\neg r(d) \cup (\text{final} \lor (d(d) \land \neg r(d)))]$ |
| Strongly Lost Data           | $EF[w(d) \land AX [\neg (r(d) \lor d(d)) \cup (w(d) \land \neg r(d))]$     |
| Weakly Lost Data             | $EF[w(d) \land AX [\neg (r(d) \lor d(d)) \cup (w(d) \land \neg r(d))]$     |
| Inconsistent Data            | $\forall t \in T: d \in \text{change}(t)$       
|                              | $EF[(\text{exec}(t) \land \exists_{t' \neq t: d \in \text{use}(t')} \text{exec}(t')) \lor pt \geq 2]$ |
| Never Destroyed             | $EF[w(d) \land AX [\neg (d(d) \lor w(d)) \cup \text{final}]$               |
| Twice Destroyed             | $EF[d(d) \land AX [\neg w(d) \cup (d(d) \land \neg w(d))]$               |
| Not Deleted on Time          | $\forall t \in T: d \in \text{data}(\text{grd}(t)) \setminus d(t)$        |
|                              | $AG[\text{exec}(t) \Rightarrow \text{exec}(t) \cup G(\neg r(d))]$     |
Conclusions

• We defined a framework for verifying workflow soundness with inclusion of data flow.
• The framework is based on workflow nets with data and model-checking.
• Data-flow properties are also considered
• Set of (anti-)patterns defined describing common properties; all notions from the literature covered.
• The approach powerful, flexible, configurable, and extendable.

• **Current work:**
  • CTL* model-checker implementation in ProM
  • Property language with ready packages for common properties (patterns)
  • Extensions to Grid workflows