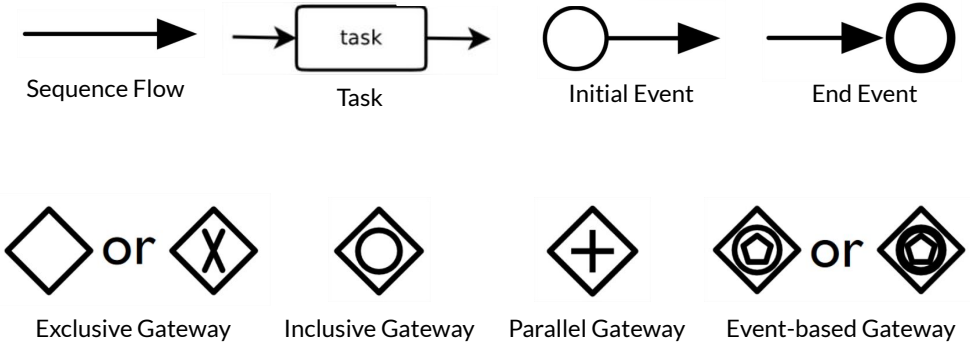
By Gwen SALAUN & **Quentin NIVON**

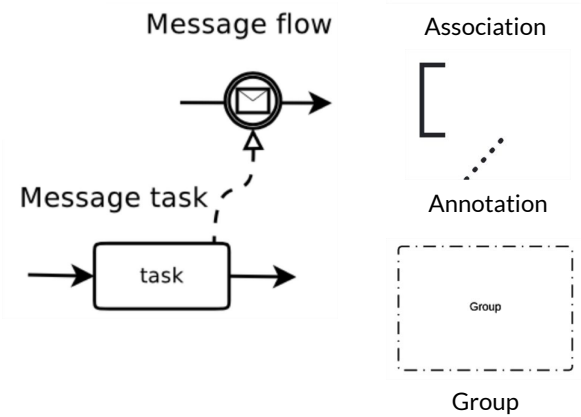
Debugging of BPMN processes using coloration techniques

Introduction

What is BPMN?

- A **workflow-based notation** created in 2004 by the Business Process Management Initiative (BPMI) and the Object Management Group (OMG).
- It aims at **representing business processes** in a way that is **understandable for both experienced and novice users.**
- An **ISO/IEC standard** since version 2.0 in 2013.
- A rich syntax:





Introduction

Context

- Companies are making use of the BPMN notation to represent their business processes.
- Some of them are interested in **performing behavioral verifications** of their BPMN processes.

Existing solutions

- Nowadays, companies are able to **verify the validity of temporal logic properties** over their BPMN processes.
- To do so, they transform their BPMN processes into specifications understandable by a **model checker** which will verify the property.
- In the end, they get one or several **counterexamples that violate the property**. Nonetheless, they **are not expressed in BPMN notation.**

Introduction

Motivations

- **Expressing all the counterexamples in BPMN notation** to make it understandable for the initial user.
- Remaining **as syntactically close as possible** to the original BPMN process.
- **Being as minimal as possible, in terms of number of nodes**, in the BPMN process returned to the user.

Plan

Preliminaries: BPMN sub-syntax

In this work, we use a **restrained subset of the entire BPMN syntax**.

The subset syntax used contains (or plan to contain):

- ✓ Gateways (exclusive, parallel, inclusive)
- ✓ Sequence Flows
- ✓ Tasks
- ✓ Events (start, end)

Occur in more than 90% of BPMN processes!
[Krishna, Poizat, Salaün – SCP 19]

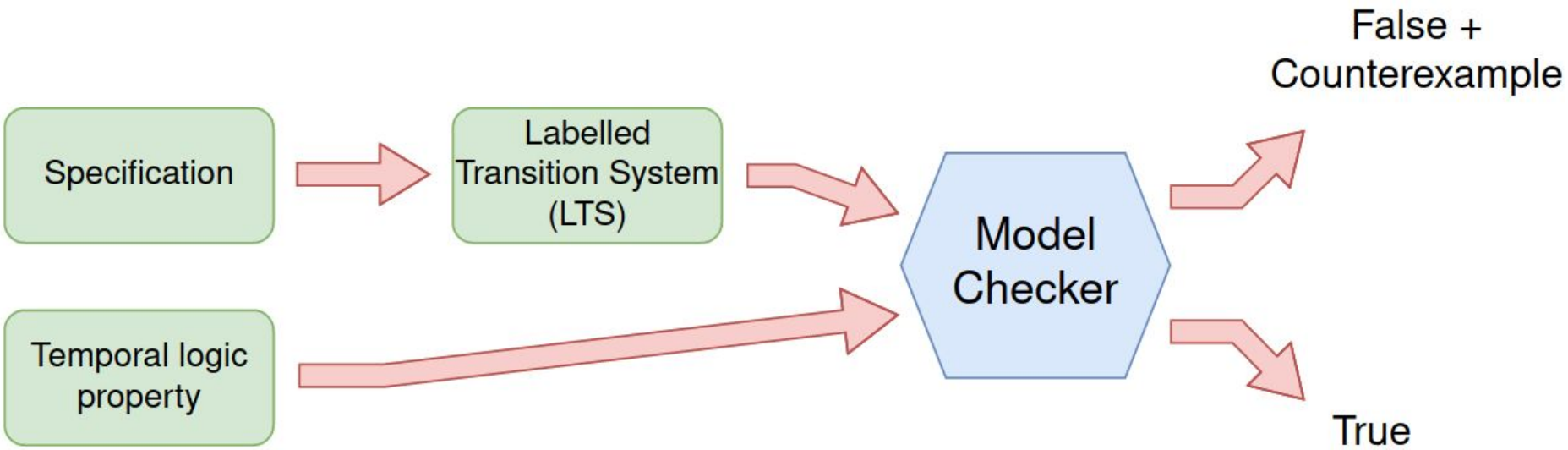
But does not contain (or plan to contain):

- ✗ Intermediary events
- ✗ Message-based gateways
- ✗ Message flows
- ✗ Associations
- ✗ Artifacts, data-objects

Occur in less than 10% of BPMN processes.

Preliminaries: Model Checking

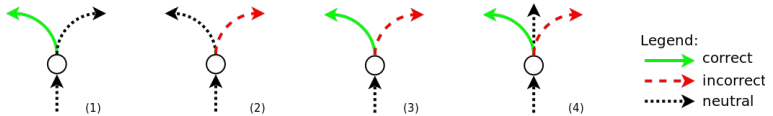
The current work makes use of already existing techniques, such as model checking:

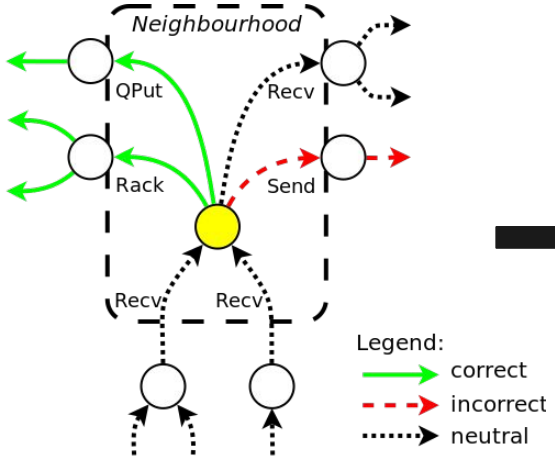


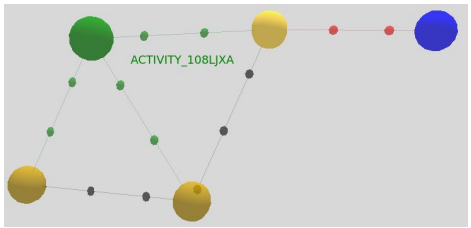
Preliminaries: Colored LTS

Or Colored LTS (CLTS):

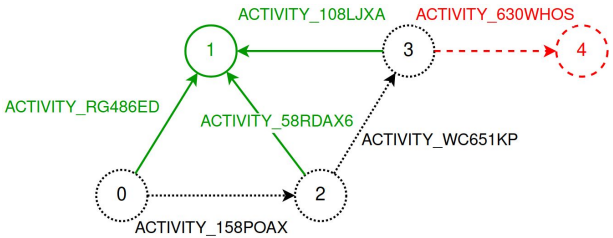
- Key idea 1: transitions categorisation (3 types)
  - correct transitions
  - incorrect transitions
  - neutral transitions
- Key idea 2: identification of faulty states
  - choices between transitions leading to **a correct or an incorrect part** of the model
  - Four kinds of faulty states:







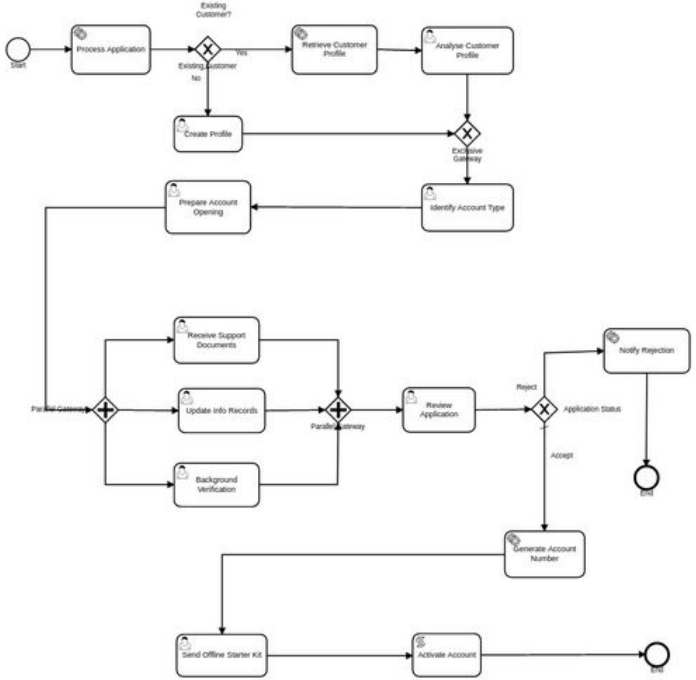
CLTS output from the CLEAR tool

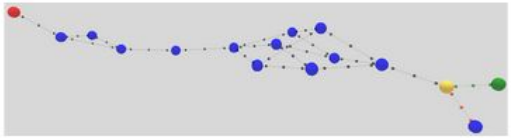


Clearer representation of the CLTS output

Preliminaries: Full process

The whole counterexample generation process can be modeled as follows:





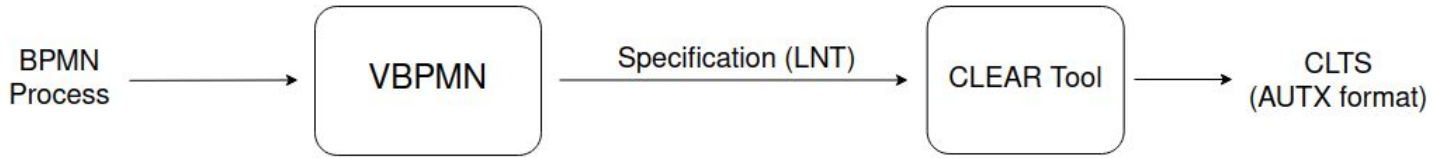
```
module AccountOpeningV5(bpmntypes) with "get" is

process init [begin:any, outf:any] is
var ident: ID in begin ; outf (?ident of ID) end var
end process

process final [incf:any, finish:any] is
var ident: ID in
loop
incf (?ident of ID); finish
end loop
end var
end process

process flow [begin:any, finish:any] (ident: ID) is
loop begin (!ident) ; finish (!ident) end loop
end process
--------
```

Formula

```
[true*.NOTIFYREJECTION.true*] false
```

Temporal Logic Property (MCL)

```
des (0, 21, 16)
(0, "PROCESSAPPLICATION":BLACK, 1)
(1, "CREATEPROFILE":BLACK, 2)
(1, "RETRIEVEPROFILE":BLACK, 3)
(2, "IDENTIFYACCOUNT":BLACK, 4)
(3, "ANALYSEPROFILE":BLACK, 2)
(4, "PREPAREACCOPENING":BLACK, 5)
(5, "BACKGROUNDVERIFICATION":BLACK, 6)
(5, "RECEIVESUPPORTDOCUMENTS":BLACK, 7)
(5, "UPDATERECORDS":BLACK, 8)
(6, "RECEIVESUPPORTDOCUMENTS":BLACK, 9)
(6, "UPDATERECORDS":BLACK, 10)
(7, "BACKGROUNDVERIFICATION":BLACK, 9)
(7, "UPDATERECORDS":BLACK, 11)
(8, "BACKGROUNDVERIFICATION":BLACK, 10)
(8, "RECEIVESUPPORTDOCUMENTS":BLACK, 11)
(9, "UPDATERECORDS":BLACK, 12)
(10, "RECEIVESUPPORTDOCUMENTS":BLACK, 12)
(11, "BACKGROUNDVERIFICATION":BLACK, 12)
(12, "REVIEWAPPLICATION":BLACK, 13)
(13:N:GR, "GENERATEACCOUNTNUMBER":GREEN, -1)
(13:N:GR, "NOTIFYREJECTION":RED, 15)
```



Plan

1. Preliminaries
   - 1.1. BPMN sub-syntax
   - 1.2. Model checking & Colored LTS
   - 1.3. Full process

2. **Solving approach**
   - **2.1. Global representation**
   - **2.2. Matching analysis**
   - **2.3. Unfolding only**
   - **2.4. Unfolding + Folding**

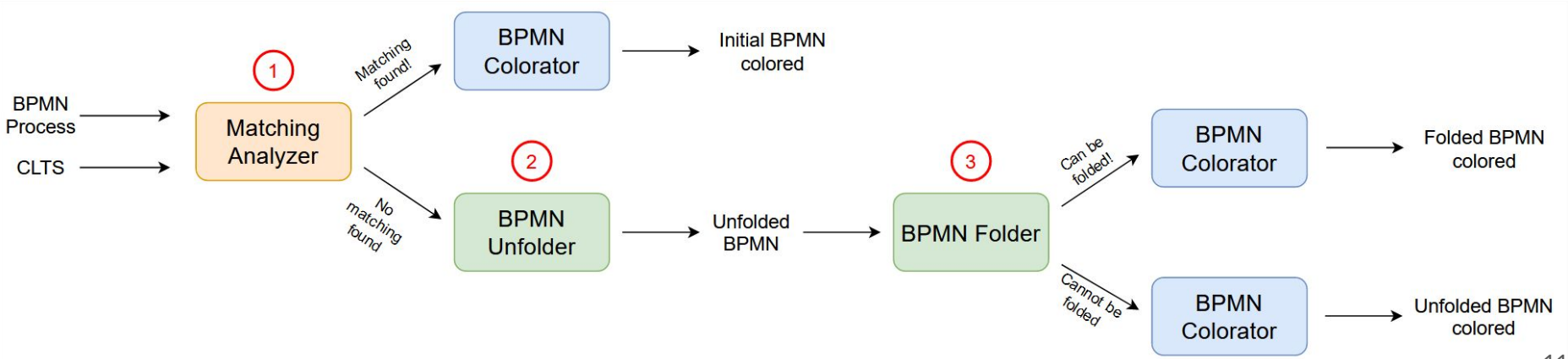
3. Implementation
   - 3.1. Generalities
   - 3.2. Empirical study
   - 3.3. Performance study

4. Conclusion

Solving approach: Global representation

The solving approach is a **3-step approach**:

- ➢ First, we try to **find a matching** between the CLTS and the BPMN process. (1)
- ➢ If no matching can be found, then the initial BPMN process is not directly colorable. In such case, we generate a new BPMN process during the **unfolding phase.** (2)
- ➢ Finally, we try to **fold** this BPMN process in order **to reduce its number of nodes.** (3)



Solving approach: Matching analysis – Idea

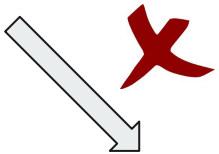
Main idea

The purpose of this step is to verify if the initial BPMN process is **directly colorable regarding the property.**

Algorithmically

- We detect whether all the **transitions** of the CLTS having the **same label** are **colored identically.**
- If so, we try to **associate each faulty state** of the CLTS **to a unique node** of the BPMN process.

A matching exists!
Then the initial BPMN process is colored directly and returned to the user

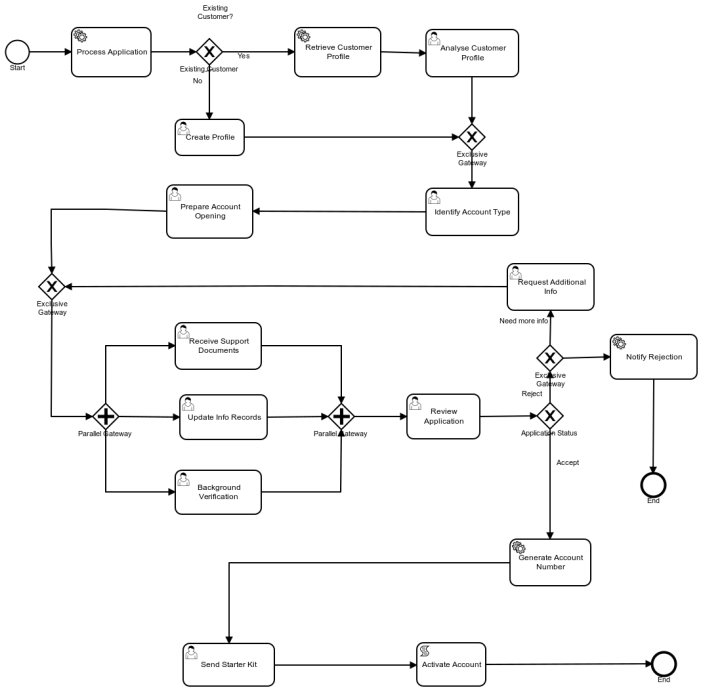


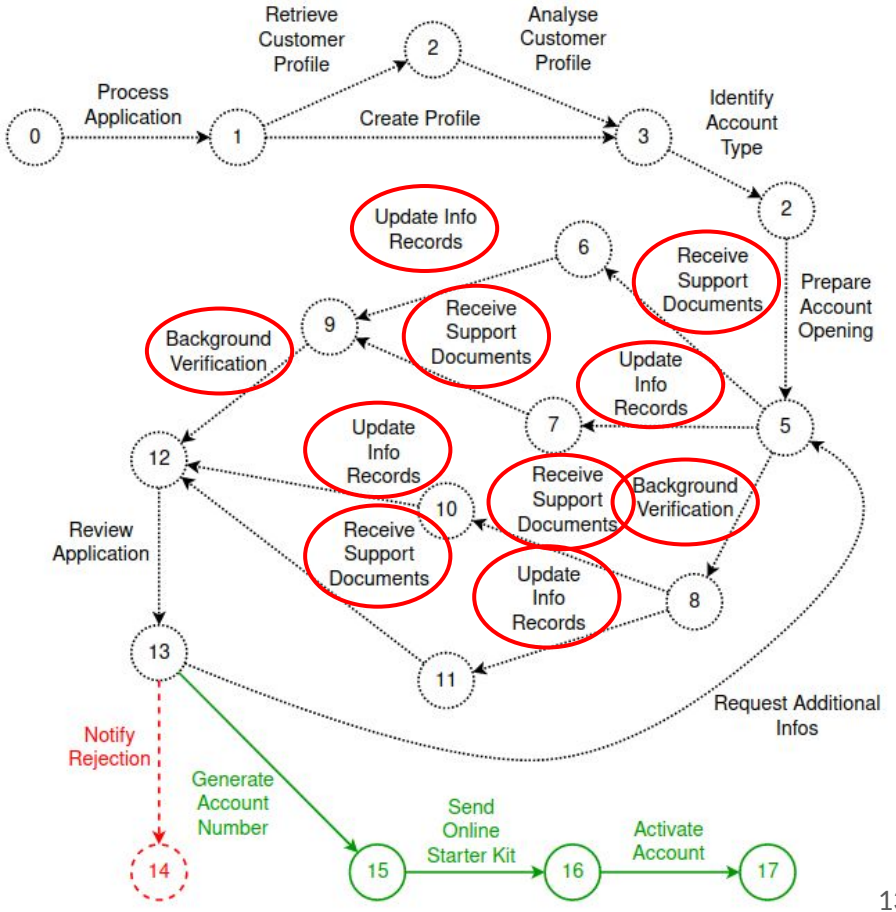
No matching exists.
Then we build a new BPMN process from the CLTS and try to minimize it

Solving approach: Matching analysis – Examples

Example 1: A perfect matching is found

Step 1: Generation of the CLTS





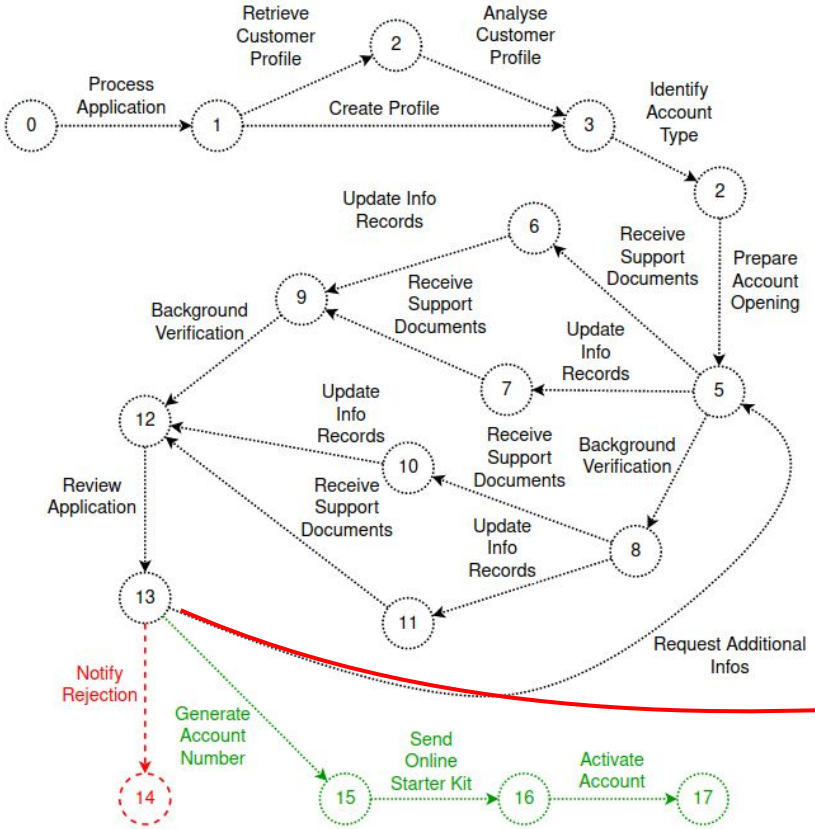
Formula

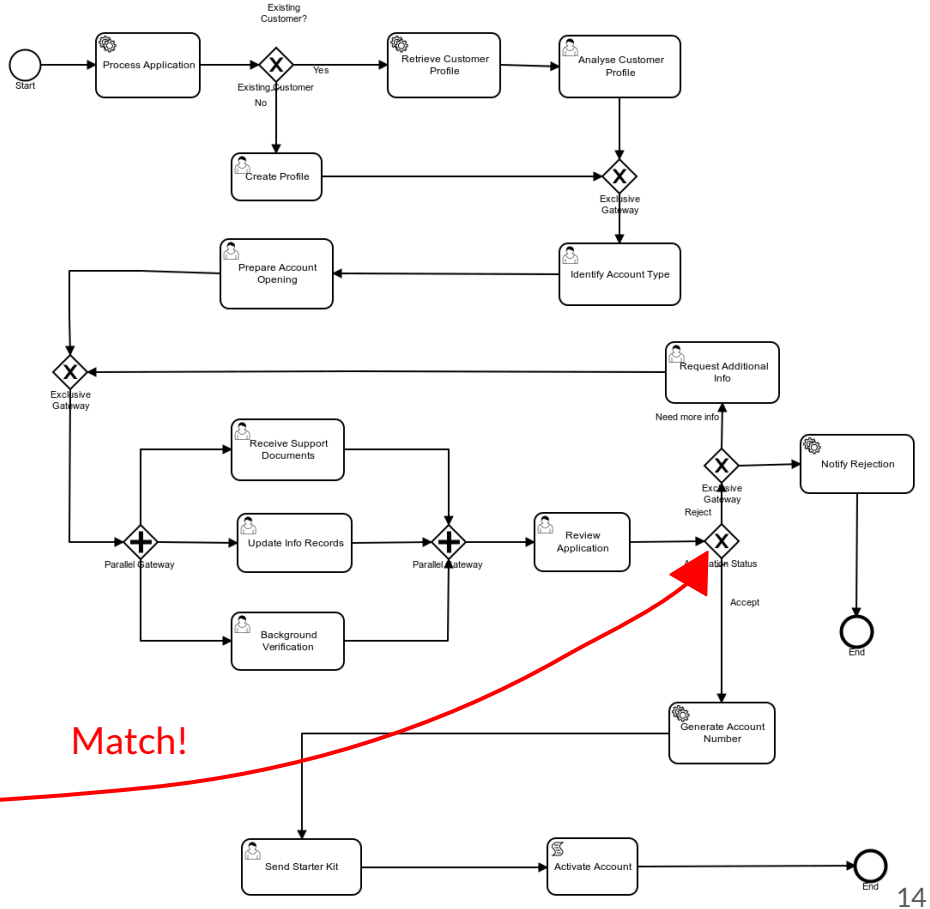
[true*.NOTIFYREJECTION.true*] false

Solving approach: Matching analysis – Examples

Example 1: A perfect matching is found

Step 2: Matching computation



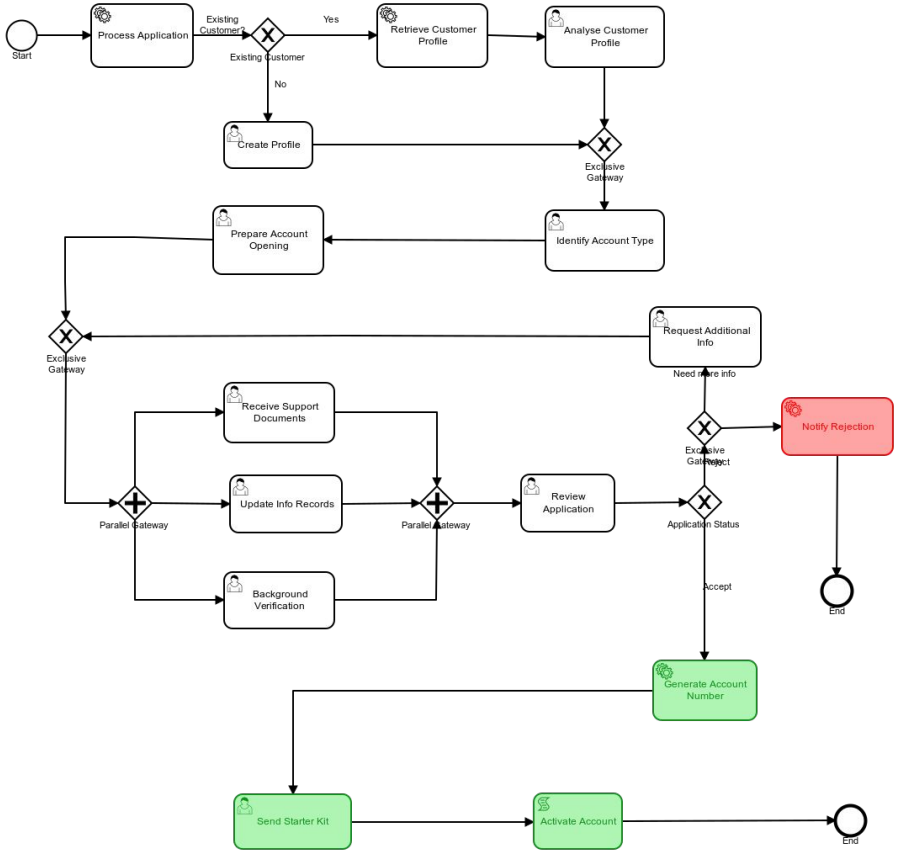


Match!

Solving approach: Matching analysis – Examples

Example 1: A perfect matching is found

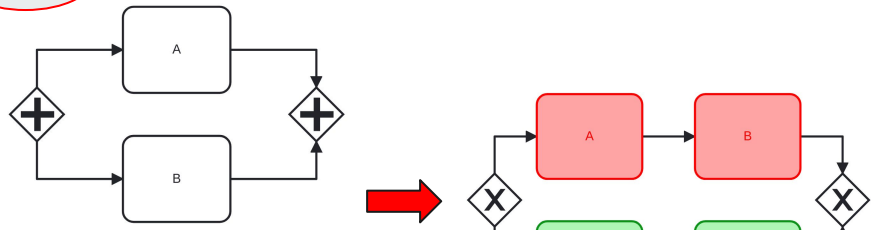
Step 3: Coloration is performed: Starting from each first red or green transition, we color each reachable BPMN task.



Solving approach: Unfolding

We transform the CLTS into a semantically equivalent BPMN process.
As the CLTS is colored, the generated BPMN process is colorable regarding the property.

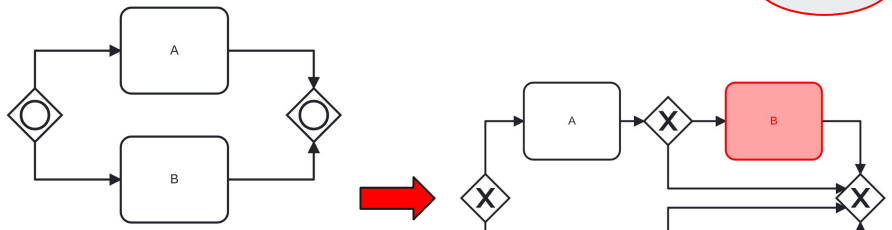
Case 1



Formula

[true*.A.true*.B.true*] false

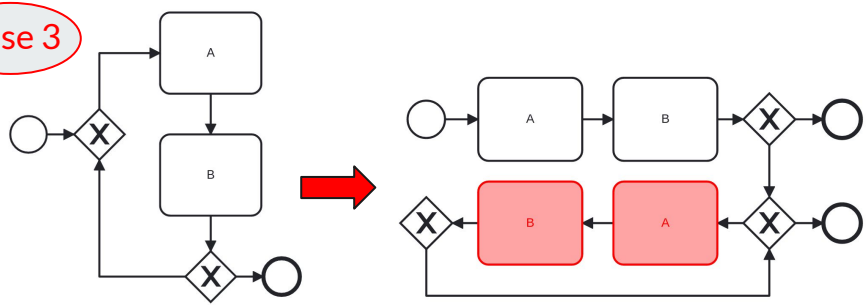
Case 2



Formula

[true*.A.true*.B.true*] false

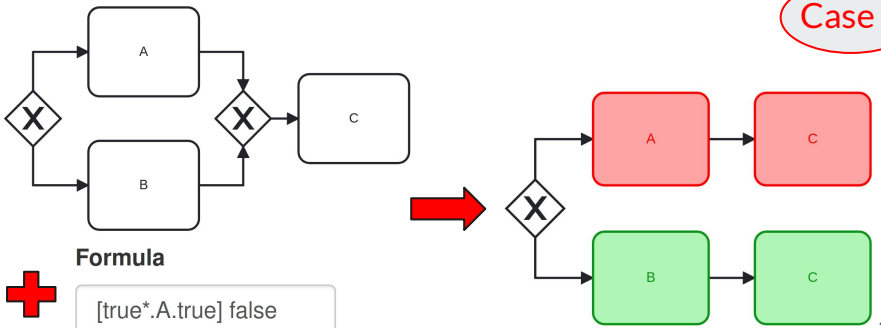
Case 3



Formula

[true*.A.true*.B.true*.A.true*.B.true*] false

Case 4



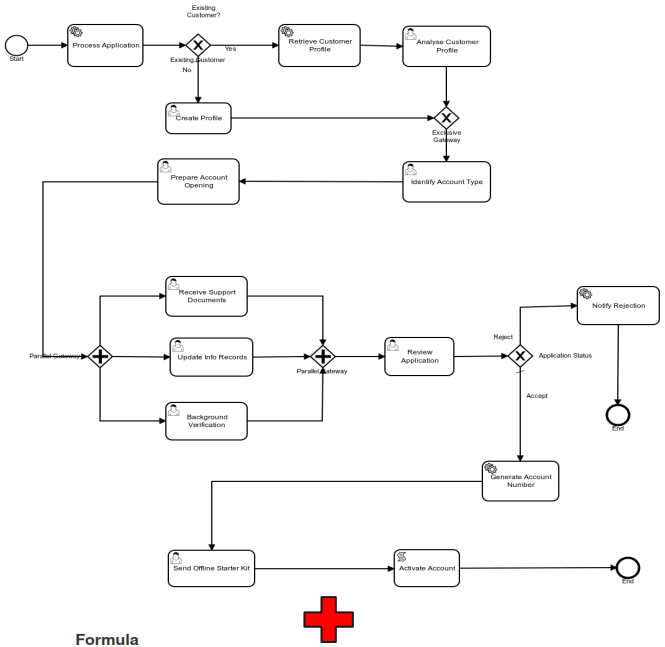
Formula

[true*.A.true] false

Solving approach: Unfolding

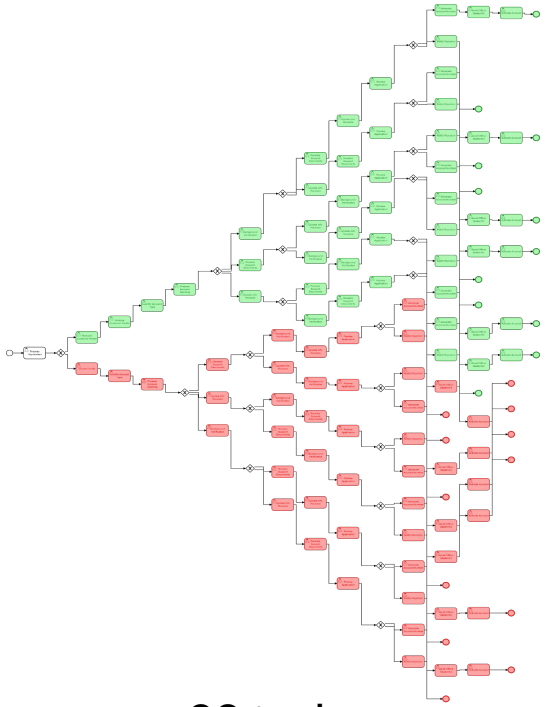
Drawback

In some cases, the generated BPMN process can be very large, which may limit the understanding of the problem for the user.



14 tasks

Unfolding



98 tasks

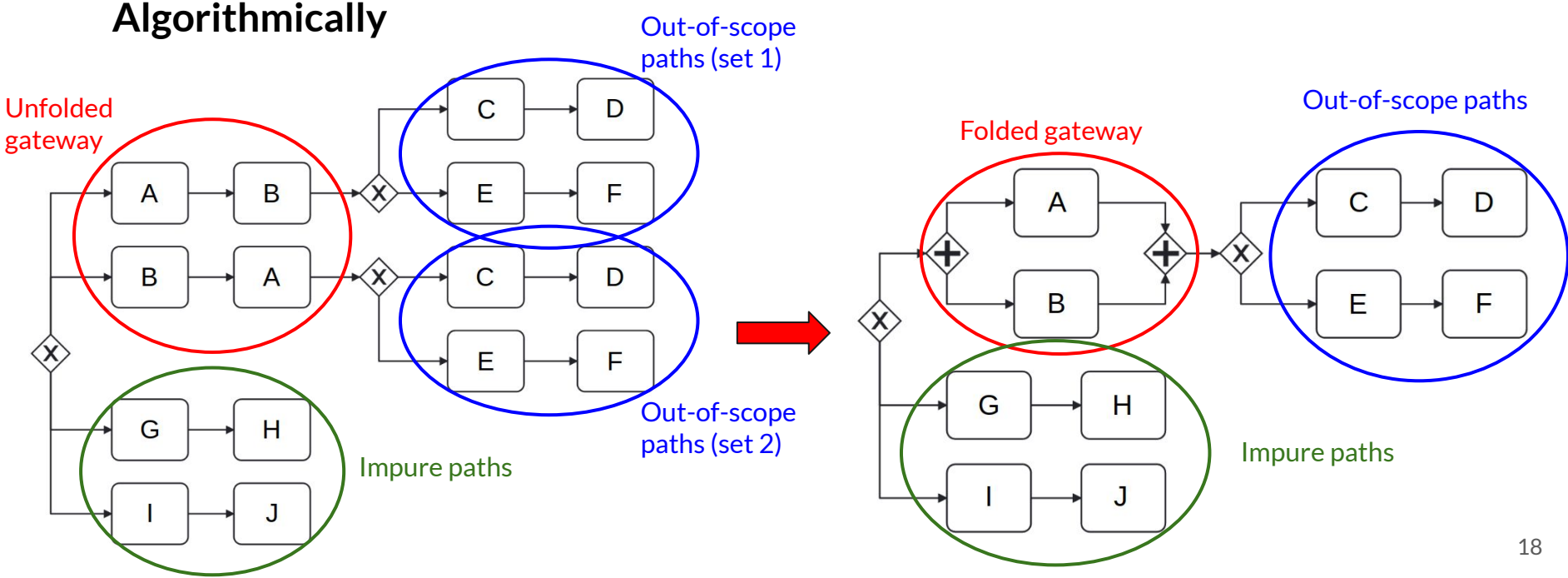
Solving approach: Unfolding + Folding

Main idea

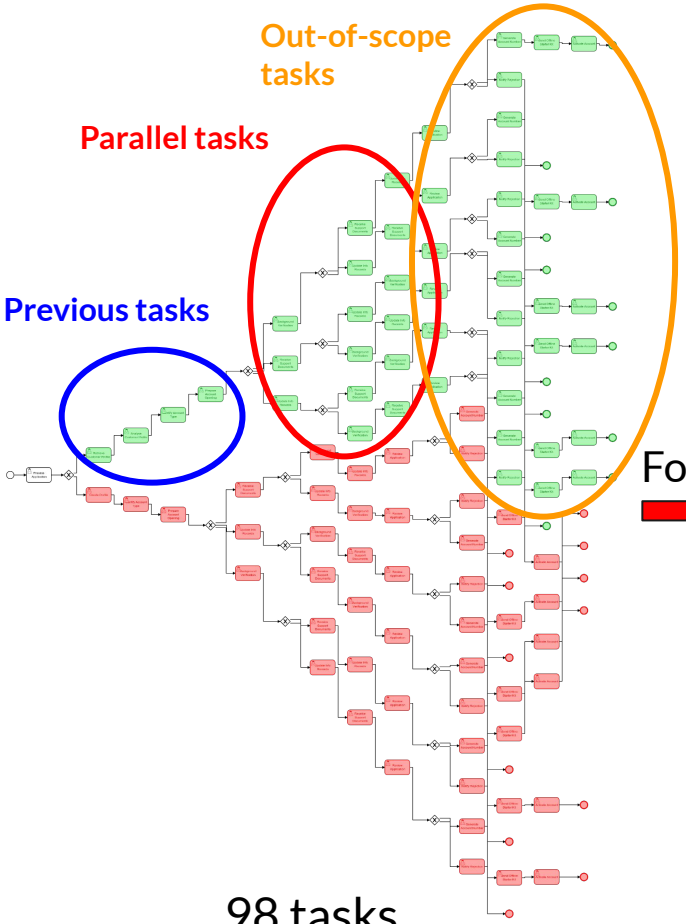
In this phase, we try to **reduce the size** – in terms of number of nodes – of the unfolded BPMN process **to make it more understandable** for the user.

Our approach focuses on the **detection of unfolded parallel gateways** in order to **replace them** in the unfolded graph **by their folded version.**

Algorithmically

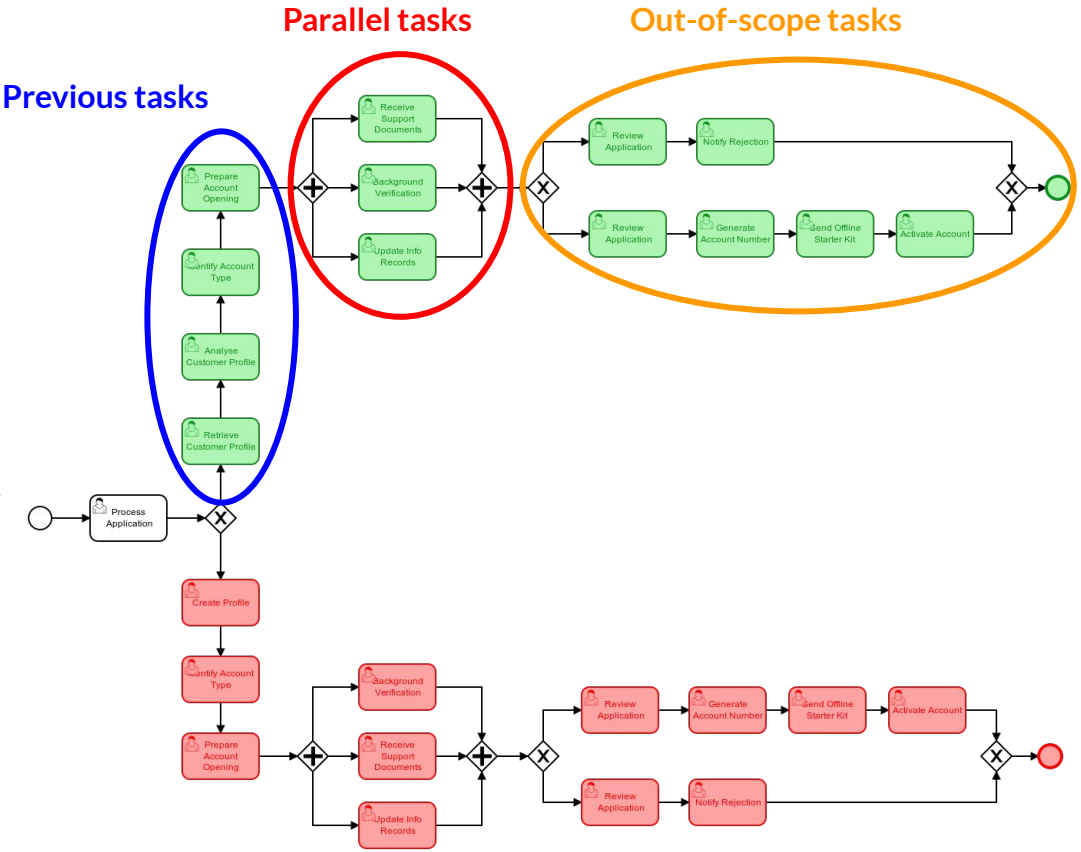


Solving approach: Unfolding + Folding – Example



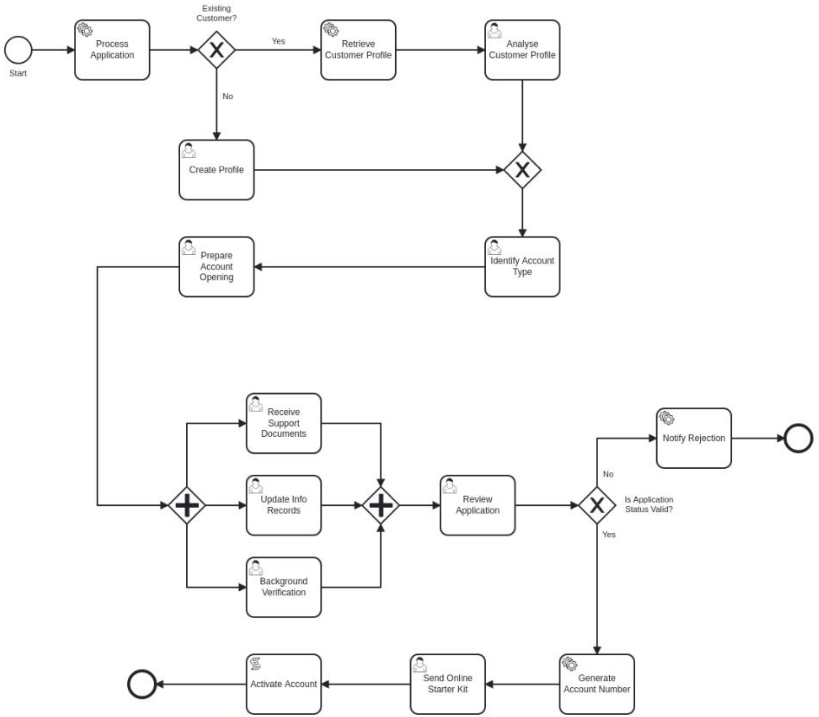
98 tasks

Folding

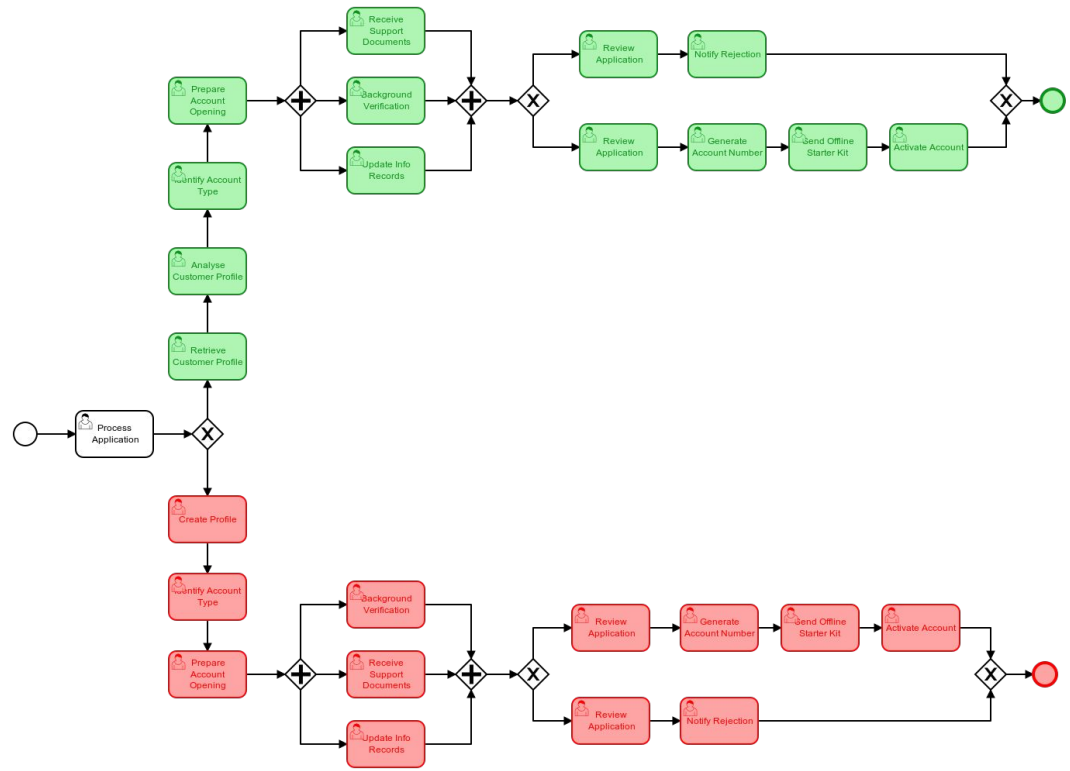


26 tasks

Solving approach: Unfolding + Folding – Example



Initial BPMN process (14 tasks)



Returned BPMN process (26 tasks)

Plan

1. Preliminaries
   - 1.1. BPMN sub-syntax
   - 1.2. Model checking & Colored LTS
   - 1.3. Full process
2. Solving approach
   - 2.1. Global representation
   - 2.2. Matching analysis
   - 2.3. Unfolding only
   - 2.4. Unfolding + Folding
3. **Implementation**
   - **3.1. Generalities**
   - **3.2. Empirical study**
   - **3.3. Performance study**
4. Conclusion

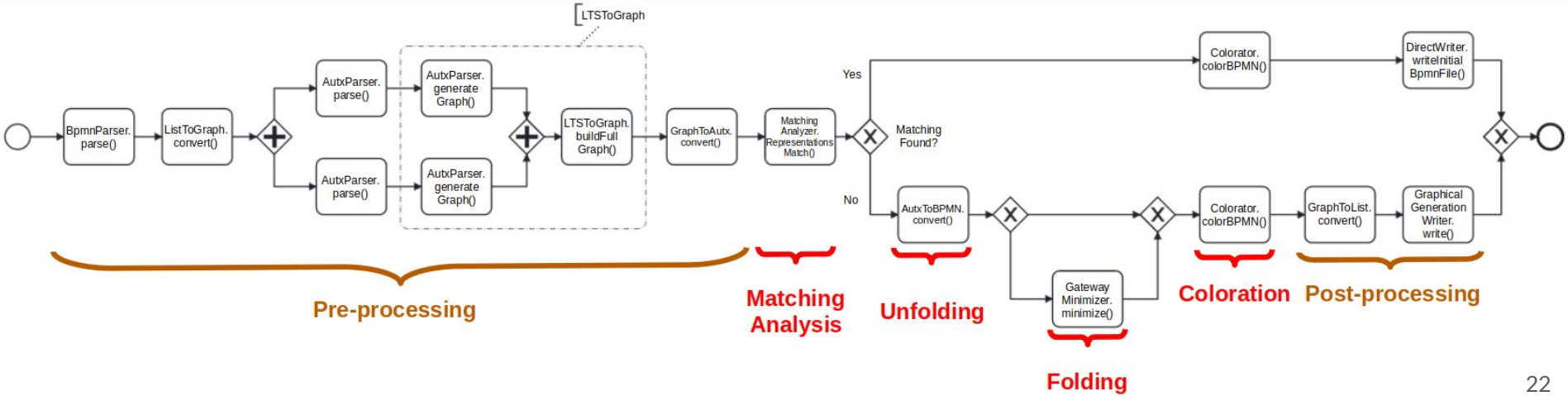
Implementation – Generalities

Prototype

The approach detailed in this presentation has been **fully implemented and tested** on more than **100 BPMN processes**. It consists of approximately **10,000 lines of Java code**.

Steps

In this presentation, we did not detail all the steps of the approach, which contain **a lot of processing phases**. These steps are detailed in the below figure.



Implementation – Empirical study

BPMN Example	Directly Colorable	Foldable
1. Account Opening	X	N/A
2. Publication Process	X	N/A
3. Credit Offer	X	N/A
4. Vacations Booking		X (12/20)
5. Account Opening		X (8/8)
6. Plane Entry		X (3/3)
7. Mortgage Application		X (2/2)
8. Denoising Process		
9. Buying Process		
10. Login Process		
11. Support Ticket		
12. Steel Transformation		
13. Business Process		
14. Job Hiring		

3/14 = 21%

4/14 = 29%

Overall, the implemented part of the approach proposes the same BPMN process or a syntactically close one to the user in **50% of cases.**

Implementation – Performance study

BPMN Example	Number of BPMN elements	Number of elements in CLEAR	Number of elements without folding	Number of elements with folding	Time taken by VBPMN (s)	Time taken by CLEAR (s)	Time taken by prototype without folding (s)	Global time without folding (s)	Time taken by prototype with folding (s)	Global time with folding (s)
1. Vacations Booking	13								3.436	3.436
2. Account Opening	22								1.209	1.209
3. Publication Process	20								1.351	1.351
4. Steel Transformation	42								1.475	1.478
5. Plane Entry	27								2.235	2.235
6. Mortgage Application	15								1.974	1.974
7. Denoising	16								1.756	1.756
8. Credit Offer	15								1.004	1.004
9. Buying Process	29								1.540	1.540
10. Business Process	15								1.325	1.325
11. Job Hiring	29								1.798	1.798
12. Login Process	17								1.204	1.204
13. Support Ticket	22								1.431	1.431
14. Generated Large 1	62	140	51	62	15.98	0.164	1.254	17.40	1.881	18.03
15. Generated Large 2	126	284	100	126	32.04	2.080	1.262	33.38	1.376	33.50
16. Generated Large 3	254	572	197	254	165.4	6.461	1.790	173.6	2.440	174.2
17. Generated Large 4	510	1148	390	510	902.3	124.3	3.024	**1030**	3.643	**1030**
18. Generated Parallel 1	10	86	27	37	12.21	0.059	0.999	13.27	1.034	13.30
21. Generated Parallel 2	19	529	203	11582	13.07	0.086	1.807	14.96	19.76	32.92
22. Generated Parallel 3	20	606	233	23165	13.15	0.089	2.099	15.34	82.68	95.92
23. Generated Parallel 4	21	683	264	43614	13.23	0.088	2.678	16.00	466.8	480.1
24. Generated Parallel 5	22	770	298	87229	13.61	0.082	4.088	17.78	3948	**3961**
25. Generated Parallel 6	23	857	333	165307	17.81	0.122	6.271	24.20	10100	**10117**

Worst-case execution time for real-world processes is **3s!**

For a BPMN process containing 254 nodes, the total execution time approximates 3m, but **only 2s are taken by the prototype!**

For a BPMN process containing **43k nodes**, the prototype **starts showing its limits** with an execution time of 8m.

Plan

1. Preliminaries
   - 1.1. BPMN sub-syntax
   - 1.2. Model checking & Colored LTS
   - 1.3. Full process
2. Solving approach
   - 2.1. Global representation
   - 2.2. Matching analysis
   - 2.3. Unfolding only
   - 2.4. Unfolding + Folding
3. Implementation
   - 3.1. Generalities
   - 3.2. Empirical study
   - 3.3. Performance study
4. **Conclusion**

Conclusion

To conclude, we proposed in our approach a way of **expressing all the counterexamples** of a temporal logic property **in BPMN notation**, while providing the user a BPMN process that is **as syntactically close as possible to the initial one**.

We also provided an **automated tool** implementing **all the proposed solutions**, for which we **assessed the scalability** through a performance study.

Short-term improvements:

- Handle inclusive gateways

Mid-term improvements:

- Support liveness properties

Questions