Declarative Process Specifications:

Reasoning and Mining Temporal Formulae

Francesco Chiariello

ANITI, IRIT, University of Toulouse
francesco.chiariello@irit.fr

Table of Contents

Table of Contents

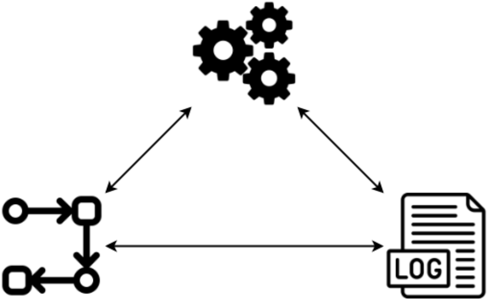
Overview

- Introduction to **Temporal Logics** and **Process Mining** (PM)
- **Declarative Process Mining** = Temporal Logics + PM
  - Linear Temporal Logics (LTL) for Declarative Specifications
- Tasks:
  - Log Generation and Conformance Checking,
  - Trace Alignment,
  - Process Discovery,
  - Process Repair.

Table of Contents

Process Mining in a nutshell

Process mining analyzes **event logs** to discover, monitor, and optimize **processes** by deriving or enhancing **process models**.



Process Models and Event Logs

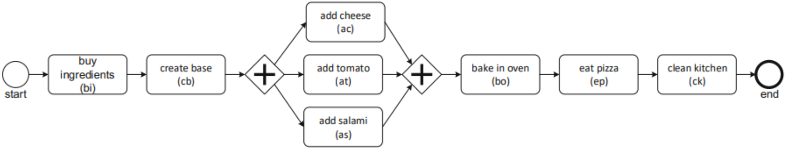


Figure: Pizza Process

- A **process trace** is a sequence of activities from start to end.
- An **event log** is a collection of traces.

Event Log Example

Case	Activity	Timestamp	Resource	Customer
pizza-56	buy ingredients (bi)	18:10	Stefano	Valentina
pizza-57	buy ingredients (bi)	18:12	Stefano	Giulia
pizza-57	create base (cb)	18:16	Mario	Giulia
pizza-56	create base (cb)	18:19	Mario	Valentina
pizza-57	add tomato (at)	18:21	Mario	Giulia
pizza-57	add cheese (ac)	18:27	Mario	Giulia
pizza-56	add cheese (ac)	18:34	Mario	Valentina
pizza-56	add tomato (at)	18:44	Mario	Valentina
pizza-56	add salami (as)	18:45	Mario	Valentina
pizza-56	bake in oven (bo)	18:48	Stefano	Valentina
pizza-57	add salami (as)	18:50	Mario	Giulia

Process Mining Tasks

- **Conformance Checking:** Verify whether the traces of a logs are compliant with a process model.
- **Log Generation:** Generate event logs from a process model (used for testing and analysis).
- **Trace Alignment:** Modify traces to make them compliant with the model.
- **Process Discovery:** Generate process model from event logs.
- **Process Model Repair:** Modify a process model to better match observed event logs.

LTL_p

- Linear Temporal Logic over process traces (LTL_p) is a formalism for specifying temporal properties of process.
- It allows reasoning about sequences of activities.
- LTL_p uses temporal operators like `Next` (**X**), `Until` (**U**), `Eventually` (**F**), and `Globally` (**G**).

LTL_p: Syntax

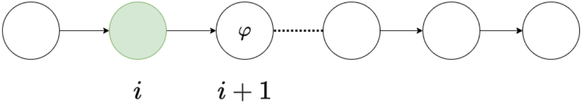
- Given a set Σ of propositional symbols, the syntax is defined by the following grammar

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

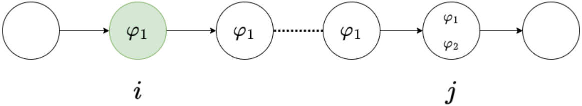
  with $a \in \Sigma$.
- Common abbreviations used are:
  - $true$, $\rightarrow$, $\vee$
  - $\mathbf{F}\varphi \equiv true\mathbf{U}\varphi$
  - $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$
  - $\tilde{\mathbf{X}}\varphi \equiv \neg\mathbf{X}\neg\varphi$
  - $\varphi_1\mathbf{W}\varphi_2 \equiv \varphi_1\mathbf{U}\varphi_2 \vee \mathbf{G}\varphi_1$

Temporal Operators

$\pi, i \models X\varphi$

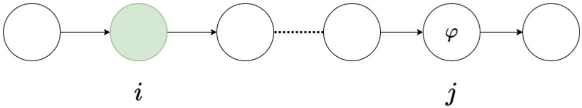


$\pi, i \models \varphi_1 U \varphi_2$

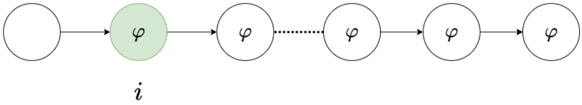


Temporal Operators (cont'd)

$\pi, i \models F\varphi$



$\pi, i \models G\varphi$



LTL_p: Semantics

- Given a formula φ, a trace $\pi = \pi_1, \pi_2, \ldots, \pi_{len(\pi)} \in \Sigma^+$, and a time instant i, with $1 \leq i \leq len(\pi)$, the semantics is defined as follows:
  - $\pi, i \models a$ iff $a = \pi_i$,
  - $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \neg\varphi$,
  - $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$,
  - $\pi, i \models \mathbf{X}\varphi$ iff $i < len(\pi)$ and $\pi, i+1 \models \varphi$,
  - $\pi, i \models \varphi_1 \mathbf{U} \varphi_2$ iff $\pi, j \models \varphi_2$ for some j, with $i \leq j \leq len(\pi)$, and $\pi, k \models \varphi_1$ for all $k = i, \ldots, j-1$.

  We write $\pi \models \varphi$, and we say that π satisfies φ, if $\pi, 1 \models \varphi$.

LTL_p and Finite-state Automata

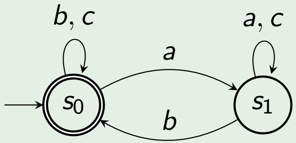
Theorem

Given an LTL_p formula φ over Σ there exists a Finite-state Automaton A_φ over Σ such that A_φ accepts exactly the traces that satisfy φ.

Example

Formula:
$\mathbf{G}(a \rightarrow \mathbf{F} b)$

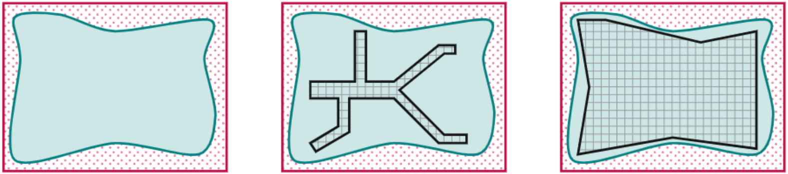
Explanation:
For all time instants, if a occurs, b must eventually follow.



Declarative Process Specifications

- Traditional approach:
  - procedural model for the process,
  - temporal logic for traces properties.
- **Declarative approach**: use temporal formulae as model:
  - A model is a set of temporal formulae,
  - Admissible traces are the ones satisfying all the formulae.
  - A declarative model specifies *what* properties a solution should obey, rather than *how* traces should be routed to satisfy the constraints.
- **Declarative Process Mining** considers declarative models.

Pros and Cons of Declarative Models



(a) A process (b) Imperative model (c) Declarative specification

- **Pros:**
  - Less prone to excluding traces that should be allowed.
- **Cons:**
  - Difficult to model processes using temporal formulae.

DECLARE Templates

Template	$\mathbf{LTL}_p$
$Init(a)$	a
$Exactly2(a)$	$\neg a\mathbf{U}(a \wedge \mathbf{X}(\neg a\mathbf{U}(a \wedge \neg\mathbf{X}(\mathbf{F}a))))$
$Response(a, b)$	$\mathbf{G}(a \rightarrow \mathbf{F}b)$
$RespondedExistence(a, b)$	$\mathbf{F}a \rightarrow \mathbf{F}b$
$AlternateResponse(a, b)$	$\mathbf{G}(a \rightarrow \mathbf{X}(\neg a\mathbf{U}b))$
$Precedence(a, b)$	$(\neg b)\mathbf{W}a$
$ChainPrecedence(a, b)$	$\mathbf{G}(\mathbf{X}b \rightarrow a) \wedge \neg b$
$Choice(a, b)$	$\mathbf{F}(a \vee b)$
$ExclusiveChoice(a, b)$	$\mathbf{F}(a \vee b) \wedge \neg(\mathbf{F}a \wedge \mathbf{F}b)$
$CoExistence(a, b)$	$\mathbf{F}a \leftrightarrow \mathbf{F}b$

Table: Some DECLARE templates and corresponding LTL_p formula.

Table of Contents

Logical Encoding

We want to show how to encode logically

- process traces,
- temporal formulae,
- the corresponding semantics.

Trace Encoding

The trace *aab* can be encoded with the following facts:

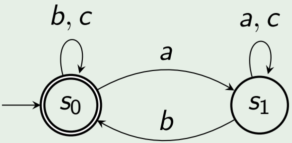
```
trace(a,1).
trace(a,2).
trace(b,3).
```

Logical Encoding (cont'd)

Formula

The formula *Response*$(a, b) = \mathbf{G}(a \rightarrow \mathbf{F}b)$ can be encoded by its corresponding automaton:

```
init(s_0).
acc(s_0).
trans(s_0,a,s_1).
trans(s_1,b,s_0).
trans(s_0,b,s_0).
trans(s_0,c,s_0).
trans(s_1,a,s_1).
trans(s_1,c,s_1).
```



Logical Encoding (cont'd)

Reading a trace

```
cur_state(S,0) :- init(S).
cur_state(S2,T) :- cur_state(S1,T-1),
                   trace(A,T),
                   trans(S1,A,S2).
```

Checking for satisfaction

```
last(T) :- trace(_,T), not trace(_,T+1).
sat  :- cur_state(S,T), last(T), accepting(S).
```

Generate a trace

```
time(1..t).
{trace(A,T}:activity(A)} = 1 :- time(T).
```

Log Generation and Conformance Checking

- **Conformance Checking**
  - **Problem:** Given a process trace π and an LTL_p formula φ, determine whether $\pi \models \varphi$ holds.
  - **Solution:** Check if the formula holds by invoking a SAT solver to verify whether `sat` is true.
- **Log Generation**
  - **Problem:** Given an LTL_p formula φ, find a set of traces π such that $\pi \models \varphi$.
  - **Solution:** invoke an allSAT solver to guess the traces.

Process Discovery

- **Problem:** Given an event log $\mathcal{L}$, derive a formula φ such that every trace in the log satisfies the formula, i.e., $\forall \pi \in \mathcal{L} : \pi \models \varphi$.
- **Solution:** Use an allSAT solver to compute a (language-minimal) formula/automaton that captures the behaviour represented in $\mathcal{L}$.
- **Special Case (Declare Templates):** Instead of deriving a new formula, verify the conformance of predefined constraints against the log (and eliminate any redundant constraints).

Trace Alignment

- **Trace Alignment** is the problem of aligning, with a minimal number of modifications, a trace π with a formula φ, producing a new trace π' satisfying φ

Example

Given $\varphi = \mathbf{G}(a \rightarrow \mathbf{F}b)$ and $\pi = aba$, the trace can be aligned by removing the last occurrence of a producing $\pi' = aba$

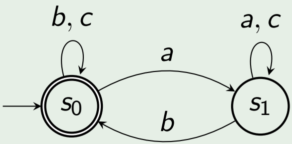
Trace Alignment: Encoding

- The problem can be encoded as cost-optimal planning.
- The action allowed are:
  - `del`: the removal of an activity,
  - `add`: the insertion of an activity.
- A special action `sync`, of cost null, is used.
- The problem reduces to a text cursor moving from left to right

Example (cont'd)

Trace $\pi' = aba$ is produced from trace $\pi = aba$ as follows:

```
State: |aba
Action: sync
State: a|ba
Action: sync
State: ab|a
Action: del
State: ab|
```



Trace Alignment: Encoding

- The problem is represented using the Planning Domain Definition Language (PDDL).
- It can be solved using any standard AI planner that supports cost actions.

sync

```
(:action sync
:parameters (?t1 - trace_state ?e - activity
             ?t2 - trace_state)
:precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
:effect(and (not (cur_state ?t1)) (cur_state ?t2)
        (forall (?s1 ?s2 - automaton_state)
                (when (and (cur_state ?s1)
                           (automaton ?s1 ?e ?s2))
                      (and (not (cur_state ?s1))
                           (cur_state ?s2))))))
```

Trace Alignment (cont'd)

add and del

```
(:action add
:parameters (?e - activity)
:effect (and (increase (total-cost) 1)
          (forall (?s1 ?s2 - automaton_state)
           (when (and (cur_state ?s1)
                      (automaton ?s1 ?e ?s2))
                 (and (not (cur_state ?s1))
                      (cur_state ?s2))))))
(:action del
:parameters (?t1 - trace_state ?e - activity
             ?t2 - trace_state)
:precondition (and (cur_state ?t1) (trace ?t1 ?e ?t2))
:effect(and (increase (total-cost) 1)
            (not (cur_state ?t1)) (cur_state ?t2)))
```

Table of Contents

Conclusion and Future Work

- **Summary:** We explored some key problems in Declarative Process Mining:
  - Log Generation, Conformance Checking, and Trace Alignment (reasoning problems over event data).
  - Process Discovery (mining/learning from event data).
- **Approach:** Reduce these problems to combinatorial search and optimization, and solve them efficiently using Automated Reasoning (SAT solvers and AI Planners).
- **Future Work:** Learn/repair process models via *local perturbations*.