A Supplementary material

Proofs

We prove the theorems witnessing correctness of our approach, i.e., that the crossover operator of Algorithm 1, as well as the mutation operators by the two strategies of Algorithm 3, preserve the satisfaction of LTL_p formulae.

Theorem 1. *Let φ be a LTL_p formula, and τ_{p_1}, τ_{p_2}, and τ be process traces over Σ, with $\tau \models \varphi$. Let $p_c \in \mathbb{R}_{[0,1]}$. Assume that Algorithm 1 is invoked by passing τ_{p_1}, τ_{p_2}, p_c, τ, and Σ_φ as input, and that it returns τ_o. Then $\tau_o \models \varphi$.*

Proof. Let $n = len(\tau)$. Upon inspection of Algorithm 1, one can see that every output τ_o produced by the algorithm relates to the input trace τ as follows:

1. $len(\tau_o) = len(\tau) = n$;
2. for every $i \in \{1, \ldots, n\}$:
   (a) if $\tau(i) \in \Sigma_\varphi$ then $\tau_o(i) = \tau(i)$;
   (b) if instead $\tau \notin \Sigma_\varphi$, that is, $\tau \in \overline{\Sigma_\varphi}$, then $\tau_o(i) \in \overline{\Sigma_\varphi}$ as well – equivalently, $\tau_o(i) \in \overline{\Sigma_\varphi}$ if and only if $\tau(i) \in \overline{\Sigma_\varphi}$.

By absurdum, imagine that $\tau_o \not\models \varphi$. Since, by property (1) above, $len(\tau_o) = len(\tau)$, this means that the violation must occur due to a mismatch in the evaluation of an atomic formula in some instant. Technically, there must exist $i \in \{1, \ldots, n\}$ and an atomic sub-formula $a \in \Sigma$ of φ (which, by definition, requires $a \in \Sigma_\varphi$) such that either:
(A) $\tau, i \models a$ and $\tau_o, i \not\models a$, or
(B) $\tau, i \not\models a$ and $\tau_o, i \models a$.
Case (A). By the LTL_p semantics, we have $\tau(i) = a$ and $\tau_o(i) \neq a$. However, this is impossible: since a belongs to Σ_φ, then by property (2a) above, we have $\tau(i) = \tau_o(i)$.
Case (B). By the LTL_p semantics, $\tau, i \not\models a$ if and only if $\tau(i) = b$ for some $b \in \Sigma \setminus \{a\}$. There are two sub-cases: either $b \in \Sigma_\varphi \setminus \{a\}$, or $b \in \overline{\Sigma_\varphi}$. In the first sub-case, impossibility follows again from the fact that, by property (2a) above, since $b \in \Sigma_\varphi$, then $\tau_o(i) = \tau(i) = b$, which implies $\tau_o, i \not\models a$. In the second sub-case, impossibility follows from the fact that $\tau_o(i)$ cannot be a, since by property (2b), the fact that $\tau(i) \in \overline{\Sigma_b}$ implies that also $\tau_o(i) \in \overline{\Sigma_b}$. □

Theorem 2. *Let φ be a LTL_p formula, τ_o a process trace over Σ s.t. $\tau_o \models \varphi$, $D = \{\mathcal{D}_i\} \subset \Sigma^{|\tau_o|}$ the domains of each gene, and $p_{mut} \in \mathbb{R}_{[0,1]}$. Assume that Algorithm 3 is invoked by passing $\tau_o, \varphi, p_{mut}, D, S$ as input (with $S \in \{\texttt{aPriori}, \texttt{Online}\}$), and that it returns τ'_o. Then $\tau'_o \models \varphi$.*

Proof. Correctness of `aPriori` is proven analogously of Theorem 1. Correctness of `Online` derives directly from the correspondence between the traces that satisfy φ, and the traces accepted by the DFA of φ, A_φ. From the definition of DFA acceptance, we have that since trace τ_o satisfies φ, there is a sequence $q_0, \ldots, q_n$ of states of A_φ, such that: *(i)* the sequence starts from the initial state q_0 of A; *(ii)* the sequence culminates in a last state, that is, $q_n \in F$; *(iii)* for every $i \in \{1, \ldots, n\}$, we have $\delta(q_{i-1}, \tau_o(i)) = q_i$.

From Algorithm 3, we have that trace τ_o is mutated through `Online` by selecting an instant i, and allowing for replacing $\tau_o(i)$ with an activity a returned from Algorithm 2. From Algorithm 2, we know that activity a is returned if the following property holds: $\delta(q_{i-1}, a) = q_i$. This means that the mutated trace τ'_o that replaces $\tau_o(i)$ with a, and maintains the rest identical, is accepted by A_φ, with the same witnessing sequence of states $q_0, \ldots, q_n$. From the correspondence between A_φ and the LTL_p semantics of φ, we thus have $\tau'_o \models \varphi$. □

Technical material

Code for reproducing experiments The *AAAI2025-temporal-constrained-counterfactuals* archive contains all the code used for re-running the experiments, including instructions on how to set up the environment and to install all the required dependencies. These instructions are provided in a Markdown file *README.md*, detailing all the steps and commands for installation of all the packages detailed above.

Code for results analysis The analysis of the results generated from the experiments is performed using a separate Jupyter notebook. This notebook includes code for loading and preprocessing the experimental results, performing statistical analysis such as calculating performance metrics, visualizing data, and comparing different methods. It also contains scripts for plotting results to visualize the performance and method comparisons.

This Jupyter notebook is included in the supplementary material and provides a comprehensive view of the analysis conducted, ensuring transparency and reproducibility of our results. For further details, the Jupyter notebook *results_analysis.ipynb* can be accessed as part of the supplementary material package.

Datasets

Instructions to download the datasets used to run the experiments above are detailed in the README.MD file in the directory containing the code.

LTL_p formulas for each dataset

For each dataset, we used different Linear Temporal Logic over Process Traces (LTL_p) formulas to check coverage at 10%, 25%, and 50%. Below are the specific formulas used for each dataset:

BPIC2012 Dataset

- **10%:** $\mathsf{F}(\text{osentcomplete}) \wedge \mathsf{G}(\text{osentcomplete} \rightarrow \neg(\text{aacceptedcomplete}) \ \mathsf{U} \ (\text{wcompleterenaanvraagcomplete})) \wedge \mathsf{F}(\text{osentbackcomplete})$
- **25%:** $\mathsf{F}(\text{osentcomplete}) \wedge \mathsf{G}(\text{osentcomplete} \rightarrow \neg(\text{aacceptedcomplete}) \ \mathsf{U} \ (\text{wcompleterenaanvraagcomplete})) \wedge \mathsf{F}(\text{osentbackcomplete}) \wedge \mathsf{G}(\text{wcompleterenaanvraagstart} \rightarrow \mathsf{F}(\text{aacceptedcomplete})) \wedge (\mathsf{F}(\text{wnabellenoffertesstart}) \wedge \mathsf{F}(\text{wnabellenoffertescomplete})) \wedge (\mathsf{F}(\text{oselectedcomplete}) \vee \mathsf{F}(\text{wvaliderenaanvraagstart}))$

- **50%:** $\mathsf{F}(\text{osentcomplete}) \wedge \mathsf{G}(\text{osentcomplete} \rightarrow \neg(\text{aacceptedcomplete}) \mathsf{U} (\text{wcompleterenaanvraagcomplete})) \wedge \mathsf{G}(\text{wcompleterenaanvraagschedule} \rightarrow \mathsf{F}(\text{wcompleterenaanvraagstart})) \wedge (\mathsf{F}(\text{wnabellenoffertesstart}) \vee \mathsf{F}(\text{wnabellenoffertescomplete})) \wedge (\mathsf{F}(\text{oselectedcomplete}) \vee \mathsf{F}(\text{wvaliderenaanvraagstart})) \wedge \text{asubmittedcomplete} \wedge \mathsf{F}(\text{oselectedcomplete} \vee \text{apartlysubmittedcomplete}) \wedge \mathsf{G}(\text{ocreatedcomplete} \rightarrow \mathsf{F}(\text{osentbackcomplete})) \wedge \mathsf{F}(\text{afinalizedcomplete}) \vee \mathsf{F}(\text{apreacceptedcomplete}) \vee \mathsf{F}(\text{wafhandelenleadscomplete})$

BPIC17 Dataset

- **10%:** $\text{acreateapplication} \wedge \neg(\text{aconcept}) \mathsf{U} (\text{wcompleteapplication})$
- **25%:** $\text{acreateapplication} \wedge \neg(\text{aconcept}) \mathsf{U} (\text{wcompleteapplication}) \wedge (\mathsf{F}(\text{ocreateoffer}) \rightarrow \mathsf{F}(\text{wcallafteroffers})) \wedge \mathsf{F}(\text{wcompleteapplication})$
- **50%:** $\text{acreateapplication} \wedge \neg(\text{aconcept}) \mathsf{U} (\text{wcompleteapplication}) \wedge \mathsf{G}(\text{ocreateoffer} \rightarrow (\mathsf{F}(\text{wcallafteroffers}) \vee \mathsf{F}(\text{wvalidateapplication}))) \wedge (\mathsf{F}(\text{ocreated}) \rightarrow X(\text{osentmailandonline} \vee \text{osentonlineonly})) \wedge \mathsf{G}((\text{aincomplete} \vee \text{apending}) \rightarrow (X(\text{wcallincompletefiles}) \wedge \mathsf{F}(\text{wvalidateapplication})))$

Claim Management Dataset

- **10%:** $\mathsf{G}(\text{contacthospital} \rightarrow X(\text{acceptclaim} \vee \text{rejectclaim}))$
- **25%:** $\mathsf{G}(\text{contacthospital} \rightarrow X(\text{acceptclaim} \vee \text{rejectclaim})) \wedge \mathsf{F}(\text{createquestionnaire})$
- **50%:** $(\mathsf{F}(\text{contacthospital}) \rightarrow \mathsf{F}(\text{highinsurancecheck})) \wedge \mathsf{G}(\text{preparenotificationcontent} \rightarrow X(\text{sendnotificationbyphone} \vee \text{sendnotificationbypost})) \wedge \mathsf{G}(\text{createquestionnaire} \rightarrow \mathsf{F}(\text{preparenotificationcontent})) \wedge \text{register}$

Predictive model Hyperparameter Configuration

For the XGBoost predictive model, we defined the following hyperparameter search space:

- The *number of estimators* (`n_estimators`) was selected from an integer range between 150 and 1000.
- The *maximum* depth of each tree (`max_depth`) was chosen as an integer value within the range of 3 to 30.
- The learning rate (`learning_rate`) was selected from a continuous uniform distribution between 0.01 and 0.5.
- The *subsample* ratio of the training instance (`subsample`) was chosen from a continuous uniform distribution between 0.5 and 1.

The hyperparameter search was performed using the Hyperopt library, which allowed us to explore this search space efficiently to find the best configuration for our XGBoost model. We ran the hyperparameter search for 20 iterations for each dataset, prefix length and LTL_p combination.

Software and Tools Used for Compliance Checking with LTL_p Formula

In our experiments, we utilized the **Declare4Py** Python package, which is available at `https://github.com/ivanDonadello/Declare4Py`. This package was specifically employed to check compliance with Linear Temporal Logic over Finite Traces (LTL_f) formulas, readily applicable for LTL_p formulas.

The **Declare4Py** package relies on several external tools to perform its compliance checking:

1. **LTLf2DFA**:
   - The **LTLf2DFA** tool is necessary for converting LTL_p formulas into Deterministic Finite Automata (DFA), which is a crucial step in the compliance checking process.
   - You can access **LTLf2DFA** at `https://github.com/whitemech/LTLf2DFA/`
   - **LTLf2DFA** itself has a dependency on the **MONA** tool.
2. **MONA**:
   - **MONA** is a C++ package required by **LTLf2DFA** to handle the conversion process from LTL_p to DFA efficiently.
   - **MONA** can be downloaded from https://www.brics.dk/mona/download.html.

These tools together enabled the effective compliance checking against the LTL_p formulas in our experimental setup.

Random Seed Settings

For reproducibility of results in both hyperparameter optimization and counterfactual generation, random seed settings are crucial. In our experiments, we use the following random seed management approaches:

- **Hyperparameter Optimization:** We use random seeds to ensure the reproducibility of the hyperparameter search process. This involves setting a fixed random seed when sampling hyperparameters, which is crucial for obtaining consistent and comparable results across different runs. For instance, we utilize the 'np.random.seed(seed_value)' function from NumPy to control randomness during hyperparameter tuning.
- **Counterfactual Generation:** Random seeds are also used to ensure reproducibility in the counterfactual generation process. This involves setting the random seed for various stochastic elements within the counterfactual generation algorithms. Again, we use 'np.random.seed(seed_value)' to achieve this.

The specific random seed values used in our experiments are detailed in the configuration settings of our code.