Trace Language: Mining Micro-configurations from Process Transition Traces

Karnika Shivhare[1], Rushikesh K Joshi[1]

[1]Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai - 400 076, India.

Abstract

The paper presents Trace Language, a language for compact encoding of process trace sets. Thirteen micro-configurations are proposed and tested over a few process mining algorithms to identify success and failure points of the latter. Trace expressions are developed for the micro-configurations using Trace Language.

Keywords

Micro-configurations, Patterns, Petri Nets, Process Mining, Process Models, Traces, Trace Language.

1. Introduction

A process is a progression of activities, and its execution stamps footprints in form of series of triggered transitions as the process progresses. These footprint trails are known as traces, which together form a *trace set* corresponding to the set of traces generated out of multiple executions of the process. Process Mining algorithms routinely use trace sets to recover or build process models from them. We observed that there are certain inherent patterns among processes which we call as micro-configurations, that get over-passed by process mining algorithms and stand as fracture points for them. At this stretch, realizing the need of a language that can represent trace sets compactly, help in automating implementations and serve as basis for refinements and improvements in mining algorithms, we propose a compact novel trace language. It serves the purpose of trace set generator when used for representing micro-configurations.

The paper first expounds the operators of the Trace Language, followed by a discussion of results enlisting Trace Language expressions for thirteen micro-configurations which were tested over five process mining algorithms in the Python (pm4py) framework [1]. The algorithms tested are Alpha Mining [2], Alpha($+$) miner [3], Heuristics Miner [4], Inductive Miner [5] and Directly-Follows Graphs (DFG) [6].

2. Trace Language Operators

Chronicle Ordering ($\rightarrow$), Alternative ($\rtimes$), Concurrent ($\|$) These are basic operations of sequence, choice and parallel composition. For example, $a \rightarrow b \rightarrow c$ represents trace set with single trace *{abc}*, $a \rtimes pqr$ represents trace set with two possible traces *{a, pqr}*, and $a \parallel b$ represents trace set with two possible traces *{ab, ba}* and $ab \parallel pq$ represents trace set *{abpq, apbq, apqb, pqab, pabq, paqb}*. In the operators below, the operands can be individual transitions or composites defined by chronicle orderings. Chronicle Ordering is the default operator if there is no operator specified, as in trace *abcd*, which represents $a \rightarrow b \rightarrow c \rightarrow d$.

Swapper ($\nparallel$) It operates on chronicle orderings as operands, treating them as atomic (indivisible) substrings. It concatenates them in both permutations to represent two possible traces. For example, $a \nparallel b$ represents trace set *{ab, ba}* and $abc \nparallel pqr$ represents trace set *{abcpqr, pqrabc}*. The operator is commutative but not associative.

Bowtie operator ($\bowtie$) This operator represents a generator of two particularly fashioned traces that are structured as ordered and pairwise concatenations of transition sequences sandwiching the common transition sequence. The common transition sequence is written as superscript on the operator symbol. For example, $< a, b > \bowtie^{c} < d, e >$ produces two traces {*acd, bce*}. The operator is neither commutative nor associative.

m-Subsequence Operator ($\mathbf{S}^{m}$) This operator represents all valid subsequences of length m for the given unary operand that represents a sequence of chronicle orderings. The subsequences are valid if they are in growing sequence of chronicles. For example, $\boldsymbol{S}^{2} < xyz >$ represents traces {*xy, xz, yz*}. With m as 1, the operator functions as a *shredder* that creates all traces of length 1. For example, $\boldsymbol{S}^{1} < xyz >$ represents traces {*x, y, z*}.

any-Subsequence Operator ($\mathbf{S}^{any}$) This unary operator represents all subsequences that are in growing sequence of chronicle orderings of the operand. For example, $\boldsymbol{S}^{any} < xy >$ represents trace set { *x, y, xy*}. There are no empty or null subsequences.

Floating Operator ($\mathbb{F}$) It represents insertion of an indivisible floating operand as prefixed, in-fixed or suffixed subsequence into the divisible host (i.e. base) operand provided as superscript in the operator symbol. For example, $< ab > F^{<c>}$ forms traces *cab*, *acb* and *abc*. Similarly,

International Workshop on Petri Nets and Software Engineering 2022, PNSE'22

karnika@cse.iitb.ac.in (K. Shivhare); rkj@cse.iitb.ac.in (R. K. Joshi)
https://www.cse.iitb.ac.in/~karnika/ (K. Shivhare); https://www.cse.iitb.ac.in/~rkj/ (R. K. Joshi)

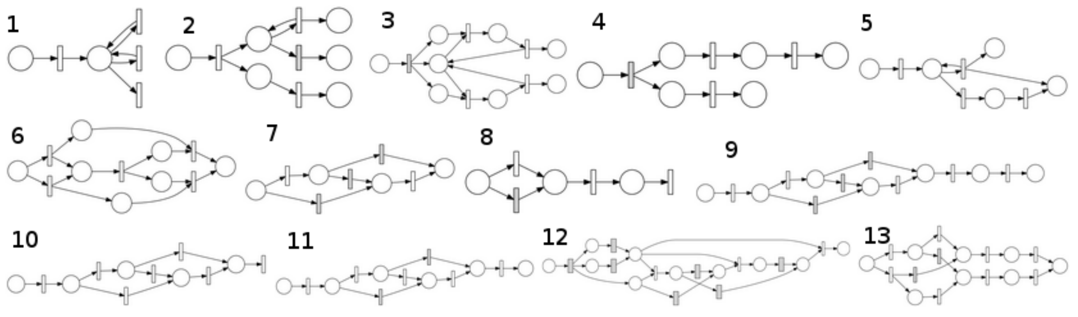

Figure 1: Micro-configurations

Table 1

Sr. No.	Microconfigurations	A	A+	H	I	D	Trace Expressions
1	Ticketed Service	✗	✗	✗	✓	✓	$a \rightarrow X \rightarrow d$ where, $X = (\phi \rtimes b \rtimes c) \rightarrow X$
2	Asynchronous Service Loop	✗	✗	✗	✓	✗	$a \rightarrow (X \parallel c)$ where, $X = (b \rightarrow X) \rtimes \epsilon$
3	Critical Section	✗	✗	✗	✗	✗	$ab \nparallel cd$
4	Concurrent Branching	✗	✓	✗	✓	✗	$< ab > F^{<c>}$
5	Early Completion Option	✗	✗	✗	✗	✗	$a \rightarrow (< c, \phi > \bowtie^{b} < \phi, d >)$
6	Bowtie	✗	✗	✗	✗	✗	$< a, b > \bowtie^{c} < d, e >$
7	Seniority	✗	✗	✓	✗	✗	$S^{any} < a \rightarrow b >$
8	Initial Bypass	✗	✗	✓	✓	✗	$\mathbb{D}^{a}(abc)$
9	Intertwined Vanilla Bypass	✗	✗	✗	✗	✓	$\mathbb{D}^{<c,b>}(abcde)$
10	Intertwined Active Bypass	✓	✓	✓	✗	✓	$\mathbb{B}^{<cd/g,bc/f>}(abcde)$
11	Intertwined Long Bypass	✗	✗	✓	✗	✓	$\mathbb{D}^{<cd,bc>}(abcde)$
12	Ordered Subsequences	✗	✗	✗	✗	✗	$S^{2} < abc >$
13	Crossover	✗	✗	✓	✗	✓	$\mathbb{B}^{<pq/a>}(pqrs) \rtimes \mathbb{B}^{<ab/p>}(abcd)$

$< ab > F^{<cd>}$ creates {*cdab, acdb, abcd*}. This does not include any trace involving *d* before *c* or *b* before a since the floating operand is indivisible. The operator is neither commutative nor associative.

Substring Bypass Operator ($\mathbb{B}$) This ternary operator operates on one host operand and a bypass pairing, which is a pair of chronicle orderings (substrings). In host trace, an occurrence of pair's former substring is replaced to generate another trace. Substring bypass operator generates host trace and also a bypass trace formed by replacement. For example, $\mathbb{B}^{x/a} < xyz >$ represents {*ayz, xyz*}. A compact notation for bulk bypass results is $\mathbb{B}^{x/a,y/b} < xyz >$ to represent {*xyz, ayz, xbz*}.

Substring Drop Operator ($\mathbb{D}$) A special case of Substring Bypass Operator, it bypasses substring in host operand with an empty substring, i.e., it drops the specified substring from the host to generate a new trace. For example, $\mathbb{D}^{p} < pqrs >$ constructs trace set {*pqrs, qrs*} and $\mathbb{D}^{p,qr} < pqrs >$ represents trace set {*pqrs, qrs, ps*}.

3. Results Table 1 shows trace expressions for microconfigurations (elementary nets in Fig. 1), and success/failures of Alpha (A), Alpha+ (A+), Heuristics (H), Inductive (I) and DFG (D) algorithms in Pm4Py[1] library.

References

[1] A. Berti, S. J. van Zelst, W. van der Aalst, Process mining for python (pm4py): Bridging the gap between process- and data science, 2019.

[2] W. Aalst, A. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowledge and Data Engineering (2004).

[3] A. Weijters, Process mining: Extending the alpha-algorithm to mine short loops (2004).

[4] A. Weijters, W. Aalst, A. A. K. Medeiros, Process mining with the HeuristicsMiner algorithm, Technische Universiteit Eindhoven, 2006.

[5] S. Leemans, D. Fahland, W. Aalst, Discovering block-structured process models from event logs - a constructive approach, in: Int. Conf. on Application and Theory of PN and Concurrency, Springer, 2013.

[6] W. Aalst, Process discovery from event data: Relating models and logs through abstractions, 2018.