Корректировка моделей процессов по логам событий

Алексей Мицюк

Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук — Лаборатория ПОИС

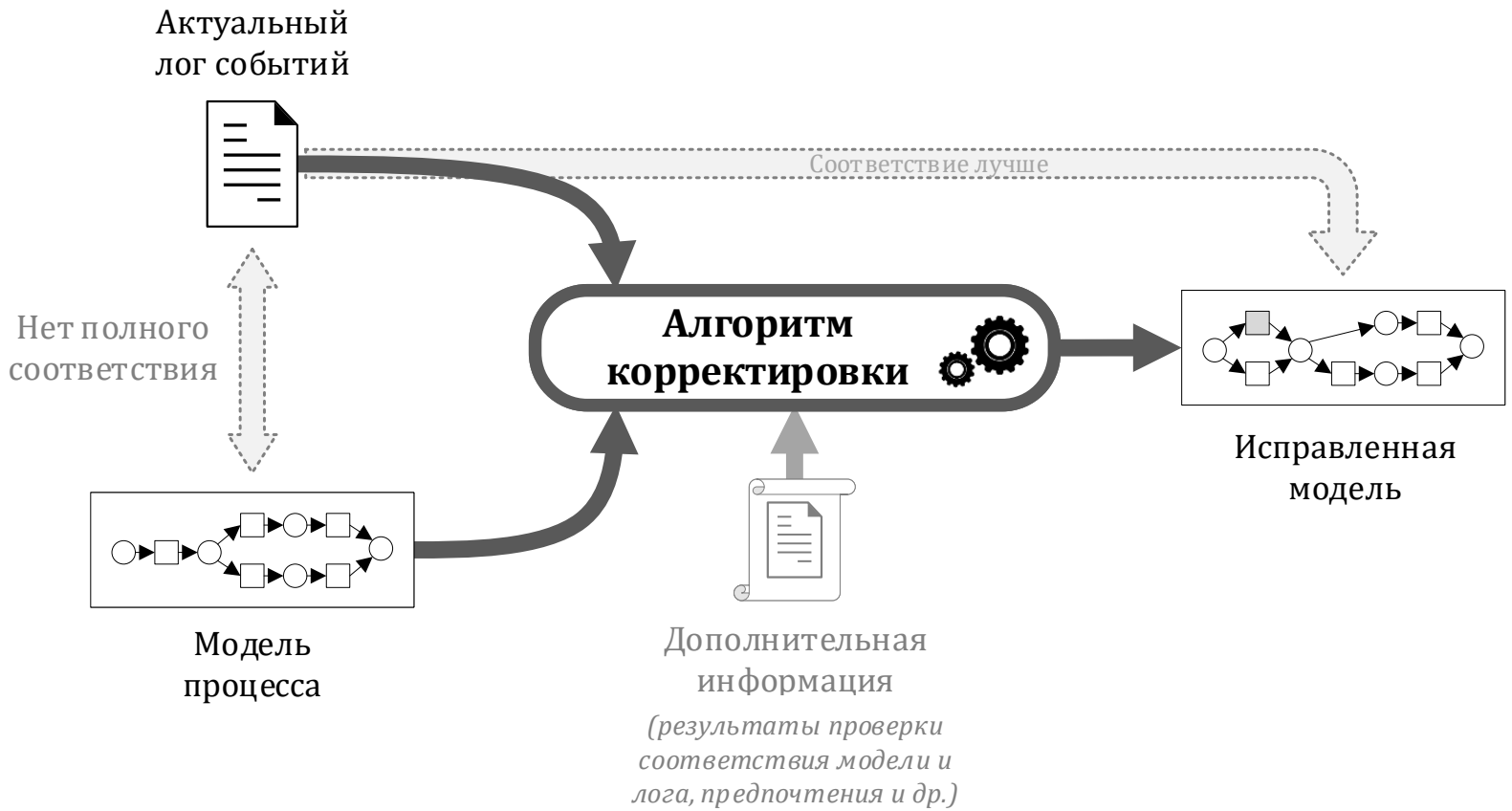
amitsyuk@hse.ru

Доклад для семинара секции ACM SIGMOD, ВМиК МГУ

Москва, 26 октября 2017 г.

Содержание доклада

Исправление (корректировка) моделей процессов



Актуальность 1

В мире:

Десятки примеров применения опубликованы на сайте http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies

Среди других о применении технологии сообщают муниципальные органы Нидерландов, компании Siemens, Phillips, оператор мобильной связи Vodafone, компании General Motors и Bayer, Copenhagen Airports A/S, Ana Aeroportos de Portugal, Bolton Council, нидерландский поставщик газа и электричества Alliander, австралийская страховая компания Suncorp, Samsung Electro-Mechanics, госпиталь Сеульского университета Bundang, Vaisala, Caverion, госпиталь Isala.

В России:

PwC Россия заявляет, что использует технологию (https://www.pwc.ru/ru/careers/events/process-mining-6-10.html)

ICL Services заявляет, что использует технологию (https://icl-services.com/services/pro-zrenie-analiz-effektivnosti-biznes-protsessov/)

Компания Ramax заявляет о внедрении технологии в банке ВТБ24 и других организациях (http://ramax.ru/process_mining_24-04-2017/)

Правительство Москвы тестирует технологию (Проект А.А.Каленковой «Алгоритмы и программные средства анализа бизнес-процессов предоставления государственных услуг» (2016-17 гг.) поддержан РФФИ и правительством города Москвы, грант 15-37-70008 мол_а_мос.)

Актуальность 2

SAP Process Mining by Celonis

Любая компания, использующая для автоматизации своей деятельности ERP-системы SAP (т.е. множество крупных компаний в Европе), может применять process mining «из коробки». (https://www.sap.com/products/process-mining.html)

Зачем исправлять модели процесса?

Исправление полезно, например, в ситуации, когда со временем процесс исполняется, модель должна быть актуальна, но и не быть совершенно новой.

Модели и логи

Модель процесса

Сеть потоков работ — помеченная сеть Петри $N = (P, T, F, l)$ с выделенными начальной разметкой M_i и конечной разметкой M_o, в которой

- есть ровно одна позиция-исток $i \in P$ такая, что $\bullet i = \emptyset$ и $M_i = [i]$,
- есть ровно одна позиция-сток $o \in P$ такая, что $o\bullet = \emptyset$ и $M_o = [o]$,
- каждая позиция или переход $n \in P \cup T$ лежит на пути из i в o.

Помеченная сеть Петри — четверка (P, T, F, l), где P и T — непересекающиеся множества позиций и переходов, а $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$ — функция потока, а $l\colon T \to \mathcal{U}_A \cup \{\tau\}$ — функция пометки. $M\colon P \to \mathbb{N}$ — разметка (текущее состояние модели).

Лог событий

- $A \subseteq \mathcal{U}_A$ — множество действий процесса,
- $\sigma \in A^*$ — конечная последовательность событий — **трасса**,
- $L \in \mathcal{B}(A^*)$ — конечное мультимножество трасс — **лог событий**.

Критерии оценки модели

Соответствие (fitness) — в какой степени трассы лога могут быть исполнены в модели?

Точность (Precision) — в какой мере модель может порождать поведение, не представленное в логе?

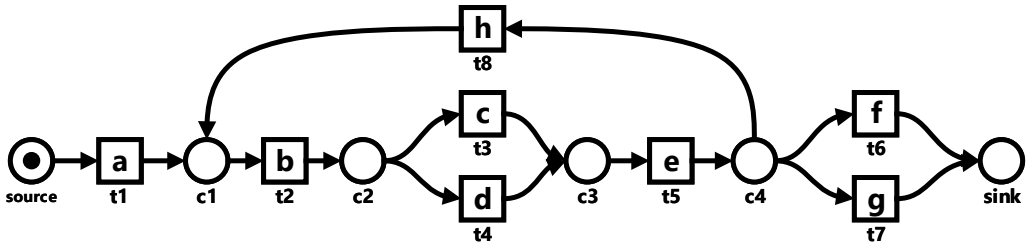
Обобщённость (Generalization) — насколько абстрактна модель? не является ли она слишком специфичной для лога?

Простота (Simplicity) — насколько компактна модель? много ли в ней лишнего?

Для оценки **сходства** (Similarity) исходной и исправленной моделей используем алгоритм вычисления GED из [Dijkman et al, 2009].

Подробности см., например, в [van der Aalst, 2016].

Пример



log	a	b	c	e	g
model	a	b	c	e	g
	t_1	t_2	t_3	t_5	t_7

log	a	d	b	≫	e	f
model	a	≫	b	d	e	f
	t_1		t_2	t_4	t_5	t_6

Как исправлять?
Рассмотрим различные подходы.

Обзор работ по теме исследования

Вопрос корректировки моделей процессов рассматривается, как минимум, в этих работах:

- Process Model Repair (D. Fahland and W. van der Aalst) — 2011–2015 [Fahland, van der Aalst, 2015];
- Impact-Driven Process Model Repair (A. Polyvyanyy et al) — 2016 [Polyvyanyy et al, 2017].
- Improving Business Process Models Using Observed Behavior (J. Buijs et al) — 2012 [Buijs et al, 2012]
- Automated Error Correction of Business Process Models (M.Gambini et al) — 2011 [Gambini et al, 2011]
- Interactive and incremental business process model repair (A. Armas-Cervantes et al) — 2017 [Armas-Cervantes et al, 2017]

*под одним пунктом может быть серия работ

Process Model Repair

Dirk Fahland и Wil van der Aalst

Process Model Repair

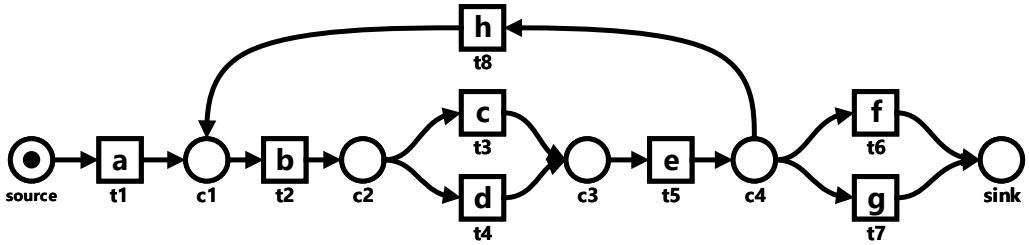
Авторы впервые сформулировали задачу исправления модели по логу.

Исправлять предлагается *соответствие* WF-сети простому логу событий.

Способ исправления — добавление искусственных подпроцессов.

*Fahland D., van der Aalst W.M.P. Model repair - Aligning process models to reality. Inf. Syst., 47 (2015), 220–243

Пример — Простой случай — 1



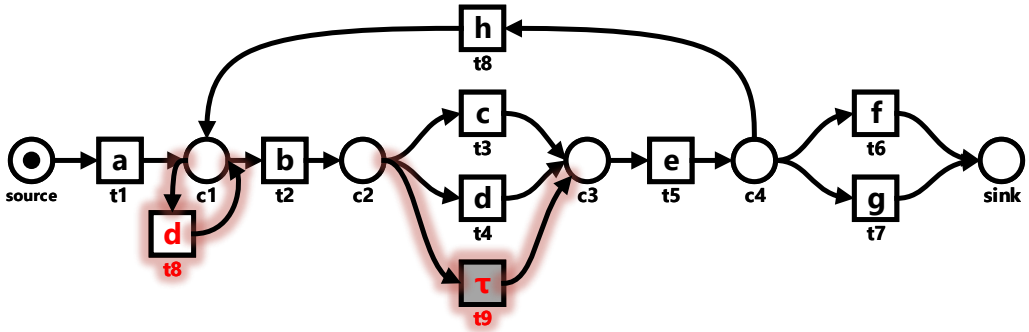
log	a	b	c	e	g
model	a	b	c	e	g
	t_1	t_2	t_3	t_5	t_7

log	a	d	b	$\gg$	e	f
model	a	$\gg$	b	d	e	f
	t_1		t_2	t_4	t_5	t_6

Как исправлять?

Будем добавлять переходы в тех местах, где требуется, чтобы выполнять поведение из трасс лога.

Пример — Простой случай — 2



log	a	b	c	e	g
model	a	b	c	e	g
	t_1	t_2	t_3	t_5	t_7

log	a	d	b	≫	e	f
model	a	≫	b	d	e	f
	t_1		t_2	t_4	t_5	t_6

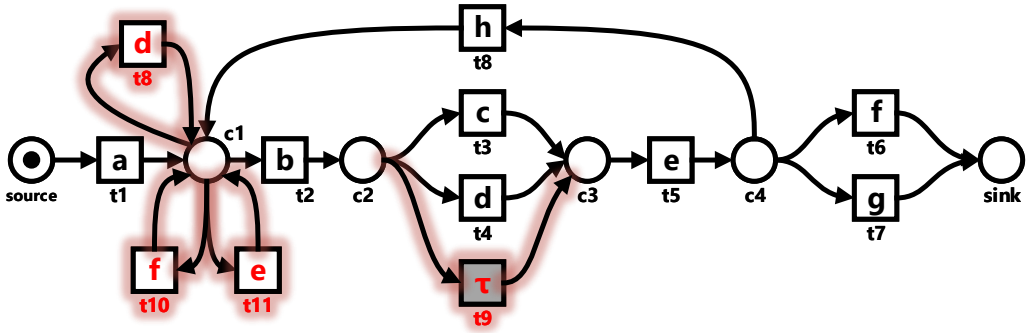
Плюс:

Ничего не удаляем, вся исходная модель остаётся без изменений.

Минус:

Добавляем новое поведение.

Пример — Более сложный случай — 1



log	a	d	f	e	b	≫	e	f
model	a	≫	≫	≫	b	d	e	f
	t_1				t_2	t_4	t_5	t_6

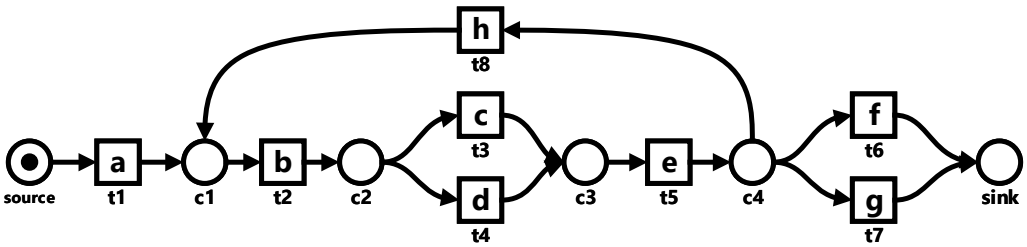
Что же делать?

Будем искать суб-трассы, которые затем объединять в суб-логи, по которым строить модели подпроцессов.

Пример — Более сложный случай — 2

Размещение шага

Шаг — пара (e_i, t_i), где e_i — событие из лога, t_i — переход модели.
Имеется последовательность срабатываний переходов $t_1 \ldots t_{i-1}$, а далее идёт шаг $(e_i, \gg)$. $loc((e_i, \gg)) = \{p \in P | M_i(p) > 0\}$, т.е. все позиции, в которых есть токены на i-ом шаге.



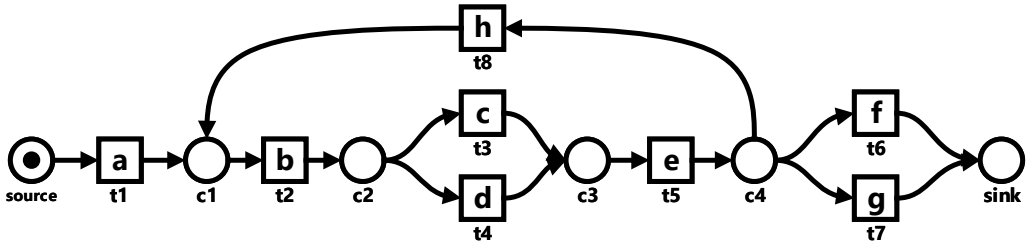
log	a	d	f	e	b	$\gg$	e	f
model	a	$\gg$	$\gg$	$\gg$	b	d	e	f
	t_1				t_2	t_4	t_5	t_6

$$loc((d, \gg)) = loc((f, \gg)) = loc((e, \gg)) = \{c_1\}$$

Пример — Более сложный случай — 3

Суб-трасса (β, Q)

Это максимальная последовательность шагов в логе (т.е. с пропусками в модели) вида $(e_i, \gg) \dots (e_{i+k}, \gg)$, имеющих общее размещение $Q = loc((e_i, \gg)) = \dots = loc((e_{i+k}, \gg))$.



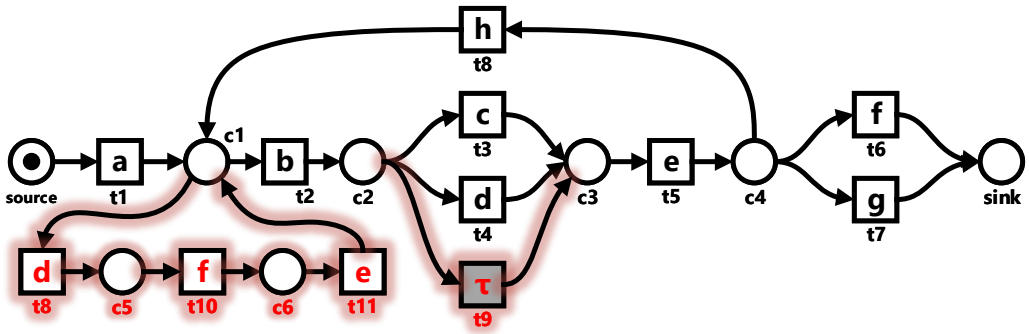
log	a	d	f	e	b	$\gg$	e	f
model	a	$\gg$	$\gg$	$\gg$	b	d	e	f
	t_1				t_2	t_4	t_5	t_6

В нашем случае: $((d, \gg)(f, \gg)(e, \gg), \{c_1\})$

Суб-лог

Это непустое множество суб-трасс такое, что пересечение их размещений непусто. Пересечение размещений всех суб-трасс называется размещением суб-лога.

Пример — Более сложный случай — 4

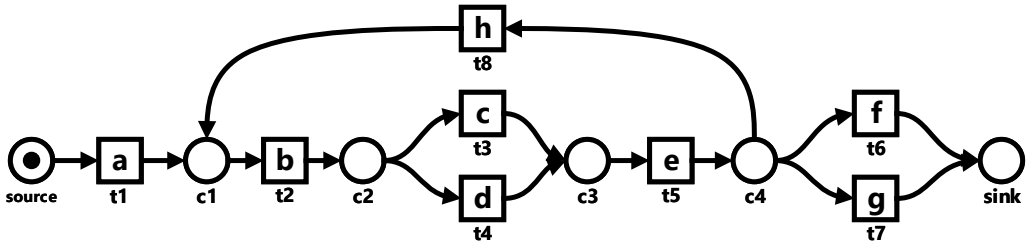


log	a	d	f	e	b	≫	e	f
model	a	≫	≫	≫	b	d	e	f
	t_1				t_2	t_4	t_5	t_6

Подпроцесс по суб-логу

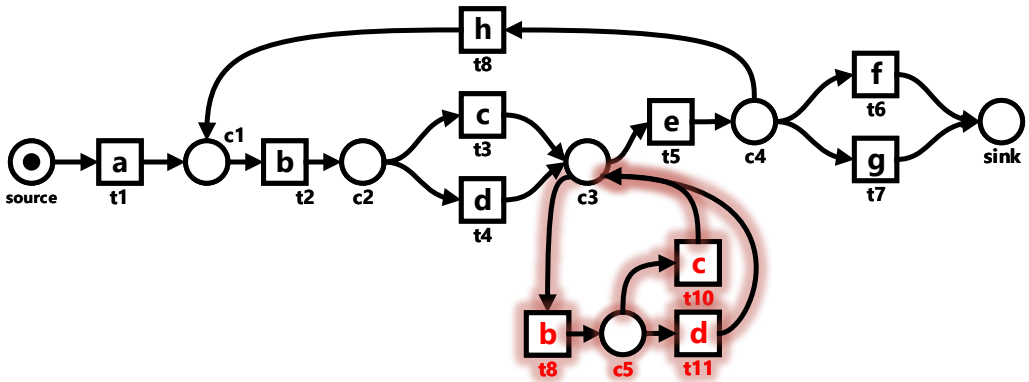
– Имеется непустой суб-лог. Каждой суб-трассе этого суб-лога добавляются искусственные события *start* и *end*.
– По суб-логу строится модель с использованием алгоритма автоматического синтеза, гарантирующего идеальное соответствие итоговой модели.
– Модель подпроцесса добавляется в размещение суб-лога.

Пример — Ещё более сложный случай — 1



log	a	b	c	b	c	b	d	e	g
model	a	b	c	≫	≫	≫	≫	e	g
	t_1	t_2	t_3					t_5	t_7

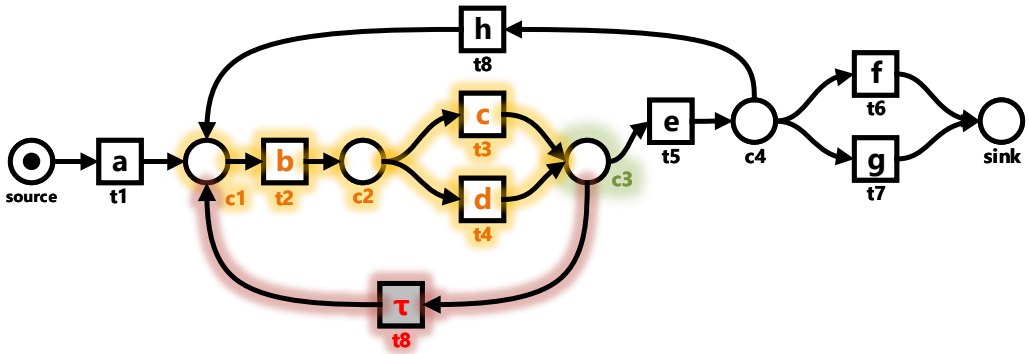
Если добавить подпроцесс, получается не очень:



Пример — Ещё более сложный случай — 2

Алгоритм поиск цикла (упрощенно)

Имеется суб-лог (S, Q), где Q — размещение суб-лога.
1. Если имеется цикл, то выходная позиция его тела лежит в Q.
2. Найдём все переходы с именами, соответствующими событиям из S, которые предшествуют позициям из Q. Эти переходы должны быть в теле цикла.
3. Найдём кратчайший путь из каждого такого перехода в позицию из Q, причём все переходы и позиции, через которые проходит путь, тоже должны быть в теле цикла.
4. Позиции, являющиеся входными для переходов из множества узлов тела цикла, которое мы строим, но не имеющие узлов-предшественников в теле цикла, являются входными для тела цикла.
5. Добавим цикл с единственным τ-переходом от выходной позиции тела к входной.



log	a	b	c	b	c	b	d	e	g
model	a	b	c	≫	≫	≫	≫	e	g
	t_1	t_2	t_3					t_5	t_7

Пример — Ещё более сложный случай — 3

Замечания

– Алгоритм строит гипотезу о наличии цикла, которая может быть неверна. Для проверки гипотезы нужно ещё раз пересчитать выравнивания для суб-лога и подсети, построенной в результате, в которых цена за пропуски хода в модели устанавливается намного выше цены за пропуски хода в логе. Если в получившихся выравниваниях нет шагов с пропуском хода в модели, то исправление для цикла построено верно. Иначе, нужно добавить τ-переходы к нужным позициям, чтобы пропускать те или иные переходы для различных исполнений цикла.

– Алгоритм выполняется ДО более простых техник, которые применяются затем для исправления ациклических проблем.

– Алгоритм не работает для случаев, когда циклическое поведение «оказалось» в разных суб-логах (последовательность не $\langle a, \mathbf{b, c, b, c}, d, e, f \rangle$, а $\langle a, \mathbf{b, c}, d, \mathbf{b, c}, e, f \rangle$ — модель другая).
Выравнивания, где поведение, относящееся к циклу, «собирается» вместе, т.е. идёт в трассах подряд, авторы называют *сохраняющими итерацию*. Авторы подобрали эвристики, которые провоцируют алгоритм поиска оптимальных выравниваний выдавать именно такие, сохраняющие выравнивания. Впрочем, нет гарантии, что алгоритм всегда вернёт именно такое выравнивание.

– Авторы предлагают и несколько других улучшений данного способа исправления, относящихся к:
1. Удалению неиспользуемых в логе фрагментов модели,
2. Выбору выравниваний для исправления, основанному на различных эвристиках,
3. Предварительной кластеризации (горизонтальной декомпозиции) трасс лога с целью упрощения починок,
4. Эвристикам выбора размещения для встраивания подпроцесса.

– Авторы произвели множество экспериментов с интересными наблюдениями (см. статью).

Impact-driven Process Model Repair

Artem Polyvyanyy, Wil van der Aalst, Arthur Ter Hofstede, Moe Wynn

Impact-driven Process Model Repair – 1

Постановка задачи такая же, как и в ранее рассмотренном примере.

Исправлять предлагается *соответствие* сети Петри простому логу событий.

Способ исправления — применение набора отдельных операций изменения модели.
Операциям присваивается степень значительности для пользователя подхода.
Идея — свести починку модели к оптимизационной задаче поиска оптимального набора изменений.

*Artem Polyvyanyy, Wil van der Aalst, Arthur Ter Hofstede, Moe Wynn. Impact-driven Process Model Repair. ACM Transactions on Software Engineering and Methodology (TOSEM), 25:4 (2017), 28:1–28:60

Impact-driven Process Model Repair – 2

Рекомендация по исправлению

Пара (R_{ins}, R_{skp}), где R_{ins} — множество действий, которое надо добавить в модель, а R_{skp} — множество действий, которые надо пропустить.

Исправление на основе выравнивания и рекомендации R

Исправление модифицирует исходную модель (получая исправленную) так, что для каждой трассы из лога, по которому исправляется модель, можно найти выравнивание между ней [этой трассой] и исправленной моделью такое, что его стоимость будет не выше стоимости оптимального выравнивания между этой трассой и исходной моделью.
Т.е. исправление, как минимум, не ухудшает соответствие.

Пример элементарной починки: Добавление одиночного перехода, в т.ч. и τ-перехода, описанные в подходе, рассмотренном ранее.
Задача: выбрать набор таких элементарных операций починки, который приведёт к наилучшему (по цене) результату.

Наивная починка

Исходная сеть S, а исправленная сеть S', исправление происходит по логу событий L в соответствии с рекомендацией R.
Наивной называется такая починка:
существуют две последовательности α и β, для которых верно, что $Set(\alpha) = R_{ins}, |\alpha| = |R_{ins}|$, а $Set(\beta) = R_{skp}, |\beta| = |R_{skp}|$;
существует последовательность сетей $\langle S = S_{\mathbf{0}}, S_{\mathbf{1}}, \ldots, S_n, S_{n+\mathbf{1}}, \ldots, S_{n+m} = S' \rangle$, которая представляет последовательность правок, в которой n добавлений одиночных переходов и m добавлений невидимых переходов (в любом порядке) таких, что они обеспечивают исправление всех асинхронных шагов во всех оптимальных выравниваниях, которые использовались для починки.

Impact-driven Process Model Repair – 3

Ограничение на починку

Тройка (C_{ins}, C_{skp}, res), где C_{ins} – стоимость вставки каждого действия, а R_{skp} – стоимость вставки пропуска каждого действия, res – общая максимальная цена починки, которая доступна.
$cost[R, C]$ – стоимость рекомендации R по отношению к ограничению C.

Оптимальная рекомендация по починке

Это такая рекомендация R, что $cost[R, C] \leq res$ и общая стоимость лога по отношению к модели для любой другой рекомендации выше, чем для оптимальной.

Задача: искать оптимальную рекомендацию.

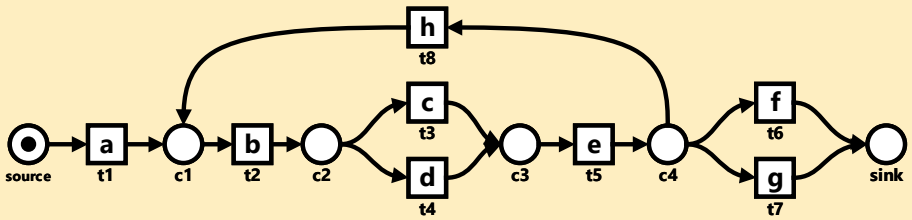
Авторы вводят алгебру рекомендаций починки, стоимость рекомендации по отношению к ограничению объявляется целевой функцией и задача сводится к оптимизационной.
Авторы приводят алгоритмы для её решения:
1. Полный перебор,
2. Оптимизированный перебор (целевая функция усложняется),
3. Алгоритм заполнения рюкзака,
4. Алгоритм Голдратта (теория ограничений),
5. Жадный алгоритм.

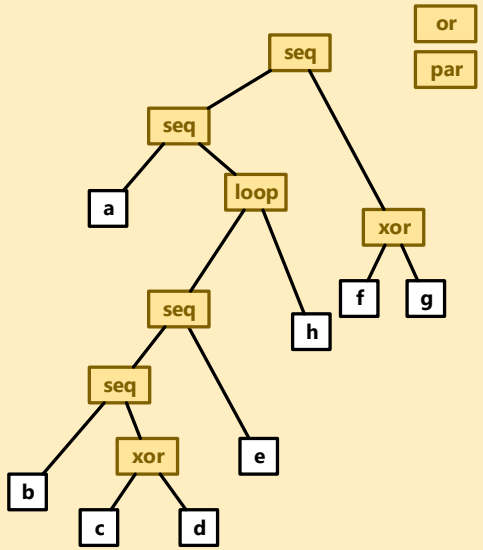
Improving Business Process Models Using Observed Behavior

Joos Buijs, Marcello La Rosa, Hajo Reijers, Boudewijn van Dongen, Wil van der Aalst

Improving Business Process Models Using Observed Behavior – 1

Модели — бездефектные, хорошо-структурированные модели, представленные деревьями процессов.

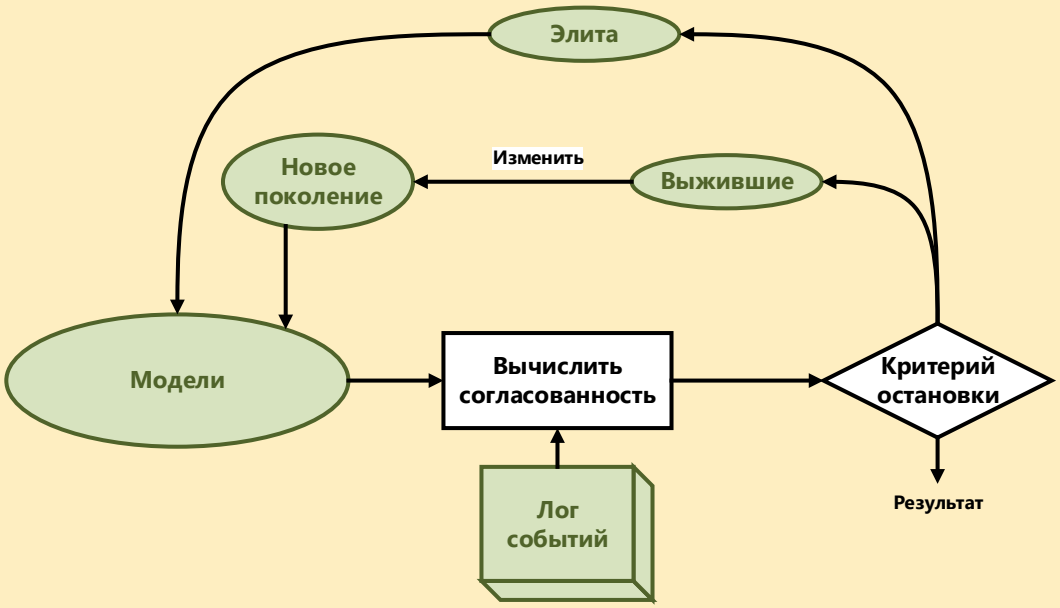




Задача: построить модель, сходную с исходной и сбалансированную по четырём метрикам согласованности с логом.

Improving Business Process Models Using Observed Behavior – 2

Применяется генетический алгоритм, устроенный примерно так:



*Joos Buijs, Marcello La Rosa, Hajo Reijers, Boudewijn van Dongen, Wil van der Aalst. Improving Business Process Models Using Observed Behavior. SIMPDA 2012, LNBIP, 162 (2013), 44–59

Automated Error Correction of Business Process Models

Mauro Gambini, Marcello La Rosa, Sara Migliorini, Arthur Ter Hofstede

Automated Error Correction of Business Process Models – 1

Дана дефектная WF-сеть.

Задача: Найти множество вариантов корректировки WF-сети, каждый из которых может быть получен из исходной сети за минимальное (не большее, чем другие имеющиеся варианты) количество изменений и содержит не больше нарушений бездефектности, чем другие имеющиеся варианты, а также исходная модель.

Бездефектность WF-сети

– возможность завершения процесса: для каждой разметки сети, достижимой из начальной, существует последовательность срабатывания переходов сети, приводящая к конечной разметке сети,
– полное завершение процесса: при достижении конечной разметки (с одним токеном в конечной позиции) ни в каких других позициях не могут оставаться *лишние* токены,
– в сети нет *мёртвых* переходов: каждый переход достижим из начальной разметки.

*Gambini M., La Rosa M., Migliorini S., Ter Hofstede A.H.M. Automated Error Correction of Business Process Models 9th International Conference Business Process Management (BPM 2011), LNCS, 6896 (2011), 148–165

Automated Error Correction of Business Process Models – 2

Корректировка заключается в выполнении некоторого набора простых операций изменения исправляемой модели. Допустимы добавление и удаление дуг, позиций, переходов с пометкой τ.

Строится множество исправленных бездефектных моделей, полученных из исходной путём применения минимального количества операций.
Для этого используется метод симуляции отжига, с помощью которого осуществляется поиск оптимального набора операций исправления.
Целевой функцией при этом является тройка усредненных значений (λ, δ, β) по моделям-кандидатам на лучшее решение конкретного шага алгоритма.
Все исправленные модели должны быть, как минимум, не хуже исходной модели по критерию бездефектности, а также более или менее сходны с ней.

Здесь λ — структурное сходство исправляемой и исходной моделей,
δ — поведенческое сходство моделей (насколько модели могут воспроизводить поведение друг друга),
а β — так называемая «скверность» модели по отношению к заданному набору исполнений исходной модели.
Составляющая β портится, если модель нарушает одно из трех условий бездефектности для исполнений, входящих в некоторое множество R.

Interactive and Incremental Business Process Model Repair

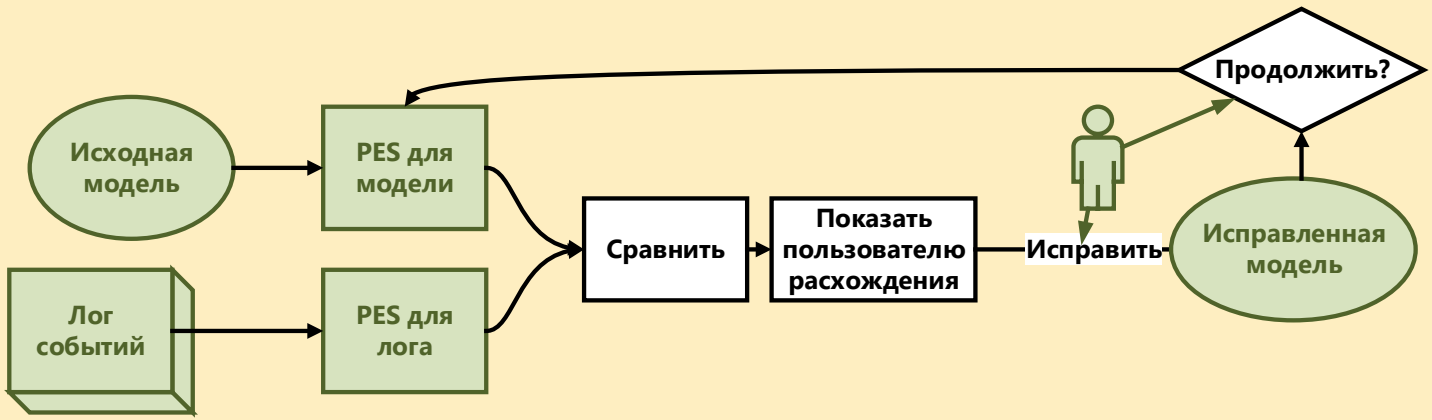
Abel Armas-Cervantes, Marcello La Rosa, Marlon Dumas,
Luciano García-Bañuelos, Nick van Beest

Interactive and Incremental Business Process Model Repair

Даны модель BPMN и лог событий.

Задача: Помочь пользователю выбрать наиболее подходящие операции починки этой модели так, чтобы результат соответствовал логу.
При сравнении используются т.н. prime event structures (набор событий, отношение каузальной зависимости, отношение конфликта), что позволяет *вербализировать* изменения, которые может совершить пользователь с моделью (типа «В модели после A выполняются оба действия B и C, а в логе они взаимно исключают друг друга. Предлагается заменить операторы *AND* на *XOR* в позициях ...»).

Схема



Корректировка моделей процессов по логу событий с использованием декомпозиции

Alexey Mitsyuk, Irina Lomazova, Wil van der Aalst

Наша постановка задачи корректировки

Дано

- Сеть потоков работ (workflow net) N — исходная модель процесса;
- Мультимножество трасс L — лог событий , представляющий текущее поведение процесса.

Задача

Построить сеть Петри N^r такую, что

- модель N^r позволяет выполнить все трассы из L (**идеальное соответствие** модели и лога);
- модель N^r допускает не «слишком много» исполнений, не представленных в логе (**высокая точность** модели);
- модели N и N^r имеют похожую графовую структуру (**сходство** моделей).

Наши дополнительные предположения

- Пометки переходов в модели и имена событий в логе берутся из одного множества действий, т.е. действия в модели и логе совпадают;
- Действия процесса не пропадают и не появляются, а только меняются местами, т.е. и наша корректировка только меняет порядок срабатывания переходов.

Критерии оценки модели

Соответствие (fitness) — в какой степени трассы лога могут быть исполнены в модели?

Точность (Precision) — в какой мере модель может порождать поведение, не представленное в логе?

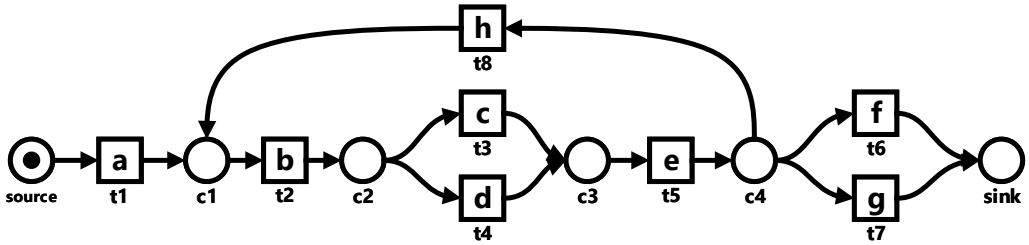
Обобщённость (Generalization) — насколько абстрактна модель? не является ли она слишком специфичной для лога?

Простота (Simplicity) — насколько компактна модель? много ли в ней лишнего?

Для оценки **сходства** (Similarity) исходной и исправленной моделей используем алгоритм вычисления GED из [Dijkman et al, 2009].

Подробности см., например, в [van der Aalst, 2016].

Пример



log	a	b	c	e	g
model	a	b	c	e	g
	t_1	t_2	t_3	t_5	t_7

log	a	d	b	≫	e	f
model	a	≫	b	d	e	f
	t_1		t_2	t_4	t_5	t_6

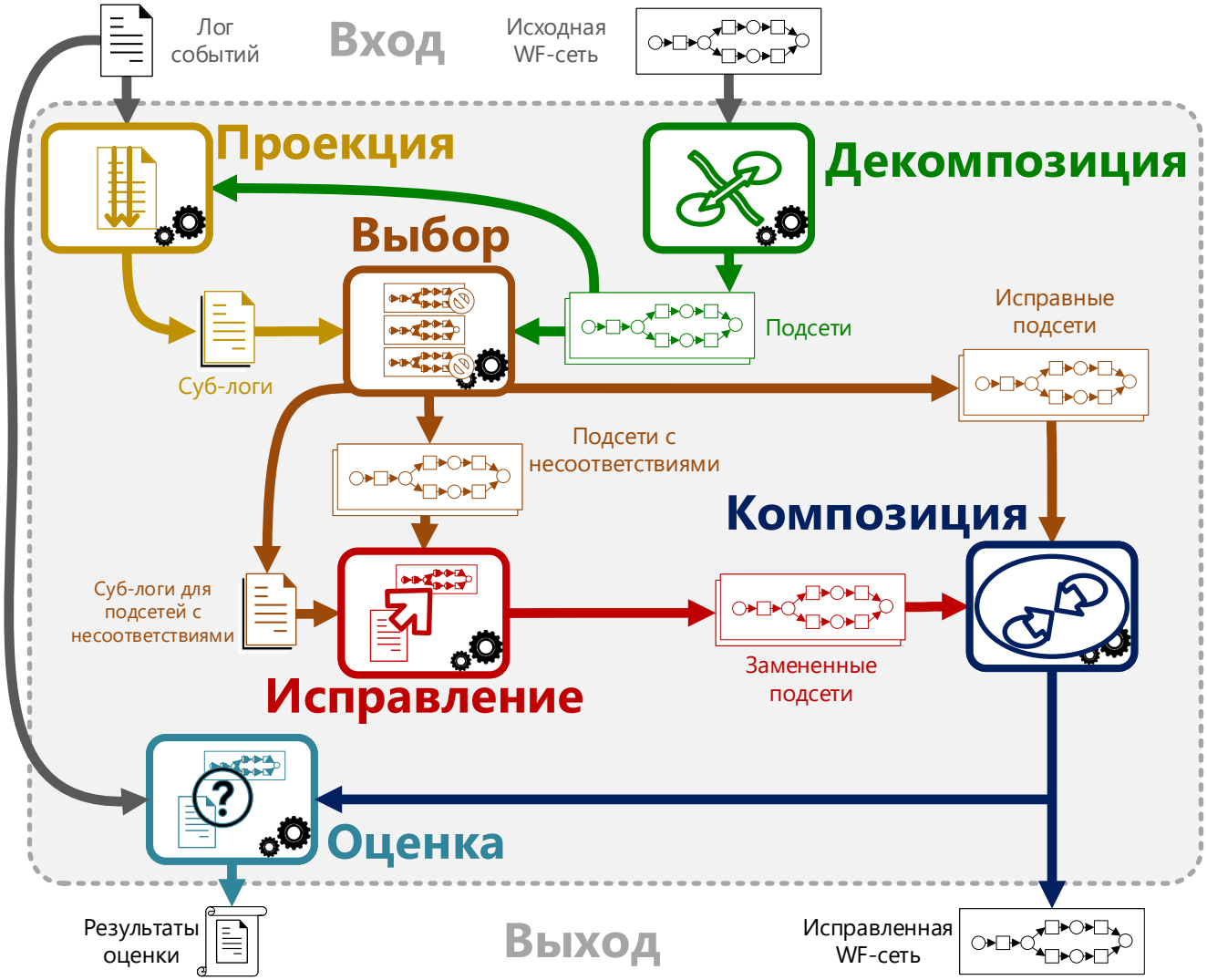
Как исправлять?

Наша идея — делать «заплатки».

Исправление моделей процессов с использованием декомпозиции

Базовая схема

Модульная схема корректировки – 1



Модульная схема корректировки – 2

Пусть $\mathcal{U}_A$ – множество действий, а $\mathcal{U}_N$ – множество всех WF-сетей с пометками из $\mathcal{U}_A$.

Модульная схема корректировки – это процедура, которая получает на вход лог $L \in \mathcal{B}(\mathcal{U}_A^*)$ и помеченную WF-сеть $N \in \mathcal{U}_N$ и выдающая в качестве результата сеть Петри $N^r = \mathit{ModularRepair}(L, N)$, которая идеально соответствует логу L, т.е. $\mathit{fitness}(L, N^r) = 1$.

Модульная схема корректировки имеет четыре процедурных параметра: *eval*, *split*, *repair*, *compose*, где

- $\mathit{eval} \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to [0; 1]$ – функция оценки,
- $\mathit{repair} \in \mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to \mathcal{U}_N$ – функция исправления,
- $\mathit{split} \in \mathcal{U}_N \to \mathcal{P}(\mathcal{U}_N)$ – функция декомпозиции, а $\mathit{compose} \in \mathcal{P}(\mathcal{U}_N) \to \mathcal{U}_N$ – парная ей функция композиции.

Работоспособность базовой схемы

Достаточное условие для модульной схемы исправления

Модульная схема исправления *ModularRepair* обеспечивает идеальное соответствие, если

1. *split* — функция *валидной* декомпозиции;
2. *repair* — *идеальная функция автоматического синтеза* модели, т. е. такая функция, которая для любого лога событий возвращает идеально соответствующую ему сеть Петри;
3. *compose* — функция композиции фрагментов, в которой используется простое слияние переходов с одинаковыми именами.

Доказательство не приводим. Оно содержится в работе [Mitsyuk et al, 2017] и основывается на теоремах о декомпозиции проверки соответствия и автоматического синтеза.

Валидная декомпозиция

Будем называть $D = \{N^{\mathbf{1}}, N^{\mathbf{2}}, \ldots, N^n\} \subseteq \mathcal{U}_{\boldsymbol{N}}$ **валидной декомпозицией**, если (1) $N^i = (P^i, T^i, F^i, l^i)$ — помеченные сети Петри для каждого $1 \leq i \leq n$, (2) $l^i = l\restriction_{T^i}$ для каждого $1 \leq i \leq n$, (3) $P^i \cap P^j = \emptyset$ для $1 \leq i < j \leq n$, (4) $T^i \cap T^j \subseteq T_v^u(\boldsymbol{N})$ для $1 \leq i < j \leq n$, а (5) $\boldsymbol{N} = \bigcup_{\mathbf{1} \leq i \leq n} N^i$.

Наивный алгоритм

Наивный алгоритм

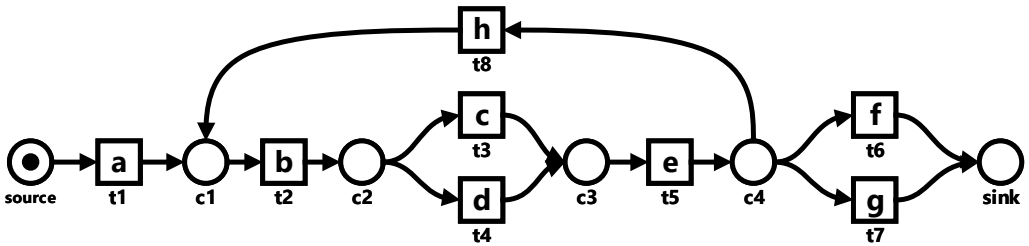
Базовый алгоритм корректировки

В базовом алгоритме корректировки зафиксируем, что

- *eval* — функция оценки соответствия, использующая т.н. «выравнивания» [Adriansyah, 2014],
- *repair* — индуктивный алгоритм автоматического синтеза (Inductive Miner) [Leemans et al, 2014], либо алгоритм, основанный на решении задачи линейного программирования (ILP Miner) [van der Werf et al, 2009], гарантирующие идеальное соответствие,
- *split* — функция т.н. «максимальной» декомпозиции (предложена в [van der Aalst, 2013], наш алгоритм построения приводится в [Mitsyuk et al, 2017]), а *compose* — парная ей функция композиции (слияние переходов с одинаковыми пометками).

Базовый алгоритм удовлетворяет достаточному условию для модульной схемы исправления, так как максимальная декомпозиция валидна (показано в [van der Aalst, 2013]), а потому успешно корректирует модель по соответствию заданному логу.

Пример работы наивного алгоритма — вход

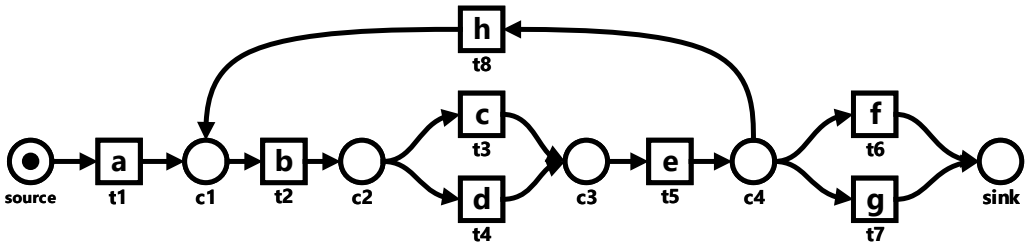


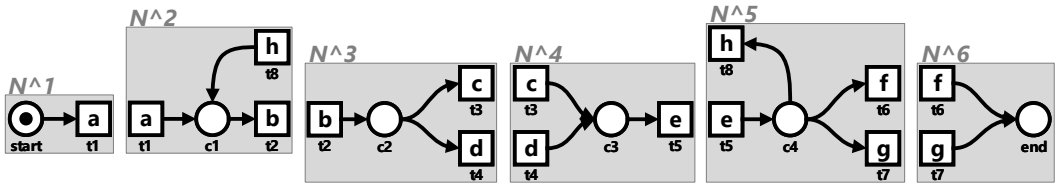
Лог событий $L =$

$[\langle a, b, c, e, g \rangle,$

$\langle a, \mathbf{d}, \mathbf{b}, e, f \rangle].$

Пример работы наивного алгоритма — декомпозиция



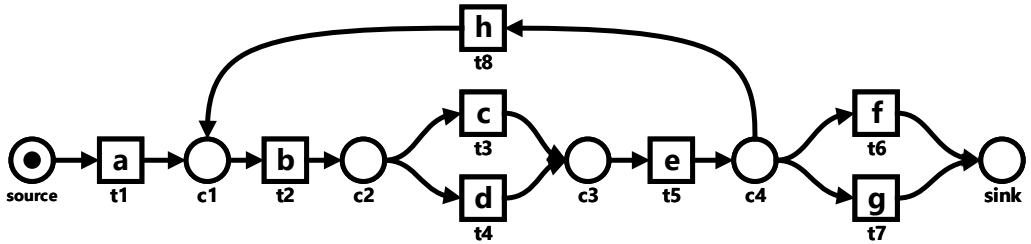


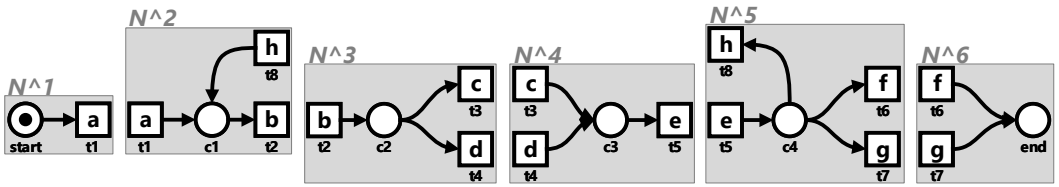
Лог событий $L =$

$[\langle a, b, c, e, g \rangle,$

$\langle a,$ **d** , **b** $, e, f \rangle].$

Пример работы наивного алгоритма — проекция





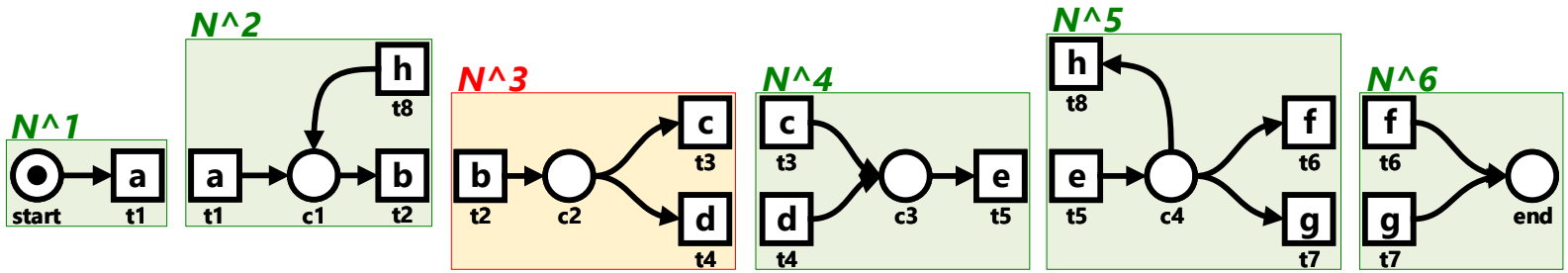
Лог событий $L =$

$[\langle a, b, c, e, g\rangle,$
$\langle a,$ **d** , **b** $, e, f\rangle].$

Множество логов:

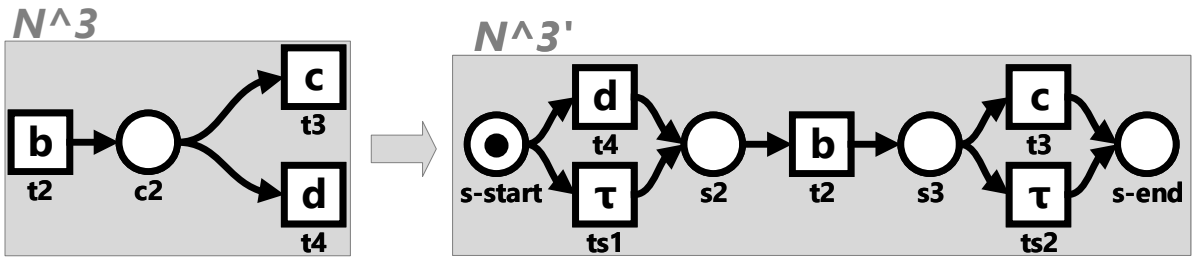
$[\langle a\rangle, [\langle a, b\rangle, [\langle d, b\rangle, [\langle d, e\rangle, [\langle e, f\rangle, [\langle f\rangle,$
$\langle a\rangle] \langle a, b\rangle] \langle b, c\rangle] \langle c, e\rangle] \langle e, g\rangle] \langle g\rangle]$

Пример работы наивного алгоритма — выбор



$$[\langle a\rangle, \langle a\rangle] \quad [\langle a, b\rangle, \langle a, b\rangle] \quad [\langle \mathbf{d}, \mathbf{b}\rangle, \langle b, c\rangle] \quad [\langle d, e\rangle, \langle c, e\rangle] \quad [\langle e, f\rangle, \langle e, g\rangle] \quad [\langle f\rangle, \langle g\rangle]$$

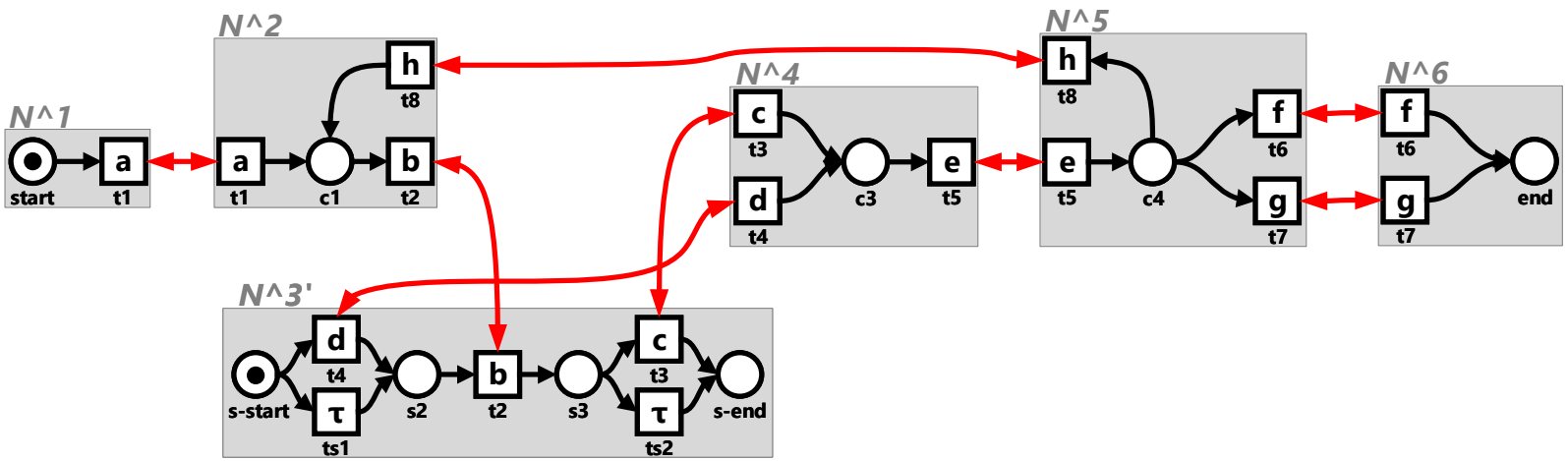
Пример работы наивного алгоритма — исправление



(используем индуктивный алгоритм)

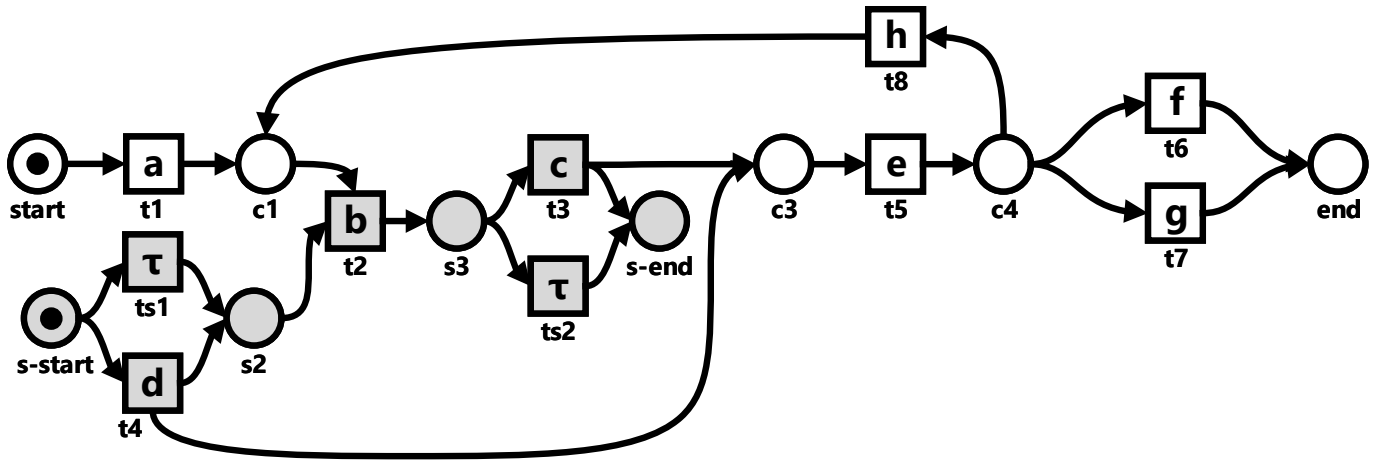
$$[\langle d, b\rangle, \langle b, c\rangle]$$

Пример работы наивного алгоритма — композиция



(используем слияние переходов с одинаковыми метками)

Пример работы наивного алгоритма — результат



Лог событий $L =$

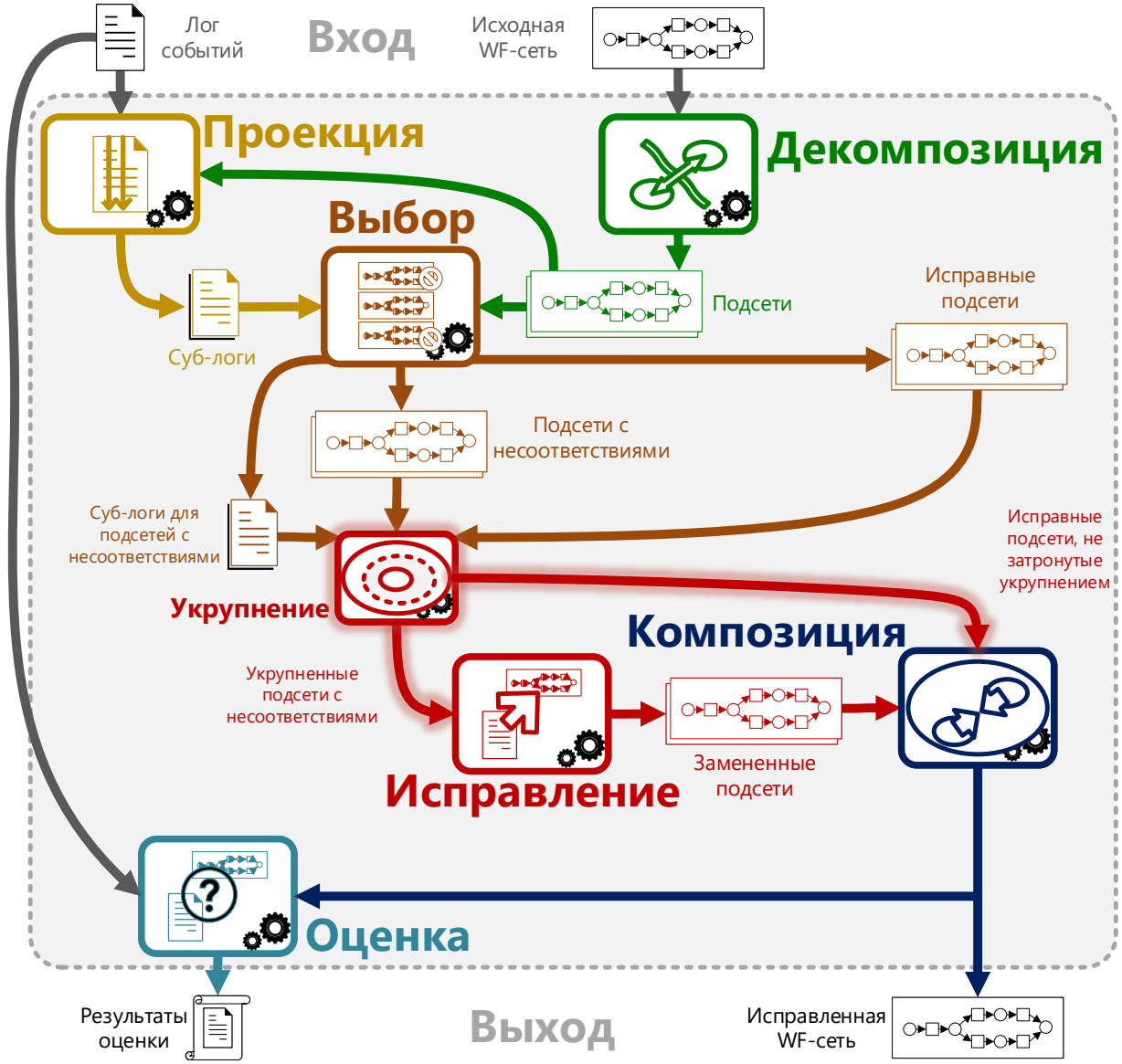
$[\langle a, b, c, e, g \rangle, \langle a, d, b, e, f \rangle]$.

— Соответствие логу L полное, можем его исполнить,
— Не умеет исполнять все трассы, которые умела исходная модель (например, цикл через t_8)
Решение: убрать позиции s_{start} и s_{end} (не влияют на соответствие), тогда сильно падает точность модели по отношению к логу, добавляется много вариантов поведения.

Что делать? Наше решение: «обернуть» подсеть перед исправлением.

Усовершенствованный алгоритм

Усовершенствованный алгоритм – 1



Усовершенствованный алгоритм – 2

Пусть D – декомпозиция сети потоков работ N, и пусть $D = D_b \cup D_c$, где D_b – множество фрагментов сети, не соответствующих своим суб-логам, а D_c – множество остальных фрагментов. Очевидно, что $D_b \cap D_c = \emptyset$.

Процедура укрупнения фрагментов состоит из следующих шагов:

1. Каждой фрагмент из D_b объединяется со своими соседями $\forall \boldsymbol{N}^k \in D_b : D_w^i = \mathcal{W}(\boldsymbol{N}^k)$, $D_n^i = \mathcal{N}(\boldsymbol{N}^k)$; Множество всех укрупненных фрагментов обозначим $D_w = \bigcup_i D_w^i$, а множество всех фрагментов, затронутых процедурой, обозначим $D_n = \bigcup_i D_n^i$; заметим, что множества соседей для каждого фрагмента могут пересекаться;
2. Все фрагменты, затронутые процедурой, удаляются из исходной декомпозиции $D_r = D_c \setminus D_n$;
3. Собирается новая декомпозиция, содержащая б*о*льшие фрагменты: $D_{\boldsymbol{N}}^w = D_w \cup D_r$.

Соседними фрагментами для $\boldsymbol{N}^i$ будем называть фрагменты, в которых имеются переходы, общие с $\boldsymbol{N}^i$. Множество соседей для фрагмента $\boldsymbol{N}^i$ определим так $\mathcal{N}(\boldsymbol{N}^i) = \left\{ \boldsymbol{N}^j \mid \boldsymbol{N}^j \in D_N \wedge T_i \cap T_j \neq \emptyset \right\}$.

Усовершенствованный алгоритм — 3

Слияние двух фрагментов оставляет декомпозицию валидной

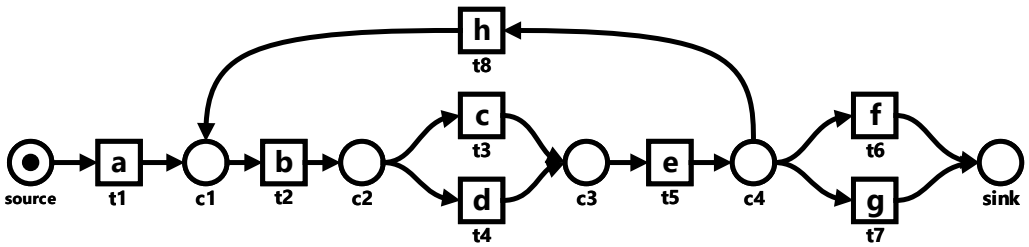
Пусть $D_{\boldsymbol{N}} = \left\{ \boldsymbol{N^1}, \ldots, \boldsymbol{N^n} \right\} \in \mathcal{D}(N)$ — любая валидная декомпозиция сети N. Пусть $D^w_{\boldsymbol{N}}$ — укрупненная декомпозиция, в которой объединены два фрагмента, то есть $\exists i, j$ такие, что $1 \leq i < j \leq n$ и $D^w_{\boldsymbol{N}} = \left\{ N^i \cup N^j \right\} \cup D_{\boldsymbol{N}} \setminus \left\{ N^i, N^j \right\}$. Тогда $D^w_{\boldsymbol{N}}$ — валидная декомпозиция, т. е. $D^w_{\boldsymbol{N}} \in \mathcal{D}(N)$.

Доказательство содержится в нашей работе [Мицюк и др., 2017].

1. Легко показать, что слияние любого количества фрагментов сводится к слиянию пар фрагментов.
2. Процедура укрупнения состоит только лишь в слиянии нескольких фрагментов, значит она не нарушает валидности декомпозиции.
3. Следовательно, усовершенствованный алгоритм удовлетворяет достаточному условию для базовой схемы и может использоваться для корректировки.

Укрупнение фрагментов положительно сказывается на точности итоговой модели (наблюдение на основе экспериментов, формально не доказывали) в случае, если поломка локальна (помещается в укрупненный фрагмент).

Пример работы усов. алгоритма — вход



Лог событий $L =$

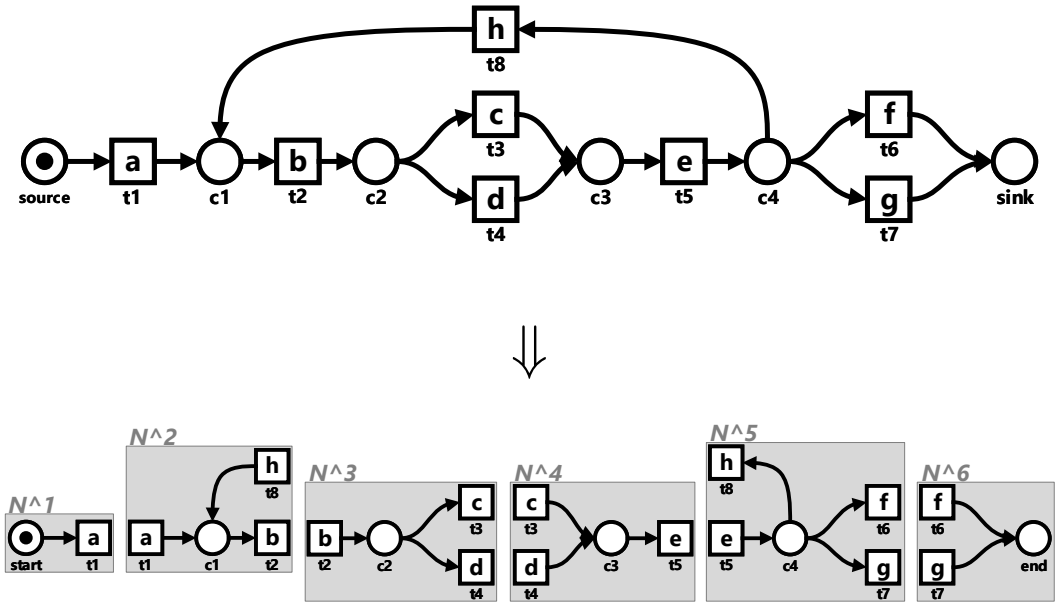
$[\langle a,$ **d** , **b** $, e, f\rangle,$

$\langle a, b, c, e, g\rangle,$

$\left\langle a, b, c, e, h,\right.$ **d** , **b** $\left., e, g\right\rangle,$

$\left\langle a,\right.$ **c** , **b** $\left., e, f\right\rangle].$

Пример работы усов. алгоритма — декомпозиция



Лог событий $L =$

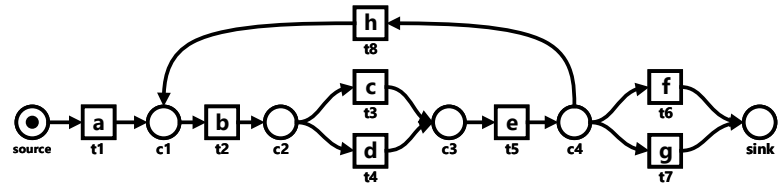
$[\langle a,$ **d** , **b** $, e, f \rangle,$

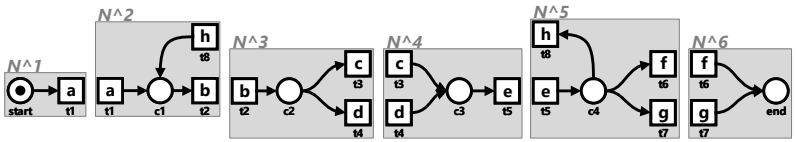
$\langle a, b, c, e, g \rangle,$

$\left\langle a, b, c, e, h,\right.$ **d** , **b** $\left., e, g \right\rangle,$

$\left\langle a,\right.$ **c** , **b** $\left., e, f \right\rangle].$

Пример работы усов. алгоритма — проекция





Лог событий $L =$

$[\langle a, \mathbf{d}, \mathbf{b}, e, f\rangle,$

$\langle a, b, c, e, g\rangle,$

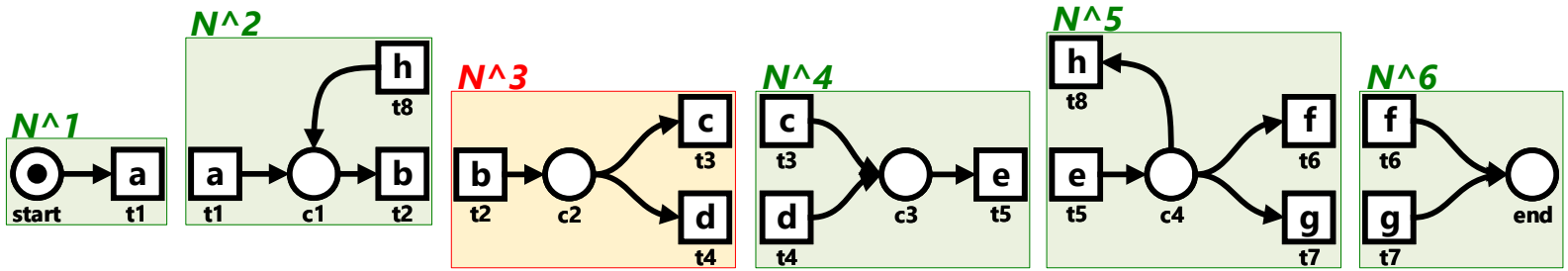
$\left\langle a, b, c, e, h, \mathbf{d}, \mathbf{b}, e, g\right\rangle,$

$\left\langle a, \mathbf{c}, \mathbf{b}, e, f\right\rangle].$

Множество суб-логов:

$[\langle a\rangle,$	$[\langle a, b\rangle,$	$[\langle d, b\rangle,$	$[\langle d, e\rangle,$	$[\langle e, f\rangle,$	$[\langle f\rangle,$
$\langle a\rangle,$	$\langle a, b\rangle,$	$\langle b, c\rangle,$	$\langle c, e\rangle,$	$\langle e, g\rangle,$	$\langle g\rangle,$
$\langle a\rangle,$	$\langle a, b, h, b\rangle,$	$\langle b, c, d, b\rangle,$	$\langle c, e, d, e\rangle,$	$\langle e, h, e, g\rangle,$	$\langle g\rangle,$
$\langle a\rangle]$	$\langle a, b\rangle]$	$\langle c, b\rangle]$	$\langle c, e\rangle]$	$\langle e, f\rangle]$	$\langle f\rangle]$

Пример работы усов. алгоритма — выбор



$[\langle a \rangle, \langle a \rangle, \langle a \rangle, \langle a \rangle]$

$[\langle a, b \rangle, \langle a, b \rangle, \langle a, b, h, b \rangle, \langle a, b \rangle]$

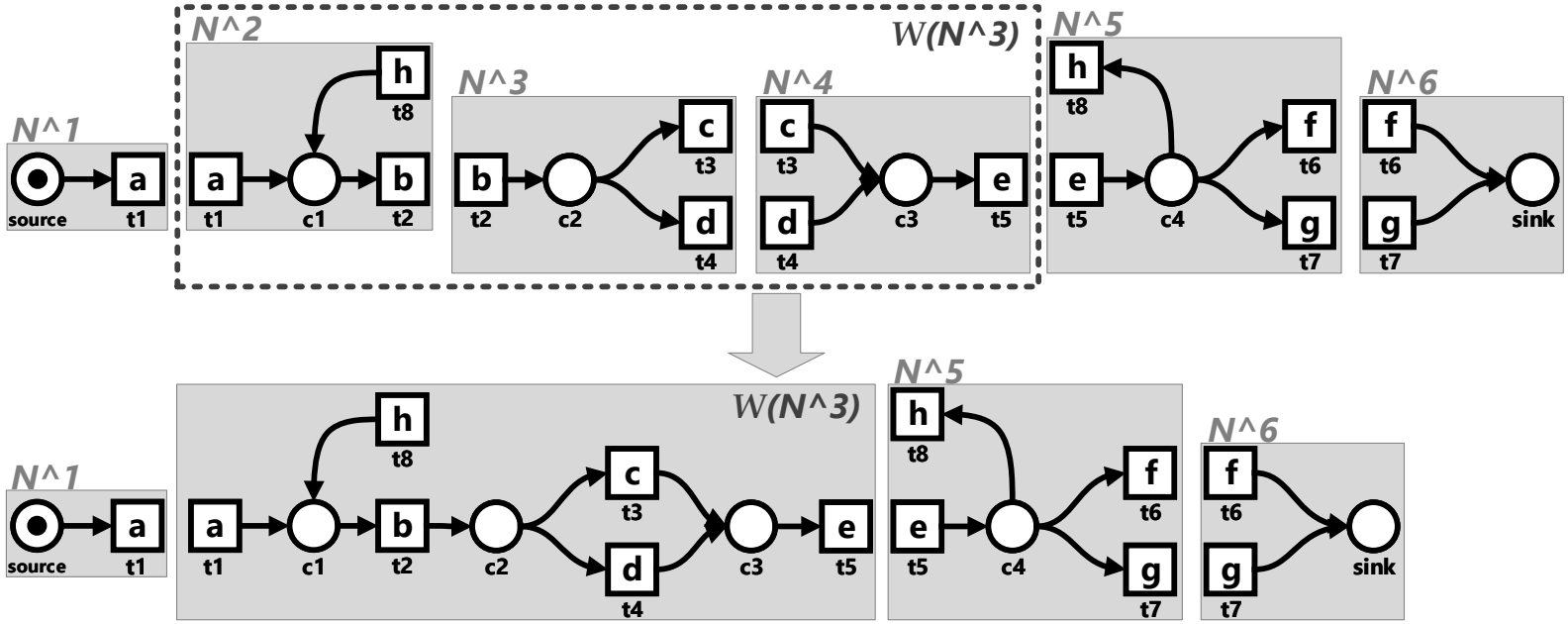
$[\langle \mathbf{d}, \mathbf{b} \rangle, \langle b, c \rangle, \langle b, c, \mathbf{d}, \mathbf{b} \rangle, \langle \mathbf{c}, \mathbf{b} \rangle]$

$[\langle d, e \rangle, \langle c, e \rangle, \langle c, e, d, e \rangle, \langle c, e \rangle]$

$[\langle e, f \rangle, \langle e, g \rangle, \langle e, h, e, g \rangle, \langle e, f \rangle]$

$[\langle f \rangle, \langle g \rangle, \langle g \rangle, \langle f \rangle]$

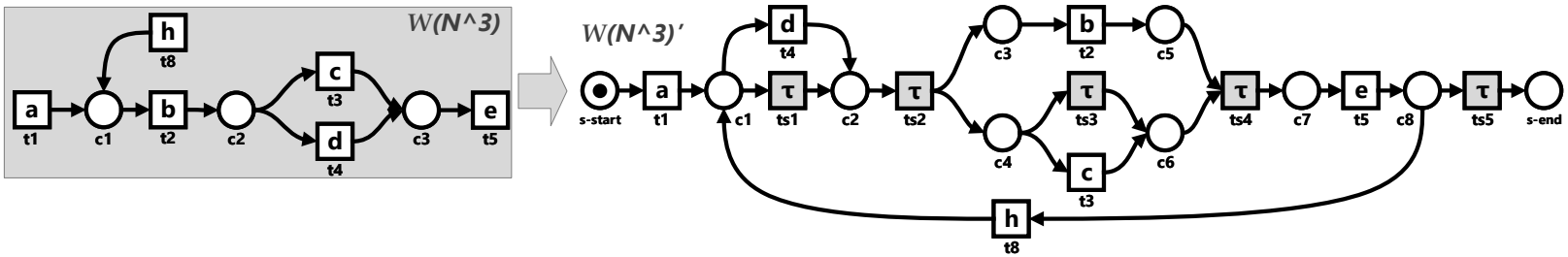
Пример работы усов. алгоритма — укрупнение



Суб-лог для подсети $\mathcal{W}(N^3)$:

$[\langle a, d, b, e\rangle,$
$\langle a, b, c, e\rangle,$
$\langle a, b, c, e, h, d, b, e\rangle,$
$\langle a, c, b, e\rangle]$.

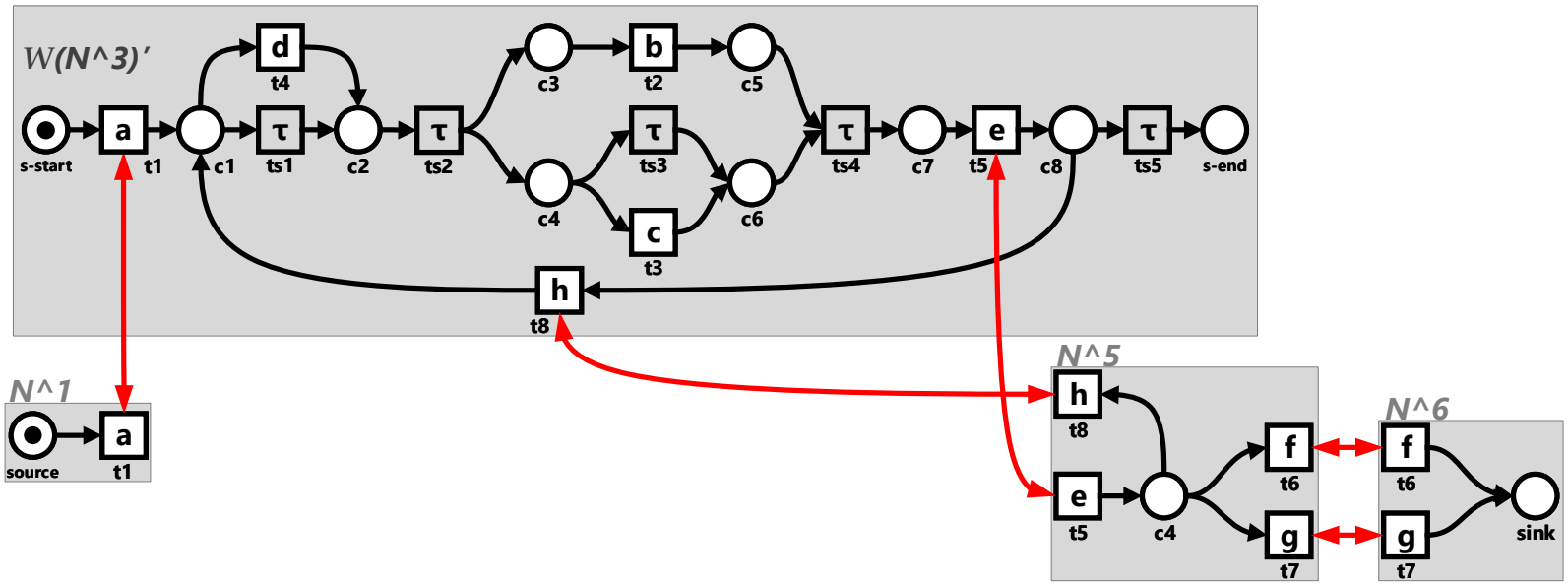
Пример работы усов. алгоритма — исправление



(используем индуктивный алгоритм)

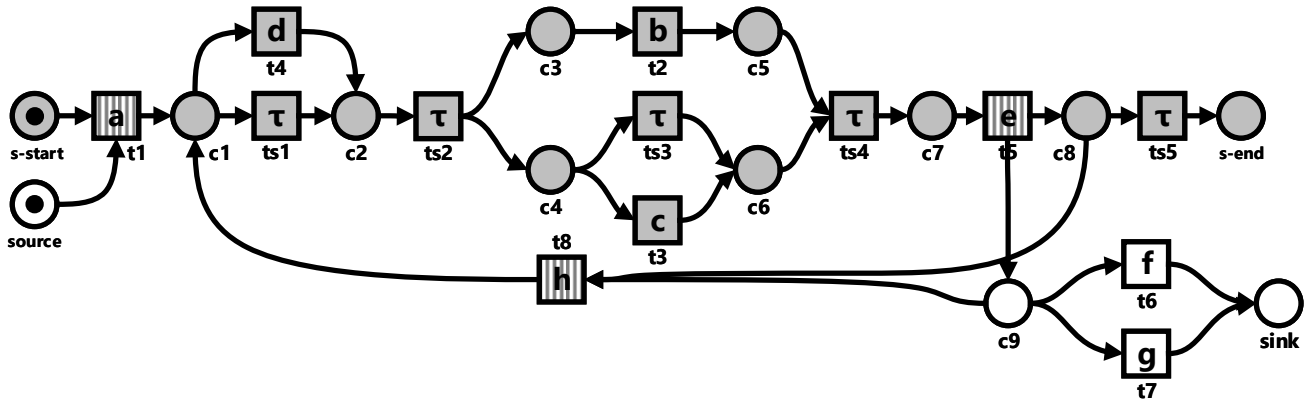
$$[\langle a, d, b, e\rangle,$$
$$\langle a, b, c, e\rangle,$$
$$\langle a, b, c, e, h, d, b, e\rangle,$$
$$\langle a, c, b, e\rangle].$$

Пример работы усов. алгоритма — композиция



(используем слияние переходов с одинаковыми метками)

Пример работы усов. алгоритма — результат



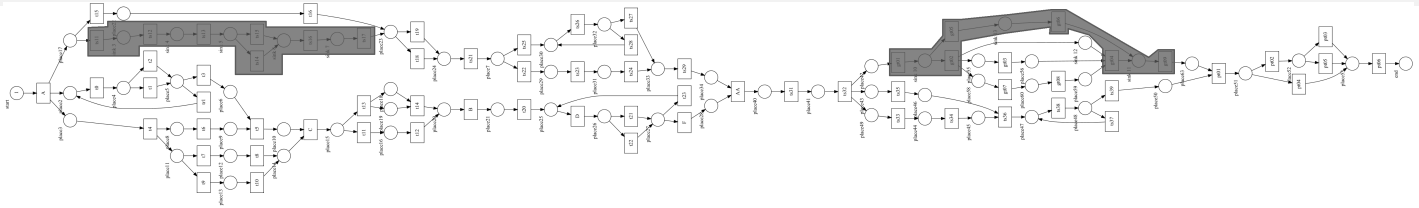
Лог событий $L =$

$[\langle a, d, b, e, f \rangle,$
$\langle a, b, c, e, g \rangle,$
$\langle a, b, c, e, h, d, b, e, g \rangle,$

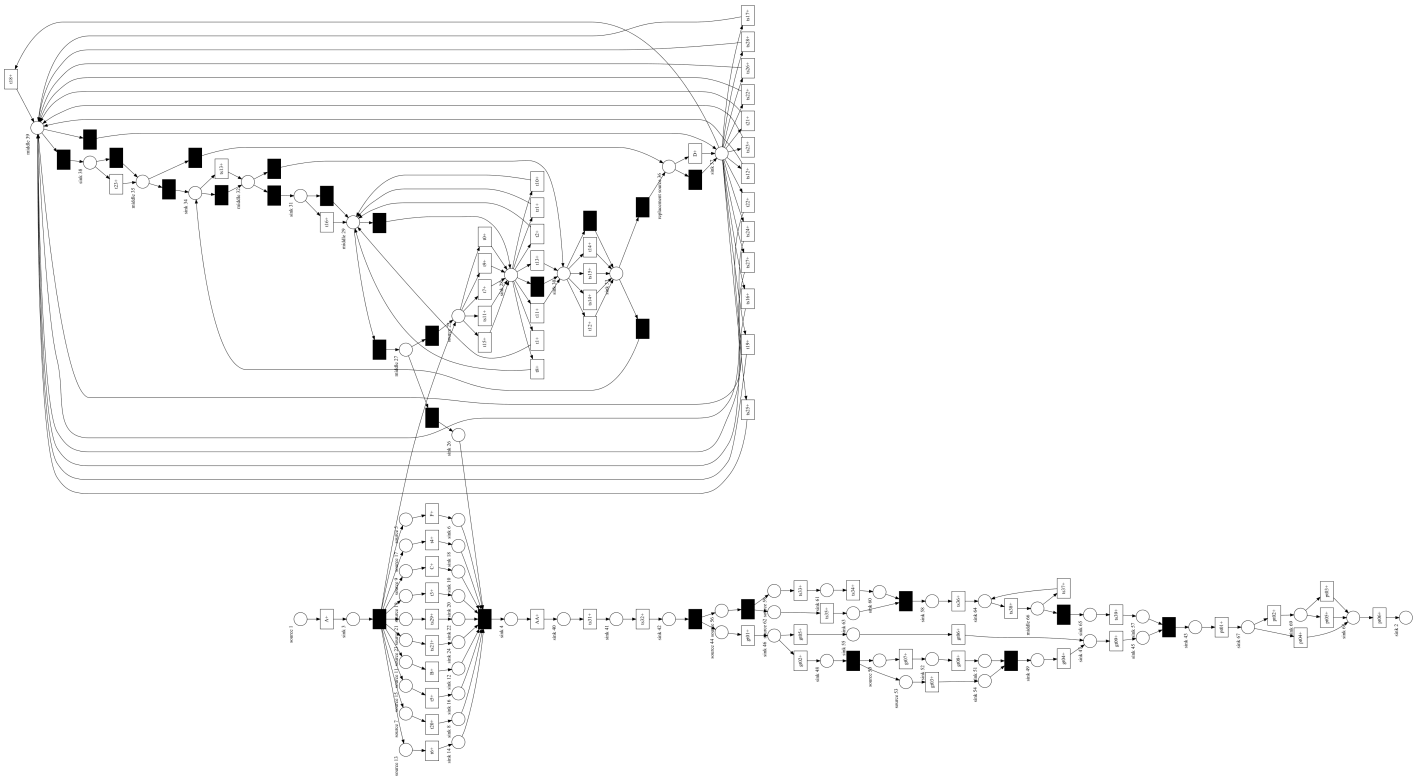
$\langle a, c, b, e, f \rangle]$.

– Соответствие логу L полное, можем его исполнить,
– *Кажется*, что умеет исполнять все трассы, которые умела исходная модель (нет доказательства, догадка)
– Точность модели соответствует точности исходной модели (см. далее, экспериментальные результаты)
– Не работает в *нелокальном* случае

Пример работы усов. алгоритма — результат 2



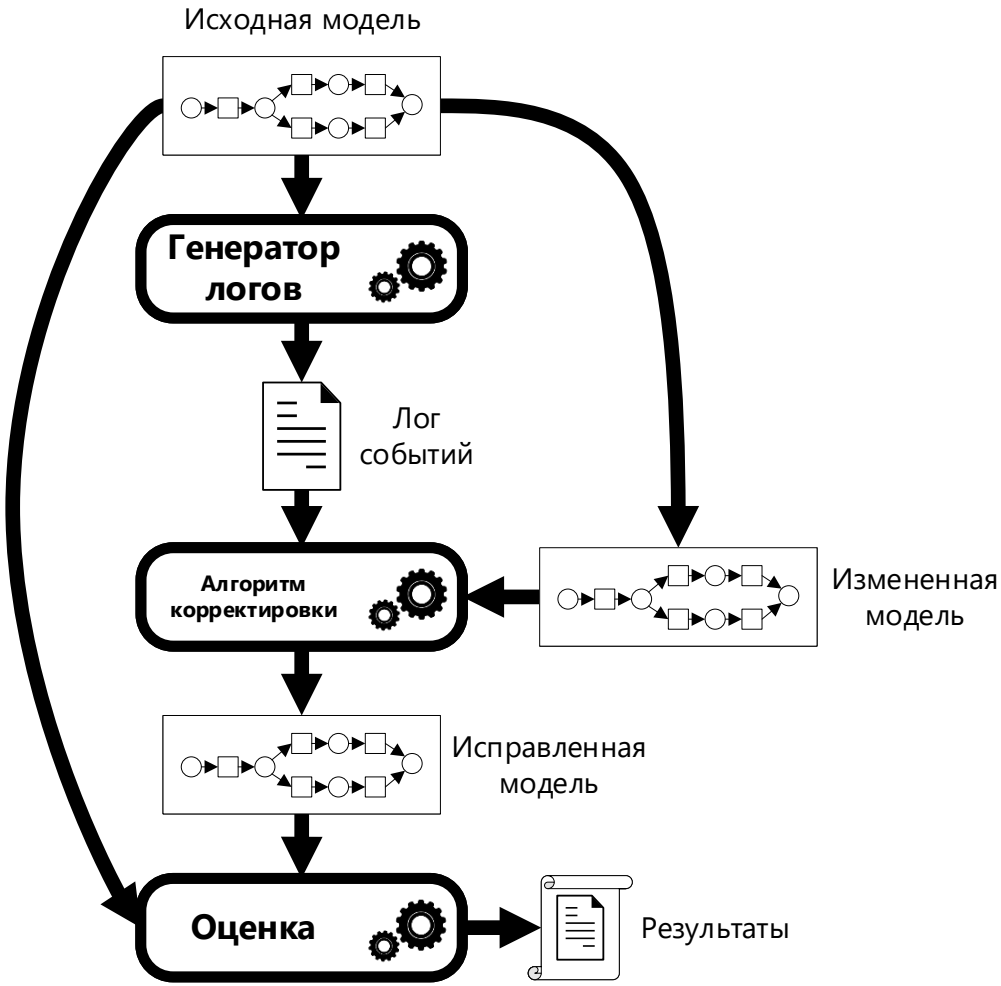
a) Усовершенствованный алгоритм корректировки



b) Модель, синтезированная по логу с помощью индуктивного алгоритма

(модель большего размера)

Схема тестирования



Устройство тестовой реализации алгоритмов описано в [Shugurov, Mitsyuk, 2015].

Результаты тестирования

Модель	Метод	Алгоритм синтеза	Соотв.	Точн.	Сходство	N-f	N-bf	Размер	Изм.	Время
SM1	Исходная модель		1	0,91	-	-	-	40	-	-
	Синтез новой модели	Ind	1	0,91	0,51	-	-	47	47	829
		ILP	1	0,91	0,57	-	-	38	38	10069
SM1-BL-1	Измененная модель		0,94	0,86	-	-	-	40	-	-
	Наивный	Ind	1	0,57	0,97	19	1	40	3	2531
		ILP	1	0,57	0,97	19	1	40	3	2531
	Улучшенный	Ind	1	0,91	0,95	19	1	40	7	2365
		ILP	1	0,91	0,95	19	1	40	7	2842
SM1-BL-2	Измененная модель		0,89	0,89	-	-	-	40	-	-
	Наивный	Ind	1	f	0,89	19	3	45	9	2983
		ILP	1	0,5	0,94	19	3	39	9	3215
	Улучшенный	Ind	1	0,91	0,82	19	1	47	15	2426
		ILP	1	0,91	0,88	19	1	41	15	4030
LM2	Исходная модель		1	0,59	-	-	-	133	-	-
	Синтез новой модели	Ind	1	f	f	-	-	166	166	2920
		ILP	1	0,53	f	-	-	137	137	1816898
LM2-BL-1	Измененная модель		0,98	0,59	-	-	-	133	-	-
	Наивный	Ind	1	0,28	f	63	2	133	7	10745
		ILP	1	0,28	f	63	2	133	7	9588
	Улучшенный	Ind	1	0,59	f	63	1	133	12	9205
		ILP	1	0,59	f	63	1	133	12	10339
LM2-BL-2	Измененная модель		0,99	0,59	-	-	-	133	-	-
	Наивный	Ind	1	0,38	f	63	1	133	3	8048
		ILP	1	0,38	f	63	1	133	3	12847
	Улучшенный	Ind	1	0,59	1	63	1	133	8	12960
		ILP	1	0,59	1	63	1	133	8	8619
LM2-BL-3	Измененная модель		0,97	0,59	-	-	-	133	-	-
	Наивный	Ind	1	0,23	f	63	3	133	10	8097
		ILP	1	0,23	f	63	3	133	10	8855
	Улучшенный	Ind	1	0,59	f	63	2	134	21	9149
		ILP	1	0,59	f	63	2	134	21	12410

Результаты опубликованы в этих работах

Мицюк А.А., Ломазова И.А., ван дер Аалст В.М.П.

Использование журналов событий для локальной корректировки моделей процессов

Моделирование и анализ информационных систем, Т. 24. ■ 4. С. 459-480, 2017

[в печати] A. A. Mitsyuk, I. A. Lomazova, I. S. Shugurov, and W. M. P. van der Aalst

Process Model Repair by Detecting Unfitting Fragments

Supplementary Proceedings of AIST 2017, CEUR-WS.org, 2017

I. Shugurov, A. Mitsyuk.

Iskra: A Tool for Process Model Repair

Proceedings of the Institute for System Programming. 2015. Vol. 27. No. 3. P. 237-254.

Shugurov I., Mitsyuk A. A.

Generation of a Set of Event Logs with Noise

Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2014). M. : -, 2014. P. 88-95.

Литература — 1

Adriansyah A.

Aligning observed and modeled behavior
Ph.D. Thesis, Technische Universiteit Eindhoven, 2014

Leemans S.J.J., Fahland D., van der Aalst W.M.P.

Discovering Block-Structured Process Models from Incomplete Event Logs
Application and Theory of Petri Nets and Concurrency, Lecture Notes in Computer Science, 8489 (2014), 91–110

van der Werf J.M.E.M., van Dongen B.F., Hurkens C.A.J., Serebrenik A.

Process Discovery using Integer Linear Programming
Fundam. Inform., 94 (2009), 387–412

van der Aalst W.M.P.

Decomposing Petri Nets for Process Mining: A Generic Approach
Distributed and Parallel Databases, 31:4 (2013), 471–507

van der Aalst W.M.P.

Process Mining — Data Science in Action, Second Edition
Springer, 2016

Buijs J.C.A.M., La Rosa M., Reijers H.A., van Dongen B.F., van der Aalst W.M.P.

Improving Business Process Models Using Observed Behavior
SIMPDA 2012, LNBIP, 162 (2013), 44–59

Fahland D., van der Aalst W.M.P.

Model repair - Aligning process models to reality
Inf. Syst., 47 (2015), 220–243

Литература — 2

Gambini M., La Rosa M., Migliorini S., Ter Hofstede A.H.M.

Automated Error Correction of Business Process Models

9th International Conference Business Process Management (BPM 2011), Lecture Notes in Computer Science, 6896 (2011), 148–165

Polyvyanyy A., van der Aalst W.M.P., ter Hofstede A.H.M., Wynn M.T.

Impact-driven process model repair

ACM Transactions on Software Engineering and Methodology (TOSEM), 25:4 (2017), 28:1–28:60

Dijkman R.M., Dumas M., Garcia-Banuelos L.

Graph matching algorithms for business process model similarity search

BPM, Lecture Notes in Computer Science, 5701 (2009), 48–63

Armas-Cervantes, Abel, La Rosa, Marcello, Dumas Menjivar, Marlon, García-Bañuelos, Luciano, van Beest, Nick R.

Interactive and incremental business process model repair

25th International Conference On Cooperative Information Systems (CoopIS 2017), 25-27 October 2017, Rhodes, Greece (Unpublished)

Спасибо за внимание!