A PRELIMINARIES

A.1 Data model

We introduce thereafter further basic notation around the data model of a property graph.

A *path* $p \coloneqq (n_0, e_1, n_1, \ldots, e_k, n_k)$ is an alternating sequence of node and edges, which starts and ends with nodes. Given a path p we denote by $\mathsf{src}(p)$ and $\mathsf{tgt}(p)$ the first and last node of p; in this case, $\mathsf{src}(p) = n_0$ and $\mathsf{tgt}(p) = n_k$. $\mathsf{len}(p) \in \mathbb{N}$ is the *length* of p, i.e, the number of edges in the path; and if $\mathsf{len}(p) = 0$ then the path consists of a single node which is both the source and the target. We can define as usual the *concatenation* $p \cdot p'$ of two paths p and p' whenever $\mathsf{tgt}(p) = \mathsf{src}(p')$.

A.2 Graph Pattern Calculus

We make a brief summary on the syntax and semantics of GPC [19], focusing only on the concepts we need to formally define our property graph transformation rules.

The *atomic* GPC patterns are node and edge patterns. A *node pattern* is of the form $(x : \ell)$ and an *edge pattern* is of the form $\overset{x:\ell}{=\!=}$. In both cases x is an optional variable (picked from a countably infinite set $\mathcal{X}$ of variables) which bounds, if present, to the matched element and ℓ is an optional label indicating that we want to restrict to ℓ-elements. In an edge pattern $=$ may indicate one of the two possible directions: forward $\rightarrow$ and backward $\leftarrow$. A GPC *pattern* denoted π is inductively constructed on top of the atomic patterns by using arbitrarily many *union* $(\pi + \pi)$, *concatenation* $(\pi \cdot \pi)$, *conditioning* $(\pi_{\langle\theta\rangle})$, and *repetition* $(\pi^{n..m})$ constructs.

A GPC *query* is of the form $\rho\,\pi$ with ρ a *restrictor* among the set of simple, trail, shortest, which purpose is to ensure a finite result set. simple prevents repetition of nodes along a path, trail prevents repetition of edges and shortest selects only the paths of minimal length among all the paths between two nodes.

The structure of a GPC query can be inspected using a *type system*, a set of typing rules [19]. A query is *well-typed* if the rules permit to deduce a unique type to every variable appearing in the query. When an expression Q is well-typed, the schema $\mathsf{sch}(Q)$ of this expression associates a type to each variable.

The *answer* of a GPC query $Q(\bar{x})$ on a property graph G, denoted $[\![Q]\!]_G^{\bar{x}}$ is a set of assignments. An *assignment* binds the variables $\bar{x}$, present in the query, to values. *Values* to be associated to variables are dependent upon the deduced type of the variable for that query. Hence, for each type τ, there is a set of values $\mathcal{V}_\tau$. Values may be references to elements in the graph, e.g., for Node and Edge types.

All answers to queries we need to define our property graph transformations will have assignments of the variables among the types Node and Edge. In our framework, we will use GPC queries extended with the capability to use conditioning on top of joins. This is not part of the specification in [19], but this is planned to be in GQL [20].

B PROPERTY GRAPH TRANSFORMATIONS

We provide additional notation and definitions that are used in the main proofs of this paper.

B.1 Generating output identifiers

Given a tuple of m variables $\bar{x} = (x_1, \ldots, x_m)$, we define the sets of *value arguments* and *arguments* for $\bar{x}$ as

$$\mathcal{V}_{\bar{x}} ::= c \mid x_i.a$$

$$\mathcal{A}_{\bar{x}} ::= x_i \mid c \mid \ell \mid x_i.a$$

where $x_i \in \bar{x}$, $c \in \mathsf{Const}$, $\ell \in \mathcal{L}$ and $a \in \mathcal{K}$. We denote by $\mathcal{A}_{\bar{x}}^k$ the set of all tuples of arguments for $\bar{x}$ of length k.

For a given property graph G, $A = (a_1, \ldots, a_k)$ a tuple of arguments for $\bar{x}$ defines a function $O^m \to (O \cup \mathsf{Const} \cup \mathcal{L})^k$ defined as $(o_1, \ldots, o_m) \mapsto (v_1, \ldots v_k)$ where, for all $1 \leq i \leq k$:

- $v_i \coloneqq o_j$ if $a_i = x_j$;
- $v_i \coloneqq c$ if $a_i = c$;
- $v_i \coloneqq \ell$ if $a_i = \ell$;
- $v_i \coloneqq c$ if $a_i = x_j.a$ and $\delta_G(o_j, a) = c$.

B.2 Semantics

Proposition 3.6. *Given an input property graph G and a property graph transformation T, Algorithm 1 always returns a valid instance of the property graph data model.*

Proof. Given T and G with identifiers from $\mathcal{S}$, let $T(G) \coloneqq \langle N, E, \lambda, \mathsf{src}, \mathsf{tgt}, \delta\rangle$ be a property graph returned by Algorithm 1. We have to check that (i) both N and E are finite and disjoint, (ii) all elements have a finite number of labels, and (iii) every edge has exactly one source and one target.

(i) The set of bindings resulting from querying a property graph G with the query $P(\bar{x})$, $[\![P]\!]_G^{\bar{x}}$, is assumed to be finite, moreover, we have a finite number of rules in T, hence the finiteness of $N \cup E$. A similar reasoning shows the finiteness of the label set for each element in $T(G)$; this is because each rule can mention at most a finite number of labels.

(ii) We now show that $N \cap E = \emptyset$. Let us assume that $o \in \mathcal{T}$ is both a node and an edge id in $T(G)$ – respectively resulting from a node rule $R := P(\bar{x}) \implies (D)$ for $\bar{o}$ and an edge rule $S := Q(\bar{y}) \implies (C_\mathrm{s}) \xrightarrow{C} (C_\mathrm{t})$ for $\bar{p}$. The *injectivity* of the Skolem function f enforces that o has been generated, in both cases, by using the same arguments. Moreover, by *injectivity* again, we necessarily have $D.\mathsf{Id}(\bar{o}) = (o_\mathrm{s}, C.\mathsf{Id}(\bar{p}), o_\mathrm{t})$ for some $o_\mathrm{s}, o_\mathrm{t} \in N$, with $C.\mathrm{Id} \in \mathcal{A}_{\bar{x}}^{k-2}$. (Note that o_s and o_t have been respectively obtained from the source and target rules of S for $\bar{p}$.) By definition of the range of the Skolem function f, o_s and o_t belong to $\mathcal{T}$; thus, they could not be equal to the first and last values of $D.\mathrm{Id}$. We conclude that $N \cap E = \emptyset$.

(iii) Finally, by *injectivity* of f, for an $o \in O$ which is an edge id in $T(G)$, there are, by definition, exactly one $o_\mathrm{s} \in N$ and one $o_\mathrm{t} \in N$ which correspond to the source and the target of this edge. □

C DETECTING CONFLICTS

We formally define the notion of *conflicts* and provide the proofs for the results in Section 4.

C.1 Consistency checking

We now formalize the notion of *node conflict*. For any $k \geq 0$, let

$$R := P(\bar{x}) \implies \underset{\langle \dots, a=v, \dots \rangle}{(A : L)} \text{ and } S := Q(\bar{y}) \implies \underset{\langle \dots, a=w, \dots \rangle}{(B : M)}$$

be two node rules in T with $L, M \in \mathcal{P}_{fin}(\mathcal{L})$. These two *node* rules are *potentially conflicting* on the property a when:

- their respective argument lists have same length, i.e., $A = (a_1, \dots, a_k) \in \mathcal{A}_{\bar{x}}^k$ and $B = (b_1, \dots, b_k) \in \mathcal{A}_{\bar{y}}^k$ for a $k \geq 0$;
- their argument lists are *compatible*, which means that for each $1 \leq i \leq k$, a_i is a value argument if and only if b_i is a value argument;
- if a_i and b_i are respectively x_i and y_j, then $\mathsf{sch}(P)(x_i)$ should be equal to $\mathsf{sch}(P)(y_j)$;
- they have *potentially conflicting properties*, which means that they both define the same property to (possibly) different values, i.e., $a \in \mathcal{K}, v \in \mathcal{V}_{\bar{x}}$ and $w \in \mathcal{V}_{\bar{y}}$.

A *node conflict* for a pair of possibly conflicting rules on the property a *occurs* in a property graph G whenever it exists $\bar{o} \in [\![P]\!]_G^{\bar{x}}$ and $\bar{p} \in [\![Q]\!]_G^{\bar{y}}$ with $A(\bar{o}) = B(\bar{p})$ and $v(\bar{o}) \neq w(\bar{p})$.

We now formalize the notion of *edge conflict*. For any $k \geq 0$, let

$$R := P(\bar{x}) \implies (A_\mathrm{s} :) \underset{\langle \dots, a=v, \dots \rangle}{\left[\xrightarrow{A:L}\right]} (A_\mathrm{t} :) \text{ and } S := Q(\bar{y}) \implies (B_\mathrm{s} :) \underset{\langle \dots, a=w, \dots \rangle}{\left[\xrightarrow{B:M}\right]} (B_\mathrm{t} :)$$

be two edge rules in T with $L, M \in \mathcal{P}_{fin}(\mathcal{L})$. These two *edge* rules are *potentially conflicting* on the property a when:

- their respective argument lists have same length, i.e., $A = (a_1, \dots, a_k) \in \mathcal{A}_{\bar{x}}^k$ and $B = (b_1, \dots, b_k) \in \mathcal{A}_{\bar{y}}^k$ for a $k \geq 0$;
- their argument lists are *compatible*, which means that for each $1 \leq i \leq k$, a_i is a value argument if and only if b_i is a value argument;
- if a_i and b_i are respectively x_i and y_j, then $\mathsf{sch}(P)(x_i)$ should be equal to $\mathsf{sch}(P)(y_j)$;
- the three previous points also apply to the pairs $(A_\mathrm{s}, A_\mathrm{t})$ and $(B_\mathrm{s}, B_\mathrm{t})$;
- they have *potentially conflicting properties*, which means that they both define the same property to (possibly) different values, i.e., $a \in \mathcal{K}, v \in \mathcal{V}_{\bar{x}}$ and $w \in \mathcal{V}_{\bar{y}}$.

An *edge conflict* for a pair of possibly conflicting rules on the property a *occurs* in a property graph G whenever it exists $\bar{o} \in [\![P]\!]_G^{\bar{x}}$ and $\bar{p} \in [\![Q]\!]_G^{\bar{y}}$ with $A(\bar{o}) = B(\bar{p})$, $A_\mathrm{s}(\bar{o}) = B_\mathrm{s}(\bar{p})$, $A_\mathrm{t}(\bar{o}) = B_\mathrm{t}(\bar{p})$ and $v(\bar{o}) \neq w(\bar{p})$.

Lemma 4.3. *The transformation consistency problem is PTime-reducible to the satisfiability problem for GPC+.*

Proof. Recall that a node conflict for a pair of possibly conflicting rules R and S on a property a occurs on a property graph G whenever it exists $\bar{o} \in [\![P]\!]_G^{\bar{x}}$ and $\bar{p} \in [\![Q]\!]_G^{\bar{y}}$ with $A(\bar{o}) = B(\bar{p})$ and $v(\bar{o}) \neq w(\bar{p})$. We can rewrite all of those conditions in a single boolean GPC query:

$$Q_{(R,S,a)}() := P(\bar{x}), Q(\bar{y}), A = B, v \neq w$$

which is satisfiable on a property graph H *iff* this specific conflict occurs on H. Note that we assume w.l.o.g. in the following construction that $\bar{x}$ and $\bar{y}$ are disjoint sets of variables.

Similarly, for an edge conflict, we obtain a single boolean GPC query with the same properties:

$$Q_{(R,S,a)}() := P(\bar{x}), Q(\bar{y}), A = B, A_\mathrm{s} = B_\mathrm{s}, A_\mathrm{t} = B_\mathrm{t}, v \neq w$$

We provide an example of the GPC pattern encoding $v \neq w$. Let assume that $v := x_i.b$ and $w := y_j.c$ and $\mathsf{sch}(P)(x_i) = \mathsf{Node}$ and $\mathsf{sch}(Q)(y_j) = \mathsf{Edge}$, the following join query encodes $v \neq w$:

$$\left[(x_i), () \xrightarrow{y_j} () \right]_{\langle \neg(x_i.b = y_j.c) \rangle}.$$

Similarly, we can apply point-wise this construction and take their join to encode $A = B, A_{\mathrm{s}} = B_{\mathrm{s}}$ and $A_{\mathrm{t}} = B_{\mathrm{t}}$.

Finally, to wrap-up the proof, it is easy to see that given a property graph transformation T, there are at most polynomially many such triplets (R, S, a) satisfying this criteria, so we can take the union of all the $Q_{(R,S,a)}()$ as the final GPC+ query on which to check for satisfiability. □

C.2 GPC satisfiability

Lemma 4.4. *The satisfiability problem for GPC is PSpace-hard.*

Proof. Let $M = (Q, \Sigma, s, F, \delta)$ be the TM that recognizes L in deterministic polynomial-space. Let w be an input word of length n. Assume that M works over w using at most $c \cdot p(n)$ for a fixed constant c and polynomial p tape cells.

We build a GPC query using the following set of properties $\mathcal{K}_0 \subset \mathcal{K}$ which contains all the following elements:

- $pos_{(i,\sigma)}$; the tape contains symbol $\sigma \in \Sigma$ at position $1 \leq i \leq c \cdot p(n)$;
- $head_{(i)}$; the head of the TM is in position $1 \leq i \leq c \cdot p(n)$;
- q; the TM is in state $q \in Q$.

Notice that we will be using only two constants values: 0 and 1.

To encode the consistency of a state represented by the set of properties of an element x, we will use formula $\theta_c(x)$ defined as the conjunction of the following formulas:

- $\neg \left(x.pos_{(i,a)} = 1\right) \vee \neg \left(x.pos_{(i,b)} = 1\right)$; for $1 \leq i \leq c \cdot p(n)$, $a, b \in \Sigma$, $a \neq b$;
- $\neg \left(x.head_{(i)} = 1\right) \vee \neg \left(x.head_{(j)} = 1\right)$; for $1 \leq i \neq j \leq c \cdot p(n)$;
- $\neg (x.q = 1) \vee \neg (x.q' = 1)$; for $q, q' \in Q, q \neq q'$
- $(x.k = 1) \vee (x.k = 0)$; for $k \in \mathcal{K}_0$.

$\theta_1(x)$ is a conjunction of the following formulas; it ensures that the set of properties of the element pointed to by x encodes the initial configuration of the TM M over w:

- $x.pos_{(i,a)} = 1$; if $w_i = a$, for $1 \leq i \leq n$; w is stored at the beginning of the tape;
- $x.pos_{(i,\square)} = 1$; for $n < i \leq c \cdot p(n)$, where $\square$ is the blank symbol; the rest of the tape is filled with blank symbols;
- $\left(x.head_{(1)} = 1\right) \wedge (x.s = 1)$; initialisation of both head and state;
- $\theta_c(x)$; consistency check.

$\theta_2(u, v)$ checks that the configuration stored in the record of v can be obtained from u in a single computation step of M; it consists in the conjunction of the following formulas:

- $\left(u.head_{(i)} = 0 \wedge u.pos_{(i,a)} = 1\right) \implies v.pos_{(i,a)} = 1$; for $1 \leq i \leq c \cdot p(n)$, $a \in \Sigma$; the tape remains unchanged unless written by head;
- $\left(u.head_{(i)} = 1 \wedge u.pos_{(i,a)} = 1 \wedge u.q = 1\right) \implies \left(v.pos_{(i,b)} = 1 \wedge v.head_{(i+d)} = 1 \wedge v.q' = 1\right)$; for $1 \leq i \leq c \cdot p(n)$, $a, b \in \Sigma$, $q, q' \in Q$, $d \in \{-1, 0, 1\}$, when $(q', b, d) \in \delta(q, a)$; there is a valid transition;
- $\theta_c(u) \wedge \theta_c(v)$; consistency checks.

Finally, θ_3 checks that we have reached an accepting configuration:

$$\theta_3(y) := \bigvee_{q \in F} y.q = 1$$

It is clear that the pattern P – which is constructed in polynomial time given an entry word w – is satisfiable if and only if there exists an accepting run for M over w using at most a polynomial amount of space, i.e., *iff* $w \in L \in$ PSpace. □

Theorem 4.5. *The satisfiability problem for GPC+ queries using only the* simple *and* trail *restrictors is PSpace-complete.*

Proof. By Lemma 4.4, it is only left to prove the upper-bound. Let Q be a GPC query. We assume a mild syntactic restriction that, all edge and node patterns must mention a variable in their descriptor. This can be achieved by picking a fresh variable when none is specified in a descriptor.

We note respectively Const_0, $\mathcal{K}_0$ and $\mathcal{L}_0$, the set of constants, keys and labels mentioned in Q. Additionally, let d be the number of occurences of conditions of the form $x.a = c$ or $x.a = y.b$ in the formula. We extend the set Const_0 with $d + 1$ fresh distinct constants. Let also $\mathcal{X}$ and $\mathcal{Y}$ be respectively the sets of variables of type Node and Edge in the schema of Q.

Preliminary remarks. We start with some key observations:

- It is clear that we cannot always guess an answer made up of a path and an assignment because some patterns are only satisfiable by paths of exponential length: e.g., we can simulate a counter to count up to 2^n with n properties and a polynomial-sized formula similar to that used in Lemma 4.4;
- The concatenation of patterns implies an implicit equality over the endpoints; thus, we need to book-keep the endpoints of a pattern alongside an assignment for it;
- The semantics and the typing rules of GPC isolate the variables under a repetition pattern; this means that, in a repetition pattern, we cannot refer to externally defined variables (i.e., non-local occurrences are not permitted); moreover, because conditions are only defined over *singleton* variables (i.e., variables of type Node or of type Edge in the schema of Q), we cannot refer in conditions to variables appearing under a repetition sub-pattern. Thus, we can assume that, if a query is satisfiable, an answer path for a pattern inside a repetition pattern can be disjoint (except for its endpoints) from the answer paths for the outside of the repetition pattern.

This shows that we can store in polynomial space an *extended* assignment corresponding to a valid answer to a path pattern π – by extended, we mean that we additionally store the two endpoints of the answer path in variables named src_π and tgt_π (that we assume to belong to $\mathcal{X}$); additionally, Group and Maybe variables will not be tracked because they can no longer be mentioned in conditions. In this case, an *assignment* $\eta(\cdot)$ stores for each variable the description of an element which is made of its label set (referred to by $\lambda(\eta(\cdot))$) and its record (referred to by $\delta(\eta(\cdot),\cdot)$); it does not contain an identifier for this element.

Saturation procedure and consistency check. In the body of the main algorithm we make use of a procedure called *saturation*. The main idea of this procedure is to propagate equality and inequality constraints between variables whenever new ones are found. We illustrate why this procedure is needed and how it works on the following pattern:

$$(u)_{\langle u.a=1\rangle} \overset{z}{\longrightarrow} (v) \overset{z}{\longrightarrow} (w)_{\langle w.a=2\rangle}$$

Because both occurrences of z must map to the same edge, the constraints over the endpoints of z enforce equalities between the variables u, v and w. Successively, these equalities imply that $u.a = w.a$, which is in conflict with the requirements of the pattern. Note that if we remove the conditions $u.a = 1$ and $w.a = 2$, this pattern remains unsatisfiable w.r.t. the simple semantics; the forced repetition of nodes in bindings of this pattern was not explicit before applying the saturation procedure because no variable was reused. However, reusing the variable z makes the pattern explicitely unsatisfiable under trail semantics.

To formalize this, we introduce the notion of an *equality graph G for a query Q* (*or a pattern π*), which is a 2-layer undirected edge-labeled graph with nodes in each layer that respectively belong to the sets $\mathcal{Y}$ and $\mathcal{X}$. There are two edge labels, = and ≠. Edges can only connect nodes from the same layer. If G is an equality graph for a pattern π, there are two distinguished nodes $\mathrm{src}_\pi, \mathrm{tgt}_\pi \in \mathcal{X}$ which are respectively called the source and the target, and abbreviated src, tgt if clear from the context. This structure supports the following set of operations:

Saturation: The first step of the *saturation procedure over a pattern π* is to equate the endpoints of edge variables. Let e be an edge variable in G, if G contains say x and y, two distinct node variables, which are both at the source or the target of an occurrence of e in π, then add $x = y$ in G.
In a second step, it performs the transitive closure on the =-edges of the graph at layer $\mathcal{X}$.
After obtaining a =-transitive graph, we pass on the inequalities: if there is an ≠-edge between say x and y, and if both $x' = x$ and $y' = y$ in G, then we add $x' \neq y'$ in G.
Finally, it does a backward step to pull-up the inequalities: if there is an ≠-edge between two elements in $\mathcal{X}$ and if both are either the source or the target of some edges in Q, then it adds an inequality edge between these two edge variables.

Check consistency: Given an assignment for all the variables mentioned in the equality graph, check if all equalities are satisfied in this assignment: if x and y are two variables with $x = y$ in G, check if their assignments are strictly the same. Moreover, check if there is no conflict in the graph: a *conflict* is when there are both an =-edge and an ≠-edge between the same pair of nodes of G; or if there is a ≠-loop.

Merge: Given two equality graphs G_1 and G_2, *merging both on nodes x and y for policy ρ* consists in:

(1) taking the union of their vertices and edges;
(2) adding an =-edge between x and y; if x and y are given as parameters;
(3) if ρ = simple, adding an ≠ edge from each node in the layer $\mathcal{X}$ of G_1 not =-connected to tgt, to each node of the layer $\mathcal{X}$ of G_2 not =-connected to src; (This is consistent with the semantics of the concatenation of $\rho_1 \cdot \rho_2$, where the target or ρ_1 must be equal to the source of ρ_2.);
(4) if ρ = trail, adding an ≠ edge from each node of the layer $\mathcal{Y}$ of G_1, to each node of the layer $\mathcal{Y}$ of G_2;
(5) applying the saturation procedure.

Note that the result of this operation is also a valid equality graph and that x, y and ρ are optional parameters.

The maximum number of nodes and edges in an equality graph for Q is polynomial in Q; and the three procedures can be implemented in polynomial time.

Inductive procedure for patterns. In the following, we describe a non-deterministic polynomial space procedure to check if a GPC pattern query $\rho\pi$ is satisfiable. This inductive procedure over the structure of π succeeds if and only if π is satisfiable under policy ρ. It returns a pair consisting of an extended assignment and an equality graph over the variables in the assignment:

- Case $\pi := (x : \ell)$. Guess a node element with a label set $L \subseteq \mathcal{L}_0$ s.t. $\ell \in L$ and a record consisting in a partial assignment from the keys in $\mathcal{K}_0$ to Const_0. Return the pair consisting of the extended assignment binding src_π, tgt_π and x to this node; and of the equality graph containing three nodes: src_π, tgt_π and x, and two edges: $\mathsf{src}_\pi = x$ and $\mathsf{tgt}_\pi = x$.
- Case $\pi := \xrightarrow{y:\ell}$. Guess an edge element with a label set $L \subseteq \mathcal{L}_0$ s.t. $\ell \in L$ and a record consisting in a partial assignment from the keys in $\mathcal{K}_0$ to Const_0. Return one of the following two possibilities:
  - Return a pair consisting of the extended assignment binding y to this edge, and src_π and tgt_π to two arbitrarily guessed endpoint nodes (which act as the source and target of y); the equality graph contains these three elements and an $\neq$-edge between src_π and tgt_π.
  - (Loop; if ρ is not simple) Return a pair consisting of the extended assignment binding this edge to y and src_π and tgt_π to the same arbitrarily guessed endpoint node; the equality graph contains these three elements and an $=$-edge between src_π and tgt_π.
- Case $\pi := \pi_1 + \pi_2$. Guess $i \in \{1, 2\}$ and return the extended assignment and equality graph obtained by a recursive call to this procedure on π_i, after removing the variables that do not appear in π_{3-i}. (This is because those variables are of type $\mathsf{Maybe}(\cdot)$ in π.)
- Case $\pi := \pi_1\pi_2$. Perform a recursive call to this procedure on both π_1 and π_2 to obtain an extended assignment and an equality graph for π_i, $i \in \{1, 2\}$. Check whether they unify (i.e., check whether the assignments of π_1 and π_2 are strictly the same on their common variables). Then, merge the two equality graphs on tgt_{π_1} and src_{π_2} under the policy ρ; and check consistency w.r.t. the unified extended assignment. Return the pair consisting of the unified extended assignment and the merged equality graph after removing tgt_{π_1} and src_{π_2} and renaming src_{π_1} to src_π and tgt_{π_2} to tgt_π.
- Case $\pi := \pi_{1\langle\theta\rangle}$. Perform a recursive call to this procedure on π_1 to obtain an extended assignment and an equality graph for π_1. Check the validity of θ (as defined in the *Semantics of conditioned patterns* in [19]) over the extended assignment of π_1 and return the same extended assignment and equality graph.
- Case $\pi := \pi_1^{n..m}$. Guess a length k between n and m. (Note that k can be assumed to be at most exponential in the size of the whole query if m is ∞ by Lemma C.1; hence, it can be written in binary using a polynomial amount of space.) Perform k successive recursive calls to this procedure on π_1. Each time drop from the obtained extended assignment and equality set all but the src and tgt variables and the equality or inequality edge between them. Check if they concatenate with the previous block obtained so far (i.e., perform the case $\pi := \pi_1\pi_2$). Return a pair consisting of an extended assignment binding the src of the very first guess to src_π and the tgt of the very last guess to tgt_π; and of the equality graph containing only the src_π and tgt_π nodes, possibly with an $=$ or an $\neq$ edge between them if should be.

Example. We illustrate how our procedure works for repetition patterns with the following example:

$$\pi := \mathsf{simple}\left(\Big[(u) \to (v)\Big]_{\langle\neg(u.a=v.a)\rangle}\right)^{0..\infty}$$

We note $\pi_1 := \Big[(u) \to (v)\Big]_{\langle\neg(u.a=v.a)\rangle}$. There are three different types of behavior depending on k:

- If $k = 0$, it returns a pair consisting of an extended assignment binding src_π and tgt_π to an arbitrarily guessed node element; and of an equality graph containing the edge $\mathsf{src}_\pi = \mathsf{tgt}_\pi$.
- If $k = 1$, it returns a pair consisting of an extended assignment binding src_π and tgt_π to two node elements having a different value for a (if both set); and of the quality graph containing $\mathsf{src}_\pi \neq \mathsf{tgt}_\pi$.
- If $k \geq 2$, it returns a pair consisting of an extended assignment binding src_π and tgt_π to two arbitrary node elements; and of the quality graph containing $\mathsf{src}_\pi \neq \mathsf{tgt}_\pi$.

Invariant. Let π be a pattern matched under a restrictor $\rho \in \{\mathsf{simple}, \mathsf{trail}\}$. The pair (η, G_π) is an output of the procedure consisting of an extended assignment and an equality graph for π *iff* there exists a property graph P such that $(p, \mu) \in [\![\pi]\!]_P$[1] and if, for all x s.t. $\mathsf{sch}(\pi)(x) \in \{\mathsf{Node}, \mathsf{Edge}\}$, we have:

- $\lambda(\mu(x)) = \lambda(\eta(x))$;
- for all key (property) a, $\delta(\mu(x), a) = \delta(\eta(x), a)$, or both are not defined;
- if $x = y$ in G then $\mu(x) = \mu(y)$; similarly, if $x \neq y$ in G then $\mu(x) \neq \mu(y)$.

In the following, we provide the key ideas for proving this invariant by induction:

- Case $\pi := (x : \ell)$ and case $\pi := \xrightarrow{y:\ell}$ are trivial as P can be obtained directly from η.
- Case $\pi := \pi_1 + \pi_2$ by applying the hypothesis on either side. Note that π may contain fewer singleton variables than π_i. (This is because the variables of π_i that do not appear in π_{3-i} are of type $\mathsf{Maybe}(\cdot)$ in π.)

[1]The set of answers to π on P [19].

- Case $\pi := \pi_1\pi_2$ relies on the merge procedure to propagate the *forced* equalities and inequalities.
- Case $\pi := \pi_{1\langle\theta\rangle}$ by noticing that we only need to check whether the condition holds over the extended assignment because conditions only apply on singleton variables. (This is enforced by the *Typing rules for the GPC type system* presented in Figure 2 of [19].) Notice that $\neg(u.a = v.a)$ does not lead to $u \neq v$ because either $u.a$ or $v.a$ may be undefined; thus, we don't need to update G. (Again, this is because of the *Semantics of conditioned patterns* of [19] where $\mu \models (x.a = x.b)$ *iff* $\delta(\mu(x), a)$ and $\delta(\eta(y), b)$ are defined and equal.)
- Case $\pi := \pi_1^{n..m}$ because π does not contain any singleton variables. Hence, only a potential equality or inequality between its endpoints is tracked throughout the k iteration steps over π_1.

Extension to queries (with join and conditioning). Let $Q := Q_1, Q_2$ be a join query with Q_1 and Q_2 matched under restrictors simple or trail; and let the pairs (η_i, G_{Q_i}), $i \in \{1, 2\}$ be returned by the previous procedure on each Q_i. The procedure (for Q) checks if the η_i's unify (i.e., check whether the η_i's agree on their common variables), and if the result of merging the G_i's remains consistent.

Note that this procedure supports *conditioning over join queries* (i.e., we add the following query expression $Q := Q_{\langle\theta\rangle}$) by checking if the condition is valid over η, similar to what is done for conditioning in patterns.

Extension to GPC+. Simply guess a GPC query among all the disjuncts, and check its satisfiability using the previous procedure.

Shortest restrictor. Note that the invariant is not valid if the shortest restrictor is used. For instance, consider the following pattern, which is not supported by the above procedure:

$$\begin{aligned}
&\textsf{shortest simple}\ (x) \left[() \rightarrow \underset{\langle a=1\rangle}{(u)} \rightarrow () + () \rightarrow () \rightarrow (u) \rightarrow () \rightarrow () \right] (y),\\
&\textsf{simple}\ (x) \rightarrow \underset{\langle a=1\rangle}{(w)} \rightarrow (y),\\
&\textsf{simple}\ \underset{\langle a=2\rangle}{(u)}
\end{aligned}$$

Nevertheless, we can easily prove that the above procedure works when all patterns in a GPC query Q that are matched under the shortest restrictor have only their endpoints for singleton variables. ◻

Lemma C.1. *Let $\pi := \pi_1^{\infty}$ be a sub-pattern of a GPC query $\rho\pi_0$. If there is an answer path for π, then there is another answer path for π consisting of at most k repetitions of π_1, with k exponential in the size of π_0, and with all answer paths for π_1 being inner disjoints.*

Proof. There are at most exponentially many distinct records for nodes with values in Const_0, keys in $\mathcal{K}_0$ and labels in $\mathcal{L}_0$. Given the *Semantics of repeated patterns* in [19], only the target node of an iteration has an impact over the next iteration by being its source; only this information is transferred across successive repetitions of π_1. Thus, we can reduce the number of repetitions of π_1 in the initial answer path because one target node necessarily gets repeated. We can moreover assume w.l.o.g. that all answer paths to π are inner disjoints, by taking disjoint copies of the initial answer paths. ◻

Lemma 4.6. *The satisfiability problem is NP-hard even for single-node GPC patterns.*

Proof. We reduce 3-SAT to the satisfiability of a GPC query. Let F be the following 3-SAT formula over n variables and m clauses:

$$\bigwedge_{1\le i\le m} (l_{i1} \vee l_{i2} \vee l_{i3})$$

where for all $1 \le i \le m$ and $j \in \{1, 2, 3\}$, l_{ij} is a *literal* which is either x_k or $\bar{x}_k$ for a $k \in \{1, \dots, n\}$.

We construct the following GPC query $Q(x) := \rho\ \pi_{\langle\theta\rangle}$ with $\pi := (x : \ell)$ and

$$\begin{aligned}
\theta := &\bigwedge_{1\le i\le n} (x_i = 1 \wedge \bar{x}_i = 0) \vee (x_i = 0 \wedge \bar{x}_i = 1)\\
\wedge &\bigwedge_{1\le j\le m}\ \bigvee_{(b_1,b_2,b_3)\in C^3} \big(l_{j1} = b_1 \wedge l_{j2} = b_2 \wedge l_{j3} = b_3\big)
\end{aligned}$$

where $C^3 = \{(0,0,1), \dots, (1,1,1)\}$; x and ℓ are optional in π; the literals of F are used as property names in Q; and size of Q is clearly polynomial in the size of F.

We now show that F is satisfiable if and only if there exists a property graph G on which $Q(x)$ returns at least a node.

($\Rightarrow$) Assume that F is satisfied by an assignment v to $\bar{x}$. We construct a property graph containing a unique ℓ-labeled node with identifier o, having the following record:

$$\forall i \in \{1, \dots, n\} \left\{ \begin{array}{ll} x_i \mapsto 1, & \\ \bar{x}_i \mapsto 0 & \text{if } v_i = \top \\ x_i \mapsto 0, & \\ \bar{x}_i \mapsto 1 & \text{if } v_i = \bot \end{array} \right\}.$$

By design, the top-most conjunct of θ is satisfied. Let C_i for any $1 \leq i \leq m$ be a clause in F; by hypothesis c_i is satisfied by v, so there exists $(b_1, b_2, b_3) \in C^3$ such that $\delta(o, l_{ij}) = b_j$ for all $1 \leq j \leq 3$.
($\Leftarrow$) Conversely, if $Q(x)$ is satisfied in a property graph G for an element o; we have that $o \in N$ and each $x_i, \bar{x}_i$ are defined in the record of o. The restrictions enforced by the top-most conjunct of θ ensure that we can retrieve a well-defined assignment for F; the last conjunct ensures that this is a valid assignment for F. □

D EXPERIMENTS

User study. The User study consists of four parts, respectively aiming to:

- Evaluate the Understandability of openCypher scripts; we asked four yes/no questions asking to the participants whether an assertion is true or not w.r.t. the behavior of a provided script in a concrete transformation scenario (e.g. Does this script create as many Director nodes as there are Person nodes that have an outgoing relationship of type DIRECTED to a Movie node?).
- Evaluate the Understandability of Property Graph Transformations; with four similar yes/no questions on a slightly different transformation to avoid biases;
- The third part required participants to modify some provided openCypher scripts and transformations to adapt to a new requirement. We collected the participants' answers and checked them.
- The last part required the participants to give their opinion on the following questions and to indicate, in a range for 1 to 5 (3 is neutral) whether they found openCypher scripts and/or transformation rules better on:
  - Which one of the two methods do you find easier to understand?
  - Which one of the two methods do you find more intuitive? (Better for describing the desired output.)
  - Which one of the two methods do you find more flexible? (Easier to adapt to a new specification.)

We collected the answers of 12 participants, that were asked to self report their level of expertise in a range from (1 - Novice) to (5 - Expert) on the following topics:

- How would you rate your knowledge about databases?
  The answers filled in by the participants ranged from 3 to 5, included.
- How would you rate your knowledge about openCypher?
  The answers filled in by the participants ranged from 2 to 4, included.
- How would you rate your knowledge about the MERGE clause of openCypher?
  The answers filled in by the participants ranged from 1 to 5, included.
- How would you rate your knowledge about property graph transformations?
  The answers filled in by the participants ranged from 1 to 3, included.

We have a pool of people that all have prior exposure to openCypher (2-4) but a great diversity w.r.t. the knowledge of the MERGE clause of openCypher (the basic tool for updates in openCypher), i.e. from novice to expert.
The results on the first two parts are as follow:

- The average number of correct answers is 50% (2.0 out of 4) for the understandability of the openCypher scripts, and 90% (3.6 out of 4) for the understandability of the transformation rules.
- 25.0%, resp. 67% of participants checked all the correct answers in the first, resp. second part.
- All participants scored higher on their individual understanding of transformation rules compared to openCypher scripts.

Given those results, it is extremely clear that it is very difficult for people to understand even the basic openCypher scripts used to transform property graphs, whereas our framework – despite being absolutely new to the participants, has been widely understood.
The results on the last part are as follow (recall that 3 is neutral, 1 is strong preference for openCypher scripts and 5 is strong preference for transformation rules):

- Which one of the two methods do you find easier to understand?
  Collected answers range from 1 to 5 with an average of 3.3.
- Which one of the two methods do you find more intuitive? (Better for describing the desired output.)
  Collected answers range from 3 to 5 with an average of 3.8.
- Which one of the two methods do you find more flexible? (Easier to adapt to a new specification.)
  Collected answers range from 3 to 5 with an average of 4.1.

Moreover, we have noticed that the participants having a low understanding of openCypher scripts (scored 2 or less out of 4 in the first part) have been more inclined to provide less credit to the transformation rules than other people. So we decided to split the participants in two groups to investigate this more.
The results on the last part are as follow only for the 8 people that have score 2 or lower out of 4 in their understanding of openCypher scripts (recall that 3 is neutral, 1 is strong preference for openCypher scripts and 5 is strong preference for transformation rules):

- Which one of the two methods do you find easier to understand?
  Collected answers range from 1 to 5 with an average of 2.75.

- Which one of the two methods do you find more intuitive? (Better for describing the desired output.)
  Collected answers range from 3 to 5 with an average of 3.9.
- Which one of the two methods do you find more flexible? (Easier to adapt to a new specification.)
  Collected answers range from 3 to 5 with an average of 3.9.

The results on the last part are as follow only for the 4 people that have score 3 or higher out of 4 in their understanding of openCypher scripts (recall that 3 is neutral, 1 is strong preference for openCypher scripts and 5 is strong preference for transformation rules):

- Which one of the two methods do you find easier to understand?
  Collected answers range from 1 to 5 with an average of 4.7.
- Which one of the two methods do you find more intuitive? (Better for describing the desired output.)
  Collected answers range from 3 to 5 with an average of 3.7.
- Which one of the two methods do you find more flexible? (Easier to adapt to a new specification.)
  Collected answers range from 3 to 5 with an average of 4.7.

It is therefore clear that those who have been less convinced that openCypher scripts are more error-prone and harder to interpret and analyze have not figured out for themselves that openCypher scripts are difficult to understand and manipulate. Moreover people with a good understanding of openCypher are clearly in favor that our framework is easier to understand.

With this study, we empirically and experimentally demonstrated that the script-based approach can be error prone, hard to interpret and analyze (i.e., less usable) and that the improvement of usability and accuracy over handcrafted, script-based solutions have clearly been attested by a majority of the participants.

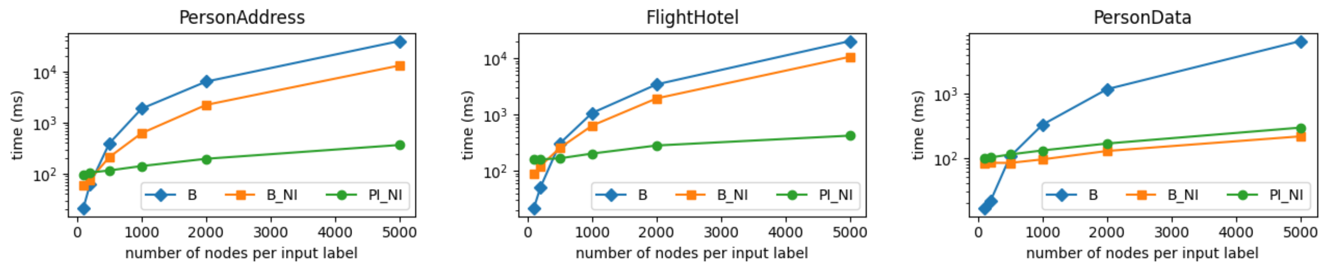

Figure 14: Run-time comparison with the baseline approach, depending on the number of input nodes of each type in G.

Comparison with native Cypher approach. Finally, we compared our framework (using PI_NI) with ad-hoc transformation scripts (B-NI, B; respectively with and without node indexes), such as the one presented in Figure 1 (ii). The result over PersonAddress, FlightHotel and PersonData are presented in Figure 14. For larger scenarios, such as GUSToBIOSQL, DBLPToAmalgam1 and Amalgam1ToAmalgam3, handcrafted transformation scripts are exceedingly large due to the number of rules and properties involved.

We can observe that our solution clearly outperforms the handcrafted solutions in most of the cases. The only exception occurs when using the PersonData scenario, for which the B-NI baseline is slightly better than our solution, while the B baseline is still outperformed. The underlying reason is due to the nature of this scenario, for which the `collect` clause contains only one element.

D.1 Offshore Leaks Dataset

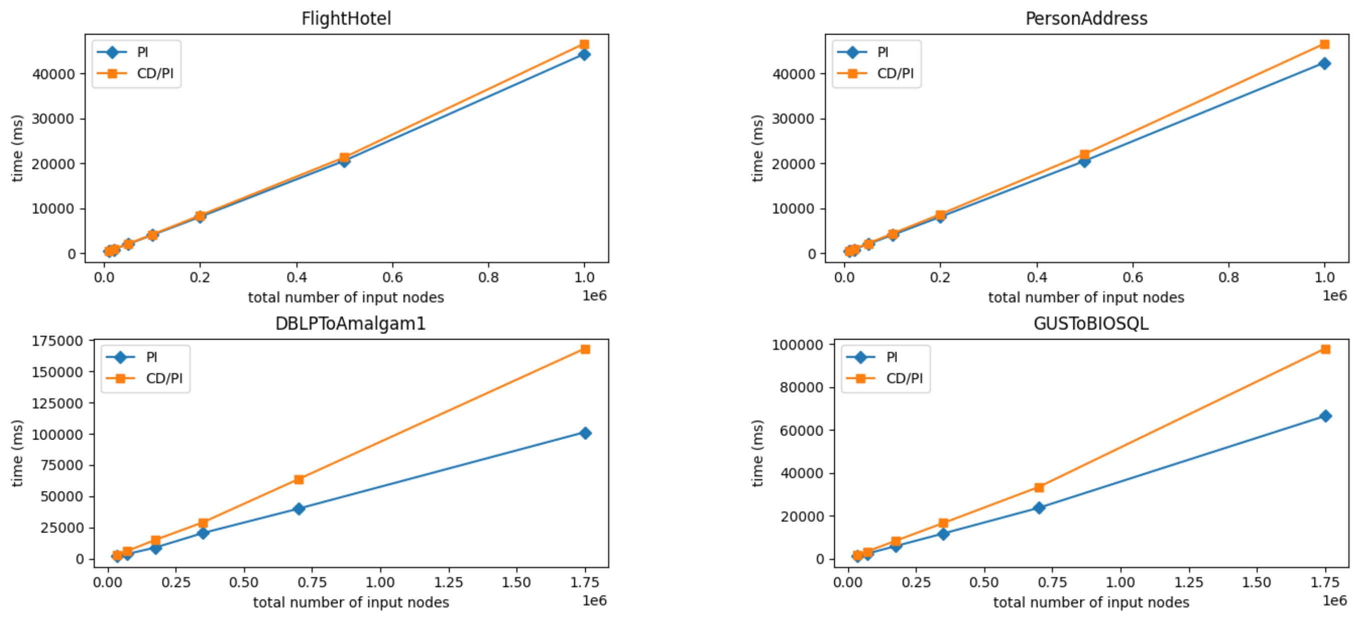

Figure 15: Horizontal scaling, with varying number of independent copies of the scenario.

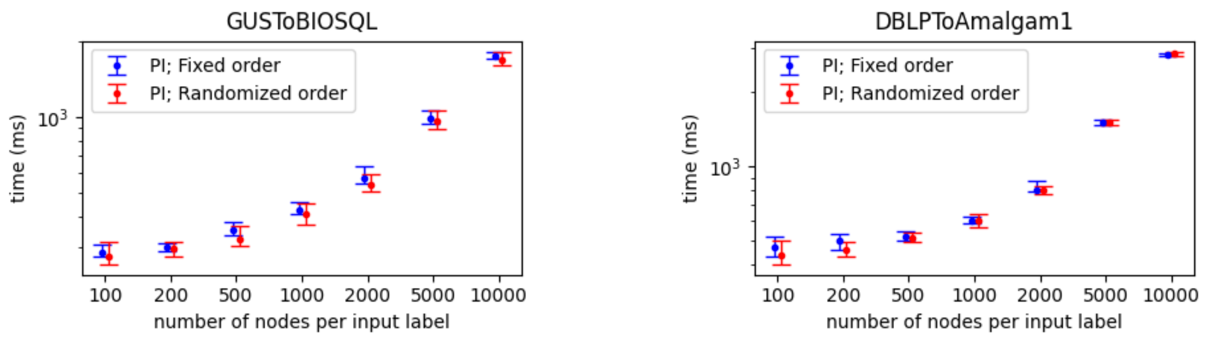

Figure 16: Average computation time for different orders of execution of the rules; error bars indicate minimum and maximum computation times observed over 20 independent runs.

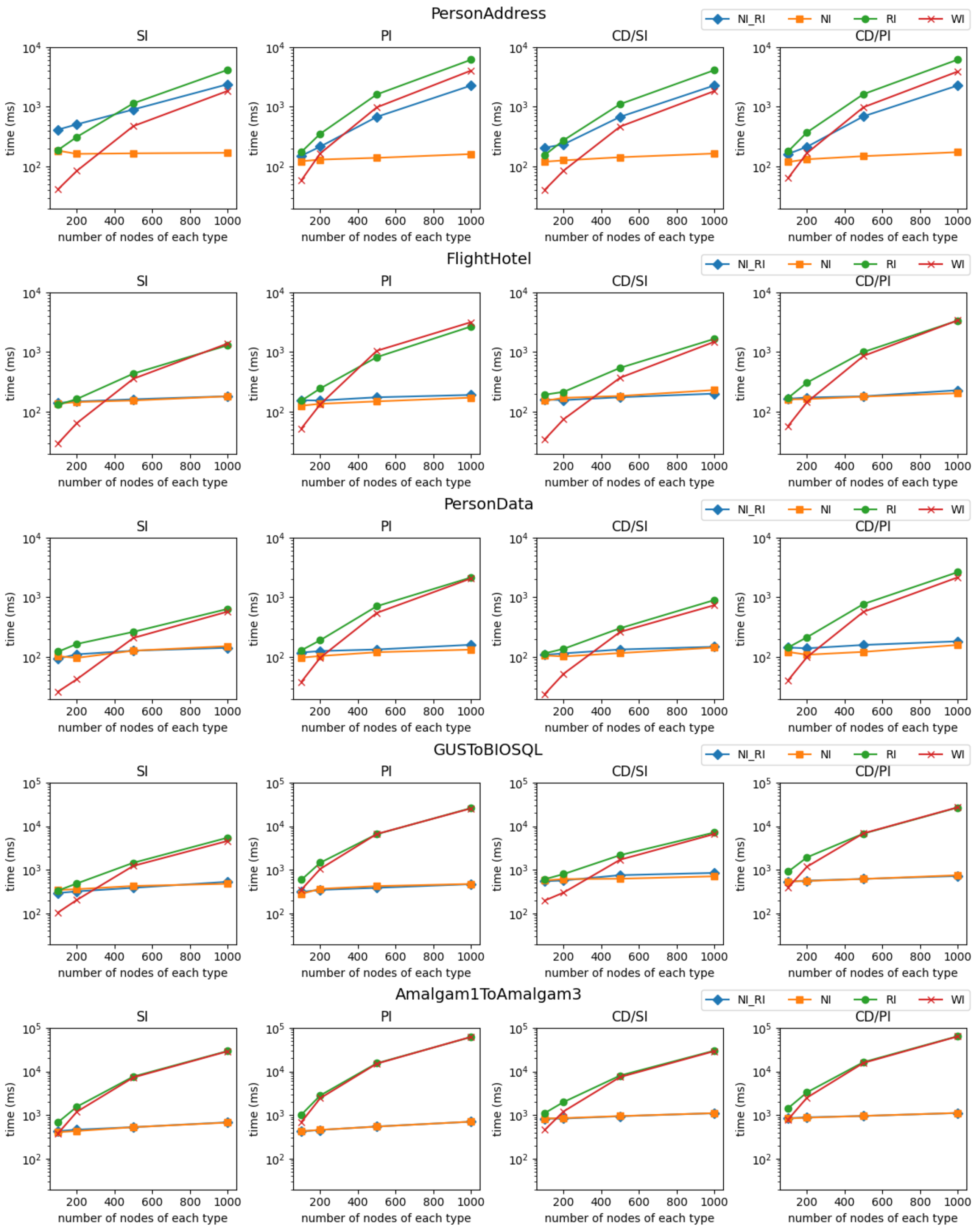

Figure 17: Impact of indexing strategies and implementation variants on the computation of $T(G)$.

$$\underset{\langle sourceID=\text{"Malta registry"},\ a.address=a.address\rangle}{(a:\text{Address})} \Longrightarrow \underset{\langle source=a.sourceID\rangle}{(x=(a):\text{T\_Address})} \xrightarrow{:\text{T\_LOCATED}} \underset{\langle name=a.country\rangle}{(y=(a.country\_code):\text{T\_Country})} \qquad (R_1)$$

$$\underset{\langle sourceID=\text{"Malta registry"},\ \neg(a.address=a.address)\rangle}{(a:\text{Address})} \Longrightarrow \underset{\langle source=a.sourceID\rangle}{(x=(a):\text{T\_Address})} \qquad (R_2)$$

$$\underset{\langle \neg(sourceID=\text{"Malta registry"}),\ a.address=a.address\rangle}{(a:\text{Address})} \Longrightarrow \underset{\langle source=a.sourceID\rangle}{(x=(a):\text{T\_Address})} \xrightarrow{:\text{T\_LOCATED}} \underset{\langle name=a.country\rangle}{(y=(a.country\_codes):\text{T\_Country})} \qquad (R_3)$$

$$\underset{\langle \neg(sourceID=\text{"Malta registry"}),\ \neg(a.address=a.address)\rangle}{(a:\text{Address})} \Longrightarrow \underset{\langle source=a.sourceID\rangle}{(x=(a):\text{T\_Address})} \qquad (R_4)$$

(i) Refactoring registered addresses ($R1 - R4$).

$$(i:\text{Intermediary}) \xrightarrow{:\text{registered\_address}} (a:\text{Address}) \Longrightarrow (x=(i):\text{T\_Intermediary}) \xrightarrow{:\text{T\_REG\_ADDRESS}} (y=(a):\text{T\_Address}) \qquad (R_5)$$

$$\underset{\langle \neg(address=\text{""}),\ address=address,\ country\_code=country\_code\rangle}{(i:\text{Intermediary})} \Longrightarrow (x=(i):\text{T\_Intermediary}) \xrightarrow{:\text{T\_REG\_ADDRESS}} \underset{\langle source=i.sourceID\rangle}{(y=(i.address):\text{T\_Address})},$$
$$(y) \xrightarrow{:\text{T\_LOCATED}} \underset{\langle name=i.countries\rangle}{(z=(i.country\_codes):\text{T\_Country})} \qquad (R_6)$$

$$\underset{\langle \neg(address=\text{""}),\ address=address,\ \neg(country\_code=country\_code)\rangle}{(i:\text{Intermediary})} \Longrightarrow (x=(i):\text{T\_Intermediary}) \xrightarrow{:\text{T\_REG\_ADDRESS}} \underset{\langle source=i.sourceID\rangle}{(y=(i.address):\text{T\_Address})} \qquad (R_7)$$

$$\underset{\langle \neg(address=\text{""}),\ address=address,\ \neg(country\_code=country\_code)\rangle}{(i:\text{Intermediary}) \xrightarrow{:\text{intermediary\_of}} (e:\text{Entity})} \Longrightarrow (x=(i):\text{T\_Intermediary}) \xrightarrow{:\text{T\_REG\_ADDRESS}} \underset{\langle source=i.sourceID\rangle}{(y=(e.address):\text{T\_Address})} \qquad (R_8)$$

$$\underset{\langle \neg(address=\text{""}),\ address=address,\ country\_code=country\_code\rangle}{(i:\text{Intermediary}) \xrightarrow{:\text{intermediary\_of}} (e:\text{Entity})} \Longrightarrow (x=(i):\text{T\_Intermediary}) \xrightarrow{:\text{T\_REG\_ADDRESS}} \underset{\langle source=i.sourceID\rangle}{(y=(e.address):\text{T\_Address})},$$
$$(y) \xrightarrow{:\text{T\_LOCATED}} \underset{\langle name=i.countries\rangle}{(z=(e.country\_codes):\text{T\_Country})} \qquad (R_9)$$

(ii) Uniformizing address information for intermediaries ($R5 - R9$).

$$(i:\text{Intermediary}) \Longrightarrow \underset{\langle name=i.name,\ status=i.status,\ valid\_until=i.valid\_until,\ source=i.sourceID\rangle}{(x=(i):\text{T\_Intermediary})} \qquad (R_{10})$$

$$(a:\text{Address}) \Longrightarrow \underset{\langle address=a.address,\ orig\_addr=a.orig\_addr,\ valid\_until=a.valid\_until\rangle}{(x=(a):\text{T\_Address})} \qquad (R_{11})$$

$$(e:\text{Entity}) \Longrightarrow \underset{\langle name=e.name,\ orig\_name=e.orig\_name,\ inact\_date=e.inact\_date,\ inc\_date=e.inc\_date,\ \ldots,\ source=e.sourceID\rangle}{(x=(e):\text{T\_Entity})} \qquad (R_{12})$$

$$(o:\text{Officer}) \Longrightarrow \underset{\langle name=o.name,\ status=o.status,\ source=o.sourceID\rangle}{(x=(o):\text{T\_Officer})} \qquad (R_{13})$$

(iii) Exporting the nodes ($R10 - R13$).

Figure 18: Rules of the ICIJ database transformation.

$$(o : \text{Officer}) \xrightarrow{:\text{similar}} (p : \text{Officer}) \implies (x = (o) : \text{T\_Officer}) \xrightarrow[\langle link\text{="similar name and address as"}\rangle]{()\,:\,\text{T\_Similar}} (y = (p) : \text{T\_Officer}) \qquad (R_{14})$$

$$(a : \text{Address}) \xrightarrow{:\text{same\_as}} (b : \text{Address}) \implies (x = (a) : \text{T\_Address}) \xrightarrow{:\text{T\_SAME\_AS}} (y = (b) : \text{T\_Address}) \qquad (R_{16})$$

$$(o : \text{Officer}) \xrightarrow{:\text{registered\_address}} (a : \text{Address}) \implies (x = (o) : \text{T\_Officer}) \xrightarrow{:\text{T\_REGISTERED\_ADDRESS}} (y = (a) : \text{T\_Address}) \qquad (R_{17})$$

(i) Improving similarity detection ($R14 - R17$).

$$\underset{\langle jurisdiction\_desc{=}jurisdiction\_desc\rangle}{(e : \text{Entity})} \implies (x = (e) : \text{T\_Entity}) \xrightarrow{:\text{T\_IN\_JURIS}} \underset{\langle juris{=}e.jurisdiction\_desc\rangle}{(y = (e.jurisdiction\_desc) : \text{T\_Jurisdiction})},$$
$$(y) \xrightarrow{:\text{T\_RELATED}} (z = (e.jurisdiction) : \text{T\_Country}) \qquad (R_{18})$$

(ii) Refactoring jurisdictions ($R18$).

Figure 19: Rules of the ICIJ database transformation. (Bis)