# Querying Graph Databases: What Do Graph Patterns Mean?

Stephan Mennicke[1](✉), Jan-Christoph Kalo[2], and Wolf-Tilo Balke[2]

[1] Institut für Programmierung und Reaktive Systeme, TU Braunschweig, Germany
mennicke@ips.cs.tu-bs.de
[2] Institut für Informationssysteme, TU Braunschweig, Germany
{kalo,balke}@ifis.cs.tu-bs.de

**Abstract.** Querying graph databases often amounts to some form of graph pattern matching. Finding (sub-)graphs isomorphic to a given graph pattern is common to many graph query languages, even though graph isomorphism often is too strict, since it requires a one-to-one correspondence between the nodes of the pattern and that of a match. We investigate the influence of weaker graph pattern matching relations on the respective queries they express. Thereby, these relations abstract from the concrete graph topology to different degrees. An extension of relation sequences, called failures which we borrow from studies on concurrent processes, naturally expresses simple presence conditions for relations and properties. This is very useful in application scenarios dealing with databases with a notion of *data completeness*. Furthermore, failures open up the query modeling for more intricate matching relations directly incorporating concrete data values.

**Keywords:** Graph databases · Query modeling · Pattern matching

## 1 Introduction

Over the last years, *graph databases* have aroused a vivid interest in the database community. This is partly sparked by intelligent and quite robust developments in information extraction, partly due to successful standardizations for knowledge representation in the Semantic Web. Indeed, it is enticing to open up the abundance of unstructured information on the Web through transformation into a structured form that is usable by advanced applications. A good example is the *Knowledge Graph* by Google, supporting sophisticated features in Web search. Other examples are the growing number of special interest graph databases created as part of the Linked Open Data (LOD) initiative. Most of these graphs provide *entity-centric data* from diverse domains: entities are represented as nodes connected by labeled edges (relations) to other entity nodes or attribute nodes (literals). Thereby, the nodes of a graph database are often referred to as *graph database objects*. Graph database query languages let the user query these graph structures in an SQL-like fashion. Examples are SPARQL as a W3C standard for querying Semantic Web data[3], Cypher as the standard query language

[3] https://www.w3.org/TR/rdf-sparql-query/

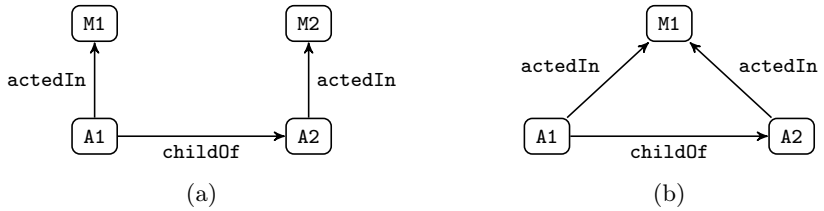


Fig. 1: Example Graph Patterns

for Neo4J[4], and Gremlin[5]. Basically, all these languages rely on the idea of graph pattern matching, complemented with suitable node substitutions. A matching mechanism common to all these query languages is graph isomorphism [11].

Unfortunately, structural identity in the sense of graph isomorphism often is too restrictive for many applications, see e. g., [5, 4]. Imagine a user query for **actors being children of other actors**. One possibility to model this query as a graph pattern is depicted in Fig. 1(a). `A1` and `A2` are two nodes being connected by a `childOf` relationship, both having an `actedIn` relationship to either node `M1` or `M2`. Thereby, it is intended that nodes `A1` and `A2` represent the desired actors from the query while `M1` and `M2` are representatives for movies they acted in. Considering isomorphic matches to the graph pattern, an answer is required to have only actors acting in at least two different movies. Isomorphic matches of the pattern depicted in Fig. 1(b) are thus ruled out.

Since the query in the previous paragraph is sufficiently vague, we may assume that matches to both graph patterns of Fig. 1 are intended. The reason why a single pattern is insufficient to cover the whole query is that graph isomorphism relies on a one-to-one-correspondence between the nodes of the graph pattern and that of a match. However, it would be quite sufficient to require that an object of a match incorporates in relation `actedIn` and a sequence of relations `childOf` and `actedIn`, abstracting from the concrete pattern structure. Sequences over relation symbols are henceforth called *traces*. Whether or not both actors took part in the same movies is not subject to the trace representation of the graph pattern. In fact, the patterns of Fig. 1 are indistinguishable up to traces, thus represent the same set of *trace matches*.

From Sect. 3 on, we study the influence of different graph pattern matching relations to the queries they express. All these notions are based on graph databases and graph patterns, introduced in Sect. 2. To the best of our knowledge, trace-based relations have not been investigated in the context of graph pattern matching for database querying to which we contribute so-called *failures* originating from studies on equivalence of concurrent processes [3]. Beyond a purely topological analysis, failures express simple presence conditions for relations and properties associated with the database objects. This allows for strong correspondences in databases enjoying *data completeness*, still without the strict

[4] `https://neo4j.com/`
[5] `http://tinkerpop.apache.org/`

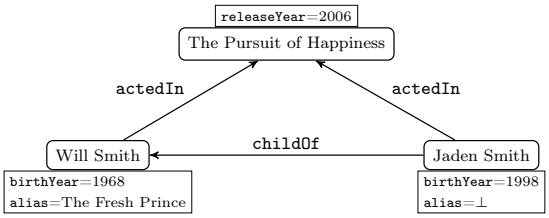


Fig. 2: A Small Graph Database Excerpt

requirements of graph isomorphism. Therefore, we incorporate object properties in our notion of graph patterns and graph pattern matching. We include *simulation semantics* as an interesting complementation to failures, since these notions constitute incomparable matching relations, i. e., there are simulating matches to patterns that are not failure matches and vice versa. Thus, the choice in favor for failures/simulation semantics is left to the concrete application scenario. Moreover, different kinds of simulation have already been applied in several graph database tasks [14, 16, 1].

Since graph databases are still concerned with data in terms of attributes attached to the database objects, we propose a preliminary extension of our graph pattern model by *data predicates* and apply them to simulation-based semantics in Sect. 4. In Sect. 5, we discuss related work. We conclude our work by Sect. 6.

## 2 Graph Databases and Graph Patterns

This section is devoted to the database model we use throughout the paper, being an RDF-style graph database. For a survey on graph database models, we refer to Angles and Gutierrez [2]. Objects in a database stem from an infinite set of object identifiers $\mathcal{O}$. In Fig. 2, an excerpt from a graph database is given. The object identifiers used there are "Will Smith", "Jaden Smith", and "The Pursuit of Happiness". Database objects are related to other objects or concrete data values (i. e., literals) by attributes, being triples over objects, relations or properties, and objects or data values. We separate the relations between objects from properties relating an object with a concrete data value to simplify the presentation of further concepts. Object relations and properties together form the attributes of a graph database.

For a fixed set of objects $O \subseteq \mathcal{O}$, relations between objects are given in terms of a directed labeled edge relation $\longrightarrow \subseteq O \times \Sigma \times O$, where $\Sigma$ is a finite set of relation symbols. As an example, reconsider Fig. 2 where nodes "Will Smith" and "Jaden Smith" are connected via relation `childOf`. Properties of database objects are referenced by projection functions $\pi_p : O \to \mathcal{D}$, where $p$ is a property from a finite set of properties $P$, and $\mathcal{D}$ is a usually infinite set of possible data values [2]. In Fig. 2, properties of objects are listed in an equation style within the boxes attached to each object node. Value $\perp \in \mathcal{D}$ denotes the absence of an association

of property $p$ with object $o$, e. g., $\pi_{\texttt{alias}}(\text{“}JadenSmith'') = \bot$ in Fig. 2. We usually leave $\bot$-valued properties implicit, e. g., $\pi_{\texttt{releaseYear}}(\text{“}WillSmith'') = \bot$, although not explicitly mentioned in Fig. 2. As a consequence, our database model is a directed edge- and node-labeled graph structure.

**Definition 1.** *Let $\Sigma$ and $P$ be finite alphabets such that $\Sigma \cap P = \emptyset$. A* graph database *is a 5-tuple $DB = (O, \Sigma, P, \longrightarrow, (\pi_p)_{p \in P})$ where $O \subset \mathcal{O}$ is a finite set of* objects *and $\longrightarrow \subseteq O \times \Sigma \times O$ together with $\pi_p : O \to \mathcal{D}$ ($p \in P$) form the* attributes *of DB.*

Relation symbols range over $a, a_1, a_2, \ldots$ while we use $p, p_1, p_2, \ldots$ for properties. We abbreviate families of property functions by $\Pi, \Pi', \Psi$ with respective projections $\pi_p, \pi'_p, \psi_p$. We write $o \xrightarrow{a} o'$ if $(o, a, o') \in \longrightarrow$ and $o \xrightarrow{a}$ if there is an $o'$ with $o \xrightarrow{a} o'$. Likewise, $o \not\xrightarrow{a}$ denotes the absence of an $o'$.

Paths from one database object to another represent fingerprints of the topology of the database that, upon traversal, define object reachability. Capturing only the sequence of relation symbols along a path forms a *trace*.

**Definition 2.** *Let $DB = (O, \Sigma, P, \longrightarrow, \Pi)$ be a graph database. Relation $\longrightarrow$ extends to* relation sequences *$\sigma \in \Sigma^*$ inductively as follows: (1) $o \xrightarrow{\varepsilon} o'$ iff $o = o'$ and (2) $o \xrightarrow{\sigma \cdot a} o'$ iff there is an object $o'' \in O$ with $o \xrightarrow{\sigma} o'' \xrightarrow{a} o'$. A* trace of $o \in O$ *is a word $\sigma \in \Sigma^*$ if there is an $o' \in O$ with $o \xrightarrow{\sigma} o'$. The set of all traces of $o$ is denoted by $\mathcal{T}_{DB}(o)$. The* traces of *DB are defined by $\mathcal{T}(DB) := \bigcup_{o \in O} \mathcal{T}_{DB}(o)$.*

For arbitrary graph databases $A_1$ and $A_2$, we denote by $A_1 \sqsubseteq_{\mathsf{T}} A_2$ that $\mathcal{T}(A_1) \subseteq \mathcal{T}(A_2)$, defining the so-called *trace preorder*. The symmetric closure of $\sqsubseteq_{\mathsf{T}}$ is called *trace equivalence* and is denoted by $\equiv_{\mathsf{T}}$.

Given a graph database $DB = (O, \Sigma, P, \longrightarrow, \Pi)$. A mechanism answering a query to *DB* tries finding *subgraphs* of *DB*, *matching* the query. A subgraph of *DB* is a database $A = (O', \Sigma, P, \longrightarrow', \Pi')$ where $O' \subseteq O$, $\longrightarrow' \subseteq \longrightarrow \cap (O' \times \Sigma \times O')$, and $\pi'_p = \pi_p \restriction O'$ (i. e., $\pi_p$ restricted to inputs from $O'$). We write $A \preceq DB$ if $A$ is a subgraph of *DB*.

Structurally, a query to a graph database *DB* is given in terms of a graph pattern with nodes from some universe of nodes $\mathcal{N}$. Note that for some graph query applications, it holds that $\mathcal{N} \subseteq \mathcal{O}$ or even $\mathcal{N} \subseteq O$, as is the case for exemplar queries [16]. A *graph pattern* $\mathcal{Q}$ is itself a graph database with a finite set of nodes $N_{\mathcal{Q}} \subseteq \mathcal{N}$, relation alphabet $\Gamma \subseteq \Sigma$, a directed $\Gamma$-labeled edge relation $\longrightarrow_{\mathcal{Q}}$, and a family of property functions $\psi_p : N_{\mathcal{Q}} \to \mathcal{D}$, referred to as $\Psi$. We call the elements of $N_{\mathcal{Q}}$ nodes, since, without relating them to an actual database, they refer to abstract objects. An answer to a query w. r. t. graph pattern $\mathcal{Q}$ is a subgraph *matching* $\mathcal{Q}$. The most common matching mechanism in the realm of graph-based models is graph isomorphism, which we introduce here as a base-line for the matching relations to come.

**Definition 3.** *Let $DB = (O, \Sigma, P, \longrightarrow, \Pi)$ be a graph database. A* graph pattern *is a graph database $\mathcal{Q} = (N_{\mathcal{Q}}, \Gamma, P, \longrightarrow_{\mathcal{Q}}, \Psi)$ where $N_{\mathcal{Q}} \subset \mathcal{N}$ is a finite set of*

*nodes and $\Gamma \subseteq \Sigma$. A subgraph $A = (O', \Sigma, P, \longrightarrow', \Pi')$ of DB is isomorphic to $\mathcal{Q}$, denoted $A \cong \mathcal{Q}$, iff there is a bijective function $\nu : N_{\mathcal{Q}} \to O'$ such that for all $n_1, n_2 \in N_{\mathcal{Q}}$, $n_1 \xrightarrow{a}_{\mathcal{Q}} n_2$ iff $\nu(n_1) \xrightarrow{a}' \nu(n_2)$. If $\mathcal{Q} \cong A$, $A$ is called an* isomorphic match of $\mathcal{Q}$. *$[\![\mathcal{Q}]\!]^{\cong}_{DB}$ denotes the set of all isomorphic matches of $\mathcal{Q}$.*

We refer to the example patterns given in Fig. 1 by $\mathcal{Q}_a$ and $\mathcal{Q}_b$. While the graph depicted in Fig. 2 is an isomorphic match of $\mathcal{Q}_b$, it is not of $\mathcal{Q}_a$, since the bijectivity requirement cannot be fulfilled in this case. Since graph isomorphism is an equivalence relation, it holds that whenever we have two isomorphic subgraphs $A_1$ and $A_2$ of $DB$, $A_1$ is an isomorphic match of $\mathcal{Q}$ iff $A_2$ is (for every graph pattern $\mathcal{Q}$).

Besides isomorphic matches, we directly derive another query answering mechanism based on traces (cf. Def. 2). The set of all *trace equivalent matches of* $\mathcal{Q}$ is defined by $[\![\mathcal{Q}]\!]^{\equiv_{\mathsf{T}}}_{DB} := \{A \preceq DB \mid \mathcal{Q} \equiv_{\mathsf{T}} A\}$. As already mentioned, the graph patterns $\mathcal{Q}_a$ and $\mathcal{Q}_b$ are indistinguishable by traces. Hence, both include the graph depicted in Fig. 2 as a trace equivalent match. Isomorphic matches show structural identity, rather than an actual similarity. Trace-based matches may be seen as the other side of the spectrum, since they are the coarsest relations discussed in this paper. It can be shown that for every graph pattern $\mathcal{Q}$, each isomorphic match also is a trace equivalent match, but not vice versa [7].

## 3 Failures in Relations and Properties

We have already seen two different matching relations. In this section, we lift our considerations from a purely topological matching by step-wise incorporating properties due to the notion of failures. We then complement failures by simulation whose matches show to be incomparable with those of the failures matching relation. We introduce both notions by means of an example and by a formal definition. In the end of this section, we discuss benefits and drawbacks for each of the relations. The (counter-)examples given throughout this section are inspired by standard examples in process theory, e. g., as presented by van Glabbeek [7].

### 3.1 Failures

As the query graphs $\mathcal{Q}_a$ and $\mathcal{Q}_b$ of Fig. 1 have shown, graph isomorphism is sensitive to even small structural changes. In contrast, the notion of traces shows

$$\mathcal{T}(\mathcal{Q}_a) = \{\varepsilon, \texttt{actedIn}, \texttt{childOf}, \texttt{childOf} \cdot \texttt{actedIn}\} = \mathcal{T}(\mathcal{Q}_b),$$

implying that both patterns allow for the same trace matches, as e. g., the one depicted in Fig. 2. However, matching by trace equivalence does not allow for a clear differentiation of objects with different relations they participate in. Traces only require that the required relations are present in a subgraph, no matter how distributed they are over the graph. Consider the small database excerpt of Fig. 3(a) together with the graph pattern depicted in Fig. 3(b). Intuitively, the graph pattern represents the query for an actor (`A`) being resident of country

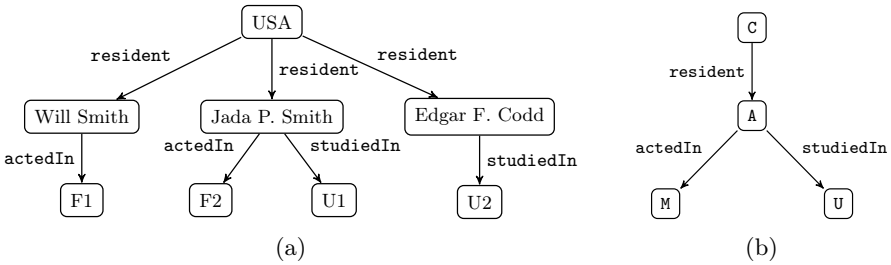


Fig. 3: (a) Another graph database excerpt with indicated `studiedIn` and `actedIn` relations such that $\pi_{\texttt{yearOfDeath}}$("Edgar F. Codd") = 2003 and $\pi_{\texttt{yearOfDeath}}$("Jada P. Smith") = $\perp$. (b) Another example graph pattern.

(`C`), who studied in some facility (`U`). An exact, i. e., isomorphic, match is given by "Jada P. Smith". However, trace equivalence matching also allows for the combination of "Will Smith" and "Edgar F. Codd", since the former features relation `actedIn` while the latter takes part only in a `studiedIn` relation. Trace equivalence misses that "Will Smith" is *not* in a `studiedIn` relation and "Edgar F. Codd" is *not* part of an `actedIn` relationship. As long as both traces `resident` · `studiedIn` and `resident` · `actedIn` are present, a subgraph is considered a trace equivalence match. If, however, we include that after relation `resident`, neither `actedIn` nor `studiedIn` is missing, we rule out the combination of "Will Smith" and "Edgar F. Codd". This is what *failures* are for.

The central idea of *failures*, originating from studies on equivalence of concurrent processes [3], is the notion of *failure pairs*. A *failure pair of an object* $o$ consists of a trace $\sigma \in \mathcal{T}_{DB}(o)$ and a set of relation symbols $X \subseteq \Sigma$ such that for at least one object $o'$ with $o \xrightarrow{\sigma} o'$, $o'$ does *not* participate in any relations $a \in X$, i. e., $o' \not\xrightarrow{a}$. Intuitively, $o'$ *fails* in participating in relations from set $X$, also called *failure set*. By this representation, we naturally obtain the ability to observe that objects like "Edgar F. Codd" do not have an `actedIn` relation. In fact, the graph pattern contains no failure pair (`resident`, $X$) such that `actedIn` $\in X$ or `studiedIn` $\in X$, i. e., in every match, if an object is target of a relation `resident`, then it features both, an `actedIn` and a `studiedIn` relation. Thus, subgraphs containing "Edgar F. Codd" or "Will Smith" are not considered a failures equivalence match.

Often, not every characteristic of an object is expressed in terms of an interrelationship with other objects, but as associations with actual data values, e. g., numerical properties like the year of birth or the *Erdős number*. If we omit the `actedIn` relation from the graph pattern of Fig. 3(b), we could ask for residents `A` of country `C`, who studied in facility `U`. Based on this pattern, we are now interested in all those people being still alive. We model this in the query graph $\mathcal{Q}$ by stating that $\psi_{\texttt{yearOfDeath}}(\texttt{A}) = \perp$, i. e., the property `yearOfDeath` is undefined for an object matching `A`. Therefore, "Jada P. Smith" would belong to a match of $\mathcal{Q}$, but "Edgar F. Codd" should not. In order to make failures aware

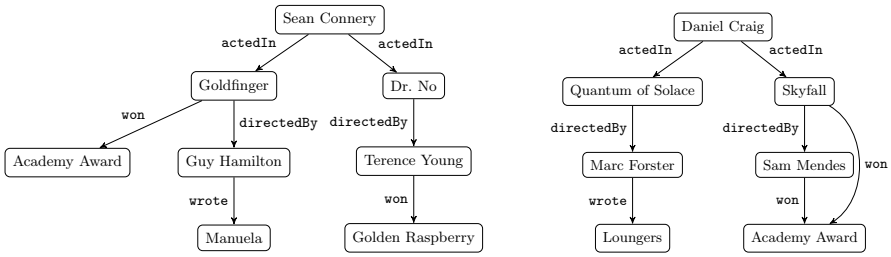


Fig. 4: Two failures equivalent graph databases, i. e., either both are failure matches of a graph pattern or none.

of relation symbols and the (non-)existence of an association between an object and a property, we include properties in failure sets.

**Definition 4.** *Let* $DB = (O, \Sigma, P, \longrightarrow, \Pi)$ *be a database and* $o \in O$*. A* failure pair of $o$ *is a pair* $(\sigma, X) \in (\Sigma^* \times 2^{\Sigma \cup P})$ *where* $\sigma \in \mathcal{T}_{DB}(o)$ *such that there is an object* $o' \in O$ *with* $o \xrightarrow{\sigma} o'$ *and* $\forall a \in X \cap \Sigma : o' \not\xrightarrow{a}$ *and* $\forall p \in X \cap P : \pi_p(o') = \bot$*.* $\mathcal{F}_{DB}(o)$ *denotes the set of all failure pairs and* $\mathcal{F}(DB) := \bigcup_{o \in O} \mathcal{F}_{DB}(o)$ *defines the failure pairs of DB. For a query graph* $\mathcal{Q}$*, define the set of* failure matches *by* $[\![\mathcal{Q}]\!]^{\equiv_{\mathsf{F}}}_{DB} := \{A \preceq DB \mid \mathcal{F}(\mathcal{Q}) = \mathcal{F}(A)\}$.

Failures still only take the topological structure of a graph pattern into account, since our family of property functions may easily be included in a graph-like representation, having also concrete data values (literals) as nodes in the graph database representation (cf. RDF). Equality of sets of failure pairs for different graph databases induces an equivalence relation called *failures equivalence*, denoted by $\equiv_{\mathsf{F}}$. Failures equivalence easily implies trace equivalence, as all the traces of an object and/or database are also enumerated within the respective sets of failure pairs, i. e., it holds that $\mathcal{T}(DB) = \{\sigma \mid (\sigma, X) \in \mathcal{F}(DB)\}$.

Matching by failures equivalence comes with limitations. Consider the two graphs depicted in Fig. 4, which are, in fact, equivalent up to failures. The difference between these two graphs is that in Fig. 4(a), the movie having won an Academy Award (Goldfinger) was directed by a writer, while in Fig. 4(b), only the movie director of the movie without any prizes attached (Quantum of Solace) also is a writer. The reason why failures equivalence does not recognize the difference is that a failure set $X$ only accounts for exactly one step in the graph database. In order to make failures aware of the difference in Fig. 4, we would need that $X$ also contains that after `actedIn` an object (here, Dr. No) does not account for the traces `won` and `directedBy` $\cdot$ `wrote`. For this representation, the graph database in Fig. 4(b) is not a match, since after trace `actedIn` either trace `won` or `directedBy` $\cdot$ `wrote` may be observed. Failures may be adapted by introducing for each $k \in \mathbb{N}$ so-called *$k$-failure pairs* which allow for strings of relation symbols in $X$ of length at most $k$. In practice, this seems infeasible, since respecting $k$-failures amounts to a lot of bookkeeping. Moreover, for each $k$, a

counterexample like the one given in Fig. 4 exists. A more elegant approach may be found in different notions of simulation, being subject of the next subsection.

### 3.2 Simulations

Simulations are not based on traces but rather follow the idea of relating nodes mimicking the behavior of one another. Each graph traversal step must be reflected by some step in the simulating graph. For *graph simulation equivalence*, it is necessary that one graph simulates the other and vice versa. Let us reconsider the example graphs in Fig. 3. The graph database depicted in Fig. 3(a) is a *simulation equivalence match* of the respective graph pattern if, for now, we ignore the properties attached to the objects. To establish this relationship, we need to show that the graph database (Fig. 3(b)) is capable of simulating the pattern (Fig. 3(a)) and vice versa. The respective simulations are represented in Table 1, where $S_1$ represents the former direction and $S_2$ the latter.

$S_1$ is divided into two columns, reading a first column entry is simulated by the second column entry. Whatever relation object "USA" is in, node `C` is capable of simulating it by a respective relation, here `resident` leading to one of three persons, in turn all simulated by node `A`. Consider first "Will Smith" which is in an `actedIn` relationship with some movie object (F1, F2, U1, U2 are placeholders). Also node `A` has an outgoing edge labeled by `actedIn`, targeting node `M`. Furthermore, also the `studiedIn` relationship of the other two objects is simulated by node `A`, this time targeting node `U`. In fact, $S_1$ proves that the graph database is simulated by the graph pattern, since every relation in the graph is reflected by an according step in the pattern.

For the converse direction, $S_2$, we observe that not every node from the graph database is present in this part of the table. This is due to the simulation requirement that every node in the pattern must be simulated by some, not every possible, node in the graph database. Therefore, it is sufficient to only consider a subgraph as simulator. The edge `C` $\xrightarrow{\texttt{resident}}$ `A` is simulated by "USA" $\xrightarrow{resident}$ "Jada P. Smith". Likewise, the other relations of `A` are simulated by the object "Jada P. Smith". Hence, also the graph database simulates the pattern, proving that both graphs are *simulation equivalent*. When reintegrating the requirement

Table 1: Two simulations showing simulation equivalence of the graph database in Fig. 3(a) and the pattern in Fig. 3(b).

| $S_1$: from Fig. 3(a) to Fig. 3(b) | | $S_2$: from Fig. 3(b) to Fig. 3(a) | |
|---|---|---|---|
| USA | `C` | `C` | USA |
| Will Smith | `A` | | |
| Jada P. Smith | `A` | `A` | Jada P. Smith |
| Edgar F. Codd | `A` | | |
| F1 | `M` | | |
| F2 | `M` | `M` | F2 |
| U1 | `U` | `U` | U1 |
| U2 | `U` | | |

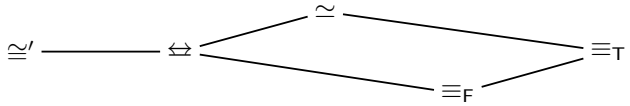


Fig. 5: The Hierarchical Order of the Presented Matching Relations from [7]

that $\psi_{\texttt{yearOfDeath}}(\texttt{A}) \neq \bot$, object "Edgar F. Codd" needs to be excluded from a match.

Formally, a simulation between two graph databases, e. g., a graph pattern and a subgraph, is a relation over the nodes of the first graph database and that of the second. For each pair of nodes $(n_1, n_2)$ in that relation, two conditions must hold. First, if $n_1$ has a defined value for property $p$, i. e., $\pi_p(n_1) \neq \bot$, $n_2$ also has a defined value for $p$. Second, if $n_1 \xrightarrow{a} n_1'$, then there must be a node $n_2'$ such that $n_2 \xrightarrow{a} n_2'$ and $n_2'$ simulates $n_1'$.

**Definition 5.** *Let DB be a graph database, $A = (O, \Sigma, P, \longrightarrow, \Pi)$ a subgraph of DB, and $\mathcal{Q} = (N_\mathcal{Q}, \Gamma, P, \longrightarrow_\mathcal{Q}, \Psi)$ a graph pattern. A simulates $\mathcal{Q}$, denoted $\mathcal{Q} \sqsubseteq_\mathsf{S} A$, iff there is a relation $S \subseteq N_\mathcal{Q} \times O$ such that (a) for every $n \in N_\mathcal{Q}$ there is an $o \in O$ with $(n, o) \in S$ and (b) for every $(n, o) \in S$, it holds that (1) for all $p \in \mathcal{P}$, $\psi_p(n) \neq \bot$ implies $\pi_p(o) \neq \bot$ and (2) $n \xrightarrow{a}_\mathcal{Q} n'$ implies that there is an $o' \in O$ such that $o \xrightarrow{a} o'$ and $(n', o') \in S$. If $A \sqsubseteq_\mathsf{S} \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_\mathsf{S} A$, $A$ and $\mathcal{Q}$ are* simulation equivalent*, denoted $A \simeq \mathcal{Q}$. The set of* simulation equivalence matches *of $\mathcal{Q}$ is defined by $[\![\mathcal{Q}]\!]^{\simeq}_{DB} := \{A \preceq DB \mid A \simeq \mathcal{Q}\}$.*

Compared to the original definitions (cf. [3, 7]), we require that each node $n$ of the graph pattern is simulated by at least one object $o$ of the match. Omitting condition (a) entails simulation equivalence trivially holds between every two graphs, e. g., by means of the empty simulation. Others [14, 16] avoid this issue by requiring maximality of simulation $S$, which means every other simulation is a subset of $S$. This requirement is unnecessarily restrictive for our purposes.

Please note, for establishing simulation equivalence, the respective simulations $S_1, S_2$ do not need to coincide in that $S_1 = S_2^{-1}$. Our example simulations in Table 1 already feature two very different simulations. If, however, there is a simulation $S$ such that $S^{-1}$ is also a simulation, then $S$ is called a *bisimulation*. If between two graph databases there exists a bisimulation, we call them bisimilar ($\leftrightarrows$). So far, all the relations introduced throughout this paper are interrelated as depicted in Fig. 5. A matching relation with a left-to-right path to another matching relation implies that other matching relation. It also means that the sets of matches of graph pattern $\mathcal{Q}$ are ordered by set inclusion. Graph isomorphism is primed ($\cong'$), since the hierarchy is only properly reflected if also graph isomorphism handles undefined properties. Therefore, we simply require for an isomorphism $\nu$ that for each $p \in P$, $\psi_p(n) \neq \bot$ iff $\pi_p(\nu(n)) \neq \bot$.

As already mentioned, simulation equivalence and failures equivalence are incomparable matching relations. We have shown the graphs depicted in Fig. 3 to be simulation equivalent by simulations $S_1$ and $S_2$ (cf. Table 1). As discussed

in the previous subsection, these two graphs differ in their sets of failures, since the graph database contains the failure pair $(\texttt{resident}, \{\texttt{actedIn}\})$ while the graph pattern does not demonstrate this failure pair. Furthermore, the graph databases depicted in Fig. 4 are equivalent up to failures, but simulation tells them apart, as none of the graphs simulates the other. Suppose there is a simulation $S$ showing that the graph in Fig. 4(b) simulates the graph in Fig. 4(a). Since none of the graphs is empty, there must be an object $o$ simulating "Goldfinger", i. e., ("Goldfinger", $o$) $\in S$. There is only one candidate for $o$, namely "Skyfall", since only then relations `directedBy` and `won` may be reflected properly. But then also "Sam Mendes" ought to simulate "Guy Hamilton", which cannot be fulfilled. Thus, the graphs are not equivalent up to simulation, showing the incomparability of simulation equivalence and failures equivalence.

In practice, the choice of the matching relation depends strongly on the underlying data model. For traditional relational databases, usually a *closed-world* is assumed, thus information not being in the database can be seen as *false* [9]. As an example, the absence of the `studiedIn` relation for "Will Smith" in Fig. 3(a) means that "Will Smith" did not attend university. Therefore, a failure query using this information can only return matches with actors that did not attend any university. In the Semantic Web, on the other hand, data is usually interpreted as incomplete [15]. The absence of information does not imply that it cannot be true, but just that the information is not known, at least to the current state of the database. As a consequence, querying an open-world assumption-based database by failures would not be meaningful. In such a scenario, a simulation-based matching relation is preferable. It is even advisable to lose the equivalence matching, resulting in the requirement that a match should only simulate the query, but not vice versa. Without care, the set of matches becomes quite large and, again, less meaningful. Applications and adjustments to the simulation requirement yields reasonable matching sizes [14, 16].

## 4 Beyond Structural Similarity: Query Patterns

In the previous section, we respected properties in the sense that we distinguish value $\perp$ from any other data value, i. e., a Boolean decision. In this section, our goal is to handle concrete data values alongside the graph pattern matching relations, letting the presented framework appear in a relational interpretation of graph database knowledge. We exemplify two ways to achieve this goal by means of simulation, which can easily be adapted to bisimulations and graph isomorphism. First, predicates over properties expressing attribute comparisons are directly attached to the nodes of the query graph. Second, global query predicates are introduced in order to express attribute comparisons between different nodes. Finally, we give a brief discussion on data integration into trace-based matchings.

Consider the example from the introduction with the graph pattern $\mathcal{Q}_a$ in Fig. 1(a). We extend the query, now asking for **actors being children of other actors, who acted in movies released after the year 2000**. Therefore, the

graph pattern needs to be associated with the requirement that a match for `M1` has a release year greater than or equal to 2000, which may be expressed by the first-order formula $\varphi = \texttt{releaseYear} \geq 2000$ attached to node `M1`. Intuitively, the graph database depicted in Fig. 2 constitutes a match for simulation equivalence and the aforementioned property. Formally, however, we need to establish some assumptions before being able to integrate data values into the query matching, yielding two notions of *query pattern*.

The key assumption is that each property $p \in P$ has a *type* $t$, denoted $p \vdash t$, e. g., $\texttt{releaseYear} \vdash \mathbb{Z}$. For concrete data values $c \in \mathcal{D}$, $c \vdash t$ also denotes that $c$ is a constant of type $t$. Furthermore, each type $t$ is equipped with a set of binary predicates $\Theta_t$, e. g., for type $\mathbb{Z}$ we have $\Theta_{\mathbb{Z}} = \{=, \neq, <, \leq, >, \geq\}$ with the expected meaning. As a base, we allow for comparisons of properties $p \in P$ with constants $c \in \mathcal{D}$ by predicates $\theta \in \Theta_t$ as $p\,\theta\,c$ whenever $p$ and $c$ have type $t$. Also properties $p_1, p_2 \in P$ may be compared by $\theta$, i. e., $p_1\,\theta\,p_2$ whenever $p_1, p_2 \vdash t$. We allow the usual propositional connectives of conjunction ($\wedge$) and disjunction ($\vee$) as well as the constant **True** with the usual meaning. The following grammar summarizes the set of formulas we allow for *query predicates* (**QP**):

$$\varphi ::= \textbf{True} \mid p\,\theta\,c \mid p_1\,\theta\,p_2 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where $c \in \mathcal{D}$ and $p, p_1, p_2 \in P$ with $c, p, p_1, p_2 \vdash t$ and $\theta \in \Theta_t$.

Let $\mathcal{Q}$ be a graph pattern with set of nodes $N_{\mathcal{Q}}$. Then function $\Phi : N_{\mathcal{Q}} \to \textbf{QP}$ associates a formula $\varphi$ with every node $n \in N_{\mathcal{Q}}$. We denote by $\varphi_n$ that $\Phi(n) = \varphi$. A *local query pattern* is a pair $(\mathcal{Q}, \Phi)$. A simulation equivalence match $A = (O, \Sigma, P, \longrightarrow, \Psi)$ of $\mathcal{Q}$ features one simulation $S \subseteq N_{\mathcal{Q}} \times O$ showing that $A$ simulates $\mathcal{Q}$. We evaluate the formulas $\varphi_n$ alongside this simulation $S$. $A$ is a simulation equivalence match of $(\mathcal{Q}, \Phi)$ iff (1) $A$ is a simulation equivalence match of $\mathcal{Q}$ and (2) for all $(n, o) \in S$, $o \models \varphi_n$. Thereby, the satisfaction relation is defined inductively over $\varphi_n$ as follows:

- $o \models \textbf{True}$
- $o \models p\,\theta\,c$ if $\psi_p(o) \neq \bot$ and $\psi_p(o)\,\theta\,c$,
- $o \models p_1\,\theta\,p_2$ if $\psi_{p_1}(p) \neq \bot$, $\psi_{p_2} \neq \bot$, and $\psi_{p_1}\,\theta\,\psi_{p_2}$,
- $o \models \varphi_1 \wedge \varphi_2$ if $o \models \varphi_1$ and $o \models \varphi_2$, and
- $o \models \varphi_1 \vee \varphi_2$ if $o \models \varphi_1$ or $o \models \varphi_2$.

In our example, we would associate every node except for `M1` with predicate **True** and $\varphi_{\texttt{M1}} = \texttt{releaseYear} \geq 2000$. Indeed, the graph database depicted in Fig. 2 is a simulation equivalence match for the query pattern $(\mathcal{Q}_a, \Phi)$, because the match of `M1` satisfies the required property of the release year.

Sometimes it is necessary to compare properties of different nodes in a match, e. g., when we ask for **actors that were born in the same year**. Works on data path querying, see Libkin et al. [13] for an overview, included comparisons of data values alongside regular path queries by so-called *binding operators*. Such a binding operator stores a data value at some point of the path, which may later be compared with another data value in another part of the path. Since our setting features a whole graph pattern $\mathcal{Q}$, we may directly access the properties of pattern nodes and compare them. The syntax of such a *global query predicate* is the same except for the access of properties. Instead of referencing $p, p_1, p_2$

directly, we now access them via the node identifiers. Suppose we have nodes `A1` and `A2`. Then predicate `A1.birthYear` = `A2.birthYear` states the aforementioned global requirement. As a consequence, a *query pattern* is a triple $(\mathcal{Q}, \Phi, \varphi_g)$ where $\mathcal{Q}$ is a graph pattern, $\Phi$ a function assigning local predicates to nodes in $\mathcal{Q}$, and $\varphi_g$ is a global query predicate. The local predicates as well as the global predicates need to be satisfied. Please note that by global predicates the local ones get obsolete, formally. However, we believe that from a query modeling perspective, local predicates deserve their existence in our framework.

For bisimilarity and graph isomorphism, the adaptations for simulation equivalence directly carry over to the respective mechanisms in these matching relations, i. e., bisimulations $S$ and isomorphisms $\nu$. In contrast, for trace-based semantics as matchings by failures, integration is not as easy as for simulation-based matchings. One way is surely the already mentioned extensions of regular path queries [12, 13]. For failure pairs $(\sigma, X)$, integrating not only relations and properties in $X$, but also local predicates that must (not) be satisfied by an object with $o \xrightarrow{\sigma} o'$ seems feasible. Alternatively, the local predicates approach could also be integrated. Whether or not path queries can be adapted to work like failures equivalence matching is a subject for future work.

## 5 Related Work

*Graph Pattern Matching* is an extensively studied topic in various domains of computer science [6]. Its applications range from social network analysis, over structural analysis of chemical entities to various applications in the database domain, particularly in graph databases. In these domains, graph pattern matching is usually based on the idea of (sub-)graph isomorphism. Recently, emerging applications showed a trend of studying and using other graph pattern matching relations with the goal of reducing structural requirements of the answer graphs. For example, recent works have implemented the idea of simulation for graph pattern matching [4, 5]. Indeed, experiments have shown advantages of simulation-based matching relations when analyzing social network patterns, as they offer the possibility to collapse several nodes into one node and vice versa.

With regard to *graph databases*, graph isomorphism has become the most common principle for query answering [11]. Almost every graph query language is built on graph pattern matching, using homomorphisms such as graph isomorphism, for retrieving results from the database. Recently, also in this field, different forms of *simulation* with lower runtime complexity compared to subgraph isomorphism have been analyzed to improve performance. On the downside, this performance improvement comes with a loss of topology for graph queries as criticized by Ma et al. [14]. Therefore, topologically more restrictive matching relations, also based on simulation, have been introduced. On the one hand there is dual simulation, a form of simulation that also considers ingoing edges of nodes. Thereby, source and target of a relation are rigorously handled equally important. However, as in the case of simulation, if a graph database contains one dual simulating match, the whole database may be seen as a match. Strong

simulation overcomes this issue by extending dual simulation in such a way that (1) the size of a match is bounded by the diameter of the pattern and (2) all occurring nodes and edges to match nodes and edges in the pattern [14]. However, they do not incorporate data attached to objects in terms of properties.

Similar to the graph queries considered in our work, *approximate graph queries* are not restricted to returning structurally and semantically identical results. Many approximate graph query models rely on measures of pattern similarity by graph edit distances [17, 18]. In comparison to our work, approximate queries might return results with a totally different structure than the query, leading to answers fulfilling only a fraction of the requirements of the original graph pattern. Also, node label similarity is studied for approximate queries [10].

## 6 Conclusion

To the best of our knowledge, this is the first work analyzing the meaning of failures and simulation, both originating from the studies of concurrent processes, compared to the standard matching relations for graph database querying. We provided an overview of the restrictions arising from isomorphic matching and showed the advantages and semantic differences of failures and simulation. Furthermore, we extended the query model by the possibility of also comparing data values, originally motivated by the findings on failures. Please note that the query predicates we introduced are all supported by state-of-the-art graph query languages like SPARQL.

Although not the focus of this paper, we briefly discuss the combined complexity (database and pattern as input) of the presented relations. Compared to graph isomorphism, boiling down to the well-known NP-complete problem of subgraph isomorphism, simulations may be computed in PTIME [8]. For the trace preorder as well as the failures preorder, we are facing the problem of language inclusion for nondeterminstic automata, thus rendering the respective matching problems in PSPACE. For traces, each node is considered initial and final state. In case of failures, each node is an initial state but we add an additional final state $q_f$. Failure sets are now part of the alphabet. For a node $n$ not allowing for edges labeled by symbols in set $X$, add transition $n \xrightarrow{X} q_f$.

The study of concurrent processes provides many more matching relations, summarized in the *linear-time branching-time spectrum* by van Glabbeek [7]. We believe that the existing knowledge on these relations may be used to get new insights on graph database querying. Since the relations in the spectrum are mainly concerned with process equivalence, the domain of graph database queries may also, in return, produce insights, giving birth to not yet studied matching relations. We believe that strong simulation [14], although very close to classical simulations, is such a new relation.

Graph patterns as we use them include a design-decision influencing every theory about graph pattern matching, namely that the pattern relation symbols ($\Gamma$) form a subset of the graph database symbols. Foremost in the Semantic Web, queries are stated without having knowledge about the database internals,

especially the names of the relations and properties. An interesting extension of our work may assume a similarity relation of relation symbols such that traces do not have to be identical, but character-wise *similar*. Furthermore, the notions discussed in this paper may be a starting point for a typing method, helping to align relation and property names.

## References


1. Abriola, S., Barceió, P., Figueira, D., Figueira, S.: Bisimulations on Data Graphs. In: KR 2016. pp. 309–318. AAAI Press (2016)
2. Angles, R., Gutierrez, C.: Querying RDF Data from a Graph Database Perspective. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. pp. 346–360. LNCS, Springer, Berlin, Heidelberg (2005)
3. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. J. ACM 31(3), 560–599 (1984)
4. Brynielsson, J., Högberg, J., Kaati, L., Mårtenson, C., Svenson, P.: Detecting Social Positions Using Simulation. In: ASONAM 2010. pp. 48–55 (2010)
5. Fan, W.: Graph Pattern Matching Revised for Social Network Analysis. In: ICDT 2012. pp. 8–21. ACM, New York, NY, USA (2012)
6. Gallagher, B.: Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. In: Papers from the AAAI FS '06. pp. 45–53 (2006)
7. van Glabbeek, R.J.: The linear time - branching time spectrum. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. pp. 278–297. Springer, Berlin, Heidelberg (1990)
8. Henzinger, M., Henzinger, T., Kopke, P.: Computing simulations on finite and infinite graphs. In: FOCS 1995. pp. 453–462. IEEE Computer Society (1995)
9. Imielinski, T., Jr., W.L.: Incomplete Information in Relational Databases. J. ACM 31(4), 761–791 (1984)
10. Khan, A., Wu, Y., Aggarwal, C.C., Yan, X.: NeMa: Fast Graph Search with Label Similarity. PVLDB Endow. 6(3), 181–192 (Jan 2013)
11. Lee, J., Han, W.S., Kasperovics, R., Lee, J.H.: An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. PVLDB Endow. 6(2), 133–144 (Dec 2012)
12. Libkin, L., Martens, W., Vrgoč, D.: Querying graph databases with XPath. In: ICDT 2013. pp. 129–140. ACM, New York, NY, USA (2013)
13. Libkin, L., Martens, W., Vrgoč, D.: Querying Graphs with Data. J. ACM 63(2), 14:1–14:53 (Mar 2016)
14. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Strong Simulation: Capturing Topology in Graph Pattern Matching. ACM Trans. Database Syst. 39(1), 4:1–4:46 (Jan 2014)
15. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. In: WWW 2007. pp. 807–816. ACM, New York, NY, USA (2007)
16. Mottin, D., Lissandrini, M., Velegrakis, Y., Palpanas, T.: Exemplar Queries: A New Way of Searching. The VLDB Journal 25(6), 741–765 (Dec 2016)
17. Zheng, W., Zou, L., Lian, X., Wang, D., Zhao, D.: Efficient Graph Similarity Search Over Large Graph Databases. IEEE Transactions on Knowledge and Data Engineering 27(4), 964–978 (Apr 2015)
18. Zheng, W., Zou, L., Peng, W., Yan, X., Song, S., Zhao, D.: Semantic SPARQL Similarity Search over RDF Knowledge Graphs. PVLDB Endow. 9(11), 840–851 (Jul 2016)