Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Sobre la complejidad del problema de encontrar data-graph repairs bajo restricciones de nodos y caminos

Tesis de Licenciatura en Ciencias de la Computación

Santiago Cifuentes

Directora: María Vanina Martínez
Codirector: Ricardo Oscar Rodriguez
Buenos Aires, 2021

SOBRE LA COMPLEJIDAD DEL PROBLEMA DE ENCONTRAR DATA-GRAPH REPAIRS BAJO RESTRICCIONES DE NODOS Y CAMINOS

Las bases de datos con forma de grafo representan de una forma efectiva relaciones binarias entre entidades, y permiten procesar y consultar por conexiones no triviales de forma eficiente. Como en el caso relacional, se espera que los datos preserven un conjunto de *restricciones de integridad* que capturen la estructura semántica del mundo que representan. Un posible enfoque para lidiar con bases de datos que no satisfacen su conjunto de reglas de integridad consiste en remplazarlas por una nueva base de datos 'similar' a la original, pero que satisfaga el conjunto de restricciones. Es decir, un *repair* de la base de datos original. En este trabajo estudiamos el problema de computar (subset y superset) repairs de bases de datos con forma de grafo con datos en los nodos usando una noción de consistencia basada en conjuntos de expresiones del lenguaje Reg-GXPath, interpretadas como restricciones de integridad. Demostramos que para los fragmentos positivos de Reg-GXPath estos problemas admiten algoritmos polinomiales mientras que el poder expresivo completo del lenguaje vuelve el problema intratable. Finalmente, también estudiamos el problema de computar *preferred repairs* sobre dos criterios de preferencia distintos, mostrando que en la mayoría de los casos la complejidad computacional del problema no cambia.

Palabras claves: data-graph, database repair, path constraints, data cleaning, Reg-GXPath

ON THE COMPLEXITY OF FINDING DATA-GRAPHS REPAIRS UNDER PATH AND NODE CONSTRAINTS

Graph databases embrace in an effective way relationships among data and allows to process and query these connections efficiently. As in the relational case, it is expected that data preserves a set of integrity constraints that define the semantic structure of the world it represents. When a database does not satisfy its integrity constraints, a possible approach is to search for a 'similar' database that does satisfy the constraints, a.k.a a repair. In this work we study the problem of computing (subset and superset) repairs for graph databases with data values using a notion of consistency based on a set of Reg-GXPath expressions as integrity constraints. We show that for positive fragments of Reg-GXPath these problems admit a polynomial time algorithm while the full expressive power of the language renders them intractable. Finally, we also study the problem of computing preferred repairs based on two different preference criteria, showing that in most cases the computation complexity is not increased.

Key Words: data-graph, database repair, path constraints, data cleaning, Reg-GXPath

AGRADECIMIENTOS

Antes de comenzar, agradezco enormemente a Vanina y Ricardo por haberme aceptado como tesista (e incluso como pasante) en aquel llamado a Pasantes de Investigación en 2020, a pesar de recién haber terminado el ciclo medio de la carrera. Lxs dos me apoyaron continuamente en los tormentosos años de la pandemia, y me ayudaron a dar mis primeros pasos en el mundo de la investigación. Gracias a ellxs pude descubrir este fascinante mundo, del cual poco a poco logro ser parte.

También agradezco a Sergio Abriola por sus enormes aportes a este trabajo, y a Edwin Pin y Nina Pardal por sus valiosas correcciones y comentarios. Lxs tres son personas maravillosas, y hoy en día tengo la suerte de poder trabajar codo a codo junto a ellxs.

Por último, agradezco a todo el Departamento de Computación. En primer lugar, por la excelentísima educación que me dio. Disfruté profundamente cada materia, gracias tanto a la calidad de las clases como también por la pasión que cada docente puso en darlas. Ese entusiasmo definitivamente llega hasta el alumnado, y es justamente lo que permitió que ahora estén leyendo estas palabras. En segundo lugar, por el Programa de Pasantías, que considero un recurso espectacular para estimular la entrada de lxs estudiantes al mundo de la Investigación en Computación. Confío en que hoy en día ya se están cosechando sus frutos, y que va a seguir dándonos alegrías por muchos años más.

Índice general

1. INTRODUCCIÓN

Las bases de datos en forma de grafo son útiles en muchas aplicaciones modernas en donde la topología de los datos (entendida como las relaciones que se mantienen entre distintas entidades) es más importante que los datos en sí mismos. Algunas de estas son el análisis de las redes sociales [19], de la procedencia de los datos (*data provenance* [3]) y la Web Semántica [6].

La estructura de la base de datos suele ser consultada mediante lenguajes de navegación como *regular path queries* o RPQs [9] que pueden capturar pares de nodos conectados mediante algún tipo específico de camino. En el caso de las RPQs, estos caminos se definen de forma similar a expresiones regulares sobre un alfabeto de etiquetas que tienen los ejes (y de ahí el nombre). Estos lenguajes de consulta suelen ser extendidos con herramientas que aumentan su expresividad, usualmente agregando complejidad al proceso de evaluación. Por ejemplo, las C2RPQs son una extensión natural de las RPQs que se definen agregando al lenguaje RPQ la capacidad de atravesar ejes en ambas direcciones (de ahí el "2" en C2RPQ) y cerrando el conjunto de expresiones bajo la conjunción (de forma similar al caso de las *Conjunctive Queries* del modelo relacional), permitiendo aparte el uso de variables libres.

Las RPQs y sus extensiones más comunes (C2RPQs o NREs [8]) solo pueden operar con los ejes del grafo, dejando de lado cualquier tipo de interacción con los datos que contengan los nodos. Esto llevó al diseño de lenguajes de consulta para el caso de *data-graphs* (i.e. bases de dato en forma de grafo donde la información está tanto en los caminos como en los propios nodos), como las RMEs y Reg-GXPath [26].

Como en el caso relacional, es usual esperar que los datos preserven una estructura semántica relacionada al mundo que representan. Estas *restricciones de integridad* pueden ser expresadas en bases de datos con forma de grafo mediante *path constraints* [1, 15].

Dada la gran interoperabilidad y distribución sobre este tipo de bases de datos, es esperable que eventualmente las mismas no satisfagan las restricciones modeladas sobre ellas. Luego, un posible enfoque para recuperar la consistencia de la información reside en buscar una base de datos "similar" que sí satisfaga las restricciones. En la literatura, a esta nueva base de datos se la denomina *repair* [5], y para definirla apropiadamente hay que decir con precisión qué significa que una base de datos sea "similar" a otra.

Tomemos por ejemplo el data-graph de la figura 1.1: los nodos representan personas, mientras que los ejes codifican distintos tipos de relaciones entre ellas (como la hermandad o paternidad). Notemos que en el data-graph todo par de nodo (x, y) conectado por un eje HERMANE_DE dirigido de x a y también tiene un eje HERMANE_DE dirigido de y a x. Esto es razonable, dado que la relación de hermandad suele ser simétrica. También notemos que los nodos (MAURO, JULIETA) están conectados a través de un eje SOBRINE_DE dirigido hacia JULIETA, y esto es algo esperable teniendo en cuenta que el padre de Mauro, Diego, es hermano de Julieta.

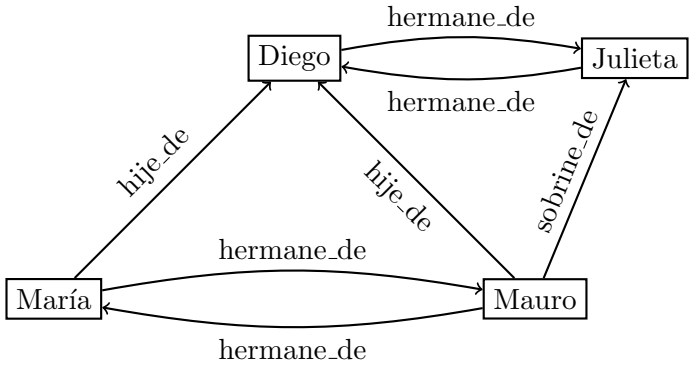

Fig. 1.1: Un ejemplo de un data-graph donde los nodos representan personas y los ejes relaciones familiares.

eleusLa estructura que este data-graph particular tiene que preservar para capturar adecuadamente nuestras nociones de hermandad y de parentesco familiar pueden ser descriptas a través de *path expressions* que capturan aquellos pares de nodos que satisfagan las restricciones semánticas. En particular el par (María, Julieta) no satisface la semántica de la relación sobrine_de, dado que Diego también es padre de María, pero no hay ningún eje sobrine_de desde María hacia Julieta. Esto puede "arreglarse" agregando el eje sobrine_de desde María hacia Julieta, o bien removiendo el eje hije_de dirigido de María a Diego. Naturalmente es preferible preservar la mayor cantidad posible de información del data-graph original.

En la literatura existen diferentes nociones de repairs. Entre otras, están los *set-based repairs* [35], *attribute-based repairs* [37], y los *cardinality-based repairs* [28]. En este trabajo estudiamos dos restricciones al problema de encontrar un set-based repair G' de un data-graph G bajo un conjunto de expresiones Reg-GXPath R consideradas como *path constraints*: cuando G' es un subgrafo de G y cuando G' es un supergrafo de G. Estos tipos de repairs suelen ser llamados *subset* y *superset* repairs respectivamente [35, 7]. Dado que puede haber más de un repair dado un grafo y un conjunto de reglas, es posible definir un orden sobre el conjunto de repairs e intentar encontrar el 'óptimo' bajo ese orden en vez de uno cualquiera [22, 34]. Por ende, también estudiamos el problema de encontrar un *preferred repair* en base a dos sistemas de preferencias distintos.

Las principales contribuciones de este trabajo son:

- Definimos un modelo para bases de datos con forma de grafo con datos en los nodos e introducimos una noción de consistencia basada en conjuntos de expresiones de Reg-GXPath que capturan un grupo importante de restricciones de integridad ya estudiadas en la literatura.

- Estudiamos el problema de, dada una base de datos en forma de grafo G y un conjunto de restricciones R, computar un subset repair (o respectivamente un superset repair) G' de G.

- Mostramos que dependiendo de la expresividad de las restricciones (dependiendo de si usan el poder completo de Reg-GXPath o bien su fragmento sin negación, conocido como Reg-GXPath^{pos}), estos problemas admiten un algoritmo polinomial o bien se vuelven intratables (en el peor caso tenemos una cota inferior de Exptime). Los resultados se resumen en la tabla 6.1

- Finalmente, desarrollamos un estudio de dos criterios de preferencia ($w-$*preferred* y *multiset-preferred repairs*) que definen un orden sobre el conjunto de repairs; y probamos que, excepto por un caso puntual, el problema de computar un repair prioritario en función del orden tiene la misma complejidad que el problema de buscar un repair cualquiera. Estos resultados están resumidos en la tabla 6.2.

El resto del trabajo se organiza de la siguiente forma. En la sección 2 introducimos la notación y los conocimientos preliminares necesarios para definir la sintaxis y la semántica de nuestro modelo de data-graph y de Reg-GXPath^{pos}, así como también la noción de consistencia y los distintos subtipos de repairs. También mostramos, en base a ejemplos, que el lenguaje propuesto captura un conjunto de restricciones de integridad que son comunes en la literatura sobre bases de datos semi estructuradas. En la sección 3 estudiamos la complejidad del problema de computar tanto subset como superset repairs. Luego, en la sección 4, desarrollamos dos propuestas distintas para asignar preferencias sobre los repairs; una basada en la asignación de pesos a los datos y a los tipos de ejes y otra basada en elevar un orden sobre los ejes y los datos a un ordenamiento sobre multiconjuntos (viendo luego a los data-graphs como multiconjuntos de ejes y datos). Para ambos casos estudiamos la complejidad computacional del problema de computar un preferred repair. Finalmente, en la sección 5 y 6 se muestran respectivamente trabajos relacionados a este y algunas conclusiones generales. La sección 8 es un apéndice con algunas demostraciones tediosas pero necesarias para los resultados finales.

2. DEFINICIONES

Fijemos un conjunto finito Σ_e de *edge labels* (i.e. etiquetas o nombres para los ejes) y un conjunto contable Σ_n de *node labels* (i.e. etiquetas o nombres para los nodos), los cuales asumimos no vacíos y disjuntos. Un grafo de datos o *data-graph* G es una tupla (V, L, D) donde V es el conjunto de nodos de G, $L : V \times V \to \mathcal{P}(\Sigma_e)$ una función que denota los ejes del grafo y $D : V \to \Sigma_n$ una función que denota los *data values* asociados a cada nodo.

Por ejemplo, si tenemos $\Sigma_e = \{e_1, e_2\}$, $\Sigma_n = \{c_1, c_2\}$ y $V = \{v_1, v_2, v_3, v_4\}$ y definimos para los ejes

$L(v_1, v_2) = \{e_1\}$
$L(v_1, v_3) = \{e_1, e_2\}$
$L(v_2, v_3) = \{e_2\}$
$L(v_3, v_2) = \{e_1\}$
$L(v_i, v_j) = \emptyset$ para cualquier otro caso no nombrado

y para los data values

$D(v_1) = D(v_2) = c_1$
$D(v_3) = D(v_4) = c_2$

obtenemos el data-graph de la figura 2.1.

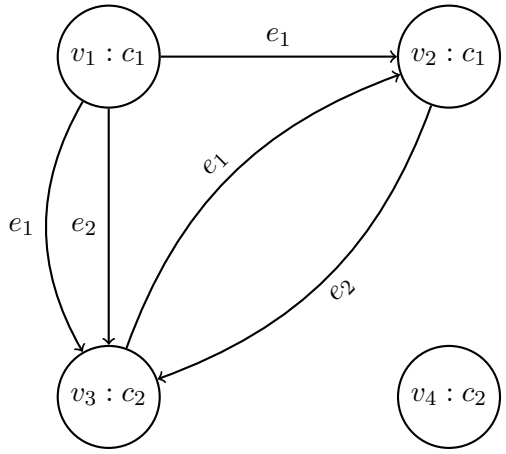

Fig. 2.1: Ejemplo de data-graph

Notemos que este modelo permite *loops* así como también multiejes entre dos nodos siempre y cuando las labels de los ejes sean distintas. Por simplicidad, en el resto de los ejemplos gráficos no anotaremos las etiquetas de los nodos (i.e. los v_i), y a cambio asignaremos a cada nodo un *data value* único para poder distinguirlos (pero remarcamos que en un data-graph cualquiera varios nodos pueden tener un mismo data value).

El lenguaje Reg-GXPath tiene dos tipos de expresiones: de camino y de nodo. Ambos tipos de expresiones se definen por recursión mutua en función de las del otro tipo. Las expresiones de camino o *path expressions* vienen dadas por la gramática:

$$\alpha, \beta = \epsilon \mid \_ \mid \mathrm{A} \mid \mathrm{A}^- \mid [\varphi] \mid \alpha \,.\, \beta \mid \alpha \cup \beta \mid \alpha \cap \beta \mid \alpha^* \mid \overline{\alpha} \mid \alpha^{n,m}$$

donde A itera sobre el conjunto Σ_e y φ es una expresión de nodo o *node expression* definida por la gramática:

$$\varphi, \psi = \neg\varphi \mid \varphi \wedge \psi \mid \langle\alpha\rangle \mid \mathrm{c}^{=} \mid \mathrm{c}^{\neq} \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle \mid \varphi \vee \psi$$

donde c itera sobre el conjunto Σ_n y α, β son path expressions.

La semántica de estas expresiones es definida en [26] extendiendo a otros lenguajes diseñados para navegar sobre grafos. En particular Reg-GXPath puede verse como una extensión de los lenguajes regulares nombrados en [9] agregando operadores de negación como $\overline{\alpha}$ y *data tests* como $\mathrm{c}^{=}$ o $\mathrm{c}^{\neq}$. El operador $\langle\alpha\rangle$ ya fue definido para un lenguaje llamado *nested regular expressions* o NRE estudiado en [8]. Dado un data-graph $G = (V, L, D)$ la semántica de estas expresiones se define de la siguiente forma:

$$[\![\epsilon]\!]_G = \{(v, v) \mid v \in V\}$$
$$[\![\_]\!]_G = \{(v, w) \mid v, w \in V, L(v, w) \neq \emptyset\}$$
$$[\![\mathrm{A}]\!]_G = \{(v, w) \mid \mathrm{A} \in L(v, w)\}$$
$$[\![\mathrm{A}^-]\!]_G = \{(w, v) \mid \mathrm{A} \in L(v, w)\}$$
$$[\![\alpha^*]\!]_G = \text{la clausura reflexo-transitiva de } [\![\alpha]\!]_G$$
$$[\![\alpha.\beta]\!]_G = \{(v, w) \mid \exists z \in V, (v, z) \in [\![\alpha]\!]_G, (z, w) \in [\![\beta]\!]_G\}$$
$$[\![\alpha \cup \beta]\!]_G = [\![\alpha]\!]_G \cup [\![\beta]\!]_G$$
$$[\![\alpha \cap \beta]\!]_G = [\![\alpha]\!]_G \cap [\![\beta]\!]_G$$
$$[\![\overline{\alpha}]\!]_G = V \times V \setminus [\![\alpha]\!]_G$$
$$[\![[\varphi]]\!]_G = \{(v, v) \mid v \in [\![\varphi]\!]_G\}$$
$$[\![\alpha^{n,m}]\!]_G = \bigcup_{k=n}^{m} ([\![\alpha]\!]_G)^k$$
$$[\![\langle\alpha\rangle]\!]_G = \pi_1([\![\alpha]\!]_G) = \{v \mid \exists w \in V\, (v, w) \in [\![\alpha]\!]_G\}$$
$$[\![\neg\varphi]\!]_G = V \setminus [\![\varphi]\!]_G$$
$$[\![\varphi \wedge \psi]\!]_G = [\![\varphi]\!]_G \cap [\![\psi]\!]_G$$
$$[\![\varphi \vee \psi]\!]_G = [\![\varphi]\!]_G \cup [\![\psi]\!]_G$$
$$[\![\mathrm{c}^{=}]\!] = \{v \in V \mid D(v) = c\}$$
$$[\![\mathrm{c}^{\neq}]\!] = \{v \in V \mid D(v) \neq c\}$$
$$[\![\langle\alpha = \beta\rangle]\!]_G = \{v \in V \mid \exists v', v'' \in V, (v, v') \in [\![\alpha]\!]_G, (v, v'') \in [\![\beta]\!]_G, D(v') = D(v'')\}$$
$$[\![\langle\alpha \neq \beta\rangle]\!]_G = \{v \in V \mid \exists v', v'' \in V, (v, v') \in [\![\alpha]\!]_G, (v, v'') \in [\![\beta]\!]_G, D(v') \neq D(v'')\}$$

En esta semántica los operadores π_1 y π_2 son respectivamente las proyecciones de la primera y segunda componente sobre pares (i.e. $\pi_1((x,y)) = x$ y $\pi_2((x,y)) = y$), y $[\![\alpha]\!]^k$ es la concatenación de α consigo misma k veces.

Usaremos $\alpha \Rrightarrow \beta$ para denotar la expresión $\overline{\alpha} \cup \beta$, y $\varphi \Rightarrow \psi$ para denotar la expresión $\neg\varphi \vee \psi$. Para facilitar la distinción entre expresiones de camino y expresiones de nodo también notamos como $\downarrow_{\text{A}}$ a la expresión A. Por ejemplo, la expresión $\text{HIJE\_DE}\ [\text{MARIA}^{=}]$ se notará como $\downarrow_{\text{HIJE\_DE}}\ [\text{MARIA}^{=}]$. Aparte, omitiremos el subíndice G cuando se entienda por contexto sobre qué data-graph se evalúa la expresión.

Ejemplo 1. Si se usan data-graphs para representar una red social luego es posible definir una expresión que capture a los "amigos transitivos" de la red. Supongamos que los nodos representan usuarios, y que a través de los ejes se definen relaciones de amistad con un edge label AMIGE_DE. Entonces la expresión

$$\alpha_1 = \downarrow^*_{\text{AMIGO\_DE}}$$

captura a todos los pares de nodos (v, w) tales que hay un camino de usuarios "amigos" desde v a w.

Esta expresión puede ser definida sobre la mayoría de lenguajes de navegación sobre grafos dado que casi todos admiten algún tipo de operador análogo al de la estrella de Kleene (uno de los principales fuertes de este tipo de bases de datos es que permiten consultar por clausuras transitivas de forma intuitiva y eficiente). En el caso de Reg-GXPath también podemos aplicar condiciones que operen sobre subcaminos. Supongamos que queremos restringirnos a los pares de usuarios que aparte de ser amigos transitivos también sigan a un cierto usuario distinguido x. Si la relación SIGUE_A está codificada sobre los ejes, entonces podemos definir la expresión

$$\alpha_2 = \langle \downarrow_{\text{SIGUE\_A}}\ [\text{X}^{=}] \rangle\ \downarrow^*_{\text{AMIGO\_DE}}\ \langle \downarrow_{\text{SIGUE\_A}}\ [\text{X}^{=}] \rangle$$

que captura justamente el conjunto que queremos.

Este ejemplo no podría haber sido construido fácilmente con la mayoría de los lenguajes para navegar sobre grafos debido a que estos no admiten ningún tipo de comparación o evaluación de valores en los nodos. Incluso en muchos de estos lenguajes ni siquiera hubiéramos podido evaluar subcaminos de nuestro camino principal.

Ejemplo 2. Dada una red social similar a la anterior en la que los labels codifican relaciones familiares (como por ejemplo PADRE_DE) podemos capturar aquellos nodos que no tengan un padre, o abuelo, o bisabuelo, o ... con mismo nombre que si mismos mediante la expresión:

$$\alpha_3 = \neg\langle \epsilon = \downarrow^+_{\text{PADRE\_DE}} \rangle$$

En esta expresión el ϵ de la izquierda denota el camino vacío, por lo que captura al mismo nodo del que parte el camino. Por otro lado, la expresión de la derecha captura a todo padre (o padre transitivo del padre) del nodo inicial. Luego hacemos la comparación de los valores dentro de los nodos capturados para ver si tienen un mismo nombre o no (suponiendo que el dato guardado en los nodos es justamente el nombre de la persona).

Notemos que en caso de que nuestro modelo requiera almacenar más de un valor en cada nodo (por ejemplo, nombre, apellido, DNI, etc) se puede extender la definición de los

data-graphs incorporando más edge labels y data values. Más formalmente, supongamos que tenemos un modelo de base de datos con forma de grafo análogo al nuestro salvo por que cada nodo tiene una tupla de atributos determinados con nombres A_1, A_2,...,A_t en los que cada conjunto tiene un dominio Σ_n^i para $1 \leq i \leq t$. Luego, podemos traducir ese framework al nuestro si definimos $\Sigma_n = \bigcup_{i=1}^{t} \Sigma_n^i$ y agregamos nuevos edge labels A_i por cada atributo posible sobre los nodos. Finalmente cualquier acceso a un valor A_i de un nodo v es reemplazado por el acceso al nodo w alcanzable desde v por un eje con label A_i.

Ejemplo 3. Volviendo al caso del ejemplo 2 la fórmula se podría extender como:

$$\alpha_4 = \neg\langle\downarrow_{\text{NOMBRE}}=\downarrow^{+}_{\text{PADRE\_DE}}\downarrow_{\text{NOMBRE}}\rangle$$

Hay dos subconjuntos de Reg-GXPath que han sido ampliamente estudiados en la literatura:

- El primero es Core-GXPath, el cual se obtiene si solo se permite aplicar la estrella de Kleene sobre path expressions que sean de la forma A o A^{-1} (es decir, se reemplaza la regla de producción α^* por A^* y $(\text{A}^{-1})^*$). Esta restricción busca limitar la estructura que puedan tener caminos arbitrariamente largos definidos por las expresiones del lenguaje, facilitando luego el cómputo de las clausuras transitivas.

- El otro fragmento importante es Reg-GXPath^{pos}, o fragmento positivo, el cual se obtiene quitando las reglas de negación $\neg\varphi$ y $\overline{\alpha}$. Esta restricción es común en el contexto de restricciones de integridad: los lenguajes que no tienen negación suelen satisfacer la regla de *monotonía*. Se dice que un lenguaje $\mathcal{L}$ tiene la propiedad de monotonía cuando para toda expresión $\mu \in \mathcal{L}$ vale que si una estructura E satisface μ luego cualquier estructura E' que contenga a E también satisface μ.

Notemos que la gramática de Reg-GXPath es redundante sobre algunos operadores como $\cup$, el cual puede ser simulado empleando complemento de caminos y $\cap$. No obstante esta redundancia nos resulta útil al momento de recortar el poder expresivo del lenguaje. Si no tuviéramos $\cap$ en la gramática entonces Reg-GXPath^{pos} no sería capaz de tomar la unión de expresiones de camino positivas.

Durante este trabajo estaremos especialmente interesados en el fragmento positivo. En particular veremos que considerando solo expresiones de ese conjunto muchos problemas de razonamiento sobre data-graphs se vuelven tratables.

Como ya comentamos, las bases de datos buscan representar una porción de interés del universo, y por lo tanto es esperable que estas respeten determinadas reglas semánticas que se satisfacen en el universo representado. Por ejemplo: si una base de datos representa los nexos familiares entre distintos individuos entonces la relación de hermandad que la misma codifica debería ser simétrica.

Es común reforzar estas restricciones sobre la base de datos con **restricciones de integridad** o *integrity constraints* que marquen como "inválidas" a aquellas bases de datos que no respeten las reglas semánticas del universo representado. Dado un conjunto de restricciones de integridad I y una base de datos D se suele decir que D es consistente respecto a I si satisface las restricciones de I. Caso contrario, se dice que D es inconsistente respecto a I.

En nuestro contexto la noción de consistencia será definida sobre restricciones de integridad expresadas mediante el lenguaje Reg-GXPath:

Definición 4 (Consistencia)**.** Sea G un data-graph y $R = P \cup N$ un conjunto de restricciones donde P es un conjunto de path expressions de Reg-GXPath y N es un conjunto de node expressions de Reg-GXPath. Decimos que G es consistente respecto a R (notado como $G \models R$) o bien que el par (G, R) es consistente si:

- $\forall x \in V_G$ y $\varphi \in N$ vale que $x \in [\![\varphi]\!]_G$
- $\forall x, y \in V_G$ y $\alpha \in P$ vale que $(x, y) \in [\![\alpha]\!]_G$

Caso contrario decimos que G es inconsistente respecto a R, o bien que el par (G, R) es inconsistente.

Durante el resto del trabajo diremos que G es (in)consistente sin especificar al conjunto de restricciones cuando este se entienda por contexto.

Ejemplo 5. Sea G un data-graph que representa relaciones de la industria del cine. En particular, hay nodos que representan a individuos de la industria del cine (como actores, directores, etc) y otros representando películas o documentales. Si quisiéramos recortar la base de datos y quedarnos con el subconjunto de actores que hayan trabajado con Philipp Seymour Hoffman en una película de Paul Thomas Anderson luego estaríamos interesados en un subconjunto de la base de datos que satisfaga la expresion:

$$\varphi = \langle \downarrow_{\text{TIPO}} [actor^{=}] \rangle \Rightarrow \langle \downarrow_{\text{ACTÚA\_EN}} \langle \downarrow_{\text{DIRIGIDA\_POR}} [Anderson^{=}] \rangle \downarrow^{-}_{\text{ACTÚA\_EN}} [Hoffman^{=}] \rangle$$

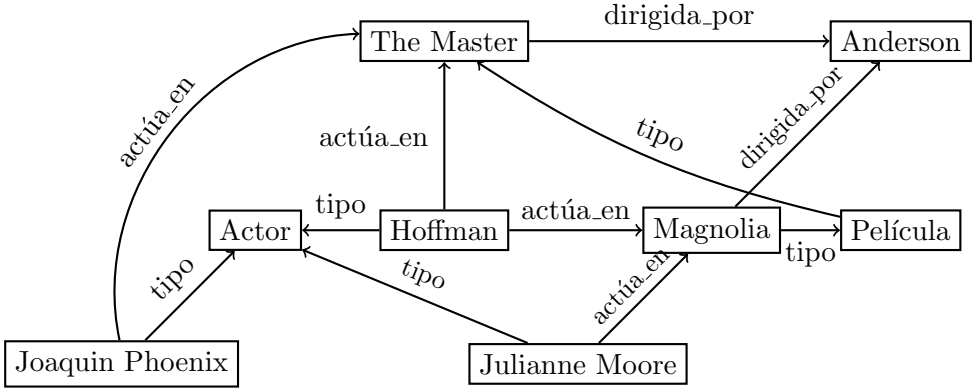

Fig. 2.2: En este data-graph φ es satisfecha dado que tanto Phoenix como Moore trabajaron con Hoffman en una película dirigida por Anderson (respectivamente en "The Master" y "Magnolia"). Notemos que la restricción también aplica a Hoffman, por lo que este tiene que estar en alguna película de Hoffman para satisfacer la restricción.

Ejemplo 6. Todas las *Regular Path Constraints* (RPCs) consideradas en [7] pueden ser escritas como expresiones de Reg-GXPath dado que el lenguaje permite implicaciones de la forma $\alpha_1 \Rrightarrow \alpha_2$ donde α_i es una expressión 2RPQ para $i \in \{1, 2\}$ (incluso podrían ser NREs). En particular las restricciones del primer ejemplo de [7] se pueden expresar en Core-GXPath de la siguiente forma:

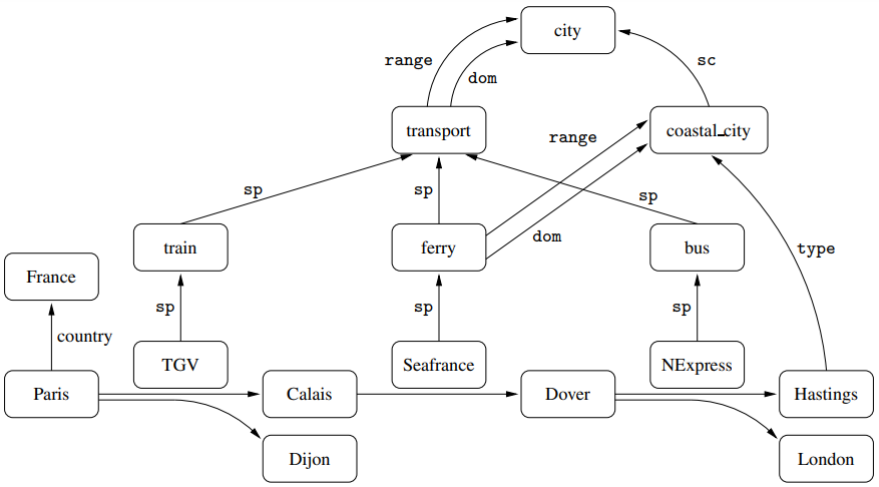

Fig. 2.3: Ejemplo de data-graph tomado de [32]. Notemos que en este data-graph cada eje entre ciudades tiene un nodo asociado que codifica características propias del eje. Visualmente este nodo está arriba del ejes, pero formalmente dado un eje $e = (x, y)$ entre ciudades hay un nodo z_e tal que (x, EDGE, z_e) y (z_e, NODE, y) son ejes del grafo. Por ejemplo, el nodo asociado al eje que une *Paris* y *Calais* es el nodo con dato TGV. Por lo tanto, una expresión puede interactuar con ese nodo si baja por un eje $\downarrow_{\text{EDGE}}$ desde *Paris*.

- La simetría de la relación HERMANE_DE se puede definir como $\alpha = \downarrow_{\text{HERMANE\_DE}} \Rrightarrow \downarrow_{\text{HERMANE\_DE}}^{-}$.
- La condición del eje SOBRINE_DE se puede definir como $\beta = \downarrow_{\text{HIJE\_DE}} \downarrow_{\text{HERMANE\_DE}} \Rrightarrow \downarrow_{\text{SOBRINE\_DE}}$.

Estas son justamente las condiciones que queríamos expresar en la figura 1.1. Ahora podemos establecer formalmente que ese data-graph es inconsistente respecto a β porque el par (MARIA, JULIETA) no pertenece a $[\![\beta]\!]$. Por otro lado, todo par de nodos del grafo pertenece al conjunto $[\![\alpha]\!]$.

Ejemplo 7. También es posible expresar restricciones de integridad análogas a las *foreign keys* o claves foráneas del modelo relacional, relacionando valores de distintas entidades. Consideremos una base de datos con información sobre ciudadanos y ciudades. Sería razonable esperar que la nacionalidad de todo ciudadano coincida con el país al que pertenece la ciudad en la que nació. Esto podría ser expresado con la fórmula

$$\varphi_1 = \langle \downarrow_{\text{TIPO}} [\text{PERSONA}^{=}] \rangle \Rightarrow \langle \downarrow_{\text{NACIO\_EN\_CIUDAD}} \downarrow_{\text{NACION}} = \downarrow_{\text{NACIONALIDAD}} \rangle$$

Ejemplo 8. Como el conjunto de expresiones NREs es un subconjunto de Reg-GXPath [26] entonces es posible expresar múltiples restricciones propias del RDF (*Resource description framework*) como aquellas consideradas en [32]. Por ejemplo, considerando la base de datos con forma de grafo empleada en la figura 1 (tomada de [32]), podemos definir la restricción de que debería ser posible llegar de Paris a cualquier otra ciudad de Francia por medio de trenes con la fórmula:

$$\varphi_2 = \langle \downarrow_{\text{COUNTRY}} [France^{=}] \rangle \Rightarrow \langle (\downarrow_{\text{EDGE}} \langle \downarrow_{\text{SP}}^{*} [train^{=}] \rangle \downarrow_{\text{NODE}})^{*} [Paris^{=}] \rangle$$

En esta base de datos es posible acceder a un nodo que representa la "entidad" de cada eje mediante la arista EDGE, y mediante una arista llamada NODE moverse hacia el nodo al

que apunta el eje. Esto permite agregar al modelo información acerca de la estructura de las relaciones codificadas en los ejes: como se puede ver en la fórmula φ_2, el consecuente se cumple si es posible llegar al nodo con nombre PARIS mediante ejes que tengan como subpropiedad ser un tren. Esta condición de propiedad se codifica mediante el eje $\downarrow_{\text{SP}}$ y la estrella de Kleene: esto permite organizar una jerarquía hereditaria de propiedades sobre los ejes y recorrerla a través del lenguaje de consulta. Por ejemplo: un cierto transporte podría ser de tipo TREN_ALTA_VELOCIDAD y tener como subpropiedad ser un TREN.

Cuando un data-graph G es inconsistente respecto a un conjunto de restricciones R podríamos estar interesados en computar un nuevo data-graph G' que satisfaga R y que difiera lo menos posible de G. Esta nueva base de datos G' es usualmente llamada *repair* de G con respecto a R, y la forma que tenga dependerá de como se defina la condición de "diferir mínimamente".

Para definir esta condición emplearemos una noción basada en conjuntos, en donde la diferencia entre G y G' se mide en función de la cantidad de nodos o ejes que hay que agregar o borrar a G para obtener G'. Este tipo de repairs suelen denominarse *set repairs* y se definen sobre el operador de diferencia simétrica sobre las bases de datos (ver [7, 31, 35]), lo cual permite modelar situaciones de agregado y borrado de información en simultáneo. En nuestro caso, las familias de data-graph repairs definidas mediante este operador son muy complejas y dificultan los problemas que definiremos más adelante, por lo cual solo daremos la noción de repair para casos en que se permite borrar datos o bien agregarlos (pero no ambos a la vez).

Decimos que un data-graph $G = (V, L, D)$ es un **subset** o subgrafo de $G' = (V', L', D')$ (notado como $G \subseteq G'$) si y solamente si $V \subseteq V'$ y para todo par de nodos $v, v' \in V$ vale que $L(v, v') \subseteq L'(v, v')$ y $D(v) = D(v')$. Si la inclusión de los conjuntos V y V' o $L(v, v')$ y $L'(v, v')$ es estricta lo notamos como $G \subset G'$. Aparte decimos que G' es un **superset** de G.

Definición 9 (Subset y superset repairs)**.** Sea R un conjunto de restricciones y G un data-graph. Decimos que G' es un **subset repair** (respectivamente, un **superset repair**) o $\subseteq$-repair (respectivamente, $\supseteq$-repair) de G si:

- (G', R) es consistente (i.e. $G' \models R$)
- $G' \subseteq G$ (respectivamente, $G' \supseteq G$).
- No existe ningún data-graph G'' tal que (G'', R) sea consistente y $G' \subset G'' \subseteq G$ (respectivamente, $G' \supset G'' \supseteq G$).

Notamos al conjunto de subset repairs de G con respecto a R como $\subseteq$-$Rep(G, R)$ (respectivamente, $\supseteq$-$Rep(G, R)$).

Ejemplo 10. En el ejemplo de la figura 1.1 se obtiene un subset repair al eliminar el eje (MARIA, hije_de, DIEGO), mientras que agregar el eje (MARIA, SOBRINE_DE, DIEGO) resulta en un $\supseteq$-repair. Borrando los dos ejes (DIEGO, HERMANE_DE, JULIETA) y (JULIETA, HERMANE_DE, DIEGO) también se obtiene un $\subseteq$-repair.

3. ENCONTRANDO REPAIRS

¿Cuál es la complejidad computacional del problema de computar, dado un data-graph G y un conjunto de restricciones R, un repair de G con respecto a R? En las siguientes secciones formalizaremos el problema y lo estudiaremos. Naturalmente, la dificultad del mismo dependerá del subconjunto de Reg-GXPath que permitamos para formar el conjunto de restricciones, y también si nos restringimos a subset o bien a superset repairs.

Antes que nada, mencionamos un resultado de [26] relacionado con la evaluación de expresiones de Reg-GXPath:

Teorema 11. *Dada una expresión de Reg-GXPath μ y un data-graph G es posible computar el conjunto $[\![\mu]\!]_G$ en tiempo polinomial en función de G y μ.*

Demostración. El conjunto $[\![\mu]\!]_G$ se puede computar eficientemente haciendo programación dinámica sobre el árbol de μ de forma bottom-up. Si para cada subexpresión necesitamos $O(|V^3|)$ operaciones (suponiendo que ya están resueltas las subexpresiones de esta) luego tenemos un algoritmo que realiza $O(|V^3| \times |\mu|)$ operaciones.

Los casos base se resuelven trivialmente siguiendo su definición. Para los casos inductivos hacemos lo siguiente:

- Para las expresiones de camino tenemos:
  - $[\![[\varphi]]\!]$ se calcula directamente en $O(V^2)$ tomando el conjunto $[\![\varphi]\!]$.
  - $[\![\beta_1 \cup \beta_2]\!]$ y $[\![\beta_1 \cap \beta_2]\!]$ se pueden calcular ordenando primero los conjuntos $[\![\beta_i]\!]$ para $i \in \{1, 2\}$ en $O(|V^2| \log(|V|))$ y haciendo una sola pasada sobre ambos conjuntos a la vez en $O(|V^2|)$.
  - $[\![\beta_1 \,.\, \beta_2]\!]$ se calcula haciendo el *natural join* entre $[\![\beta_1]\!]$ y $[\![\beta_2]\!]$. Una forma de hacerlo en $O(|V^3|)$ es armando n buckets de tamaño a lo sumo n particionando a los pares de $[\![\beta_2]\!]$ de acuerdo a su primer elemento. Luego, para cada $(v, w) \in [\![\beta_1]\!]$ solo se revisa el bucket de los pares de la forma (w, z) para algún z.
  - Para calcular $[\![\beta^*]\!]$ se arma un grafo H que tiene como nodos a V y donde hay un eje entre v y w si y solamente si $(v, w) \in [\![\beta]\!]$. Luego decidir si hay un camino entre z_1 y z_2 en H equivale a decidir si $(z_1, z_2) \in [\![\beta^*]\!]$. Esto se puede calcular en $O(|V^3|)$ haciendo DFS desde cada nodo de H, teniendo en cuenta que $|H| = O(|V^2|)$.
  - $[\![\overline{\beta}]\!]$ se puede resolver ordenando $[\![\beta]\!]$ y haciendo una sola pasada sobre el conjunto ordenado.
  - $\beta^{n,m}$ se puede computar mediante exponenciación binaria. Primero se computa $[\![\beta^n]\!]$ recursivamente teniendo en cuenta que $[\![\beta^n]\!] = [\![\beta^{\frac{n}{2}} \,.\, \beta^{\frac{n}{2}}]\!]$ si n es par o bien $[\![\beta^n]\!] = [\![\beta^{\frac{n}{2}} \,.\, \beta^{\frac{n}{2}} \,.\, \beta]\!]$ si n es impar. Por ende, obtenemos $[\![\beta^n]\!]$ en $O(|V^3| \log(n))$, y esto es lineal en el tamaño de la subexpresión (que tiene tamaño $\log(n) + \log(m)$). Luego computamos $[\![\beta^{\leq m-n}]\!]$, entendido como el conjunto de los pares de nodos (v, w) tales que hay un camino $z_1 z_2 \ldots z_k$ tal que $z_1 = v$, $z_k = w$, $k \leq m - n$ y $(z_i, z_{i+1}) \in [\![\beta]\!]$. Para calcular $[\![\beta^{\leq m-n}]\!]$ armamos el mismo grafo H nombrado anteriormente y ejecutamos BFS desde todos los nodos. Luego

un par (v, w) está en $[\![\beta^{\leq m-n}]\!]$ si y solamente si el camino mínimo de v a w en H tiene distancia menor o igual a $m - n$. Esto puede hacerse en $O(|V^3|)$. Finalmente, $[\![\beta^{n,m}]\!] = [\![\beta^n \ . \ \beta^{\leq m-n}]\!]$.

- Para las expresiones de nodo tenemos:
  - Los casos $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $\langle\beta\rangle$ y $\varphi_1 \vee \varphi_2$ son triviales en base a su definición.
  - Solo quedan los casos $\langle\beta_1 \star \beta_2\rangle$ con $\star \in \{=, \neq\}$. Primero, armamos como antes los n buckets partiendo a cada $[\![\beta_i]\!]$ en función de la primera componente, lo cual hacemos en $O(n^2)$ operaciones. Luego, también en $O(n^2)$ calculamos para cada $v \in V$ y $i \in \{1, 2\}$ el conjunto $D_{v,i} = \{D(w) : (v, w) \in [\![\beta_i]\!]\}$. Si $D_{v,1} \cap D_{v,2} \neq \emptyset$ luego $v \in [\![\langle\beta_1 = \beta_2\rangle]\!]$. Por otro lado, para ver si $v \in [\![\langle\beta_1 \neq \beta_2\rangle]\!]$ alcanza con verificar que valga que los dos conjuntos $D_{v,i}$ sean no vacíos y que $|D_{v,i}| > 1$ para alguno de los dos o bien que $D_{v,1} \neq D_{v,2}$.

□

Gracias a este hecho sabemos que podemos, dado un conjunto R de restricciones de Reg-GXPath donde $|R| = \Sigma_{\alpha \in R}|\alpha|$ y un data-graph G, verificar si G es consistente con R en tiempo polinomial en función de $|G| + |R|$.

A continuación estudiamos el problema de computar subset repairs. Notamos que en la mayoría de los casos obtenemos cotas suporiores e inferiores finas tanto para la complejidad combinada como para la *data complexity*. Esta última fue definida en [36], y es la complejidad del problema cuando el conjunto de restricciones se considera fijo[1].

3.1. Encontrando subset repairs

Como el data-graph vacío (aquel cuyo conjunto de vértices es $\emptyset$) satisface cualquier conjunto de restricciones luego vale que todo data-graph G tiene un subset repair dado cualquier conjunto de restricciones R. Para estudiar con precisión la complejidad de encontrarlo definimos el siguiente problema de decisión:

PROBLEMA: $\exists$SUBSET-REPAIR$_{\mathcal{L}}$
INPUT: Un data-graph G y un conjunto R de expresiones del lenguaje $\mathcal{L}$.
OUTPUT: Decidir si G tiene un subset-repair $G' \neq \emptyset$ con respecto a R.

En este trabajo $\mathcal{L}$ será siempre algún subconjunto de Reg-GXPath. Si dejamos fijo el segundo parámetro (es decir, el conjunto de expresiones) con algún conjunto particular R entonces queda definido el problema $\exists$SUBSET-REPAIR$_{\mathcal{L}}(R)$, que recibe como input únicamente al data-graph G.

Notemos que este problema se reduce al de computar efectivamente un subset repair, por lo que las cotas inferiores que obtengamos podrán ser usadas para ese problema.

$\exists$SUBSET-REPAIR$_{\mathcal{L}}$ está en NP para cualquier $\mathcal{L}$ que permita evaluar la consistencia de un grafo cualquiera respecto a R en tiempo polinomial. Si un par (G, R) es una instancia

[1] Este tipo de complejidad fue definida originalmente para el problema de evaluar una cierta *query* o consulta sobre una base de datos relacional. En esos casos el tamaño de la consulta muchas veces es despreciable en función del tamaño de la base de datos, por lo que es razonable ignorarla al momento de diseñar un algoritmo

positiva del problema luego alcanza con pedir como certificado un data-graph H tal que $\emptyset \subset H \subset G$ y $H \models R$. Ambas condiciones pueden verificarse en tiempo polinomial partiendo de nuestra condición sobre $\mathcal{L}$.

A continuación probamos que el problema también es NP-*hard* si Reg-GXPath $\subseteq \mathcal{L}$:

Teorema 12. *Existe un conjunto de Reg-GXPath node expressions R tal que el problema* $\exists$SUBSET-REPAIR$_{\textit{Reg-GXPath}}(R)$ *es* NP-HARD.

Demostración. Vamos a reducir el problema 3-SAT a $\exists$SUBSET-REPAIR(Reg-GXPath) con un conjunto de restricciones R fijo (que será definido más adelante) incluido en $\mathcal{L}$. Dada una fórmula ϕ en *forma normal conjuntiva* con 3 variables por cláusula, variables $x_1, \ldots, x_n$ y cláusulas $c_1, \ldots, c_m$ queremos construir un data-graph $G = (V, L, D)$ tal que G tiene un subset repair no trivial con respecto a R si y solamente si ϕ es satisfacible.

Para empezar, definimos los nodos de nuestro grafo: habrá un nodo por cada variable y cláusula, y dos nodos representando los valores booleanos $\top$ y $\bot$. Definimos luego

$$V = V_{var} \cup V_{clause} \cup V_{value}$$

donde $V_{var} = \{\ x_i \mid 1 \leq i \leq n\}$, $V_{clause} = \{\ c_j \mid 1 \leq j \leq m\}$ y $V_{value} = \{\bot, \top\}$. El conjunto de data values será $\Sigma_n = \{var, clause, \bot, \top\}$, y definimos $D(x_i) = var$ para $i \in \mathbb{N}$ tal que $1 \leq i \leq n$, $D(c_j) = clause$ para $j \in \mathbb{N}$ tal que $1 \leq j \leq m$ y $D(\star) = \star$ para $\star \in \{\bot, \top\}$.

Para el conjunto de ejes usaremos las siguientes edge labels:

- VALUE: este eje será empleado para escoger la asignación de cada variable. En un subset repair de G cada nodo de tipo variable tendrá un unico eje VALUE saliente dirigido hacia uno de los nodos "booleanos".
- APPEARS_IN: si una variable x_i aparece en una cláusula c_j (sin negación) luego habrá un eje de x_i a c_j con esta label.
- APPEARS_NEGATED_IN: es análogo al eje APPEARS_IN, pero para el caso en que x_i aparece negada.
- EXISTS: este label será empleado para evitar que haya un subset repair en que se haya borrado un nodo. Su utilidad será evidente cuando veamos el conjunto de restricciones R.

Notemos que $\Sigma_e = \{$VALUE, APPEARS_IN, APPEARS_NEGATED_IN, EXISTS$\}$.

Ahora describamos el conjunto de ejes de G. Definimos:

$$L(x_i, c_j) = \begin{cases} \{\text{APPEARS\_IN, APPARS\_NEGATED\_IN}\} & \text{si } x_i \text{ aparece negada y sin negación en } c_j \\ \{\text{APPEARS\_IN}\} & \text{si } x_i \text{ aparece sin negación en } c_j \\ \{\text{APPEARS\_NEGATED\_IN}\} & \text{si } x_i \text{ aparece negada en } c_j \\ \emptyset & \text{en cualquier otro caso} \end{cases}$$

$$L(x_i, \bot) = L(x_i, \top) = \{\text{VALUE}\}$$

También agregamos, dada una enumeración arbitraria de los nodos $p_1, \ldots, p_k$, un eje con label EXISTS entre p_k y p_{k+1} (ciclando cuando $n = k$). Esto es, EXISTS $\in L(p_k, p_{k+1})$. Remarcamos que los ejes de label EXISTS solo aparecen entre estos nodos.

El grafo que definimos tiene toda la información para reconstruir ϕ, la cual está contenida en los ejes con label APPEARS_IN y APPEARS_NEGATED_IN. Intuitivamente, en este data-graph toda variable está asignada a los dos valores booleanos. El conjunto de restricciones será definido de tal forma que en un subset repair no trivial cada nodo variable tendrá un único eje dirigido a algún nodo booleano, y aparte cada cláusula tendrá alguna variable sin negar en ella asignada a $\top$ o bien una variable negada asignada a $\bot$.

Estas dos condiciones pueden ser expresada con las siguientes fórmulas:

$$\psi_1 = [var^{=}] \Rightarrow (\langle \downarrow_{\text{VALUE}} \rangle \wedge \neg \langle \downarrow_{\text{VALUE}} \neq \downarrow_{\text{VALUE}} \rangle) \tag{3.1}$$

$$\psi_2 = [clause^{=}] \Rightarrow (\langle \downarrow^{-}_{\text{APPEARS\_IN}} \downarrow_{\text{VALUE}} [\top^{=}] \rangle \ \vee \langle \downarrow^{-}_{\text{APPEARS\_NEGATED\_IN}} \downarrow_{\text{VALUE}} [\bot^{=}] \rangle) \tag{3.2}$$

La fórmula 3.1 fuerza a todo nodo variable a tener una asignación única en los repairs, mientras que la fórmula 3.2 asegura que cada cláusula sea satisfecha.

Finalmente, hay un detalle técnico: dadas estas reglas, es posible que un subgrafo de G donde una cláusula sea completamente borrada sea un repair válido. Para prevenir esto usamos los ejes con label EXISTS. Definimos la restricción

$$\psi_3 = \langle \downarrow_{\text{EXISTS}} \rangle \tag{3.3}$$

Esta última restricción implica que en todo subset repair no trivial todos los nodos de G deberán estar presentes. Notemos que caso contrario podemos tomar a i como el menor índice tal que p_i no pertenece al repair, y por lo tanto p_{i-1} no estará en $[\![\psi_3]\!]$ (en el caso límite en que $i = 1$ se puede hacer en razonamiento similar definiendo apropiadamente el índice en el que la regla falla).

Ahora probemos que $\phi(\overline{x})$ es satisfacible si y solamente si G tiene un repair no trivial con respecto a $R = \{\psi_1, \psi_2, \psi_3\}$.

$\Longrightarrow$) Si $\phi(\overline{x})$ es satisfacible entonces existe una asignación f de sus variables tal que ϕ evalúa a verdadero. Tomemos el data-graph G, y eliminemos de él cada eje $(x_i, \text{VALUE}, \top)$ (o $(x_i, \text{VALUE}, \bot)$) donde $f(x_i) \neq \top$ (respectivamente, $f(x_i) \neq \bot$). Es fácil ver que este grafo G' no es vacío, y que es consistente con respecto a R. Por lo tanto G tiene un subset repair no trivial.

$\Longleftarrow$) Sea G' un subset repair no trivial de G con respecto a R. Dado que 3.3 se satisface en G' luego podemos estar seguros que todo nodo de G está en G'. Aparte, gracias a 3.1 cada variable de tipo nodo tiene un único eje VALUE dirigido a un nodo booleano, y por ende podemos definir una asignación f como $f(x_i) = \top$ si y solamente si $(x_i, \text{VALUE}, \top)$ es un eje en G'. Finalmente, gracias a que G' satisface 3.2 sabemos que toda cláusula de ϕ tiene que evaluar a verdadero con esta asignación, y por lo tanto ϕ es satisfacible. □

Como ya explicamos antes, podemos usar esta cota al problema de decisión para obtener resultandos sobre el problema de computar un subset repair:

Corolario 13. *El problema de computar un subset repair de un grafo G dado un conjunto de restricciones R de expresiones Reg-GXPath es intratable.*

Dado que el problema de computar un subset repair considerando todo el poder de Reg-GXPath para el conjunto de restricciones es un problema NP-HARD procedemos a considerar subconjuntos del lenguaje que nos permitan desarrollar algoritmos polinomiales (por supuesto, a costo de perder capacidad de expresión en las restricciones). Dos fragmentos simples e importantes de Reg-GXPath son Reg-GXPath^{pos} y Reg-GXPath$^{path-pos}$: el primero se obtiene quitando las reglas de producción $\neg\varphi$ y $\overline{\alpha}$, mientras que el segundo solo quitando $\neg\varphi$[26].

Consideremos el fragmento Reg-GXPath^{pos}. Una propiedad útil que vale para estas expresiones es la siguiente:

Lema 14 (Monotonía de Reg-GXPath^{pos})**.** *Sea G un data-graph, α una expresión de camino de Reg-GXPath^{pos}, φ una expresión de nodo de Reg-GXPath^{pos} y G' un data-graph tal que $G \subseteq G'$. Luego vale que:*

- $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$.
- $[\![\varphi]\!]_G \subseteq [\![\varphi]\!]_{G'}$.

Demostración. Intuitivamente, si $(v, w) \in [\![\alpha]\!]_G$ y $\alpha \in$ Reg-GXPath^{pos} entonces existe un cierto "camino" (con comillas porque puede ser un camino con varios subcaminos) que va de v a w y es testigo de que (v, w) satisface α[2]. Como $G \subseteq G'$ ese mismo testigo va a estar presente en G'.

Ver el apéndice para los detalles técnicos. □

Gracias a este hecho podemos definir un algoritmo eficiente para encontrar un subset repair, basándonos en la siguiente corolario:

Corolario 15. *Si un nodo v de G no satisface una node expression φ entonces no hay ningún subconjunto de G en el cual v satisfaga φ.*

En general, podemos probar que:

Teorema 16. *Sea G un data-graph, R un conjunto de restricciones de Reg-GXPath^{pos} y v un nodo de G que incumple una node expression de R. Luego vale que:*

$$\subseteq\text{-}Rep(G, R) = \subseteq\text{-}Rep(G_{V_G \setminus \{v\}}, R)$$

donde G_V es el subgrafo de G inducido por los nodos $V \subseteq V_G$.

Demostración. Probemos primero la inclusión $\subseteq$: Sea $H \in \subseteq\text{-}Rep(G, R)$. En base al lema 14 y el corolario 15 sabemos que v no es un nodo de H. Esto implica que $H \subseteq G_{V_G \setminus \{v\}}$, y como H también es un subconjunto maximal consistente de G con respecto a R entonces podemos concluir que es un subconjunto maximal consistente de $G_{V_G \setminus \{v\}}$, y por ende $H \in \subseteq\text{-}Rep(G_{V_G \setminus \{v\}}, R)$.

Para la otra inclusión, sea $H \in \subseteq\text{-}Rep(G_{V_G \setminus \{v\}}, R)$. Como H satisface R y $H \subseteq G$ solo hace falta probar que es maximal. Por el absurdo, supongamos que existe un data-graph H' tal que $H \subset H' \subseteq G$ que satisface R. Sabemos que v no es un nodo de H' por el lema 14, y por ende podemos concluir que $H' \subseteq G_{V_G \setminus \{v\}}$. Esto implica que H no es un subset repair de $G_{V_G \setminus \{v\}}$, lo cual es absurdo. Concluímos entonces que H es un subset repair de G con respecto a R.

□

[2] Esto no pasa necesariamente en Reg-GXPath, dado que si la fórmula es de la pinta $\overline{\beta}$ entonces no hay ningún subgrafo que pueda ser testigo de que la fórmula se satisface

Es más, si R solo contiene node expressions de Reg-GXPath^{pos} entonces hay un único subset repair, dado que las expresiones de nodo de Reg-GXPath^{pos} están cerradas respecto a la unión: dados dos subconjuntos H y H' de G que satisfacen R podemos considerar el data-graph

$$H'' = (V_H \cup V_{H'}, L_H \cup L_{H'}, D_G|_{V_H \cup V_{H'}})$$

donde $D_G|_V$ es la restricción de D al conjunto V y $L_H \cup L'_H$ es la función definida como $L(v,w) = L_H(v,w) \cup L_{H'}(v,w)$. Este data-graph satisface R por el lema 14, y contiene tanto a H como a H'. Esto implica que no pueden existir dos subset repairs distintos.

En base a estos hechos proponemos el siguiente algoritmo que encuentra un subset repair dado un data-graph G y un conjunto de node expressions de Reg-GXPath^{pos} R:

Algorithm 1 *SubsetRepair*(G, R)

Require: G es un data-graph y R un conjunto de Reg-GXPath^{pos} node expressions.

```
while G ⊭ R do
    V⊥ ← { v | v ∈ V_G, ∃ φ ∈ R, v ∉ [[φ]]_G }
    G ← G_{V_G \ V⊥}
end while
return G
```

Este algoritmo es correcto porque el teorema 16 implica que $Rep(G,R) = Rep(G_{V_G \setminus V_\perp}, R)$. También termina, dado que en cada iteración se quitan nodos del grafo G y el conjunto $(\emptyset, R)$ es consistente. El conjunto $V_\perp$ se computa en tiempo polinomial, y dado que hay a lo sumo $|V_G|$ iteraciones concluímos que:

Teorema 17. *Dado un data-graph G y un conjunto de node expressions de Reg-GXPath^{pos} es posible computar el único subset repair de G con respecto a R en tiempo polinomial.*

¿Y qué pasa cuando hay expresiones de camino? Podemos hacer una demostración de *NP-hardness* más fina que la del teorema 12 que muestra que este caso también es intratable:

Teorema 18. *Existe un conjunto R de path expressions de Reg-GXPath^{pos} tal que el problema* $\exists$SUBSET-REPAIR$_{\text{Reg-GXPath}^{pos}}(R)$ *es* NP-COMPLETO.

Demostración. Como antes, vamos a reducir el problema 3-SAT a $\exists$SUBSET-REPAIR(Reg-GXPath^{pos}). Dada una fórmula 3CNF ϕ con n variables x_i y m cláusulas c_j queremos construir un data-graph G y un conjunto de restricciones R compuesto por path expressions de Reg-GXPath^{pos} tal que G tiene un subset repair no trivial con respecto a R si y solamente si ϕ es satisfacible. R no va a depender de ϕ, y por ende lo podemos considerar fijo.

Primero definimos el conjunto de nodos. Vamos a tener dos nodos booleanos por cada variable de ϕ, representando las posibles asignaciones. También tendremos un nodo por cada cláusula c_j.

$$V = \{\ \perp_i \mid 1 \le i \le n\ \} \cup \{\ \top_i \mid 1 \le i \le n\ \} \cup \{\ c_j \mid 1 \le j \le m\ \}$$

En un subset repair válido solo queremos que uno de los nodos $\top_i$ o $\perp_i$ permanezca, y de esa forma obtendremos una asignación sobre las variables de ϕ.

Como en la demostración anterior, codificaremos la estructura de ϕ en los ejes del grafo. Por cada nodo cláusula c_j habrá un eje hacia $\bot_i$ (respectivamente $\top_i$) si la variable x_i aparece negada en c_j (respectivamente sin negación). Para esto emplearemos el conjunto de edge labels $\Sigma_e = \{\text{NEEDS}, \text{EXISTS}, \text{UNIQUE}, \text{VALID}\}$ y definiremos:

$$L(c_j, \top_i) = \{\text{NEEDS}\} \text{ si } x_i \text{ aparece sin negación en } c_j$$
$$L(c_j, \bot_i) = \{\text{NEEDS}\} \text{ si } x_i \text{ aparece negada en } c_j$$

En un subset repair válido queremos que cada cláusula c_j mantenga uno de estos ejes, ya que representará que la asignación satisface a la cláusula.

Hay un problema técnico similar al que vimos antes: podría pasar que en un subset repair no estén todos los nodos. Para evitar esto agregamos algunos ejes y una expresión que fuerce a cada nodo cláusula a estar presente (a menos que el subset repair sea el trivial). Definimos una estructura cíclica con ejes EXISTS de la siguiente forma:

$$L(c_j, c_{j+1}) = \{\text{EXISTS}\}$$
Agregamos ejes EXISTS a $L(c_m, \top_1)$, $L(c_m, \bot_1)$, $L(\top_n, c_1)$ y $L(\bot_n, c_1)$
$$L((\star^1)_i, (\star^2)_{i+1}) = \{\text{EXISTS}\} \text{ for } \star^1, \star^2 \in \{\top, \bot\}$$

Usando estos ejes y la restricción:

$$\beta_1 = \downarrow^+_{\text{EXISTS}} \tag{3.4}$$

nos aseguramos que todo nodo cláusula permanezca y algún nodo $\star_i$ para $\star \in \{\bot, \top\}$. Intuitivamente hicimos un ciclo que recorre todos los c_i, y después recorre los $\bot_i$ y $\top_i$ pero sin pasar por ambos $\bot_k$ y $\top_k$ para ningún k.

Ahora, para también pedir que uno de estos dos nodos $\bot_i$ y $\top_i$ se vaya agregamos los ejes UNIQUE a todo par de nodos salvo los de la forma $(\top_i, \bot_i)$ para algún i.

Notemos que todos los pares de nodos (salvo por los de la forma $(\top_i, \bot_i)$) pertenecen a la relación UNIQUE. Por ende, agregando la restricción:

$$\beta_2 = \downarrow_{\text{UNIQUE}} \tag{3.5}$$

Con esto ya evitamos la situación en que un repair contiene a ambos nodos $\top_i$ y $\bot_i$ para algún i.

Hasta el momento sabemos que en un repair válido todo nodo cláusula va a estar, aparte de uno de los nodos $\bot_i$ y $\top_i$ por cada i. Para asegurarnos también de que la asignación definida por un subset repair no trivial satisfaga ϕ agregamos ejes con label VALID y definimos una nueva regla. A todo par de nodos (v, v') que no sea de la forma $v = c_j$ y $v' = \star_i$ para algún j, i y $\star \in \{\top, \bot\}$ agregamos el edge label VALID, y consideramos la restricción:

$$\beta_3 = \downarrow_{\text{VALID}} \cup \downarrow_{\text{NEEDS}} \downarrow_{\text{VALID}} \tag{3.6}$$

Cada par de nodos pertenece a la relación VALID excepto por aquellos de la forma $(c_j, \bot_i)$ o $(c_j, \top_i)$. Para que estos satisfagan la expresión 3.6 debe pasar que uno de los ejes NEEDS haya permanecido por cada nodo cláusula. Estos ejes apuntaban hacia los nodos

variable que satisfacían la cláusula, y por ende nos aseguramos que la asignación que represente el subset repair no trivial haga verdadera cada cláusula[3].

Recordamos que, intuitivamente, si existe un repair no trivial G con respecto a $R = \{\beta_1, \beta_2, \beta_3\}$ entonces este consistirá de una selección de nodos booleanos tal que toda cláusula mantenga un eje NEEDS hacia un nodo variable, y esto representará implícitamente una asignación que satisface ϕ.

Ahora probemos que $\phi(\overline{x})$ es satisfacible si y solamente si G tiene un repair no trivial con respecto a R.

$\Longrightarrow$) Si $\phi(\overline{x})$ es satisfacible entonces existe una asignación f de sus variables tal que ϕ evalúa a verdadero. Consideremos el data-graph G, y saquemos de él todos los nodos $\star_i$ con $\star \in \{\bot, \top\}$ para los cuales $f(x_i) \neq \star$.

Como solo borramos un nodo booleano por cada i la path expression 3.4 se satisface. La expresión 3.5 también se satisface, ya que no existe ningún i para el cual tanto $\top_i$ como $\bot_i$ pertenezcan al grafo. Finalmente, como f es una asignación que satisface las cláusulas de ϕ sabemos que cada nodo c_j mantuvo uno de sus ejes NEEDS, y por ende 3.6 se satisface.

Este subgrafo luego satisface R, lo que implica que existe un subset repair no trivial de G con respecto a R.

$\Longleftarrow$) Sea G' un repair no trivial de G con respecto a R. Como tanto (3.4) como (3.5) se satisfacen sabemos que para todo i solo uno de los nodos $\bot_i$ y $\top_i$ está en G'. Luego definimos una asignación f sobre las variables x_i como $f(x_i) = \top \iff \top_i \in V_{G'}$. Como (3.6) se satisface en G' sabemos que por lo menos un literal de cada cláusula evalúa a verdadero mediante f. Por ende, ϕ es satisfacible.

□

Una vez más observamos que este es un resultado más fuerte que el teorema 12, dado que el conjunto de expresiones $\mathcal{L}$ con el que parametrizamos al problema ∃SUPERSET-REPAIR en este caso está incluido (estrictamente) en el de aquella otra demostración. También notemos que en ambos casos el conjunto R estaba fijo, y por ende estas cotas inferiores aplican sobre la data complexity del problema.

Concluimos la sección observando que a pesar de que el problema en general resultó ser NP-COMPLETO para un conjunto simple de path expressions muchos ejemplos de los que mostrarmos emplean (o pueden ser simulados mediante) node expressions positivas. Esto se debe a que en la mayoría de los casos se usa la negación para armar "implicaciones", pero muchas se pueden reescribir evitando la negación si se asume cierta estructura en la base de datos.

En particular, los ejemplos usan las implicaciones para hacer "restricciones por tipo de dato" sobre el data-graph. Sería razonable asumir que en esos casos una restricción de la forma $\langle\downarrow_{\text{TYPE}}\rangle$ va a estar presente en el data-graph, aparte de que la cantidad de tipos de datos será finita; y podríamos explotar esta condición para escribir, en vez de $type(\alpha) \implies restriction$ algo como $\bigvee_{\beta \neq \alpha} type(\beta) \vee restriction$. Esta nueva expresión mantiene la misma semántica, y como el conjunto Σ_n está fijo tiene el mismo orden de tamaño.

También remarcamos que el algoritmo que opera sobre node expressions de Reg-GXPath^{pos} seguirá funcionando en la medida en que las expresiones satisfagan la propiedad de monotonía que definimos. Esto significa que es posible agregarle más herramientas al lenguaje

[3] En la demostración del teorema 12 esta condición era mucho más simple de escribir dado que teníamos la negación. Ahora hay que agregar muchos ejes para poder "seleccionar" sobre qué nodos queremos que se cumple la condición.

y este algoritmo seguirá siendo correcto (y seguirá haciendo una cantidad polinomial de operaciones si la evaluación de la nueva familia de expresiones se puede hacer en tiempo polinomial en función del tamaño de la expresión y el del data-graph).

3.2. Encontrando superset repairs

Empezamos la sección notando que

Comentario 19. Existe un data-graph G y un conjunto de expresiones de Reg-GXPath R tal que no existe un supergrafo de G que satisfaga R.

Por ejemplo, tomemos $G = (\{v\}, L, D)$ donde $D(v) = c$, $L(v, v) = \emptyset$ y R es un conjunto de una única restricción $\phi = [c^{\neq}]$. Todo superset G' de G va a contener el nodo v con dato c, por lo que $v \notin [\![\phi]\!]_{G'}$, y por lo tanto G' no es consistente con respecto a R.

Para estudiar la complejidad del problema de encontrar un superset repair definimos el siguiente problema de decisión:

PROBLEMA: $\exists$SUPERSET-REPAIR$_{\mathcal{L}}$ INPUT: Un data-graph G y un conjunto R de expresiones del lenguaje $\mathcal{L}$. OUTPUT: Tiene G un superset repair con respecto a R?

Cuando $\mathcal{L}$ contiene toda posible expresión de Reg-GXPath podemos probar la siguiente cota inferior:

Teorema 20. *El problema* $\exists$SUPERSET-REPAIR$_{\textit{Reg-GXPath}}$ *es* EXPTIME*-hard.*

Demostración. Para probar este teorema nos apoyamos de un resultado obtenido en [21] relacionado al problema de satisfacibilidad de expresiones del lenguaje **Downward XPath**. Este problema consiste en decidir, dada una expresión de nodo de **Downward XPath**, si existe un *data tree* tal que la expresión es satisfecha en la raíz del árbol. Un data tree es ese contexto se define como una tupla $< \mathcal{T}, \Sigma, \delta >$ donde $\mathcal{T}$ es un subconjunto no vacío y *prefix-closed*[4] de $\mathbb{N}^*$ representando el conjunto de nodos del árbol tal que si $x(i+1) \in \mathcal{T}$ entonces $xi \in \mathcal{T}$; $\sigma : \mathcal{T} \to \Sigma$ es una función que asigna *labels* a los nodos de $\mathcal{T}$ y $\delta : \mathcal{T} \to \triangle$ es una función que asigna *data values* a los nodos de $\mathcal{T}$. Σ es un conjunto finito, mientras que $\triangle$ es un conjunto contable cualquiera. En [21] se demuestra que el problema de satisfacibilidad para fórmulas de XPath$(\downarrow^*, =)$ es EXPTIME-COMPLETE, donde la gramática que genera las expresiones de XPath$(\downarrow^*, =)$ es:

$$\alpha, \beta = o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha \,.\, \beta \mid \alpha \cup \beta \qquad o \in \{\downarrow^*, \epsilon\}$$

$$\varphi, \psi = a \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle\alpha\rangle \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle \qquad a \in \Sigma_s$$

Observemos que estos data trees tienen tanto labels como data values. Esto es importante considerando la semántica de las fórmulas de XPath$(\downarrow^*, =)$, dado que la expresión de nodo a es verdadera evaluándola en cualquier nodo con label a (esto es, todo v tal que $\sigma(v) = a$) mientras que la expresión de comparación $\langle\alpha = \beta\rangle$ es verdadera para aquellos nodos v desde los que se pueden alcanzar dos nodos w y z respectivamente a través de

[4] Un conjunto de cadenas S es *prefix-closed* si para toda cadena $s \in S$ vale que todo prefijo de s está en S

caminos en α y β que tengan el mismo data value (es decir, $\delta(w) = \delta(z)$). La semántica del operador $\neq$ es igual, salvo porque pide que $\delta(w) \neq \delta(z)$. Notemos también que la semántica del operador $[\cdot]$ es la misma que para Reg-GXPath.

Reduciremos el problema de satisfacibilidad de expresiones XPath$(\downarrow^*, =)$ al problema $\exists$SUPERSET-REPAIR$_{\text{Reg-GXPath}}$. Para hacer esto, dada una expresión de nodo φ en XPath$(\downarrow^*, =)$ construiremos un data-graph G, un conjunto R de expresiones de Reg-GXPath y una fórmula φ' de Reg-GXPath tal que todo supergrafo G' de G que sea consistente con R va a representar un data tree, y si ese grafo también satisface φ' entonces podremos construir a partir de G' un data tree T que satisfaga la fórmula original φ.

Primero, definimos los conjuntos fijos Σ_e y Σ_n. Estos estarán relacionados con los conjuntos fijos de labels (o símbolos) y data values del problema de **downward XPath**: Σ_s y $\triangle$ (recordemos que Σ_s es finito, mientras que $\triangle$ no necesariamente).

En el modelo de los data trees solo hay un tipo de eje, al cual llamaremos DOWN. Para los data-graphs también usaremos dos tipos más: ROOT y VALUE. El primero será útil para distinguir la raíz en el data-graph G, mientras que el segundo será usado para simular los data values de los data trees. Por ende, considermos $\Sigma_e = \{\text{DOWN}, \text{VALUE}, \text{ROOT}\}$.

Por otro lado, definimos $\Sigma_n = \Sigma_s \cup \triangle \cup \{r\}$, donde r es un símbolo nuevo (no perteneciente a Σ_s ni a $\triangle$) que le asignaremos a la raíz del árbol representado por cualquier superconjunto de G. Tanto los símbolos como los data values de los data trees serán "simulados" mediante los data values de los data-graphs. Para lograr esto tendremos que dar una cierta estructura a la familia de repairs del grafo G, y esto lo haremos mediante las restricciones en R.

Nuestro data-graph inicial G se define de la siguiente forma: $G = (V, L, D)$ donde $V = \{\ v_c \mid c \in \Sigma_s \cup \{r\}\}$, $L(v_r, v_c) = \{\text{ROOT}\}$ para $c \in \Sigma_s$ y $L(v, w) = \emptyset$ para cualquier otro par de nodos, y $D(v_c) = c$ para $c \in \Sigma_s \cup \{r\}$. Notemos que este data-graph no tiene ciclos, y que puede considerarse una arborescencia[5] enraizada en v_r.

La idea es que en un superset repair válido habrá algún hijo z de v_r conectado a través de un eje ROOT tal que el subárbol enraizado en z representará un data tree que satisface la fórmula de entrada φ para el problema de satisfacibilidad sobre downward XPath. Por ende, tendremos que condicionar los repairs mediante R para que satisfagan estas condiciones.

Para asegurar que todo repair siga siendo una arborescencia enraizada en v_r y que no haya agregado ningún eje ROOT podemos usar las siguientes expresiones:

- $\alpha_{connected} = (\_^*\_^{-*})^*$: el grafo subyacente está conectado.
- $\alpha_{noCycles} = \epsilon \Rrightarrow \overline{\_^+}$: el grafo no tiene ciclos dirigidos.
- $\alpha_{0or1parents} = \_ \,.\, \_^- \Rrightarrow \epsilon$: Todo nodo tiene uno o ningún ancestro.
- $\psi_{root} = [r^=] \Rightarrow \neg\langle\_^-\rangle$: El nodo v_r no podrá tener ejes entrantes (y por ende será una raíz).
- $\psi_{rootEdges} = \langle\downarrow_{\text{ROOT}}\rangle \Rightarrow [r^=]$: Todo eje ROOT debe partir desde v_r.

[5] Una arborescencia es un digrafo $G = (V, E)$ tal que $|E| = |V| - 1$, su grafo subyacente es un árbol y hay un nodo r desde el cuál se puede alcanzar cualquier otro nodo del digrafo, respetando el sentido de los ejes.

En algunas restricciones usamos la expresión $\_^-$ que no está incluida en la gramática de Reg-GXPath. No obstante, puede ser simulada enumerando toda label $e \in \Sigma_e$ como e^- dado que Σ_e es finito.

Es fácil convencerse que todo data-graph que satisfaga estas expresiones tendrá que ser una arborescencia enraizada en v_r. Para empezar, el data-graph es conexo (ignorando el sentido de los ejes) debido a $\alpha_{connected}$. Aparte no puede tener ciclos dirigidos debido a $\alpha_{noCycles}$, y tampoco ciclos no dirigidos debido a $\alpha_{0or1parents}$: si hubiera un ciclo no dirigido entonces habría un nodo en el ciclo con dos ejes entrantes a él, lo cual impide esta regla. Esto ya obliga al data-graph a ser una arborescencia, y por ende a tener una raíz. Es entonces cuando la regla ψ_{root} obliga a que el nodo v_r no tenga ancestros, y por ende a que sea la raíz del data-graph. La regla $\psi_{rootEdges}$ es útil para luego enunciar la formula φ' de forma más concreta.

Ahora también tenemos que condicionar a los data-graphs de tal forma que todo nodo tenga un data value y un símbolo a la vez. Para esto, cada nodo tendrá como dato un valor en Σ_s, pero a su vez del nodo saldrá un eje de tipo VALUE dirigido hacia un nodo que poseerá un valor en $\triangle$ (a estos nodos los llamaremos nodos *valor*). Para que esto siempre ocurra alcanza con pedir que: (1) Los nodos del "árbol principal" solo contengan valores de Σ_s, (2) Ningún nodo valor pueda tener hijos, (3) Todo nodo valor debe tener un data value de $\triangle$ y (4) todo nodo del árbol tiene que tener un único eje hacia un nodo valor. Podemos forzar estas condiciones con las siguientes reglas:

- $\psi_1 = \downarrow^{-*}_{\text{DOWN}} \downarrow^{-}_{\text{ROOT}} [r^=] \Rrightarrow \bigvee_{c \in \Sigma_s} [c^=]$: Los nodos del "árbol principal" son justamente aquellos alcanzables desde v_r mediante un eje ROOT y otros DOWN. Esta regla fuerza a estos ejes a tener como dato un valor en Σ_s.
- $\psi_2 = \bigwedge_{c \in \Sigma_s \cup \{r\}} [c^{\neq}] \Rightarrow \neg\langle\downarrow_{\text{DOWN}}\rangle \wedge \neg\langle\downarrow_{\text{VALUE}}\rangle$: Como los nodos valor son los que tienen valores en $\triangle = \Sigma_n \setminus (\Sigma_s \cup \{r\})$ pedimos que estos no puedan tener ejes salientes (en particular no pueden tener ejes ROOT salientes debido a la regla $\psi_{rootEdges}$).
- $\psi_3 = \langle\downarrow^{-}_{\text{VALUE}}\rangle \Rightarrow \bigwedge_{c \in \Sigma_s \cup \{r\}} [c^{\neq}]$: Si un nodo es nodo valor (i.e. tiene un eje de label VALUE entrante) entonces tiene un dato en $\triangle$.
- $\psi_4^1 = \bigvee_{c \in \Sigma_s} [c^=] \Rightarrow \langle\downarrow_{\text{VALUE}}\rangle$: Si un nodo pertenece al árbol principal entonces tiene un dato perteneciente a Σ_s.
- $\psi_4^2 = \downarrow^{-}_{\text{VALUE}} \downarrow_{\text{VALUE}} \Rrightarrow \epsilon$: Ningún nodo tiene dos ejes VALUE salientes.

Veamos por qué estas reglas alcanzan, asumiendo que nuestro data-graph ya es una arborescencia. El árbol principal definido como aquellos nodos alcanzables mediante ejes DOWN (luego de salir de la raíz mediante un eje ROOT) tienen siempre un valor en Σ_s debido a ψ_1. Todos ellos tienen un eje saliente de label VALUE debido a ψ_4^1, y este es único debido a ψ_4^2. Estos nodos tienen un dato en $\triangle$ gracias a ψ_3, y no tienen ejes salientes a causa de ψ_2. Por lo tanto, el data-graph tiene la estructura que queremos.

Ahora tenemos que adaptar la fórmula de entrada de XPath φ a nuestro contexto. Definimos una función $f : XPath(\downarrow^*, =) \to$ Reg-GXPath tal que:

- $f(\epsilon) = \epsilon$

- $f(\downarrow^*) = \downarrow^*_{\text{DOWN}}$
- $f(a) = [a^=]$ for $a \in \Sigma_s$
- $f(\alpha[\varphi]) = f(\alpha)[f(\varphi)]$
- $f([\varphi]\alpha) = [f(\varphi)]f(\alpha)$
- $f(\alpha.\beta) = f(\alpha).f(\beta)$
- $f(\alpha \cup \beta) = f(\alpha) \cup f(\beta)$
- $f(\neg\varphi) = \neg f(\varphi)$
- $f(\varphi \wedge \psi) = f(\varphi) \wedge f(\psi)$
- $f(\langle\alpha\rangle) = \langle f(\alpha)\rangle$
- $f(\langle\alpha = \beta\rangle) = \langle f(\alpha) \downarrow_{\text{VALUE}} = f(\beta) \downarrow_{\text{VALUE}}\rangle$
- $f(\langle\alpha \neq \beta\rangle) = \langle f(\alpha) \downarrow_{\text{VALUE}} \neq f(\beta) \downarrow_{\text{VALUE}}\rangle$

f solo concatena un eje VALUE al final de las expresiones de comparación (para que se aceda al valor correcto) y reemplaza los ejes sin etiqueta $\downarrow^*$ por ejes $\downarrow^*_{\text{DOWN}}$.

El siguiente lema prueba que f es una función que preserva la semántica de las expresiones de XPath en nuestro contexto de data-graphs que representan árboles:

Lema 21. *Sea T un data tree etiquetado (en el sentido de [21]), T' su representación como data-graph, y η una fórmula **downward XPath**. Luego, η se satisface en la raíz de T si y solamente si $f(\eta)$ se satisface en la raíz de T'.*

Demostración. Ver el apéndice. □

Finalmente, definimos φ', la expresión de Reg-GXPath que fuerza a los data-graphs que representan superset repairs a contener la representación de algún árbol que satisfaga φ:

$$\psi_{sat} = [r^=] \Rightarrow \downarrow_{\text{ROOT}} f(\varphi)$$

En resumen, dada una expresión de downward XPath φ construímos un data-graph G y el conjunto de restricciones $R = R_{arbor} \cup R_{dataTree} \cup \{\psi_{sat}\}$ donde:

$$R_{arbor} = \{\alpha_{connected}, \alpha_{noCycles}, \alpha_{0or1parents}, \psi_{root}, \varphi_{rootEdges}\}$$
$$R_{dataTree} = \{\psi_1, \psi_2, \psi_3, \psi_4^1, \psi_4^2\}.$$

El tamaño de G depende de $|\Sigma_s|$, que es un valor finito y fijo. El tamaño de R depende linealmente de $|\Sigma_s|$ y de $|\varphi|$.

Solo nos falta probar que existe un data tree T con raíz r_T tal que $r_T \in [\![\varphi]\!]_T$ si y solamente si existe un superset repair de G con respecto a R.

$\Longrightarrow$) Sea T un data tree con raíz r_T con símbolo $a \in \Sigma_s$ tal que $r_T \in [\![\varphi]\!]_T$. Sea T' su conversión a data-graph con raíz $r_{T'}$. Entonces sabemos que $r_{T'} \in [\![f(\varphi)]\!]_{T'}$.

Ahora, unifiquemos el hijo de v_r en G con símbolo a con $r_{T'}$. De esta forma, estamos cambiando al nodo v_r en G por el árbol T' También agreguemos al resto de los hijos de v_r un eje saliente de tipo VALUE hacia un nuevo nodo (un nodo por cada uno de los hijos de v_r) con algún data value de $\triangle$. R es satisfecho por este data-graph, y por lo tanto existe un superset repair de G con respecto a R.

$\Longleftarrow$) Sea G' un superset repair de G con respecto a R. Dado que $G' \models R_{arbor}$ sabemos que G' es una arborescencia enraizada en v_r. Como $G' \models \psi_{sat}$ entonces tiene que haber un hijo v_c de v_r tal que $v_c \in [\![f(\varphi)]\!]_{G'}$. Como $f(\varphi)$ solo 'baja' (es decir, no puede subir por los ejes y por lo tanto no puede interactuar con v_r) sabemos que $v_c \in [\![f(\varphi)]\!]_T$, donde T es la arborescencia de G' enraizada en v_c. Dado que $G' \models R_{dataTree}$ podemos concluir que T es la representación de algún data tree T' que satisface φ en su raíz.

□

Fijando el parámetro R el problema sigue siendo intratable en general:

Teorema 22. *Existe un conjunto de expresiones $R \subseteq$ Reg-GXPath tal que* $\exists$SUPERSET-REPAIR$_{\textit{Reg-GXPath}}(R)$ *es* NP-HARD.

Demostración. Como hicimos en el teorema 12 vamos a reducir 3-SAT a $\exists$SUPERSET-REPAIR$_{\text{Reg-GXPath}}$. Dada una formula 3-CNF ϕ con n variables y m cláusulas vamos a definir un data-graph G y un conjunto R de expresiones de Reg-GXPath tal que ϕ es satisfacible si y solamente si G tiene un superset repair con respecto a R.

La construcción es similar a la empleada en el teorema 12. Tendremos nodos representando las cláusulas, los nodos y los valores booleanos:

$$V = V_{var} \cup V_{clause} \cup V_{node}$$

donde $V_{var} = \{\top, \bot\}$, $V_{clause} = \{\ c_j \mid 1 \leq j \leq m\}$, $V_{node} = \{\ x_i \mid 1 \leq i \leq n\}$. Cada 'tipo' de nodo tendrá un data value distinto: definimos $D(x_i) = c_{variables}$, $D(c_j) = c_{clauses}$, $D(\bot) = \bot$ y $D(\top) = \top$.

Las labels que vamos a usar para los ejes son:

- VALUE: En un repair válido todo nodo variable tendrá un único eje de tipo VALUE apuntando a un nodo de tipo booleano. Esto definirá implícitamente una asignación de las variables.
- APPEARS_IN$_i$ para $i \in \{1, 2, 3\}$: la información de la fórmula ϕ está codificada en estos ejes. Esto es, dada una cláusula c y las tres variables v_1, v_2 y v_3 presentes en c se agregan a G un eje APPEARS_IN$_i$ por cada $i \in \{1, 2, 3\}$ tal que v_i aparece sin negación en c.
- APPEARS_NEGATED_IN$_i$ para $i \in \{1, 2, 3\}$: igual que en el caso anterior, pero considerando si la variable aparece negada en c.

Dada la descripción de cada *label* es fácil intuir la definición de G:

$$E_1 = \{\ (x_i, \text{APPEARS\_IN}_k, c_j) \mid x_i \text{ es la } k\text{-ésima variable de } c_j \text{ y aparece sin negación, para } k \in \{1, 2, 3\}\ \}$$
$$E_2 = \{\ (x_i, \text{APPEARS\_NEGATED\_}in_k, c_j) \mid x_i \text{ es la } k\text{-ésima variable de } c_j \text{ y aparece negada, para } k \in \{1, 2, 3\}\ \}$$
$$E = E_1 \cup E_2$$

Ahora queremos construir un conjunto R de expresiones Reg-GXPath tal que un superset repair de G con respecto a R defina una asignación de las variables de ϕ que la

evalúe a verdadero. Para asegurar que de cada nodo variable salga un único eje VALUE podemos restringir con la siguiente expresión[6].

$$\psi_1 = [c^{=}_{variable}] \Rightarrow (\langle\downarrow_{\text{VALUE}}\rangle \wedge \langle\downarrow_{\text{VALUE}} ([\bot^{=}] \vee [\top^{=}]\rangle \wedge \neg\langle\downarrow_{\text{VALUE}} \neq \downarrow_{\text{VALUE}}\rangle) \tag{3.7}$$

Luego, usamos la siguiente expresión para asegurarnos de que cada cláusula quede satisfecha por la asignación de las variables definida por los ejes VALUE:

$$\psi_2 = [c^{=}_{clause}] \Rightarrow \bigvee_{k\in\{1,2,3\}} \downarrow^{-}_{\text{APPEARS\_IN}_k}\downarrow_{\text{VALUE}} [\top^{=}] \vee \downarrow^{-}_{\text{APPEARS\_NEGATED\_IN}_k}\downarrow_{\text{VALUE}} [\bot^{=}] \tag{3.8}$$

Tenemos que evitar también dos situaciones.

Primero, en el repair se podrían agregar ejes APPEARS que no estaban en el grafo G. Por ejemplo, si c_j tiene un eje dirigido hacia él con label APPEARS_IN_k entonces no queremos que en ningún repair aparezca un eje $\text{APPEARS\_NEGATED\_IN}_k$ incidiendo en c_j (en general, solo queremos que se agreguen los ejes de label VALUE).

Para lograr esto definimos:

$$\psi_3 = [c^{=}_{clause}] \Rightarrow \bigwedge_{k\in 1,2,3} \neg\langle\downarrow_{\text{APPEARS\_IN}_k}\rangle \vee \neg\langle\downarrow_{\text{APPEARS\_NEGATED\_IN}_k}\rangle \tag{3.9}$$

Segundo, no queremos que un nodo cláusula c_j tenga dos ejes APPEARS_IN_k incidiendo a él. Para evitar esto definimos el siguiente conjunto de expresiones:

$$\alpha^k_1 = \downarrow_{\text{APPEARS\_IN}_k}\downarrow^{-}_{\text{APPEARS\_IN}_k} \Rrightarrow \epsilon \tag{3.10}$$

$$\alpha^k_2 = \downarrow_{\text{APPEARS\_NEGATED\_IN}_k}\downarrow^{-}_{\text{APPEARS\_NEGATED\_IN}_k} \Rrightarrow \epsilon$$

para $k \in \{1, 2, 3\}$.

Supongamos que una cláusula c_j tiene dos ejes incidentes de label APPEARS_IN_1 desde v y w. Entonces el par (v, w) no cumpliría la condición impuesta por α^1_1. De la misma forma, este conjunto de expresiones considera cada caso posible de ejes de tipo APPEARS_IN.

Finalmente, $R = \{\psi_1, \psi_2, \psi_3\} \cup \{\alpha^k_i \mid i \in \{1, 2\}, k \in \{1, 2, 3\}\}$. Claramente G y R pueden ser construidos en tiempo polinomial dado ϕ. Ahora probaremos que ϕ es satisfacible si y solamente si existe un superset repair de G con respecto a R.

$\Longrightarrow$) Supongamos que ϕ es satisfacible, y sea f una asignación de sus variables que la satisface. Dado G agregamos por cada variable x_i el eje $(x_i, \text{VALUE}, f(x_i))$. El grafo resultante satisface R, dado que la única regla no satisfecha originalmente era ψ_1, y arreglamos esta agregando los ejes de tipo VALUE. Este agregado no puede afectar la satisfacibilidad del resto de las reglas porque no dependen de los ejes de tipo VALUE dentro del grafo. Este nuevo data-graph es un superset de G y es consistente respecto a R, lo cual implica que existe un superset repair de G con respecto a R.

$\Longleftarrow$) Sea G' un superset repair de G con respecto a R. Cada nodo variable original de G está presente en G', y dado que vale ψ_1 sabemos que tienen un eje saliente de tipo

[6] En realidad con esto aseguramos no haya dos ejes VALUE apuntando a nodos con data values distintos. Podría pasar que en el superset repair se agregue un nodo con dato $\bot$ o $\top$ y que un nodo variable tenga dos ejes salientes VALUE hacia dos nodos con el mismo dato $\bot$ o $\top$.

VALUE, y que estos ejes VALUE están dirigidos a los nodos $\bot$ o $\top$. Luego definimos una valuación de las variables de ϕ f como $f(x_i) = \top$ si y solamente si el nodo x_i tiene un eje VALUE entrando al nodo con data value $\top$ en G'.

Ahora probemos que esta valuación satisface ϕ. Gracias a α_i^k y ψ_3 sabemos que ningún eje de tipo APPEARS fue agregado a los nodos cláusula originales. Luego, ya que ψ_2 es satisfecha en G', sabemos que cada cláusula c_j contiene una variable sin negación que evalúa a verdadero mediante f o bien una variable que está negada en c_j y evalúa a falso. Esto implica que la valuación f satisface ϕ, lo cual termina la demostración.

□

En la ausencia de data tests cualquier data-graph G tiene un superset repair si consideramos solo conjuntos de expresiones en el fragmento positivo de Reg-GXPath. Esto ocurre porque podemos agregar ejes con todos los labels $l \in \Sigma_e$ entre cada par de nodos $v, v' \in V_G$ y el grafo resultante va a satisfacer cualquier restricción. Intuitivamente, para satisfacer la expresión hay que encontrar un camino testigo, y como de cada nodo se puede tomar cualquier edge label entonces va a ser posible trazar cualquier camino.

Por lo tanto, la complejidad en el problema surge al momento de hacer *data tests* o *data equalities*. Antes de avanzar, definimos algunos conceptos relacionados a estas expresiones:

Definición 23. Sea η una expresión de Reg-GXPath. Definimos el conjunto de data values presente en η como el conjunto de todos aquellos $c \in \Sigma_n$ tales que la subexpresión $[c^=]$ o $[c^{\neq}]$ es empleada en η. Lo definimos como $\boldsymbol{\Sigma_n^{\eta}}$.

De la misma forma definimos el conjunto de data values empleado en un conjunto de expresiones de Reg-GXPath R como $\boldsymbol{\Sigma_n^R} = \bigcup_{\eta \in R} \Sigma_n^{\eta}$.

También denotamos al conjunto de data values empleados en un data-graph como $\boldsymbol{\Sigma_n^G}$.

Aunque Σ_n puede ser infinito, cuando consideramos solamente expresiones de Reg-GXPath^{pos} podemos ver que vale lo siguiente:

Lema 24. *Sea G un data-graph y R un conjunto de expresiones Reg-GXPath^{pos}. Si existe un superset repair G' de G con respecto a las restricciones R, entonces hay otro superset repair H que solo usa data values en $\Sigma_n^R \cup \Sigma_n^G$ y a lo sumo dos extra data values no mencionados en R.*

Demostración. La idea principal de este lema es que no se necesitan muchos data values más que aquellos en Σ_n^R, lo cual es intuitivo: si la subexpresión $[c^=]$ no está en ninguna fórmula de R entonces no hace falta tener un nodo con dato c en el data-graph. En realidad, solo hacen falta más debido a que podría haber una subexpresión de la forma $\langle \alpha \neq \beta \rangle$, la cual requiere que el data-graph contenga al menos dos datos distintos. Lo interesante es que **alcanza con dos datos distintos**, debido fundamentalmente a que si agregamos todos los ejes posibles entonces desde cualquier nodo siempre vamos a poder satisfacer la fórmula $\langle \alpha \neq \beta \rangle$ yendo a los dos nodos con datos distintos.

Esta demostración requiere de un enorme trabajo técnico (o más bien no encontramos ninguna forma limpia de hacerla). En el apéndice están los detalles. □

Este lema nos permite hacer la siguiente observación:

Observación 25. *Si existe un superset repair G con respecto a R, entonces existe un superset repair de G con respecto a R que usa una cantidad de data values distintos que depende linealmente de $|G| + |R|$.*

Notemos que esta observación no implica la existencia de un superset repair de tamaño lineal en $|G|+|R|$: podría pasar que haya una cantidad exponencial de nodos con el mismo data value. Sin embargo, podemos 'colapsar' todos los nodos con el mismo data value preservando ciertos ejes de tal forma que el grafo resultante siga satisfaciendo cualquier expresión de Reg-GXPath^{pos} que ya era satisfecha en el grafo original:

Lema 26. *Sea G un data-graph que satisface una restricción de Reg-GXPath^{pos} η. Sea V_d el conjunto de nodos de G con el mismo data value d. Si definimos un nuevo data-graph H donde*

$$V_H = (V_G \setminus V_d) \cup \{v_d\}$$

$$L_H(v, w) = L_G(v, w)\ \forall v, w \in V_G \setminus V_d$$

$$L_H(v, v_d) = \{e \in \Sigma_n \mid \exists w \in V_d \text{ tal que } e \in L_G(v, w)\}\ \forall v \in V_G \setminus V_d$$

$$L_H(v_d, v) = \{e \in \Sigma_n \mid \exists w \in V_d \text{ tal que } e \in L_G(w, v)\}\ \forall v \in V_G \setminus V_d$$

$$L_H(v_d, v_d) = \{e \in \Sigma_n \mid \exists w_1, w_2 \in V_d \text{ tal que } e \in L(w_1, w_2)\}$$

$$D_H(v) = D_G(v)\ \forall v \in V_G \setminus V_d$$

$$D_H(v_d) = d$$

entonces H satisface η.

Demostración. En el nuevo data-graph colapsamos todos los nodos con dato d en un único nodo. Notemos que esto no afecta a las expresiones de Reg-GXPath^{pos} debido a que estas ignoran la "identidad" de los nodos, y solo pueden ver el data value. Aparte en el colapso de los vértices agregamos todos los ejes necesarios para preservar los caminos que satisfacen las expresiones. Como antes, los detalles están en el apéndice. □

Como ya mencionamos, en el grafo H el conjunto de nodos V_d fue colapsado en un único nodo v_d, preservando todos aquellos ejes incidentes a V_d (esta operación suele llamarse 'contracción de vértices'). Usando estos dos lemas, podemos probar el siguiente hecho útil:

Teorema 27. *Sea G un data-graph y R un conjunto de expresiones de Reg-GXPath^{pos}. Si hay un superset repair G con respecto a R, entonces existe un superset repair de G con respecto a R de tamaño polinomial en $|G| + |R|$.*

Demostración. Vamos a definir un data-graph H tal que $G \subseteq H$, $H \models R$, y el tamaño de H depende polinomialmente de $|G| + |R|$. Esto alcanza para probar el teorema.

Sea G' un superset repair de G con respecto a R que usa un número de data values que depende linearmente de $|G|+|R|$ (un superset repair de estas características siempre existe debido al lema 24 y la hipótesis). Ahora, para cada data value $c \in \Sigma_n^{G'} \setminus \Sigma_n^G$, podemos 'contraer' todos los nodos con data value c a un único nodo. Sea H' el grafo resultante tras contraer esos nodos. Notemos que H' satisface R debido al lema 26.

Para cada otro nodo $v \in V_{G'} \setminus V_G$ en H' que tiene un data value en Σ_n^G podemos 'contraerlo' con el nodo perteneciente en G con el mismo data value. Este nuevo grafo es el H que estamos buscando, dado que una vez más, gracias al lema 26, H satisface R y el número de nodos de H es exactamente $|V_G| + |\Sigma_n^{G'} \setminus \Sigma_n^G|$, que depende linealmente de $|G| + |R|$. Por ende el tamaño de H puede ser a lo sumo cuadrático en $|G| + |R|$.

□

Este último teorema muestra que el problema ∃SUPERSET-REPAIR(Reg-GXPath^{pos}) está en NP, dado que podemos pedir el superset repair como testigo. Pero mas aún, podemos usar su demostración para definir un algoritmo que compute un superset repair del data-graph G. Si existe tal superset repair de G con respecto a R entonces hay un 'pequeño' grafo H tal que $H \models R$ y H tiene una estructura muy específica: los únicos nodos agregados a G para obtener H se corresponden con algún data value que no está presente en G pero si en R, junto a potencialmente uno o dos data values más (de ahora en más llamaremos a esta estructura de un supergrafo de G la *forma estándar* de un superset repair). Esto significa que podemos encontrar un data-graph minimal (con respecto a inclusión de nodos) H' que satisface $H' \models R$ y $G \subseteq H'$ iterando sobre todos los posibles subconjuntos S de $(\Sigma_n^R \cup \{c, d\}) \setminus \Sigma_n^G$ (donde c y d son los 'nuevos' data values mencionado en el lema 24), agregando un nodo a G por cada data value en S y cada posible eje entre todo par de nodos, y verificando luego si el data-graph resultante satisface R.

Si ninguno de estos data-graph satisface R entonces podemos concluir en base al teorema 27 que no existe un superset repair. Caso contrario, luego de computar el data-graph minimal (respecto a la inclusión de nodos) H' podemos encontrar un repair quitando ejes iterativamente de H': si después de quitar un eje e de H' sucede que R ya no es satisfecha entonces usando el lema 14 sabremos que no hay un superset repair de H' sin el eje e que satisfaga R (y que siga siendo un data-graph que contenga a G). Esto implica que podemos estar seguros que e va a ser parte del repair final.

Teniendo en cuenta todas estas observaciones podemos definir el siguiente algoritmo:

Algorithm 2 *SupersetRepair*(G, R)

Require: G es un data-graph y R un conjunto de expresiones Reg-GXPath^{pos}.

```
for S ∈ P(Σ_n^R ∪ {c, d} \ Σ_n^G) do
    H ← buildGraph(G, S)
    if H ⊨ R and H ⊆ H' then
        H' ← H
    end if
end for
if H' no está inicializada then
    return 'No existe un superset repair'
end if
for e ∈ E_H' \ E_G do
    if (V_H', E_H' \ {e}, D_H') ⊨ R then
        H' ← (V_H', E_H' \ {e}, D_H')
    end if
end for
return H'
```

El primer **for** va a ser ejecutado a lo sumo $2^{\Sigma_n^R+2}$ veces. Si Σ_n es finito, entonces este número de operaciones va a ser constante (pues $\Sigma_n^R \leq \Sigma_n \leq k$ con k una constante). De otra forma, podría depender exponencialmente de $|R|$. Todo lo que está dentro del loop principal puede ser computado en tiempo polinomial gracias al teorema 11. El segundo loop va a correr a lo sumo $(|V_G| + \Sigma_n^R)^2$ veces, lo cual es cuadrático en el tamaño la de entrada. Dado que el algoritmo es correcto, podemos concluir que:

Teorema 28. *Hay un algoritmo que dada un data-graph G y un conjunto de expresiones*

Reg-GXPath^{pos} computa en tiempo polinomial un superset repair de G con respecto a R si Σ_n es finito.

Si Σ_n es infinito todavía podemos computar un superset repair en tiempo polinomial si R esta fijo.

Si no restringimos de ninguna forma el problema este se vuelve una vez más intratable:

Teorema 29. *El problema* $\exists$SUPERSET-REPAIR$_{Reg\text{-}GXPath^{pos}}$ *es* NP-COMPLETO *en general.*

Demostración. Reducimos el problema 3-SAT a una instancia de $\exists$SUPERSET-REPAIR(Reg-GXPath^{pos}) donde $|R|$ depende de la fórmula de entrada al problema 3-SAT y Σ_n está fijo y es infinito (si no se cumple alguna de estas dos condiciones entonces el algoritmo 3.2 resolvería el problema en tiempo polinomial). Usaremos $\Sigma_e = \{\text{DOWN}\}$ y $\Sigma_n = \{x_i \mid i \in \mathbb{N}\} \cup \{\neg x_i \mid i \in \mathbb{N}\} \cup \{null\}$.

Dada la fórmula en 3-CNF ϕ con n variables y m cláusulas definimos el data-graph $G = (V, L, D)$ donde:

$$V_G = \{v\}$$

$$L(v, v) = \emptyset$$

$$D(v) = null$$

Queremos que todo superset repair de G represente una valuación (o por lo menos una valuación parcial) de las variables x_i. Intuitivamente consideraremos que una variable x_i evalúa a $\top$ dado un superset repair G' si existe un nodo en G' con data value x_i, y evalúa a $\bot$ si existe un nodo en G' con data value $\neg x_i$.

Para obtener una valuación válida alcanza con lograr que x_i y $\neg x_i$ no estén nunca presentes en un mismo repair. Para lograr esto definimos las siguientes restricciones:

$$\alpha_i = ([x_i^{=}] \downarrow^*_{\text{DOWN}} [\neg x_i^{\neq}]) \cup ([x_i^{\neq}] \downarrow^*_{\text{DOWN}} [\neg x_i^{=}]) \cup ([x_i^{\neq}] \downarrow^*_{\text{DOWN}} [\neg x_i^{\neq}]) \tag{3.11}$$

donde i itera sobre $1 \leq i \leq n$. La expresión definida por 3.11 no puede ser satisfecha por ningún grafo que tenga data values x_i y $\neg x_i$ para algún i: notemos que en ninguna de estas tres subexpresiones el camino puede terminar y empezar en x_i y $\neg x_i$ respectivamente. En particular esta fórmula fuerza al grafo a ser fuertemente conexo.

Luego agregamos algunas restricciones para asegurar que en todo superset repair la valuación definida implícitamente satisfaga a todas las cláusulas:

$$\psi_j = (\downarrow^*_{\text{DOWN}} [(c_1^j)^{=}] \downarrow^*_{\text{DOWN}}) \cup (\downarrow^*_{\text{DOWN}} [(c_2^j)^{=}] \downarrow^*_{\text{DOWN}}) \cup (\downarrow^*_{\text{DOWN}} [(c_3^j)^{=}] \downarrow^*_{\text{DOWN}}) \tag{3.12}$$

donde j itera sobre $1 \leq j \leq m$ y c_k^j es el k-ésimo literal que aparece en c_j: x_i o $\neg x_i$, para algún i. Cualquier repair de G va a satisfacer todas las restricciones de 3.12, lo que implica que en la valuación definida por cualquier repair toda cláusula contendrá un literal que evalúe a verdadero.

En resumen, definimos $R = \{\alpha_i \mid 1 \leq i \leq n\} \cup \{\psi_j \mid 1 \leq j \leq m\}$. Ahora probamos que ϕ es satisfacible si y solamente si G tiene un superset repair con respecto a R.

$\Longrightarrow$) Sea f la valuación de las variables de ϕ que evalúa ϕ a verdadero. Definimos el data-graph $H = (V_H, L_H, D_H)$ como:

$$V_H = \{v\} \cup \{x_i \mid 1 \leq i \leq n \text{ y } f(x_i) = \top\} \cup \{\neg x_i \mid 1 \leq i \leq n \text{ y } f(x_i) = \bot\}$$

$$L(a, b) = \Sigma_e \text{ para todo } a, b \in V_H$$

$$D(v) = null$$
$$D(x_i) = x_i \text{ para todos los nodos } x_i \in V_H$$
$$D(\neg x_i) = \neg x_i \text{ para todos los nodos } \neg x_i \in V_H$$

Toda expresión α_i es satisfecha, dado que no hay ningún par de nodos con data values 'opuestos' y el data-graph está totalmente conectado. Las expresiones ψ_j también están satisfechas debido que para todo c_j hay algún literal en la cláusula que evalúa a verdadero presente como data value en H. Como $G \subseteq H$ y $H \models R$ entonces existe un superset repair G' de G con respecto a R.

$\Longleftarrow$) Sea G' un superset repair de G con respecto a R. Definimos una valuación de las variables de ϕ considerando los data values presentes en G'. Si el data value x_i está presente entonces $f(x_i) = \top$, y caso contrario definimos $f(x_i) = \bot$. Observemos que como α_i es satisfecha para todo i esta es una valuación correcta. También notemos que como ψ_j es satisfecha para todo j esta valuación hace que todas las cláusulas evalúen a verdadero.

Tomemos una cláusula arbitraria c_j. Dado que ψ_j es satisfecha en G' luego alguno de los literales x_k^j para $k \in \{1, 2, 3\}$ tiene que aparecer como data value en G'. Si hay un literal positivo (es decir, sin negación) entonces es evaluado como verdadero a través de f, y c_j es satisfecha. Caso contrario hay un literal negativo de c_j como data value en G', y evalúa a falso a través de f, por lo que c_j evalúa a verdadero. Esto muestra que ϕ es satisfacible.

□

4. ENCONTRANDO REPAIRS CON CRITERIOS DE PREFERENCIA

Ya hemos observado que dado un data-graph G y un conjunto de restricciones R puede existir más de un repair. En algunas aplicaciones ciertos repairs podrían ser preferibles en función de algún valor concreto de los datos que contiene o de la estructura que preserva. En esta sección se proponen dos criterios de preferencia distintos para intentar priorizar algunos repairs sobre otros. El primero está basado en la asignación de pesos, mientras que el segundo en generar un orden sobre data-graphs partiendo de uno sobre los edge labels y data values.

4.1. Preferencia basada en pesos

Como se hace en [14] para Logicas de descripción (*Description Logic Knowledge Bases*) definimos una noción de *pesos* sobre data-graphs, que puede ser trasladada a un sistema de preferencias a través de un "puntaje" que se asigna a cada data-graph.

Pesos sobre ejes y data values Al considerar un criterio para elegir sobre un conjunto de repairs un primer enfoque es considerar que algunos edge labels o data values son "preferibles" en el repair a otros. Para traducir esta idea le asignamos pesos a los distintos labels y data values, y esto define naturalmente un peso que se le asigna al data-graph completo. Luego se puede elegir el data-graph perteneciente al conjunto de repairs que maximice (o bien minimice) el peso definido a partir de los pesos de los "elementos" del data-graph.

Definimos entonces una función sobre edge labels y data values:

$$w : \Sigma_e \sqcup \Sigma_n \to \mathbb{N}$$

(donde $\sqcup$ denota la unión disjunta).

Extendemos luego la función a un data-graph cualquiera $G = (V, L_e, D)$ definido sobre Σ_e y Σ_n como:

$$w(G) = \sum_{x,y \in V} \left(\sum_{z \in L_e(x,y)} w(z) \right) + \sum_{x \in V} w(D(x))$$

Finalmente, decimos que $G_1 <_w G_2$ si y solamente si $w(G_1) < w(G_2)$, en cuyo caso decimos que G_1 es ***w*-preferido** por sobre G_2.

Comentario 30. Notemos que una función que asigne el mismo peso a todos los elementos del grafo termina definiendo una preferencia basada en la cardinalidad del repair. Es sabido que la complejidad de los problemas de razonamiento que usan como criterio de optimalidad la cardinalidad suelen ser intratables [28, 29]. Por lo tanto, es esperable que los problemas de razonamiento con este criterio de preferencia no admitan algoritmos eficientes.

Por otro lado, asignar un peso alto a los data values va a hacer que, para el caso de superset repairs, agregar ejes sea más conveniente a agregar nodos; mientras que para el caso de subset repairs eliminar nodos va a ser preferible a eliminar ejes.

Ejemplo 31. Consideremos un contexto en el cual los data-graphs representan redes físicas, y donde los ejes representan conexiones que pueden ser de dos calidades distintas (por ejemplo, variando robustez, seguridad o resistencia a ataques) las cuales se denotan como $\downarrow_{\text{LOW}}$ y $\downarrow_{\text{HIGH}}$.

Sea $R = \{\alpha_{connected\_dir}, \alpha_{2l \to good}\}$ un conjunto de restricciones, donde $\alpha_{connected\_dir} = \_^*$ expresa la noción de conexión dirigida, y $\alpha_{2l \to good} = \downarrow_{\text{LOW}}\downarrow_{\text{LOW}} \Rrightarrow \downarrow_{\text{HIGH}}\downarrow_{\text{LOW}} \cup \downarrow_{\text{LOW}}\downarrow_{\text{HIGH}} \cup \downarrow_{\text{HIGH}}\downarrow_{\text{HIGH}} \cup \downarrow_{\text{HIGH}} \cup \downarrow_{\text{LOW}} \cup \epsilon$ establece que si un nodo es alcanzable mediante dos ejes de baja calidad luego también tiene que ser posible alcanzarlo por un 'buen' camino, lo cual quiere decir que puede ser alcanzado en un paso o bien en dos empleando algún eje de alta calidad.

Podríamos definir una función de pesos que asigne distinta prioridad a ambos tipos de ejes y a las distintas construcciones representadas por los nodos. Por ejemplo, podría asignarle un peso uniforme a todos los data values, definiendo $w(x) = 20$; un costo bajo a los ejes de baja calidad, definiendo $w(\downarrow_{\text{LOW}}) = 1$; y un costo alto a los ejes de alta calidad, definiendo $w(\downarrow_{\text{HIGH}}) = 5$.

Ahora, dado un data-graph G que no satisface las restricciones, un superset repair w-preferido puede ser interpretado como la forma más costo-eficiente de crear un superset de la red que respete las restricciones estructurales de R y a la vez minimizando el costo de construcción definido por w. Para un ejemplo concreto, ver las figuras 4.1, 4.2, y 4.3.

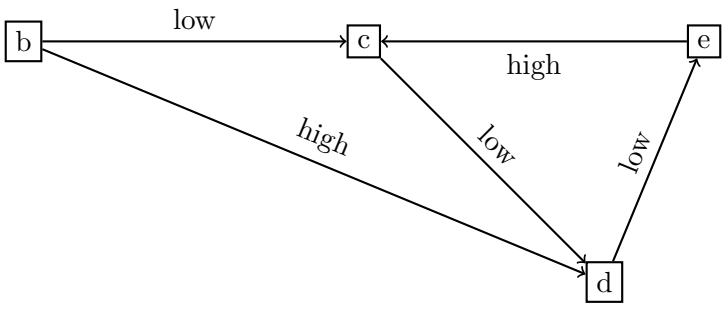

Fig. 4.1: Un data-graph que no satisface las restricciones $\alpha_{connected\_dir}$ ni $\alpha_{2l \to good}$ del ejemplo 31: no está conectado (interpretado como un grafo dirigido), y el par (c, e) está conectado a través de dos ejes $\downarrow_{\text{LOW}}$ pero no puede conectarse a través de un 'buen' camino.

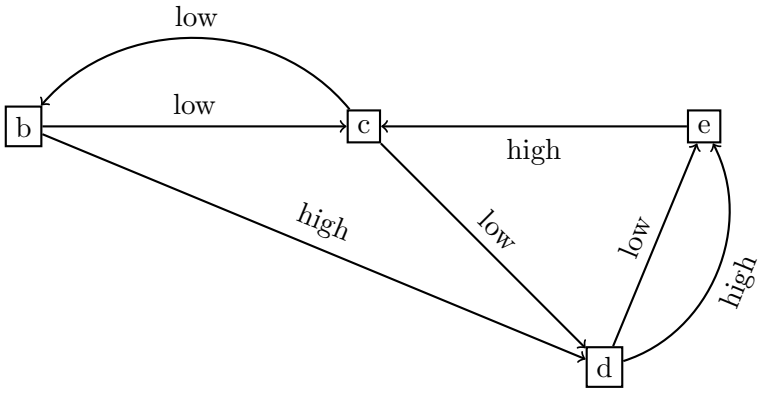

Fig. 4.2: Un posible w superset repair con respecto al ejemplo de la figura 4.1; notemos que si se remueve cualquier eje de este grafo entonces el data-graph resultante no satisface R. El peso asociado a este repair es el peso del data-graph original mas 1 + 5 (de los ejes LOW y HIGH agregados).

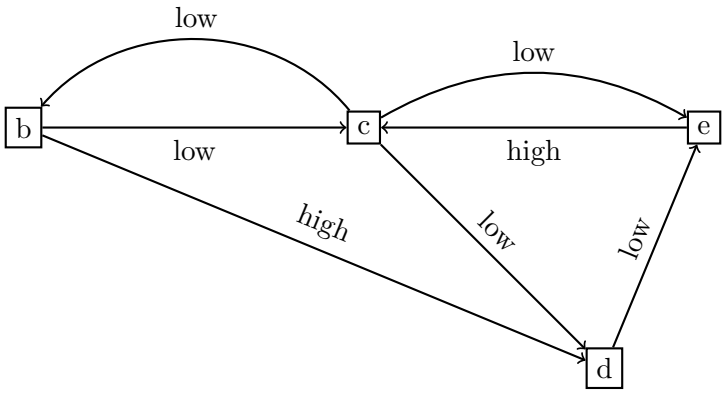

Fig. 4.3: Un w-preferred superset repair con respecto al ejemplo de la figura 4.1. El peso adicional asociado a este repair es 2, y puede ser demostrado que no hay ningún otro superset repair cuyo peso adicional sea menor a 2.

Computando repairs w-preferidos Definimos el siguiente problema:

PROBLEMA: Computo de $\star$-repair w-**preferido** INPUT: Un data-graph G y un conjunto R de expresiones en $\mathcal{L}$. OUTPUT: Un $\star$-repair G' de G con respecto a R y w-**preferido** por sobre todos los otros $\star$-repairs.

donde $\star \in \{\subseteq, \supseteq\}$. La función w está fija, y debe ser una función 'fácil' de computar. Esto es, consideramos que dada una codificación razonable de x vale que $w(x) \leq 2^{p(|x|)}$ para todo x y algún polinomio p; y que el resultado de $w(x)$ puede ser computado en tiempo polinomial en función del tamaño de x. Pueden hacerse otras restricciones a w (por ejemplo, que $w(x) \leq q(|x|)$ para algún polinomio q) dependiendo del tipo de sistema de pesos que se quiera modelar, pero notamos que, sin ningún tipo de restricción, w podría incluso ser incomputable.

Ahora estudiamos el problema para subset y superset repairs por separado.

4.1.1. Subset repairs w-preferidos

En el contexto de subset repairs es más razonable querer **maximizar** el peso de los repairs en vez de minimizarlo. Esta intuición se basa en la idea de que queremos preservar el mayor 'valor' posible del data-graph G original.

Damos un problema de decisión relacionado al cómputo de subset repairs w-preferidos:

PROBLEMA: Computo de un $\subseteq$-repair K **w-preferido** INPUT: Un data-graph G, un natural K y un conjunto R de expresiones de $\mathcal{L}$. OUTPUT: Existe un subset repair G' de G con respecto a R tal que $w(G') \geq K$?

Observación 32. *Hay un algoritmo polinomial para resolver el problema de computar un $\subseteq$-repair w-preferido si y solamente si existe un algoritmo polinomial para resolver el problema de decidir si existe un subset repair K w-preferido.*

Demostración. Solo hay que seguir la estrategia clásica para pasar de problemas de decisión a problemas de búsqueda. La dirección $\implies$ es trivial. Para la otra, podemos encontrar el valor del repair óptimo M haciendo búsqueda binaria en el valor del repair. Luego, podemos encontrar los nodos y ejes que pertenecen al repair objetivo iterando sobre los distintos elementos del grafo, quitándolos, y verificando mediante el problema de decisión si sigue existiendo el subset repair M-preferido contenido dentro del grafo resultante. Este algoritmo hace una cantidad polinomial de consultas al problema de decisión y finalmente realiza una cantidad polinomial de operaciones. □

El problema de decisión está claramente en NP, y es más general que el problema $\exists$-SUBSET-REPAIR, lo cual implica que decidir si existe un $\subseteq$-repair K w-preferido es un problema NP-COMPLETO cuando $\mathcal{L}$ contiene todas las expresiones de Reg-GXPath o bien las expresiones de camino de Reg-GXPath^{pos}. Notemos que si solo se consideran las expresiones de nodo de Reg-GXPath^{pos} entonces hay un único repair, lo cual implica que este problema se puede resolver en tiempo polinomial en ese caso.

Teorema 33. *El problema de computar un $\subseteq$-repair K $w-$preferido es* NP-COMPLETE *cuando $\mathcal{L}$ contiene Reg-GXPath o el conjunto de expresiones de camino de Reg-GXPath^{pos}. Puede ser resuelto en tiempo polinomial (con el mismo algoritmo empleado para resolver el problema de encontrar un subset repair cualquiera) si $\mathcal{L}$ contiene solo expresiones de nodo de Reg-GXPath^{pos}.*

4.1.2. Superset repairs w-preferidos

Consideremos el problema de decisión que planteamos para subset repairs, pero ahora en el contexto de superset repairs:

PROBLEMA: Computo de un $\supseteq$-repair K **w-preferido** INPUT: Un data-graph G, un natural K y un conjunto R de expresiones de $\mathcal{L}$. OUTPUT: Existe un superset repair G' de G con respecto a R tal que $w(G') \leq K$?

Si bien no está claro que podamos hacer una observación análoga a 32 para este problema, es fácil ver que si existe un algoritmo para computar superset repairs w-preferido entonces se podría resolver el problema de decisión que recién plateamos, y por lo tanto podemos derivar cotas inferiores del problema de búsqueda estudiando nuestro problema de decisión.

Cuando se usa el poder completo de Reg-GXPath obtenemos el siguiente resultado:

Teorema 34. *Si $\mathcal{L} \supseteq$ Reg-GXPath entonces existe un conjunto R tal que el problema de decidir si existe un superset repair K w-preferido es* NP-HARD.

Demostración. Observemos que si asignamos valores adecuados a los edge labels y node values del grafo usado en la demostración del teorema 22 podemos forzar a que todos los superset repairs solo puedan agregar ejes con label VALUE a los nodos variable. Esto implica que podemos reducir el problema 3-SAT al de decidir si existe un $\supseteq$-repair K w-preferido incluso cuando R está fijo. □

Para el caso de Reg-GXPath^{pos}, notemos que el repair de tamaño polinomial que construíamos en función del teorema 27 no agregaba ni ejes ni nodos al repair original (la contracción de nodos borraba nodos y no podía aumentar la cantidad de ejes, dado que cada eje que agregábamos al nodo producto de la contracción se correspondía con alguno incidente en uno de los nodos contraídos). Es decir, si existe un repair w-preferido de G con respecto a R entonces este tiene tamaño polinomial en función de $|G|$ y $|R|$ (cuando se consideran solo expresiones de Reg-GXPath^{pos}).

Esto nos permite concluir fácilmente que el problema de decisión está en NP para este caso (podemos pedir el superset repair w-preferido como testigo). Ahora probamos que el problema también es NP-HARD.

Teorema 35. *Existe un conjunto de expresiones de nodo de Reg-GXPath^{pos} R tal que el problema de decidir si existe un superset repair K w-preferido es* NP-HARD.

Demostración. Reducimos 3-SAT al problema K w-**preferido** con un R fijo. Para la reducción consideramos $\Sigma_e = \{\text{VALUE\_OF}, \text{APPEARS\_IN}, \text{APPEARS\_NEGATED\_IN}\}$ y $\Sigma_n = \{clause, var, \top, \bot\}$, con la misma semántica que en la demostración del teorema 12. Definimos la función de pesos w como $w(c) = 2$ para $c \in \Sigma_n \cup \{\text{APPEARS\_IN}, \text{APPEARS\_NEGATED\_IN}\}$ y $w(\text{VALUE\_OF}) = 1$.

Dada una fórmula en 3-CNF ϕ con n variables y m cláusulas construimos un data-graph G, un conjunto de expresiones de nodo de Reg-GXPath^{pos} R y un número K tal que ϕ es satisfacible si y solamente si G tiene un superset repair G' con respecto a R tal que $w(G') \leq K$.

El grafo es similar al empleado en el teorema 12, y lo definimos como $G = (V_G, L_G, D_G)$ donde:

$$V_G = \{x_i \mid 1 \leq i \leq n\} \cup \{c_j \mid 1 \leq j \leq m\} \cup \{\bot, \top\}$$

$$L_G(x_i, c_j) = \begin{cases} \{\text{APPEARS\_IN}, \text{APPEARS\_NEGATED\_IN}\} \text{ if } x_i \text{ and } \neg x_i \text{ appear in } c_j \\ \{\text{APPEARS\_IN}\} \text{ if } x_i \text{ appears in } c_j \\ \{\text{APPEARS\_NEGATED\_IN}\} \text{ if } \neg x_i \text{ appears in } c_j \\ \emptyset \text{ otherwise} \end{cases}$$

$$L_G(v, w) = \emptyset \text{ para todo otro par de nodos } v, w \in V_G$$

$$D(\bot) = \bot,\ D(\top) = \top$$
$$D(x_i) = var \text{ para } 1 \leq i \leq n$$
$$D(c_j) = clause \text{ para } 1 \leq j \leq m$$

La estructura de ϕ está codificada en los ejes de tipo APPEAR. Tomamos $K = w(G)+n$, y queremos que cualquier superset repair de G con respecto a R con peso K codifique una asignación de los nodos variable usando los ejes VALUE_OF. Para hacer esto usamos las expresiones de Reg-GXPath^{pos}

$$\psi_1 = \langle [var^{\neq}] \cup \downarrow_{\text{VALUE\_OF}} [\top] \cup \downarrow_{\text{VALUE\_OF}} [\bot] \rangle \tag{4.1}$$

$$\psi_2 = \langle [clause^{\neq}] \cup \downarrow^{-}_{\text{APPEARS\_IN}} \downarrow_{\text{VALUE\_OF}} [\top] \cup \downarrow^{-}_{\text{APPEARS\_NEGATED\_IN}} \downarrow_{\text{VALUE\_OF}} [\bot] \rangle \tag{4.2}$$

La expresión 4.1 fuerza a que todos los nodos variable tengan un eje de label VALUE dirigido a un nodo booleano, mientras que la expresión 4.2 fuerza a todas las cláusulas a estar "satisfechas" en todo repair de G. Por lo tanto, definimos $R = \{\psi_1, \psi_2\}$. Ahora probamos que ϕ es satisfacible si y solamente si G tiene un superset repair con respecto a R con peso a lo sumo $w(G) + n$.

$\Longrightarrow$) Sea f una valuación de las variables de ϕ que evalúa ϕ a verdadero. Luego agregamos ejes a G de la siguiente forma: si $f(x_i) = \top$ agregamos el eje $(x_i, \text{VALUE\_OF}, \top)$, y caso contrario el eje $(x_i, \text{VALUE\_OF}, \bot)$. Este grafo satisface ambas expresiones de R y tiene costo $w(G) + n$, dado que $w(\text{VALUE\_OF}) = 1$.

$\Longleftarrow$) Sea G' un superset repair de G con respecto a R con peso a lo sumo $w(G) + n$. Como todo superset repair de G tiene que agregar por lo menos n ejes de label VALUE y estos tienen costo 1 sabemos que G' tiene que ser exactamente el grafo original G con n ejes VALUE agregados (uno por cada nodo variable). Por lo tanto podemos definir una valuación sobre las variables de ϕ usando estos ejes: si el eje $(x_i, \text{VALUE\_OF}, \top)$ está presente en G' tomamos $f(x_i) = \top$, y caso contrario $f(x_i) = \bot$. Como ψ_2 es satisfecha en G' entonces esta valuación satisface ϕ.

□

Concluimos finalmente que:

Corolario 36. *El problema de decidir si existe un superset repair K w-preferido* NP-COMPLETO *cuando $\mathcal{L} \supseteq$ Reg-GXPath^{pos}.*

Observemos que sin pesos el problema podía ser tratable bajo ciertas condiciones (con Σ_n finito o bien R fijo), por lo que el criterio de preferencia afectó la complejidad computacional del problema en el caso de superset repairs.

4.2. Preferencia basada en multiconjuntos

Ahora vamos a introducir un criterio de preferencias basado en un orden sobre multiconjuntos. Específicamente vamos a construir, dado un orden sobre los elementos de los data-graphs (i.e. sobre los data values y edge labels), un orden sobre los multiconjuntos que representan a los grafos. Notemos que al representar a un data-graph como un multiconjunto se pierde la capacidad de detectar propiedades topológicas asociadas con la forma en que están distribuidos los ejes.

Dado un conjunto A, su conjunto de multiconjuntos finitos $\mathcal{M}_{<\infty}(A)$ se define formalmente como

$$\mathcal{M}_{<\infty}(A) = \{f : A \to \mathbb{N} \mid \{x \mid f(x) \neq 0\} \text{ es finito}\}.$$

Intuitivamente un multiconjunto finito sobre A es simplemente una función de A a $\mathbb{N}$ que tiene una finita cantidad de valores para los que es no nula. Dados dos multiconjuntos M_1, M_2 es común usar la notación $M_1 \cup M_2$ para denotar al multiconjunto $M_1 + M_2$ y también representar a los multiconjuntos como conjuntos con elementos repetidos (por ejemplo, $\{0, 3, 3, 4, 4, 4, 10\}$).

Dado un orden parcial estricto $(A, <)$ se puede definir un orden sobre $\mathcal{M}_{<\infty}(A)$ (i.e. la extensión a multiconjuntos $(\mathcal{M}_{<\infty}(A), <_{mset})$) de la siguiente forma (tomada de [18, 24]):

$$M_1 <_{mset} M_2 \iff M_1 \neq M_2 \text{ y para todo } x \in A, \text{ si } M_1(x) > M_2(x), \text{ entonces existe un } y \in A \text{ tal que } x < y \text{ y } M_1(y) < M_2(y).$$

Intuitivamente, $M_1 <_{mset} M_2$ si para todo valor de x ocurre que $M_1(x) \leq M_2(x)$ (salvo para algún x en donde la desigualdad es estricta) o bien si hay algún x para el que $M_1(x) > M_2(x)$ entonces hay algún y más adelante en el orden (es decir, tal que $x < y$) que cumple $M_1(y) < M_2(y)$. Podemos pensarlo de la siguiente forma: si bien M_1 le gana a M_2 en x, M_1 pierde contra M_2 en y, y y es mas "importante" que x (ya que $x < y$). Cuando $(A, <)$ es un orden total esta definición es equivalente a

$$M_1 <_{mset} M_2 \iff M_1 \neq M_2 \text{ y, si } d = \text{máx}\{z \in A \mid M_1(z) \neq M_2(z)\}, \text{ tenemos que } M_1(d) < M_2(d).$$

Ejemplo 37. Considerando los multiconjuntos $\mathcal{M}_{<\infty}(\mathbb{N})$ tenemos que $\{0, 0, 0, 1, 1, 1, 2\} <_{mset} \{2, 2\}$.

Si $(A, <)$ es un orden parcial estricto (resp. total) entonces $(\mathcal{M}_{<\infty}(A), <_{mset})$ es un orden parcial (resp. total). Si $(A, <)$ es un orden bien fundado[1] entonces también lo es $(\mathcal{M}_{<\infty}(A), <_{mset})$[18].

Generando un criterio de preferencia a partir de un orden sobre multiconjuntos Primero consideramos el caso en el cual hay un orden parcial sobre los edge labels y los data values, el cual queremos "elevar" a un orden sobre multiconjuntos para poder definir una noción de preferencia sobre data-graphs.

Definición 38. Dado un data-graph $G = (V, L_e, D)$, definimos su **multiconjunto de edge labels y data values** como el multiconjunto:

$$G^{\mathcal{M}} = \bigcup_{x,y \in V} L_e(x, y) \cup \bigcup_{x \in V} D(x)$$

Definición 39. Sean G_1, G_2 dos data-graphs sobre Σ_e y Σ_n, y sea $<$ un orden parcial definido sobre la unión (disjunta) $\Sigma_e \sqcup \Sigma_n$. Decimos que $G_1 <_{Gmset} G_2$ (G_1 es **multiset-preferido** por sobre G_2) si:

$$G_1^{\mathcal{M}} <_{mset} G_2^{\mathcal{M}}$$

Decimos que $G_1 \equiv_{Gmset} G_2$ si:

$$G_1^{\mathcal{M}} = G_2^{\mathcal{M}}$$

[1] Es decir, para todo $S \subseteq A$, si $S \neq \emptyset$ entonces existe un $m \in S$ tal que no hay ningún $s \in S$ tal que $s < m$.

Comentario 40. Al usar preferencias basada en multiconjuntos para los superset repairs los edge labels o data values con una "posición" menor en $(\Sigma, <)$ van a ser preferibles en los repairs. Para el caso de los subset repairs, los edge labels y data values con una posición alta van a ser removidos primero en caso de que haya que quitar elementos del grafo original.

Ejemplo 41. Consideremos una base de datos que representa una región geográfica y los distintos caminos que hay entre ubicaciones específica (denotadas por los nodos). Sobre esta base esperamos que si dos puntos están conectados por un camino entonces también debería estar indicado si la conexión es mediante un camino de asfalto o bien uno de tierra. Tomemos un orden sobre el conjunto de labels tal que DIRT $<$ ASPHALT $<$ ROAD, y un conjunto de restricciones $R = \{\alpha_{road-type-specified}\}$, que fuerza a los nodos conectados a 'informar' qué tipo de camino los conecta:

$$\alpha_{road-type-specified} = \text{ROAD} \Rrightarrow \text{DIRT} \cup \text{ASPHALT}$$

En estas condiciones, un superset repair basado en un orden de preferencia sobre multiconjuntos representa la asunción conservadora de que un camino entre dos nodos es de baja calidad antes que de alta (ya que al buscar un superset repair agregas ejes DIRT es preferible a los ASPHALT, sin importar la cantidad de los mismos).

Ejemplo 42. Sea $\Sigma_e = \{\downarrow\}$, $\Sigma_n = \{\text{C}, \text{D}, \text{R}\}$. G consiste de un único nodo con data value C. Sea $R = \{\varphi, \alpha_{noCycles}\}$, donde $\alpha_{noCycles} = \epsilon \Rrightarrow \overline{\_^+}$ y $\varphi = \text{C}^= \Rightarrow (\langle\downarrow [\text{R}^=]\rangle \vee \langle(\downarrow [\text{D}^=])^{1000,1000}\rangle)$. Si nuestro orden sobre $\Sigma_e \sqcup \Sigma_n$ esta dado por $\downarrow < \text{C} < \text{D} < \text{R}$, un superset que agregue 1000 nodos (con data value D) y ejes $\downarrow$ (ver figura 4.5) va a ser preferido a uno en que solo tenemos que agregar un nodo con data value R y un eje (ver figura 4.4).

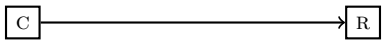

Fig. 4.4: Un superset repair para el data-graph G del ejemplo 42

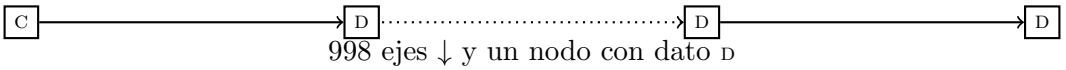

Fig. 4.5: Un superset repair multiset-preferido del data-graph G del ejemplo 42, dado el orden subyacente $\downarrow < \text{C} < \text{D} < \text{R}$.

Pedir un orden parcial sobre $\Sigma_e \sqcup \Sigma_n$ puede ser muy restrictivo considerando preferencias basadas en multiconjuntos. Por ejemplo, si el data value C es menor que D (C $<$ D), entonces un superset repair que solo agregue un nodo con data value C puede ser menos preferido a uno que requiera agregar una cantidad arbitraria de nodos con data value D (dependiendo también del resto de los elementos que tenga que agregarse al grafo para satisfacer las restricciones). Ver el ejemplo 42 para un modelo de juguete que represente esta situación. Este comportamiento puede ser molesto en situaciones en que la diferencia semántica entre distintos data values es poca, como por ejemplo entre los datos 3000 y 3001 en una base de datos en donde los nodos representan objetos y los datos valores en alguna moneda; o bien cuando queremos definir "grupos" de data values sin imponer un orden interno a cada grupo, lo cual podría pasar si los nombres representan cadenas de

caracteres (nombres) y tenemos categorías de datos como "Usuarios de una red social" y "títulos de libros".

Este tipo de comportamiento puede ser evitado si relajamos los requerimientos que le pedimos a la relación subyacente $\Sigma_e \sqcup \Sigma_n$. Por ejemplo, solo pidiendo que sea un preorden. Esta definición permitiría que muchos elementos pertenezcan a la misma clase de equivalencia definida por $\leq$, la cual es una forma de agrupar elementos que se comporten de formas semánticamente iguales.

Especificamos esta definición más general de la siguiente forma:

Elevando un preorden a un criterio de preferencia sobre multiconjuntos Si $\Sigma = \Sigma_e \sqcup \Sigma_n$ tiene un preorden asociado $\leq_q$ (i.e. una relación reflexiva y transitiva), podemos tomar el *quotient set* de clases de equivalencias de la relación de equivalencia dada por la definición $x \equiv y \Leftrightarrow x \leq_q y \wedge y \leq_q x$, obteniendo luego un orden parcial $(\Sigma_{/\equiv}, \leq)$ del cual podemos extraer un orden parcial estricto $(\Sigma_{/\equiv}, <)$. Por ende, aplicando el cociente, podemos extender nuestra noción de preferencia basada en multiconjuntos para que surja de un preorden definido sobre los conjuntos Σ_n y Σ_e.

Definición 43. Sean G_1, G_2 dos data-graphs sobre Σ_e y Σ_n, y sea $\leq_q$ un orden parcial definido sobre la unión disjunta $\Sigma_e \sqcup \Sigma_n$.

Desde $(\Sigma, \leq_q)$ definimos $(\Sigma_{/\equiv}, <)$ como se hizo antes. Nuestra noción de *multiset-preferido* está dada por:

$$G_1 <_{Gmset} G_2 \iff {G_1}^{\mathcal{M}}_{/\equiv} <_{mset}^{(\Sigma_{/\equiv}, <)} {G_2}^{\mathcal{M}}_{/\equiv},$$

donde para todo $[x] \in \Sigma_{/\equiv}$ tenemos:

$${G_i}^{\mathcal{M}}_{/\equiv}([x]) = \sum_{y \in [x]} G_i^{\mathcal{M}}(y)$$

También definimos:

$$G_1 \equiv_{Gmset} G_2 \iff {G_1}^{\mathcal{M}}_{/\equiv} = {G_2}^{\mathcal{M}}_{/\equiv}$$

Observación 44. *La definición 43 es efectivamente una extensión de la definición 39, dado que coinciden cuando el orden subyacente es parcial (las clases de equivalencia sobre ordenes parciales son singletons).*

Comentario 45. Si bien podremos considerar un orden conjunto sobre $\Sigma_e \sqcup \Sigma_n$, en muchos casos tiene más sentido tomar dos ordenes distintos: uno sobre Σ_e y otro sobre Σ_n. Estos dos ordenes luego se extienden a un orden disjunto sobre la unión de los conjuntos.

Comentario 46. Dependiendo del dominio de uso de la base de datos, podría pasar que los nodos tengan un valor especial distinto al de los ejes, y que por lo tanto se prefiera que para buscar un repair solo se hagan operaciones sobre los ejes. Este caso puede ser fácilmente modelado mediante la noción de multiset-preferido.

Supongamos que se está buscando un superset repair. Entonces, queremos un ordenamiento tal que todos los data values sean equivalentes, y que para todo edge label e valga que e es menor (en el orden) a cualquier data value: a partir de $(\Sigma_e, <)$ definimos el preorden $(\Sigma_e \sqcup \Sigma_n, \leq')$ como $x \leq' y \iff x = y$, o $x, y \in \Sigma_e$ y $x < y$, o $x, y \in \Sigma_n$, o $x \in \Sigma_e, y \in \Sigma_n$. Así, todos los data values tiene una prioridad idéntica y mayor a cualquier edge label.

En el caso de que se estén buscando subset repairs alcanza con invertir el orden propuesto.

Computando repairs multiset-preferidos Definimos el siguiente problema

PROBLEMA: cómputo de un $\star$-repair **multiset-preferido** INPUT: Un data-graph G y un conjunto R de expresiones de $\mathcal{L}$. OUTPUT: Un $\star$-repair **multiset-preferido** G' de G con respecto a R tal que no exista otro repair H que sea **multiset-preferido** a G'.

donde $\star \in \{\subseteq, \supseteq\}$. Asumimos un orden parcial estricto fijo $(\Sigma, <)$ sobre el conjunto $\Sigma = \Sigma_e \sqcup \Sigma_n$ de data values y edge labels. Como hicimos en el caso de los repairs w-preferidos condicionamos al orden parcial estricto a ser eficientemente computable: asumimos que dados dos elementos $x, y \in \Sigma_e \sqcup \Sigma_n$ podemos decidir si $x < y$ en tiempo polinomial en función del tamaño de la codificación de x y y.

Para estudiar el problema de computar repairs multiset-preferidos definimos el siguiente problema de decisión:

PROBLEMA: $\star$-repair **multiset-preferido** acotado INPUT: Un data-graph G, un conjunto R de expresiones de $\mathcal{L}$ y un elemento $d \in \Sigma_n \sqcup \Sigma_e$. OUTPUT: Existe un repair multiset-preferido de G con respecto a R tal que no haya ningún elemento $d' \in \Sigma_n^G \sqcup \Sigma_e^G$ que satisfaga $d < d'$?

Este problema de decisión es similar al definido para el caso de los repairs w-preferidos: queremos decidir si d es una 'cota superior' (basada en el orden $<$) sobre los elementos de algún repair multiset-preferido de G con respecto a R. Si el orden es total este problema de decisión es muy intuitivo, pero si es parcial podría ser extraño. En ese caso se puede pensar que el problema recibe un conjunto de elementos $\mathcal{D}$ y que queremos encontrar un repair que sea acotado por todos los elementos en $\mathcal{D}$.

4.2.1. Subset repairs multiset-preferidos

En el caso de subset repairs está claro que el problema está en NP, dado que como certificado positivo alcanza con pedir un data-graph G' y verificar que $G' \subseteq G$, $G' \models R$ y que para todo $d' \in \Sigma_n^{G'} \sqcup \Sigma_e^{G'}$ valga que $d \not< d'$.

Notemos que si el orden es "despreciable" el función del grafo de entrada entonces probar la existencia de un superset repair multiset preferido es lo mismo que probar la de uno cualquiera. Con "despreciable" queremos decir que el d de entrada al problema ya sea un elemento que cumpla que $d' \leq d$ para todo elemento del grafo G. Por lo tanto, podemos usar todas las cotas que ya deducimos para el caso sin preferencias. Con esto concluímos que:

Teorema 47. *El problema $\star$-repair multiset-preferido acotado es* NP-HARD *cuando $\mathcal{L} \supseteq$ Reg-GXPath^{pos}, incluso si R está fijo.*

Para el caso en que R solo contiene expresiones de nodo el problema es tratable, dado que hay un único candidato a ser un repair.

4.2.2. Superset repairs multiset-preferidos

Como ya habíamos visto, encontrar un superset repair es un problema intratable (incluso si R está fijo) cuando se permite escribir restricciones que usen todo el poder de Reg-GXPath. Por lo tanto, estudiamos la complejidad del problema cuando las restricciones se definen con Reg-GXPath^{pos}.

Notamos que, una vez más, la transformación del teorema 27 no agrega ejes ni data values. Por lo tanto sabemos que si existe un superset repair multiset preferido G' de G con respecto a R acotado por d entonces tiene que haber un con la forma 'estándar': uno que agrega a G un nodo por cada data value nuevo en $\Sigma_n^{G'} \setminus \Sigma_n^G$. Esto implica que si la respuesta al problema de decisión es positiva luego podemos encontrar un certificado de tamaño polinomial en $|G| + |R|$, y por lo tanto concluímos que el problema está en NP.

Como vimos en el caso de subset, si el orden es "despreciable" el problema es equivalente al de buscar un superset repair cualquiera. Por lo tanto:

Teorema 48. *Si R no está fijo y Σ_n puede ser infinito entonces existe un caso en el cual el problema de encontrar un $\supseteq$-repair multiset-preferido dada una cierta cota es* NP-HARD.

Demostración. Se puede usar la reducción del teorema 29 considerando un orden tal que ningún elemento es comparable con la cota de entrada. Esto hace que cualquier repair cumpla estar acotado por el valor d. □

Antes encontramos un caso tratable para el problema de computar superset repairs: si R está fijo o $\Sigma_n < \infty$. En este caso podemos usar un algoritmo similar al de entonces para decidir la existencia de un superset repair multiset preferido acotado por d:

Algorithm 3 $SupersetRepairBound(G, R, d)$

Require: G es un datagraph, R un conjunto de expresiones de Reg-GXPath^{pos} y $d \in \Sigma_n \cup \Sigma_e$.

1: $c_1, c_2 \leftarrow$ dos data values tales que $d \not\sim c_i$ y $c_i \notin \Sigma_n^R \cup \Sigma_n^G$ (si es posible)
2: $\Sigma_n^{d\not\sim} \leftarrow$ edge labels e tales que $d \not\sim e$
3: **for** $S \in \mathcal{P}(\Sigma_n^{R_d} \cup \{c_1, c_2\} \setminus \Sigma_n^G)$ **do**
4: $\quad H \leftarrow buildGraph(G, S, \Sigma_n^{d\not\sim})$
5: $\quad$ **if** $H \models R$ y está acotado por d **then**
6: $\quad\quad$ **return** H como un certificado positivo
7: $\quad$ **end if**
8: **end for**
9: **return** No existe un superset repair que tenga a d como cota.

La función $buildGraph(G, S, \Sigma)$ genera un nuevo grafo agregando a G un nodo v_c por cada data value $c \in S$, definiendo $D(v_c) = c$ y agregando todos los ejes de Σ entre cada par de nodos. Ahora probamos la correctitud del algoritmo:

Teorema 49. *Dado un data-graph G, un conjunto de expresiones de Reg-GXPath^{pos} R y una cota d el algoritmo SupersetRepairBound devuelve un certificado H si y solamente si G tiene un superset repair multiset-preferido con respecto a R acotado por d.*

Demostración. $\implies$) Supongamos que el algoritmo devuelve un data-graph H. Como $H \models R$ entonces existe un superset repair G' de G con respecto a R contenido en H. Si

G' también es multiset-preferido terminamos, dado que como H está acotado por d ocurre que G' también lo está. Entonces, supongamos que existe otro superset repair I tal que $I <_{Gmset} G'$ y que no hay otro repair I' tal que $I' <_{Gmset} I$. Probemos que I está acotado por d.

Por contradicción, supongamos que hay un elemento d' en I tal que $d < d'$. Como $I <_{Gmset} G'$ sabemos que G' contiene una cantidad de elementos d' mayor o igual a la de I, o bien hay otro elemento d'' tal que $d' < d''$ y $G'(d'') > I(d'')$. El primer caso no puede suceder porque implicaría que G' contiene un elemento mayor a d (de acuerdo al orden), y por construcción G' solo contiene elementos en H, el cual no contenía elementos mayores a d. El segundo caso tampoco puede valer, dado que por la transitividad de $<$ se puede deducir que $d < d''$, lo cual también implica que H contiene un elemento mayor a d.

Por lo tanto existe un superset repair multiset-preferido de G con respecto a R acotado por d.

$\Longleftarrow$) Asumamos que G tiene un superset repair multiset-preferido G' con respecto a R que está acotado por d. Notemos que G' tiene que estar en forma estándar, pues caso contrario podemos contraer dos nodos con el mismo data value y obtener un superset repair mas chico (que sería priorizado en el orden siguiendo la semántica que definimos). Observemos que podemos usar el lema 24 para concluir que dado G' podemos construir un nuevo data-graph K que solo usa dos data values más que aquellos mencionados en $\Sigma_n^R \cup \Sigma_n^G$ y que también satisface R. Siguiendo la prueba del lema 24 tomamos dos data values distintos c_1 y c_2 en $\Sigma_n^{G'} \setminus (\Sigma_n^R \cup \Sigma_n^G)$ y cambiamos los data values de todos aquellos nodos con dato distintos a c_1 o c_2 para que valgan c_1. Contraemos después todos los nodos con dato c_1, y obtenemos un grafo que sigue siendo consistente con respecto a R. Llamemos a este grafo K'.

Por construcción cada elemento x de K' satisface $d \not< x$ (notemos que partimos el procedimiento desde G' que estaba acotado por d, y obtuvimos K' borrando elementos de G' o agregando data values ya presentes en G') y K' está en forma estándar. También el conjunto de ejes que usa K' es claramente un subconjunto de $\Sigma_n^{d\not<}$. Notemos que si 'completamos' K' agregando entre cada nodo los edge labels en $\Sigma_n^{d\not<}$ entonces el grafo resultante K'' sigue satisfaciendo R debido al teorema 14 (monotonía), y también está acotado por d.

Supongamos que K'' tiene dos data values f_1 y f_2 distintos a aquellos mencionados en R, y sean c_1 y c_2 los data values elegidos por el algoritmo en el primer paso. Si cambiamos los data values f_1 y f_2 en K'' por c_1 y c_2 el data-graph resultante va a satisfacer R, dado que ninguno de los nuevos valores está mencionado en R, y cada desigualdad de datos que interactuaba con los datos c_1 y c_2 va a seguir valiendo. Notemos que este data-graph va a ser encontrado por el algoritmo, y será devuelto como certificado.

□

Corolario 50. *Si la primera línea del algoritmo 3 puede ser computada en tiempo polinomial en función del tamaño de entrada del problema luego el problema de decidir si existe un superset repair multiset-preferido acotado por un cierto valor puede ser resuelto en tiempo polinomial si $\mathcal{L} \supseteq$ Reg-GXPath^{pos} si Σ_n es finito o R está fijo.*

Esto se deduce de asumir que la primera línea del algoritmo puede ser realizada eficientemente y observando que la cantidad de iteraciones está acotada por una constante si R está fijo o Σ_n es finito.

La suposición de que la primera linea puede ser computada eficientemente es razonable si el orden es tal que, dado un elemento d, se puede enumerar eficientemente el conjunto de elementos X tal que $d \not< x$ para todo $x \in X$. Esto pasaría, por ejemplo, si el orden $<$ representa un orden lexicográfico.

5. TRABAJO RELACIONADO

Modelos para bases de datos con forma de grafo y *knowledge graphs* han sido desarrollados intensamente desde los 90s, junto a lenguajes de consulta y restricciones de integridad para los mismos[4, 9]. Durante los últimos años fueron ganando relevancia de parte de la industria y la academia debido a su eficiencia al momento de modelar datos diversos, dinámicos y a gran escala (ver [23] para un extenso tutorial sobre el estado del arte). Hay importantes knowledge graphs *open source* como YAGO [33] o DBPEDIA [25] que implementan muchos de los *features* estudiados por la comunidad académica.

Reg-GXPath es un lenguaje de consulta desarrollado para bases de datos con forma de grafo con datos en los nodos [26] en gran parte inspirado en *XPath* [10]; un lenguaje útil para definir caminos sobre *data trees* (i.e. documentos XML). La expresividad de Reg-GXPath es entendida con precisión en relación a otros lenguajes de consulta [26], pero aun hay algunos problemas abiertos relacionados al mismo, como los problemas de *Query Containment* o *Query Equivalence* [16, 17].

Otros lenguajes que fueron candidatos para este trabajo son RQMs y RDQs [27]. El primero es más expresivo que Reg-GXPath, pero su evaluación pertenece a PSPACE; mientras que las RDQs son menos expresivas que Reg-GXPath y por lo tanto su evaluación es más simple. La principal ventaja de Reg-GXPath en comparación a otros lenguajes de consulta es su gran balance entre expresividad y complejidad de evaluación (siendo está polinomial en el tamaño del grafo y la consulta), aparte del hecho de que puede interactuar con la topología del grafo y al mismo tiempo los datos que contiene.

Diferentes tipos de restricciones de integridad han sido definidas y estudiadas en el contexto de bases de datos semi estructuradas [15, 1, 7]. Aún no hay una definición estándar de *consistencia* para un data-graph sobre un conjunto de restricciones, y es por eso que desarrollamos nuestra propia noción basada en ejemplos típicos presentes en la literatura. Otro tipo de restricciones pueden ser expresadas mediante *graph patterns* y *graph dependencies* [20].

La noción de *database repair* y del problema de *repair computing*, asi como el problema CQA o *Consistent Query Answering*; fueron introducidos en el caso relacional en [5]. Desde encontes CQA ha recibido mucha atención por la comunidad académica en el área de *data management*, en donde se desarrollaron técnicas y algoritmos eficientes bajo distintas nociones de repair y considerando varias combinaciones de restricciones de integridad y lenguajes de consulta (ver [12] o [13] para una introducción). Estos conceptos fueron extendidos a diversos modelos de datos como los data trees o las lógicas de descripción [11, 35, 30, 38]. En el caso de los data-graphs hay trabajos relacionados al problema de CQA en grafos sin datos [7], pero, hasta donde sabemos, ningún trabajo ha estudiado el problema de computar repairs (considerando grafos con o sin datos).

Cuando se estudian los problemas de CQA o de computar repairs en contextos como lógicas de descripción es más natural concentrarse en subset repairs, ya que las semánticas de las DLs (*Description Logics*) es *open world* por naturaleza. Esto quiere decir que la no presencia de un hecho en la base de datos (tanto explícito como implícito) no alcanza para derivar la negación de ese hecho. En el caso de bases de datos con forma de grafo hay aplicaciones que pueden corresponderse tanto con una semántica *closed* como *open world*, y por lo tanto ambos tipos de set-based repairs pueden ser relevantes. En este

trabajo consideramos a los dos, pero notamos que dejamos para el futuro el estudio de los *symmetric-based repairs* los cuáles sí fueron considerados en [7]. Usualmente razonar sobre repairs definidos sobre la diferencia simétrica es mucho más difícil, y en el caso de grafos con datos la definición de *diferencia simétrica* no es tan directa de formalizar como en el caso de los modelos estudiados por [7].

6. CONCLUSIONES

En este trabajo presentamos un modelo de base de datos con forma de grafo junto a una noción de consistencia basada en un conjunto de expresiones de Reg-GXPath. Probamos varios resultados relacionados a la complejidad del problema de computar subset y superset repairs de data-graphs dado un data-graph y un conjunto de restricciones (los cuales están resumidos en las tablas 6.1 y 6.2). En muchos casos obtuvimos una cota superior de NP (en todos salvo cuando consideramos superset repairs con el poder completo de Reg-GXPath).

Cuando restringimos el lenguaje para las restricciones al fragmento positivo de Reg-GXPath, llamado Reg-GXPath^{pos}, obtuvimos algoritmos polinomiales para el problema de computar superset repairs, y si solo se permiten expresiones de nodo de Reg-GXPath^{pos} también definimos un algoritmo polinomial para encontrar subset repairs. Notamos que estos algoritmos corren en tiempo polinomial dependiendo tanto en el tamaño del data-graph G como del tamaño del conjunto de restricciones R (esto es, considerando la *complejidad combinada* del problema). Aparte, la mayoría de las cotas superiores fueron demostradas en reducciones donde el conjunto de restricciones estaba fijo, y por lo tanto obtuvimos resultados sobre la *data complexity* del problema.

Finalmente, desarrollamos dos sistemas de preferencia *data-aware* para definir un orden sobre el conjunto de repairs, y mostramos que en muchos casos el problema sigue siendo tan difícil como computar un repair cualquiera (ver table 6.2).

Hay algunas preguntas que siguen abiertas en este contexto, como por ejemplo la complejidad precisa del problema de computar un superset repair, tanto con como sin preferencias, al considerar todo el lenguaje Reg-GXPath. También sería interesante estudiar la complejidad de estos problemas cuando se considera una noción de consistencia *local*, como en [7]: en vez de requerir que las expresiones se satisfagan en cada nodo podríamos considerar un nodo especial o y pedir que para todo otro nodo x se satisfaga que el par (o, x) pertenezca a la evaluación de cada expresión de camino.

La complejidad de CQA en este contexto aún no fue estudiada. Tomando en cuenta los resultados en [35] y [7] es razonable esperar que este problema sea más difícil en el caso de superset repairs, pero la complejidad precisa del problema es desconocida. De todas formas, notemos que en base a los resultados de este trabajo se puede mostrar que el problema de CQA es trivial en algunos casos: por ejemplo, cuando se consideran expresiones de Reg-GXPath^{pos} y se buscan subset repairs sabemos que hay un único candidato, y por lo tanto CQA se puede resolver evaluando la consulta sobre ese único repair.

	Subset Subset $w-$preferred Subset **multiset**-preferred	Superset
Reg-GXPath	NP-COMPLETE	Cota inferior EXPTIME
Reg-GXPath^{pos}	PTIME con solo expresiones de nodo NP-COMPLETE con expresiones de camino	NP-COMPLETE si R no está fijo y Σ_n es infinito PTIME en los otros casos

Tab. 6.1: Complejidad de los problemas de encontrar subset, subset preferred y superset repairs. No demostramos ninguna cota superior para el caso de superset usando cualquier expresión de Reg-GXPath. Estos resultados aplican también al problema **multiset-preferred** repair bound.

	Superset w-preferred	Superset **multiset**-preferred
Reg-GXPath^{pos}	PTIME con solo expresiones de nodo NP-COMPLETE con expresiones de camino	NP-COMPLETE si R no está fijo y Σ_n es infinito PTIME en cualquier otro caso

Tab. 6.2: Complejidad del problema de computar superset preferred repairs. Estos resultados aplican también al problema **multiset-preferred** repair bound.

7. COMENTARIOS Y PUBLICACIONES DERIVADAS

Parte de esta tesis fue escrita en el marco de las Pasantías de Investigación del DC del año 2020. María Vanina Martinez fue la Supervisora del proyecto, mientras que Edwin Pin Baque fue mi Mentor.

En el GLyC (Grupo de Lógica y Computabilidad) seguimos trabajando con estos temas. Algunas preguntas que quedan abiertas de esta tesis (como la cota superior para el problema de $\exists$SUPERSETREPAIR cuando $\mathcal{L} = $ Reg-GXPath) ya fueron resueltas, y estamos preparando un paper para publicar prontamente con estos nuevos resultados.

También estamos trabajando con algunas extensiones a modelos un poco distintos, con nociones probabilísticas. Como resultado de esto tenemos el paper [2], que está en vías de publicación en un journal internacional. En este se estudia una reversión del problema de repairing, en el cual el conjunto de repairs se define con una distribución de probabilidad $\mathcal{I}$, y se cuenta con una función $\mathcal{R}$ que asigna probabilidades a la evolución de un repair G a un data-graph "sucio" G'. El objetivo entonces es, a grandes rasgos, el de encontrar el repair G que maximiza $\mathcal{I}(G) \times \mathcal{R}(G)(G')$, donde G' es la base de datos observada. En el GLyC seguimos trabajando en la búsqueda de condiciones sobre $\mathcal{I}$ y $\mathcal{R}$ que permitan que el problema sea tratable y al mismo tiempo permita capturar casos prácticos de uso.

8. APÉNDICE

Las demostraciones de esta sección son en su mayoría por inducción estructural. Al principio de cada demostración hay un pequeño comentario que cuenta la idea de la inducción.

Prueba del lema 14. En esta demostración queremos probar esa intuición de que si $G \subseteq G'$ y $v \in [\![\alpha]\!]_G$ entonces $v \in [\![\alpha]\!]_{G'}$, donde $\alpha \in \text{Reg-GXPath}^{pos}$. Como ya se dijo antes, la idea es que un par de nodos (v, w) satisface una expresión positiva α si y solamente si hay efectivamente un camino que se puede trazar de v a w como testigo. Como $G \subseteq G'$ este camino también va a estar en G' si ya está en G.

Lo vamos a probar por inducción en la estructura de las expresiones. Notamos $G = (V_G, L, D)$ y $G' = (V_G, L', D')$.

En el caso base de una path expression α, esta debe ser de la forma ϵ, $\_$, A o A^-. En todos estos casos la propiedad vale: como $G \subseteq G'$, entonces tenemos que $\{(x,x) \mid x \in V_G\} \subseteq \{(x,x) \mid x \in V_{G'}\}$, que $\{(x,y) \mid x,y \in V_G \wedge L(x,y) \neq \emptyset\} \subseteq \{(x,y) \mid x,y \in V_{G'} \wedge L'(x,y) \neq \emptyset\}$, y de la misma forma para los otros dos casos.

Para el caso base de una node expression φ, este debe ser de la forma $\mathrm{c}^{=}$ o $\mathrm{c}^{\neq}$, y es fácil ver que la propiedad también vale.

Para el caso inductivo consideramos todos los casos:

- Si $\alpha = [\psi]$ entonces $[\![\psi]\!]_G \subseteq [\![\psi]\!]_{G'}$ (lo cual vale por HI) implica $[\![[\psi]]\!]_G \subseteq [\![[\psi]]\!]_{G'}$.

- Si $\alpha = \beta_1.\beta_2$ entonces $(v,w) \in [\![\alpha]\!]_G \iff \exists z$ tal que $(v,z) \in [\![\beta_1]\!]_G$ y $(z,w) \in [\![\beta_2]\!]_G$. Dado que $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ para $i \in \{1,2\}$, luego $(v,z) \in [\![\beta_1]\!]_{G'}$ y $(z,w) \in [\![\beta_2]\!]_{G'}$, lo cual implica que $(v,w) \in [\![\alpha]\!]_{G'}$.

- Si $\alpha = \beta_1 \cup \beta_2$ entonces $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ para $i \in \{1,2\}$ que rápidamente implica $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$.

- Si $\alpha = \beta^*$ entonces $(v,w) \in [\![\beta^*]\!]_G \iff \exists z_1, z_2, ..., z_m$ tal que $z_1 = v$, $z_m = w$ y $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ para $1 \leq i < m$. Dado que $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ entonces $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$ para $1 \leq i < m$, lo cual implica $(v,w) \in [\![\beta^*]\!]_{G'}$.

- Si $\alpha = \beta^{n,m}$ entonces $(v,w) \in [\![\beta^{n,m}]\!]_G \iff \exists z_1, ..., z_k, n+1 \leq k \leq m+1$ tal que $z_1 = v$, $z_k = w$ y $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ para $1 \leq i \leq k-1$. Por la HI entonces $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$, lo cual implica $(z,w) \in [\![\alpha^{n,m}]\!]_{G'}$.

- Si $\alpha = \beta_1 \cap \beta_2$ entonces $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$ para $i \in \{1,2\}$ implica $[\![\beta_1 \cap \beta_2]\!]_G = [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G \subseteq [\![\beta_1]\!]_{G'} \cap [\![\beta_2]\!]_{G'} = [\![\beta_1 \cap \beta_2]\!]_{G'}$.

- Si $\varphi = \psi_1 \wedge \psi_2$ entonces $[\![\psi_i]\!]_G \subseteq [\![\psi_i]\!]_{G'}$ para $i \in \{1,2\}$ lo cual implica $[\![\psi_1 \wedge \psi_2]\!]_G = [\![\psi_1]\!] \cap [\![\psi_2]\!] \subseteq [\![\psi_1]\!]_{G'} \cap [\![\psi_2]\!]_{G'} = [\![\psi_1 \wedge \psi_2]\!]_{G'}$

- El caso $\varphi = \psi_1 \vee \psi_2$ puede ser visto de la misma forma.

- Si $\varphi = \langle\beta\rangle$ entonces $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ implica $[\![\langle\beta\rangle]\!]_G \subseteq [\![\langle\beta\rangle]\!]_{G'}$

- Si $\varphi = \langle \beta_1 \star \beta_2 \rangle$, con $\star \in \{=, \neq\}$, entonces $v \in [\![\varphi]\!]_G \iff \exists z_1, z_2 \in V_G$ tal que $(v, z_i) \in [\![\beta_i]\!]_G$ para $i \in \{1, 2\}$ y $D(z_1) \star D(z_2)$. Dado que $[\![\beta_i]\!]_G \subseteq [\![\beta_i]\!]_{G'}$, entonces $(v, z_i) \in [\![\beta_i]\!]_{G'}$. Y como también tenemos que $D(z_i) = D'(z_i)$ (dado que los z_i están en V_G), obtenemos que $v \in [\![\langle \beta_1 \star \beta_2 \rangle]\!]_{G'}$, como queríamos.

□

Prueba del lema 21. Antes de empezar, definimos formalmente lo que llamamos una 'representación' de un *labeled data tree* con un data-graph. Dado un labeled data tree T definimos su representación $T' = (V_{T'}, L, D)$ de $T = (V_T, \sigma, \delta)$ como:

$$V_{T'} = \{v_{label} \mid \text{para } v \in V_T\} \cup \{v_{data} \mid \text{para } v \in V_T\}$$

$$L(v_{label}, v_{data}) = \{\text{VALUE}\}$$
$$L(v_{label}, w_{label}) = \{\text{DOWN}\} \text{ si y solamente si } w \text{ es un hijo de } v \text{ en } T$$
$$L(v, w) = \emptyset \text{ para todo otro par de nodos}$$

$$D(v_{label}) = \sigma(v)$$
$$D(v_{data}) = \delta(v)$$

Por cada nodo de T agregamos dos nodos: uno que va a tener el label y otro el data value, y un eje del nodo "label" al nodo "value". Luego agregamos ejes DOWN entre los nodos label formando una estructura isomorfa al árbol original. Cada subárbol enraizado en cualquier nodo que sea un hijo de v_r en la prueba del teorema 20 se va a corresponder con la representación de un labeled data tree gracias a las restricciones impuestas.

Ahora probamos la proposición mediante inducción estructural en ambas direcciones. Probamos que una node expression en **Downward XPath** es satisfecha en un nodo v de T si y solamente si $f(\varphi)$ es satisfecha en el nodo v_{label} de T', y que el par (v, w) pertenece a $[\![\alpha]\!]_T$ si y solamente si (v_{label}, w_{label}) pertenece a $[\![f(\alpha)]\!]_{T'}$. Recordemos que la función f esencialmente solo agrega la indirección necesaria para acceder al data value en vez de al label en las subexpresiones de comparación, como $\langle \alpha = \beta \rangle$.

Los casos base son:

- $\alpha = \epsilon \implies f(\alpha) = \epsilon$: la proposición vale trivialmente.
- $\alpha = \downarrow^* \implies f(\alpha) = \downarrow^*_{\text{DOWN}}$: si $(v, w) \in [\![\alpha]\!]_T$ entonces debe haber un camino entre v_{label} y w_{label} con ejes DOWN por construcción, lo cual implica que $(v, w) \in [\![f(\alpha)]\!]_{T'}$. La otra dirección se deduce también por construcción.
- $\varphi = a \implies f(\varphi) = [a^=]$: si $v \in [\![a]\!]_T$ entonces $D_{T'}(v_{label}) = a$ por construcción, lo cual implica que $r_{T'} \in [\![[a^=]]\!]_{T'}$. La otra dirección se deduce también por inducción.

Y ahora, los casos inductivos:

- $\alpha = \beta[\psi] \implies f(\alpha) = f(\beta)[f(\psi)]$: sea $(v, w) \in [\![\beta[\psi]]\!]_T$, lo cual implica que $(v, w) \in [\![\beta]\!]_T$ y $w \in [\![\psi]\!]_T$. Por HI entonces $(v_{label}, w_{label}) \in [\![f(\beta)]\!]_{T'}$ y $w_{label} \in [\![f(\psi)]\!]_{T'}$, y podemos deducir que $(v_{label}, w_{label}) \in [\![f(\beta)[\psi]]\!]_{T'}$. La otra dirección se deduce de la misma forma.
- $\alpha = [\psi]\beta \implies f(\alpha) = [f(\psi)]\beta$: idem el caso anterior.

- $\alpha = \beta_1.\beta_2 \implies f(\alpha) = f(\beta_1).f(\beta_2)$: si $(v,w) \in [\![\beta_1.\beta_2]\!]_T$ entonces existe $x \in V_T$ tal que $(v,x) \in [\![\beta_1]\!]_T$ y $(x,w) \in [\![\beta_2]\!]_T$. Luego, por HI $(v_{label}, x_{label}) \in [\![f(\beta)]\!]_{T'}$ y $(x_{label}, w_{label}) \in [\![f(\beta_2)]\!]_{T'}$, lo cual implica que $(v_{label}, w_{label}) \in [\![f(\beta_1).f(\beta_2)]\!]_{T'}$. La prueba para la otra dirección es idéntica.

- $\alpha = \beta_1 \cup \beta_2 \implies f(\alpha) = f(\beta_1) \cup f(\beta_2)$: si $(v,w) \in [\![\beta_1 \cup \beta_2]\!]_T$ entonces sin pérdida de generalidad podemos asumir que $(v,w) \in [\![\beta_1]\!]_T$, y el resultado se deduce directamente usando la HI.

- $\varphi = \neg\psi \implies f(\varphi) = \neg f(\psi)$: sea $v \in [\![\neg\psi]\!]_T$. Esto implica que $v \notin [\![\psi]\!]_T$, y por HI sabemos que $v_{label} \notin [\![f(\psi)]\!]_{T'}$, y por lo tanto concluímos que $v_{label} \in [\![\neg f(\psi)]\!]_{T'}$

- $\varphi = \psi_1 \wedge \psi_2 \implies f(\varphi) = f(\psi_1) \wedge f(\psi_2)$: ambas direcciones se deducen por inducción.

- $\varphi = \langle\beta\rangle \implies f(\varphi) = \langle f(\beta)\rangle$: si $v \in [\![\langle\beta\rangle]\!]_T$ entonces existe un nodo $x \in V_T$ tal que $(v,x) \in [\![\beta]\!]_T$. Luego, por HI sabemos que $(v_{label}, x_{label}) \in [\![f(\beta)]\!]_{T'}$, y entonces por definición $v_{label} \in [\![\langle\beta\rangle]\!]_{T'}$.

- $\varphi = \langle\beta_1 \star \beta_2\rangle \implies f(\varphi) = \langle\beta_1 \downarrow_{\text{VALUE}} \star \beta_2 \downarrow_{\text{VALUE}}\rangle$ $(\star \in \{=, \neq\})$: si $v \in [\![\langle\beta_1 \star \beta_2\rangle]\!]$ entonces existen nodos $x^1, x^2 \in V_T$ tal que $(v,x^1) \in [\![\beta_1]\!]_T$, $(v,x^2) \in [\![\beta_2]\!]_T$ y $\delta(x_1) \star \delta(x_2)$. Por HI sabemos que $(v_{label}, x^i_{label}) \in [\![f(\beta_i)]\!]_{T'}$ para $i \in \{1,2\}$. Por construcción sabemos que ambos x^i_{label} tienen ejes salientes a nodos x^i_{data} tales que $D(x^1_{data}) \star D(x^2_{data})$. Ergo podemos concluir que $v \in [\![\langle f(\beta_1) \downarrow_{\text{VALUE}} \star f(\beta_2) \downarrow_{\text{VALUE}}\rangle]\!]_{T'}$.

□

Prueba del lema 24. Para probar este lema tenemos que usar otra propiedad de las expresiones de Reg-GXPath^{pos} que es sumamente oscura:

Lema 51. *Sea α una expresión de camino de Reg-GXPath^{pos}, φ una expresión de nodo de Reg-GXPath^{pos}, $E \subseteq \Sigma_e$ un conjunto de edge labels y K un data-graph donde para cada par de nodos $v, w \in V_K$ es el caso que $L(v,w) = E$. Entonces vale que:*

1. *Si $(v,w) \in [\![\alpha]\!]_K$, $v \neq w$, y $D_K(w) \notin \Sigma^\alpha_n$, entonces vale que $(v,z) \in [\![\alpha]\!]_K$ para cada nodo $z \in V_K$ tal que $D_K(z) \notin \Sigma^\alpha_n$.*

2. *Si $(w,v) \in [\![\alpha]\!]_K$, $v \neq w$, y $D_K(w) \notin \Sigma^\alpha_n$, entonces vale que $(z,v) \in [\![\alpha]\!]_K$ para cada nodo $z \in V_K$ tal que $D_K(z) \notin \Sigma^\alpha_n$.*

3. *Si $(v,v) \in [\![\alpha]\!]_K$ y $D_K(v) \notin \Sigma^\alpha_n$ entonces $(z,z) \in [\![\alpha]\!]_K$ para cada nodo $z \in V_K$ tal que $D_K(z) \notin \Sigma^\alpha_n$.*

4. *Si $v \in [\![\varphi]\!]_K$ para algún $v \in V_K$ tal que $D_K(v) \notin \Sigma^\varphi_n$ entonces $z \in [\![\varphi]\!]_K$ para todo nodo $z \in V_K$ que satisface $D_K(z) \notin \Sigma^\varphi_n$.*

Este lema dice fundamentalmente lo siguiente: si un nodo v satisface la formula φ y su data value no está mencionado en φ, entonces cualquier nodo que no tenga un data value en φ va a satisfacerla. Esto ocurre porque desde cualquier nodo los caminos son alcanzables (el grafo es completo sobre un conjunto de edge labels) y porque el data value de v es "indistinguible" del resto de data values no mencionados en φ. Para las path expressions hay que hacer un enunciado más complicado, pero la idea es la misma.

Prueba del lema 51. Presentamos una prueba de las cuatro propiedades en simultáneo usando inducción estructural. Para los casos base:

- $\alpha = \_$: si $E \neq \emptyset$ entonces todo par $v, z \in V_K$ está en $[\![\_]\!]_K$, y este caso se deduce trivialmente. Caso contrario no hay ejes en el grafo, y no hay ninguna hipótesis que se satisfaga.
- $\alpha = \text{A}$ y $\alpha = \text{A}^-$: si hay un par de nodos $v, w \in V_K$ tal que $(v, w) \in [\![\text{A}]\!]_K$ entonces $\text{A} \in E$, y cada par de nodos $x, y \in V_K$ satisface $(x, y) \in [\![\text{A}]\!]_K$. El mismo razonamiento vale para A^-.
- $\alpha = \epsilon$: este caso se deduce trivialmente, dado que las hipótesis de 1 y 2 no pueden ser satisfechas, y el consecuente de 3 siempre se satisface.
- $\varphi = \text{E}^=$: la hipótesis de 4 no puede ser satisfecha en este caso, y entonces vale trivialmente.
- $\varphi = \text{E}^{\neq}$: cada nodo $v \in V_K$ tal que $D_K(v) \notin \Sigma_n^{\text{E}^{\neq}} = \{\text{E}\}$ satisface $v \in [\![\text{E}^{\neq}]\!]_K$.

Ahora procedemos con los casos inductivos:

- $\alpha = [\psi]$: las hipótesis de las proposiciones 1 y 2 no pueden ser satisfechas en este caso, y entonces solo demostramos la proposición 3. Supongamos que hay un nodo $v \in V_K$ tal que $D_K(v) \notin \Sigma_n^{[\psi]}$ y $(v, v) \in [\![[\psi]]\!]_K$. Por hipótesis cualquier otro nodo $z \in V_K$ tal que $D_K(z) \notin \Sigma_n^{\psi}$ satisface $z \in [\![\psi]\!]_K$, lo cual entonces implica que $(z, z) \in [\![[\psi]]\!]_K$.
- $\alpha = \beta_1.\beta_2$: para la proposición 1, sea $v, w \in V_K$ un par de nodos que satisface su hipótesis ($(v, w) \in [\![\alpha]\!]_K$, $v \neq w$, $D(w) \notin \Sigma_n^{\alpha}$), y sea x un nodo tal que $(v, x) \in [\![\beta_1]\!]_K$ y $(x, w) \in [\![\beta_2]\!]_K$. Luego, si $x \neq w$ sabemos por HI que $(x, z) \in [\![\beta_2]\!]_K$ para cada $z \in V_K$ con $D_K(z) \notin \Sigma_n^{\beta_2}$ (que es un conjunto que contiene a $\Sigma_n^{\beta_1.\beta_2}$), y entonces podemos concluir que $(v, z) \in [\![\beta_1.\beta_2]\!]_K$. Si $w = x$ el resultado se deduce considerando que $(v, z) \in [\![\beta_1]\!]_K$ y entonces $(z, z) \in [\![\beta_2]\!]_K$ por la proposición 3. La proposición 2 se demuestra de la misma forma.

  Ahora, para la proposición 3 sea v un nodo de K que satisface la hipótesis ($(v, v) \in [\![\alpha]\!]_K$ y $D_K(v) \notin \Sigma_n^{\alpha}$) y sea x el nodo 'intermedio'. Sea z un nodo tal que $D_K(z) \notin \Sigma_n^{\beta_1.\beta_2}$. Si $x = v$ se deduce usando la proposición 3 como HI que $(z, z) \in [\![\beta_1.\beta_2]\!]_K$. Caso contrario (i.e. $x \neq v$) usando la proposición 1 y 2 como HI obtenemos el mismo resultado.
- $\alpha = \beta_1 \cup \beta_2$: este caso se deduce por HI considerando, sin pérdida de generalidad, que $(v, w) \in [\![\beta_1]\!]_K$ (y respectivamente (w, v) y (v, v) para las proposiciones 2 y 3).
- El caso $\alpha = \beta_1 \cap \beta_2$ puede ser tratado de la misma forma.
- $\alpha = \beta^*$: si $(v, w) \in [\![\beta^*]\!]_K$ entonces o bien $(v, w) \in [\![\epsilon]\!]_K$ o $(v, w) \in [\![\beta^n]\!]_K$ para algún $n > 0$. El primer escenario ya fue demostrado, mientras que el segundo se deduce de lo que observamos en el caso de la concatenación.
- $\alpha = \beta^{n,m}$: el resultado se deduce con el mismo argumento usado para el caso β^* dado que $(v, w) \in [\![\beta^{n,m}]\!] \iff (v, w) \in [\![\beta^k]\!]$ para algún $n \leq k \leq m$.

- $\varphi = \psi_1 \wedge \psi_2$: sea $v \in V_K$ un nodo tal que $v \in [\![\varphi]\!]_K$ y $D_K(v) \notin \Sigma_n^\varphi$. Por HI cualquier otro nodo $z \in V_K$ con $D_K(z) \notin \varphi$ satisface $z \in [\![\psi_i]\!]_K$ para $i \in \{1, 2\}$, lo cual implica que $z \in [\![\varphi]\!]_K$.

- $\varphi = \psi_1 \vee \psi_2$: sea $v \in V_K$ un nodo que satisface la hipótesis de la proposición 4, y sin pérdida de generalidad asumamos que $v \in [\![\psi_1]\!]_K$. Por HI cualquier otro nodo $z \in V_K$ tal que $D_K(z) \notin \Sigma_n^\varphi$ satisface $z \in [\![\psi_1]\!]_K$, lo cual implica que $z \in [\![\psi_1 \vee \psi_2]\!]_K$.

- $\varphi = \langle\beta\rangle$: sea $v \in V_K$ un nodo que satisface la hipótesis de la proposición 4. Luego $(v, x) \in [\![\beta]\!]_K$ para algún x. Gracias a la proposición 2 podemos deducir que si $x \neq v$ entonces $(z, x) \in [\![\beta]\!]_K$ para cada z tal que $D_K(z) \notin \Sigma_n^{\langle\beta\rangle}$. Si $x = v$ usamos la proposición 3.

- $\varphi = \langle\beta_1 = \beta_2\rangle$: sea $v \in V_K$ un nodo que satisface la hipótesis de la proposición 4, x_1 y x_2 nodos tales que $(v, x_i) \in [\![\beta_i]\!]_K$ para $i \in \{1, 2\}$ y $D_K(x_1) = D_K(x_2)$, y $z \in V_K$ otro nodo tal que $D_K(z) \notin \Sigma_n^\varphi$. Si ambos x_i son diferentes de v entonces $(z, x_i) \in [\![\beta_i]\!]_K$ gracias a la proposición 2. Caso contrario, supongamos $x_1 = v$ sin pérdida de generalidad, lo cual entonces implica que $(z, z) \in [\![\beta_1]\!]_K$ debido a la proposición 3. Como $D_K(x_2) = D_K(x_1) = D_K(v) \notin \Sigma_n^\varphi$ podemos garantizar que $(z, z) \in [\![\beta_2]\!]_K$: si $x_2 = v$ entonces esto se deduce de la proposición 3, y si $x_2 \neq v$ entonces en base a la proposición 2 sabemos que $(x_2, x_2) \in [\![\beta_2]\!]_K$ y entonces que $(z, z) \in [\![\beta_2]\!]_K$ a través de la proposición 3.

- $\varphi = \langle\beta_1 \neq \beta_2\rangle$: sea $v \in V_K$ un nodo que satisface la hipótesis de la proposición 4, x_1 y x_2 nodos tal que $(v, x_i) \in [\![\beta_i]\!]_K$ para $i \in \{1, 2\}$ y $D_K(x_1) \neq D_K(x_2)$, y $z \in V_K$ otro nodo tal que $D_K(z) \notin \Sigma_n^\varphi$. Si $v \neq x_i$ para $i \in \{1, 2\}$ entonces el resultado se deduce por la proposición 2. Caso contrario consideremos que $x_1 = v$, lo cual entonces implica que $(z, z) \in [\![\beta_1]\!]_K$ por la proposición 3. Si $x_2 \neq v$ entonces por la proposición 2 podemos concluir que $(z, x_2) \in [\![\beta_2]\!]_K$. Si $D_K(x_2) \neq D_K(z)$ terminamos, y si no, sabiendo que $D_K(x_2) \notin \Sigma_n^\varphi$ (esto se deduce de $D_K(x_2) = D_K(z)$), podemos considerar que $(z, v) \in [\![\beta_2]\!]_K$ por la proposición 1 y que $D_K(x_2) \neq D_K(x_1) = D_K(v)$. El caso $x_2 = v$ no puede ocurrir, dado que asumimos $x_1 = v$ y $D_K(x_1) \neq D_K(x_2)$.

□

Ahora continuamos con el lema 24, donde vemos que, dado un repair G' de G, podemos tomar todos los nodos con datos que no estaban presentes en G ni R y mandarlos a solo dos data values distintos, manteniendo la satisfacibilidad de R (siempre y cuando agreguemos algunos ejes). La idea es que al agregar todos los ejes posibles los caminos se mantienen (e incluso aparecen nuevos), y que, apoyados en el lema recién demostrado, cambiar los datos que no aparecían en la fórmula no afectan al conjunto de nodos que satisfacen la fórmula. Esto es porque las únicas subexpresiones que se pueden romper son las de la forma $\langle\alpha \neq \beta\rangle$, pero a estas las salvamos al dejar dos datos distintos y permitir todos los caminos posibles que antes eran desde cualquier nodo.

Sea G' un superset repair de G con respecto a R. Si $\Sigma_n^{G'} \subseteq \Sigma_n^G \cup \Sigma_n^R \cup \{c, d\}$ para algunos $c, d \in \Sigma_n \setminus \Sigma_n^R$, $c \neq d$, entonces G' ya es el testigo que queremos para este lema. Caso contrario, tomemos $c, d \in \Sigma_n^{G'} \setminus (\Sigma_n^G \cup \Sigma_n^R)$, $c \neq d$. Definimos el data-graph $H = (V_H, L_H, D_H)$ donde:

$$V_H = V_{G'}$$

$$L_H(v,w) = \Sigma_e^{G'} \quad \forall v, w \in V_H$$

$$D_H(v) = \begin{cases} D_{G'}(v) & \text{si } D_{G'}(v) \in \Sigma_n^G \cup \Sigma_n^R \cup \{c,d\} \\ c & \text{caso contrario} \end{cases}$$

donde $\Sigma_e^{G'}$ es el conjunto de todos los edge labels mencionados en G' (i.e. $\Sigma_e^{G'} = \{l \in \Sigma_e \mid \exists v, w \in V_{G'} \text{ tal que } l \in L_{G'}(v,w)\}$).

Intuitivamente cambiamos todos los data values que no estaban presentes en $\Sigma_n^G \cup \Sigma_n^R \cup \{c,d\}$ por c, y también agregamos cualquier eje posible considerando aquellos labels ya presentes en G'. Notemos que $\Sigma_n^H \subseteq \Sigma_n^G \cup \Sigma_n^R \cup \{c,d\}$, y por lo tanto mostrar que $H \models R$ es suficiente para probar el lemma. Probaremos esto por inducción en la estructura de la fórmula, de tal forma que si un par $v, w \in V_{G'}$ satisface $(v,w) \in [\![\alpha]\!]_{G'}$ entonces $(v,w) \in [\![\alpha]\!]_H$, y tal que si $v \in [\![\varphi]\!]_{G'}$ entonces $v \in [\![\varphi]\!]_H$ (donde las expresiones consideradas solo usan data values de Σ_n^R).

Para los casos base:

- $\alpha = \_$: Todo par $v, w \in V_H$ está en $[\![\_]\!]_H$, por lo que este caso vale trivialmente.
- $\alpha = \text{A}$ y $\alpha = \text{A}^-$: si $(v,w) \in [\![\text{A}]\!]_{G'}$ entonces claramente $\text{A} \in \Sigma_e^{G'}$ y $\text{A} \in L_H(v,w)$.
- $\alpha = \epsilon$: vale trivialmente.
- $\varphi = \text{E}^=$: si $v \in [\![\text{E}^=]\!]_{G'}$ entonces $D_{G'}(v) = \text{E}$. dado que $\text{E} \in \Sigma_n^R$ el data value fue preservado en H ($D_H(v) = \text{E}$) y por lo tanto $v \in [\![\text{E}^=]\!]_H$.
- $\varphi = \text{E}^{\neq}$: sea $v \in [\![\text{E}^{\neq}]\!]_{G'}$. Si $D_{G'}(v) \in \Sigma_n^G \cup \Sigma_n^R\{c,d\}$ entonces el data value de v fue preservado, y $v \in [\![\text{E}^{\neq}]\!]_H$. Caso contrario $D_H(v) = \text{C}$, y dado que $\text{C} \notin \Sigma_n^R$ sabemos que $v \in [\![\text{E}^{\neq}]\!]_H$.

Ahora, para los casos inductivos:

- $\alpha = [\psi]$: sea $(v,v) \in [\![[\psi]]\!]_{G'}$, lo cual implica que $v \in [\![\psi]\!]_{G'}$. Por HI tenemos que $v \in [\![\psi]\!]_H$, y entonces $(v,v) \in [\![[\psi]]\!]_H$.
- $\alpha = \beta_1.\beta_2$: si $(v,w) \in [\![\beta_1.\beta_2]\!]_{G'}$ entonces existe $z \in V_{G'}$ tal que $(v,z) \in [\![\beta_1]\!]_{G'}$ y $(z,w) \in [\![\beta_2]\!]_{G'}$. Por HI $(v,z) \in [\![\beta_1]\!]_H$ y $(z,w) \in [\![\beta_2]\!]_H$, lo cual implica que $(v,w) \in [\![\beta_1.\beta_2]\!]_H$.
- $\alpha = \beta_1 \cup \beta_2$: sea $(v,w) \in [\![\beta_1 \cup \beta_2]\!]_{G'}$. Sin pérdida de generalidad podemos asumir que $(v,w) \in [\![\beta_1]\!]_{G'}$. Por HI $(v,w) \in [\![\beta_1]\!]_H$, y entonces $(v,w) \in [\![\beta_1 \cup \beta_2]\!]_H$.
- $\alpha = \beta_1 \cap \beta_2$: dado que $(v,w) \in [\![\beta_1 \cap \beta_2]\!]_G \iff (v,w) \in [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G$ entonces la HI implica que $(v,w) \in [\![\beta_1]\!]_H \cap [\![\beta_2]\!]_H = [\![\beta_1 \cap \beta_2]\!]_H$.
- $\alpha = \beta^*$: $(v,w) \in [\![\beta^*]\!]_{G'}$ si existe $z_1, ...z_m \in V_{G'}$ tal que $z_1 = v$, $z_m = w$ y $(z_i, z_{i+1}) \in [\![\beta]\!]_{G'}$ para $1 \leq i \leq m-1$. Por HI $(z_i, z_{i+1}) \in [\![\beta]\!]_H$ para $1 \leq i \leq m-1$, y entonces $(v,w) \in [\![\beta^*]\!]_H$.

- El caso $\alpha = \beta^{n,m}$ se deduce del mismo argumento usado para el caso $.^*$.

- $\varphi = \psi_1 \wedge \psi_2$: sea $v \in [\![\psi_1 \wedge \psi_2]\!]_{G'}$. Dado que $v \in [\![\psi_i]\!]_{G'}$ para $i \in \{1,2\}$ sabemos por HI que $v \in [\![\psi_i]\!]_H$ para $i \in \{1,2\}$, lo cual implica que $v \in [\![\psi_1 \wedge \psi_2]\!]_H$.

- $\varphi = \psi_1 \vee \psi_2$: podemos asumir sin pérdida de generalidad que si $v \in [\![\psi_1 \vee \psi_2]\!]_G$ entonces $v \in [\![\psi_1]\!]_G$, y podemos rápidamente concluir por HI que $v \in [\![\psi_1]\!]_H \subseteq [\![\psi_1 \vee \psi_2]\!]_H$.

- $\varphi = \langle\beta\rangle$: si $v \in [\![\langle\beta\rangle]\!]_{G'}$ entonces existe $z \in V_{G'}$ tal que $(v,z) \in [\![\beta]\!]_{G'}$. Luego por hipótesis $(v,z) \in [\![\beta]\!]_H$ y $v \in [\![\langle\beta\rangle]\!]_H$.

- $\varphi = \langle\beta_1 = \beta_2\rangle$: si $v \in [\![\langle\beta_1 = \beta_2\rangle]\!]_{G'}$ entonces existe $z_1, z_2 \in V_{G'}$ tal que $(v,z_1) \in [\![\beta_1]\!]_{G'}$, $(v,z_2) \in [\![\beta_2]\!]_{G'}$ y $D_{G'}(z_1) = D_{G'}(z_2)$. Por construcción $D_H(z_1) = D_H(z_2)$, y por hipótesis $(v,z_1) \in [\![\beta_1]\!]_H$ y $(v,z_2) \in [\![\beta_2]\!]_H$, lo cual implica que $v \in [\![\langle\beta_1 = \beta_2\rangle]\!]_H$.

- $\varphi = \langle\beta_1 \neq \beta_2\rangle$: este es el caso más interesante. Notemos que los caminos usados en G para satisfacer la expresión ahora podrían llevar a nodos con el mismo data value. Si $v \in [\![\langle\beta_1 \neq \beta_2\rangle]\!]_{G'}$ entonces existen nodos $z_1, z_2 \in V_{G'}$ tal que $(v,z_1) \in [\![\beta_1]\!]_{G'}$, $(v,z_2) \in [\![\beta_2]\!]_{G'}$ y $D_{G'}(z_1) \neq D_{G'}(z_2)$. Si $D_H(z_1) \neq D_H(z_2)$ entonces podemos concluir usando la HI que $v \in [\![\langle\beta_1 \neq \beta_2\rangle]\!]_H$. Si este no es el caso, entonces ambos nodos tenían en G' como data value o bien c o bien uno de aquellos que fueron cambiados (i.e. uno de $\Sigma_n^{G'} \setminus (\Sigma_n^G \cup \Sigma_n^R \cup \{c,d\})$. Notemos que como $D_{G'}(z_1) \neq D_{G'}(z_2)$, debe valer que $z_1 \neq v$ o bien $z_2 \neq v$; sin pérdida de generalidad podemos asumir que $z_1 \neq v$. Sea v_d un nodo de H que tiene como data value d (por construcción tiene que haber al menos un nodo con este data value). Dado que $(v,z_1) \in [\![\beta_1]\!]_H$, $v \neq z_1$, $D_H(z_1) \notin \Sigma_n^R$ y H es un 'grafo completo' bajo los ejes de $\Sigma_e^{G'}$ podemos deducir usando la proposición 1 del lema 51 que $(v,v_d) \in [\![\beta_1]\!]_H$. Luego podemos concluir que $v \in [\![\langle\beta_1 \neq \beta_2\rangle]\!]_H$ teniendo en cuenta que $(v,z_2) \in [\![\beta_2]\!]_H$ debido a la HI.

Podemos entonces concluir, dado que $G' \models R$, que $H \models R$.

□

Prueba del lema 26. En este lema queremos ver que contraer los nodos con mismo dato no afecta la satisfacibilidad de las expresiones. Esta prueba aprovecha algo troncal de las expresiones de Reg-GXPath^{pos}: no pueden distinguir un nodo de otro si tienen el mismo dato y permiten recorrer los mismos caminos.

Sea $f : V_G \to V_H$ un mapeo entre los nodos de G y H tal que $f(v) = v$ si $v \in V_G \setminus V_d$, y $f(v) = v_d$ caso contrario. Ahora mostramos, por inducción en la estructura de una fórmula arbitraria, que si un par de nodos $v, w \in V_G$ satisface $(v,w) \in [\![\alpha]\!]_G$, entonces $(f(v), f(w)) \in [\![\alpha]\!]_H$, y que si $v \in [\![\varphi]\!]_G$ entonces $f(v) \in [\![\varphi]\!]_H$. Esto es suficiente para probar el lema.

Para los casos base:

- $\alpha = \_$: si ambos v y w no estuvieran en V_d entonces sus ejes serían preservados. Si $w \in V_d$ y $v \notin V_d$ entonces por construcción $L_G(v,w) \subseteq L_H(v,v_d)$. El caso cuando $v \in V_d$ es análogo. Si ambos v y w están en V_d usamos el hecho de que $L_G(v,w) \subseteq L_H(v_d,v_d)$.

- $\alpha = \text{A}$ o $\alpha = \text{A}^-$: idem el caso anterior.

- $\alpha = \epsilon$: este caso se deduce trivialmente.
- $\varphi = \mathrm{c}^{=}$: v y $f(v)$ tienen el mismo data value, por lo que este caso vale.
- $\varphi = \mathrm{c}^{\neq}$: idem el caso anterior.

Ahora, los casos inductivos:

- $\alpha = [\psi]$: Si $(v, v) \in [\![[\psi]]\!]_G$ entonces $v \in [\![\psi]\!]_G$, lo cual implica por HI que $f(v) \in [\![\psi]\!]_H$ y entonces $(f(v), f(v)) \in [\![[\psi]]\!]_H$.
- $\alpha = \beta_1.\beta_2$: si $(v, w) \in [\![\beta_1.\beta_2]\!]_G$ entonces existe $z \in V_G$ tal que $(v, z) \in [\![\beta_1]\!]_G$ y $(z, w) \in [\![\beta_2]\!]_G$. Luego por HI $(f(v), f(z)) \in [\![\beta_1]\!]_H$ y $(f(z), f(w)) \in [\![\beta_2]\!]_H$, lo cual implica que $(f(v), f(w)) \in [\![\beta_1.\beta_2]\!]_H$.
- $\alpha = \beta_1 \cup \beta_2$: supongamos sin pérdida de generalidad que $(v, w) \in [\![\beta_1]\!]_G$. Luego por inducción $(f(v), f(w)) \in [\![\beta_1 \cup \beta_2]\!]_H$.
- $\alpha = \beta_1 \cap \beta_2$: si $(v, w) \in [\![\beta_1 \cap \beta_2]\!]_G$ entonces $(v, w) \in [\![\beta_1]\!]_G \cap [\![\beta_2]\!]_G$, y por HI podemos deducir que $(f(v), f(w)) \in [\![\beta_1]\!]_H \cap [\![\beta_2]\!]_H = [\![\beta_1 \cap \beta_2]\!]_H$.
- $\alpha = \beta^*$: $(v, w) \in [\![\beta^*]\!]_G$ si existe $z_1, ..., z_m \in V_G$ tal que $z_1 = v$, $z_m = w$ y $(z_i, z_{i+1}) \in [\![\beta]\!]_G$ para $1 \leq i \leq m-1$. Por hipótesis se sigue entonces que $(f(z_i), f(z_{i+1})) \in [\![\beta]\!]_H$, lo cual implica que $(f(v), f(w)) \in [\![\beta^*]\!]_H$.
- $\alpha = \beta^{n,m}$: este caso se deduce usando los mismos argumentos que para el caso $.^*$.
- $\varphi = \psi_1 \wedge \psi_2$: $v \in [\![\psi_1 \wedge \psi_2]\!]_G$ si $v \in [\![\psi_1]\!]_G$ y $v \in [\![\psi_2]\!]_G$. Usando la HI concluímos que $f(v) \in [\![\psi_1]\!]_H$ y $f(v) \in [\![\psi_2]\!]_H$, lo cual implica que $f(v) \in [\![\psi_1 \wedge \psi_2]\!]_H$.
- $\varphi = \psi_1 \vee \psi_2$: si $v \in [\![\psi_1 \vee \psi_2]\!]_G$ entonces sin pérdida de generalidad podemos asumir que $v \in [\![\psi_1]\!]_G$. Luego, por HI $f(v) \in [\![\psi_1]\!]_H \subseteq [\![\psi_1 \vee \psi_2]\!]_H$.
- $\varphi = \langle\beta\rangle$: $v \in [\![\langle\beta\rangle]\!]_G$ si $(v, z) \in [\![\beta]\!]_G$ para algún z. Por hipótesis $(f(v), f(z)) \in [\![\beta]\!]_H$, y entonces $f(v) \in [\![\langle\beta\rangle]\!]_H$.
- $\varphi = \langle\beta_1 \star \beta_2\rangle$, con $\star \in \{=, \neq\}$: $v \in [\![\langle\beta_1 \star \beta_2\rangle]\!]$ si existe $z_1, z_2 \in V_G$ tal que $(v, z_1) \in [\![\beta_1]\!]_G$, $(v, z_2) \in [\![\beta_2]\!]_G$ y $D_G(z_1) \star D_G(z_2)$. En H ambos z_1 y z_2 preservan sus data values (i.e. $D_G(z_i) = D_H(f(z_i))$ para $i \in \{1, 2\}$), y por HI $(f(v), f(z_i)) \in [\![\beta_i]\!]_H$ para $i \in \{1, 2\}$. Por lo tanto $f(v) \in [\![\langle\beta_1 \star \beta_2\rangle]\!]_H$.

□

Bibliografía

[1] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[2] Sergio Abriola, Santiago Cifuentes, María Vanina Martínez, Nina Pardal, and Edwin Pin. An epistemic approach to model uncertainty in data-graphs. *arXiv preprint arXiv:2109.14112*, 2021.

[3] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, volume 10, pages 287–298, 2010.

[4] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

[5] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, volume 99, pages 68–79. Citeseer, 1999.

[6] Marcelo Arenas and Jorge Pérez. Querying semantic web data with sparql. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316, 2011.

[7] Pablo Barceló and Gaëlle Fontaine. On the data complexity of consistent query answering over graph databases. *Journal of Computer and System Sciences*, 88:164–194, 2017.

[8] Pablo Barceló, Jorge Pérez, and Juan L Reutter. Relative expressiveness of nested regular expressions. *AMW*, 12:180–195, 2012.

[9] Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 175–188. ACM, 2013.

[10] Michael Benedikt and Christoph Koch. XPath leashed. *ACM Computing Surveys (CSUR)*, 41(1):1–54, 2009.

[11] Leopoldo Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76, 2006.

[12] Leopoldo Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.

[13] Leopoldo Bertossi. Database repairs and consistent query answering: Origins and further developments. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 48–58, 2019.

[14] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[15] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000.

[16] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 149–158, 1998.

[17] Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Containment of regular path queries under description logic constraints. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[18] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, August 1979.

[19] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21, 2012.

[20] Wenfei Fan. Dependencies for graphs: Challenges and opportunities. *Journal of Data and Information Quality (JDIQ)*, 11(2):1–12, 2019.

[21] Diego Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic (TOCL)*, 13(4):1–40, 2012.

[22] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Preferred database repairs under aggregate constraints. In *International Conference on Scalable Uncertainty Management*, pages 215–229. Springer, 2007.

[23] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320*, 2020.

[24] Gérard Huet and Derek C Oppen. Equations and rewrite rules: A survey. In *Formal Language Theory*, pages 349–405. Elsevier, 1980.

[25] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[26] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *Journal of the ACM (JACM)*, 63(2):1–53, 2016.

[27] Leonid Libkin and Domagoj Vrgoč. Regular path queries on graphs with data. In *Proceedings of the 15th International Conference on Database Theory*, pages 74–85, 2012.

[28] Andrei Lopatenko and Leopoldo Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *In ICDT*, pages 179–193. Springer, 2007.

[29] Thomas Lukasiewicz, Enrico Malizia, and Andrius Vaicenavičius. Complexity of inconsistency-tolerant query answering in datalog+/–under cardinality-based repairs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2962–2969, 2019.

[30] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I Simari. From classical to consistent query answering under existential rules. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[31] Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I Simari. Complexity of inconsistency-tolerant query answering in datalog+/–. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, pages 488–500. Springer, 2013.

[32] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nsparql: A navigational language for rdf. *Journal of Web Semantics*, 8(4):255–270, 2010.

[33] Thomas Rebele, Fabian Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *International semantic web conference*, pages 177–185. Springer, 2016.

[34] Sławek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence*, 64(2):209–246, 2012.

[35] Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 22–33, 2012.

[36] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

[37] Jef Wijsen. Condensed representation of database repairs for consistent query answering. In *Proceedings of the 9th International Conference on Database Theory*, ICDT '03, pages 378–393. Springer-Verlag, 2002.

[38] Jef Wijsen. Database repairing using updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768, 2005.