mediKanren: A System for Bio-medical Reasoning

ANONYMOUS AUTHOR(S)

We introduce *mediKanren*, a combination of miniKanren, a database describing relationships between medical concepts, and a graphical user interface (GUI) to simplify data exploration and common queries. All features of the faster-miniKanren implementation are available for queries, including typical constructs like `run`, `conde`, `fresh`, various constraints, and (potentially recursive) user-defined relations. We provide the database as a set of miniKanren relations. To make queries fast, we represent the data backing these relations as specially formatted files on disk, with indexes for fast retrieval. We use the miniKanren `project` syntax to write Racket code that interfaces with this representation. Performance is sufficient for the GUI to support low latency querying in the common case.

In this paper we provide an overview of mediKanren, give examples of queries in low-level and medium-level query languages, and give an example of how query results have resulted in treatment for a patient with a rare genetic disease.

Additional Key Words and Phrases: relational programming, miniKanren, Racket, Scheme

ACM Reference Format:
Anonymous Author(s). 2020. mediKanren: A System for Bio-medical Reasoning. 1, 1 (May 2020), 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With over 1.5 million publications per year and more than 50 million total peer-reviewed articles, the rate and volume of novel discoveries has surpassed our ability to fully utilize and understand what is known [Jinha 2010]. This problem, described as the unknown known, is characterized by two specific features: 1) "forgotten facts"—facts that are published but not widely known 2) "uninferred facts"—facts that have not been deduced from existing published research. While all professional fields are subject to the effects of the "forgotten" or "uninferred" facts—the cost of the unknown known for healthcare providers is measured in human lives.

To tackle the issue of the uninferred medical knowledge, the University of Alabama at Birmingham Precision Medicine Institute (UAB-PMI) has developed an artificial intelligence software tool called *mediKanren* as a part of a multi-institutional grant funded by the National Center for Advancing Translational Science (NCATS). Since the tool's inception, mediKanren has been successful in finding novel FDA approved therapeutic recommendations for disorders ranging from undiagnosed and purely symptomatic disease to genetically diagnosed metabolic disorders [Ross C 2019; Shepard 2019].

2 BACKGROUND

We assume the reader is familiar with the basics of miniKanren and Racket. For readers unfamiliar with miniKanren, the language is described in [Friedman et al. 2018] and [Byrd 2009]. Racket is described in [Felleisen et al. [n.d.]].

2.1 Introduction to Knowledge Graphs

Several key concepts are required for understanding how mediKanren interacts with biomedical data sources to return a result to the user: 1) CURIEs 2) Concept Normalization 3) Knowledge Graphs (KGs). Compact Uniform Resource Identifiers (CURIEs) serve as machine readable markers for different databases [W3C 2010]. The process of gathering different CURIEs for a single concept, known as concept normalization, is critical for finding all identifiers for that concept across all publicly available databases. Finally, knowledge graphs provide a unified way of organizing relations between concepts that exist in disparate locations and have been utilized in a variety

2020. XXXX-XXXX/2020/5-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

of fields ranging from medicine to biodiversity and ecology [RDM 2019; Rotmensch M 2017]. As a part of a collaboration between UAB Precision Medicine Institute and the NCATS Data Translator consortium, several knowledge graphs have been indexed into mediKanren. Figure 1 below shows mediKanren statistics for concept to concept relations (ie query types) commonly used by UAB-PMI.

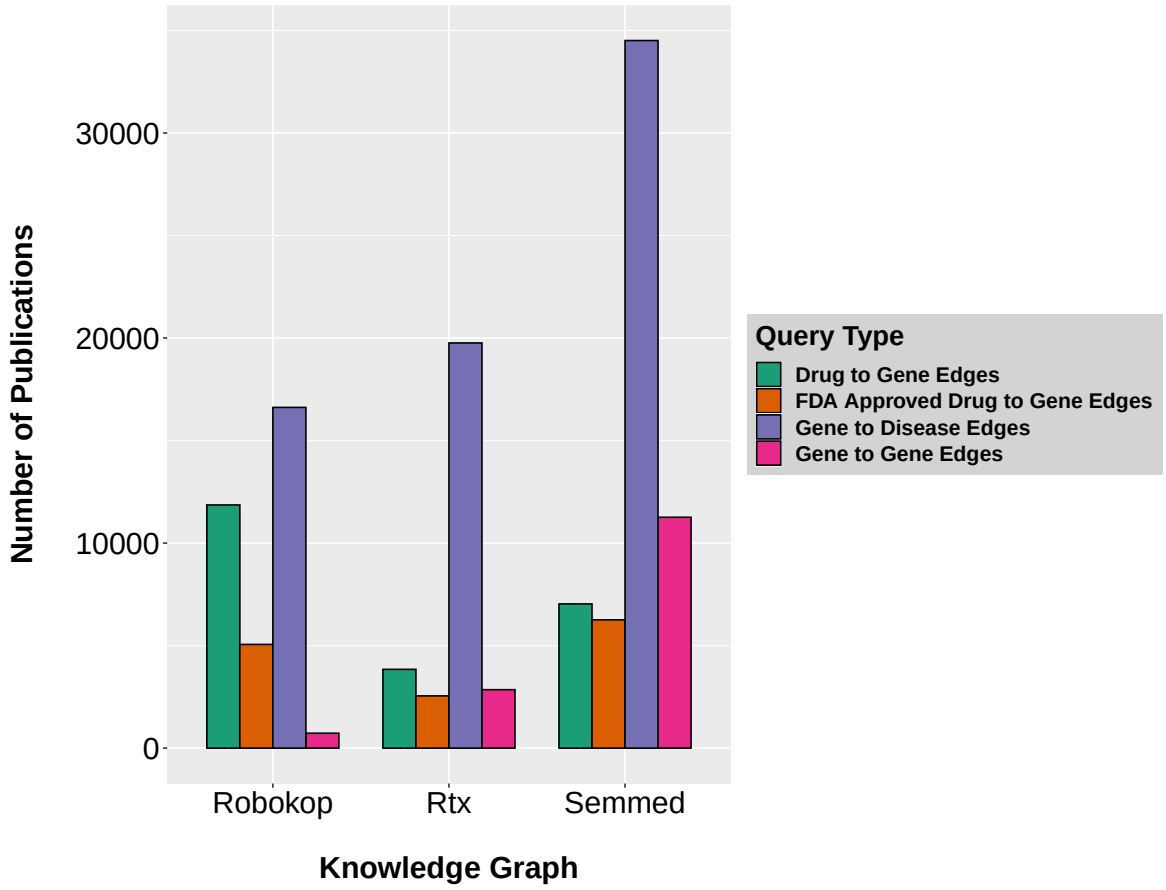

Fig. 1. mediKanren metrics shows the number of distinct edges supported by pubmed cited literature (x-axis) for 4 common query types: Drug to Gene, Gene to Disease, Gene to Gene and FDA approved Drug to Gene. Results are categorically separated (y-axis) by specific knowledge graph.

Figures 2 and 3 outline the mapped edges contained within the RTX and Robokop KGs.

3 GRAPH DATABASE

Knowledge graphs consist of medical concepts, represented as graph vertices, and relationships between them, represented as directed graph edges. Edges between concepts have a subject-verb-object structure, as in the Resource Description Framework (RDF) data model. Both concepts and edges are attributed. Edge attributes include metadata about provenance and evidence supporting the relationship claim.

We use Racket to implement a simple graph database using a mixture of files containing either plain text s-expressions or data in a custom binary format. These files represent the underlying data, as well as indices supporting fast lookup needed for common queries over the graph structure. Aside from indices for graph structure lookup, we also include indices for full-text search over concept names, supporting responsive lookup in our GUI. In order to run mediKanren on underpowered laptops with little memory, we choose to perform most lookups directly from disk. The simplicity of this database implementation has allowed us to quickly prototype new features, but trades off some performance and compact data encoding. Now that we have enough experience to know which features we need to support, we have recently started a new database implementation that is less naive, to reclaim some lost performance and data compactness.

We expose concept and edge information to a miniKanren program by defining the relations `concepto` and `edgeo`. These relations are defined using miniKanren's `project` construct, which allows us to implement them with Racket code that accesses our database representation, choosing an appropriate index to use based on groundness of different portions of the concept or edge argument.

4 KNOWLEDGE GRAPHS

To improve interoperability, the knowledge graphs we work with have had their structure standardized according to the Biolink Data Model [Team 2020b]. This standardization allows us to ingest and process new data from different sources using the same pipeline.

This pipeline begins by converting a data source to CSV (or TSV) format using the Knowledge Graph eXchange (KGX) tool [Team 2020a], which was developed by a collaborative effort for general use. Given the resulting CSVs (or TSVs), we then run a second processing step specific to our implementation, to convert this input data to our database representation.

Comparing the two KGs reveals a design similarity in using the HGNC CURIE to index gene concepts (see green highlighted CURIES); however, it also reveals critical differences. The Robokop KG omitted cross-reference indexes to other commonly used gene indexes—NCBIGene, Ensembl, CUI, NCIT and MESH (Figure 3). Moreover, the Robokop KG chose the PANTHER-FAMILY CURIEs to index protein concepts instead of UniProtKB. Beyond concept CURIE differences, the graphs differ significantly with respect to predicate designations. Robokop's `part_of` predicate (as opposed to RTX's encodes predicate) is used to traverse between a specific gene and its respective coding gene/protein family. These differences may at first glance appear trivial, but the underlying differences in these graphs make cross-knowledge graph queries inoperable.

5 QUERY LANGUAGES

mediKanren currently supports two query languages: a low-level language that is essentially raw miniKanren, extended with relations to traverse knowledge graphs; and a more abstract interface, `query/graph`, that makes standard queries easier to write and to read.

5.1 Low-level miniKanren Query Language

Knowledge graphs can be queried directly in miniKanren, with the help of relations in mediKanren's miniKanren/graph database interface library. Graph database relations `db:~name-concepto` and `db:edgeo` can be used to find concepts (drugs, genes, diseases, etc.) and to find edges in a knowledge graph that unify against a given subject-predicate-object triple, for example.

While `db:edgeo` is a pure relation, `db:~name-concepto` requires that its second argument be a ground string. This is typical of the design of mediKanren—while we try to preserve relationality when possible, in some cases it doesn't seem useful or scalable to make a relation operate for "all modes." Complex mediKanren queries often chain multiple miniKanren queries, using Racket code to post-process each set of results. In these two ways,

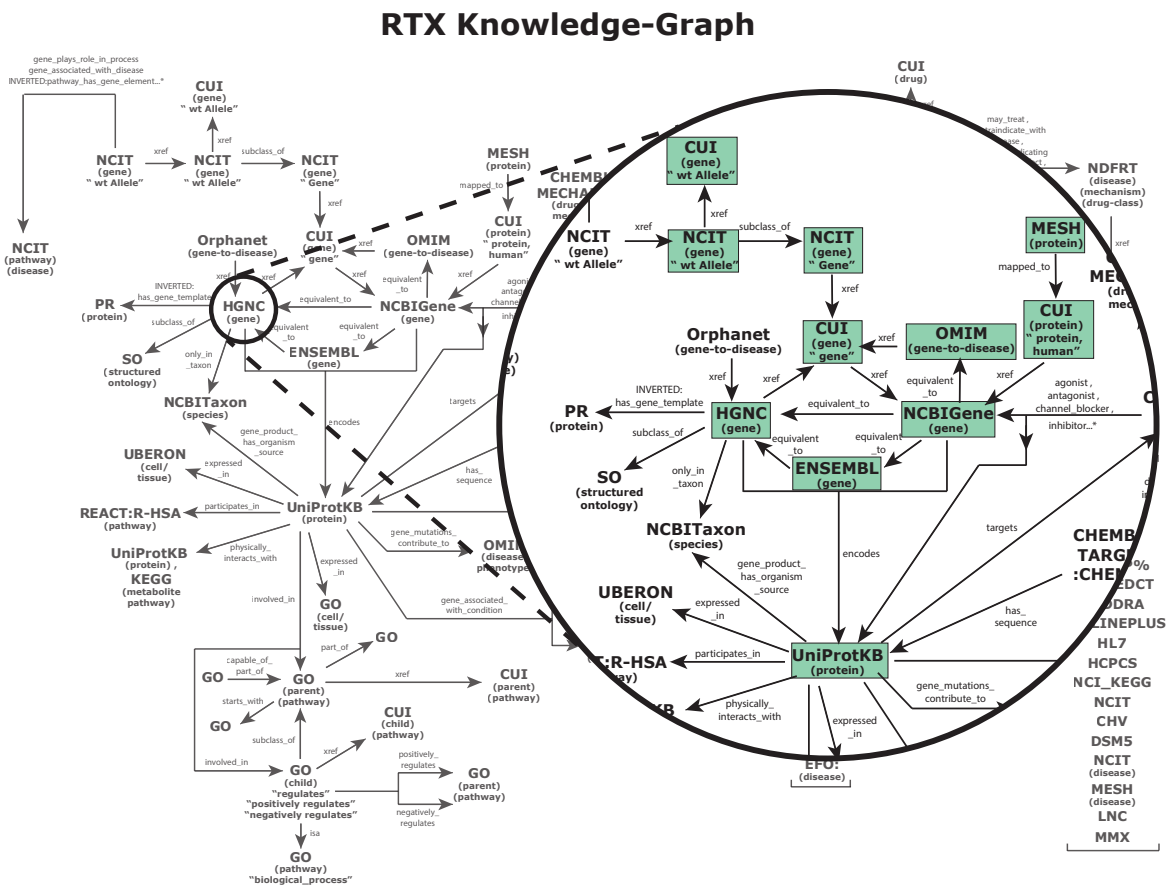

Fig. 2. Zoomed-in and green highlighted concepts represent gene and protein concept specific CURIES, and the predicate paths that connect them. Concept types are noted in parenthesis below each database name (bold). Edge predicates are located above each arrow, directing the relation between concepts.

mediKanren queries are less "pure" than traditional relational miniKanren code used in relational interpreters, for example [Byrd et al. 2017].

Here are several low-level mediKanren helpers, operating over the SemmedDB, Monarch[1], and RTX knowledge graphs, to give a sense of what the code looks like. edgeo is a pure relation that takes a list e representing a subject-predicate-object edge in one of the three knowledge graphs; e may be a fresh logic variable, or contain fresh variables. ~name-concepto is an impure relation that takes a string n and a concept c; n must be a ground string, while c can be a fresh logic variables, or contain logic variables.

```
(define (edgeo e)
  (conde
```

[1]The older "Monarch" knowledge graph described in this code has been replaced with the "Orange" knowledge graph in the current version of mediKanren.

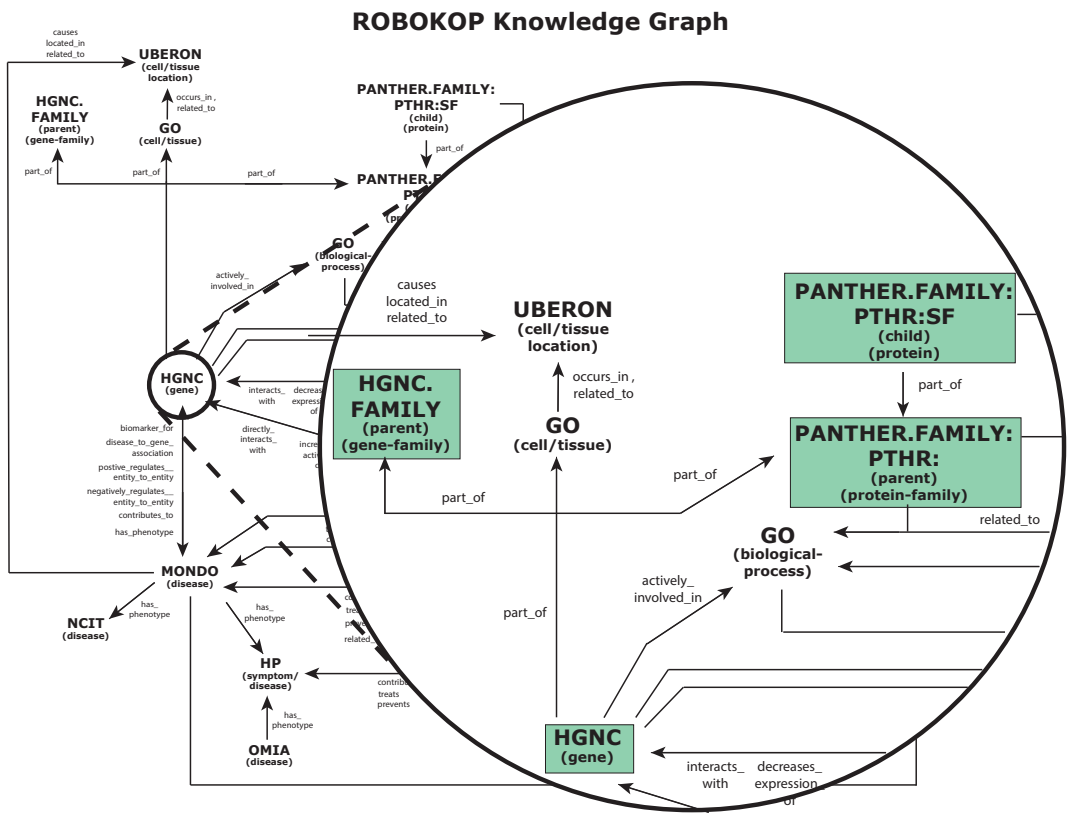

Fig. 3. Zoomed-in and green highlighted concepts represent gene and protein concept specific CURIES, and the predicate paths that connect them. Concept types are noted in parenthesis below each database name (bold). Edge predicates are located above each arrow, directing the relation between concepts.

```
    ((fresh (ee) (== `(semmed . ,ee) e) (db:edgeo semmed ee)))
    ((fresh (ee) (== `(monarch . ,ee) e) (db:edgeo monarch ee)))
    ((fresh (ee) (== `(rtx . ,ee) e) (db:edgeo rtx ee)))))

(define (~name-concepto n c)
  (conde
    ((fresh (cc) (== `(semmed . ,cc) c) (db:~name-concepto semmed n cc)))
    ((fresh (cc) (== `(monarch . ,cc) c) (db:~name-concepto monarch n cc)))
    ((fresh (cc) (== `(rtx . ,cc) c) (db:~name-concepto rtx n cc)))))
```

DECREASES and INCREASES are pure relations that act as “meta-predicates” by enumerating over the individual concrete predicates that express the notion of decreasing or increasing gene expression, severity of a disease, etc.

The newer knowledge graphs include hundreds of predicates, which make such "meta-predicates" very helpful when writing queries by hand. In the future we hope to use predicate ontology mappings to make it easier to reason about the connection between concrete predicates.

```
(define (DECREASES pred)
  (fresh (_)
    (conde
      [(== `(,_ . "treats") pred)]
      [(== `(,_ . "prevents") pred)]
      [(== `(,_ . "negatively_regulates") pred)])))

(define (INCREASES pred)
  (fresh (_)
    (conde
      [(== `(,_ . "produces") pred)]
      [(== `(,_ . "positively_regulates") pred)])))
```

Finally we come to the query itself, which is wrapped in calls to Racket functions to sort the query results, once duplicates have been removed. This query produces all subject-predicate-object edges in which the object concept is the gene DNM1L. This query is very tedious and error-prone to write by hand—we will see a more abstract interface in Section 5.2.

```
(sort
 (rem-dups
   (run* (q)
     (fresh (edge eid tacrine-details subject object pred eprops db
                  scid scui sname sdetails
                  ocid ocui oname odetails
                  pred-id pred-name)
       (== `(,eid ,subject ,object ,pred . ,eprops) edge)
       (~name-concepto "DNM1L" `(,db . ,object))
       (edgeo `(,db . ,edge))
       (== `(,pred-id . ,pred-name) pred)
       (== `(,scid ,scui ,sname . ,sdetails) subject)
       (== `(,ocid ,ocui ,oname . ,odetails) object)
       (== `(,db ,sname ,pred-name ,oname) q))))
 (lambda (e1 e2)
   (match `(,e1 ,e2)
     [`((,db1 ,sname1 ,pred-name1 ,oname1)
        (,db2 ,sname2 ,pred-name2 ,oname2))
      (string<? pred-name1 pred-name2)])))
```

This query produces 502 answers, including:

```
'((semmed "Post-Translational Protein Processing" "affects" "DNM1L wt Allele")
  (semmed "RNA Interference" "affects" "DNM1L wt Allele")
  (semmed "Signal Transduction Pathways" "affects" "DNM1L wt Allele")
  ...
  (rtx "GTP" "directly_interacts_with" "DNM1L")
  (rtx "H2O" "directly_interacts_with" "DNM1L")
```

```
  (rtx "Orthophosphate" "directly_interacts_with" "DNM1L")
  ...
  (monarch "DGUOK" "interacts_with" "DNM1L")
  ...
  (monarch "CASP8" "interacts_with" "DNM1L")
  ...
  (monarch "PEX14" "interacts_with" "DNM1L")
  ...
  (rtx "MIR516A2" "regulates" "DNM1L"))
```

5.2 query/graph Query Language

As can be seen in the code above, raw miniKanren is a low-level, and often clumsy, interface for expressing biomedical queries. To make it easier, faster, and less error-prone to write and read queries, mediKanren also supports a medium-level query interface, `query/graph`. Here is an example of a two-hop `query/graph` query that attempts to find a therapeutic option for a UAB-PMI participants with rare mutations resulting in over-expression/gain-of-function of the RHOBTB2 gene. The clinical manifestations of RHOBTB2 over-expression cause seizures, severe global developmental delay, movement disorders, ataxia and generalized decrease in muscle tone. Thus, the query aims to find FDA Approved drugs `X` that either directly (one-hop) or indirectly (two-hop), through some gene or protein `Y`, inhibit the over-production of the RHOBTB2 gene.

```
(query/graph
  ((X       drug)
   (Y       gene-or-protein)
   (rhobtb2 "CUI:C1425762")
   (T       #f))
  ((X->Y       negatively-regulates)
   (Y->rhobtb2 positively-regulates)
   (X->T       drug-safe))
  (X X->Y Y Y->rhobtb2 rhobtb2)
  (X X->T T))
```

Of the 11 "safe" drugs, 8 of them are scored with a "confidence" score of 0.925, including celecoxib. (This confidence score should not be taken too literally—this does not mean we have 92% confidence in the results! However, we can use relative confidence scores to help rank results.)

```
(0.925
   (("CHEBI:41423" . "CUI:C0812258") ("CUI:C0812258" . "CUI:C1425762"))
   (("celecoxib" . "E2f transcription factor 1")
    ("E2f transcription factor 1"
     .
     "Rho-related btb domain-containing protein 2")))
```

As described in Figure 4, the FDA approved compound celecoxib that was returned is an indirect inhibitor of RHOBTB2. Upon sharing this result with the participants' physician, celecoxib therapy was initiated by admission into clinical trials. As of January of 2020, the participants' have reported a decreased number of ataxic events with an increase in focus and attention span.

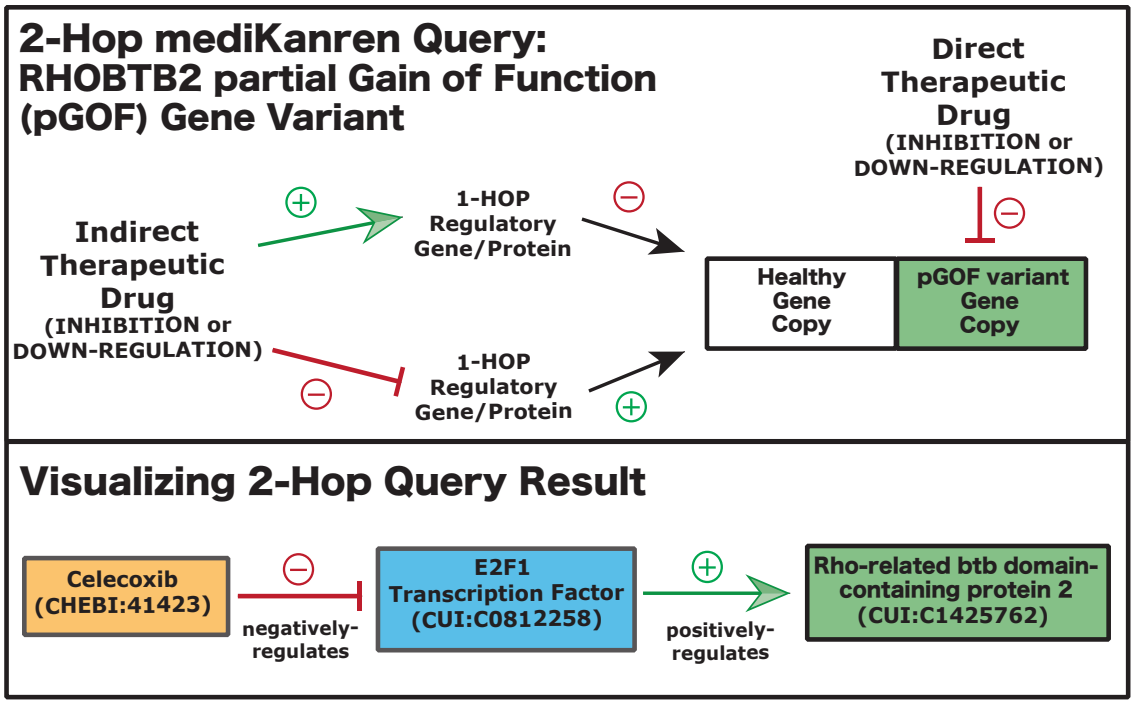

Fig. 4. Two paneled diagram shows the therapeutic strategy (top) for partial gain of function (pGOF) genetic variants such as the RHOBTB2 case from UAB-PMI. The Bottom panel serves as a visualization aid for the 2-hop mediKanren query result. The FDA-Approved drug Celecoxib was found to be an indirect inhibitor of the E2F1 Transcription Factor, which is responsible for expression of the RHOBTB2 gene. Concept CURIEs are notated in parentheses.

6 GRAPHICAL USER INTERFACE

As can be seen in Section 5, the query languages supported by mediKanren require knowledge of Racket and of relational programming. It is easy to make mistakes when hand-writing queries in these languages. Also, while expressive, these programmatic queries can take a while to write.

To support users who are non-programmers (or non-Racket/non-miniKanren programmers), we have created a graphical user interface (GUI) that is easy and fast to use. This GUI interface only supports a limited subset of mediKanren queries:

(1) one-hop queries of the form *known subject concept S is related to unknown object concept O through known predicate P*;
(2) one-hop queries of the form *unknown subject concept S is related to known object concept O through known predicate P*;
(3) and two-hop queries of the form *known subject concept S1 is related to unknown concept X through predicate P1, and concept X is also related to object O2 through known predicate P2.*

Figure 5 is a screenshot of the mediKanren GUI, showing a simple two-hop drug repurposing query provided by NCATS. This query is trying to determine if the cancer drug imatinib may also treat asthma through some unknown mechanism. Effectively, the query asks, "is there some concept X such that imatinib reduces the severity of/inhibits X, and where X causes/increases the severity of asthma?" One of the X's in this case is the gene VEGFA. The user can click on one of the PubMed identifier links in the lower-right of the GUI window to open up that paper in a Web browser.

7 DESIGN CONSIDERATIONS

The datasets we deal with, while many GB in size, are still small enough that an expensive computing cluster is unnecessary. In fact, mediKanren has been designed to run on fairly modest laptops to allow scientists to use their personal machines to explore the data without needing internet access or other support. For this reason, our custom database implementation performs most data lookups directly from disk without prefetching it into RAM. This choice reduces both memory usage and startup time. Despite this tradeoff, mediKanren is still fast enough for interactive use during case reviews or other situations where it's necessary to perform many simple queries quickly.

Rather than build on an existing relational or graph database, we chose to implement our system using miniKanren and Racket, which are both flexible and expressive languages. We made this choice to reduce the risk of painting ourselves into a corner as we discovered requirements during development, as we were more confident in our ability to adapt miniKanren and Racket as necessary. In particular, the faster-miniKanren implementation is fairly small and simple, making it quite hackable.

8 FUTURE WORK

We have started a new implementation of our database with a more compact data representation, higher performance, and a more direct integration with miniKanren, including the ability to precompute and persist finite relations. The new implementation will also support stratified aggregation and computing fixed points, in the sense of Datalog, allowing us to reduce the amount of ad-hoc Racket code we have to write when expressing complex queries. Data ingestion and processing will be possible to express relationally, rather than requiring ad-hoc Racket code. At some point, we would also like to support temporally stratified relations, allowing us to describe dynamic systems, such as the UI layer, relationally as well, reducing the effort needed to prototype new UIs.

Another key feature we aim to improve on is intelligent use of ontology data sources. Ontologies have been extensively in biomedical and translation science to organize and structure knowledge with child-parent relationships (ie symptoms comprising a disease) and/or functional class information (ie a drug or gene classified by its mechanism of action or function) [Denny P 2018; Marcos Martínez-Romero 2017]. A key issue in gathering information from ontologies are the inherent asymmetries that exists between data sources. Figure 6 outlines the structural differences in the Medical Subject Headings (MESH) and the Chemicals Entities of Biological Interest (CHEBI) ontologies [de Matos P 2010] with respect categorizing anticoagulant drug concepts. To date, mapping concepts across ontologies remains an ongoing struggle and area of active research [Groß A 2016]. In the future, we hope to develop a systematic way of identifying and reconciling differences across ontologies during import without losing data integrity.

REFERENCES

William E. Byrd. 2009. *Relational Programming in miniKanren: Techniques, Applications, and Implementations.* Ph.D. Dissertation. Indiana University.

William E. Byrd, Michael Ballantyne, Gregory Rosenblatt, and Matthew Might. 2017. A unified approach to solving seven programming problems (functional pearl). *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 8.

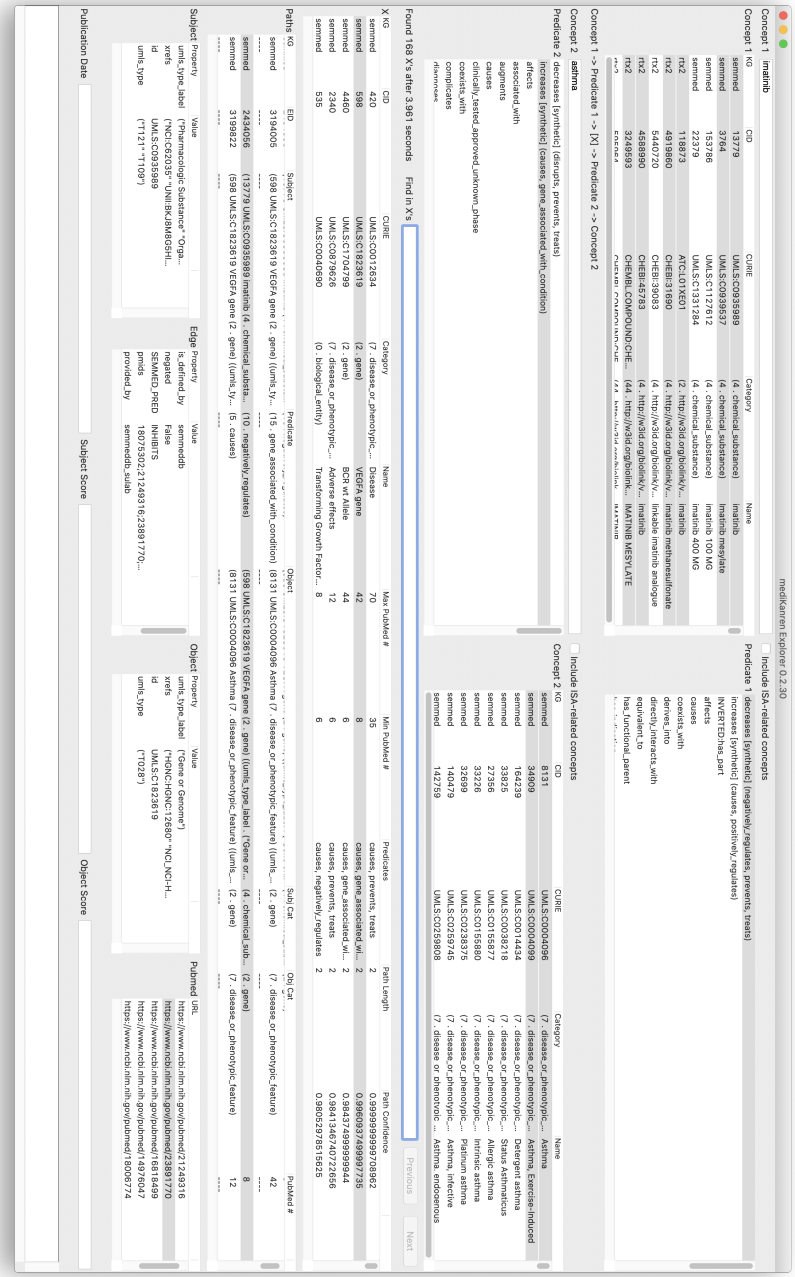

Fig. 5. mediKanren Graphical User Interface, showing 2-hop imatinib/asthma query

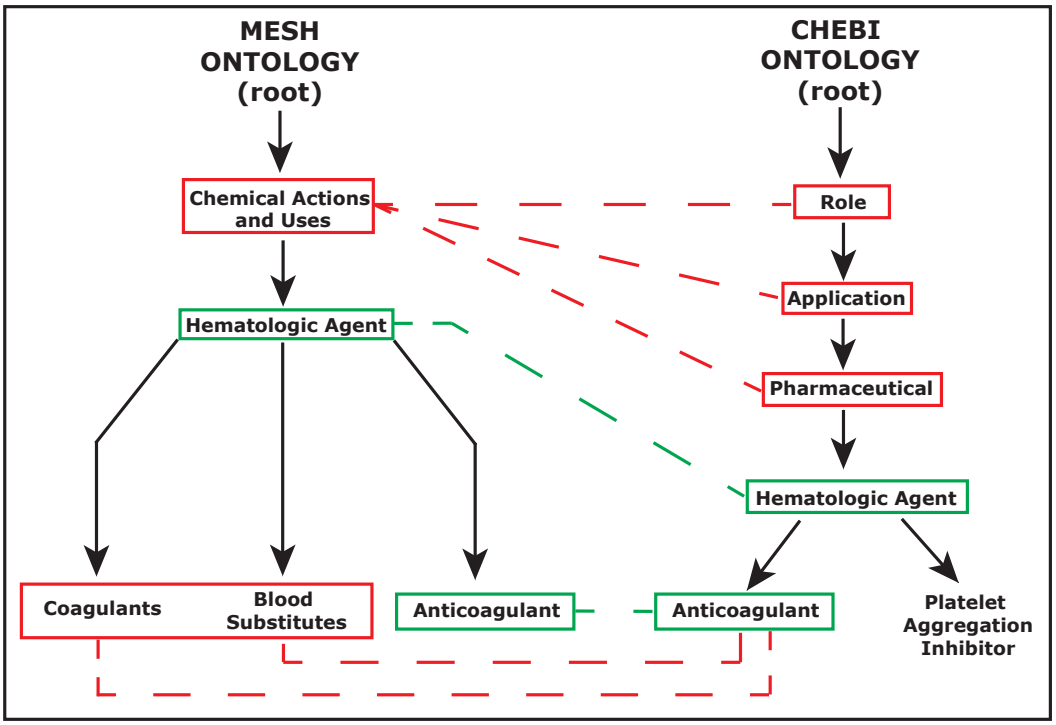

Fig. 6. MESH and CHEBI chemical ontologies display some similarities (dashed green) but many more structural differences (dashed red) for organization of drug class architecture.

Dekker A et al. de Matos P, Alcántara R. 2010. Chemical Entities of Biological Interest: an update. *Nucleic Acids Res* (2010), D249-D254. Issue 38. doi:10.1093/nar/gkp886.

Hill DP Lovering RC Plun-Favreau H Roncaglia P Denny P, Feuermann M. 2018. Exploring autophagy with Gene Ontology. *Journal Biomedical Semantics* 14 (2018), 419–436. doi:10.1080/15548627.2017.1415189.

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, and Sam Tobin-Hochstadt. [n.d.]. https://felleisen.org/matthias/manifesto/.

Daniel P. Friedman, William E. Byrd, Oleg Kiselyov, and Jason Hemann. 2018. *The Reasoned Schemer* (2nd ed.). The MIT Press, Cambridge, MA, USA.

Rahm E. Groß A, Pruski C. 2016. Evolution of biomedical ontologies and mappings: Overview of recent approaches. *Comput Struct Biotechnol J.* 14 (2016), 333–340. doi:10.1016/j.csbj.2016.08.002.

Arif E. Jinha. 2010. Article 50 million: an estimate of the number of scholarly articles in existence. 23 (2010), 258–263. doi:10.1087/20100308.

Martin J. O'Connor John Graybeal Alejandro Pazos Mark A. Musen Marcos Martínez-Romero, Clement Jonquet. 2017. *Journal Biomedical Semantics* 8 (2017), 21. doi: 10.1186/s13326-017-0128-y PMCID: PMC5463318.

Page RDM. 2019. Ozymandias: a biodiversity knowledge graph. *Scientific Reports* 7 (2019), e6739. doi:10.7717/peerj.6739.
Robbins R Ross C, Empinado H. 2019. An AI expert's toughest project: writing code to save his son's life - STAT. https://www.statnews.com/2019/07/25/ai-expert-writing-code-save-son/.
Tlimat A Horng S Sontag D. Rotmensch M, Halpern Y. 2017. Learning a Health Knowledge Graph from Electronic Medical Records. *Scientific Reports* 7, 1 (2017), 5994. doi:10.1038/s41598-017-05778-z.
B. Shepard. 2019. Diagnosis in 2.127 seconds: Solving a years-long vomiting mystery using AI, research and brain power - News. https://www.uab.edu/news/health/item/10703-diagnosis-in-2-127-seconds-solving-a-years-long-vomiting-mystery-using-ai-research-and-brain-power.
Biomedical Data Translator Tangerine Team. 2020a. Knowledge Graph eXchange tools for Biolink Model compliant graphs. https://github.com/NCATS-Tangerine/kgx.
Biolink Model Team. 2020b. Biolink Model. https://biolink.github.io/biolink-model/.
W3C. 2010. CURIE Syntax 1.0: A syntax for expressing Compact URIs. https://www.w3.org/TR/curie/.