M2IF CR3: graph-based knowledge representation

Angela Bonifati (PR, UCBL1)
angela.bonifati@univ-lyon1.fr

Russ Harmer (CR, CNRS@ENSL)
russell.harmer@ens-lyon.fr

Fridays
13h30–15h30 & 15h45–17h45
[may change]
no physical presence
[may change, keep an eye on the PAD]

• Angela
  • BBB: https://classe-info.univ-lyon1.fr/bon-ask-qvm-n82
  • access code: 564999
• Russ
  • BBB: https://ent-services.ens-lyon.fr/entVisio
  • meeting: M2IF-CR3
  • password: m2cr3

• schedule [will be confirmed on the PAD]
  • 11/09 @ 15h45: Angela
  • 18/09: Russ
  • 25/09: Angela
  • 2/10: Russ
  • 9/10: Angela
  • 16/10: Russ
  • 23/10: Angela

• protocol
  • please leave your microphone switched off — unless you have a question
  • you can also use the public chat to ask questions
context
RDB vs. GDB

- data shaped like graphs is increasingly common
  - social networks: friends, followers, colleagues, Covid contacts, …
  - complex systems: molecular biology, neuroscience, energy grids, smart cities…
  - &c.
- traditional DBs are tuple-based (relational)
  - can nonetheless encode graph-shaped data
- what about updates?
  - an update may be valid but create an invalid encoding of a graph [failure of abstraction barrier]
  - solution #1: express updates in terms of graphs and translate into SQL (or whatever)
  - solution #2: create a graph-specific data model (+ DDL + DML), e.g. RDF or property graphs (PGs)
- what about queries?
  - graph-shaped queries may be awkward to write and inefficient to execute in SQL [more abstraction failure]
  - solution #1: as above but with extra black magic
  - solution #2: as above but with a general query language (QL), e.g. SparQL for RDF
- what about schemas?
  - in a relational DB, the main purpose of a schema is to enforce typing on columns
  - in a graph DB, there is further scope for structural constraints…
    - node and edge types, property types, arity constraints, &c.
    - RDFS is a W3C ‘standard’
    - an ISO standard for PGs and PG schemas is currently under development
in this class

- Angela
  - property graph data model, graph queries & query languages
  - schema validation, evolution and inference [for PGs]
- Russ
  - graph rewriting, concretely and abstractly
  - graph hierarchies for schemas and knowledge representation

- graph rewriting
  - the concrete setting of simple graphs and homomorphisms
  - the abstract setting of categories: push-outs, pull-backs and pull-back complements

- graph hierarchies
  - propagation of rewriting: data/schema co-evolution
  - knowledge representation: overview/demo of the KAMI bio-curation system

- assessment
  - two homework assignments
  - presentation of a research paper
simple graph rewriting

- a(n undirected) simple graph has
  - a finite set of nodes $N$
  - a (symmetric) binary relation $E$ on the set of nodes

- how might we rewrite a simple graph?
  - add or delete nodes or edges
  - merge or clone nodes

- how could we specify a rewrite of a simple graph?
  - need two graphs: a LHS that matches the current graph and a RHS that determines the changes
  - what LHS and RHS do we need to add a node? an edge? to delete? &c.
  - how does a LHS match the current graph?

- need a notion of homomorphism
  - a homomorphism $h : G_1 \rightarrow G_2$ is a function from $N_1$ to $N_2$ such that if $n E_1 m$ then $h(n) E_2 h(m)$
  - a rewriting rule is a homomorphism
    - its LHS has an injective homomorphism to the current graph [the matching or instance]
    - it is applied by replacing the image of the LHS by its RHS
  - an expansive instance of $r : L \rightarrow R$ in $G$ is an injective homomorphism $m : L \rightarrow G$
    - when applied, can add and merge
  - a restrictive instance of $r : L \leftarrow R$ in $G$ is an injective homomorphism $m : L \rightarrow G$
    - when applied, can delete and clone
rule-based modelling
(simplified)

- nodes and edges represent proteins and bonds between them
  - there can be many different types of proteins (think of them as ‘colours’ for now)
  - rewriting rules can create and destroy bonds (and proteins): node + node ⇔ node—node
  - given many rules and an initial graph [state], this defines a transition system [ad hoc]
- cf. Petri nets AKA multi-set rewriting
  - one colour per distinct connected component [but need to enumerate all possibilities in advance]
  - rewriting rules create and destroy nodes only: red + blue ⇔ purple [but number of colours explodes]
  - no longer need edges and straightforward to define a transition system [ad hoc]
- multi-set rewriting as typed graph rewriting without edges
  - multi-set = function from a set of individuals to a set of types of individuals, i.e. colours
  - natural notion of homomorphism of multi-sets, i.e. preserve colours
  - the ideas of the previous slide still apply and define the ad hoc transition system as a special case
- rule-based modelling as typed graph rewriting
  - fix a graph $T$ that defines all types of proteins and the permitted bonds between them
  - $h : G \rightarrow T$ is a typed graph with a natural notion of homomorphism, i.e. preserve typing
  - the ideas of the previous slide still apply and define the ad hoc transition system as a special case
  - note that the edges of $G$ are constrained by those of $T$: a notion of schema
towards graph hierarchies

• what if we want to delete a node in $T$?
  • delete all its instances in $G$?
  • or only allow this if there are no instances in $G$?
• what if we want to add a node to $G$ that has no typing node in $T$?
  • add a node to $T$?
  • never allow this?
• &c.

• a graph hierarchy is a collection of graphs connected by homomorphisms
  • a rewrite of one graph canonically propagates to maintain the consistency of the hierarchy
  • enables data/schema co-evolution: prescriptive and descriptive updates
  • more generally, a generic foundation for graph-based knowledge representation
• in reality, this happened backwards
  • we started building a KR system for protein-protein interactions in cellular signalling
  • was convenient to represent each interaction as a small graph typed by a summary graph
  • these small graphs often needed to be updated—and this sometimes affected the summary
    • the birth of hierarchies …
    • … and of propagation of rewriting
plan for my classes

• 18/09
  • categories, monos, push-outs
  • expansive rewriting
  • forward propagation
  • homework assignment #1

• 2/10
  • pull-backs, pull-back complements
  • restrictive rewriting
  • backward propagation
  • homework exercise #2

• 16/10
  • graph-based knowledge representation
  • the KAMI bio-curation system
  • choice of research papers to present