Programming III COMP2209

Coursework 2

Individual Coursework *(cf. collaboration policy)*

- *Aims of this coursework:*
  1. *Programming in the DrRacket environment.*
  2. *Programming recursive functions on S-expressions, n-ary trees, and graphs.*
- *Deadline: Week 10, Thursday December 05, 4.00pm.*
- *Electronic submission: Five files:*
  1. `clause-match.scm`*: answer to question 1.*
  2. `apply-bindings.scm`*: answer to question 2.*
  3. `pattern-hierarchy.scm`*: answer to question 3.*
  4. `pattern-match-graph.scm`*: answer to question 4.*
  5. `path-between.scm`*: answer to question 5.*
- *Marking scheme: 20% for each question.*

PROV is the standard for provenance on the Web [1], [2].

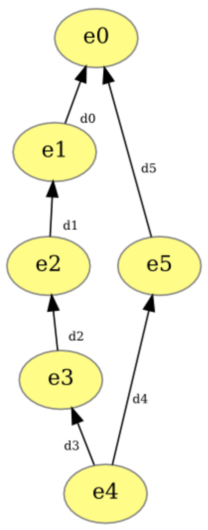
> Provenance is defined as a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing. In particular, the provenance of information is crucial in deciding whether information is to be trusted, how it should be integrated with other diverse information sources, and how to give credit to its originators when reusing it. In an open and inclusive environment such as the Web, where users find information that is often contradictory or questionable, provenance can help those users to make trust judgements.

For the purpose of this coursework, we consider the subset of PROV consisting of entities and derivations.

> An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary.
>
> A derivation is a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity.

Figure 1 illustrates an example of this subset of PROV in which 6 entities are identified and some dependencies between them are expressed as derivations. Such a representation of provenance forms a graph structure, referred to as a provenance graph, in which nodes are entities and edges are derivations.

A provenance graph may be CYCLIC.



```
document
  default <http://example.org/>

  entity(e0, [prov:label="entity e0", prov:location="London"])
  entity(e1)
  entity(e2)
  entity(e3)
  entity(e4, [prov:location="Southampton"])
  entity(e5)

  wasDerivedFrom(d0;e1,e0, [prov:type='prov:Revision'])
  wasDerivedFrom(d1;e2,e1, [prov:type='prov:Revision'])
  wasDerivedFrom(d2;e3,e2, [prov:type='prov:Revision'])
  wasDerivedFrom(d3;e4,e3, [prov:type='prov:Revision'])
  wasDerivedFrom(d4;e4,e5)
  wasDerivedFrom(d5;e5,e0)
endDocument
```

Figure 1: An Example of Provenance Graph in PROV-N Notation

The provenance graph of Figure 1 is encoded as a symbolic expression in Figure 2.

- A provenance graph is defined as a list of statements, for entities and derivations.
- The identifiers (`e0`, `e1`, ..., `d0`, `d1`, ...) occur in first position after the keywords `entity`/`wasDerivedFrom`, and are always symbols. They can be obtained by the accessor `statement-id`, defined below.
- Statements can occur in any order.
- There is at most one occurrence of a statement with a given identifier in a provenance graph.
- The namespace declaration is not represented and can be ignored in this coursework.
- Derivation statements are of the form `(wasDerivedFrom id source destination` $\langle attrs... \rangle$ `)` where `source` and `destination` denote symbols representing entity identifiers. Such statements correspond to directed edges in Figure 1. Source and destination can respectively be obtained by the accessors `derivation-src`, `derivation-dst`, defined below. Source and destination may be followed by $\langle attrs... \rangle$, optional attributes expressed as symbol-value pairs, and accessible by the accessor `derivation-attrs`, defined below.
- Entity statements are of the form `(entity id` $\langle attrs... \rangle$ `)` where `id` is an identifier; it may be followed by $\langle attrs... \rangle$, optional attributes expressed as symbol-value pairs, accessed by `entity-attrs`, defined below.

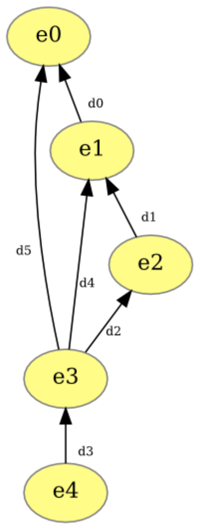
When manipulating provenance graphs, the following accessors MUST be used where appropriate.

```
(define statement-id cadr)
(define derivation-src caddr)
(define derivation-dst cadddr)
(define derivation-attrs cddddr)
(define entity-attrs cddr)
```

Figure 3 illustrates another provenance graph.

```
(define graph1
  '((entity e0 (prov:label "entity e0") (prov:location "London"))
    (entity e1)
    (entity e2)
    (entity e3)
    (entity e4 (prov:location "Southampton"))
    (entity e5)
    (wasDerivedFrom d0 e1 e0 (prov:type prov:Revision))
    (wasDerivedFrom d1 e2 e1 (prov:type prov:Revision))
    (wasDerivedFrom d2 e3 e2 (prov:type prov:Revision))
    (wasDerivedFrom d3 e4 e3 (prov:type prov:Revision))
    (wasDerivedFrom d4 e4 e5)
    (wasDerivedFrom d5 e5 e0)))
```

Figure 2: Encoding of the provenance graph of Figure 1 as a symbolic expression.



```
(define graph2
  '((entity e0)
    (entity e1)
    (entity e2)
    (entity e3)
    (entity e4)
    (wasDerivedFrom d0 e1 e0)
    (wasDerivedFrom d1 e2 e1)
    (wasDerivedFrom d2 e3 e2)
    (wasDerivedFrom d3 e4 e3)
    (wasDerivedFrom d4 e3 e1)
    (wasDerivedFrom d5 e3 e0)))
```

Figure 3: Another Provenance Graph

We define a pattern language by the following grammar

$$\begin{array}{rcl}
\langle \mathit{pattern} \rangle & ::= & (\ \langle \mathit{pattern\ clause} \rangle^{+} \ ) \\
\langle \mathit{pattern\ clause} \rangle & ::= & (\mathsf{entity}\ \langle \mathit{atom} \rangle \ \langle \mathit{attributes} \rangle \ ) \\
 & & \mid (\mathsf{wasDerivedFrom}\ \langle \mathit{atom} \rangle \ \langle \mathit{atom} \rangle \ \langle \mathit{atom} \rangle \ \langle \mathit{attributes} \rangle \ ) \\
\langle \mathit{atom} \rangle & ::= & \langle \mathit{var} \rangle \mid \langle \mathit{number} \rangle \mid \langle \mathit{string} \rangle \mid \langle \mathit{symbol\ not\ starting\ with\ '?'} \rangle \\
\langle \mathit{var} \rangle & ::= & \langle \mathit{symbol\ starting\ with\ '?'} \rangle \\
\langle \mathit{attributes} \rangle & ::= & [\ .\ \langle \mathit{var} \rangle \ ]?
\end{array}$$

where the following operators are used:

- $[xx\ yy]$ matches term xx followed by yy
- xx ? matches 0 or 1 occurrence of xx
- xx + matches one occurrence of xx or more

1. Define a function `clause-match` that takes a pattern clause and a statement and returns a list of bindings (between variables and values) if the statement matches the patterns, and false otherwise.

```
(clause-match
  '(wasDerivedFrom ?der ?e1 ?e2 . ?attrs_derv)
  '(wasDerivedFrom der   e1 e2  (prov:type "Revision")))
-> ((?der . der)
    (?e1 . e1)
    (?e2 . e2)
    (?attrs_derv (prov:type "Revision")))


(clause-match
  '(wasDerivedFrom ?der ?e1 e2 . ?attrs_derv)
  '(wasDerivedFrom der   e1 ex  (prov:type "Revision")))
-> #f

(clause-match
  '(entity ?e)
  '(entity e  (prov:type "Revision")))
-> #f
```

   For the purpose of programming `clause-match`, consider a clause to be a S-expression and use a recursion on S-expressions.

   Assume that a given variable can occur at most once in a pattern clause.

2. Define a function `apply-bindings` that takes a list of bindings (as returned by `clause-match`) and a pattern clause, and returns a new pattern clause, in which variables bound in the list of bindings have been replaced by their values.

```
(apply-bindings '((?e . e))
                '(wasGeneratedBy ?gen ?e ?a . ?attrs_gen)))
-> (wasGeneratedBy ?gen e ?a . ?attrs_gen)
```

```
(define *der* '(wasDerivedFrom der
                               e1
                               e2
                               (prov:type . "Revision")))
(apply-bindings (clause-match '(wasDerivedFrom ?der
                                                ?e1
                                                ?e2 . ?attrs)
                              *der*)
                *der*)

-> (wasDerivedFrom der
                   e1
                   e2
                   (prov:type . "Revision"))
```

3. Patterns have been defined as a flat list of pattern clauses. In this exercise, we consider patterns formed of `wasDerivedFrom` clauses only. Define a function `pattern-hierarchy` that constructs an n-ary tree representing this pattern. The root of the tree is label by the source of the first `wasDerivedFrom` clause in the pattern.

   Any subtree of the resulting n-ary tree has the shape

```
(?e
  (?e1 ...)
  ...
  (?en ...))
```

   whenever there are clauses `(wasDerivedFrom ?der1 ?e ?e1 .  ?attr1)`,..., `(wasDerivedFrom ?dern ?e ?en .  ?attrn)` in the pattern.

   In the following example `?e2` is followed by `?e1`, itself followed by `?e0` and `?ex`. The latter have no successors.

```
(pattern-hierarchy '((wasDerivedFrom ?der1 ?e2 ?e1)
                     (wasDerivedFrom ?der2 ?e1 e0)
                     (wasDerivedFrom ?der2 ?e1 ex)))
--->
(?e2
  (?e1
     (e0)
     (ex)))
```

   In the following examples `?e1` has four successors. We note that `?e1` has `?e2` as a successor. This pattern is intended to match cycles. In that case `pattern-hierarchy` generates a tree where `?e2` occurs as an internal node in the n-ary tree and also as a leaf.

```
(pattern-hierarchy '((wasDerivedFrom ?der1 ?e2 ?e1 . ?attrs_deriv1)
                     (wasDerivedFrom ?der2 ?e1 ?e0 . ?attrs_deriv2)
                     (wasDerivedFrom ?der2 ?e0 ?ez . ?attrs_deriv3)
                     (wasDerivedFrom ?der2 ?e1 ?ex . ?attrs_deriv4)
                     (wasDerivedFrom ?der2 ?e1 ?ey . ?attrs_deriv5)
                     (wasDerivedFrom ?der2 ?e1 ?e2 . ?attrs_deriv6)))
--->
(?e2
  (?e1
    (?e0
      (?ez))
    (?ex)
    (?ey)
    (?e2)))
```

4. Given a pattern and a graph, the purpose of this exercise is to find all the possible matches for that pattern in the graph. The function `pattern-match-graph` takes a pattern and a graph and returns a list of alternate lists of bindings; each list of bindings can be applied to the pattern to find the nodes in the graph matching that pattern.

   Assume that a given variable can occur at most once in a pattern.

   The following example shows four matches for a pattern involving two derivations $?e_2 \rightarrow ?e_1 \rightarrow ?e_0$.

```
(pattern-match-graph '((wasDerivedFrom ?der1 ?e2 ?e1 . ?attrs_deriv1)
                       (entity ?e2 . ?attrs1)
                       (wasDerivedFrom ?der2 ?e1 ?e0 . ?attrs_deriv2))
                      graph1)
-->
(((?der2 . d0)
  (?e0 . e0)
  (?attrs_deriv2 (prov:type prov:Revision))
  (?attrs1)
  (?der1 . d1)
  (?e2 . e2)
  (?e1 . e1)
  (?attrs_deriv1 (prov:type prov:Revision)))

 ((?der2 . d1)
  (?e0 . e1)
  (?attrs_deriv2 (prov:type prov:Revision))
  (?attrs1)
  (?der1 . d2)
  (?e2 . e3)
  (?e1 . e2)
  (?attrs_deriv1 (prov:type prov:Revision)))

 ((?der2 . d2)
  (?e0 . e2)
  (?attrs_deriv2 (prov:type prov:Revision))
  (?attrs1 (prov:location "Southampton"))
  (?der1 . d3)
  (?e2 . e4)
  (?e1 . e3)
  (?attrs_deriv1 (prov:type prov:Revision)))

 ((?der2 . d5)
  (?e0 . e0)
  (?attrs_deriv2)
  (?attrs1 (prov:location "Southampton"))
  (?der1 . d4)
  (?e2 . e4)
  (?e1 . e5)
  (?attrs_deriv1)))
```

   The following example shows a success, with a list of bindings, which happens to be empty because there is no variable to bind.

```
(pattern-match-graph '((entity e1))
                      graph1)
-->
(())
```

   The following example shows failure, with no list of bindings in the result.

```
(pattern-match-graph '((entity e10))
                      graph1)
-->
()
```

5. Define a function `path-between` that takes a symbol `from`, a symbol `to` and a graph, and returns a shortest path between the nodes identified by `from` and `to`.

   Your solution will make use of the `linked` function predefined in the lecture. This function and any auxiliary functions must be submitted as part as your solution.

```
(path-between 'e4 'e0 graph1)
--> (e4 e5 e0)

(path-between 'e4 'e10 graph1)
--> #f
```

References

[1] Luc Moreau, Paolo Missier (eds.), Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. PROV-DM: The PROV Data Model. W3C Recommendation REC-prov-dm-20130430, World Wide Web Consortium, April 2013. (url: `http://www.w3.org/TR/2013/REC-prov-dm-20130430/`).

[2] Luc Moreau, Paolo Missier (eds.), James Cheney, and Stian Soiland-Reyes. PROV-N: The Provenance Notation. W3C Recommendation REC-prov-n-20130430, World Wide Web Consortium, April 2013. (url: `http://www.w3.org/TR/2013/REC-prov-n-20130430/`).