Collections in PROV

Goal:

- To model structured data in terms of changes to its content as an effect of updates

Approach:

- assume a very general data structure
  - set of key-value pairs (map, associative arrays, dictionaries...)
    - where values can be collections (nesting)
  - these subsume more complex structures
    - ordered nested lists, relational tables, etc.
- Introduce data types to identify collection entities:
  - prov:Collection, prov:EmptyCollection
- Introduce relations to capture the effect of create, insert, remove operations

Relations for collection updates

Insertion relation:

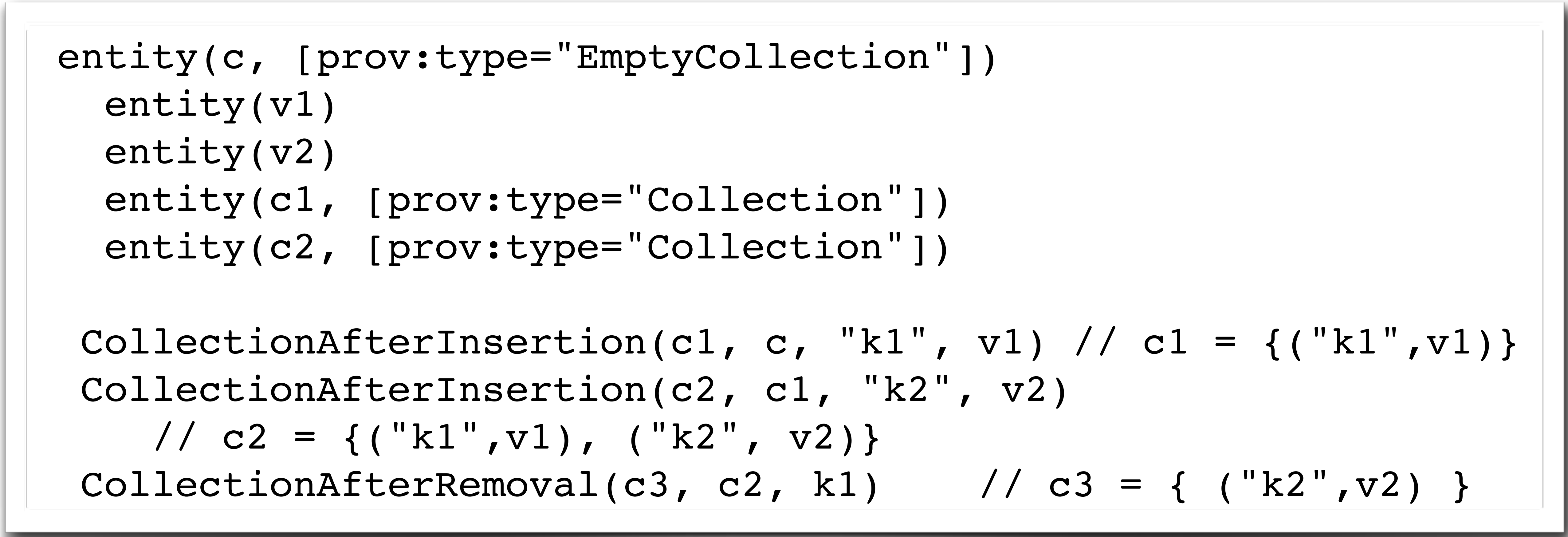
CollectionAfterInsertion(c2, c1, k, v)

states that c2 is the state of the collection following the insertion of pair (k,v) into collection c1;

Removal relation:

CollectionAfterRemoval(c2,c1, k)

states that c2 is the state of the collection following the removal of the pair corresponding to key k from c1.



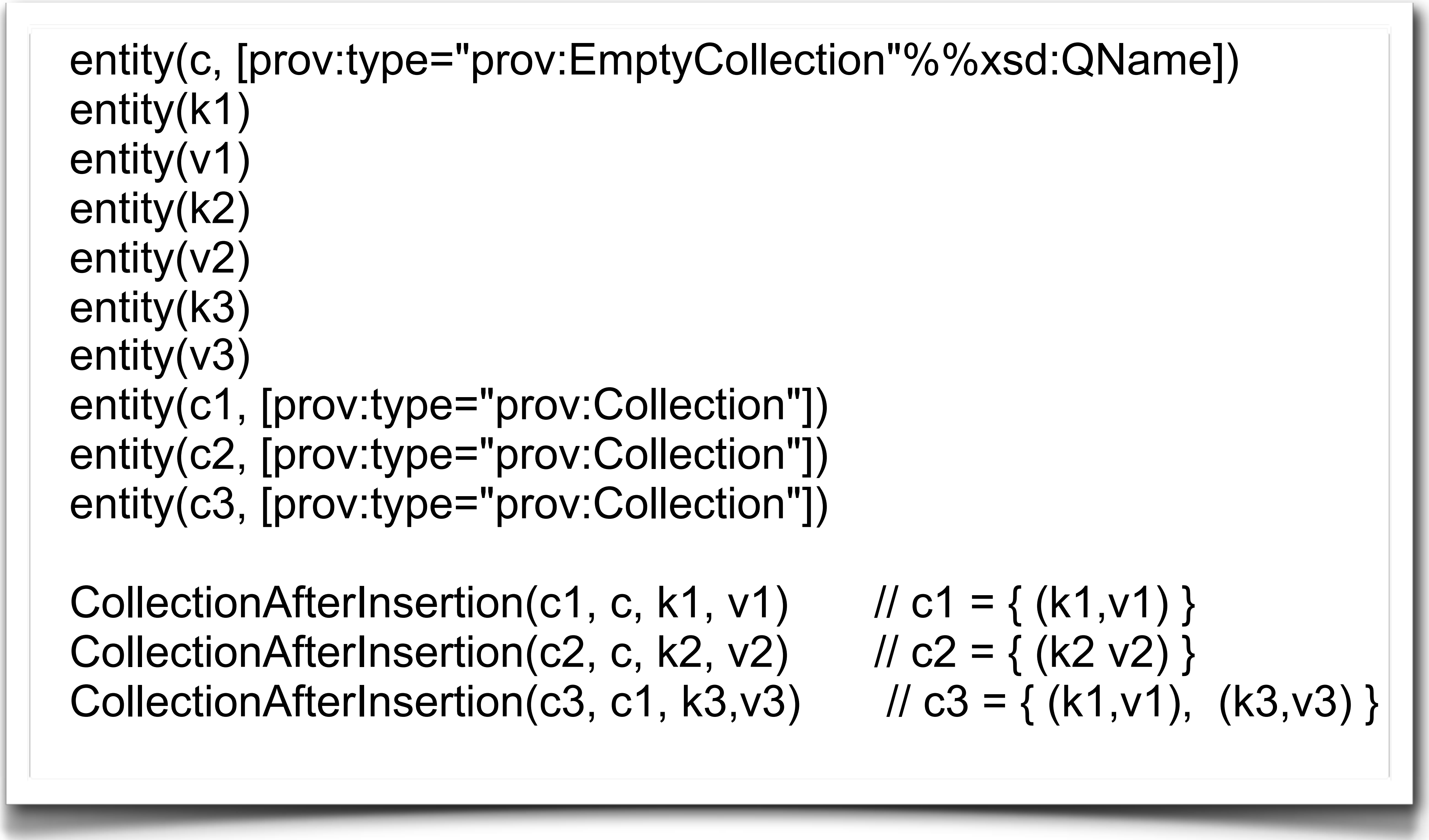
```
entity(c, [prov:type="EmptyCollection"])
  entity(v1)
  entity(v2)
  entity(c1, [prov:type="Collection"])
  entity(c2, [prov:type="Collection"])

 CollectionAfterInsertion(c1, c, "k1", v1) // c1 = {("k1",v1)}
 CollectionAfterInsertion(c2, c1, "k2", v2)
    // c2 = {("k1",v1), ("k2", v2)}
 CollectionAfterRemoval(c3, c2, k1)    // c3 = { ("k2",v2) }
```

Some considerations

- Collections are abstract
  - No assumptions are made regarding the underlying data structure used to store and manage collections
  - In particular, no assumptions are needed regarding the mutability of a data structure that is subject to updates.
- The state of a collection (i.e., the set of key-value pairs it contains) at a given point in a sequence of operations is never stated explicitly.
  - Rather, it can be obtained by querying the chain of derivations involving insertions and removals.
  - Entity type emptyCollection can be used in this context as it marks the start of a sequence of collection operations.
- ...further considerations and constraints omitted for simplicity
  - please ask!

Further cases and examples

- It is possible to have multiple derivations from a single root collection



```
entity(c, [prov:type="prov:EmptyCollection"%%xsd:QName])
entity(k1)
entity(v1)
entity(k2)
entity(v2)
entity(k3)
entity(v3)
entity(c1, [prov:type="prov:Collection"])
entity(c2, [prov:type="prov:Collection"])
entity(c3, [prov:type="prov:Collection"])

CollectionAfterInsertion(c1, c, k1, v1)       // c1 = { (k1,v1) }
CollectionAfterInsertion(c2, c, k2, v2)       // c2 = { (k2 v2) }
CollectionAfterInsertion(c3, c1, k3,v3)        // c3 = { (k1,v1),  (k3,v3) }
```

Batch updates

One can have multiple assertions regarding the state of a collection following a set of insertions, for example:

CollectionAfterInsertion(c2,c1, k1, v1)
CollectionAfterInsertion(c2,c1, k2, v2)
...

This is interpreted as " c2 is the state that results from inserting (k1, v1), (k2, v2) etc. into c1

Keys are unique.
The following set of insertions:

CollectionAfterInsertion(c1, c, k, v1)
CollectionAfterInsertion(c1, c, k, v2)

entails v1==v2.