NegativeWrites Case Study: simplog

Program P :

```
node("a", "a")@1 ;
node("a", "b")@1 ;
node("a", "c")@1 ;
node("b", "a")@1 ;
node("b", "c")@1 ;
node("c", "a")@1 ;
node("c", "b")@1 ;
bcast("a", "hello")@1 ;

node(Node, Neighbor)@next          :- node(Node, Neighbor) ;
log(Node, Pload)@next              :- log(Node, Pload) ;
log(Node2, Pload)@async            :- bcast(Node1, Pload), node(Node1, Node2) ;
log(Node, Pload)                   :- bcast(Node, Pload);
missing_log(A, Pl)                 :- log(X, Pl), node(X, A), notin log(A, Pl) ;
pre(X, Pl)                         :- log(X, Pl), notin bcast(X, Pl)@1 ;
post(X, Pl)                        :- log(X, Pl), notin missing_log(_, Pl);
```

Partial list of new rules for P' derived from NegativeWrites :

```
not_missing_log( A, Pl )     :- notin log( _, Pl ), dom_post_att2( Pl ), dom_node_att2( A )
not_missing_log( A, Pl )     :- notin node( _, A ), dom_post_att2l( Pl ), dom_node_att2( A )
not_missing_log( A, Pl )     :- log( A, Pl ), dom_post_att2( Pl ), dom_node_att2( A )

dom_post_att2( “hello” ) ;
dom_node_att2( “a” ) ;
dom_node_att2( “b” ) ;
dom_node_att2( “c” ) ;
```

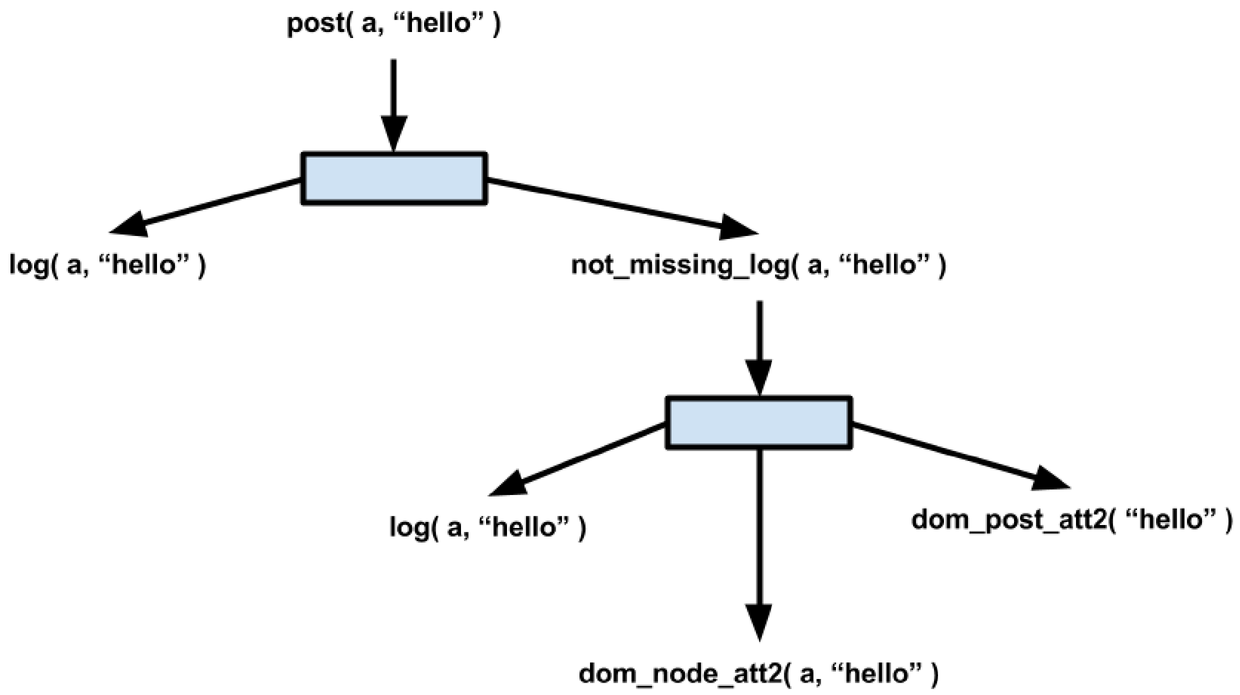
Observations :

The subgoals “***dom_post_att_pl***” and “***dom_node_att_a***” are unary EDBs in ***P'*** populated by skimming the data in the indicated attribute from the corresponding relation realizations output by the preceding evaluation of ***P***. For example, ***dom_post_att2*** contains the entire range of values in the second attribute of the ***post*** relation, which consists of the range of payloads relevant to the creation of all post tuples in the evaluation of ***P***. Similarly, ***dom_node_att2*** contains the entire range of values in the second attribute of the ***node*** relation, which consists of the range of destination nodes relevant to the creation of all ***missing_log*** tuples in the evaluation of ***P***. The latter domain constraint is interesting as a deviation from previous discussions focusing on domain push-downs from parent to child rules as the sole propagation mechanism for domain constraint information through the rewritten program.

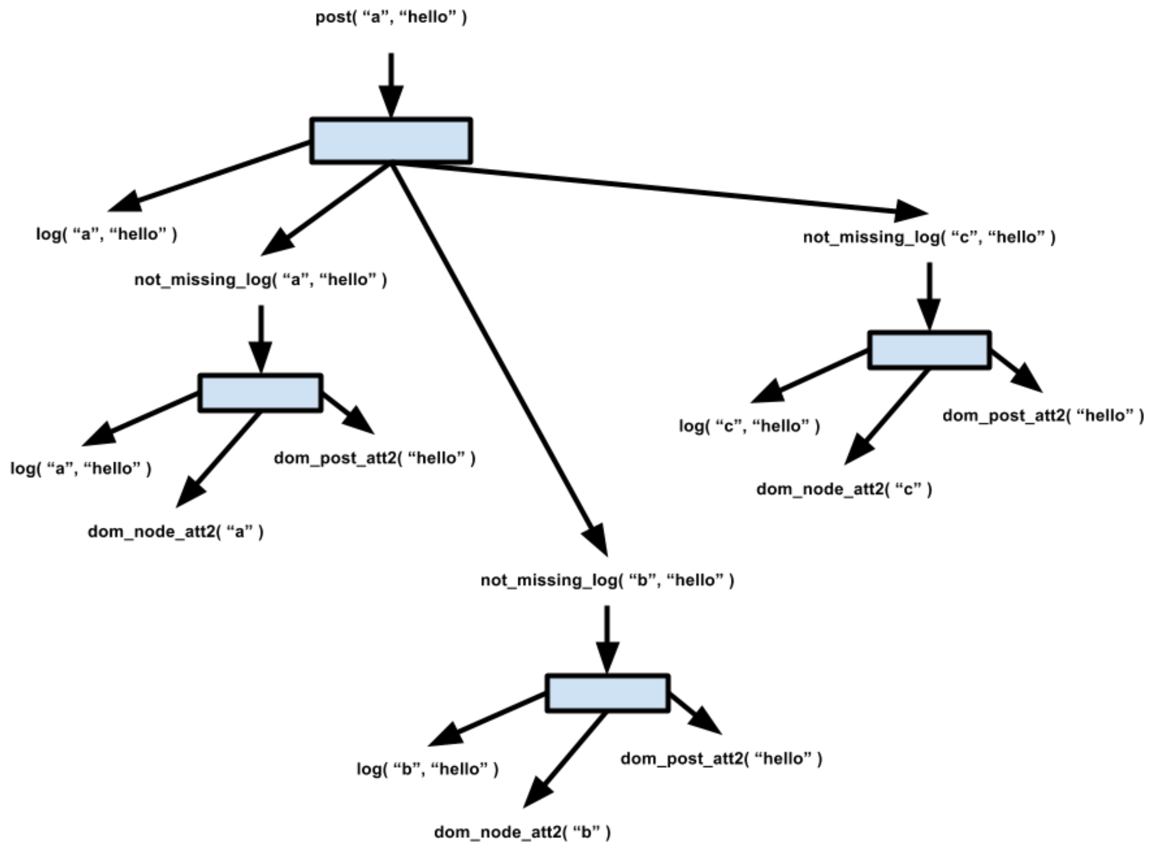
The rewrites also encompass a number of important safety precautions. Each new rule is safe as a result of appending subgoals constraining the domains of attributes of interest, which manifest exclusively as the goal attributes. Accordingly, no goal attribute will ever not be defined by some positive subgoal per rule. Additionally, existential variables not involved in valid joins per new rule are converted into wildcards. As a result, no existential attributes will ever appear solely in the set of negated subgoals within a new rule.

Example *simplog* ***Provenance Tree with NegativeWrites :***
*Not displaying positive log provenance for simplicity of illustration.

Old prov graph :



Prov graph v2 :



//EOF