Call parse analysis

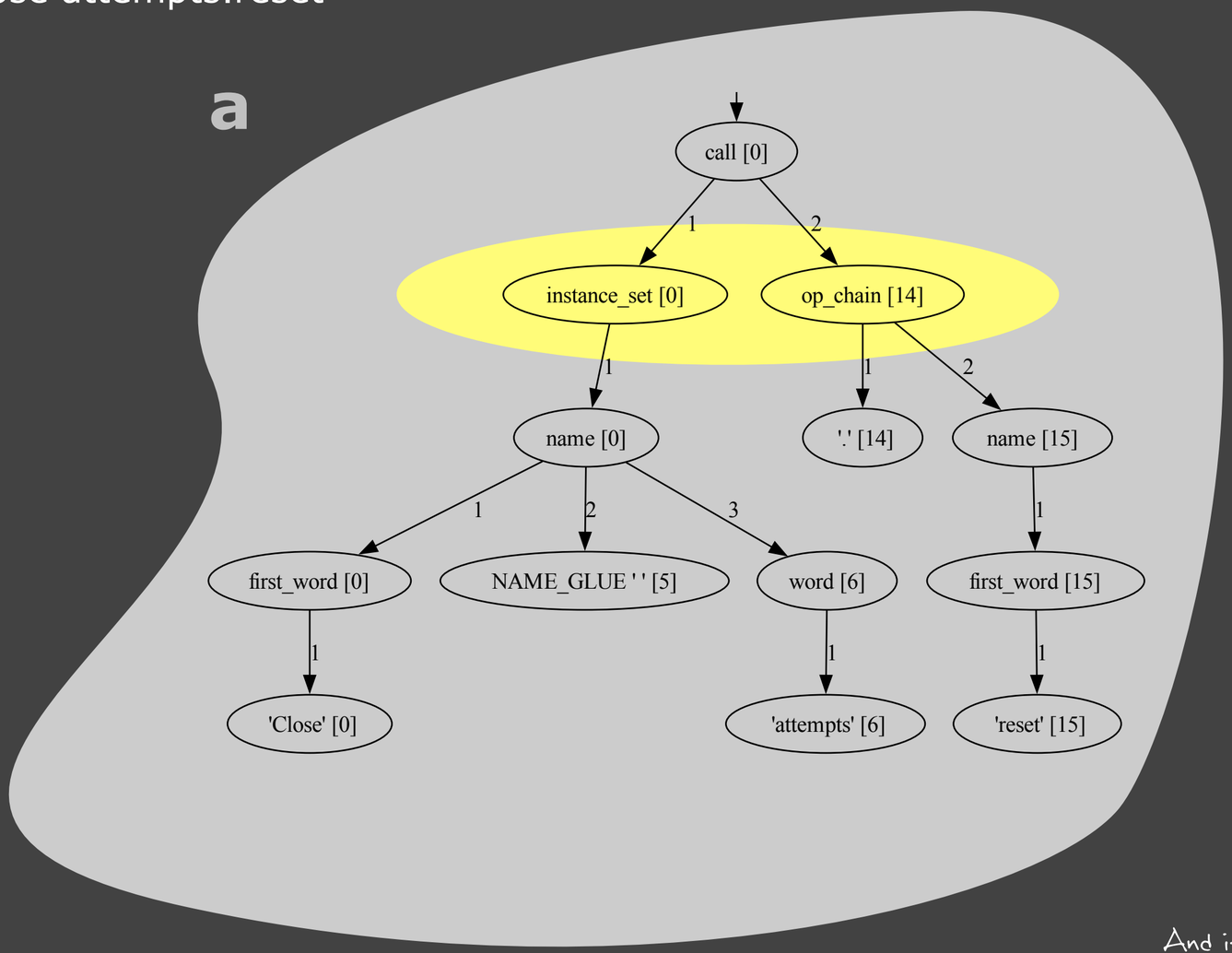
Leon Starr
2025-10-17/ v0.8

Here we study the Scrall parser tree and named tuples resulting from a parse of a variety of operation chains following an attribute write action so that we can figure out how to populate the actions in the xUML populate tool

Copyright © 2025, Leon Starr at
MODEL INTEGRATION, LLC

```
> statement = {Call_a} Call_a(call=N_a(name='Close attempts'), op_chain=Op_chain_a(components=[N_a(name='reset')]))
```

Close attempts.reset



The key idea is that if a call begins with an attribute (qualified or unqualified), we are using a type action to write to that attribute. We know this because we aren't doing an assignment, so converting the value of an attribute has no effect if it is not written back to that same attribute.

At this point, Close attempts could be either an attribute name, (which it is, in our example) OR the name of a single instance flow qualifying the attribute name in the first component element of the op_chain field.

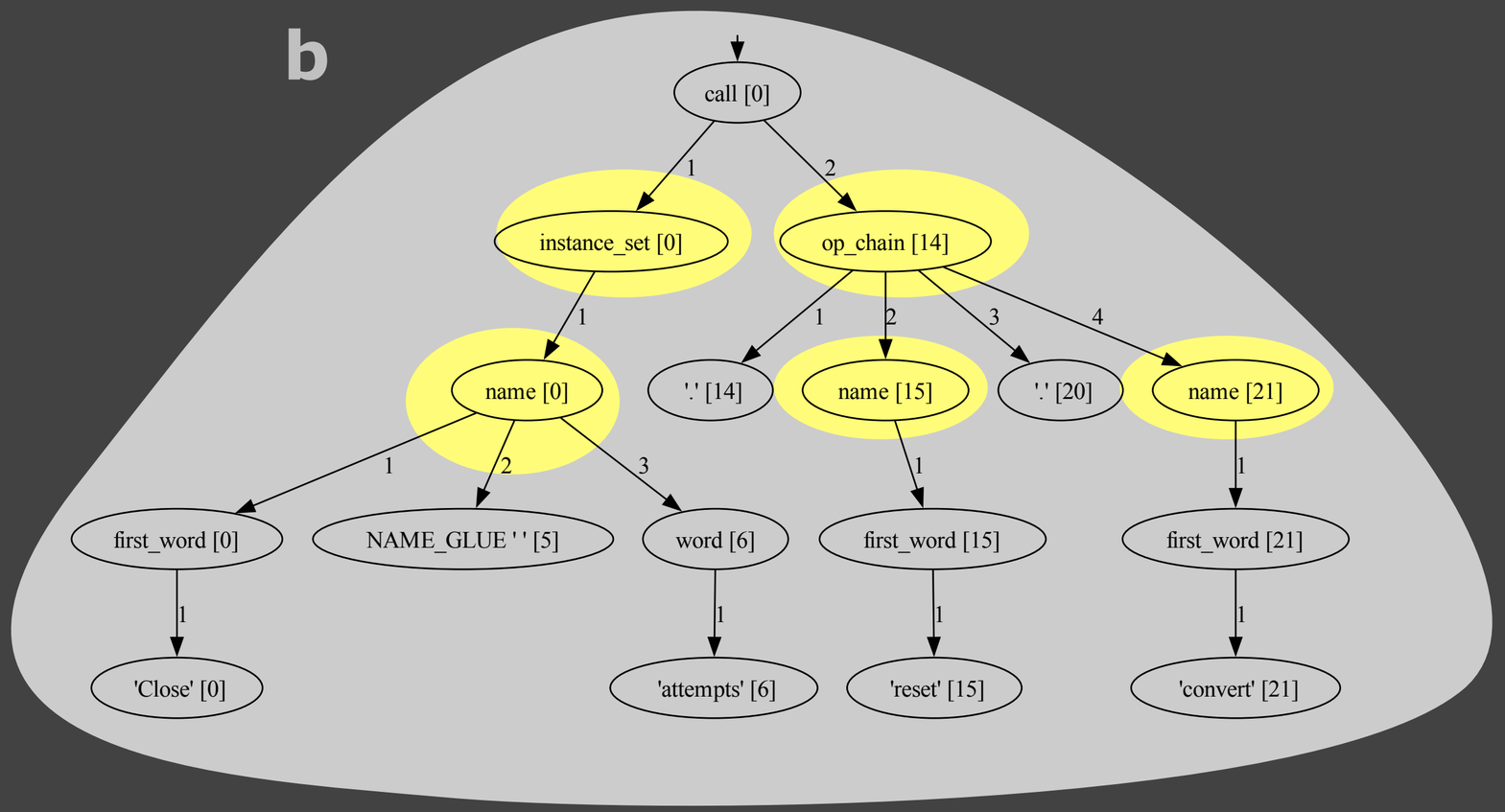
It's not the name of a method since we don't see any () which are required in the Scrall syntax for a method call, even if no parameters are supplied. And definitely not an external service since the ~ symbol would force an INST_a tuple instead of an N_a in the call field.

And if we AREN'T starting off with an attribute, a method or external service is being invoked via an INST_a

This means that if we have a call starting off with and N_a or IN_a tuple and we don't detect a qualified or unqualified attribute, this is an error.

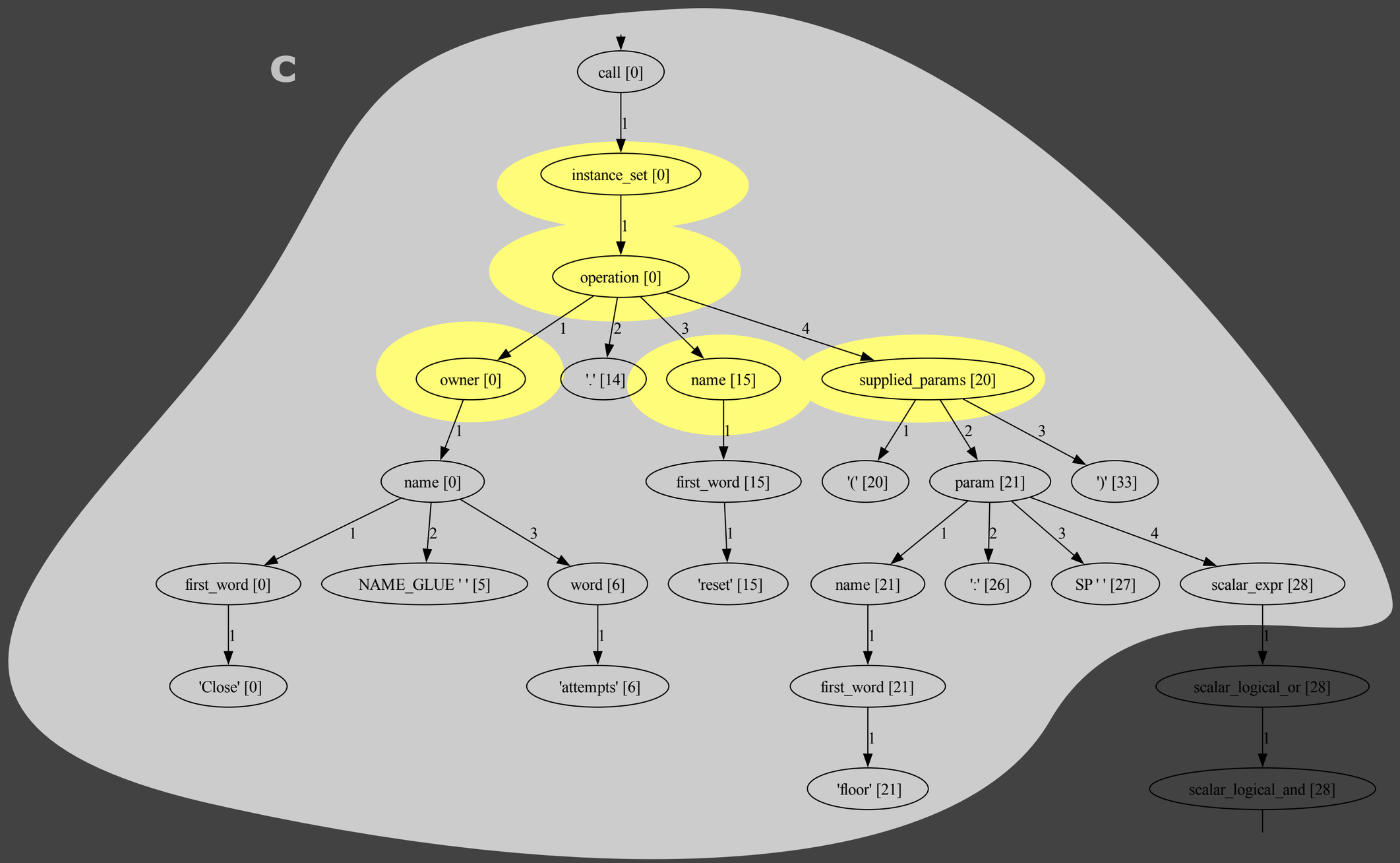
Close attempts.reset.convert

```
call = {N_a} N_a(name='Close attempts')
op_chain = {Op_chain_a} Op_chain_a(components=[N_a(name='reset'), N_a(name='convert')])
```



Close attempts.reset(floor: level)

```
call = {INST_a} INST_a(components=[Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))])])
  components = {list} [Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))])]
    0 = {Op_a} Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))])
      op_name = {str} 'reset'
      owner = {str} 'Close attempts'
      supplied_params = {list} [Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))]
```



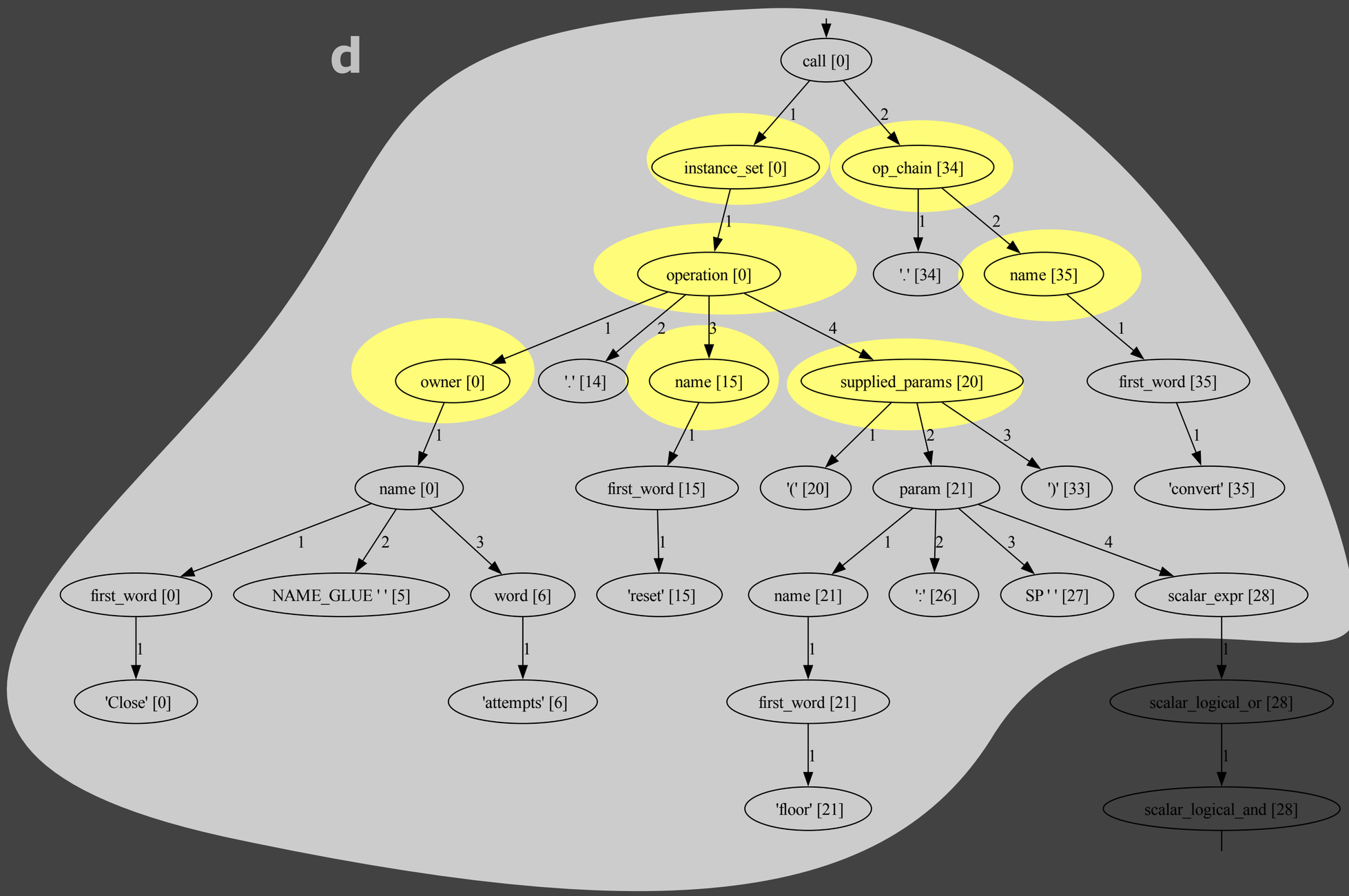
Close attempts.reset(floor: level).convert

call = {INST_a} INST_a(components=[Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))])])

components = {list} [Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='floor', sval=N_a(name='level'))])]

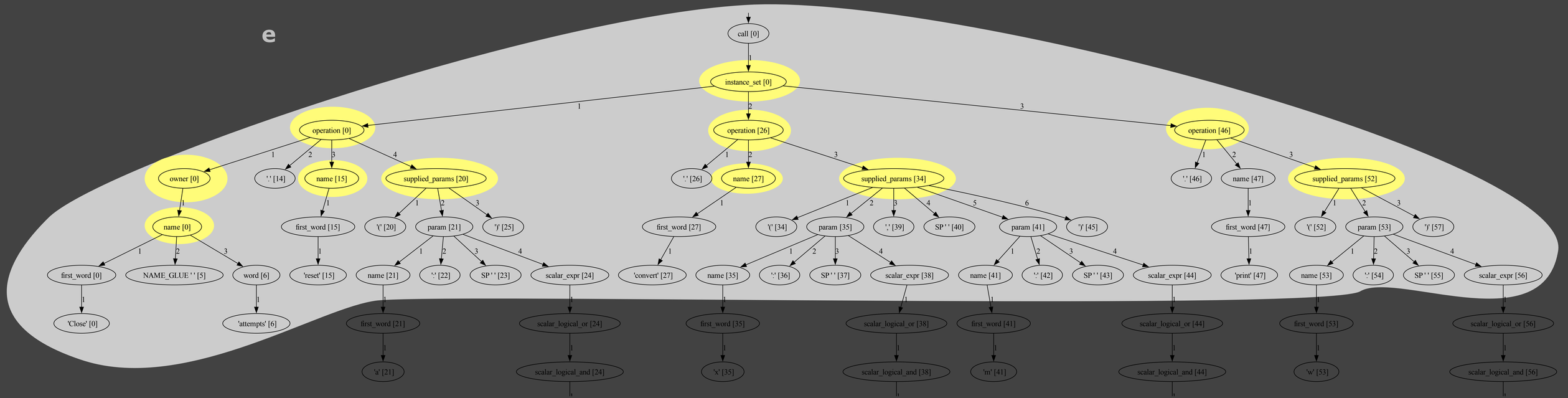
op_chain = {Op_chain_a} Op_chain_a(components=[N_a(name='convert')])

d



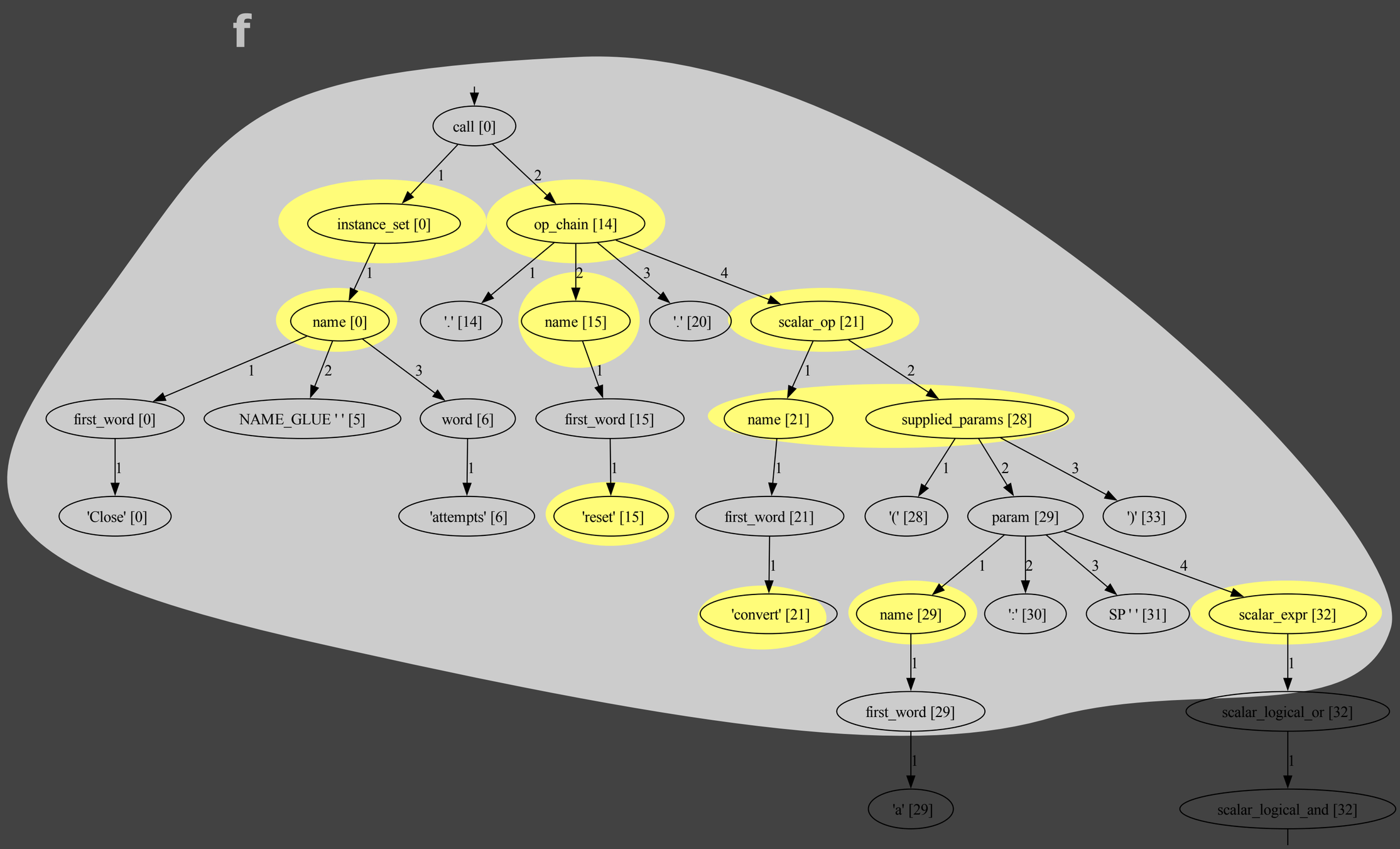
Close attempts.reset(a: b).convert(x: y, m: n).print(w: z)

```
call = {INST_a} INST_a(components=[Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(nam...name='n'))]), Op_a(owner='_implicit', o
  components = {list} [Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))]), Op_a(owner='_implicit', op_name='convert',
    0 = {Op_a} Op_a(owner='Close attempts', op_name='reset', supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))])
    1 = {Op_a} Op_a(owner='_implicit', op_name='convert', supplied_params=[Supplied_Parameter_a(pname='x', sval=N_a(name='y')), Supplied_Parameter_a(pname='m', sval=N_a(name='n'))])
    2 = {Op_a} Op_a(owner='_implicit', op_name='print', supplied_params=[Supplied_Parameter_a(pname='w', sval=N_a(name='z'))])
op_chain = {NoneType} None
```



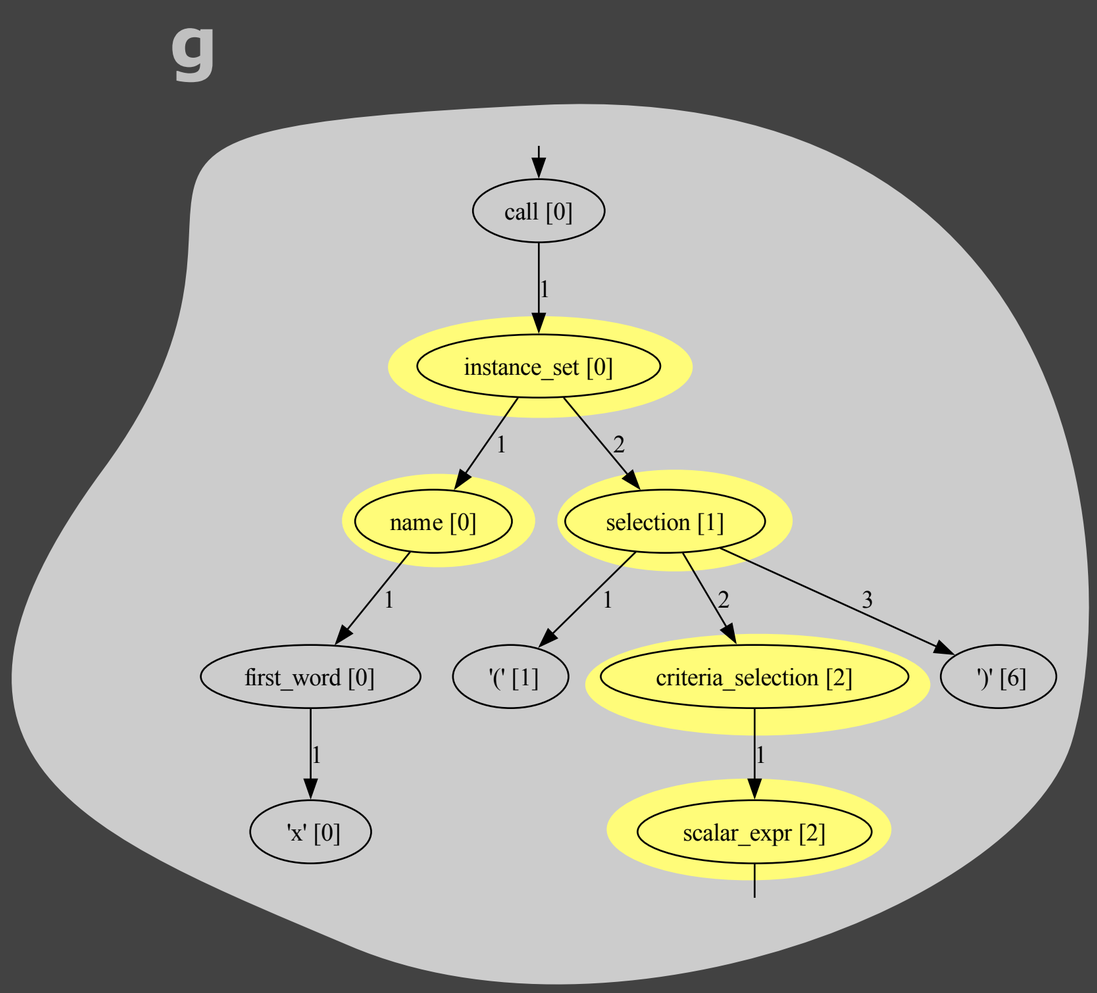
Close attempts.reset.convert(a: b)

```
call = {N_a} N_a(name='Close attempts')
op_chain = {Op_chain_a} Op_chain_a(components=[N_a(name='reset'), Scalar_op_a(name=N_a(name='convert'), supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))])])
  components = {list} [N_a(name='reset'), Scalar_op_a(name=N_a(name='convert'), supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))])]
    0 = {N_a} N_a(name='reset')
    1 = {Scalar_op_a} Scalar_op_a(name=N_a(name='convert'), supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))])
```



x(a: b)

```
call = {INST_a} INST_a(components=[N_a(name='x'), Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='a'), N_a(name='b')]))])
  components = {list} [N_a(name='x'), Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='a'), N_a(name='b')]))]
  0 = {list} [N_a(name='x'), Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='a'), N_a(name='b')]))]
  __len__ = {int} 1
  Protected Attributes
op_chain = {NoneType} None
```



ERROR!

Here we see a Criteria Selection, but you can't do that without being on the RHS of an assignment.

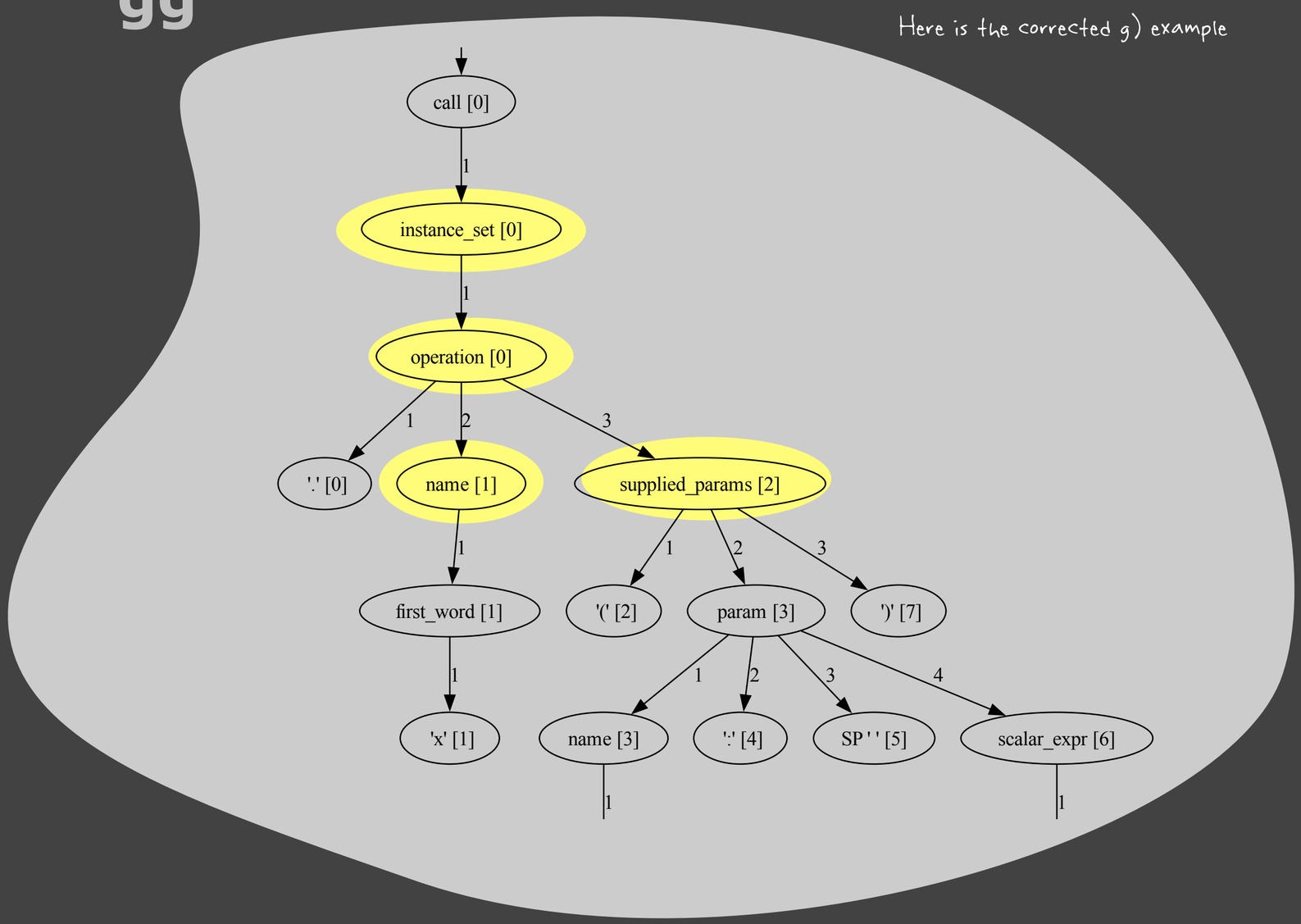
Scrall requires that you preface an unqualified method invocation with a '.'. So it should be .x(a:b)

.x(a: b)

```
call = {INST_a} INST_a(components=[Op_a(owner='_implicit', op_name='x', supplied_params=[Supplied_Parameter_a(pname='a', sval=N_a(name='b'))])])
op_chain = {NoneType} None
```

gg

Here is the corrected g) example

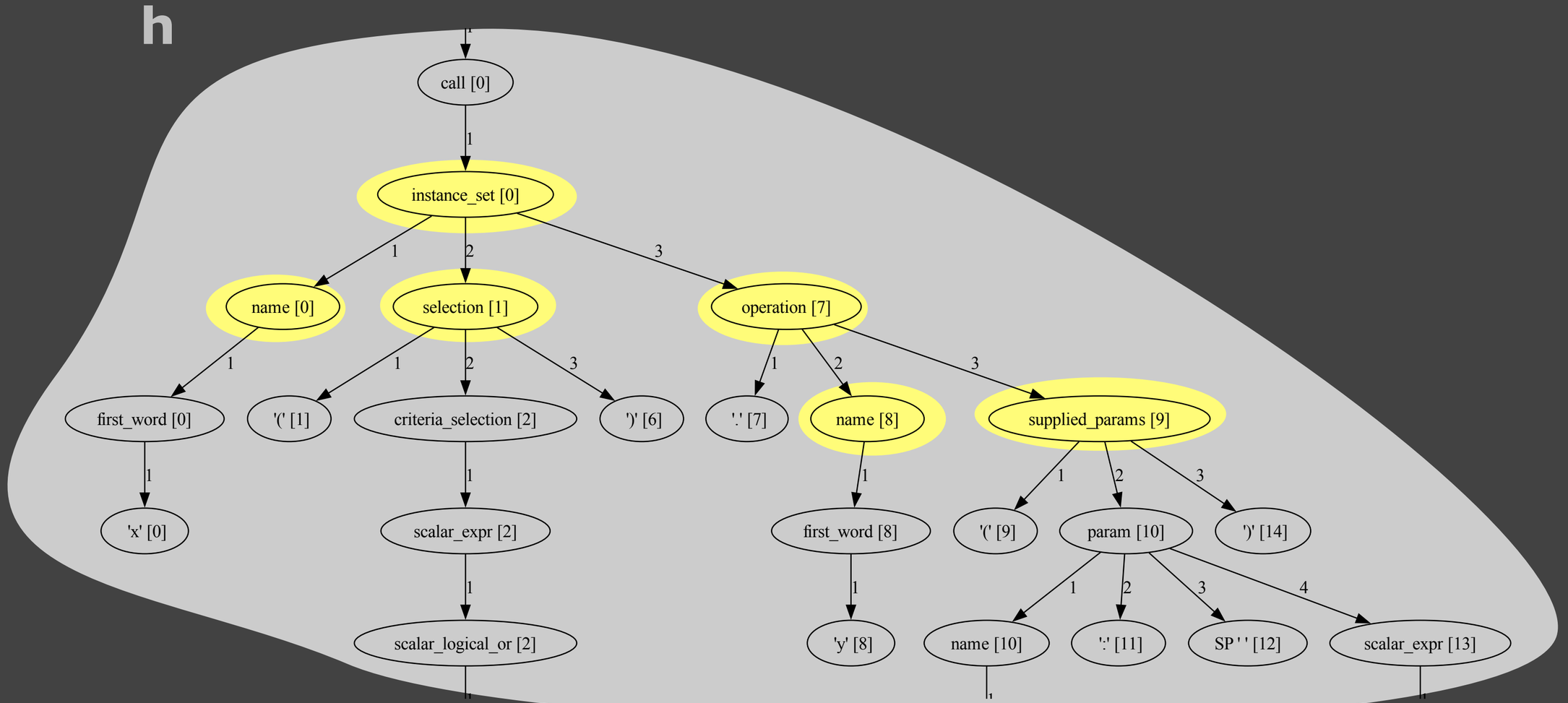


x(a: b).y(c: d)

```
call = {INST_a} INST_a(components=[N_a(name='x'), Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='
  components = {list} [N_a(name='x'), Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='a'), N_a(name
    0 = {N_a} N_a(name='x')
    1 = {Criteria_Selection_a} Criteria_Selection_a(card='ALL', criteria=BOOL_a(op='==', operands=[N_a(name='a'), N_a(name='b')]))
    2 = {Op_a} Op_a(owner='_implicit', op_name='y', supplied_params=[Supplied_Parameter_a(pname='c', sval=N_a(name='d'))])
```

```
op_chain = {NoneType} None
```

h



/R4/Door.Lock requested.set

```
> call = {INST_a} INST_a(components=[PATH_a(hops=[R_a(rnum='R4'), N_a(name='Cabin')])])
> op_chain = {Op_chain_a} Op_chain_a(components=[N_a(name='Ping')])
```

```
> call = {INST_a} INST_a(components=[PATH_a(hops=[R_a(rnum='R4'), N_a(name='Door')])])
> op_chain = {Op_chain_a} Op_chain_a(components=[N_a(name='Lock requested'), N_a(name='set')])
```

i