A Fine-Grained Small-Step Semantics for Interleaving Search

FLOPS 2026

Brysen Pfingsten Jason Hemann

Seton Hall University

{**pfingsbr**, hemannja}@shu.edu

Interleaving DFS

Interleaving DFS

$$\texttt{DOGS}(x) \leftarrow (x = \texttt{dog}) \vee \texttt{DOGS}(x)$$

$$\texttt{CATS}(x) \leftarrow (x = \texttt{cat}) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(x) \leftarrow \texttt{DOGS}(x) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(q)$$

Interleaving DFS

$$\texttt{DOGS}(x) \leftarrow (x = \texttt{dog}) \vee \texttt{DOGS}(x)$$

$$\texttt{CATS}(x) \leftarrow (x = \texttt{cat}) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(x) \leftarrow \texttt{DOGS}(x) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(q)$$

DFS (Prolog)

```
dog, dog, dog, dog, ...
```

Interleaving DFS

$$\texttt{DOGS}(x) \leftarrow (x = \texttt{dog}) \vee \texttt{DOGS}(x)$$

$$\texttt{CATS}(x) \leftarrow (x = \texttt{cat}) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(x) \leftarrow \texttt{DOGS}(x) \vee \texttt{CATS}(x)$$

$$\texttt{DOGS-CATS}(q)$$

DFS (Prolog)

```
dog, dog, dog, dog, ...
```

Interleaving DFS (miniKanren)

```
dog, cat, dog, cat, ...
```

Interleaving DFS

$$\texttt{SAME}(x, y) \leftarrow (x = y)$$

$$(\texttt{SAME}(q, \texttt{turtle}) \vee \texttt{SAME}(q, \texttt{cat}) \vee (q = \texttt{dog})) \vee \texttt{SAME}(q, \texttt{fish})$$

Interleaving DFS

$$\texttt{SAME}(x, y) \leftarrow (x = y)$$

$$(\texttt{SAME}(q, \texttt{turtle}) \vee \texttt{SAME}(q, \texttt{cat}) \vee (q = \texttt{dog})) \vee \texttt{SAME}(q, \texttt{fish})$$

DFS (Prolog)

```
turtle, cat, dog, fish
```

Interleaving DFS

$$\texttt{SAME}(x, y) \leftarrow (x = y)$$

$$(\texttt{SAME}(q, \texttt{turtle}) \vee \texttt{SAME}(q, \texttt{cat}) \vee (q = \texttt{dog})) \vee \texttt{SAME}(q, \texttt{fish})$$

DFS (Prolog)

```
turtle, cat, dog, fish
```

Interleaving DFS (miniKanren)

```
fish, turtle, dog, cat
```

Interleaving DFS

- Suspends and resumes evaluation of branches

Interleaving DFS

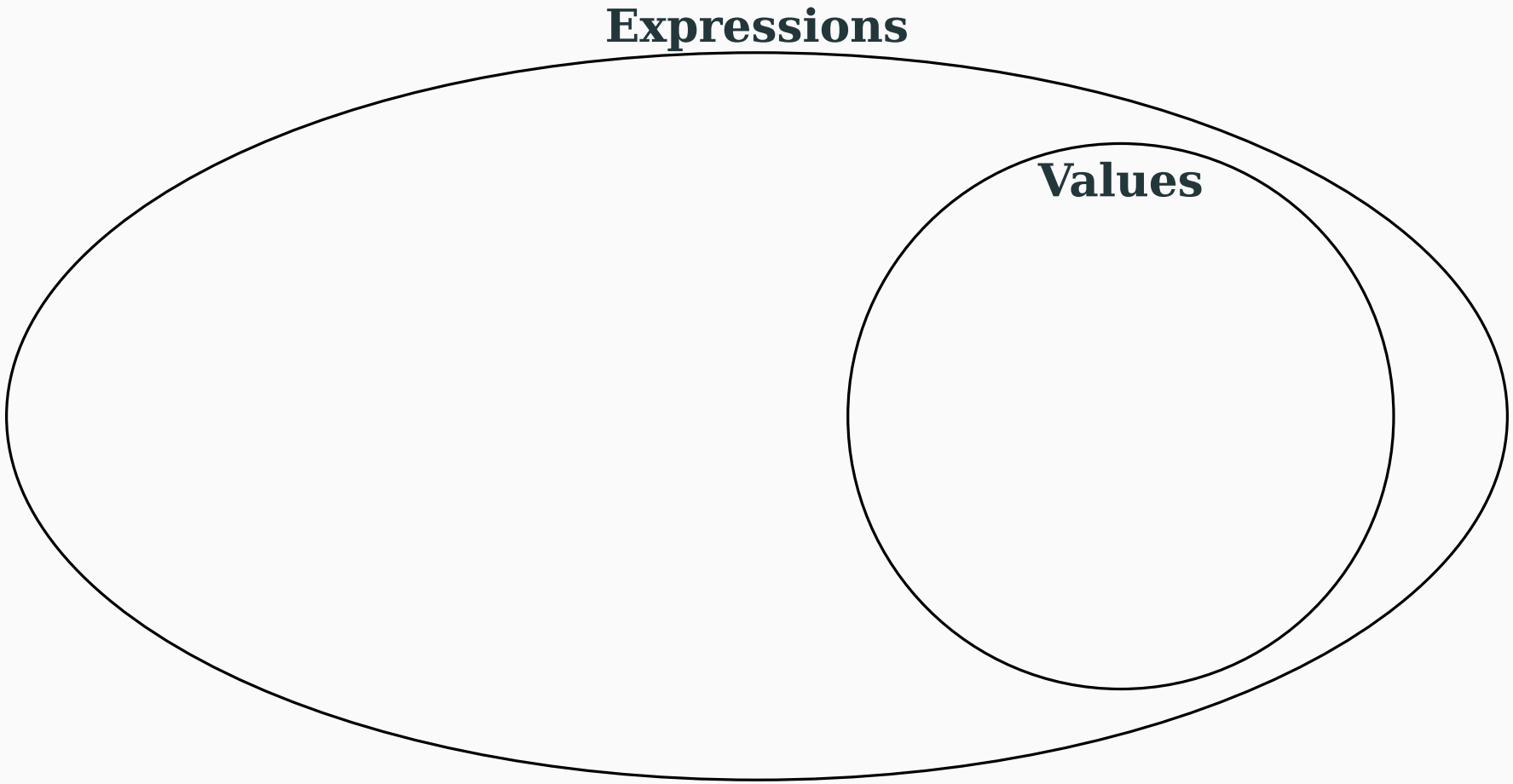
- Suspends and resumes evaluation of branches
- Fair and complete search
  - Kiselyov et al. 2005, Rozplokhas et al. 2020
  - Avoids divergence on an infinite branch

Interleaving DFS

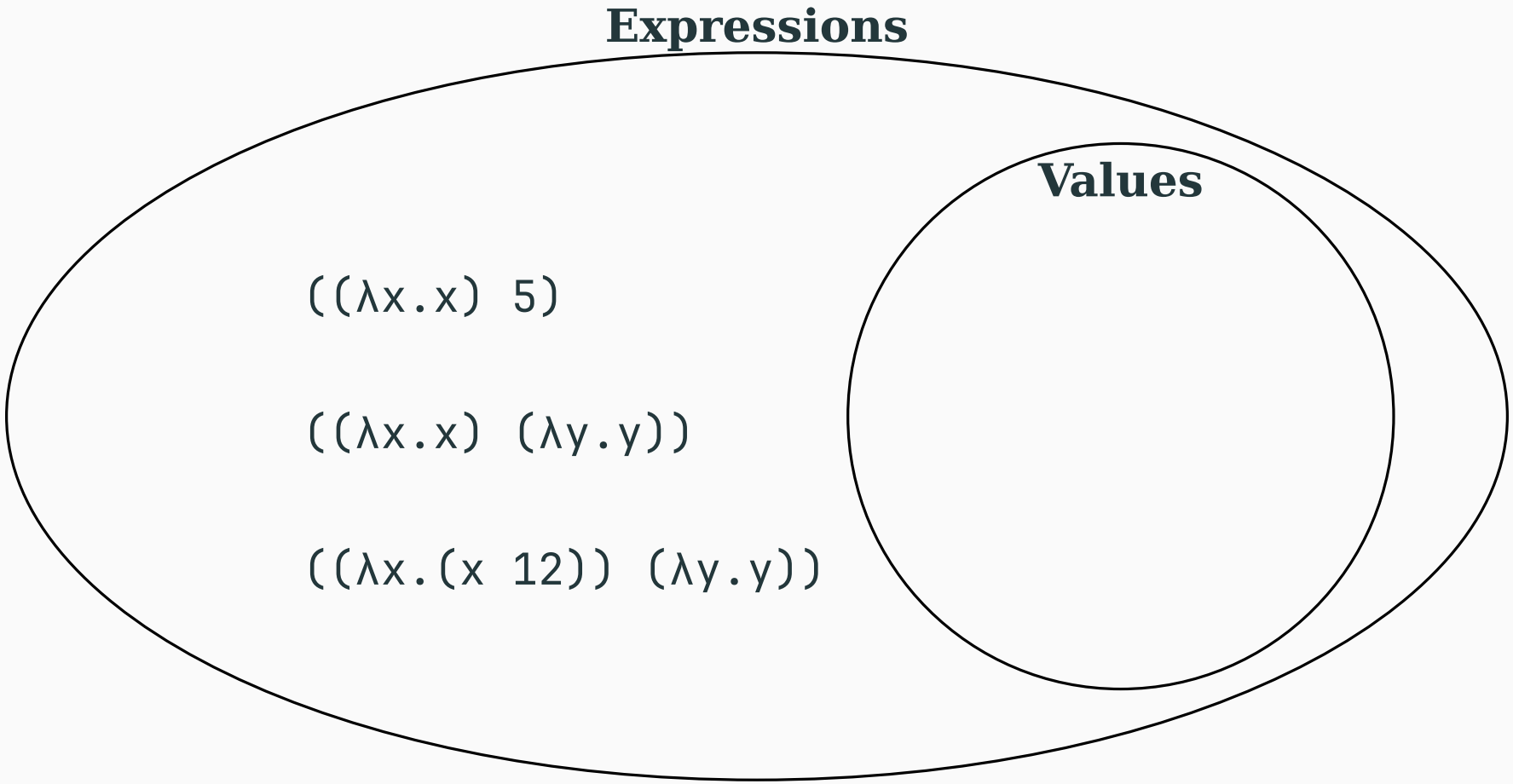
- Suspends and resumes evaluation of branches
- Fair and complete search
  - Kiselyov et al. 2005, Rozplokhas et al. 2020
  - Avoids divergence on an infinite branch
- **Unintuitive behavior**

Calculi, Expressions, Values

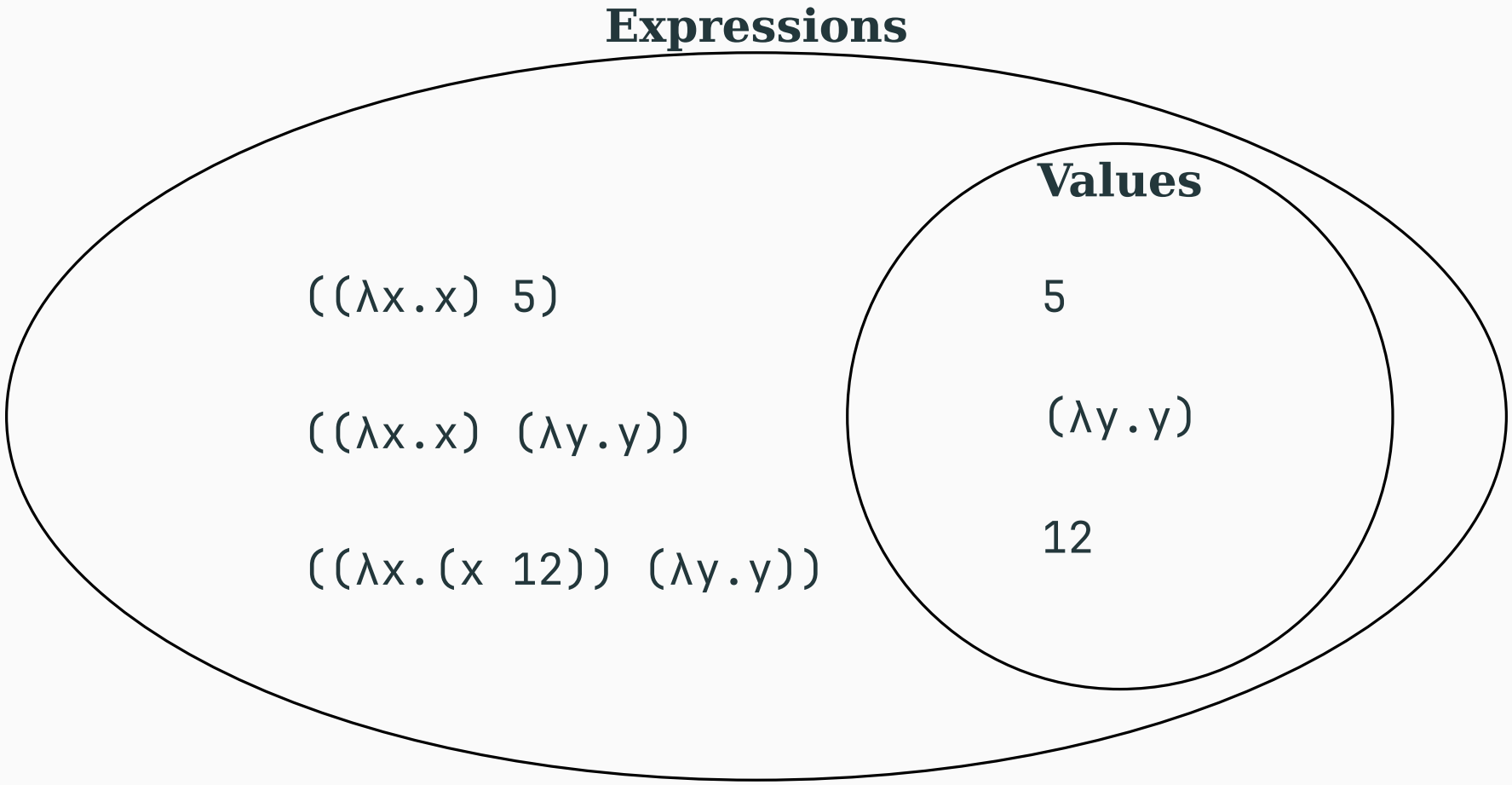
Calculi, Expressions, Values



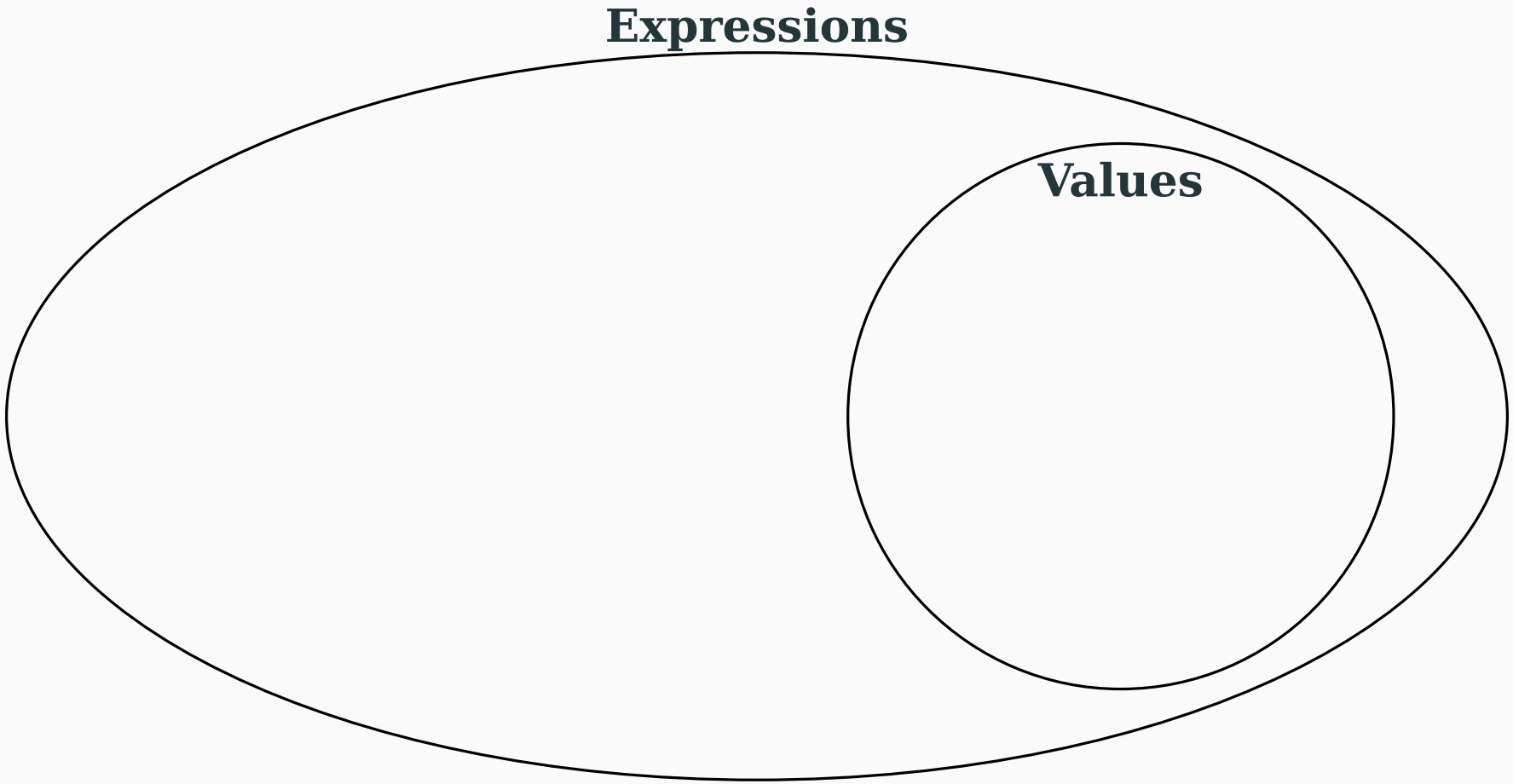
Calculi, Expressions, Values



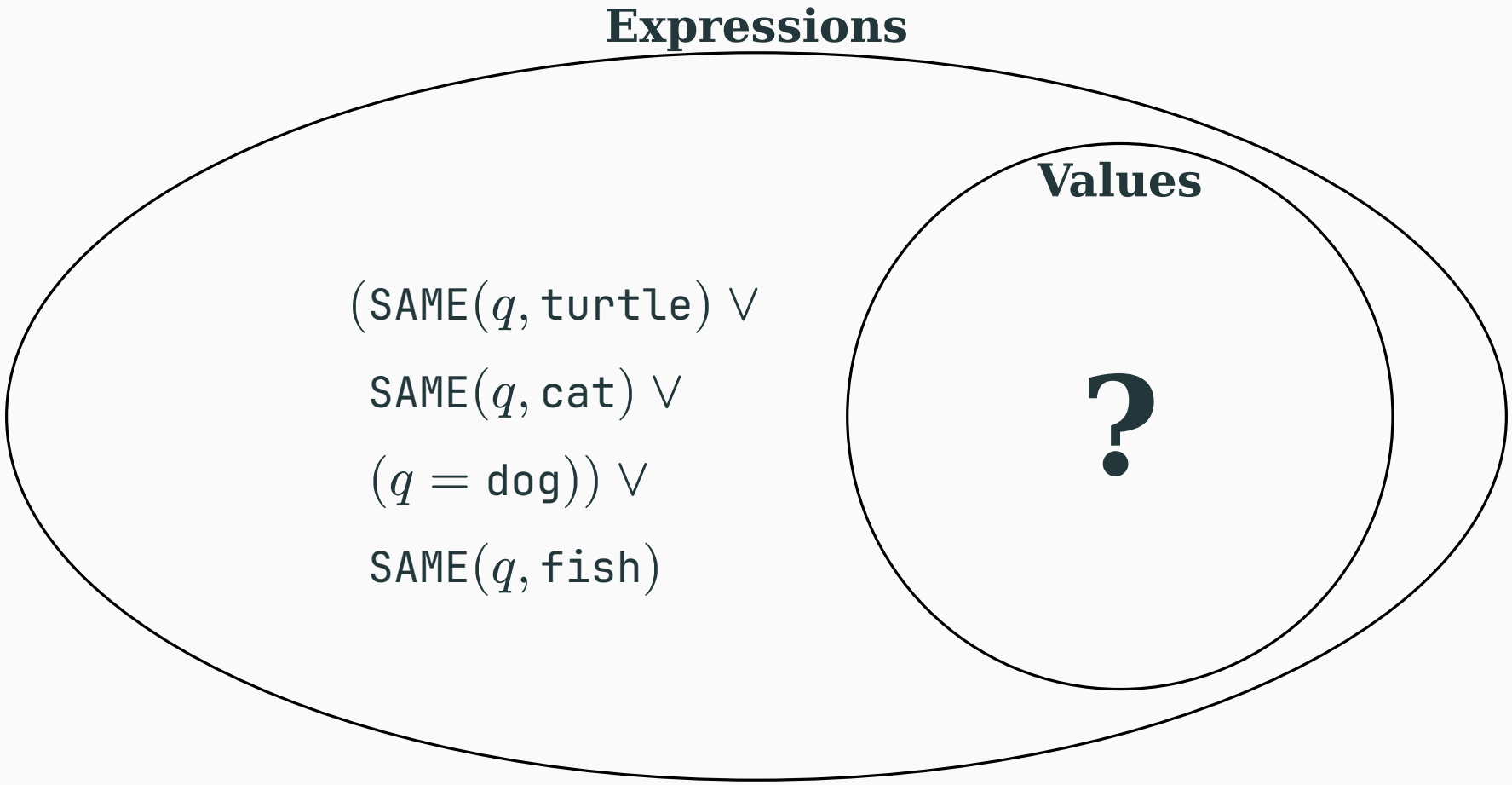
Calculi, Expressions, Values



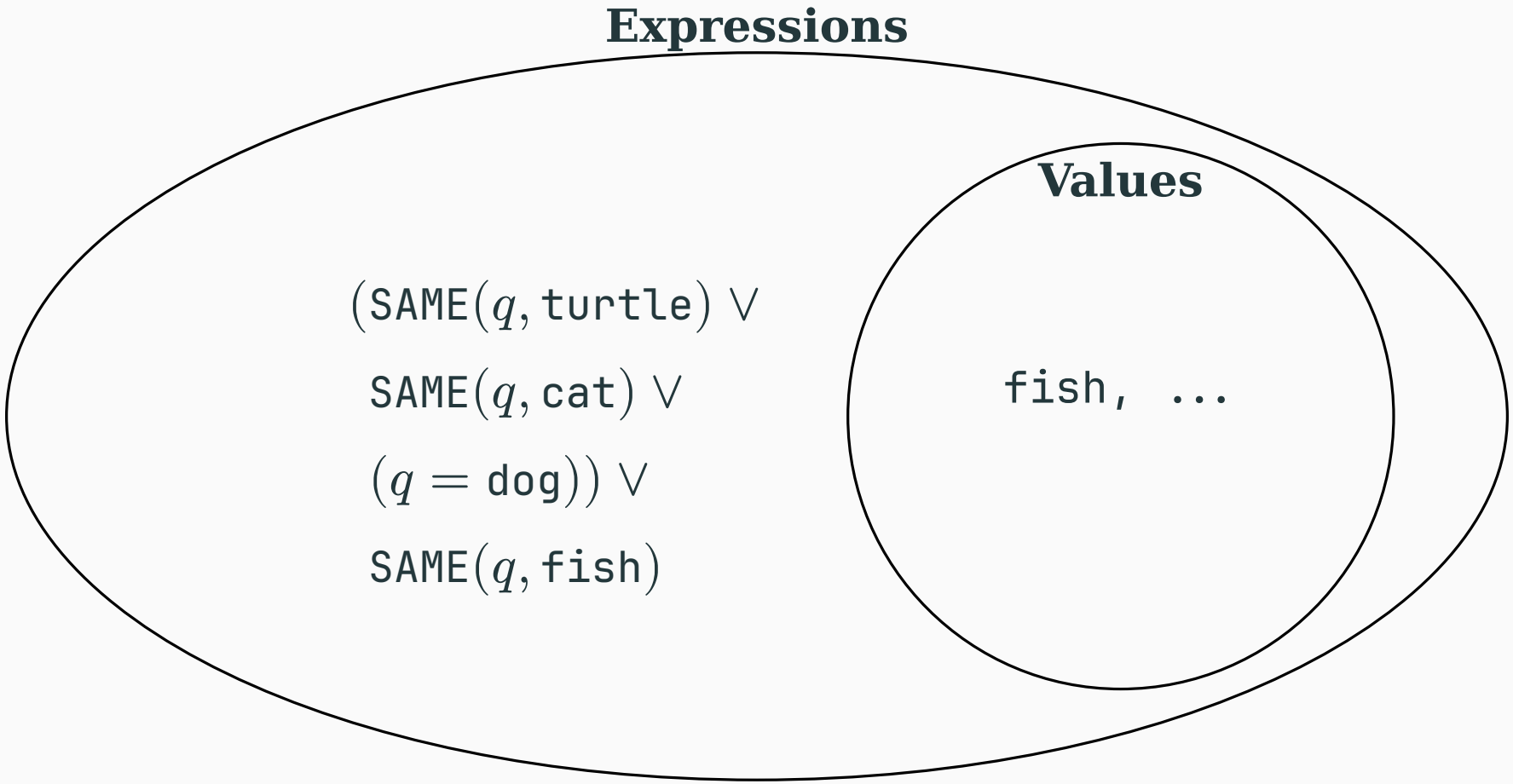
Calculi, Expressions, Values



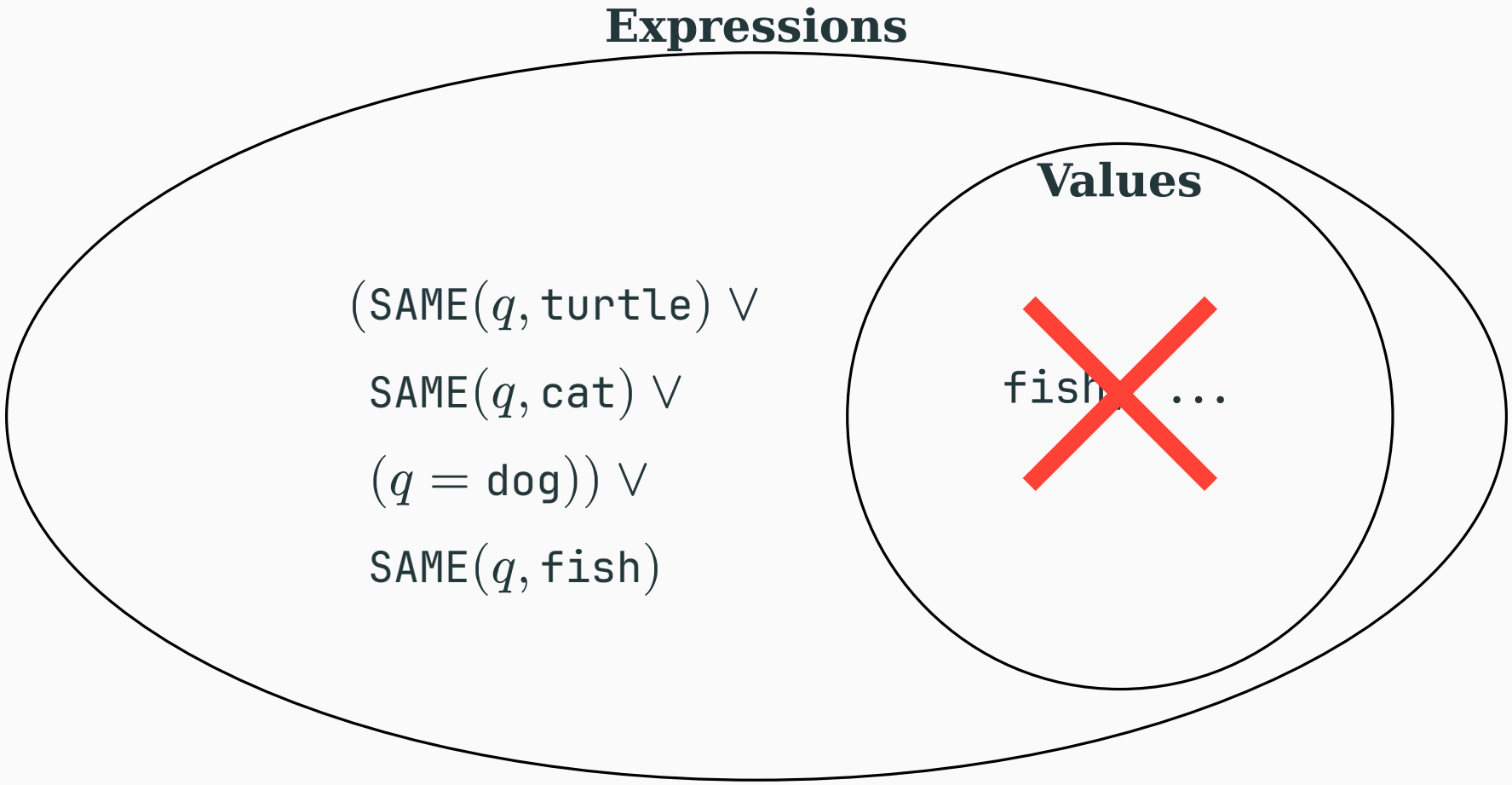
Calculi, Expressions, Values



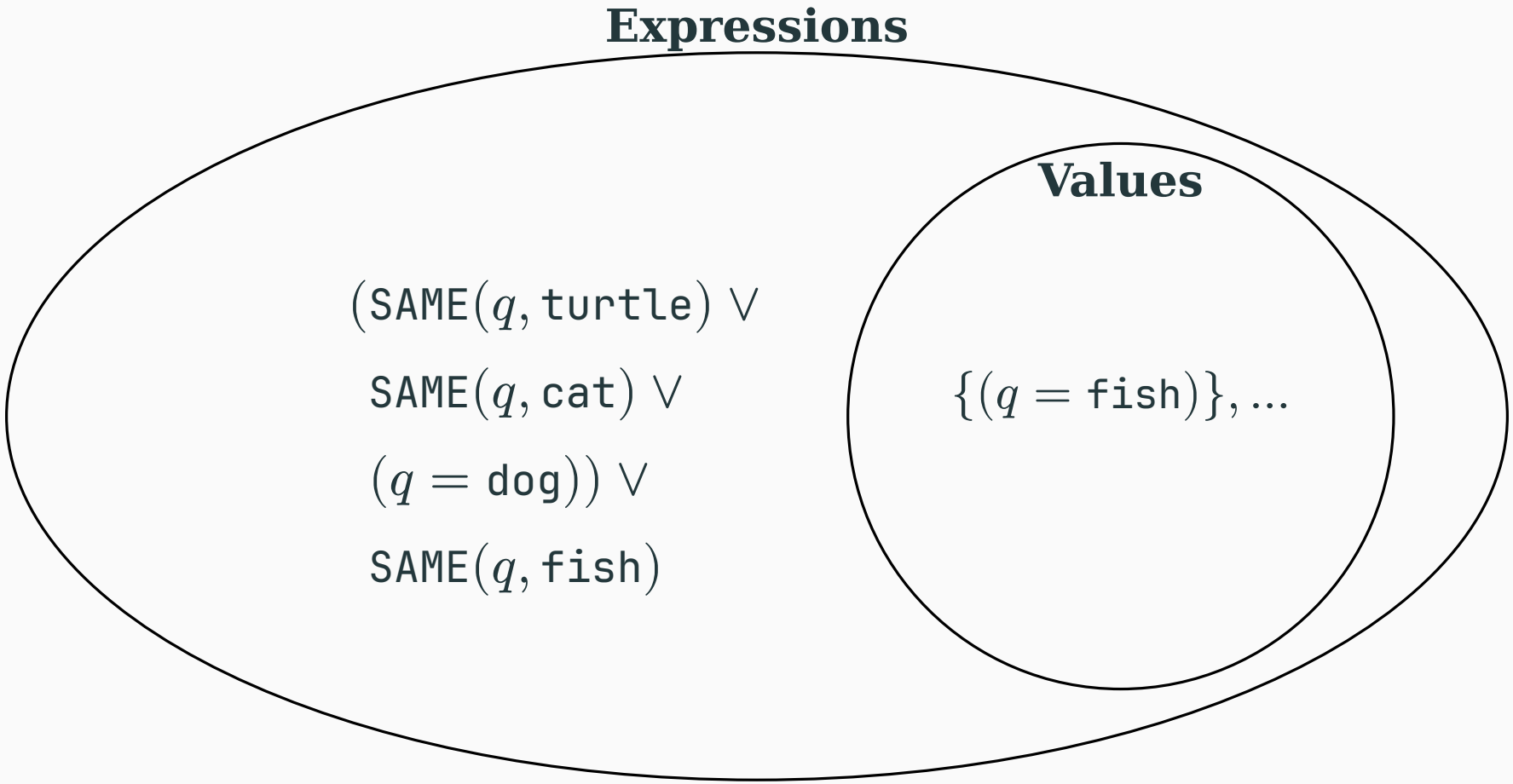
Calculi, Expressions, Values



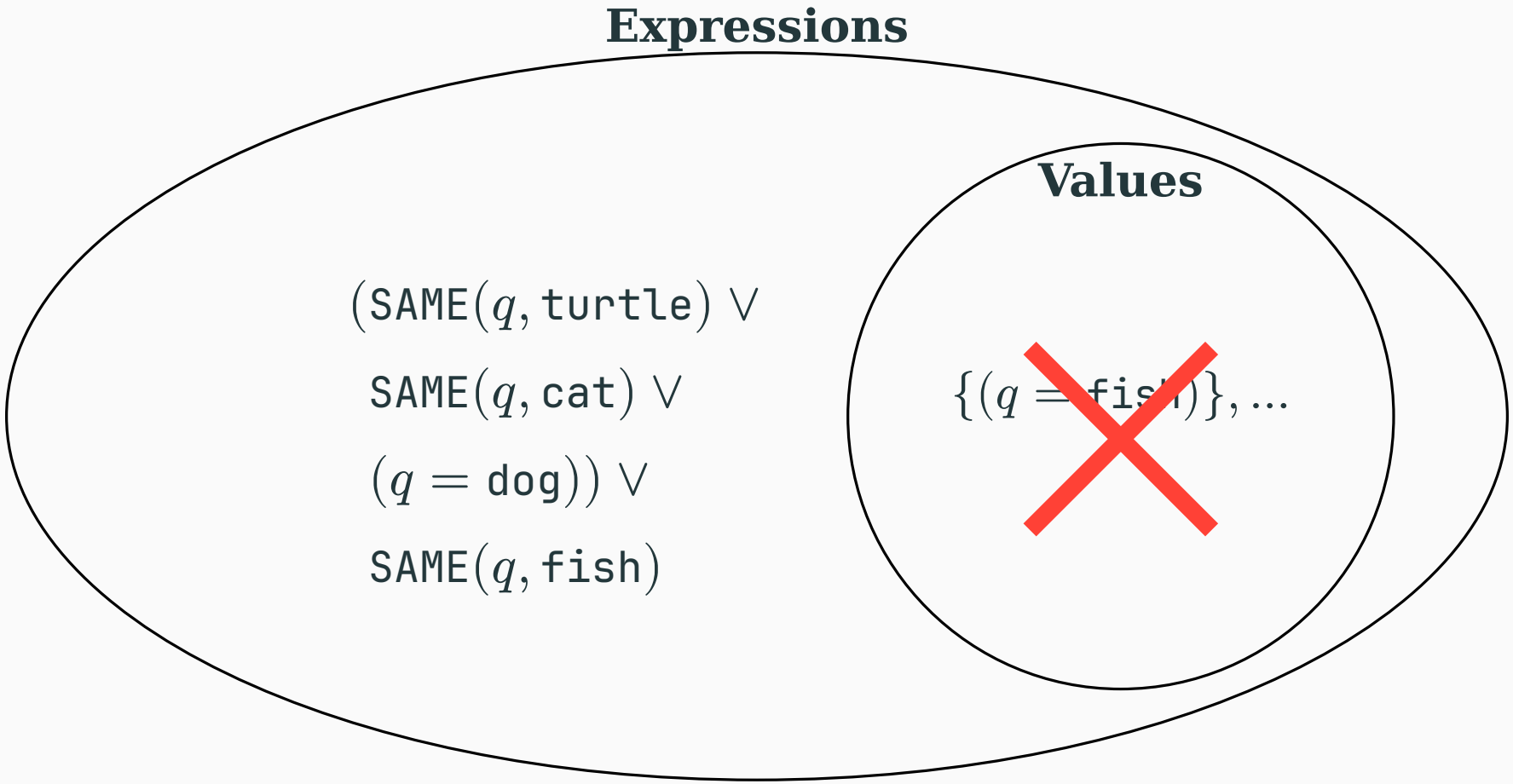
Calculi, Expressions, Values



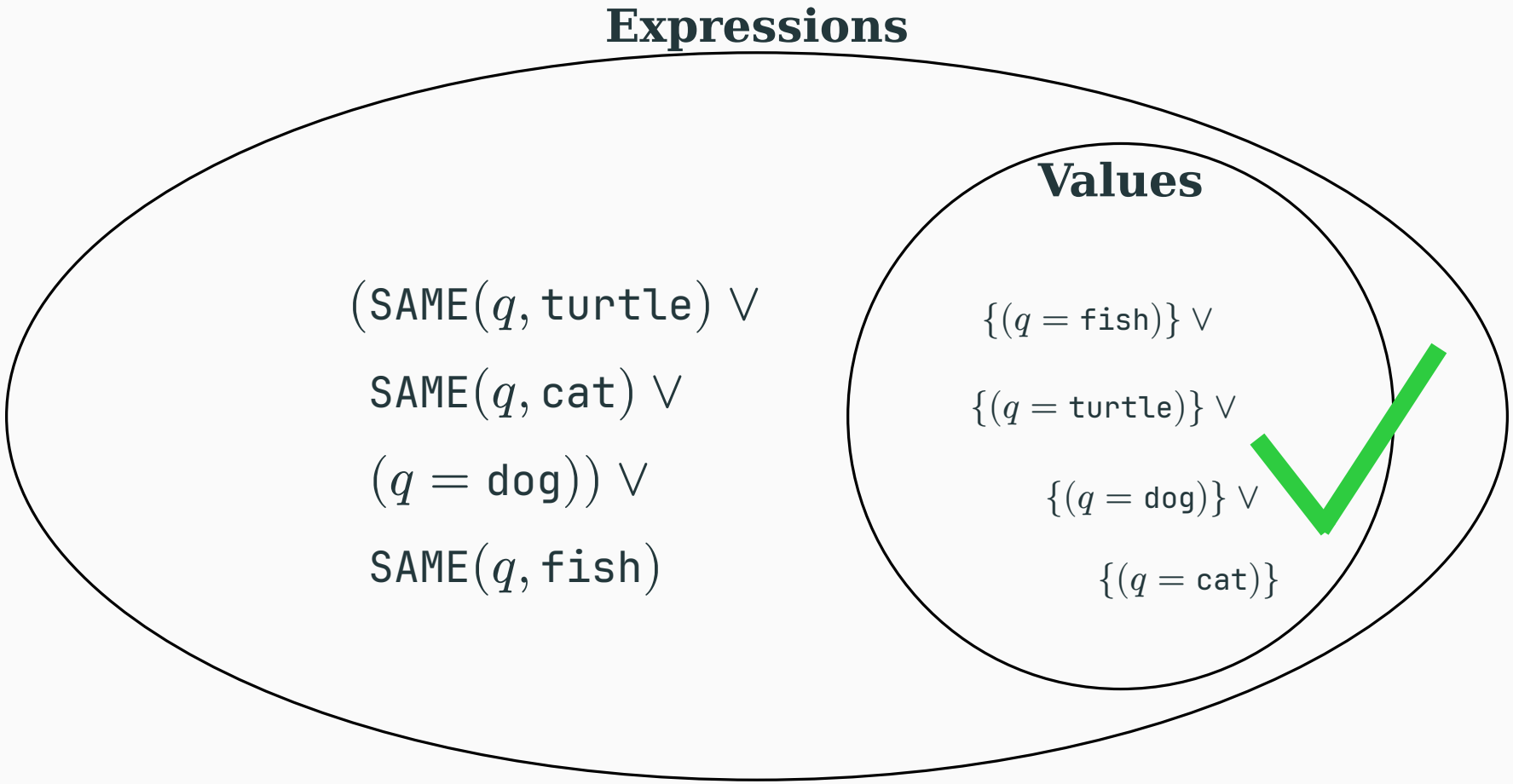
Calculi, Expressions, Values



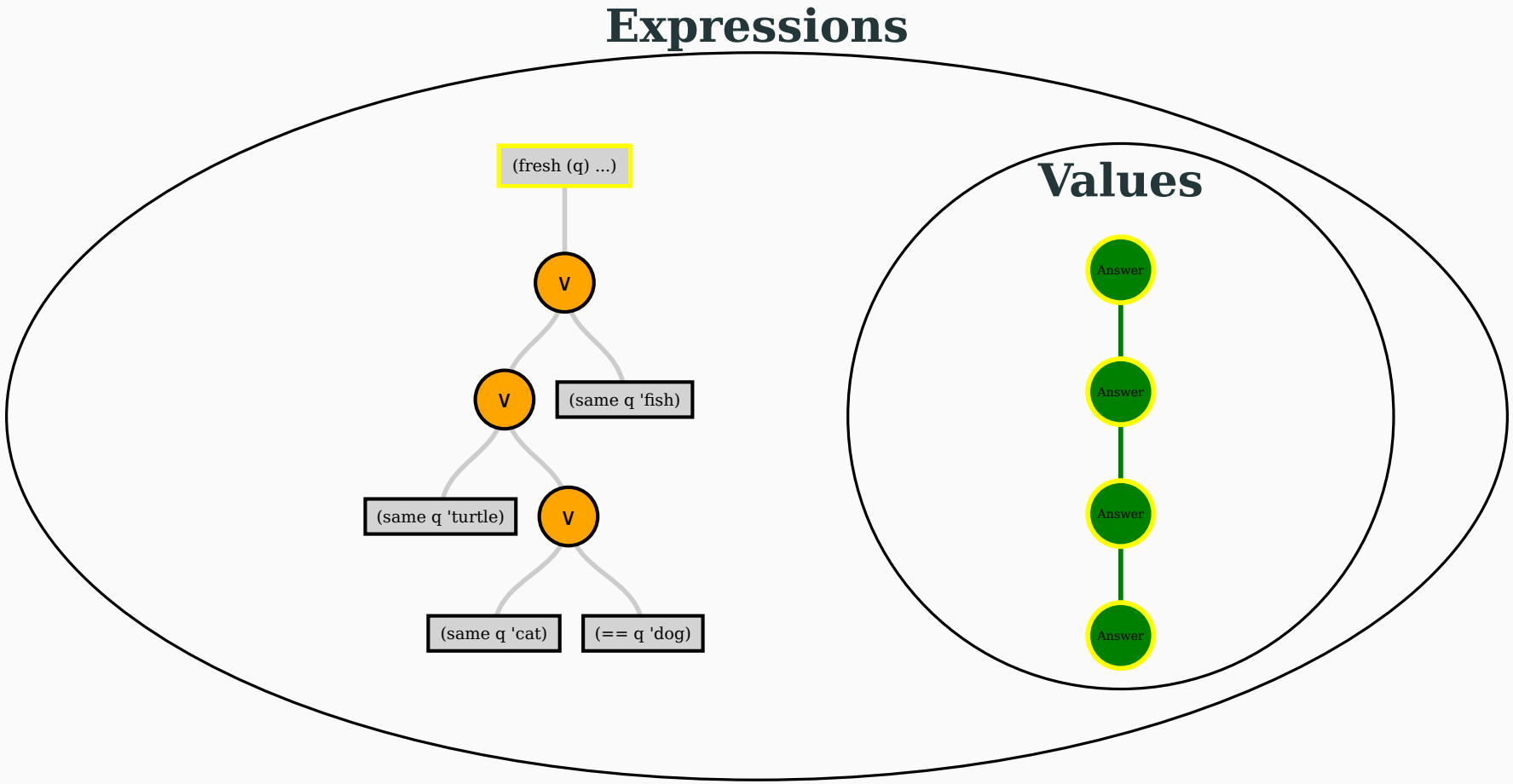
Calculi, Expressions, Values



Calculi, Expressions, Values



Calculi, Expressions, Values



The values are normalized search trees!

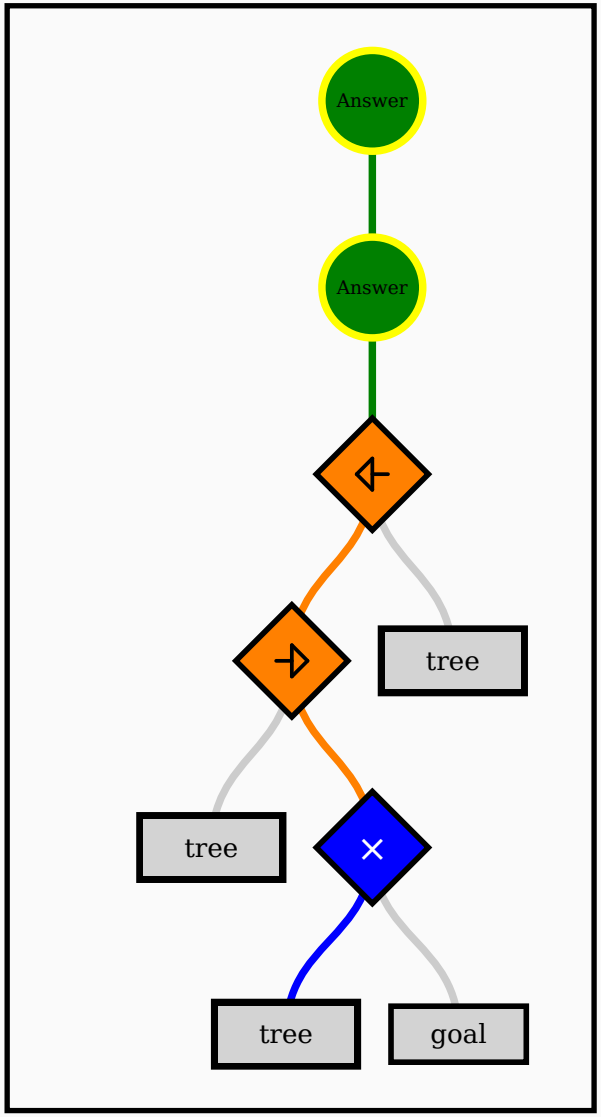
Contributions

Contributions

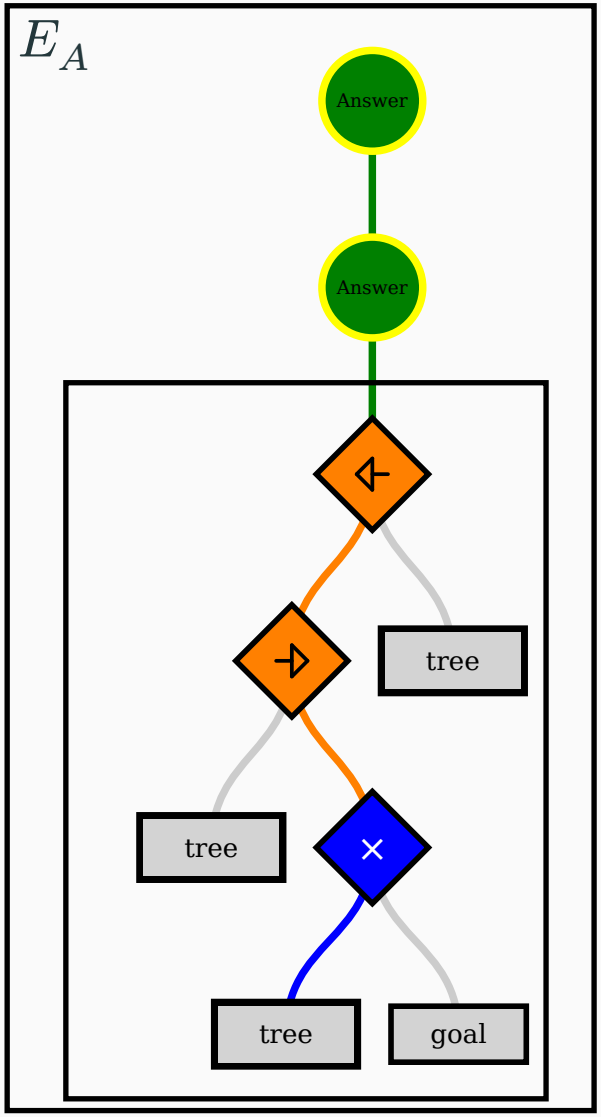
1. Small-Step Operational Semantics for miniKanren
   - Deterministic tree $\rightarrow$ tree rewrites
   - Mechanized and tested in PLT Redex
2. Tree based visual stepper
   - Executable semantics at the core
   - Implemented as a web app

Reduction Semantics

Evaluation Contexts

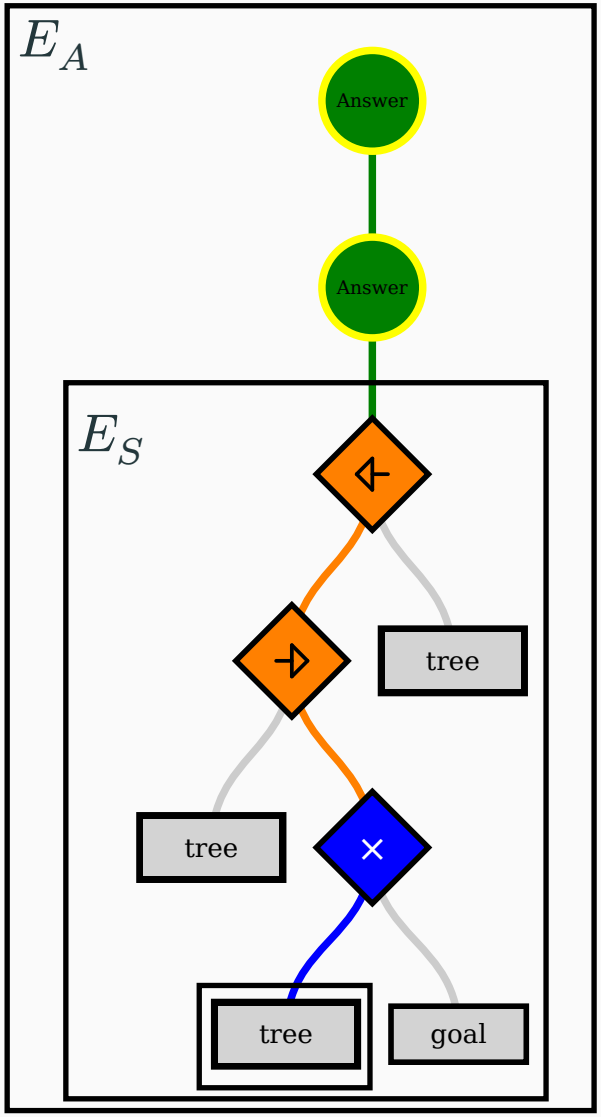


Evaluation Contexts



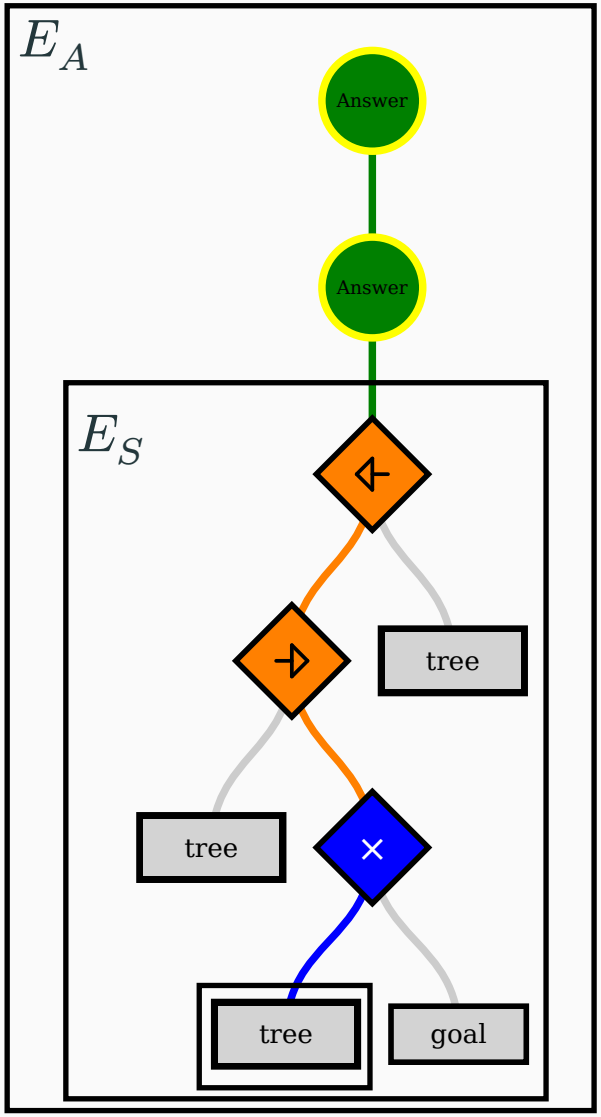
- E_A: Below answer stream

Evaluation Contexts



- E_A: Below answer stream
- E_S: Subtree within search tree

Evaluation Contexts

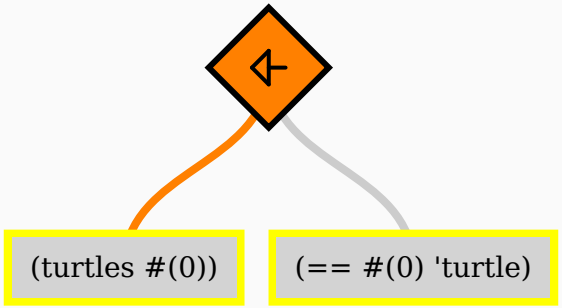


- E_A: Below answer stream
- E_S: Subtree within search tree
- $E = E_A[E_S\ \square]$

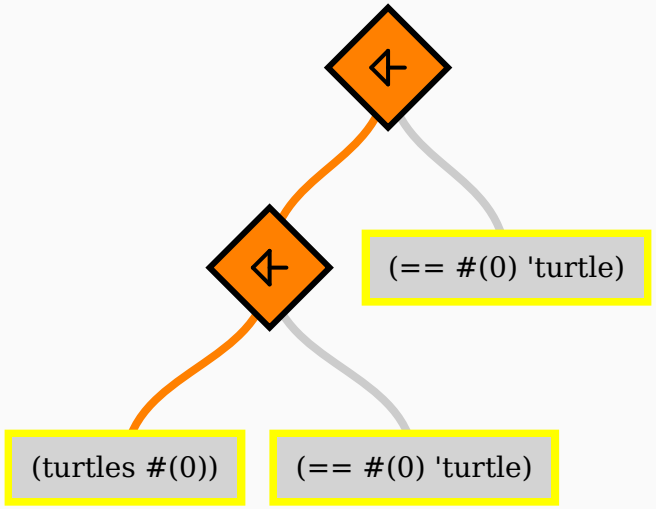
Turtles ... (DFS)

$$\texttt{TURTLES}(x) \leftarrow \texttt{TURTLES}(x) \vee (x = \texttt{turtle})$$

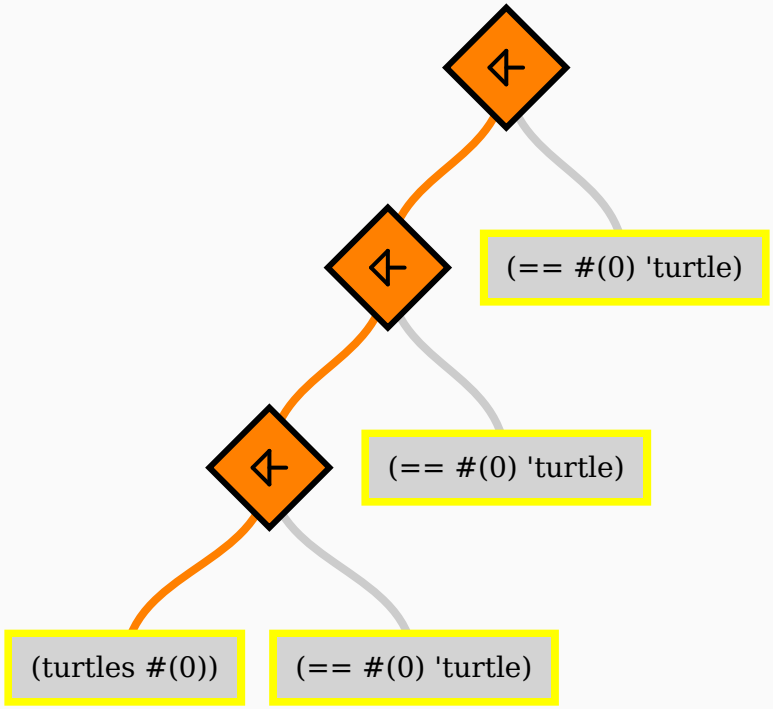
Turtles … (DFS)



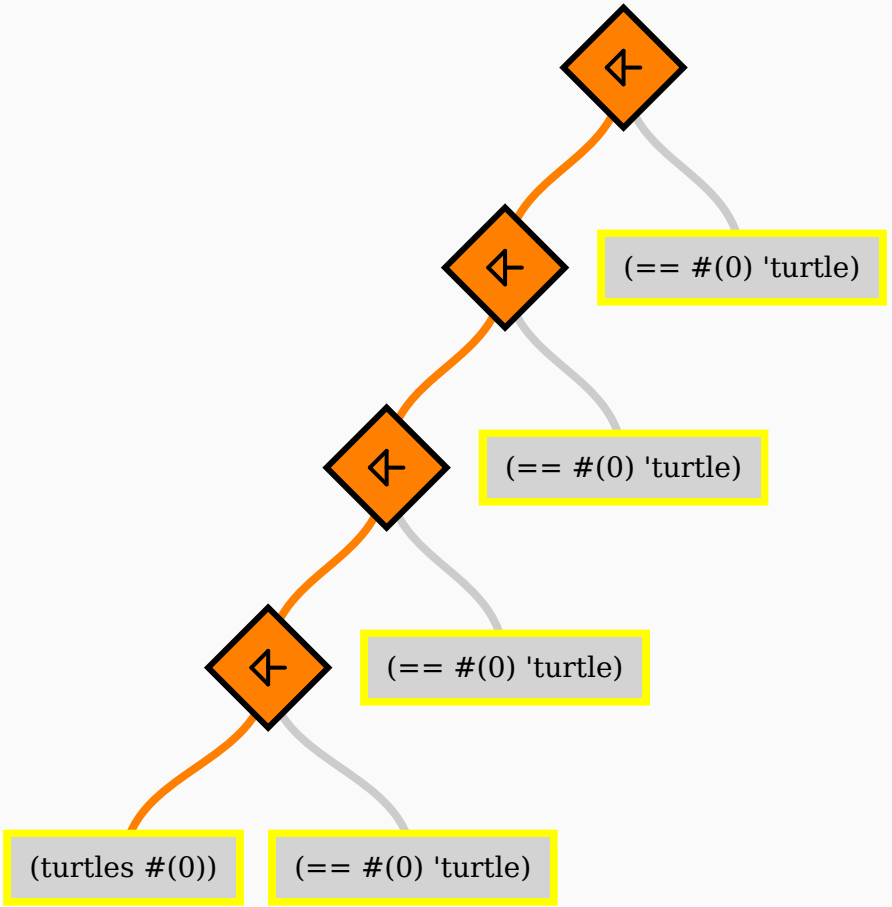
Turtles ... (DFS)



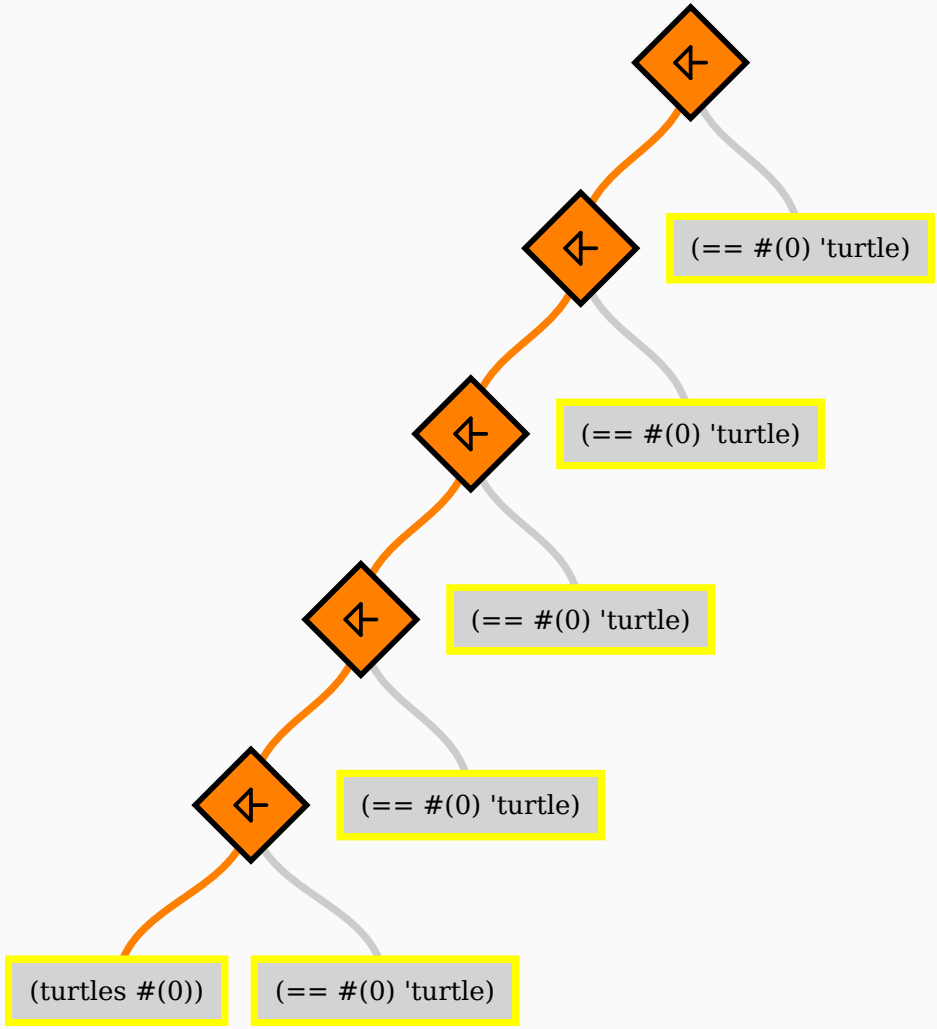
Turtles ... (DFS)



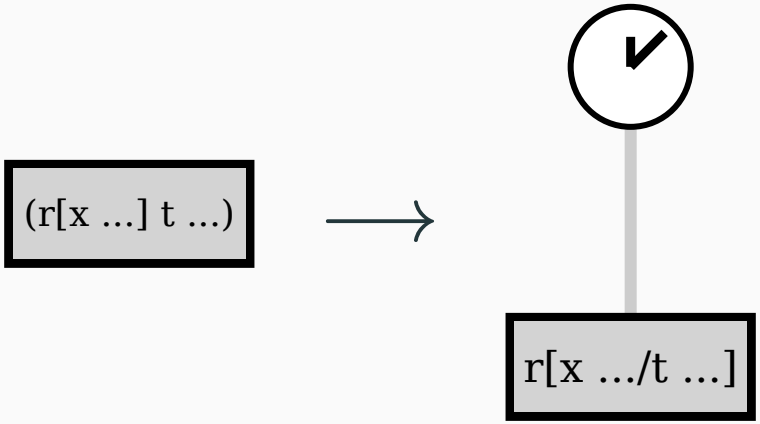
Turtles ... (DFS)



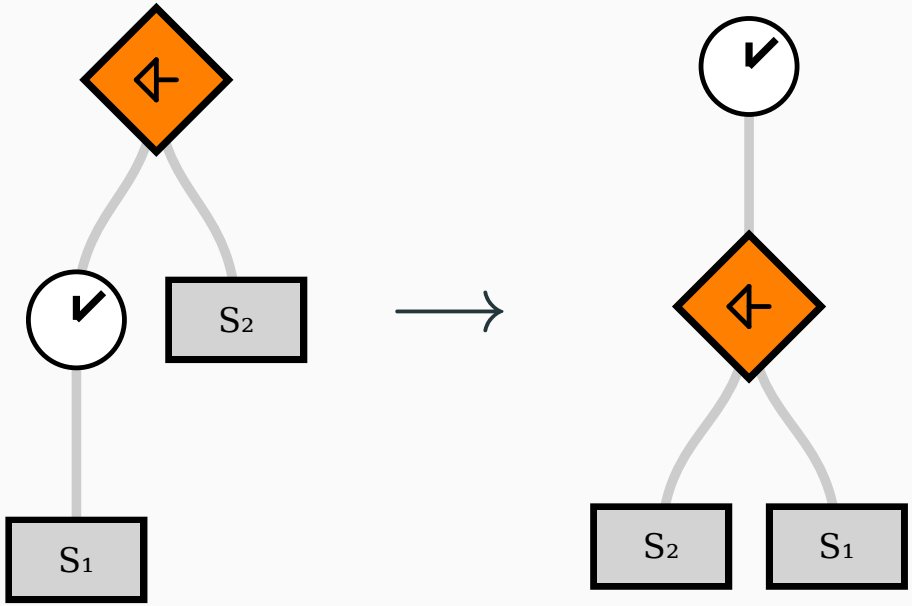
Turtles ... (DFS)



$$E[(r[x...]\ t...)\ \sigma] \longrightarrow E[\texttt{delay}\ \ r[x.../t...]\ \sigma]\ [\text{Delay}]$$

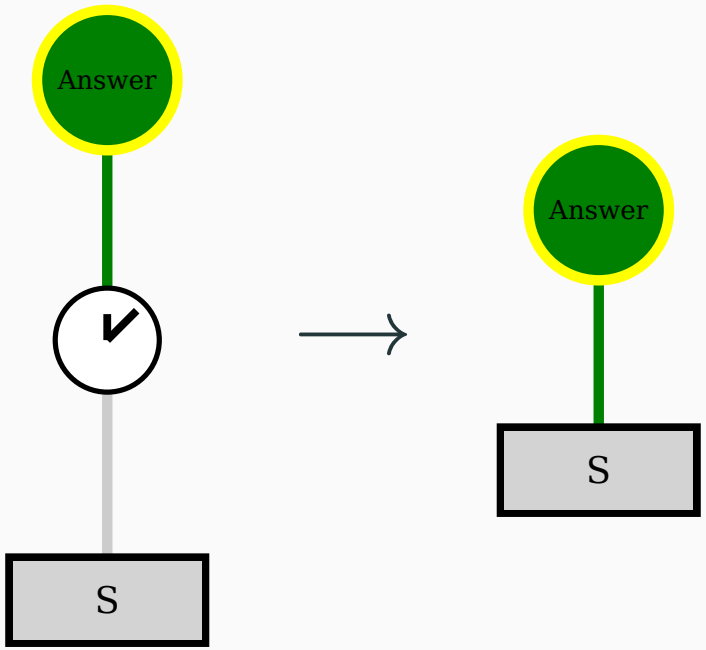


$$E[(\texttt{delay}\ S_1) \leftarrow S_2] \longrightarrow E[\texttt{delay}\ (S_2 \leftarrow S_1)]\ \text{[Interleave]}$$

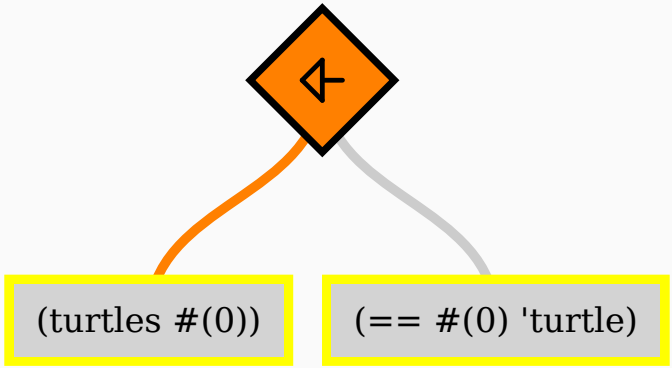


Semantics :: Release Delay

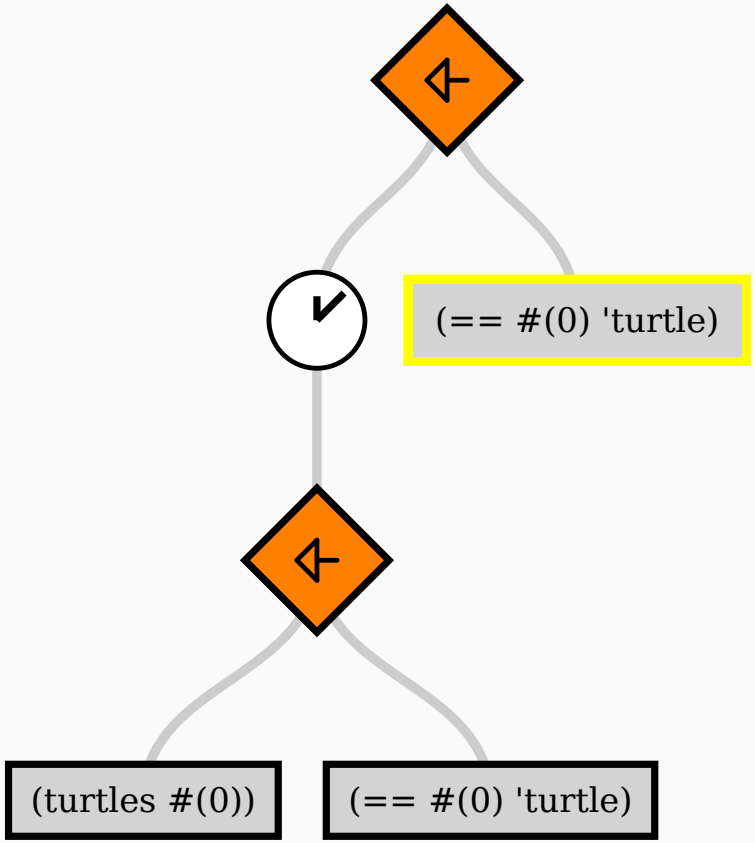
$$E_A[(\texttt{delay}\ S)] \longrightarrow E_A[S]\ [\text{ReleaseDelay}]$$



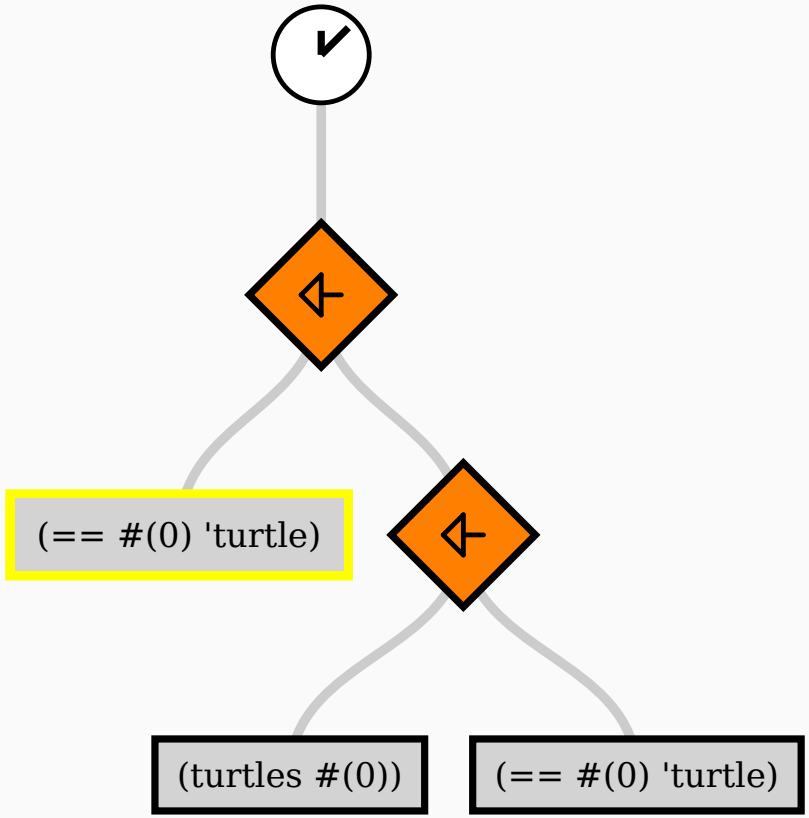
Turtles ... (Interleaving DFS)



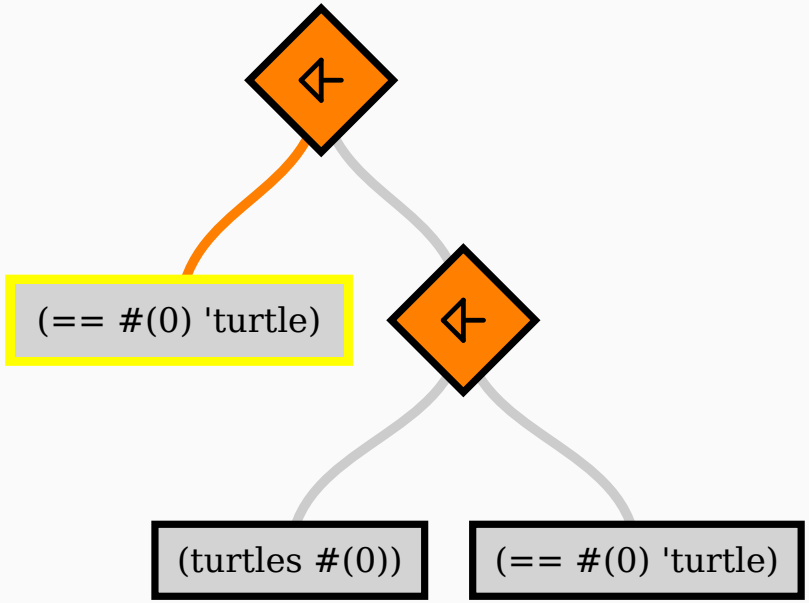
Turtles ... (Interleaving DFS)



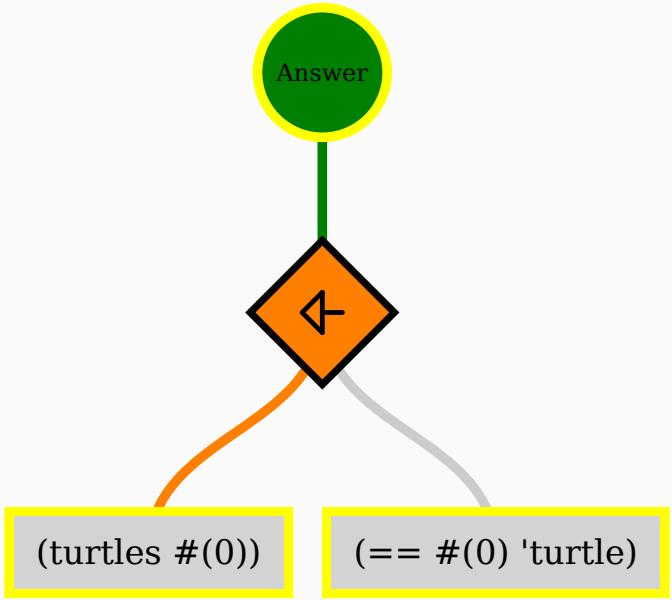
Turtles ... (Interleaving DFS)



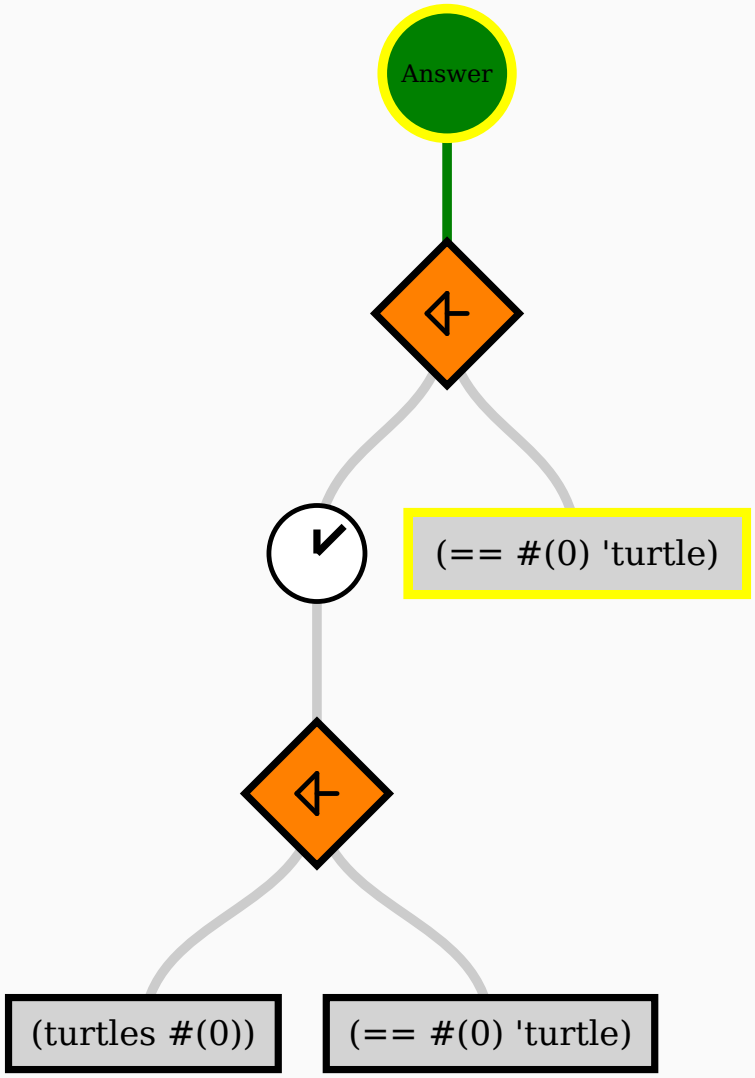
Turtles ... (Interleaving DFS)



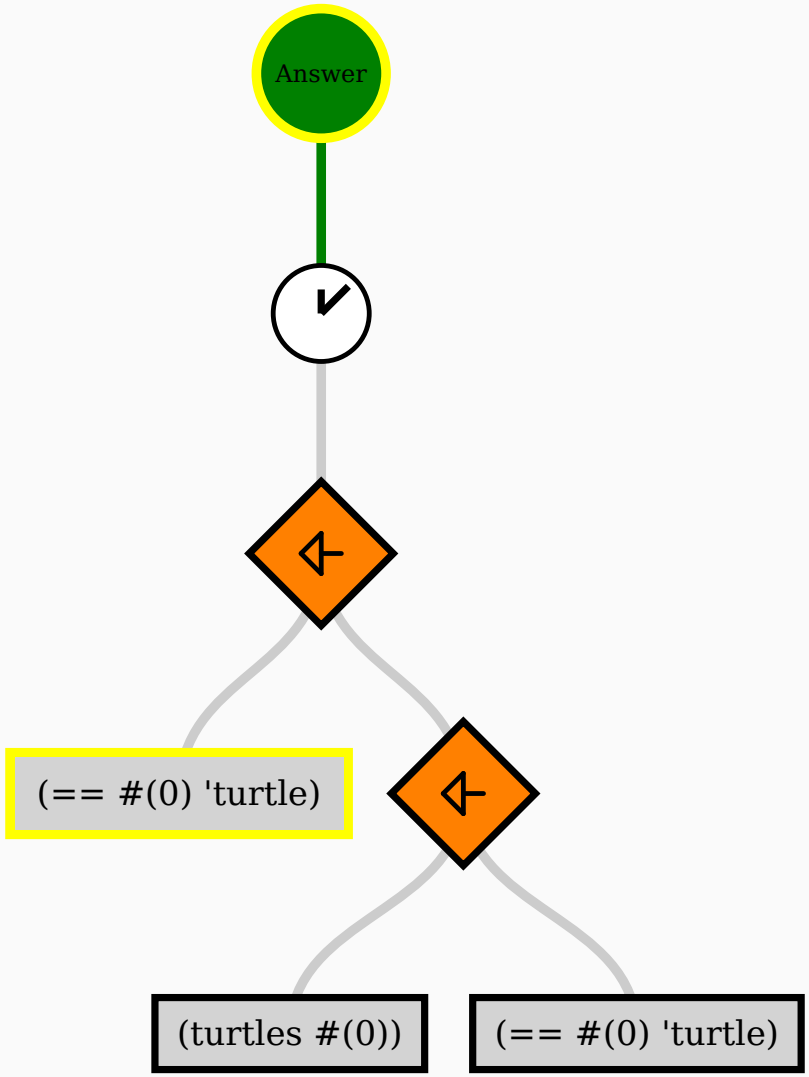
Turtles ... (Interleaving DFS)



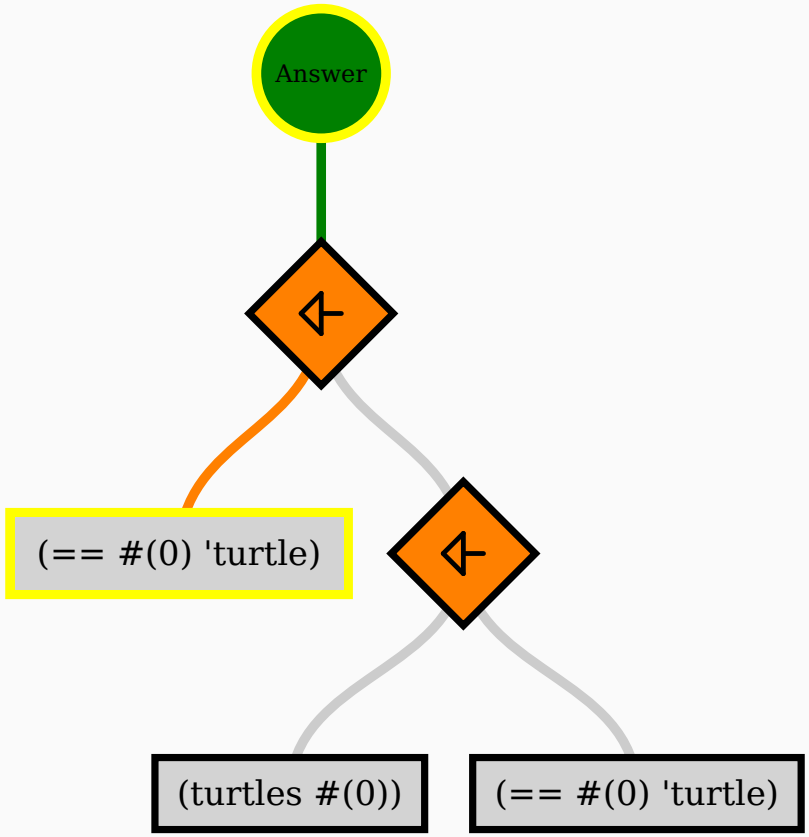
Turtles ... (Interleaving DFS)



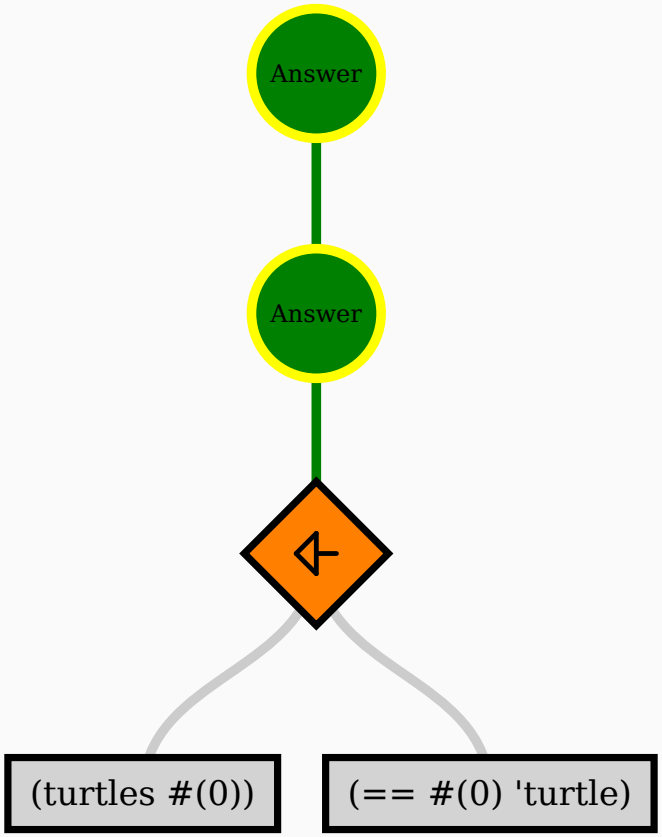
Turtles ... (Interleaving DFS)



Turtles ... (Interleaving DFS)



Turtles ... (Interleaving DFS)



$$\text{Search Trees} \in S ::= S \leftarrow S$$

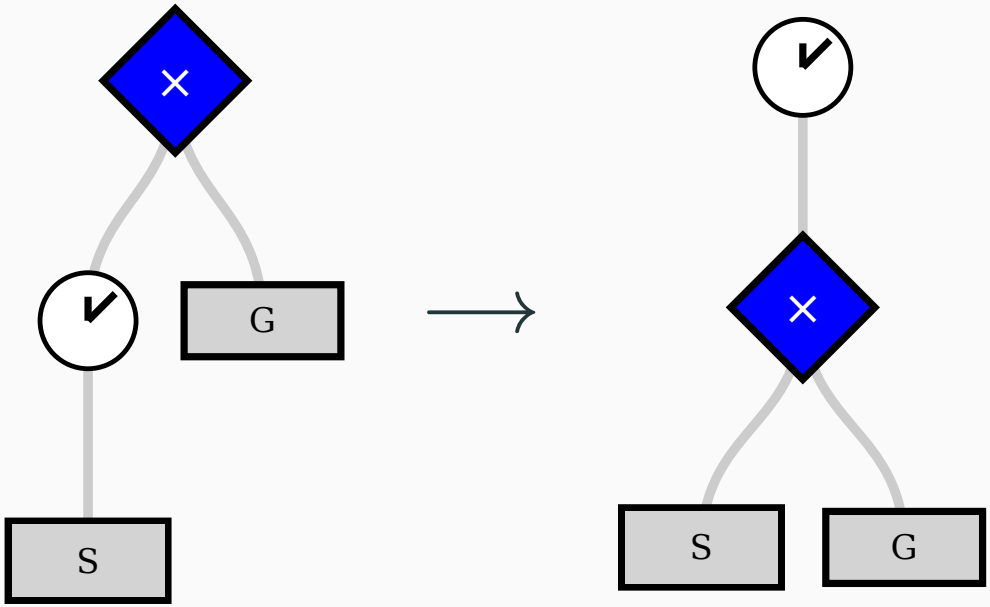
$$\mid \texttt{delay}\ S$$

$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

$$\mid S \times G$$

$$E[((\texttt{delay}\ S) \times G)] \longrightarrow E[(\texttt{delay}\ (S \times G))]\ [\text{DelayConj}]$$



$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

$$\mid S \times G$$

$$\mid G\ \sigma$$

Goal State

Goals

$$\text{Goals} \in G ::= r(x...) \qquad \text{Relation Call}$$

Goals

$$\text{Goals} \in G ::= r(x...) \quad \text{Relation Call}$$
$$\mid t = t \quad \text{Unification}$$

Goals

$$
\begin{aligned}
\text{Goals} \in G ::= {} & r(x\ldots) && \text{Relation Call} \\
| \; & t = t && \text{Unification} \\
| \; & \exists(x\ldots)G && \text{Fresh}
\end{aligned}
$$

Goals

$$
\begin{aligned}
\text{Goals} \in G ::= {} & r(x\ldots) && \text{Relation Call} \\
\mid {} & t = t && \text{Unification} \\
\mid {} & \exists(x\ldots)G && \text{Fresh} \\
\mid {} & G \vee G && \vee
\end{aligned}
$$

Goals

$$
\begin{aligned}
\text{Goals} \in G ::= {} & r(x\ldots) && \text{Relation Call} \\
\mid {} & t = t && \text{Unification} \\
\mid {} & \exists(x\ldots)G && \text{Fresh} \\
\mid {} & G \vee G && \vee \\
\mid {} & G \wedge G && \wedge
\end{aligned}
$$

Search Trees :: Success and Failure

$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

$$\mid S \times G$$

$$\mid G\ \sigma$$

Goal State

$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

$$\mid S \times G$$

$$\mid G\,\sigma$$

Goal State

$$\mid \top\,\sigma$$

$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

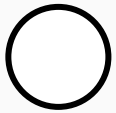
$$\mid S \times G$$

$$\mid G\,\sigma$$

Goal State

$$\mid \top\,\sigma$$

$$\mid \texttt{empty}$$



Answer Streams

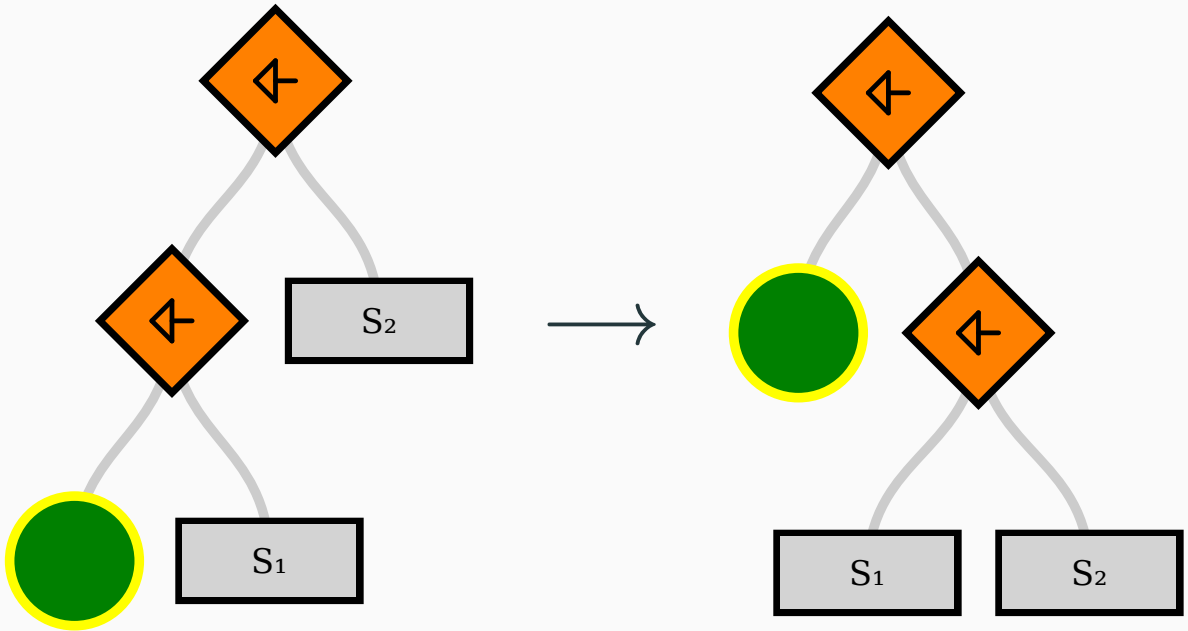
$$\text{Answer Streams} \in A ::= (\top\ \sigma) + A$$
$$\mid S$$

Semantics :: Goal Evaluation

$$E[(G\ \sigma)] \longrightarrow E[(\top \sigma)]\ \text{[GoalSucc]}$$

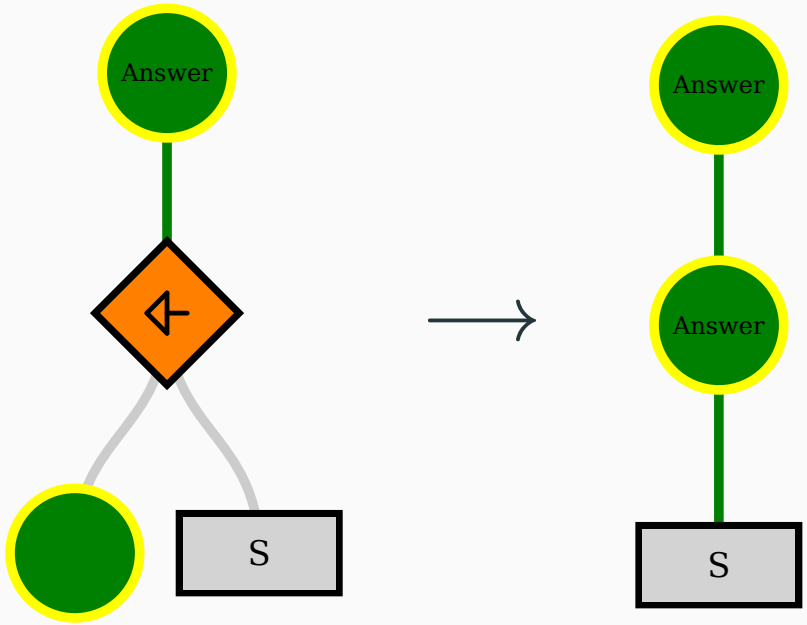
$$E[(G\ \sigma)] \longrightarrow E[\texttt{empty}]\ \text{[GoalFail]}$$

$$E[((\top\ \sigma) \leftarrow S_1) \leftarrow S_2] \longrightarrow E[(\top\ \sigma) \leftarrow (S_1 \leftarrow S_2)]\ \text{[ReassocDisj]}$$



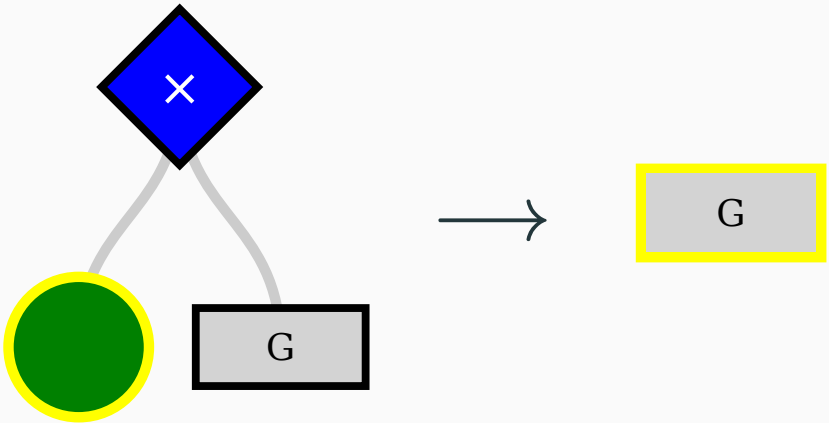
Semantics :: Promote

$$E_A[(\top\ \sigma) \leftarrow S] \longrightarrow E_A[(\top\ \sigma) + S]\ \text{[Promote]}$$

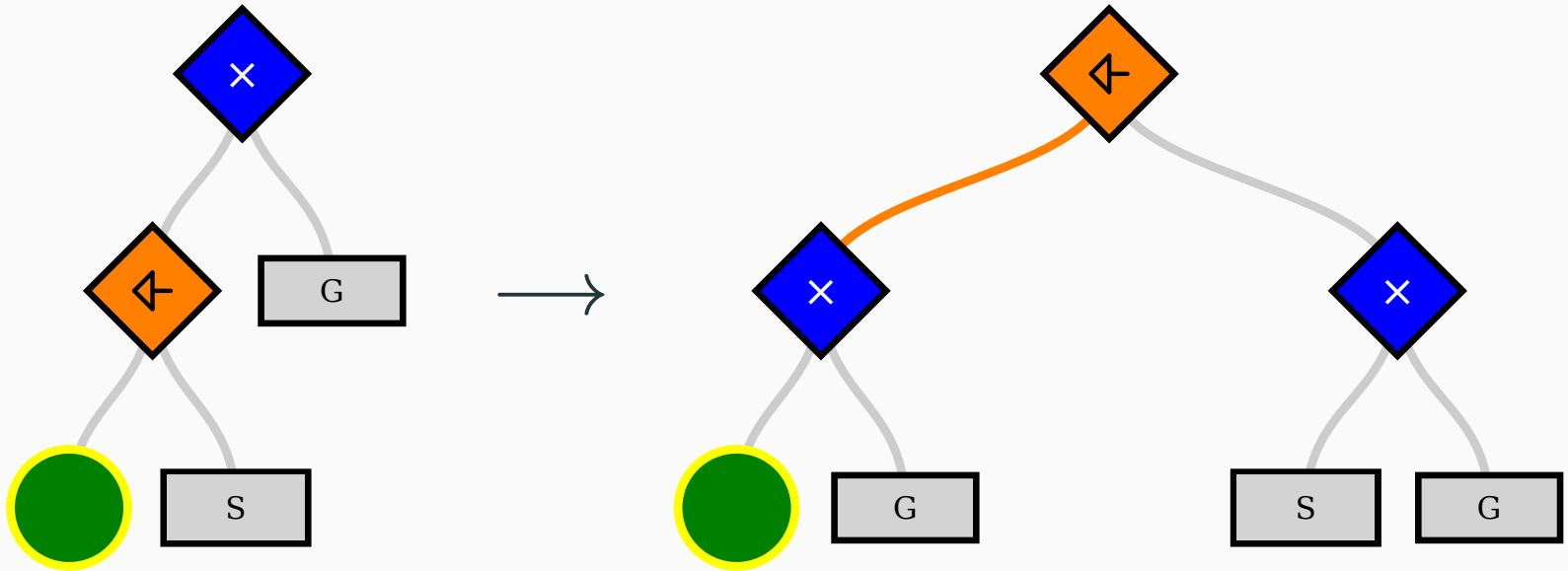


Semantics :: SuccConj

$$E[(\top\ \sigma) \times G] \longrightarrow E[G\ \sigma]\ \text{[SuccConj]}$$

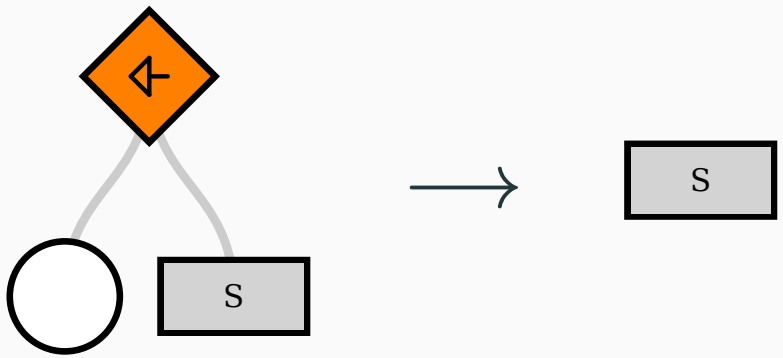


$$E[((\top\ \sigma) \leftarrow S) \times G] \longrightarrow E[((\top\ \sigma) \times G) \leftarrow (S \times G)]\ \text{[DistrConj]}$$

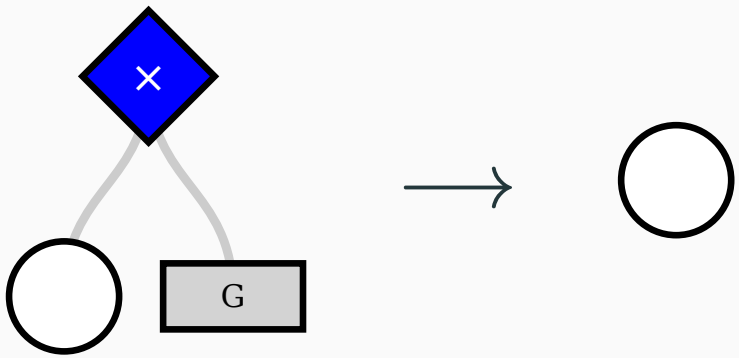


Semantics :: Pruning

$E[\texttt{empty} \leftarrow S] \longrightarrow E[S]$ [PruneDisj]



$E[\texttt{empty} \times G] \longrightarrow E[\texttt{empty}]$ [PruneConj]



Validation

Validation

- **Syntactic Preservation**
  - Every reduction step preserves well-formedness.
    - Valid substitution structure
    - Properly scoped logic variables
    - Relation existence and arity

Validation

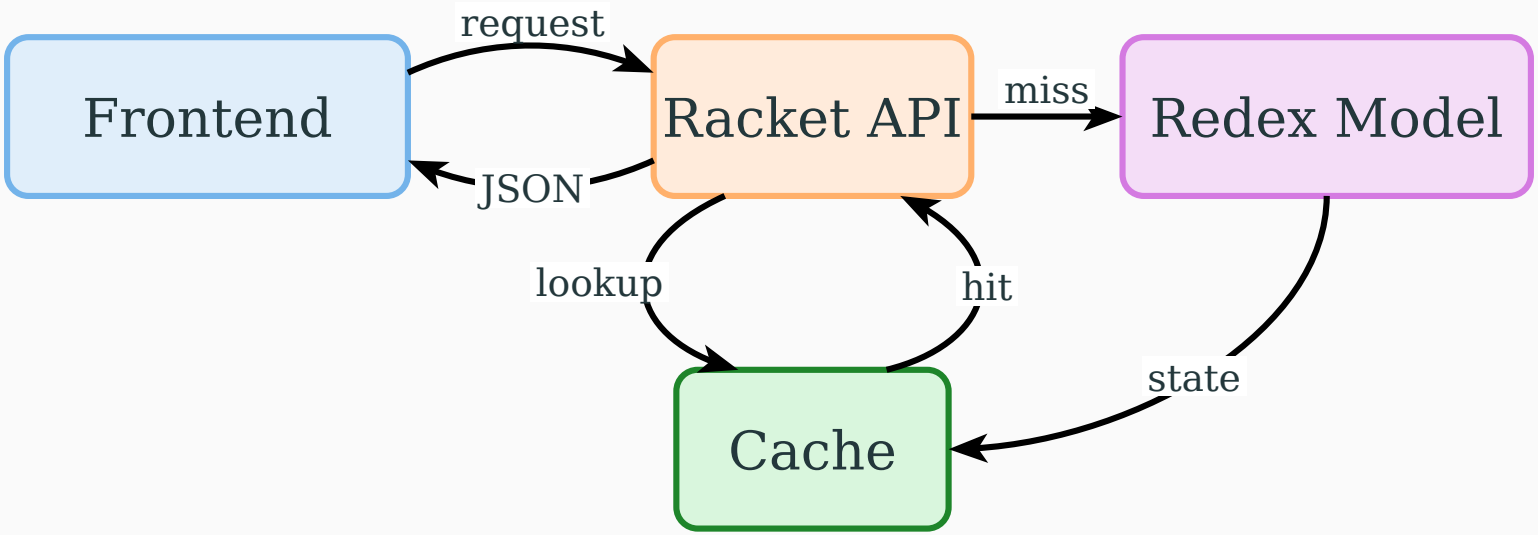
- **Syntactic Preservation**
  - Every reduction step preserves well-formedness.
    - Valid substitution structure
    - Properly scoped logic variables
    - Relation existence and arity
- **Determinism**
  - Every well-formed program can take at most one step

Validation

- **Syntactic Preservation**
  - Every reduction step preserves well-formedness.
    - Valid substitution structure
    - Properly scoped logic variables
    - Relation existence and arity
- **Determinism**
  - Every well-formed program can take at most one step
- **Monotonic Propagation of Delayed Subtrees**
  - Delayed subtrees cannot become stranded
  - Resumption occurs at the top of the search tree

Visualizer

Visualizer :: Architecture



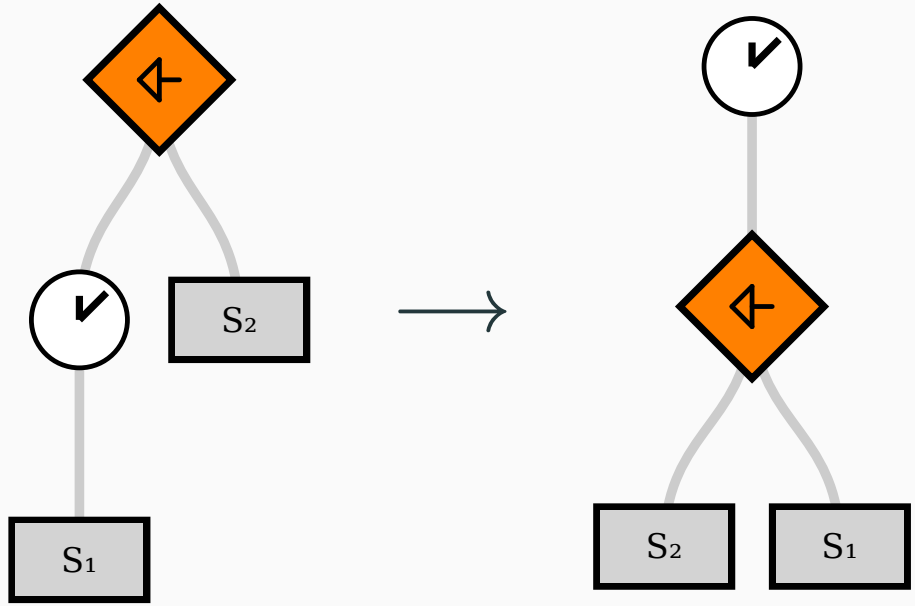
UX Enhanced Model

1. Railway Model (Directed Disjunctions)
2. Lazy Relation Expansion
3. Tagging for Bidirectional Tracing
4. Trail (History of Equations)
5. Current Reification/Projection

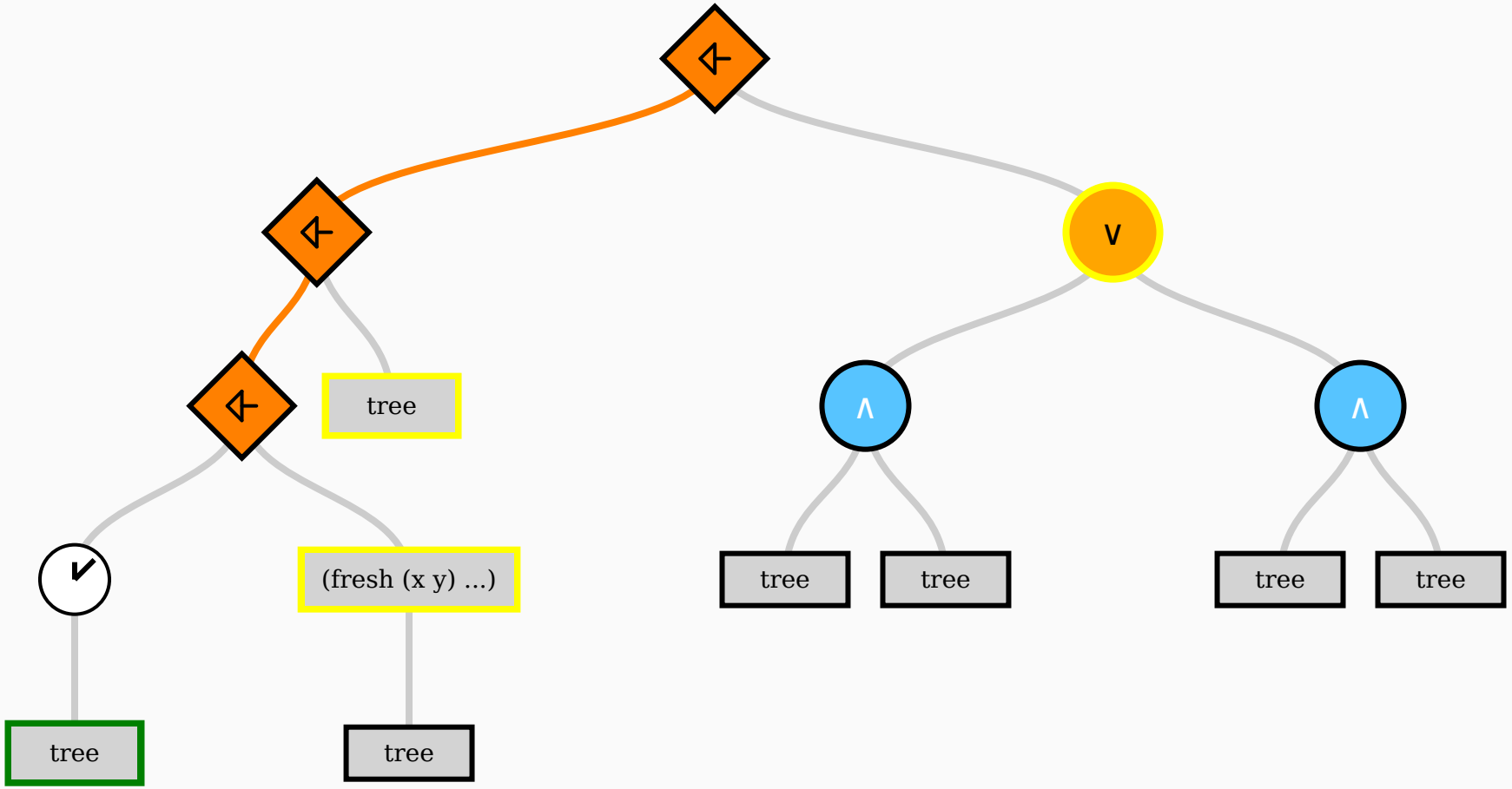
Railway Model

Old Interleaving

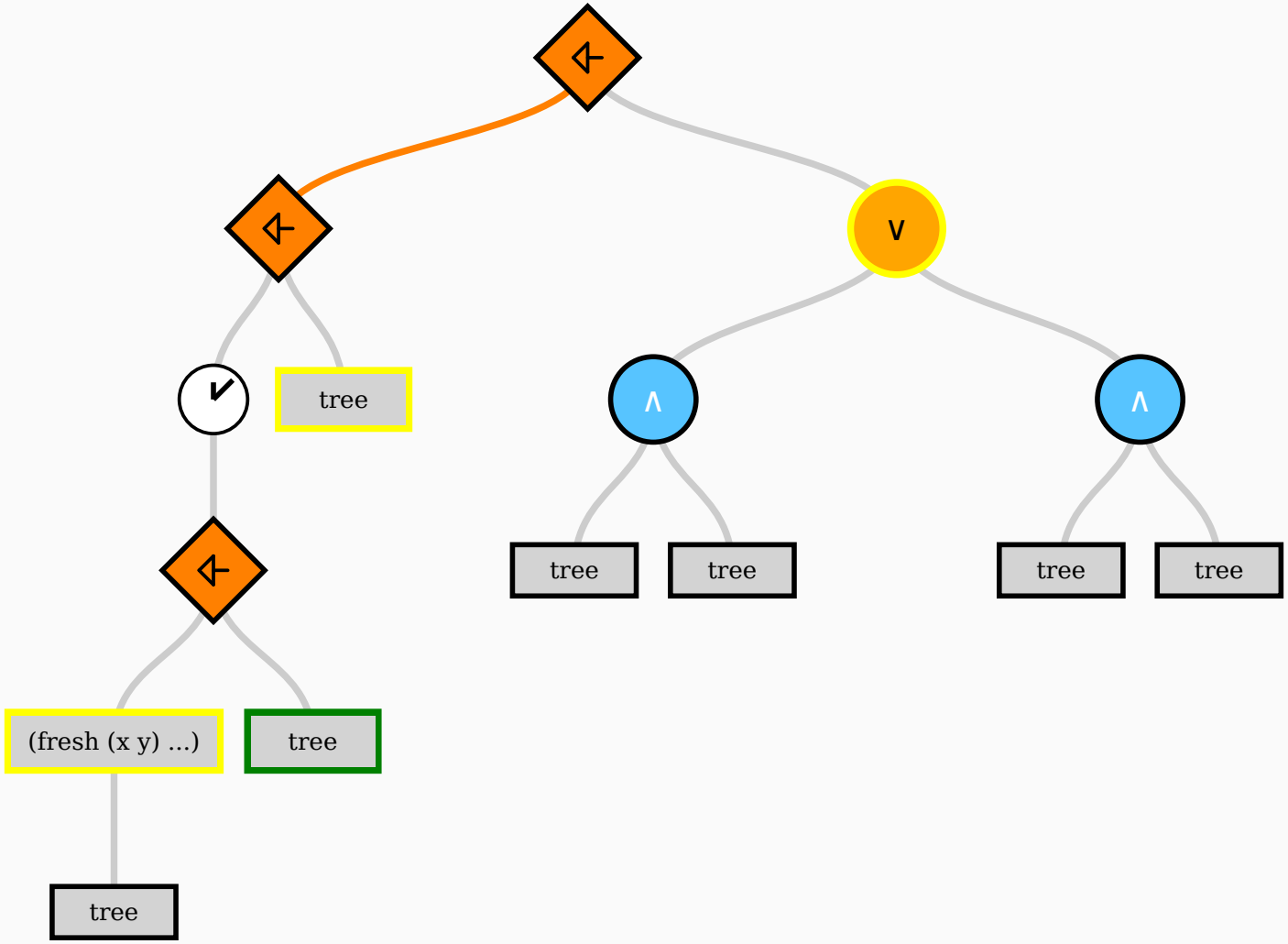
$$E[(\texttt{delay}\ S_1) \leftarrow S_2] \longrightarrow E[\texttt{delay}\ (S_2 \leftarrow S_1)]\ [\text{Interleave (Old)}]$$



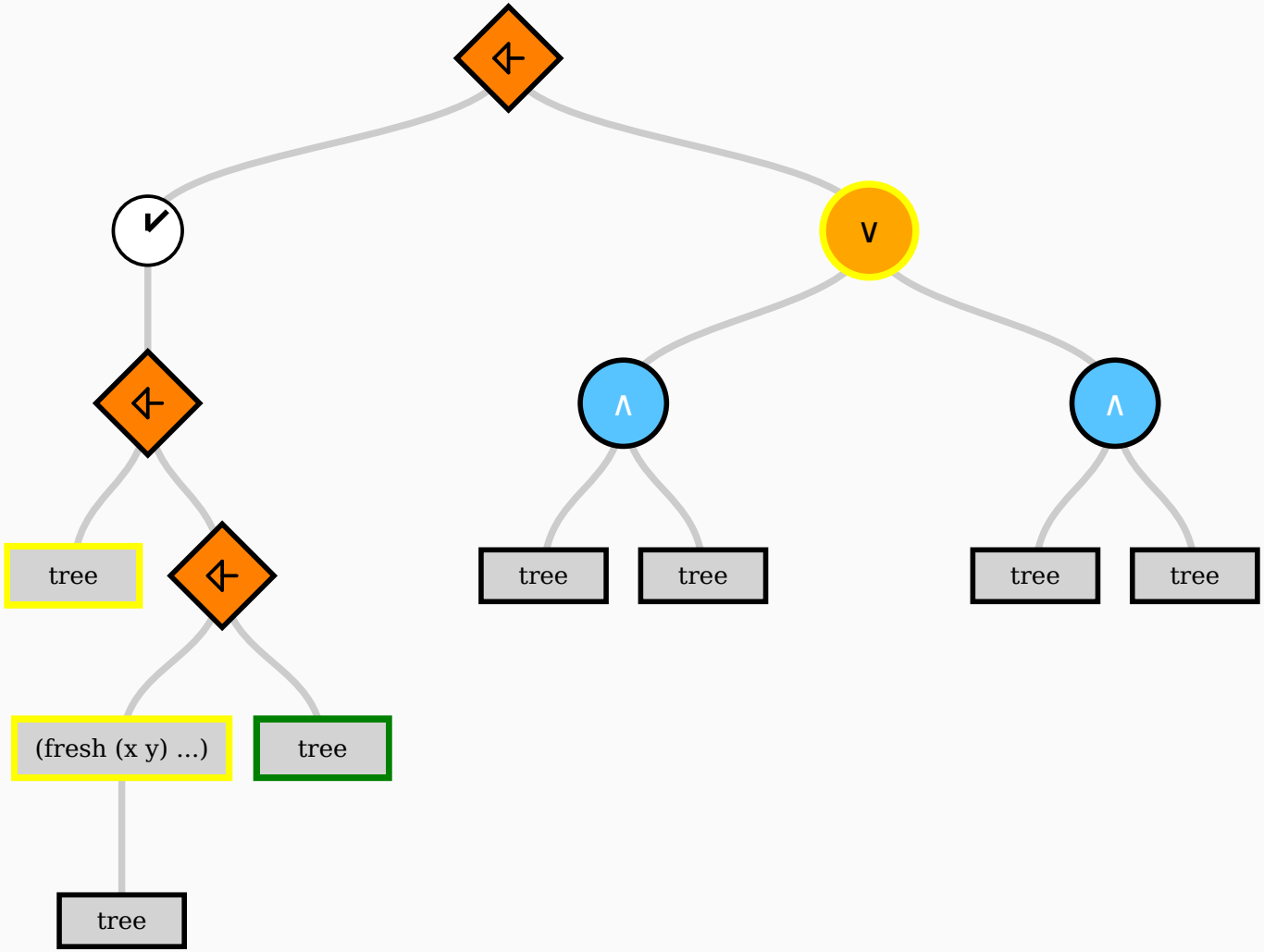
Old Interleaving



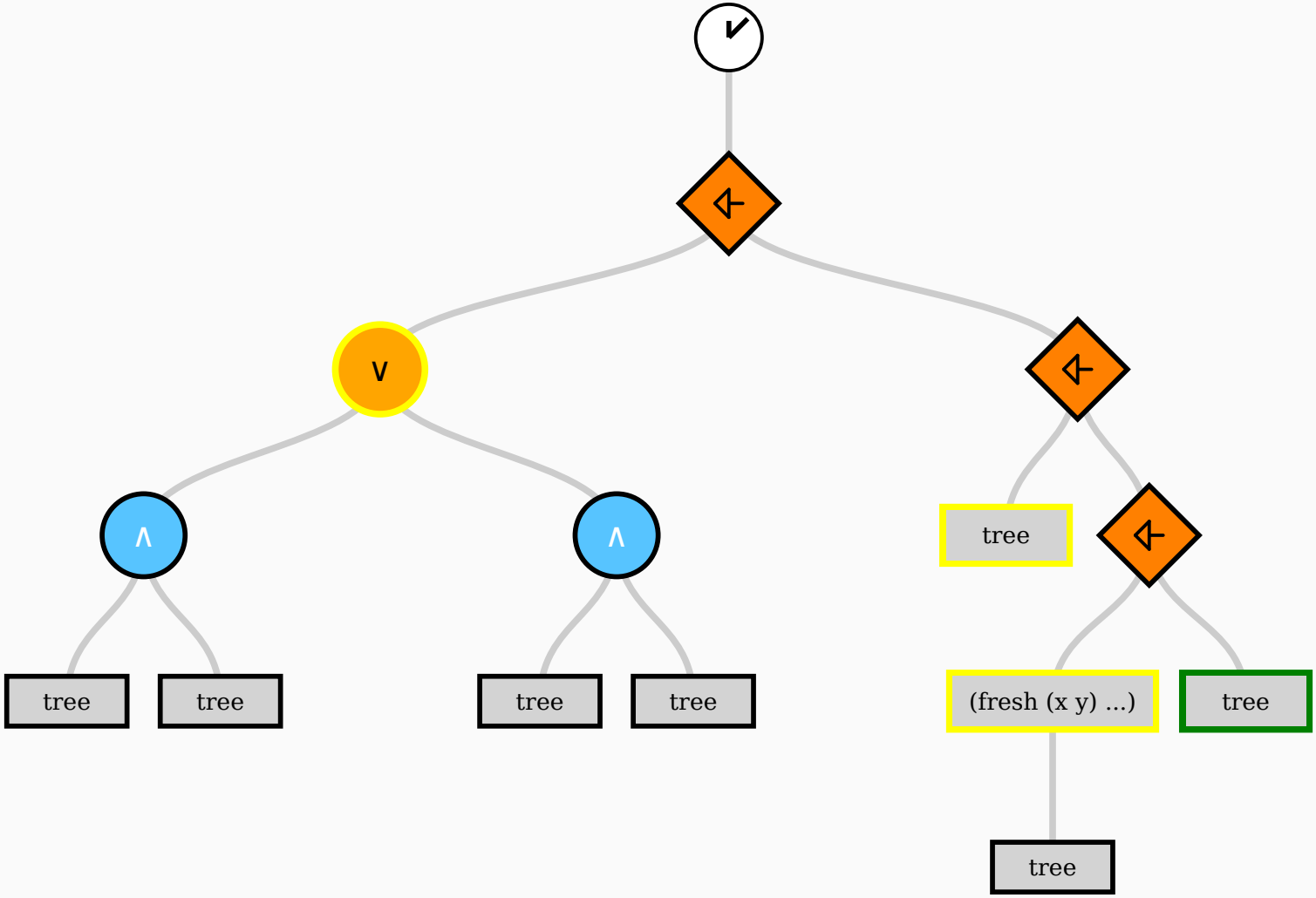
Old Interleaving



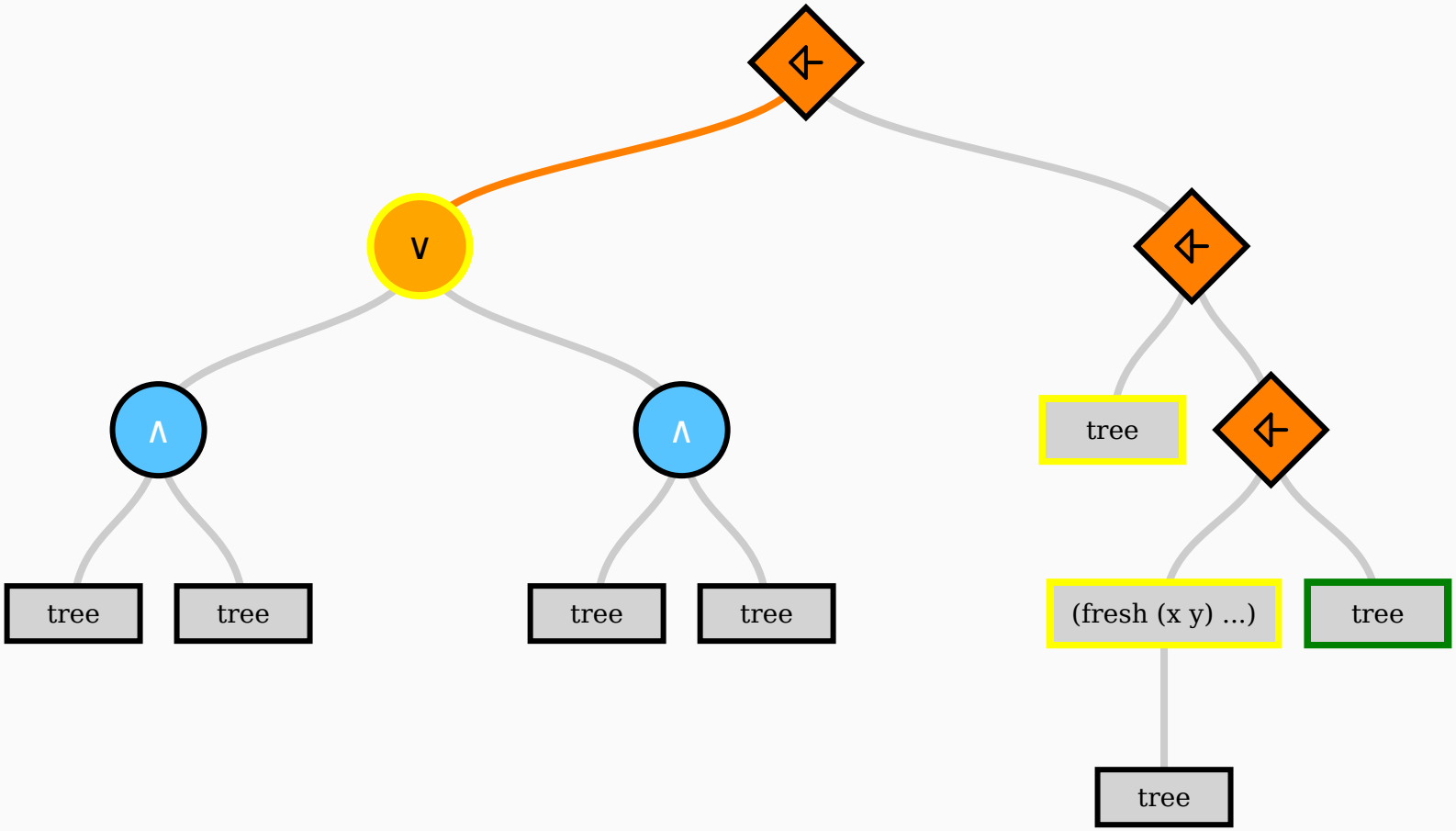
Old Interleaving



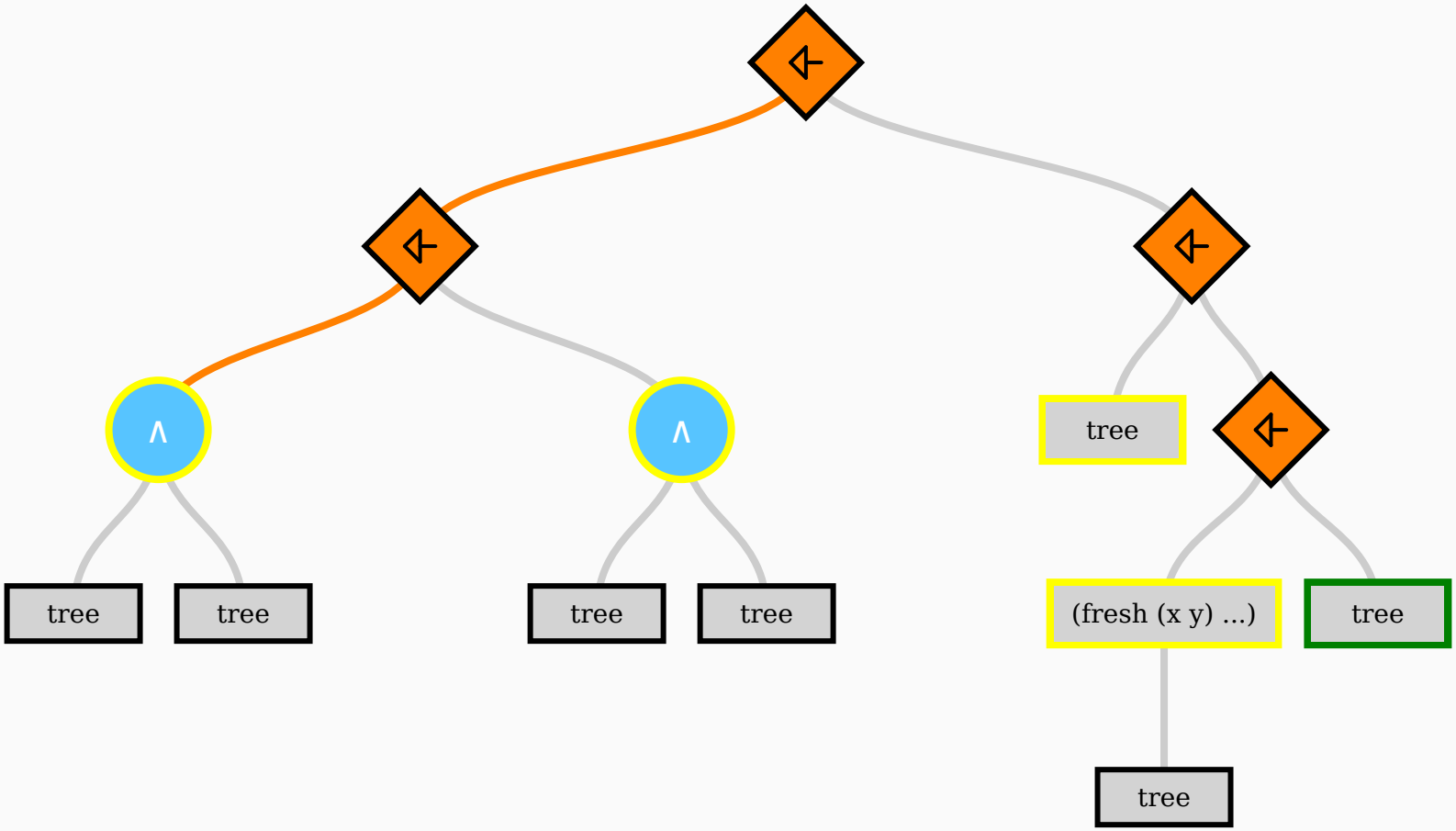
Old Interleaving



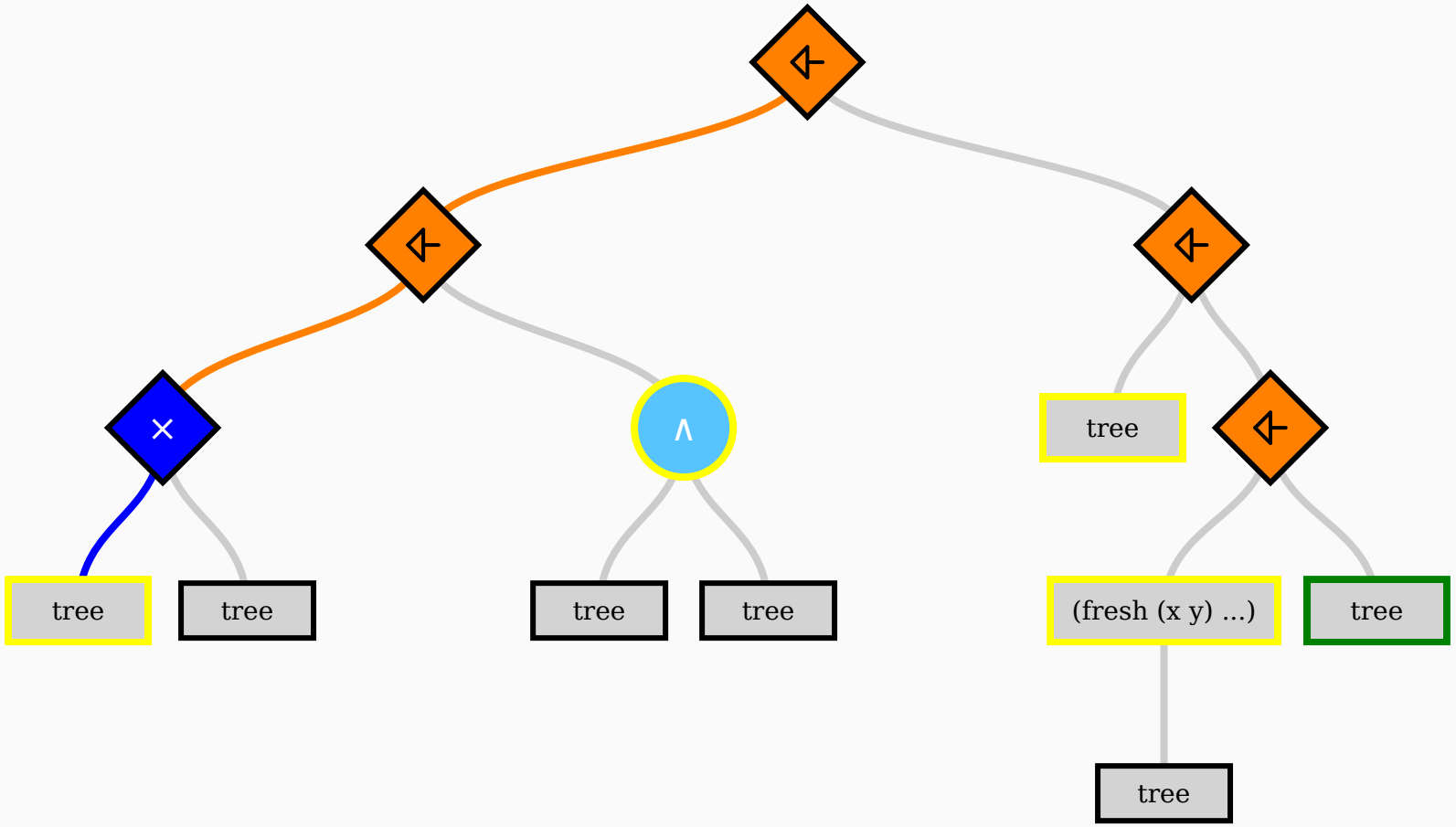
Old Interleaving



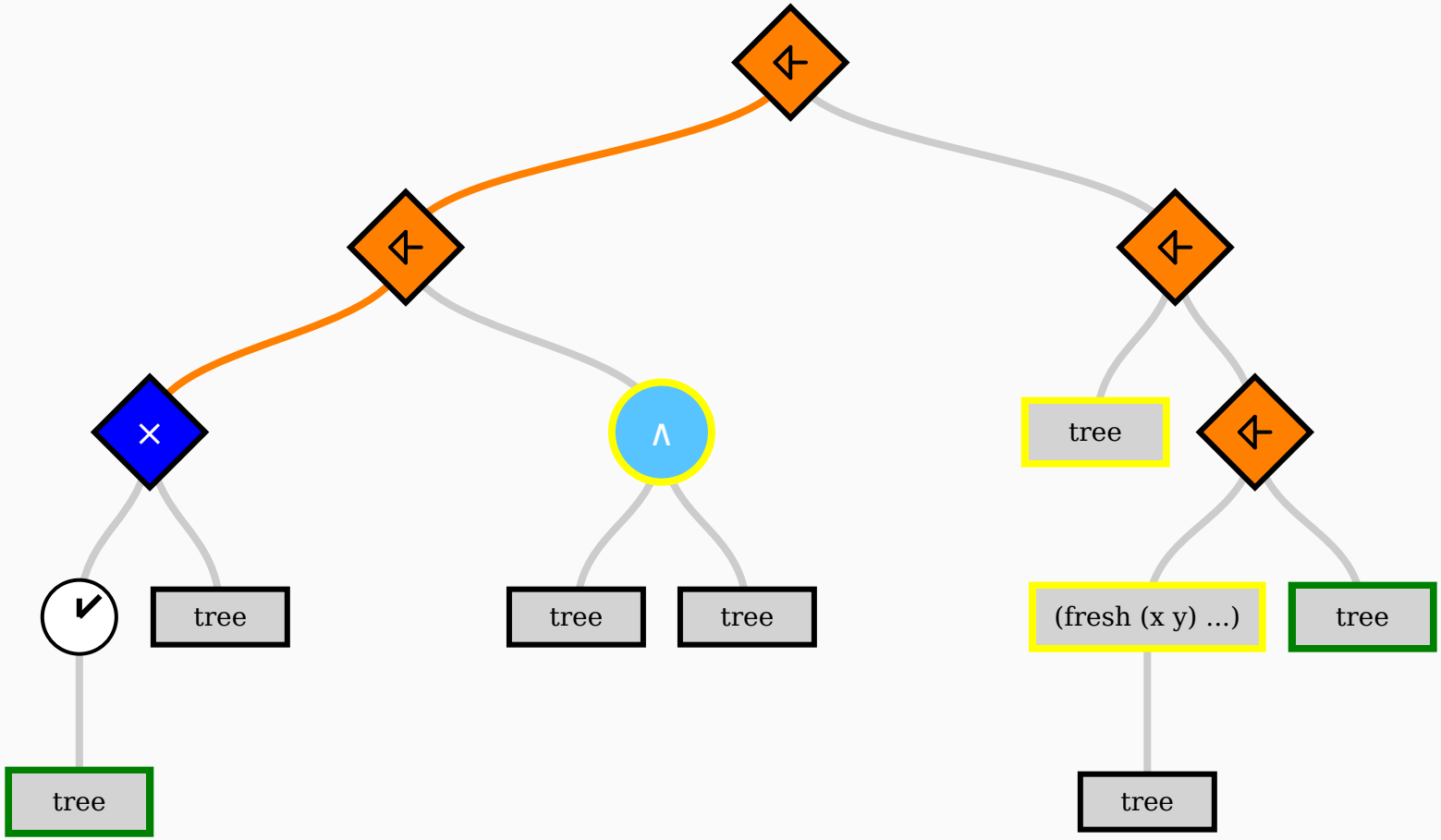
Old Interleaving



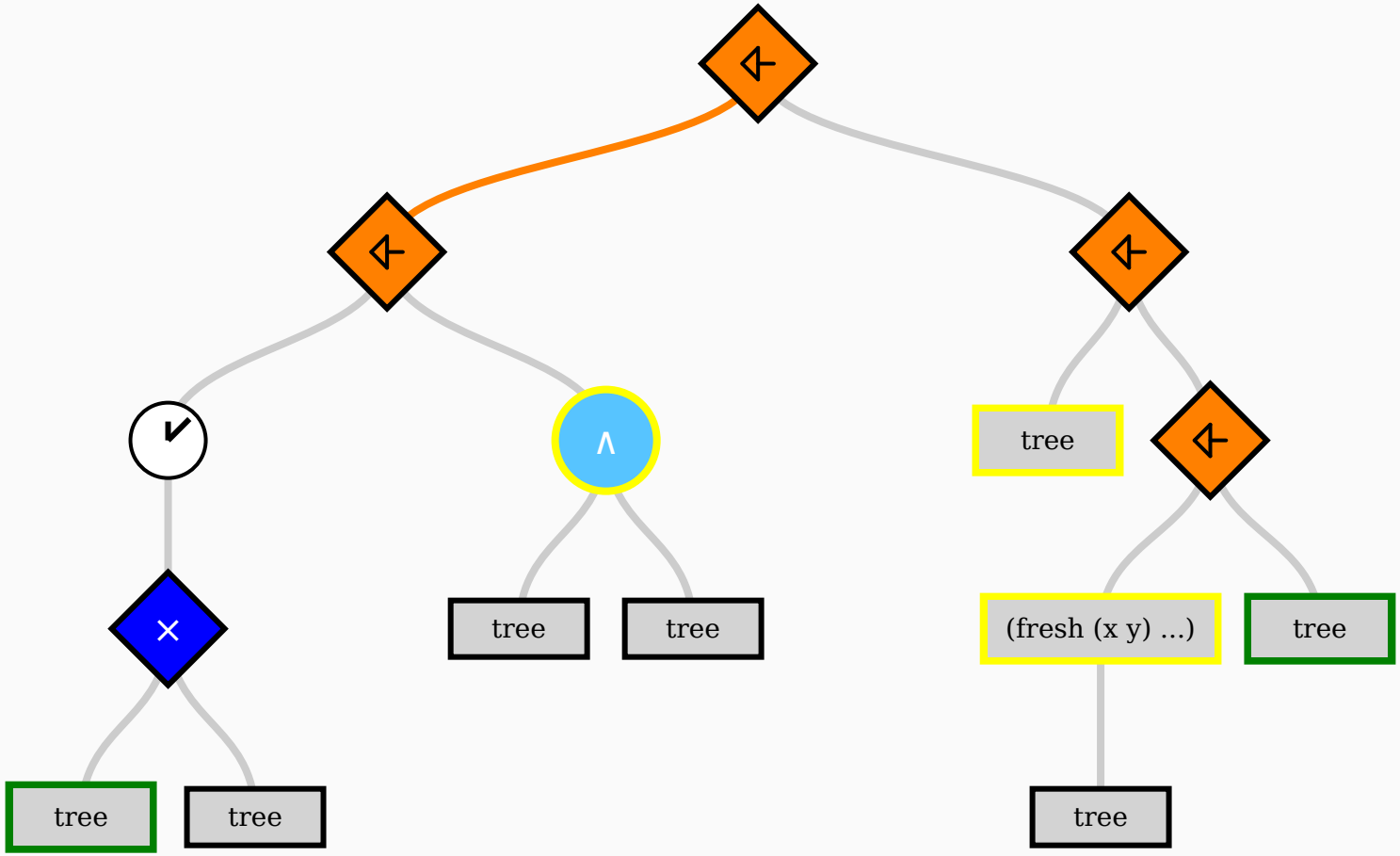
Old Interleaving



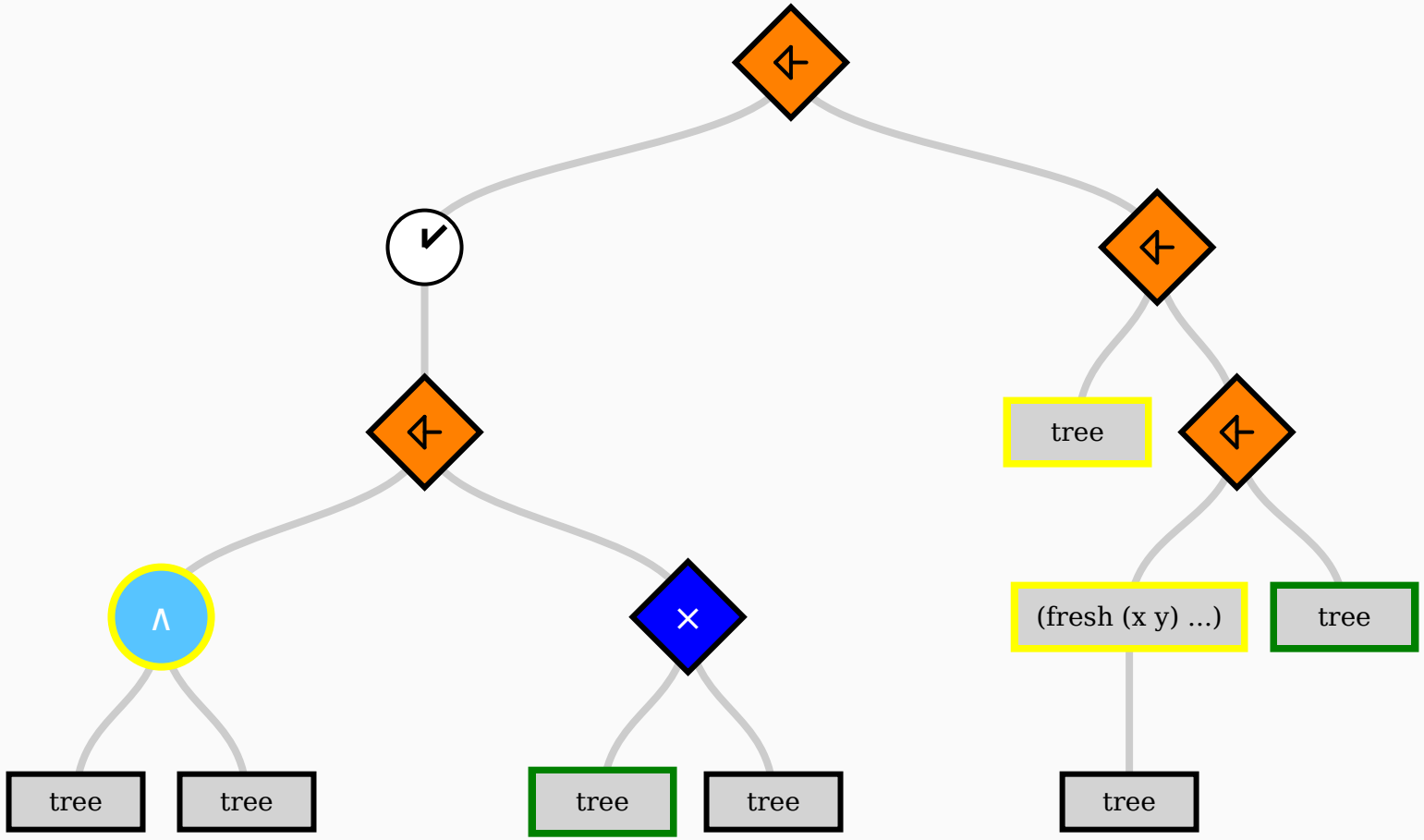
Old Interleaving



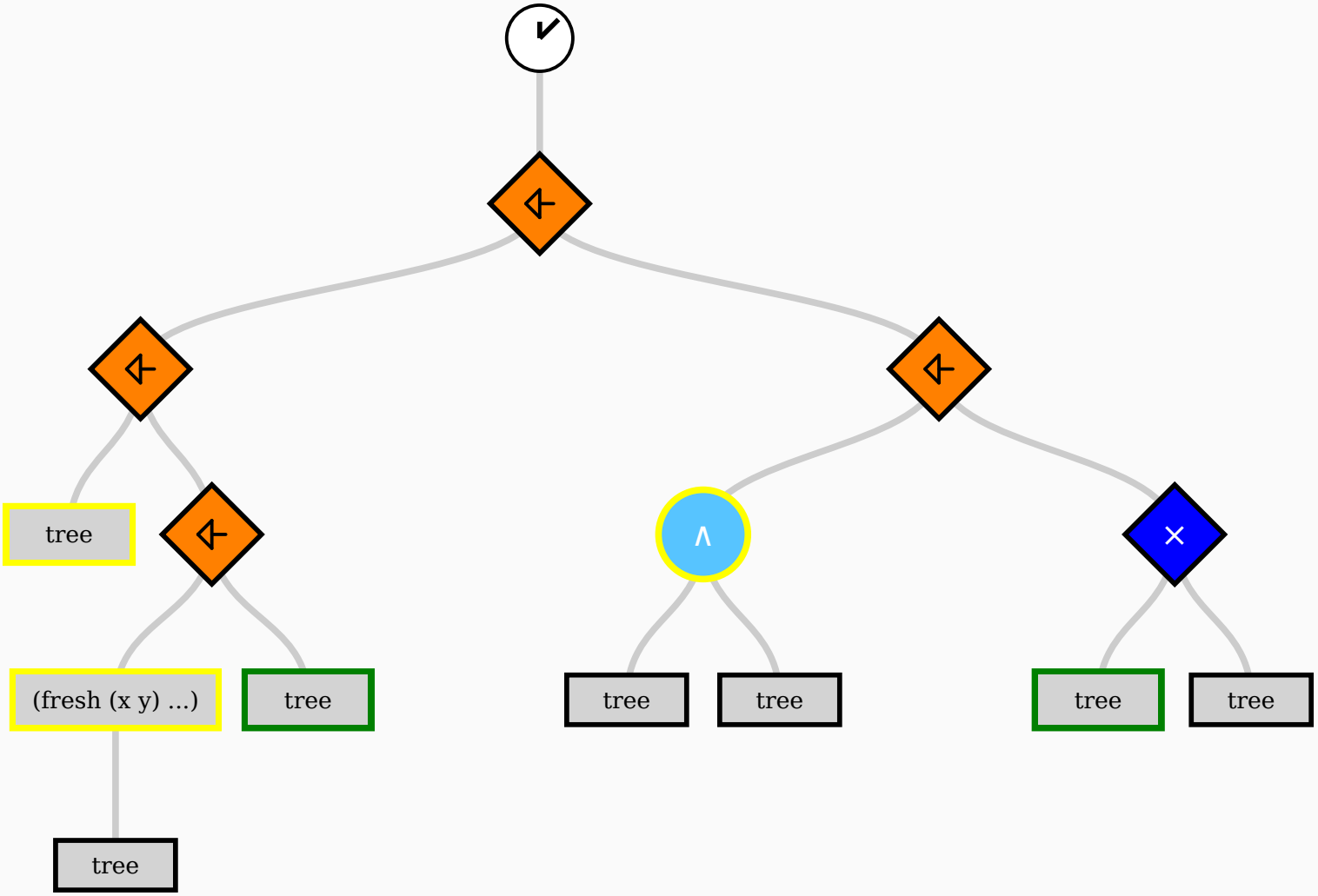
Old Interleaving



Old Interleaving



Old Interleaving



Railway Model :: Right Disjunctions

$$\text{Search Trees} \in S ::= S \leftarrow S$$

$$\mid \texttt{delay}\ S$$

Railway Model :: Right Disjunctions

$$\text{Search Trees} \in S ::= S \leftarrow S$$

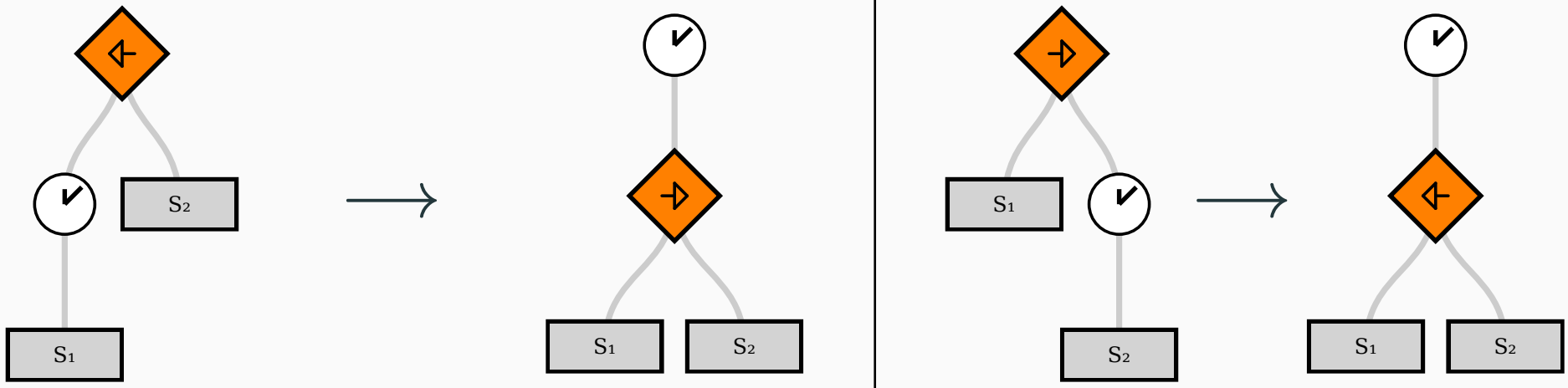
$$\mid S \rightarrow S$$

$$\mid \texttt{delay}\ S$$

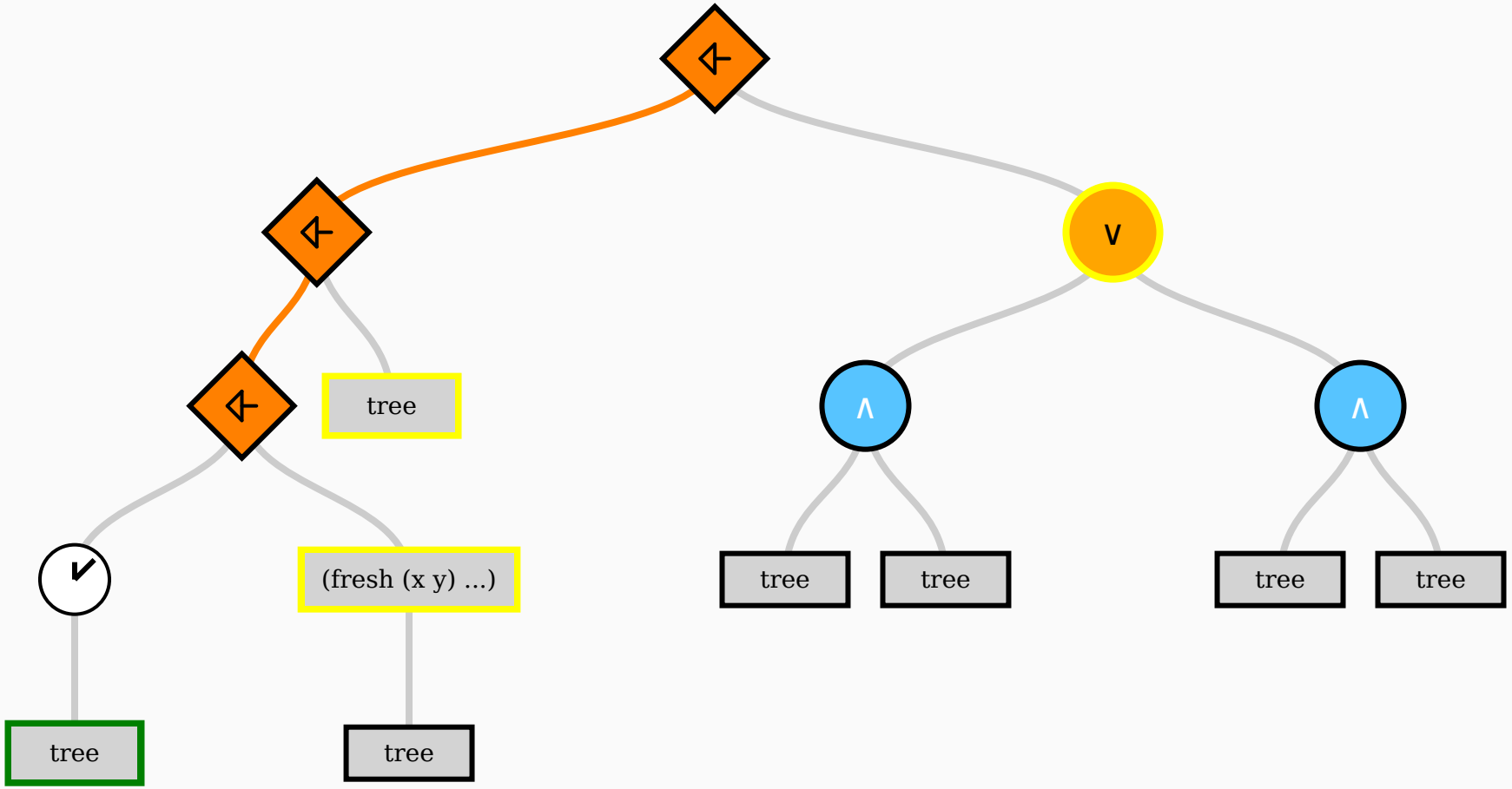
Railway Model :: Interleave Rules

$$E[(\texttt{delay}\ S_1) \leftarrow S_2] \longrightarrow E[\texttt{delay}\ (S_1 \rightarrow S_2)] \quad [\text{InterleaveLeft}]$$

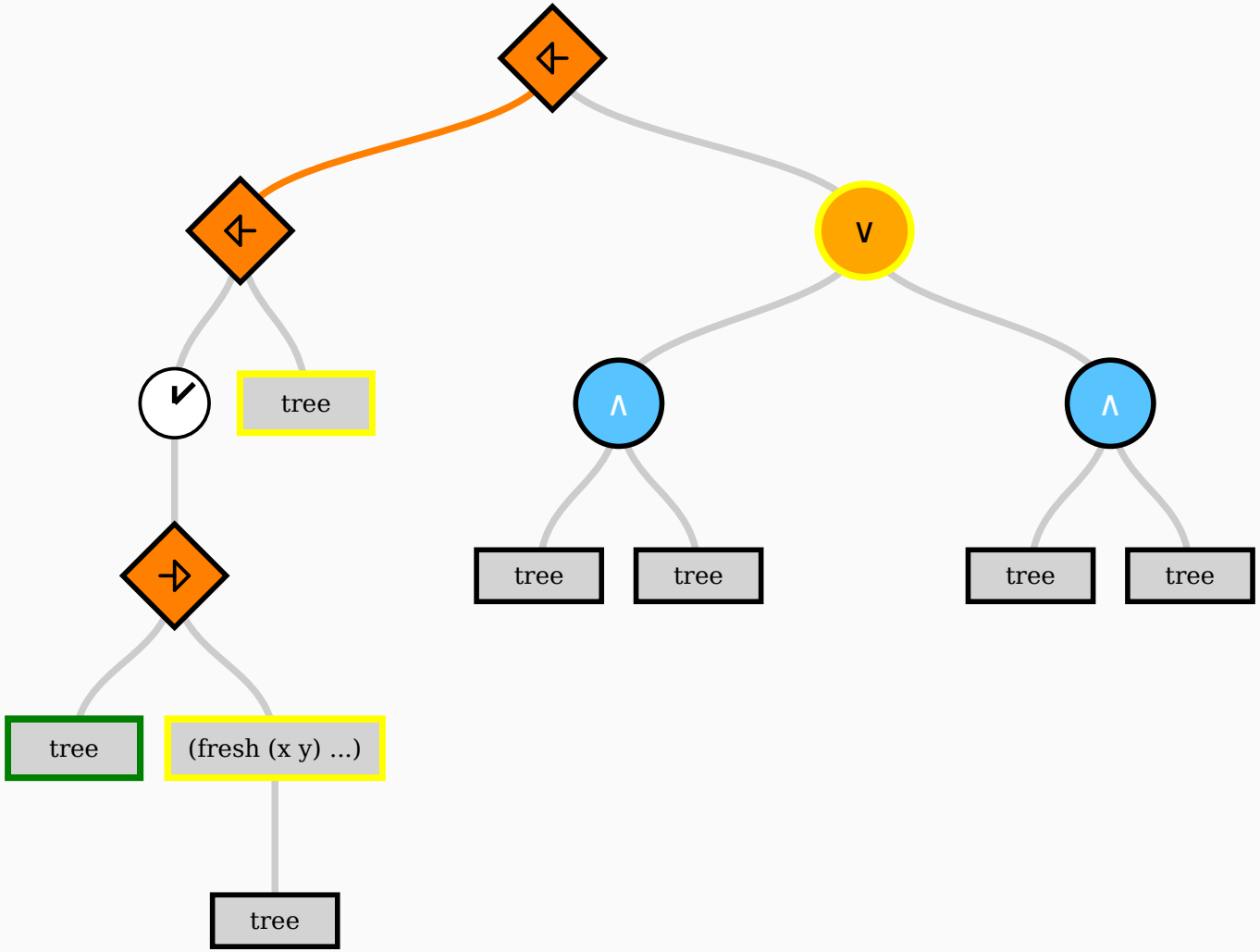
$$E[S_1 \rightarrow (\texttt{delay}\ S_2)] \longrightarrow E[\texttt{delay}\ (S_1 \leftarrow S_2)] \ [\text{InterleaveRight}]$$



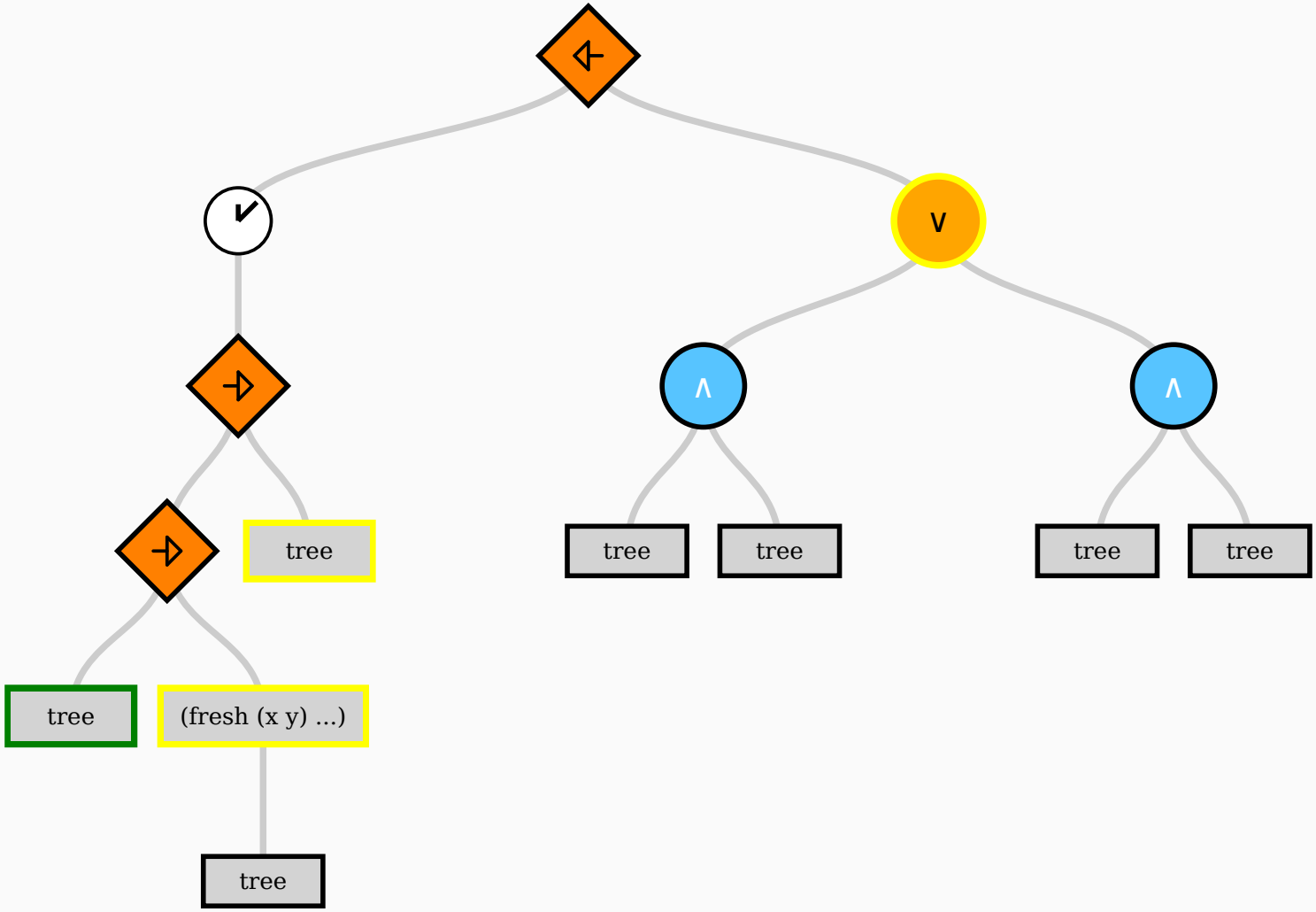
Railway Model



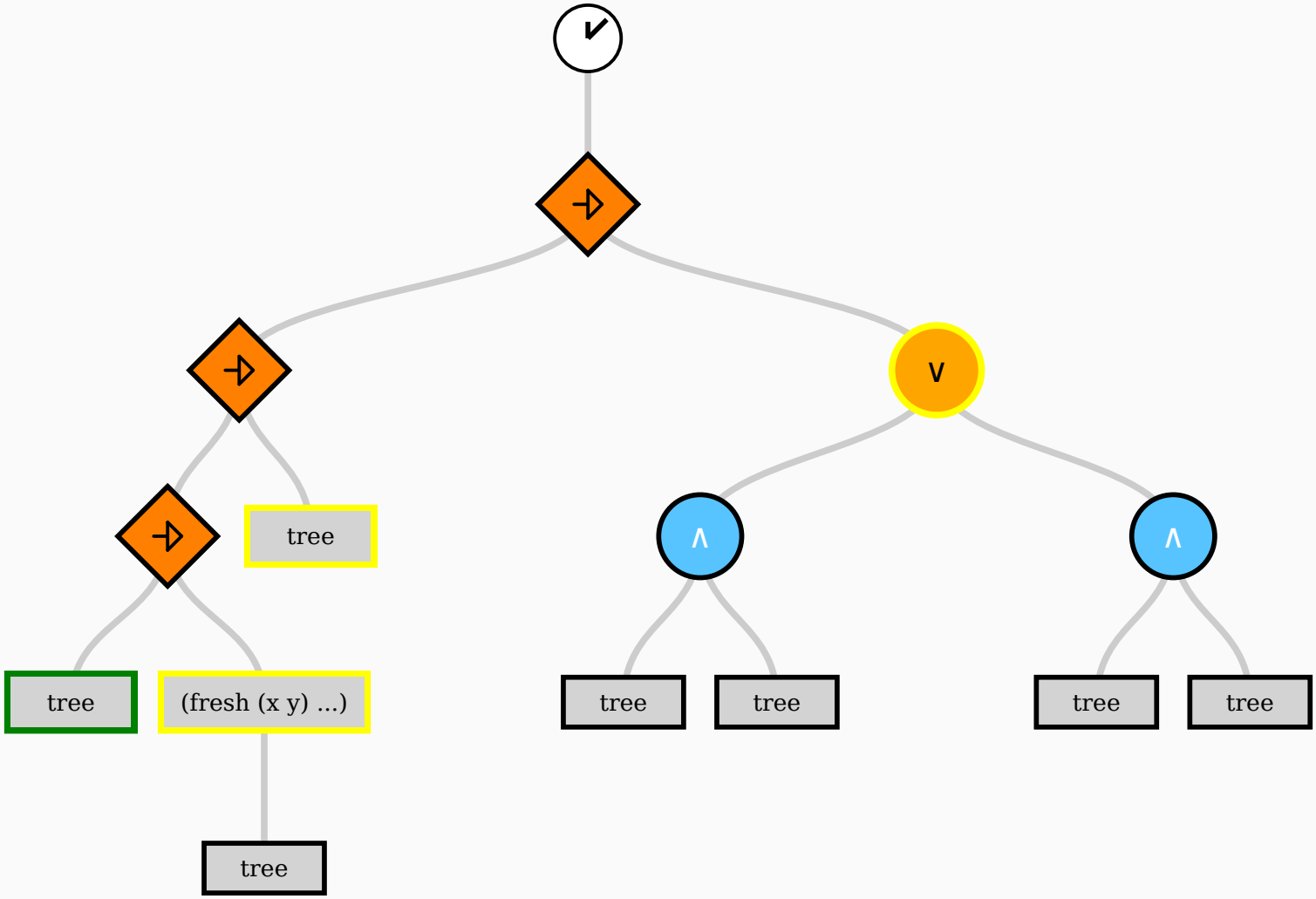
Railway Model



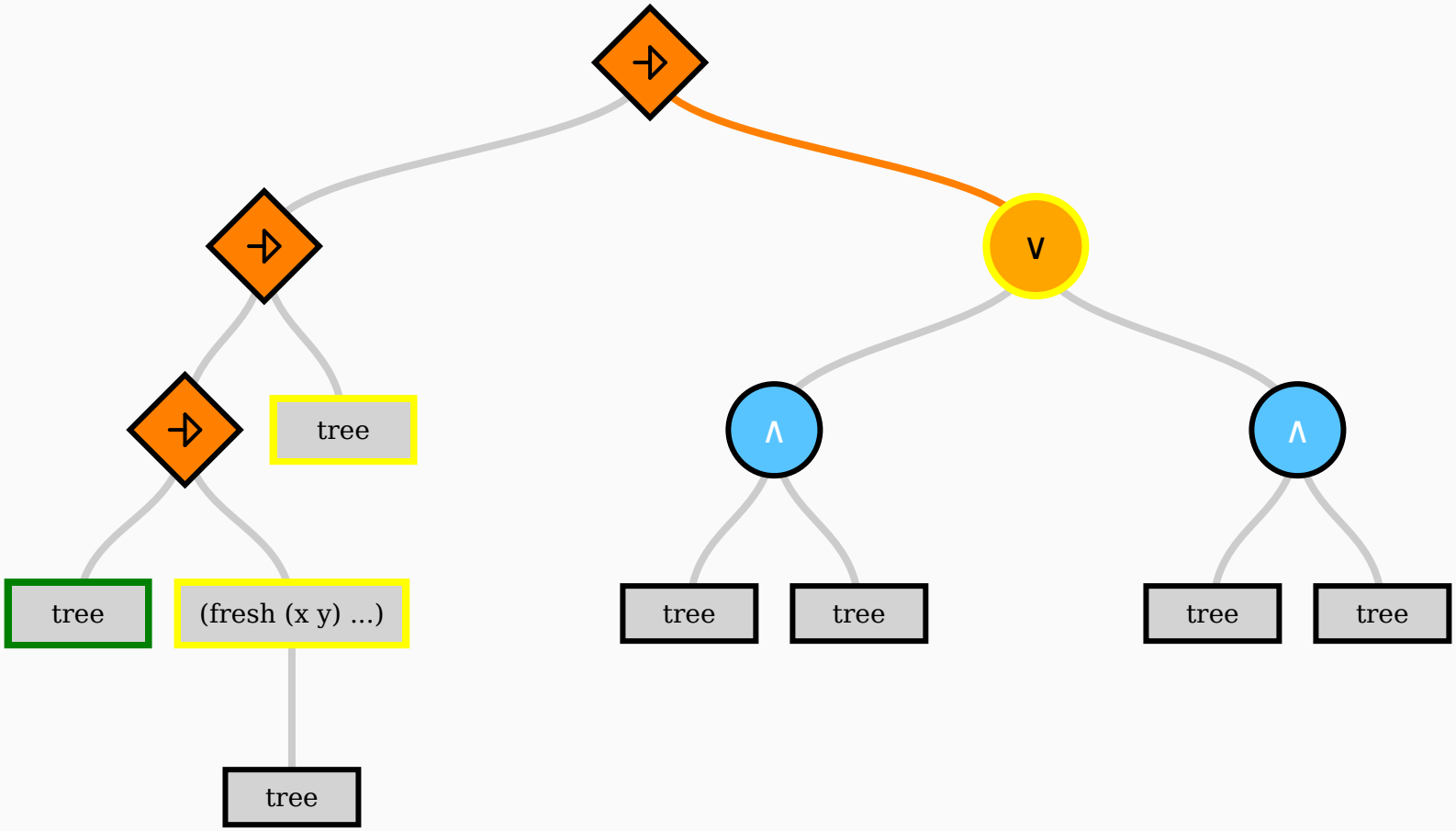
Railway Model



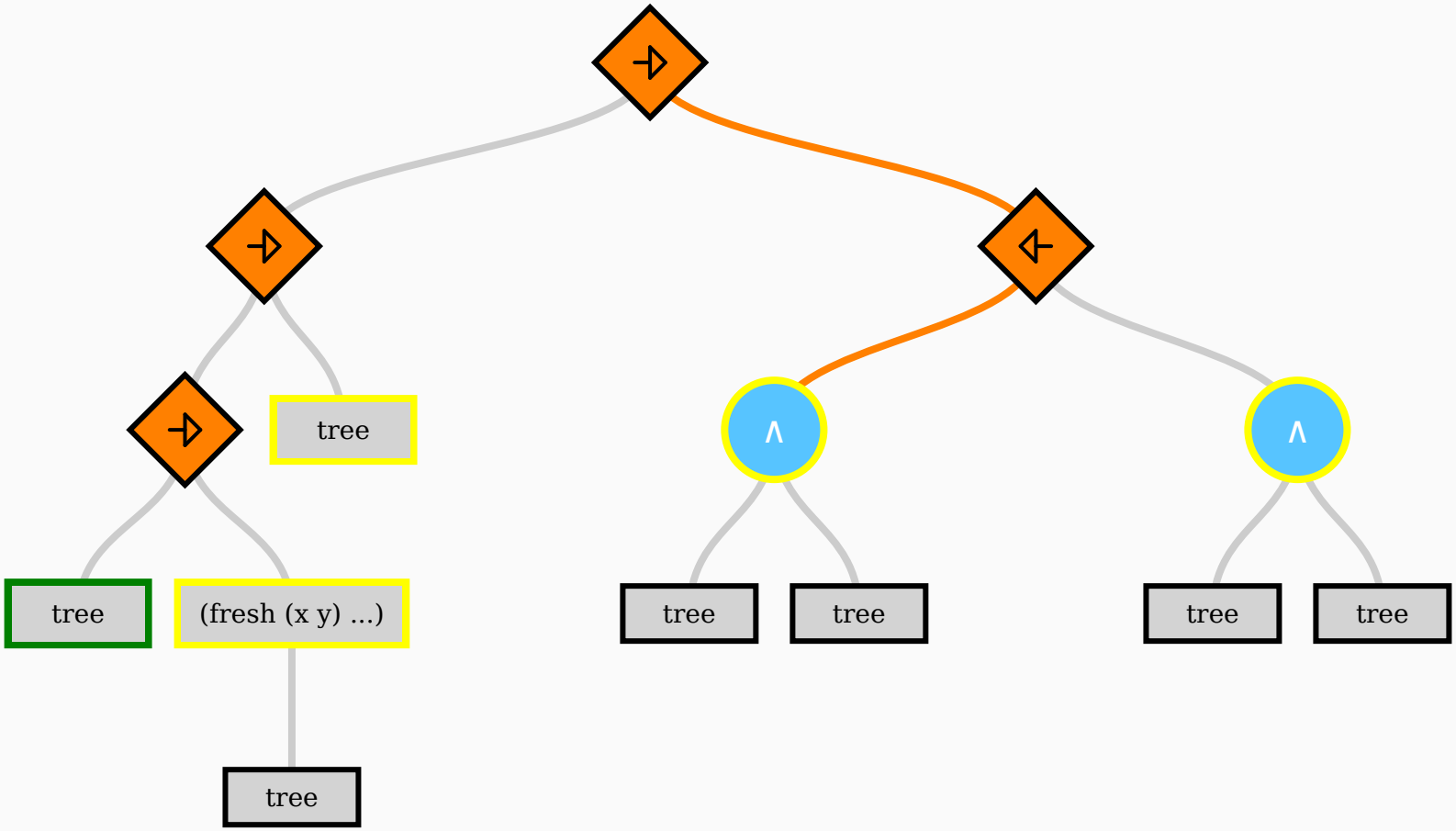
Railway Model



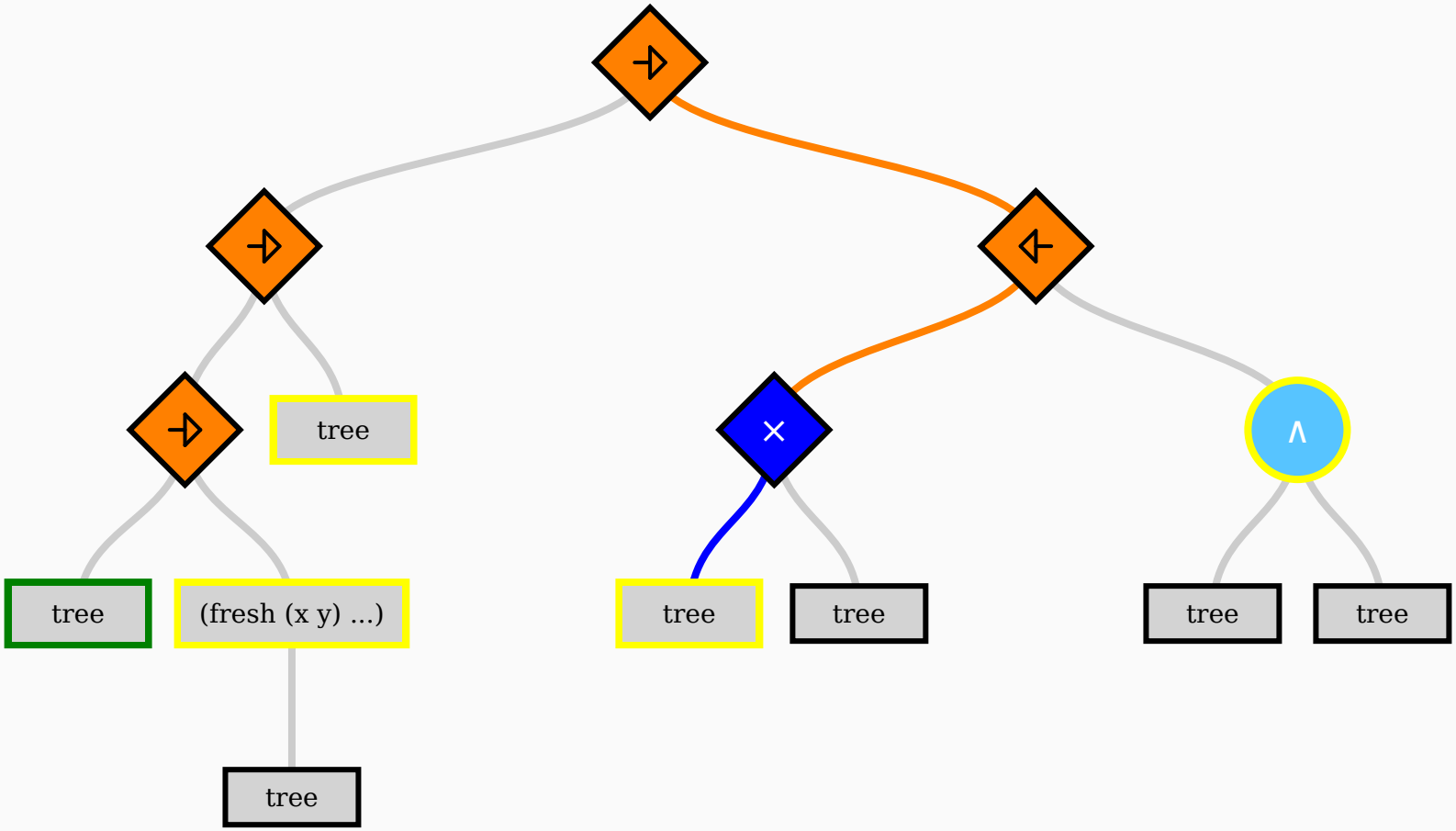
Railway Model



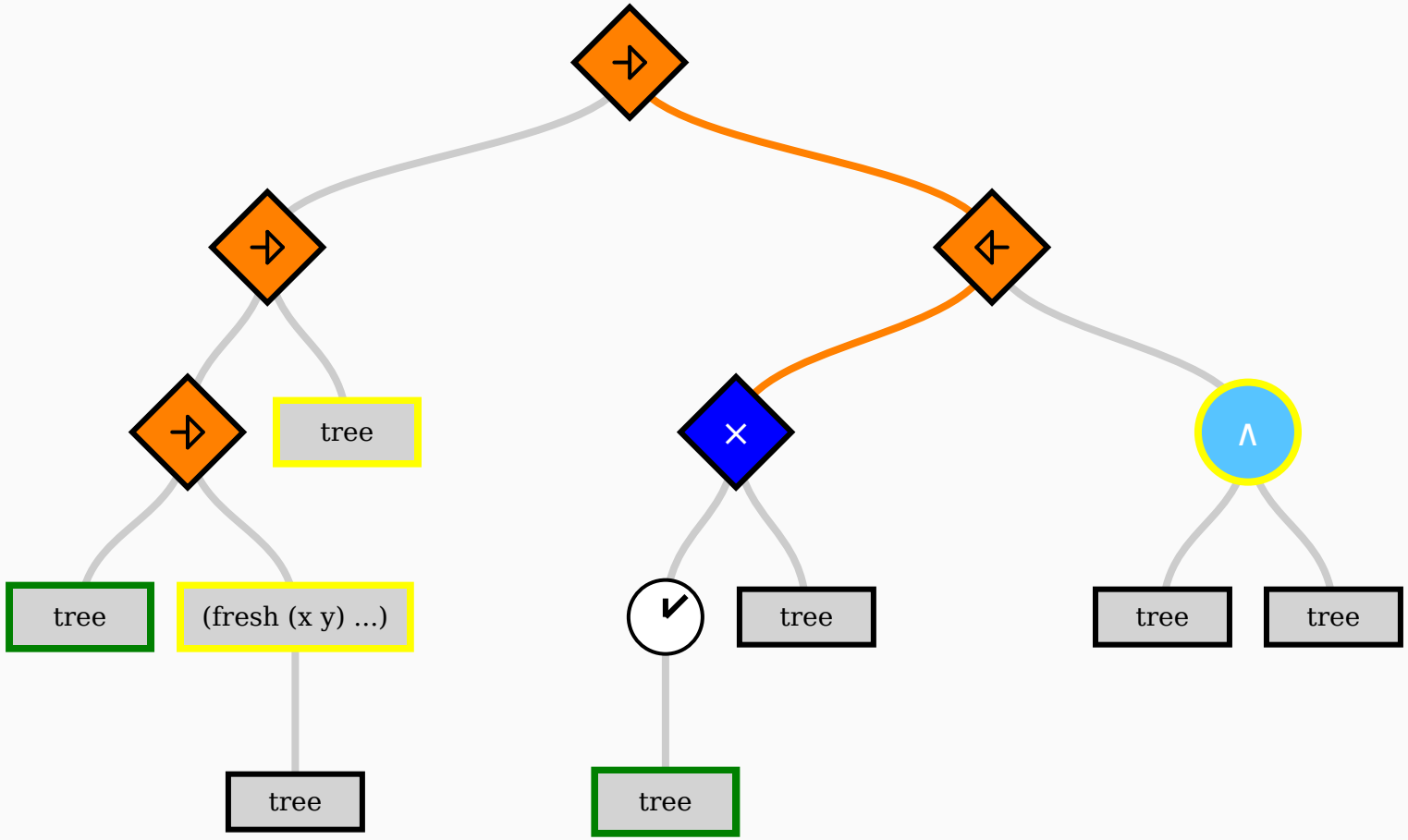
Railway Model



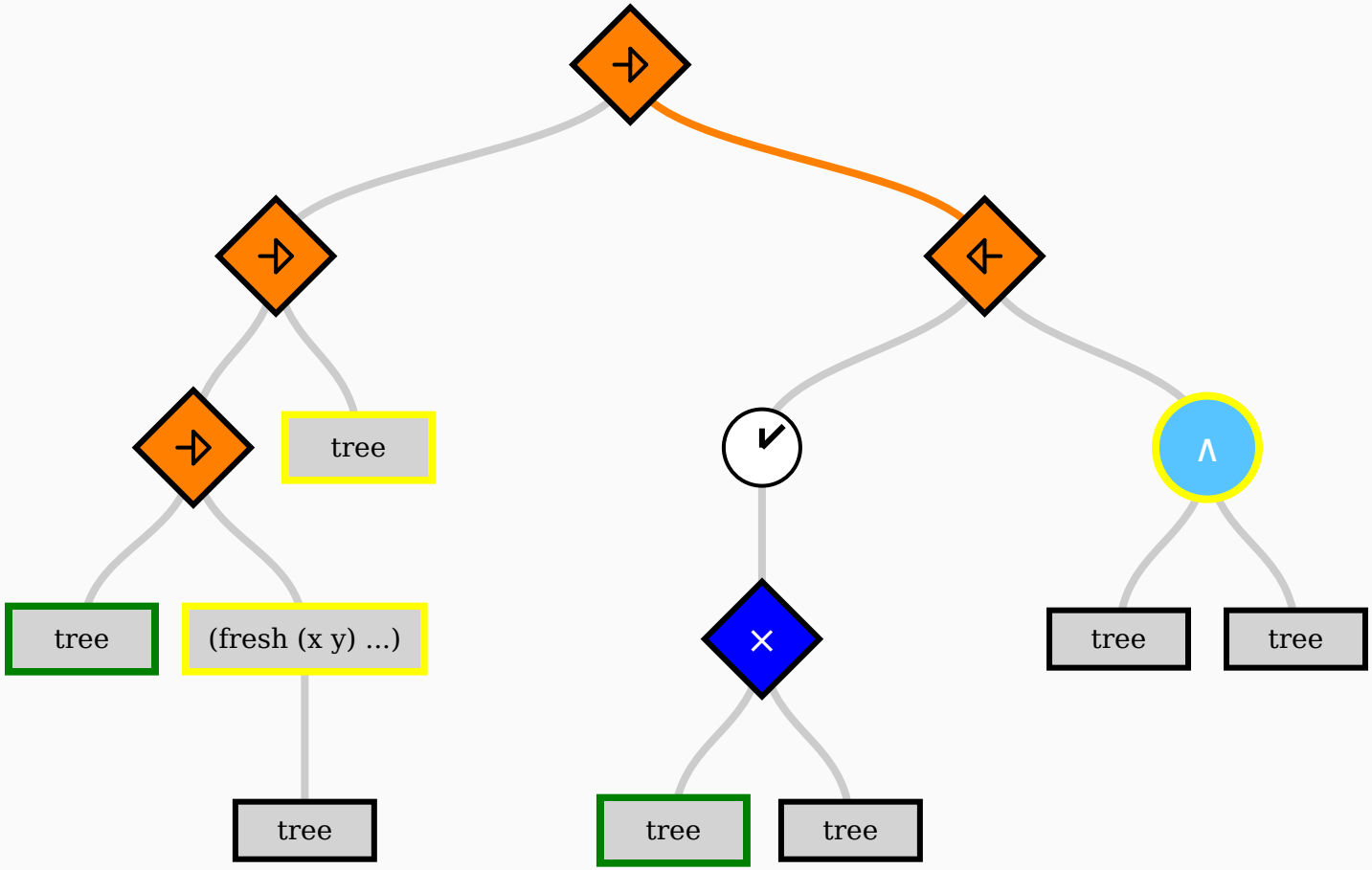
Railway Model



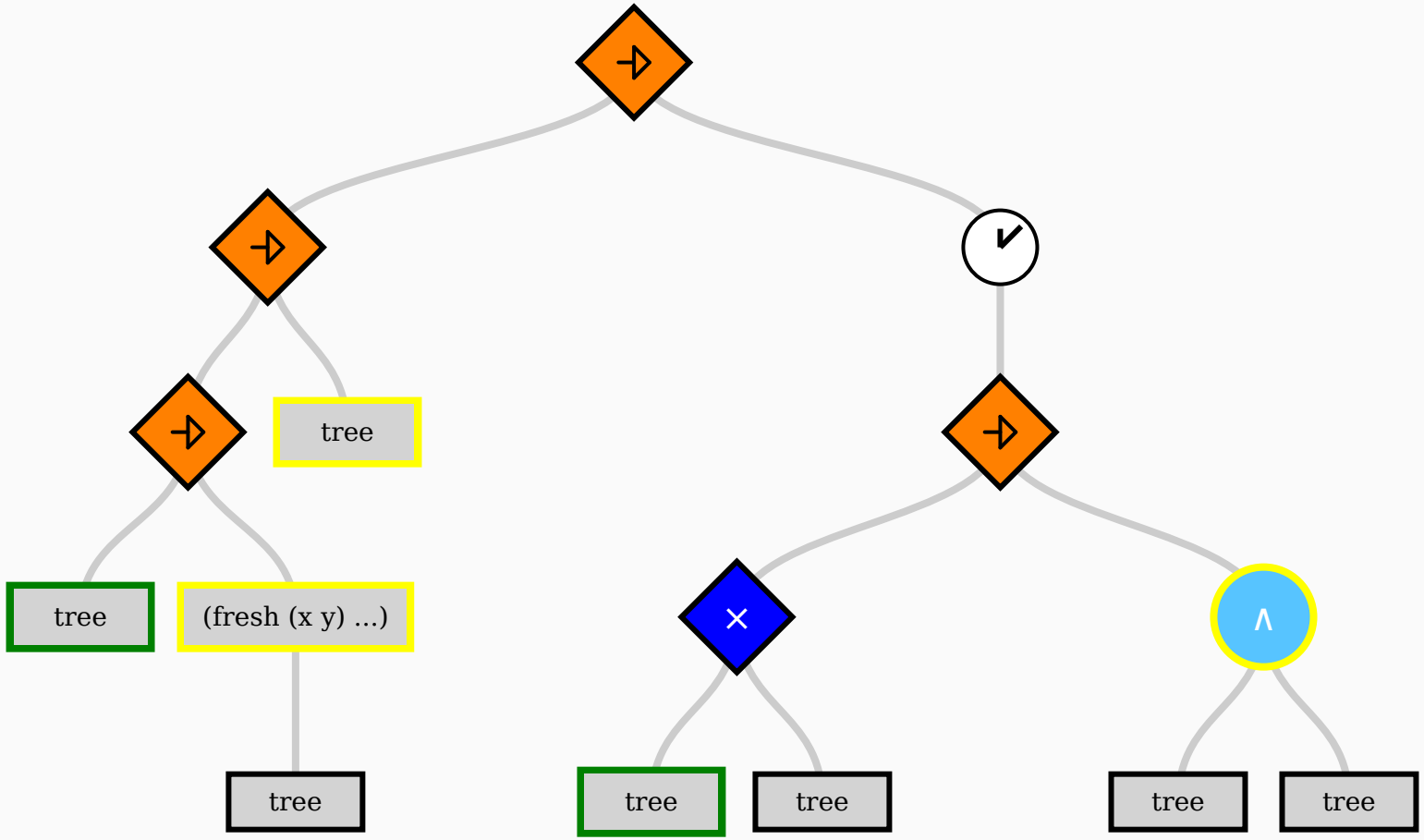
Railway Model



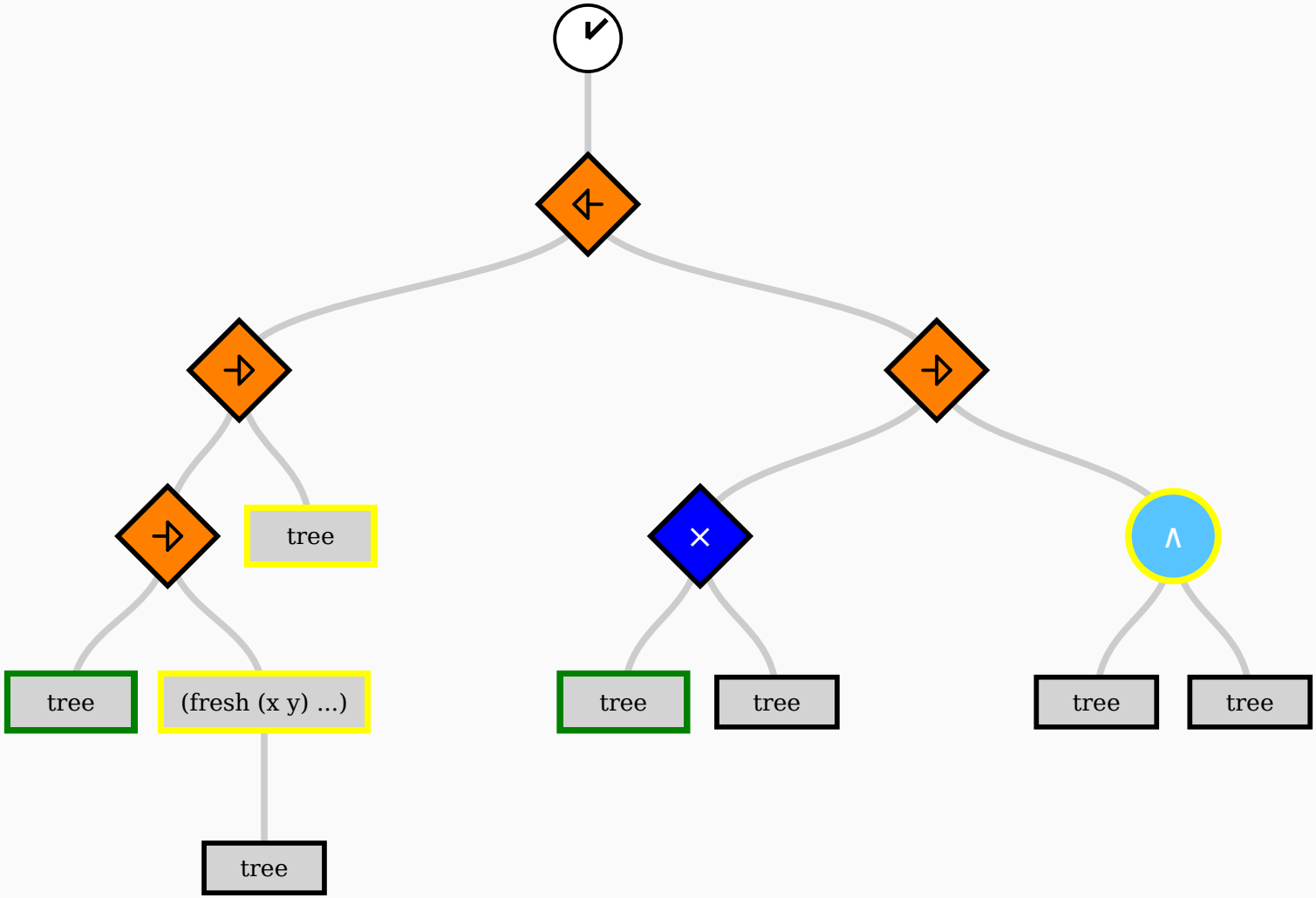
Railway Model



Railway Model



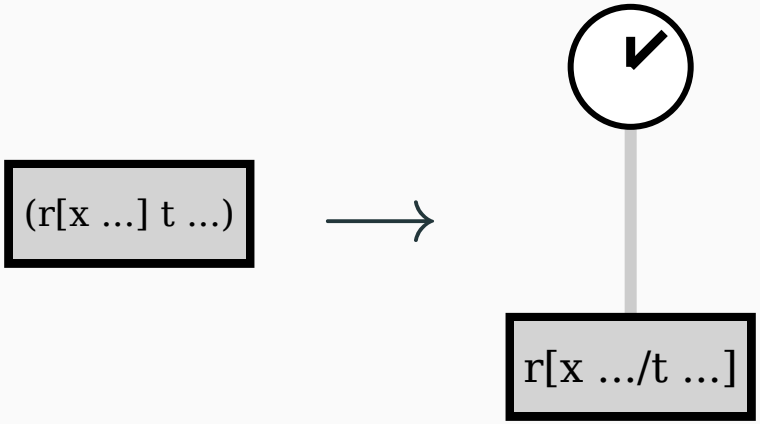
Railway Model



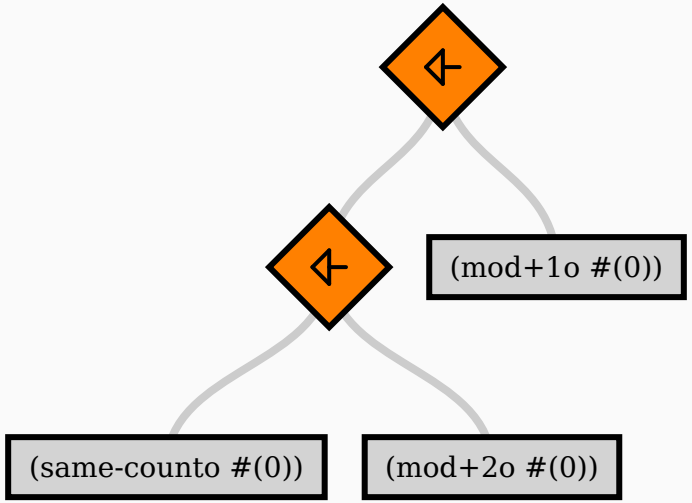
Lazy Relation Expansion

Old Delay

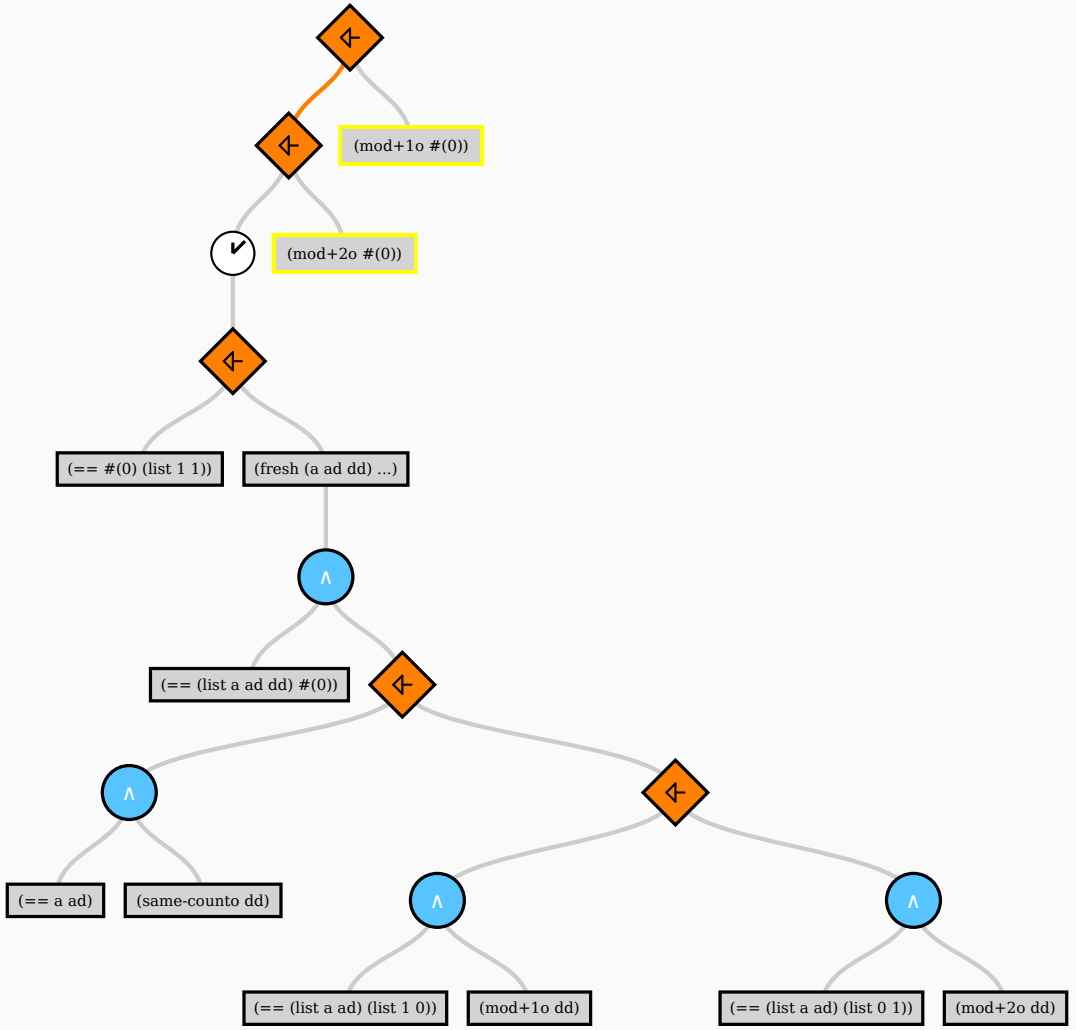
$$E[(r[x...]\ t...)\ \sigma] \longrightarrow E[\texttt{delay}\ \ r[x.../t...]\ \sigma]\ \text{[Delay (Old)]}$$



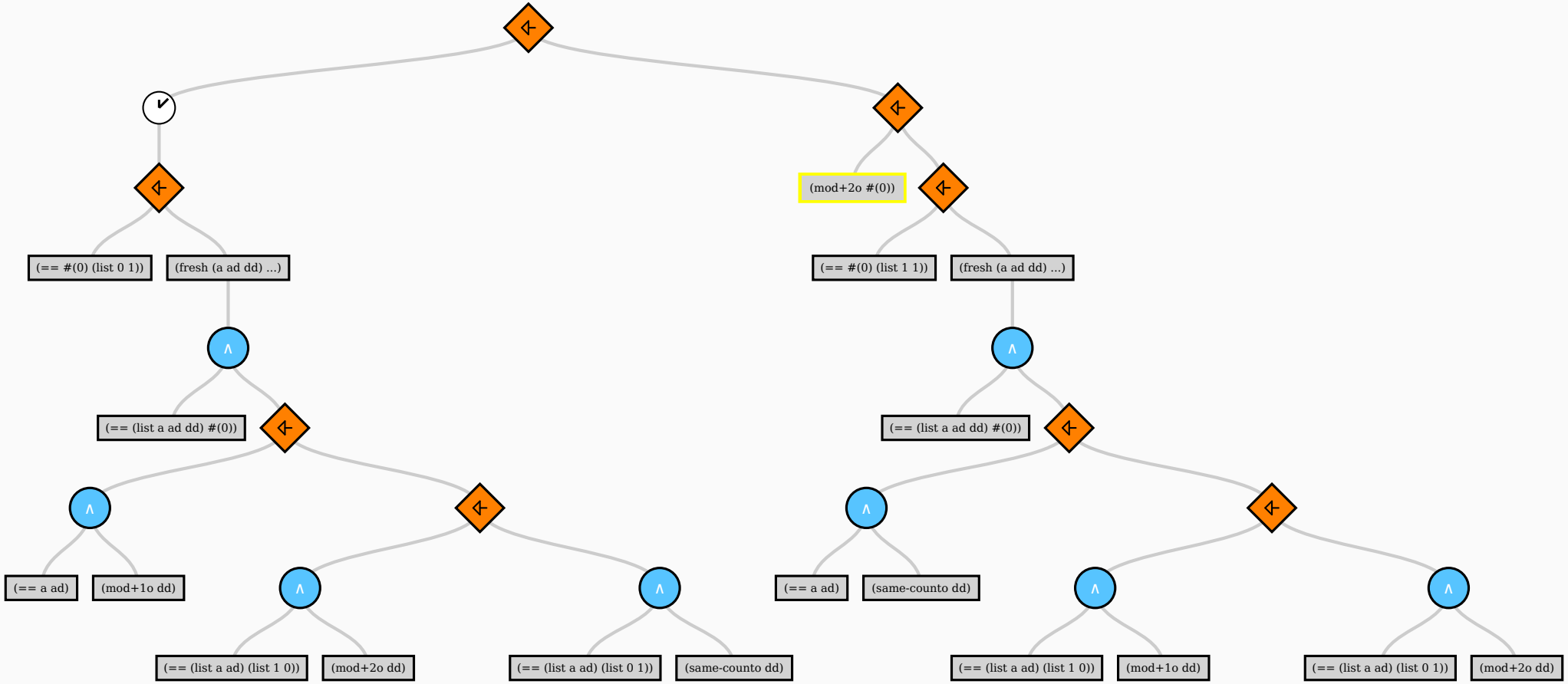
Old Delay



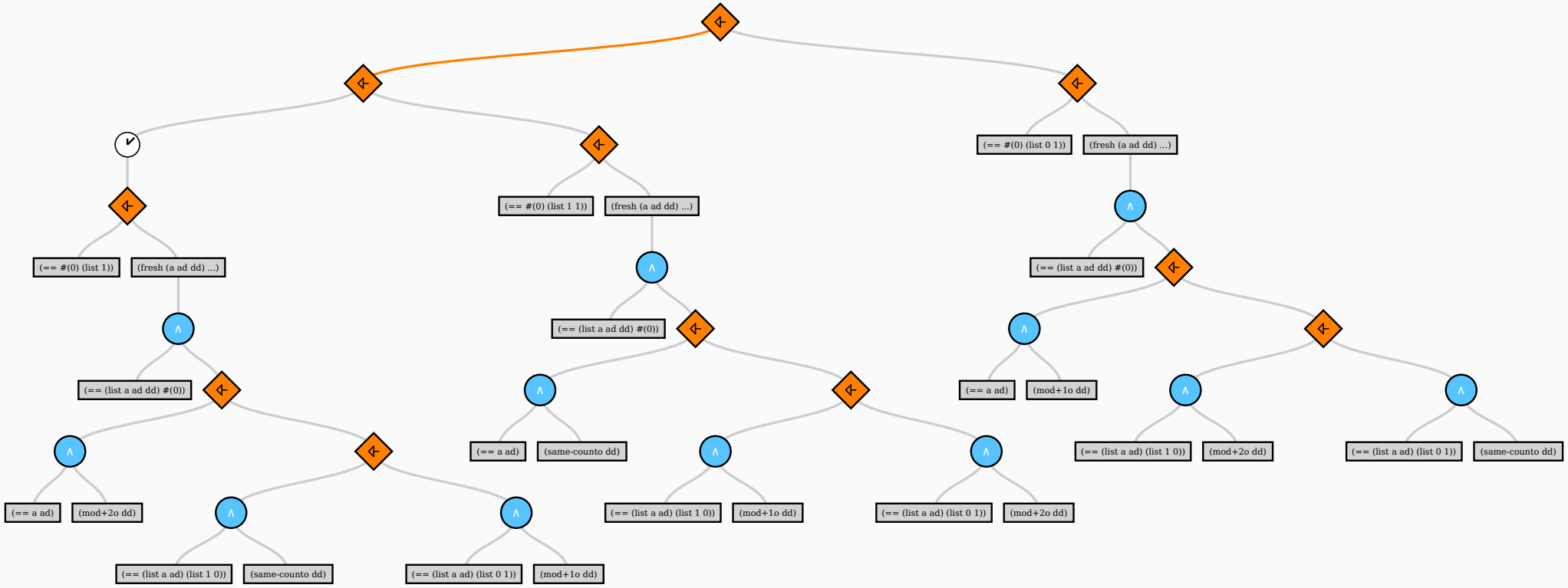
Old Delay


(mod+1o #(0))
(mod+2o #(0))
(== #(0) (list 1 1))
(fresh (a ad dd) ...)
∧
(== (list a ad dd) #(0))
∧
(== a ad)
(same-counto dd)
∧
(== (list a ad) (list 1 0))
(mod+1o dd)
∧
(== (list a ad) (list 0 1))
(mod+2o dd)

Old Delay

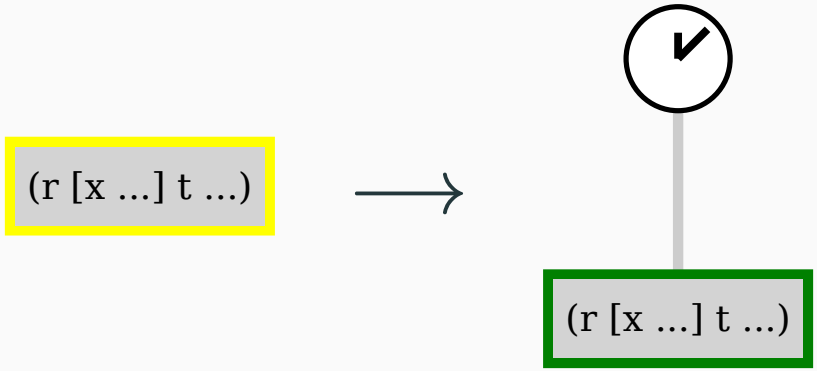

(== #(0) (list 0 1))
(fresh (a ad dd) ...)
(== (list a ad dd) #(0))
(== a ad)
(mod+1o dd)
(== (list a ad) (list 1 0))
(mod+2o dd)
(== (list a ad) (list 0 1))
(same-counto dd)
(mod+2o #(0))
(== #(0) (list 1 1))
(fresh (a ad dd) ...)
(== (list a ad dd) #(0))
(== a ad)
(same-counto dd)
(== (list a ad) (list 1 0))
(mod+1o dd)
(== (list a ad) (list 0 1))
(mod+2o dd)

Old Delay


(== #(0) (list 1))
(fresh (a ad dd) ...)
(== (list a ad dd) #(0))
(== a ad)
(mod+2o dd)
(== (list a ad) (list 1 0))
(same-counto dd)
(== (list a ad) (list 0 1))
(mod+1o dd)
(== #(0) (list 1 1))
(fresh (a ad dd) ...)
(== (list a ad dd) #(0))
(== a ad)
(same-counto dd)
(== (list a ad) (list 1 0))
(mod+1o dd)
(== #(0) (list 0 1))
(fresh (a ad dd) ...)
(== (list a ad dd) #(0))
(== a ad)
(mod+1o dd)
(== (list a ad) (list 1 0))
(mod+2o dd)
(== (list a ad) (list 0 1))
(same-counto dd)
(== (list a ad) (list 0 1))
(mod+2o dd)

Lazy Relation Expansion

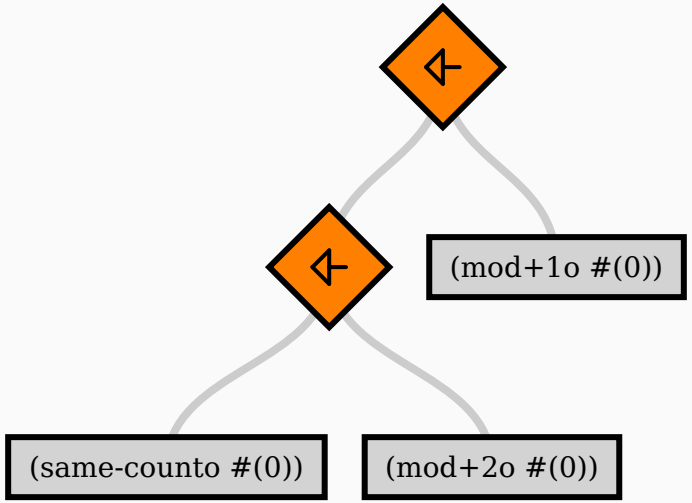
$$E[(r[x...]\ t...)\ \sigma] \longrightarrow E[\texttt{delay}\ (\texttt{go}\ (r[x...]\ t...)\ \sigma)]\ [\text{Delay}]$$



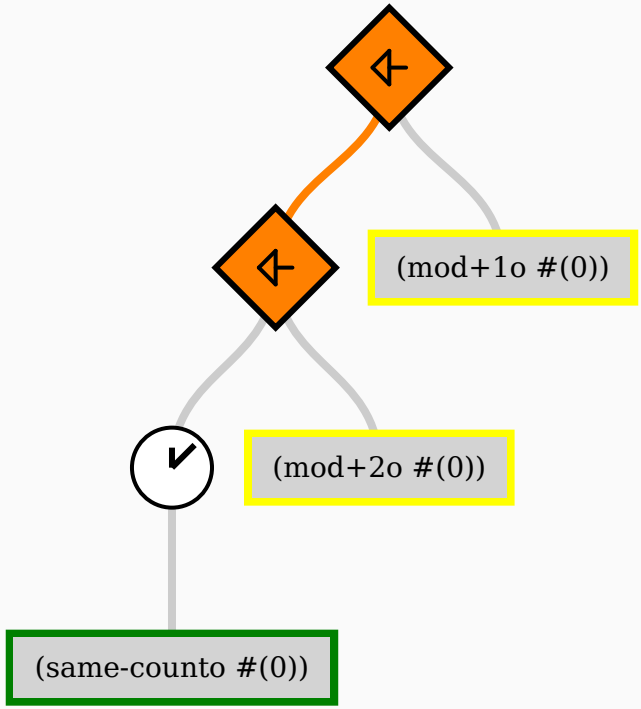
Lazy Relation Expansion

$$E[\texttt{go}\ (r[x\ldots]\ t\ldots)\ \sigma] \longrightarrow E[r[x\ldots/t\ldots]\ \sigma]\ \text{[Expand]}$$

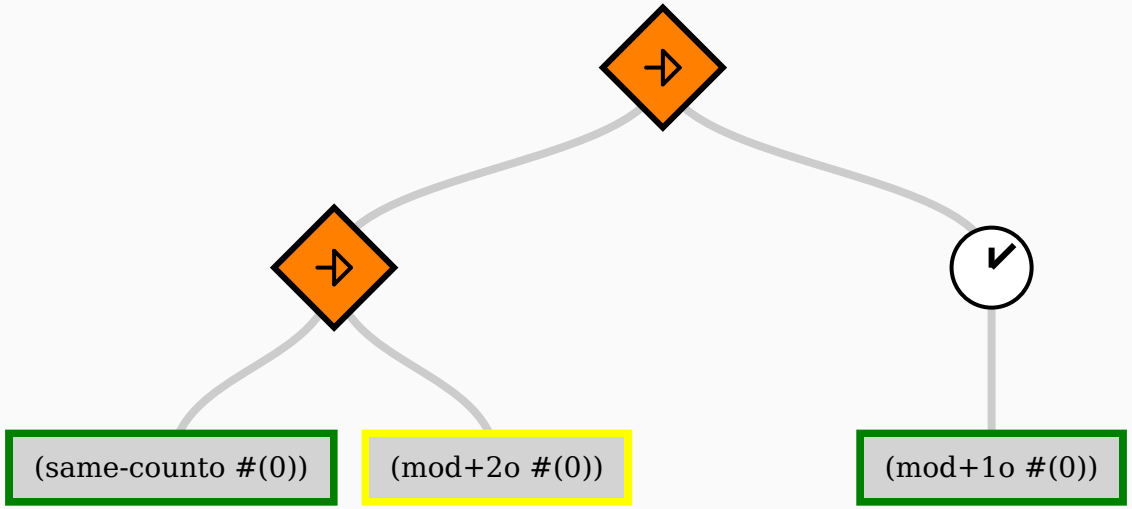
Lazy Relation Expansion



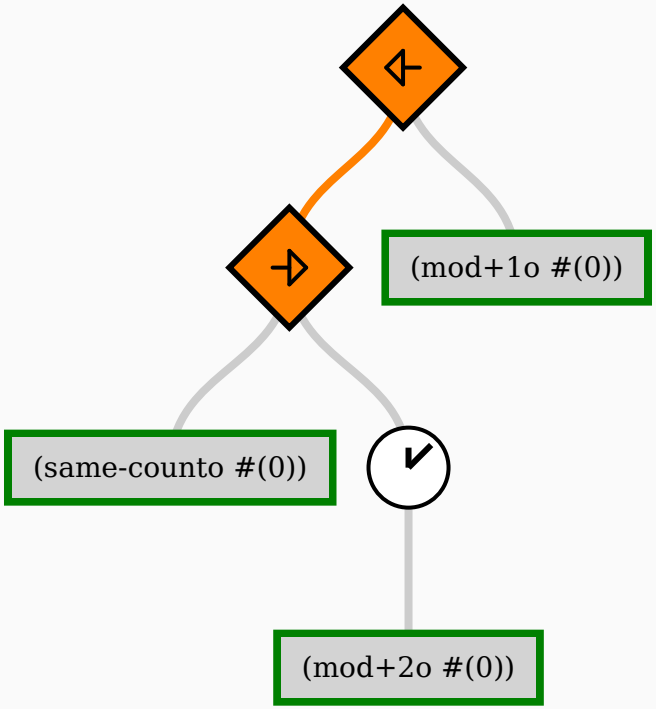
Lazy Relation Expansion



Lazy Relation Expansion



Lazy Relation Expansion

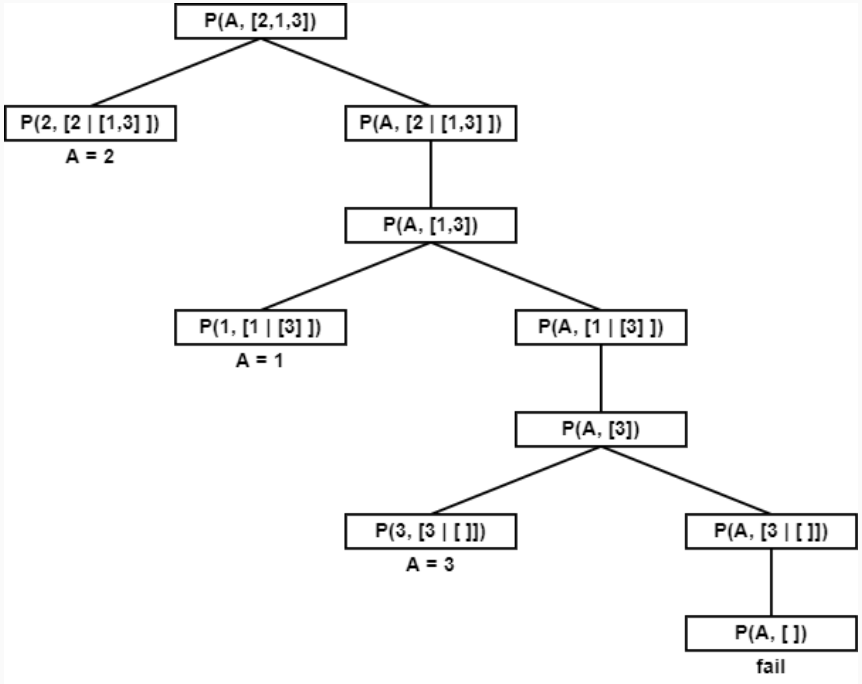


Demo

Existing Logic Programming Tooling (Prolog)

```
[trace]  ?- member(A, [2, 1, 3]).

Call : (12) member(_41716, [2, 1, 3])
Exit : (12) member(2, [2, 1, 3])

A = 2 ;

Redo : (12) member(_41716, [2, 1, 3])
Call : (13) member(_41716, [1, 3])
Exit : (13) member(1, [1, 3])
Exit : (12) member(1, [2, 1, 3])

A = 1 ;

Redo : (13) member(_41716, [1, 3])
Call : (14) member(_41716, [3])
Exit : (14) member(3, [3])
Exit : (13) member(3, [1, 3])
Exit : (12) member(3, [2, 1, 3])

A = 3 ;

Redo : (14) member(_41716, [3])
Call : (15) member(_41716, [])
Fail : (15) member(_41716, [])
Fail : (14) member(_41716, [3])
Fail : (13) member(_41716, [1, 3])
Fail : (12) member(_41716, [2, 1, 3])

false.
```

Byrd Box



AND/OR Tree

Existing Logic Programming Tooling (miniKanren)

```
================================================================================
Current Depth: 2              Number of Choices: 2

| Choice 1:
|   x = (1)
|   y = (1 2 3)
|   No constraints

| Choice 2:
|   x = (1 1 . #s(var b 21))
|   y = #s(var y 16)
|   Constraints:
|   * (appendoh #s(var b 21) #s(var y 16) (1 2 3))

[h]elp, [u]ndo, or choice number>
```

Textual Interactive Stepper - Rosenblatt et al. 2019

Conclusion

Future Directions

1. **Composable Search Strategies / Schedulers**
2. **Comparative Semantics**
3. **Mechanical Verification**

Conclusion

- **Semantics:** We define a fine-grained, deterministic small-step operational semantics making miniKanren's interleaving strategy explicit.
- **Visualizer:** A step-by-step tool exposes search control, interleaving choices, and suspended goals.

github.com/brysenPfingsten/mk-redex-viz

minikanrenredex-prod.shu.edu

Thank You!