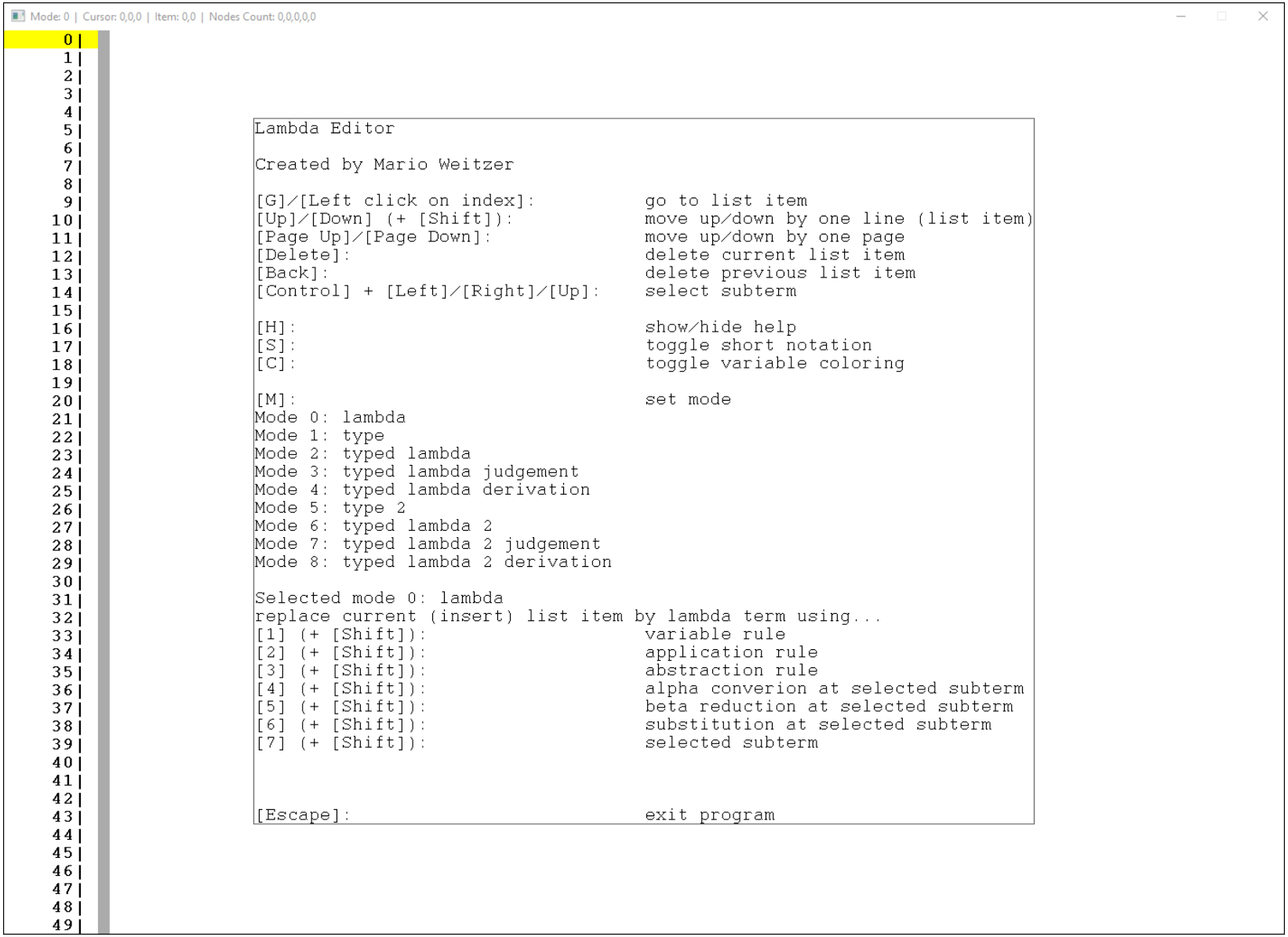
Formalizing p-adic arithmetic and dynamics in Lean (theorem prover)

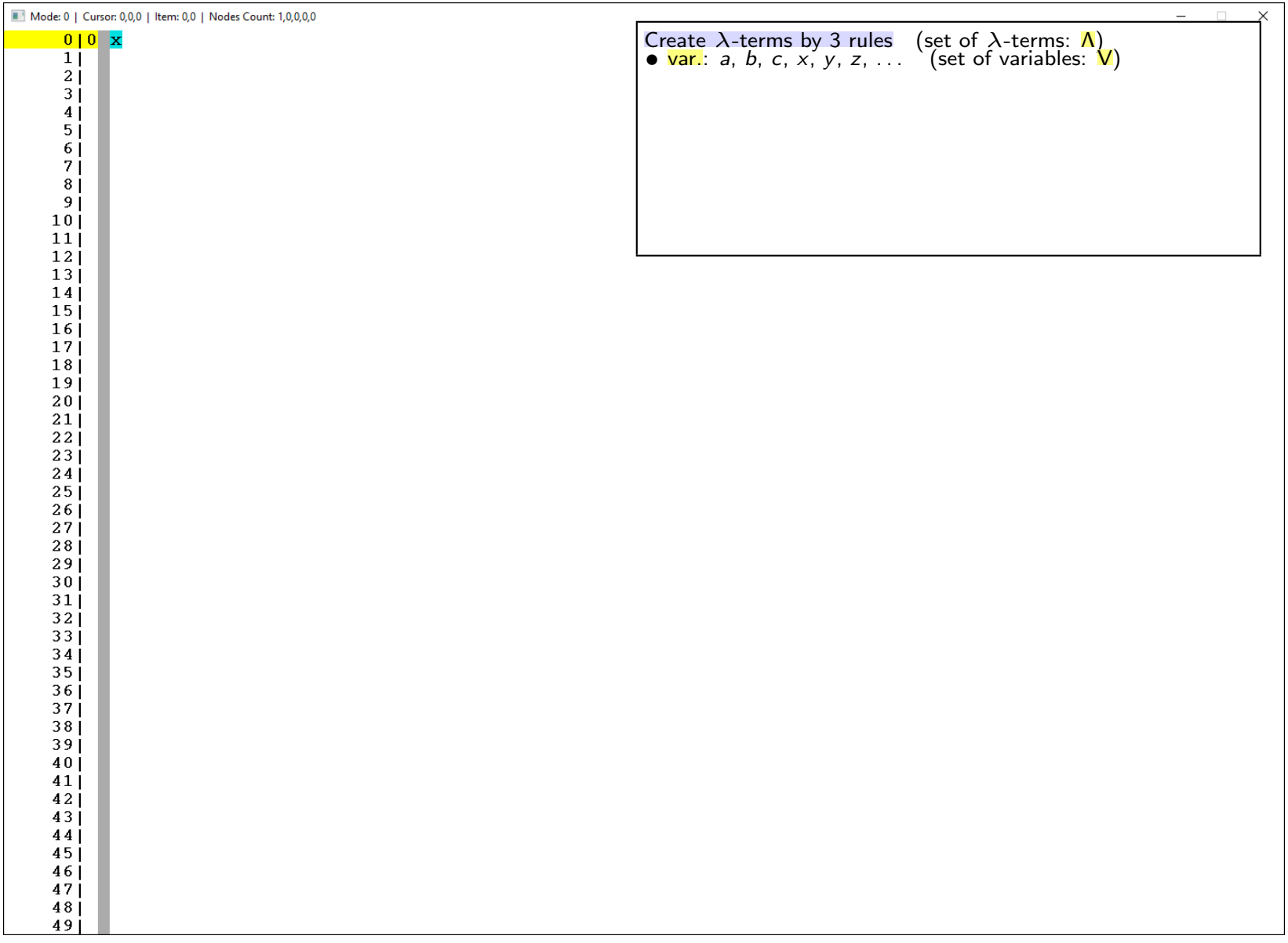
Mario Weitzer

University of Leoben, Austria

Advanced Topics in Discrete Mathematics
Graz, October 11, 2024

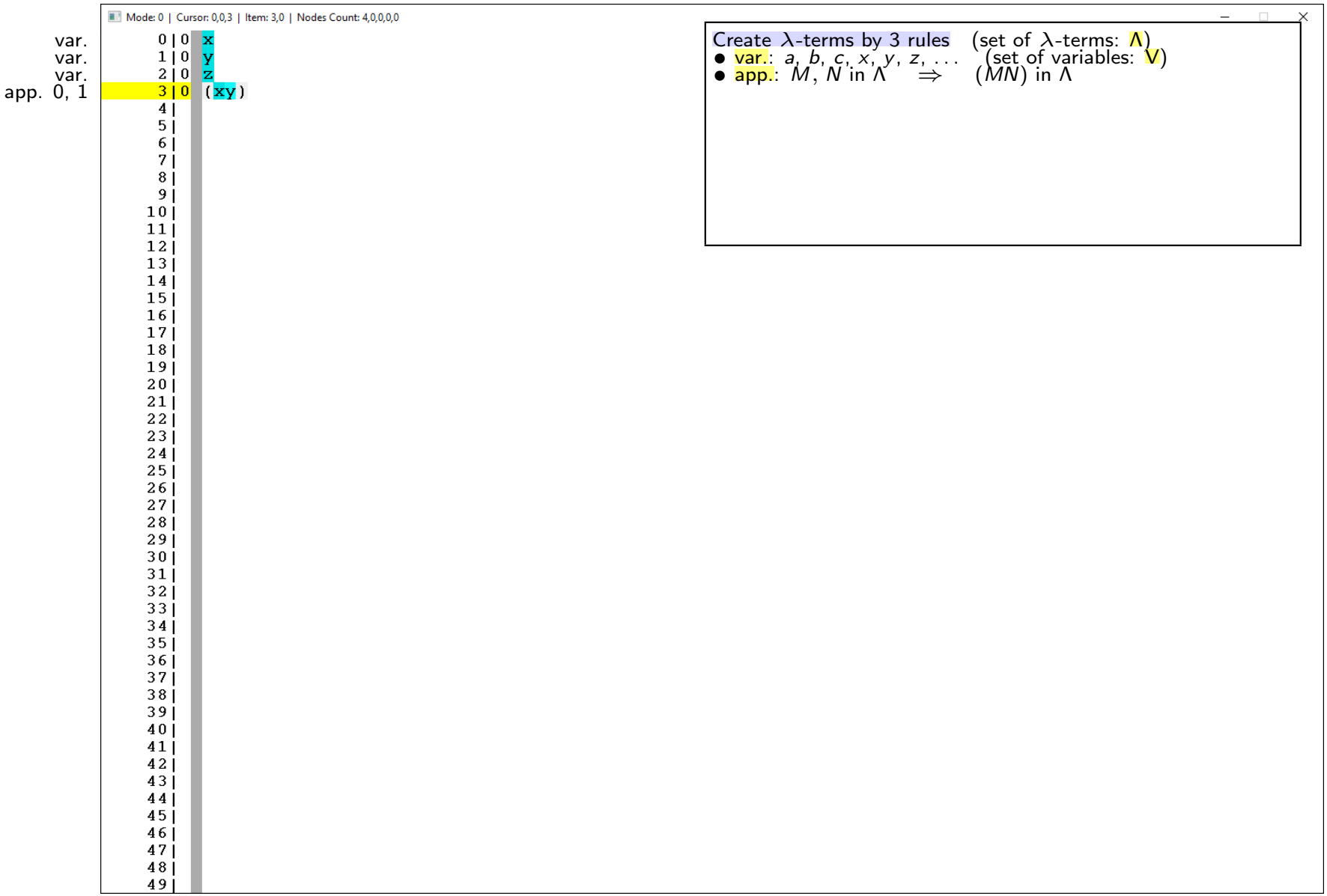

Mode: 0 | Cursor: 0,0,0 | Item: 0,0 | Nodes Count: 0,0,0,0,0
Lambda Editor
Created by Mario Weitzer
[G]/[Left click on index]: go to list item
[Up]/[Down] (+ [Shift]): move up/down by one line (list item)
[Page Up]/[Page Down]: move up/down by one page
[Delete]: delete current list item
[Back]: delete previous list item
[Control] + [Left]/[Right]/[Up]: select subterm
[H]: show/hide help
[S]: toggle short notation
[C]: toggle variable coloring
[M]: set mode
Mode 0: lambda
Mode 1: type
Mode 2: typed lambda
Mode 3: typed lambda judgement
Mode 4: typed lambda derivation
Mode 5: type 2
Mode 6: typed lambda 2
Mode 7: typed lambda 2 judgement
Mode 8: typed lambda 2 derivation
Selected mode 0: lambda
replace current (insert) list item by lambda term using...
[1] (+ [Shift]): variable rule
[2] (+ [Shift]): application rule
[3] (+ [Shift]): abstraction rule
[4] (+ [Shift]): alpha converion at selected subterm
[5] (+ [Shift]): beta reduction at selected subterm
[6] (+ [Shift]): substitution at selected subterm
[7] (+ [Shift]): selected subterm
[Escape]: exit program

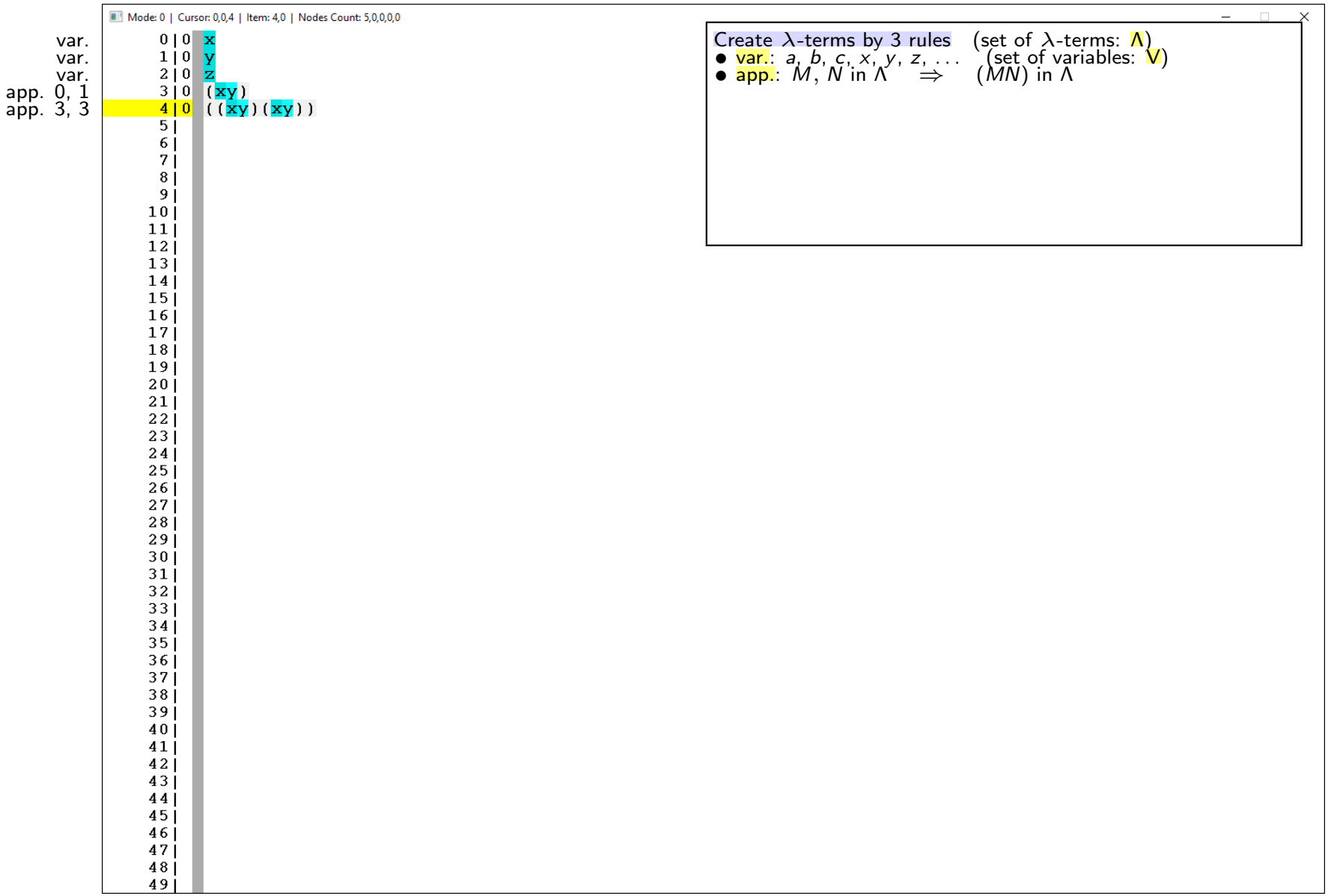
var.

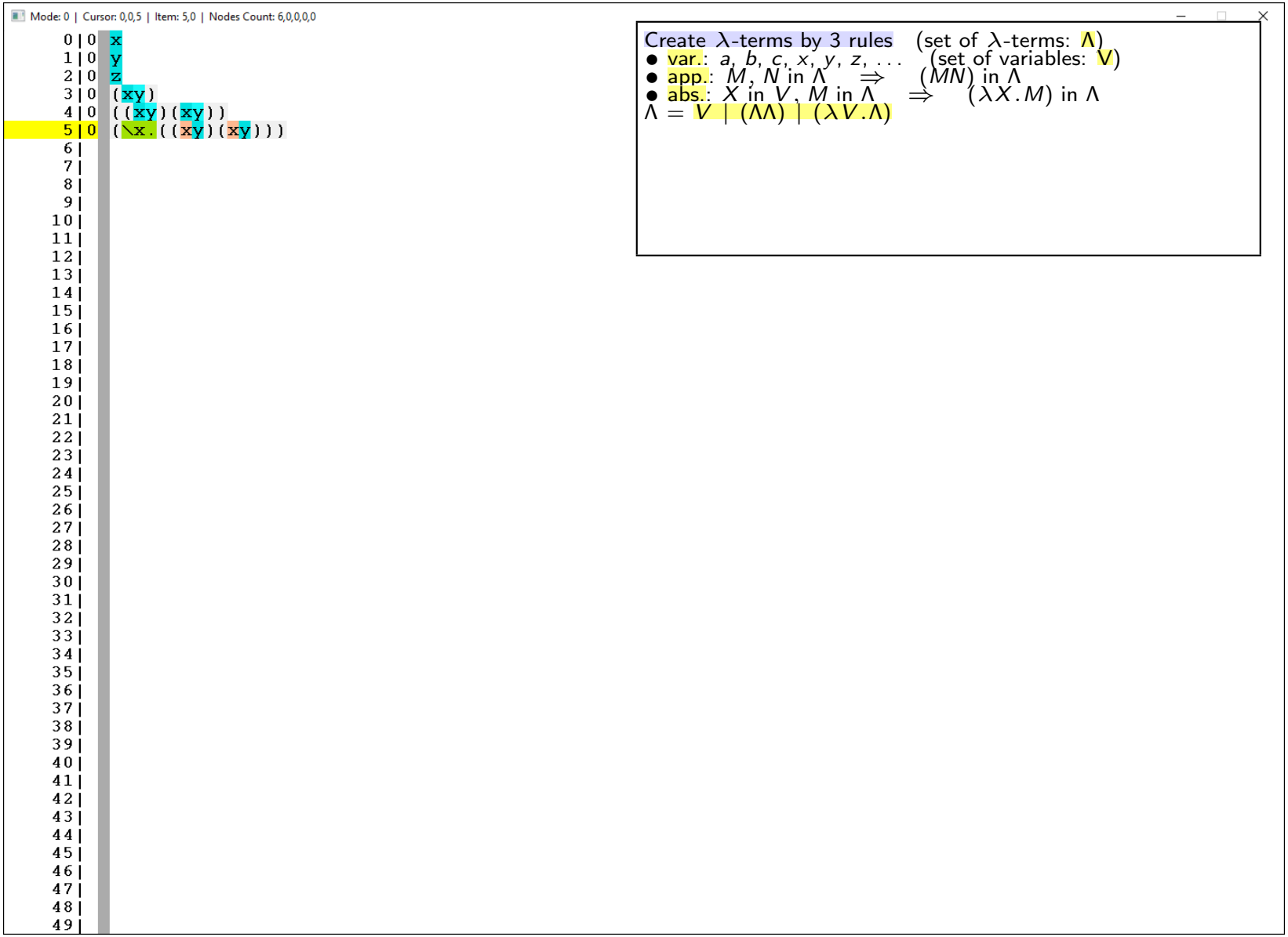

Mode: 0 | Cursor: 0,0,0 | Item: 0,0 | Nodes Count: 1,0,0,0,0
0 | 0 x
Create λ-terms by 3 rules (set of λ-terms: Λ)
● var.: a, b, c, x, y, z, ... (set of variables: V)

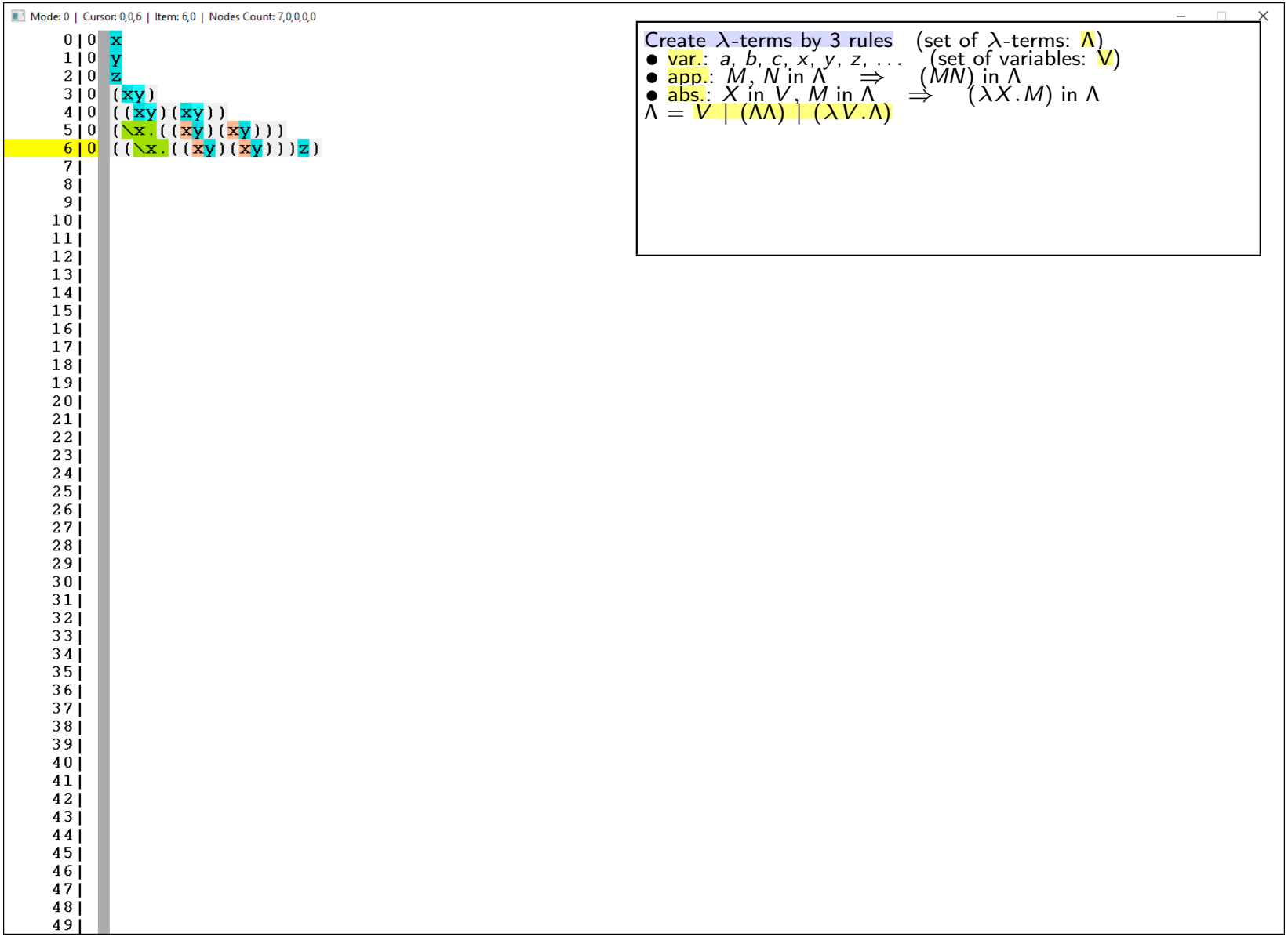

Mode: 0 | Cursor: 0,0,1 | Item: 1,0 | Nodes Count: 2,0,0,0,0
var.
var.
0 | 0 x
1 | 0 y
Create λ-terms by 3 rules (set of λ-terms: Λ)
● var.: a, b, c, x, y, z, ... (set of variables: V)

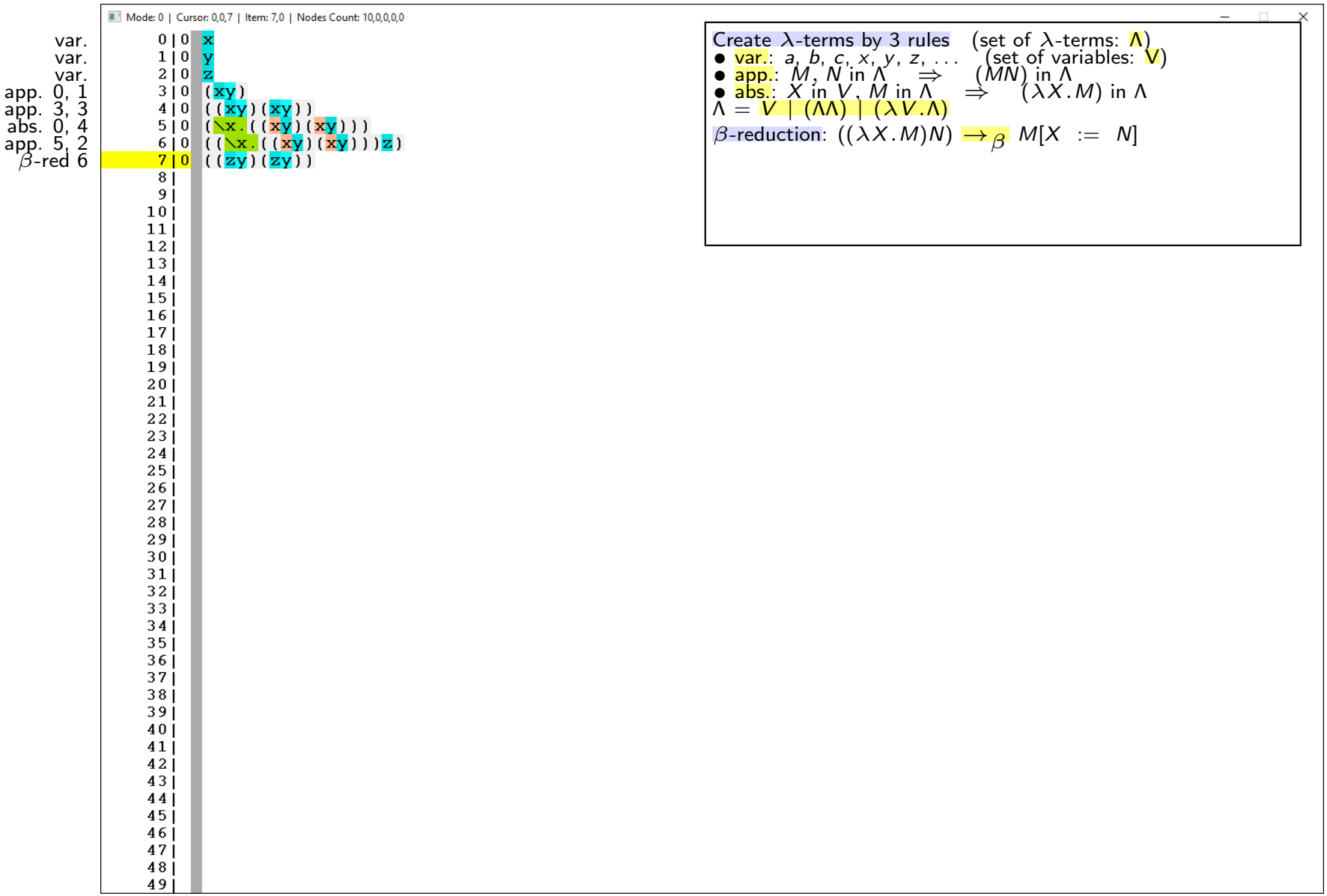

Mode: 0 | Cursor: 0,0,2 | Item: 2,0 | Nodes Count: 3,0,0,0,0
var.
var.
var.
0 | 0 x
1 | 0 y
2 | 0 z
Create λ-terms by 3 rules (set of λ-terms: Λ)
● var.: a, b, c, x, y, z, ... (set of variables: V)

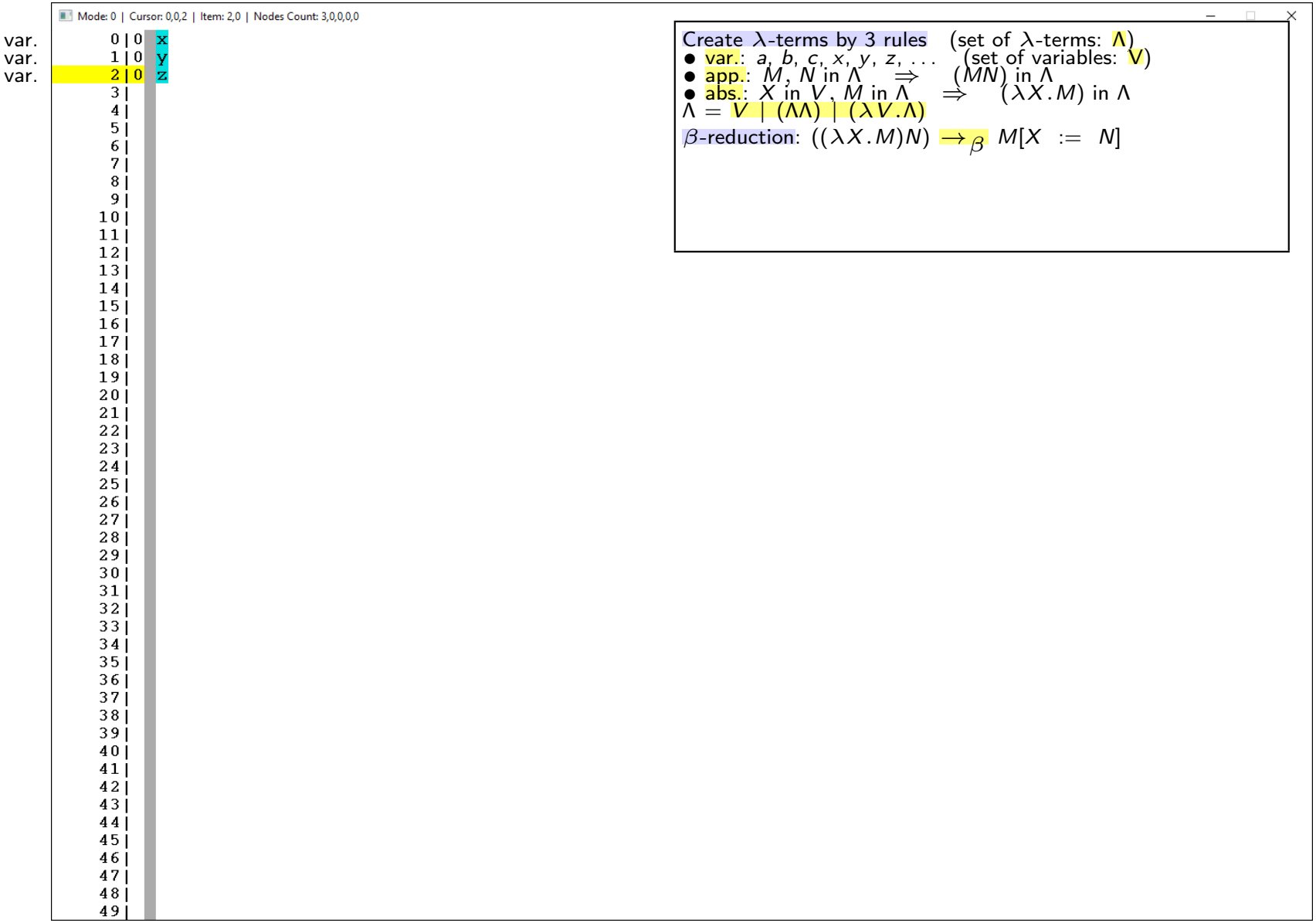

Mode: 0 | Cursor: 0,0,3 | Item: 3,0 | Nodes Count: 4,0,0,0,0
var.
var.
var.
app. 0, 1
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 (xy)
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ

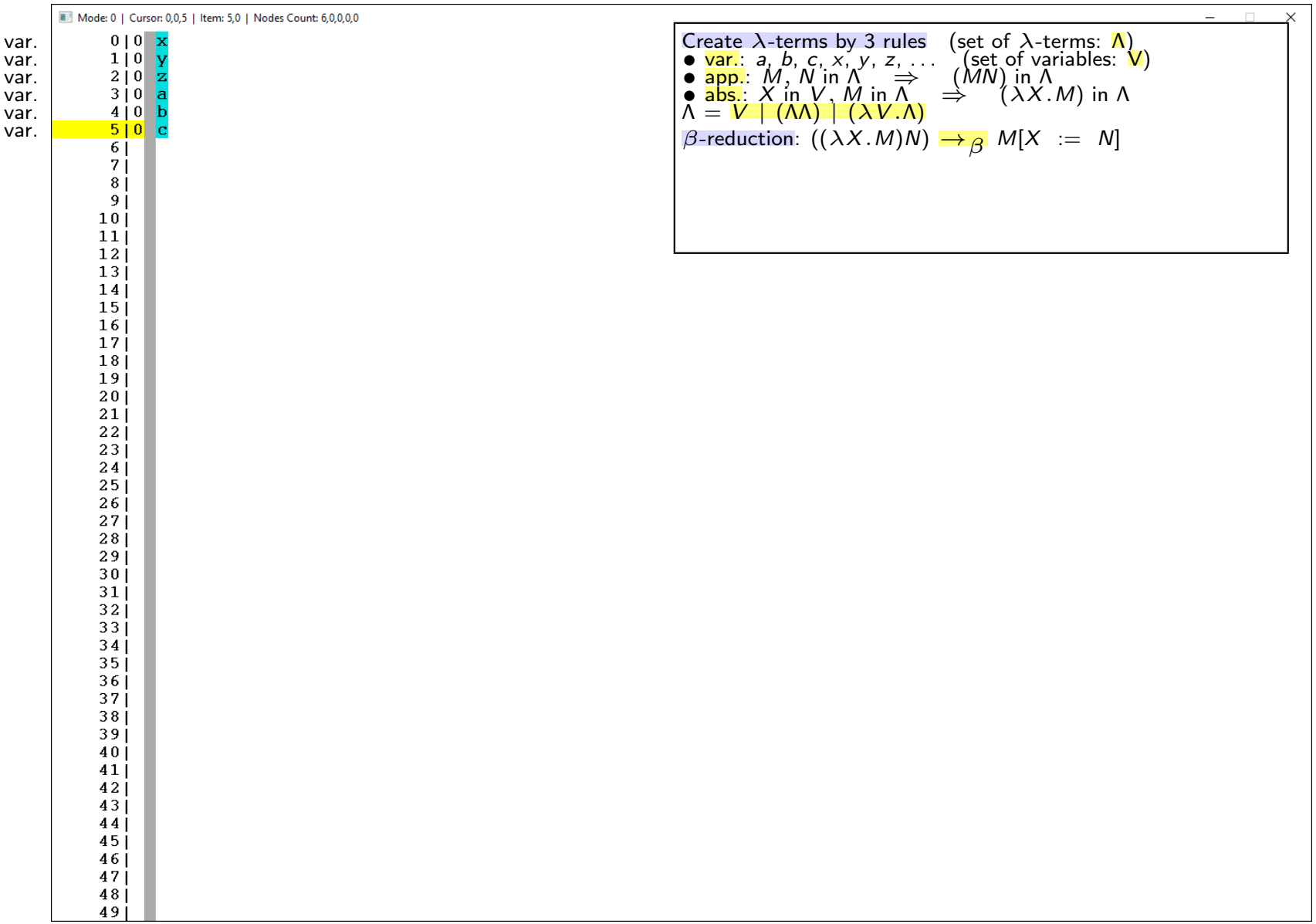

Mode: 0 | Cursor: 0,0,4 | Item: 4,0 | Nodes Count: 5,0,0,0,0
var.
var.
var.
app. 0, 1
app. 3, 3
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 (xy)
4 | 0 ((xy)(xy))
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ

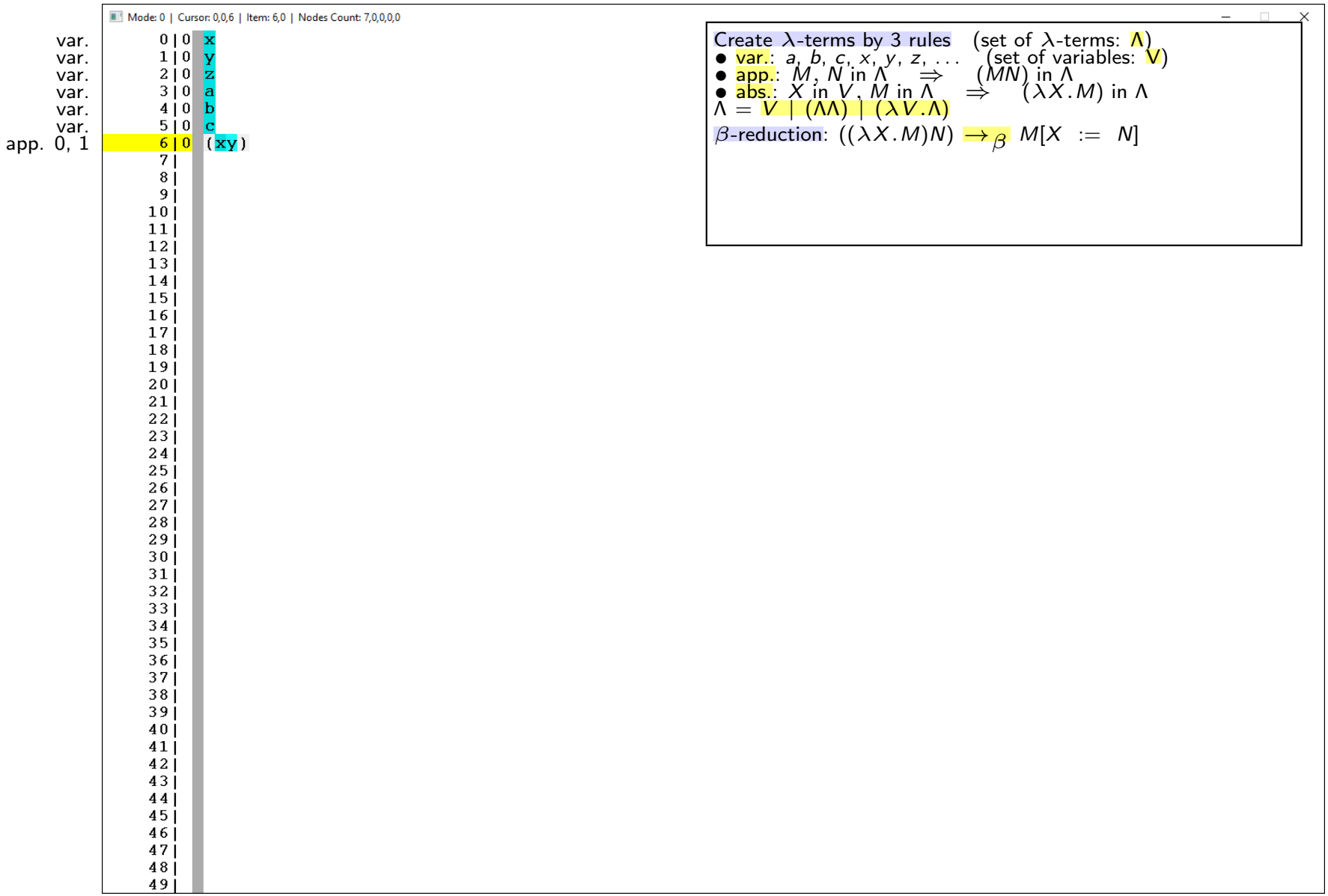

Mode: 0 | Cursor: 0,0,5 | Item: 5,0 | Nodes Count: 6,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
app. 0, 1 3 | 0 (xy)
app. 3, 3 4 | 0 ((xy)(xy))
abs. 0, 4 5 | 0 (\x.((xy)(xy)))
Create λ-terms by 3 rules (set of λ-terms: Λ)
● var.: a, b, c, x, y, z, ... (set of variables: V)
● app.: M, N in Λ ⇒ (MN) in Λ
● abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)

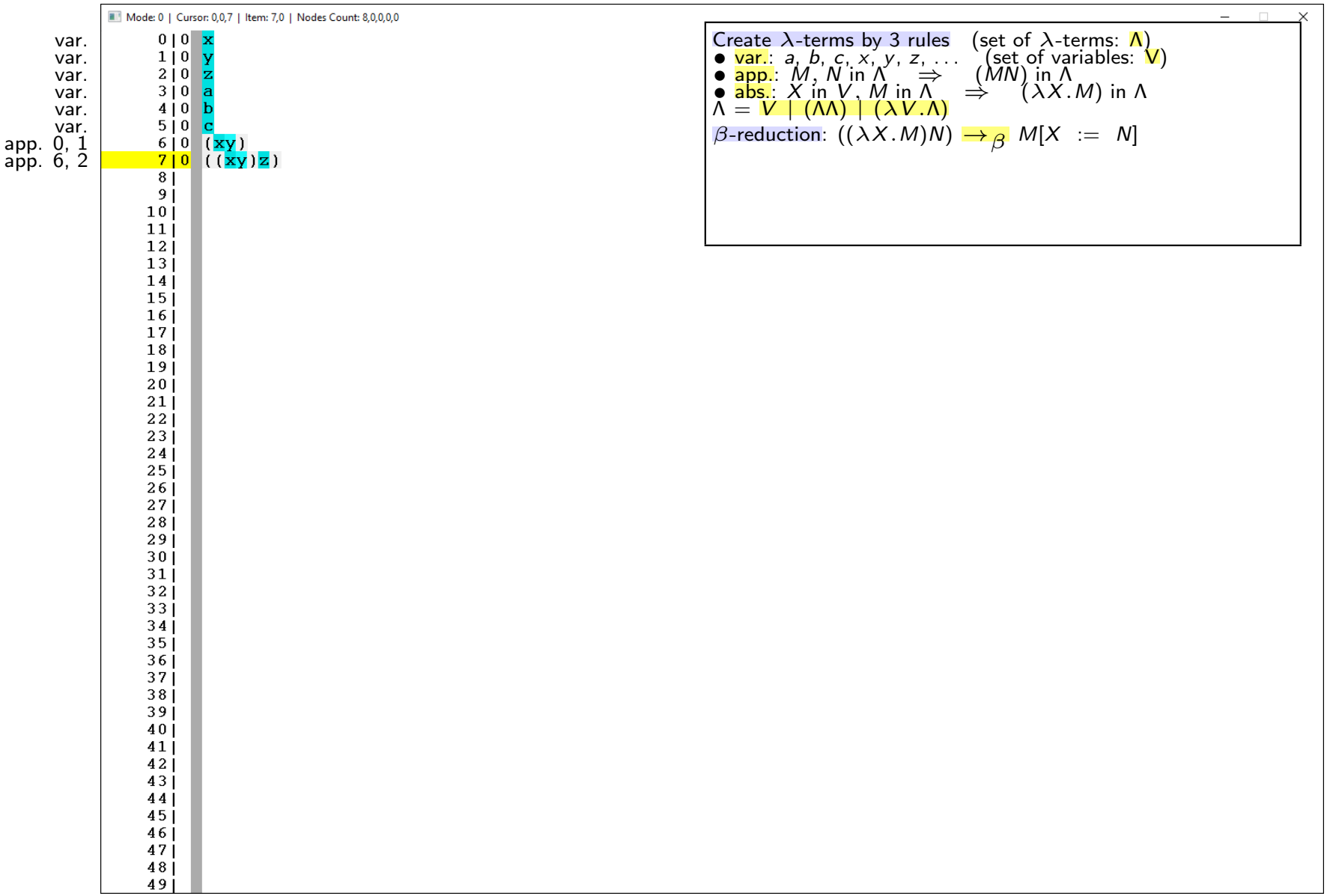

Mode: 0 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 7,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
app. 0, 1 3 | 0 (xy)
app. 3, 3 4 | 0 ((xy)(xy))
abs. 0, 4 5 | 0 (\x.((xy)(xy)))
app. 5, 2 6 | 0 ((\x.((xy)(xy)))z)
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)

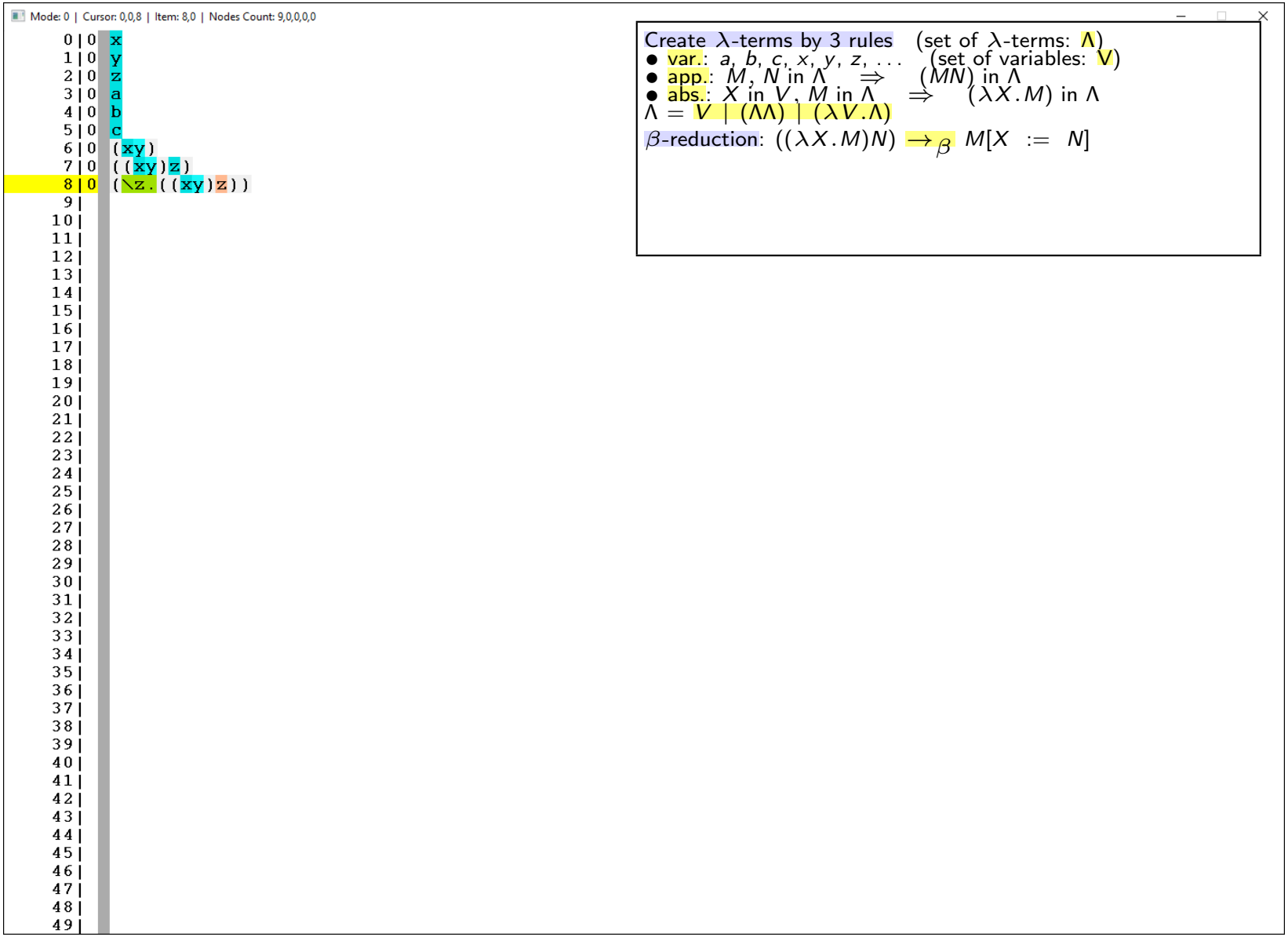

Mode: 0 | Cursor: 0,0,7 | Item: 7,0 | Nodes Count: 10,0,0,0,0
var.
var.
var.
app. 0, 1
app. 3, 3
abs. 0, 4
app. 5, 2
β-red 6
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 (xy)
4 | 0 ((xy)(xy))
5 | 0 (\x.((xy)(xy)))
6 | 0 ((\x.((xy)(xy)))z)
7 | 0 ((zy)(zy))
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

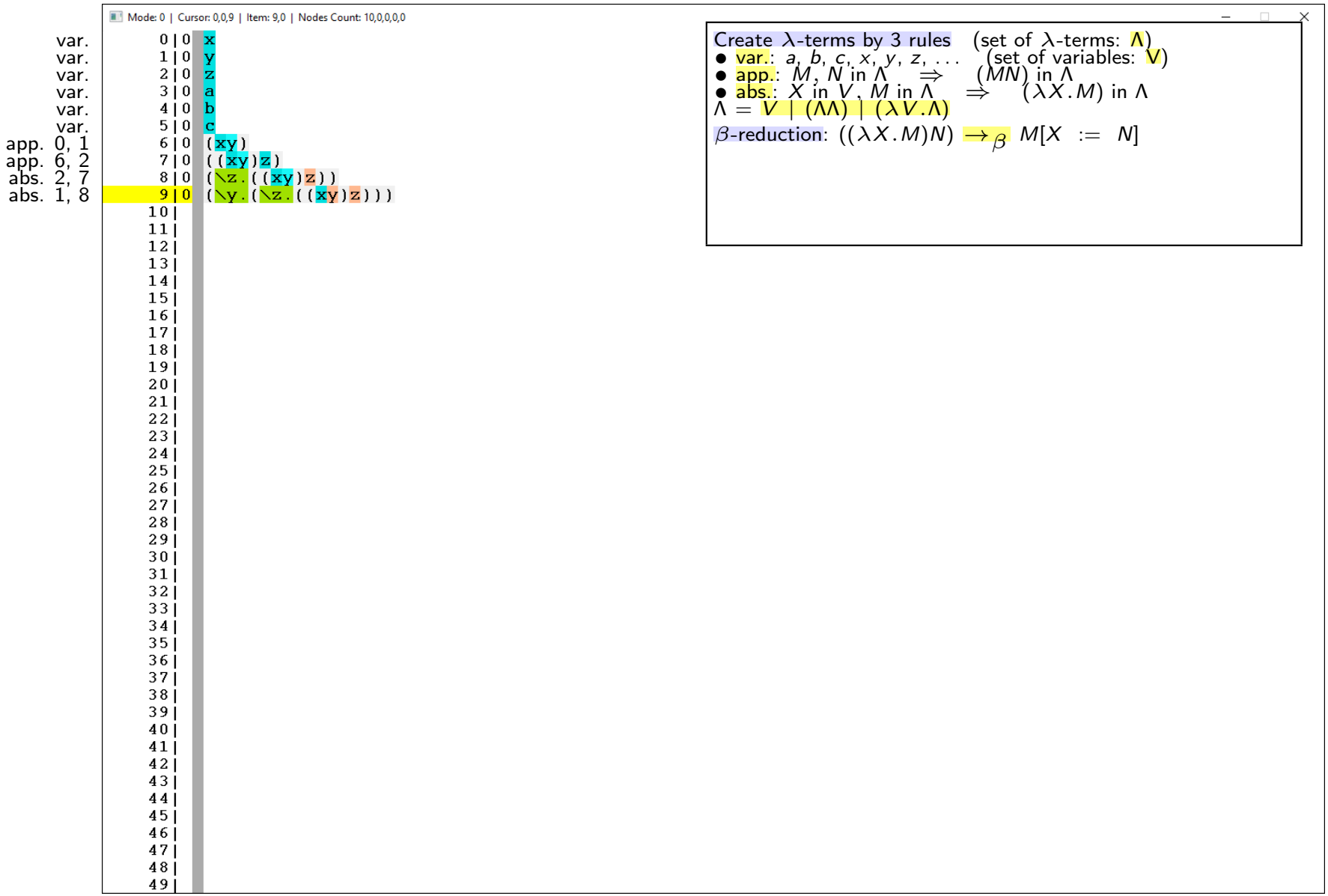

Mode: 0 | Cursor: 0,0,2 | Item: 2,0 | Nodes Count: 3,0,0,0,0
var.
var.
var.
0 | 0 x
1 | 0 y
2 | 0 z
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

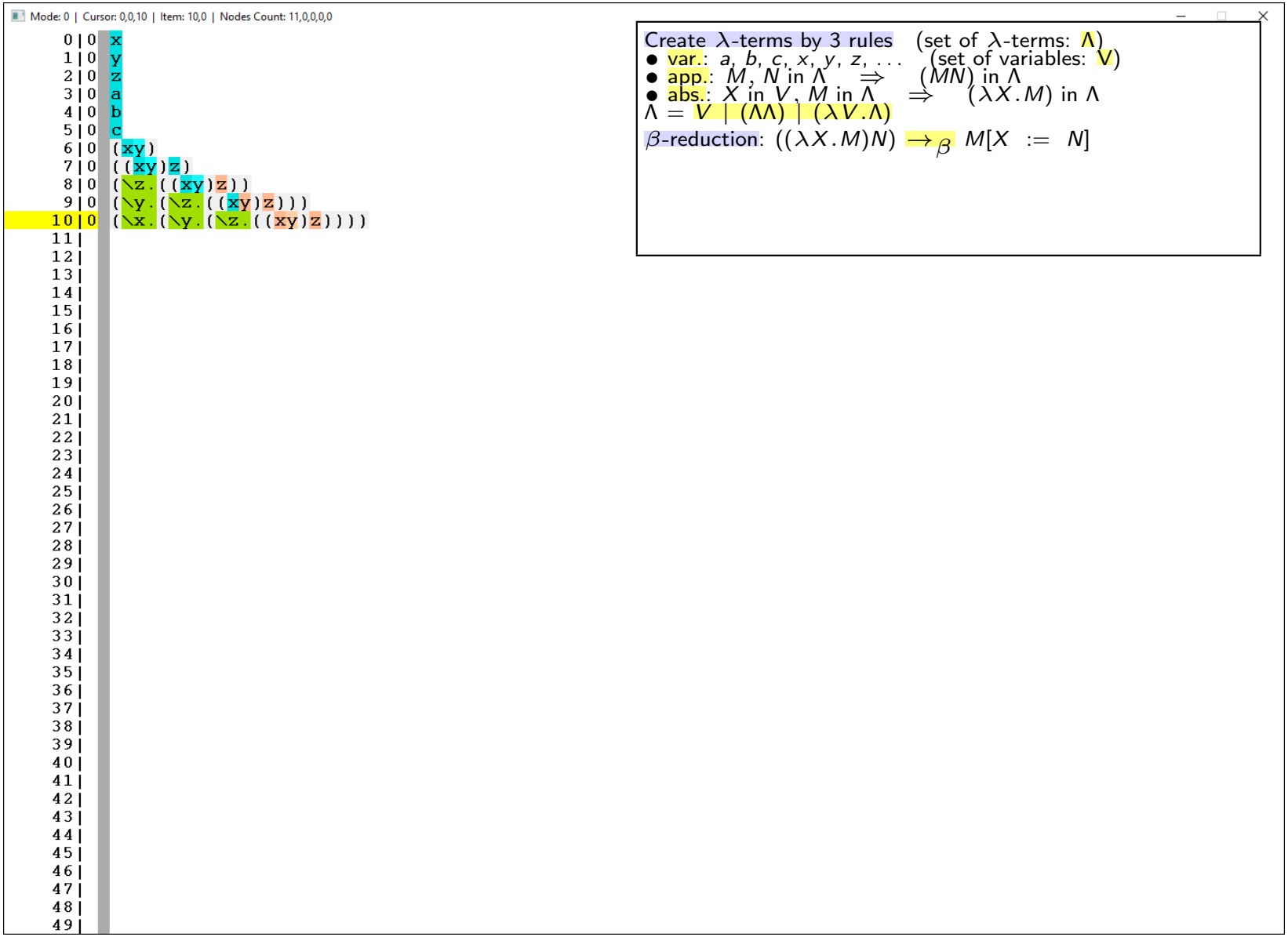

Mode: 0 | Cursor: 0,0,5 | Item: 5,0 | Nodes Count: 6,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

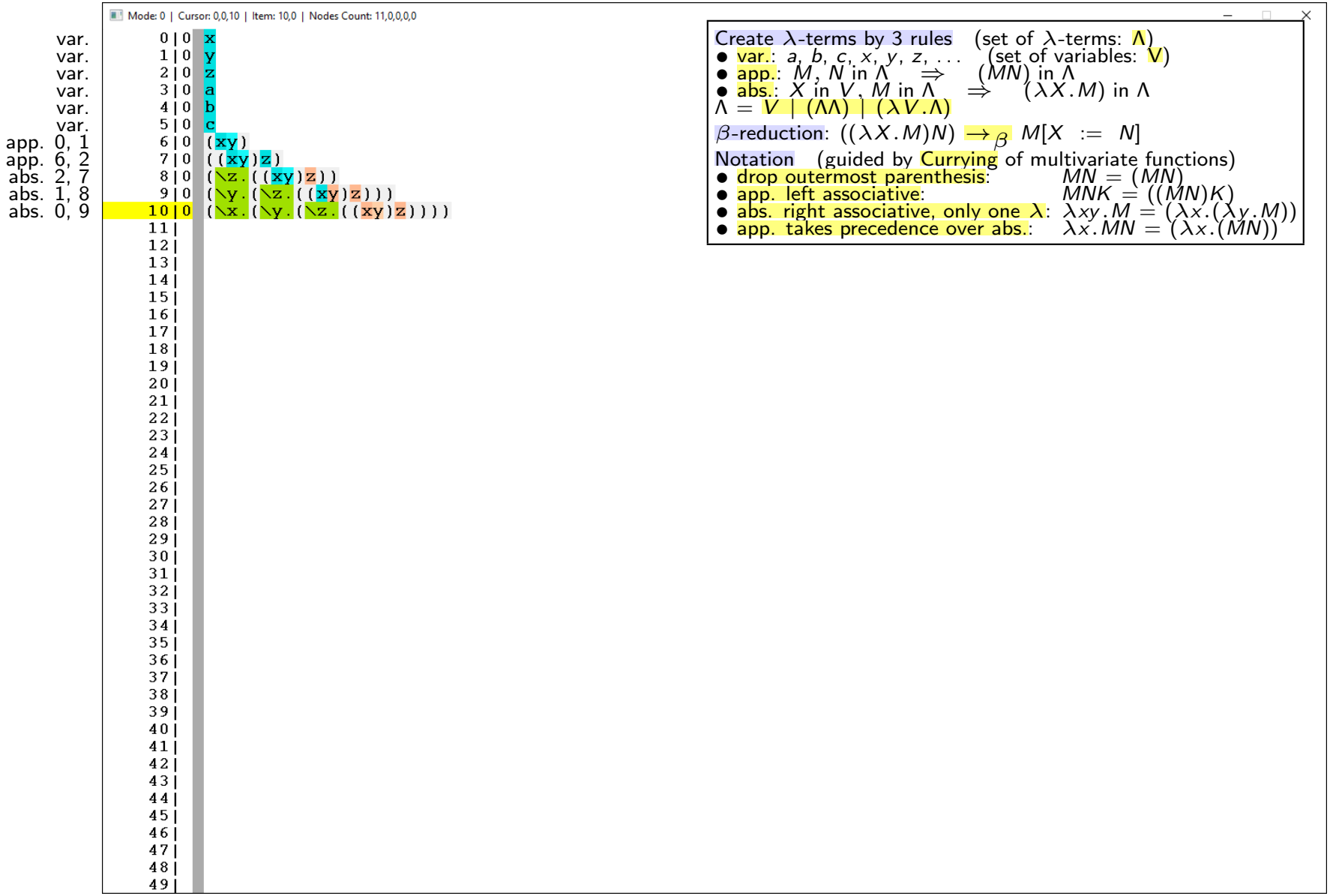

Mode: 0 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 7,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 (xy)
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

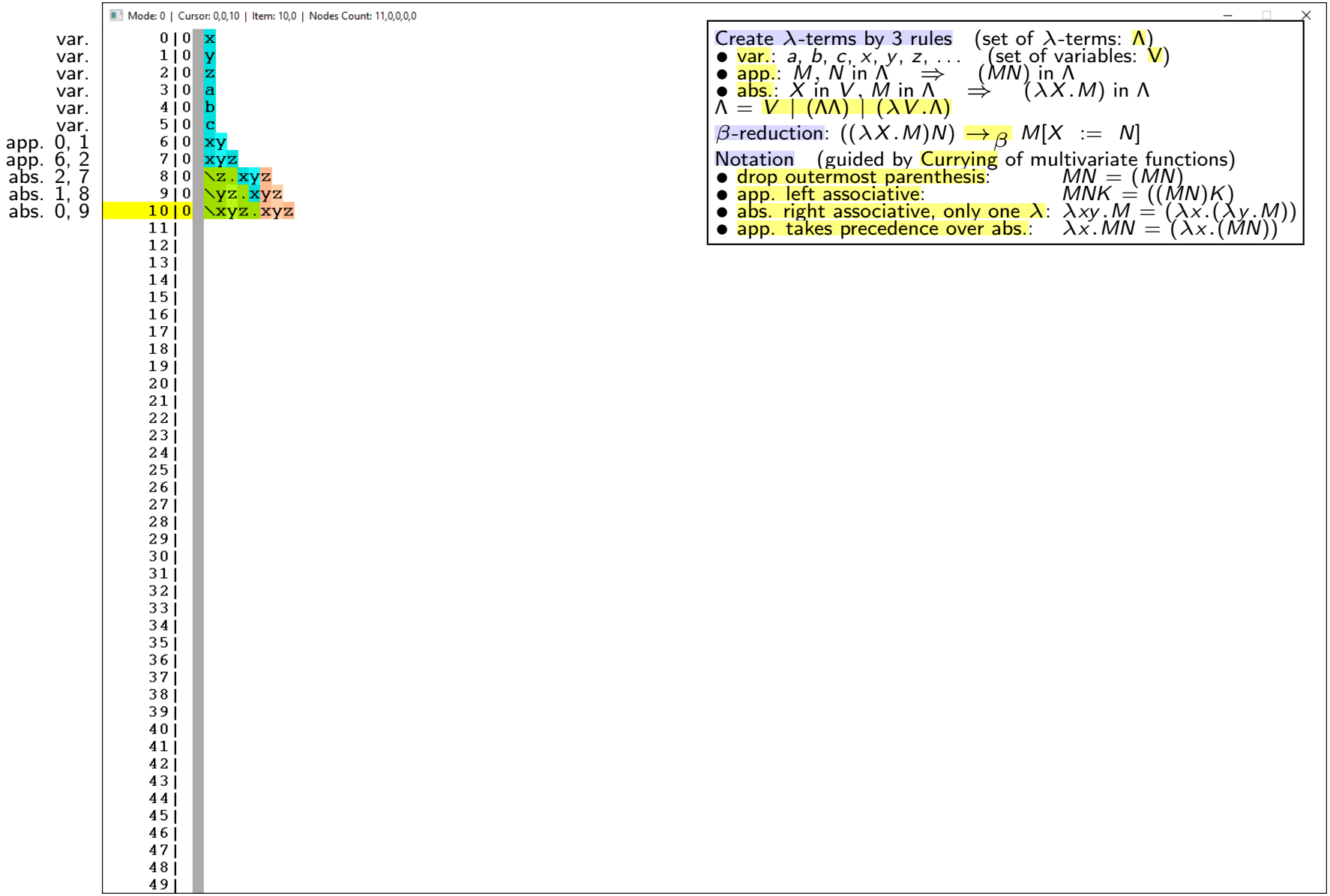

Mode: 0 | Cursor: 0,0,7 | Item: 7,0 | Nodes Count: 8,0,0,0,0
var.
var.
var.
var.
var.
var.
app. 0, 1
app. 6, 2
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 a
4 | 0 b
5 | 0 c
6 | 0 (xy)
7 | 0 ((xy)z)
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

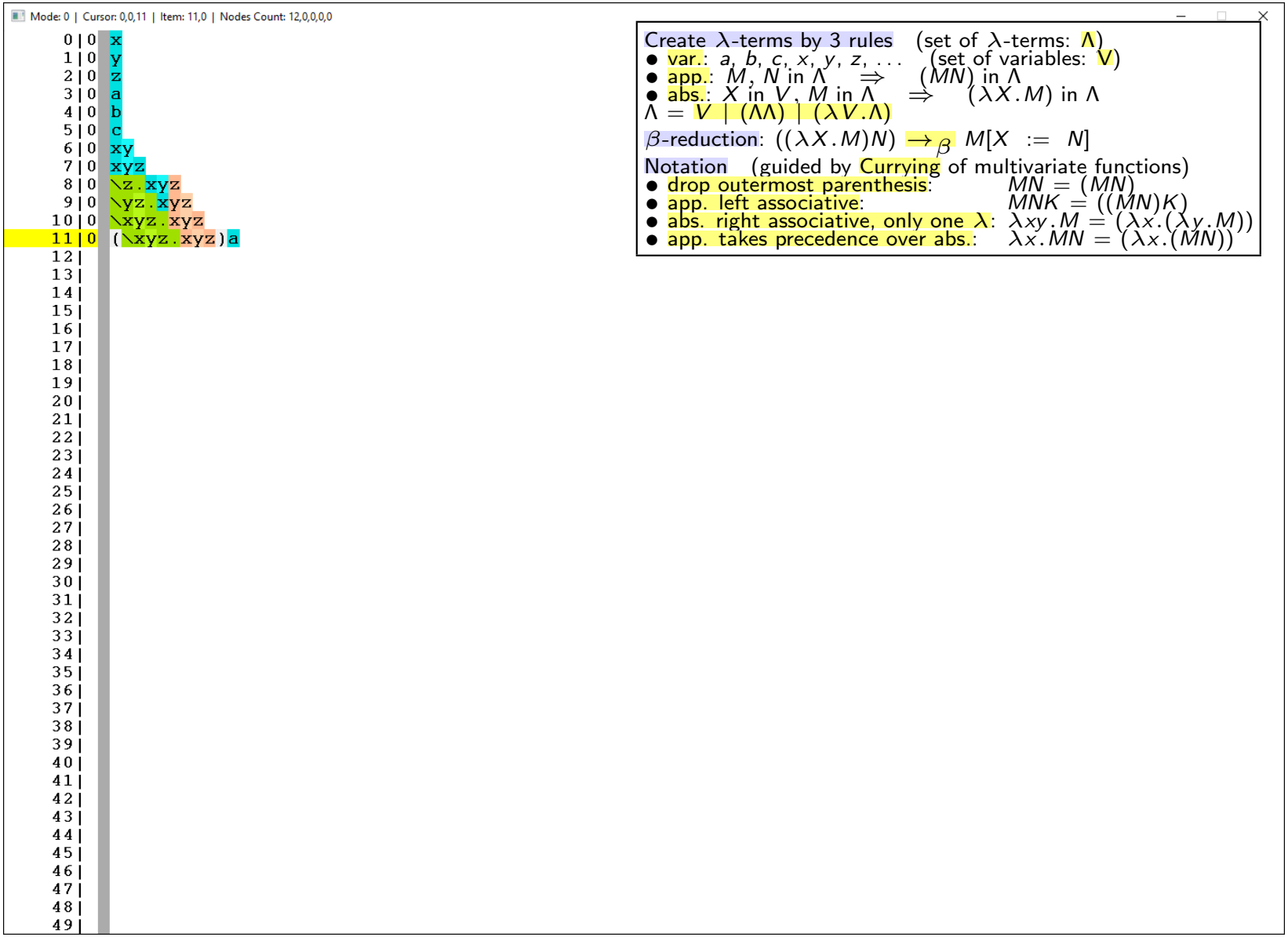

Mode: 0 | Cursor: 0,0,8 | Item: 8,0 | Nodes Count: 9,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 (xy)
app. 6, 2 7 | 0 ((xy)z)
abs. 2, 7 8 | 0 (\z.((xy)z))
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]

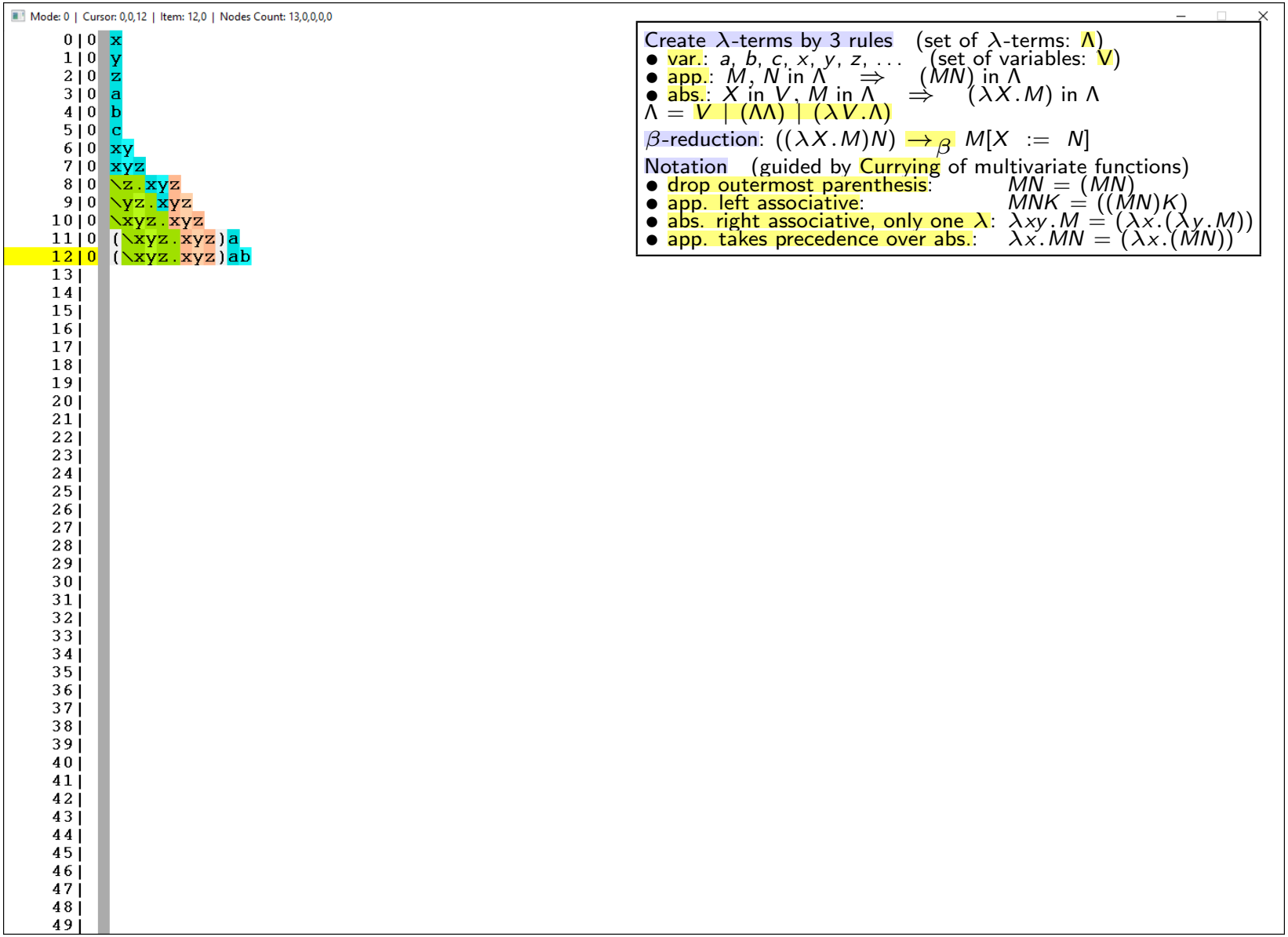

Mode: 0 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 10,0,0,0,0
var.
var.
var.
var.
var.
var.
app. 0, 1
app. 6, 2
abs. 2, 7
abs. 1, 8
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 a
4 | 0 b
5 | 0 c
6 | 0 (xy)
7 | 0 ((xy)z)
8 | 0 (\z.((xy)z))
9 | 0 (\y.(\z.((xy)z)))
Create λ-terms by 3 rules (set of λ-terms: Λ)
● var.: a, b, c, x, y, z, ... (set of variables: V)
● app.: M, N in Λ ⇒ (MN) in Λ
● abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]


Mode: 0 | Cursor: 0,0,10 | Item: 10,0 | Nodes Count: 11,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 (xy)
app. 6, 2 7 | 0 ((xy)z)
abs. 2, 7 8 | 0 (\z.((xy)z))
abs. 1, 8 9 | 0 (\y.(\z.((xy)z)))
abs. 0, 9 10 | 0 (\x.(\y.(\z.((xy)z))))
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]


Mode: 0 | Cursor: 0,0,10 | Item: 10,0 | Nodes Count: 11,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 (xy)
app. 6, 2 7 | 0 ((xy)z)
abs. 2, 7 8 | 0 (\z.((xy)z))
abs. 1, 8 9 | 0 (\y.(\z.((xy)z)))
abs. 0, 9 10 | 0 (\x.(\y.(\z.((xy)z))))
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
• abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,10 | Item: 10,0 | Nodes Count: 11,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))


Mode: 0 | Cursor: 0,0,11 | Item: 11,0 | Nodes Count: 12,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
app. 10, 3 11 | 0 (\xyz.xyz)a
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))



Create λ-terms by 3 rules (set of λ-terms: Λ)

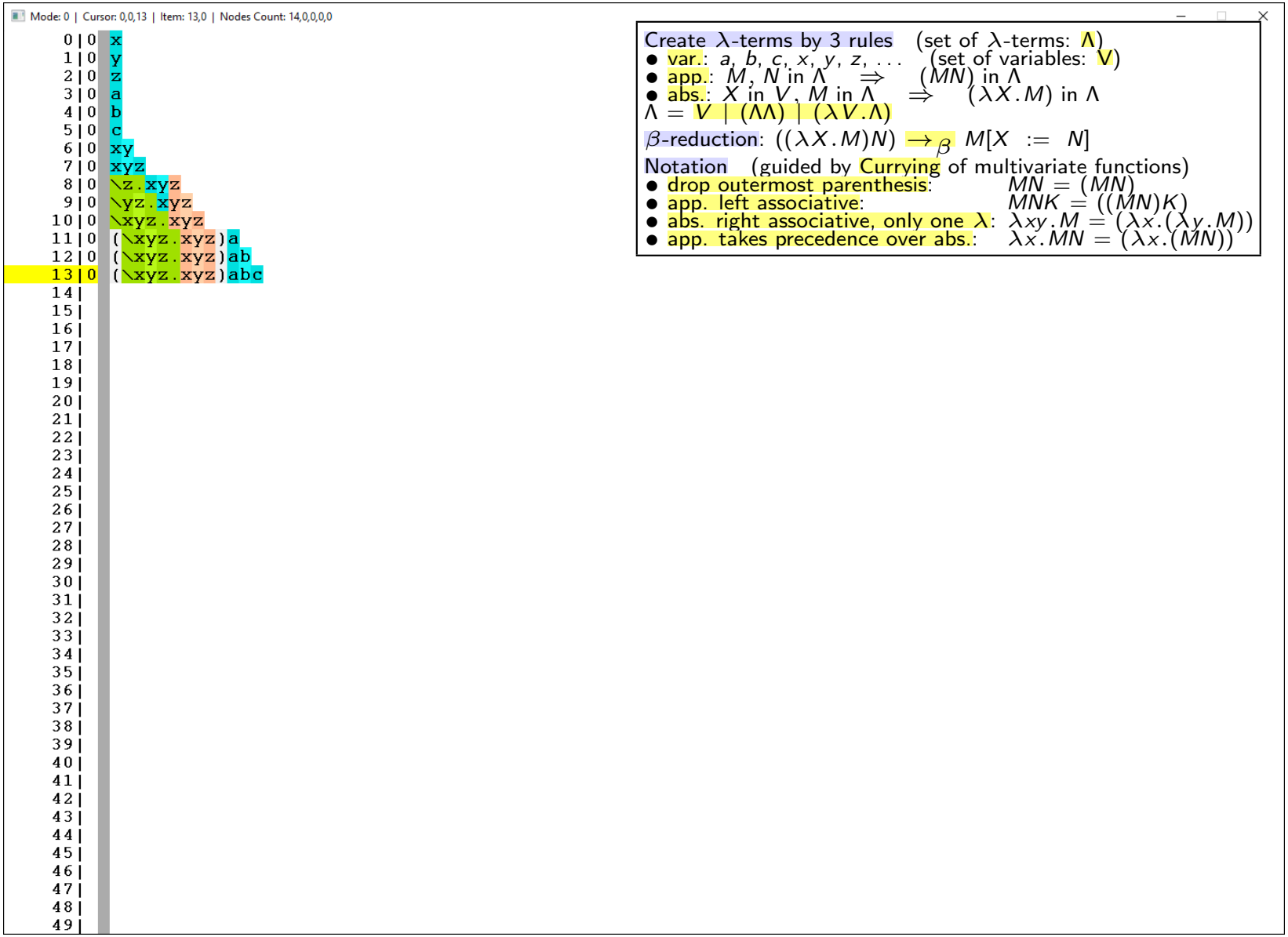
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

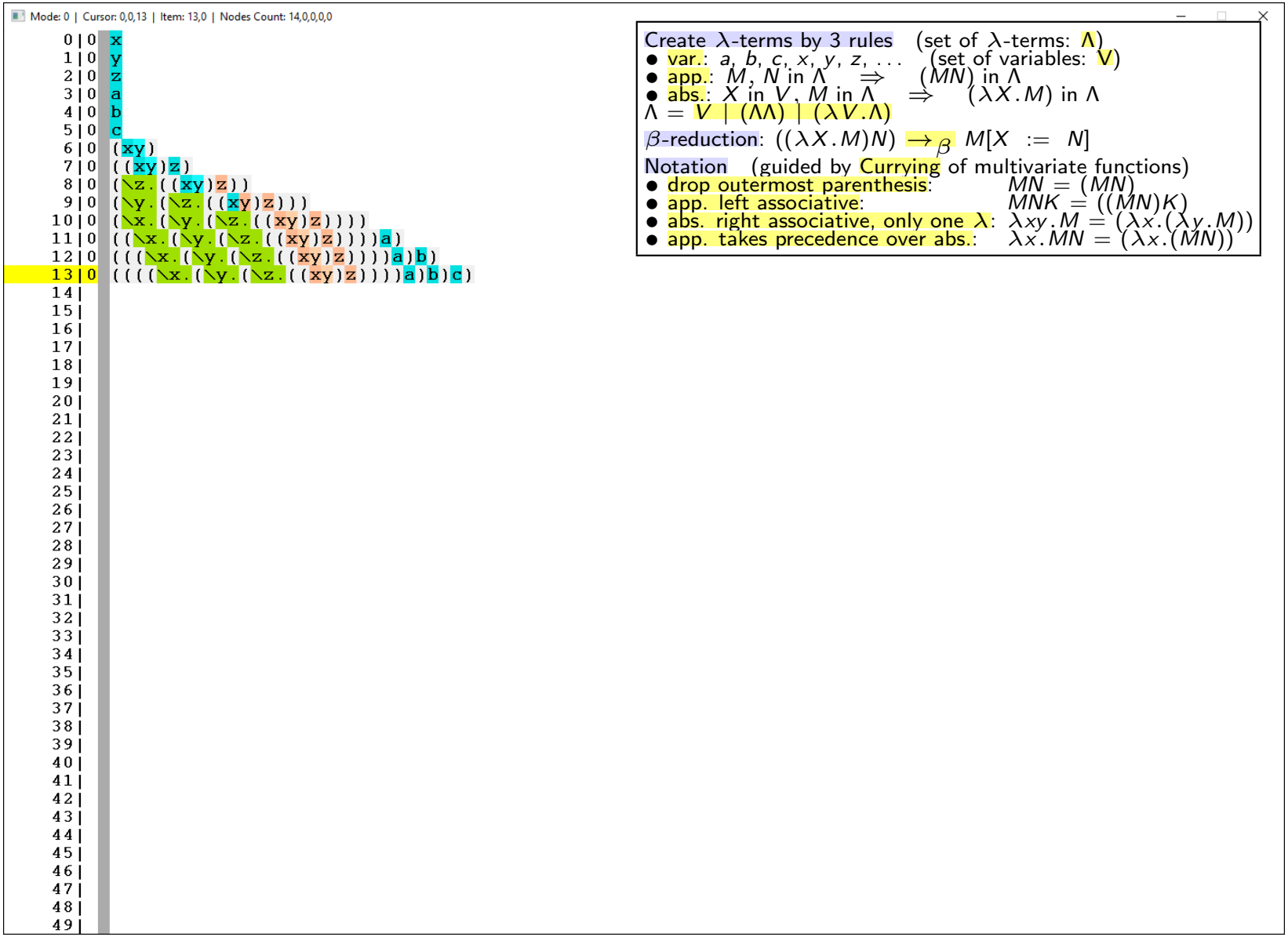
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,13 | Item: 13,0 | Nodes Count: 14,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
app. 10, 3 11 | 0 (\xyz.xyz)a
app. 11, 4 12 | 0 (\xyz.xyz)ab
app. 12, 5 13 | 0 (\xyz.xyz)abc
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
• abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

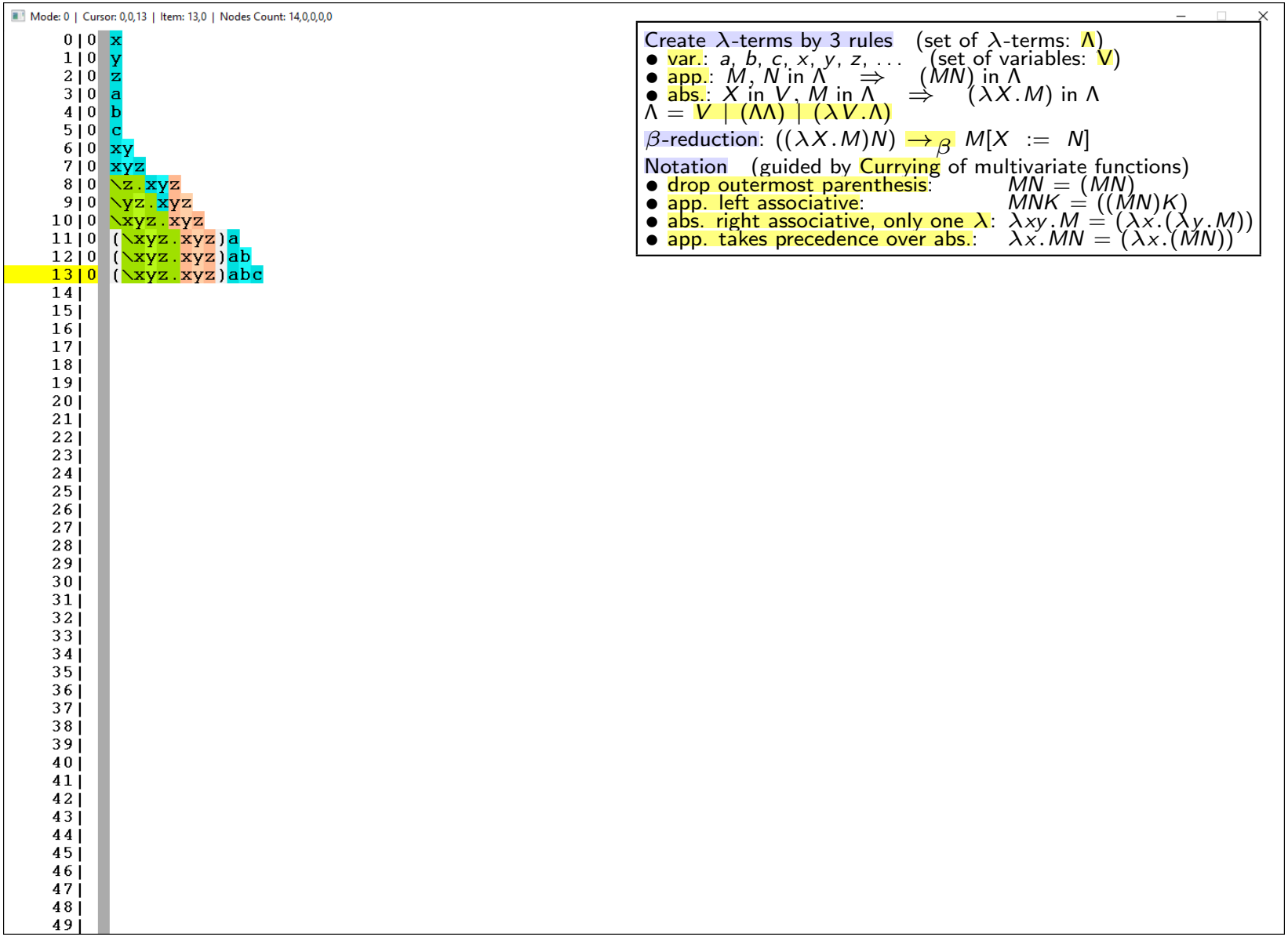
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

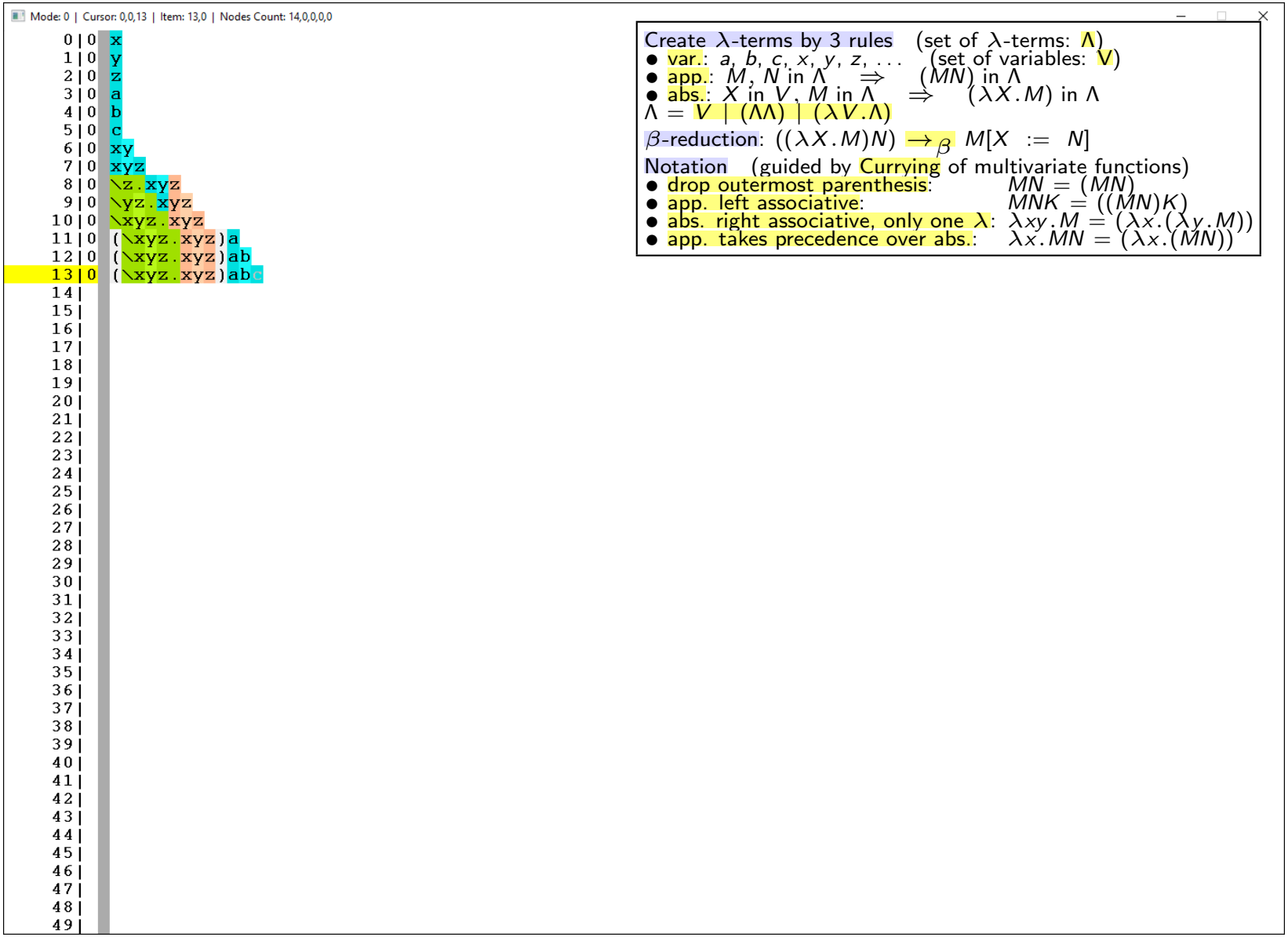
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

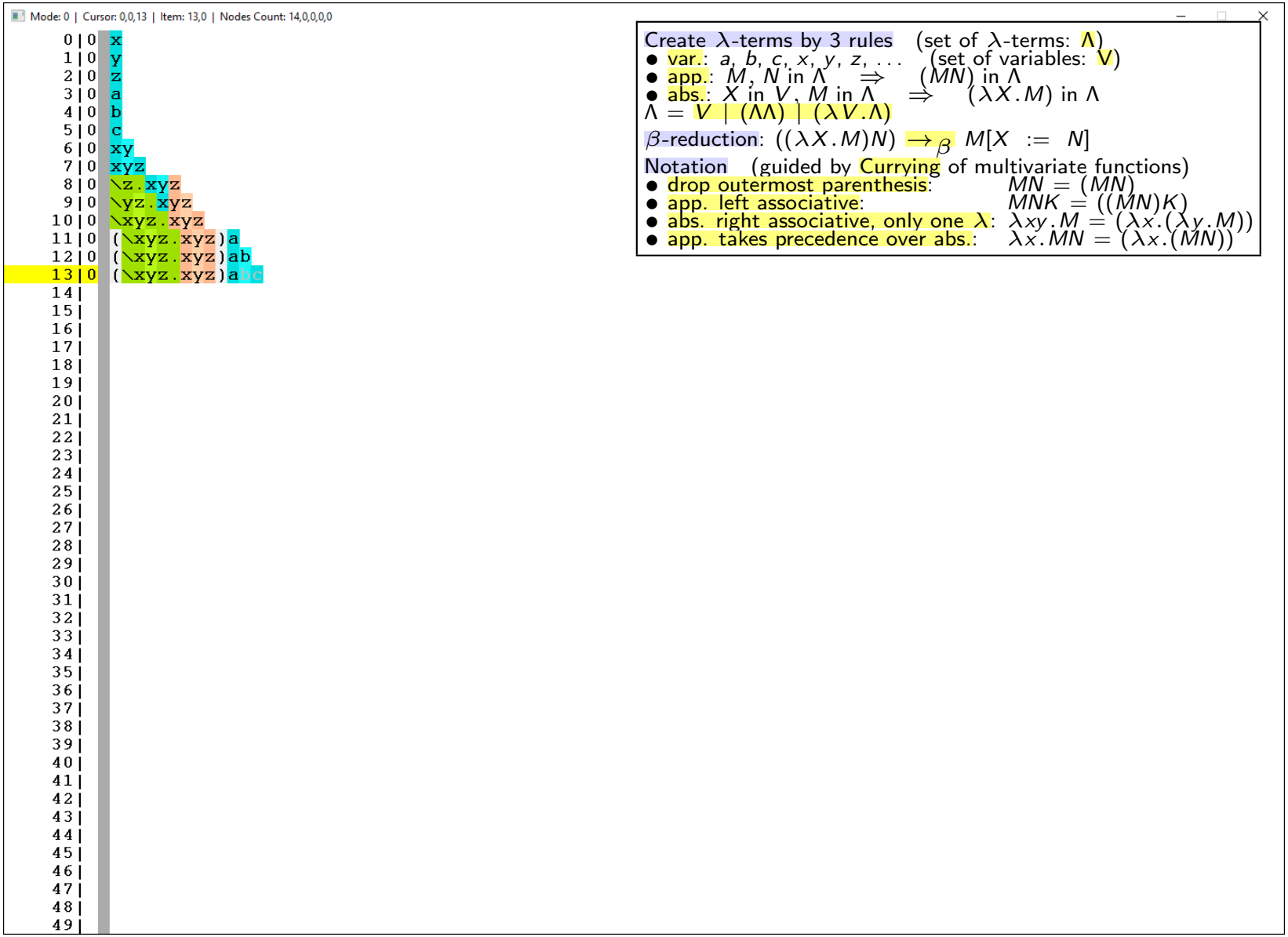
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

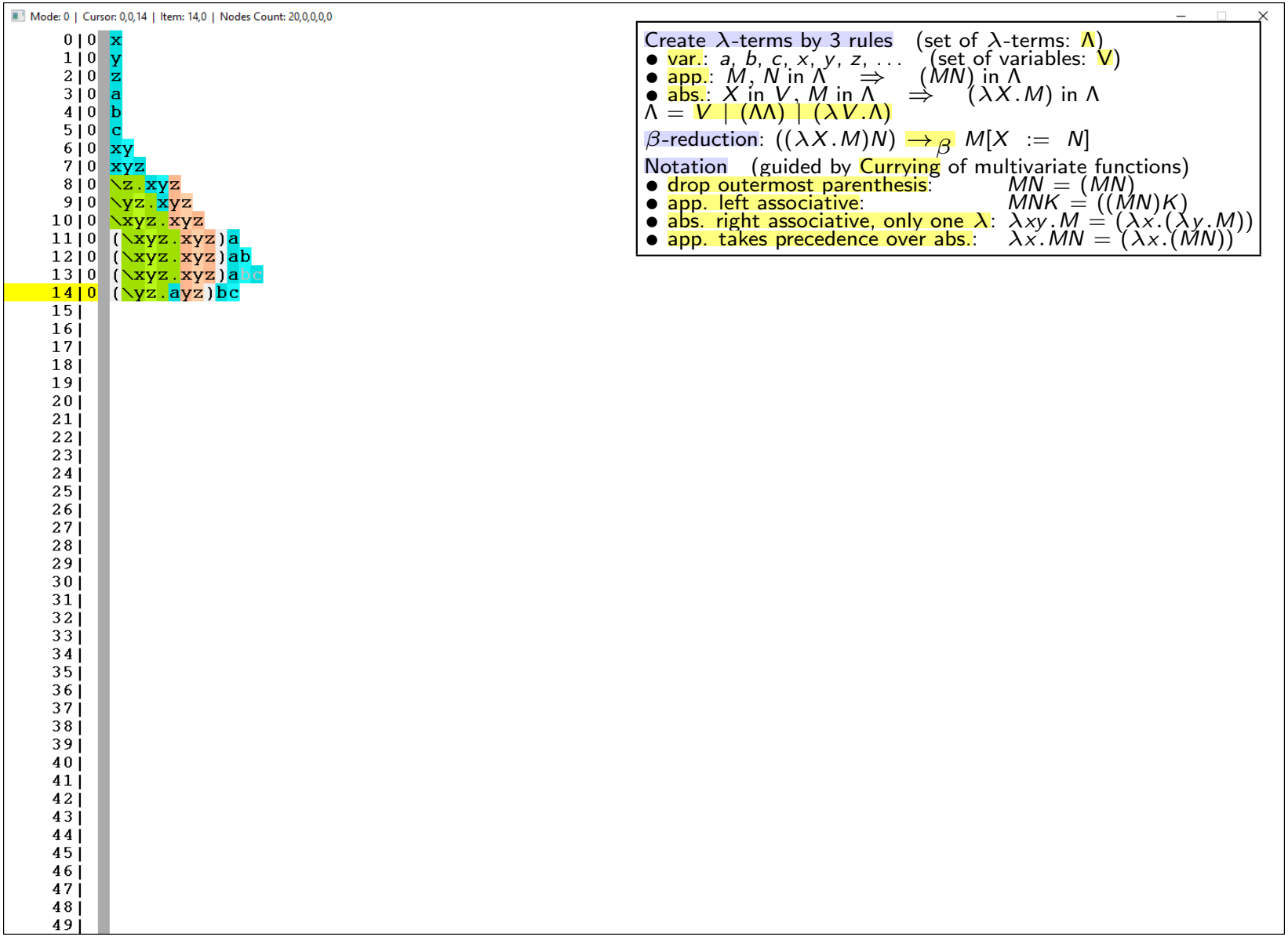
Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,13 | Item: 13,0 | Nodes Count: 14,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
app. 10, 3 11 | 0 (\xyz.xyz)a
app. 11, 4 12 | 0 (\xyz.xyz)ab
app. 12, 5 13 | 0 (\xyz.xyz)abc
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
• abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,13 | Item: 13,0 | Nodes Count: 14,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
app. 10, 3 11 | 0 (\xyz.xyz)a
app. 11, 4 12 | 0 (\xyz.xyz)ab
app. 12, 5 13 | 0 (\xyz.xyz)abc
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
• abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,13 | Item: 13,0 | Nodes Count: 14,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 0, 1 6 | 0 xy
app. 6, 2 7 | 0 xyz
abs. 2, 7 8 | 0 \z.xyz
abs. 1, 8 9 | 0 \yz.xyz
abs. 0, 9 10 | 0 \xyz.xyz
app. 10, 3 11 | 0 (\xyz.xyz)a
app. 11, 4 12 | 0 (\xyz.xyz)ab
app. 12, 5 13 | 0 (\xyz.xyz)abc
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
• abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

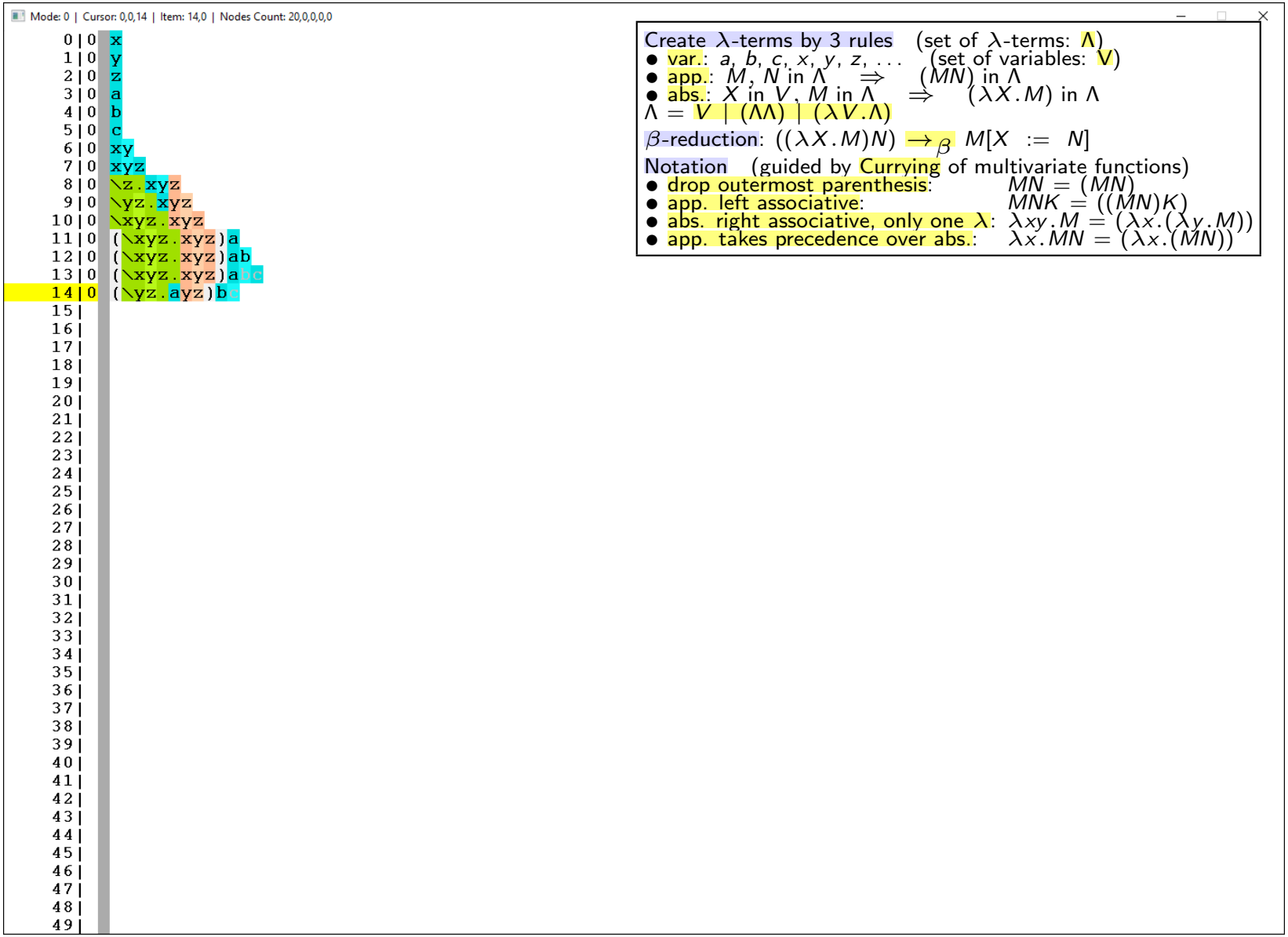
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

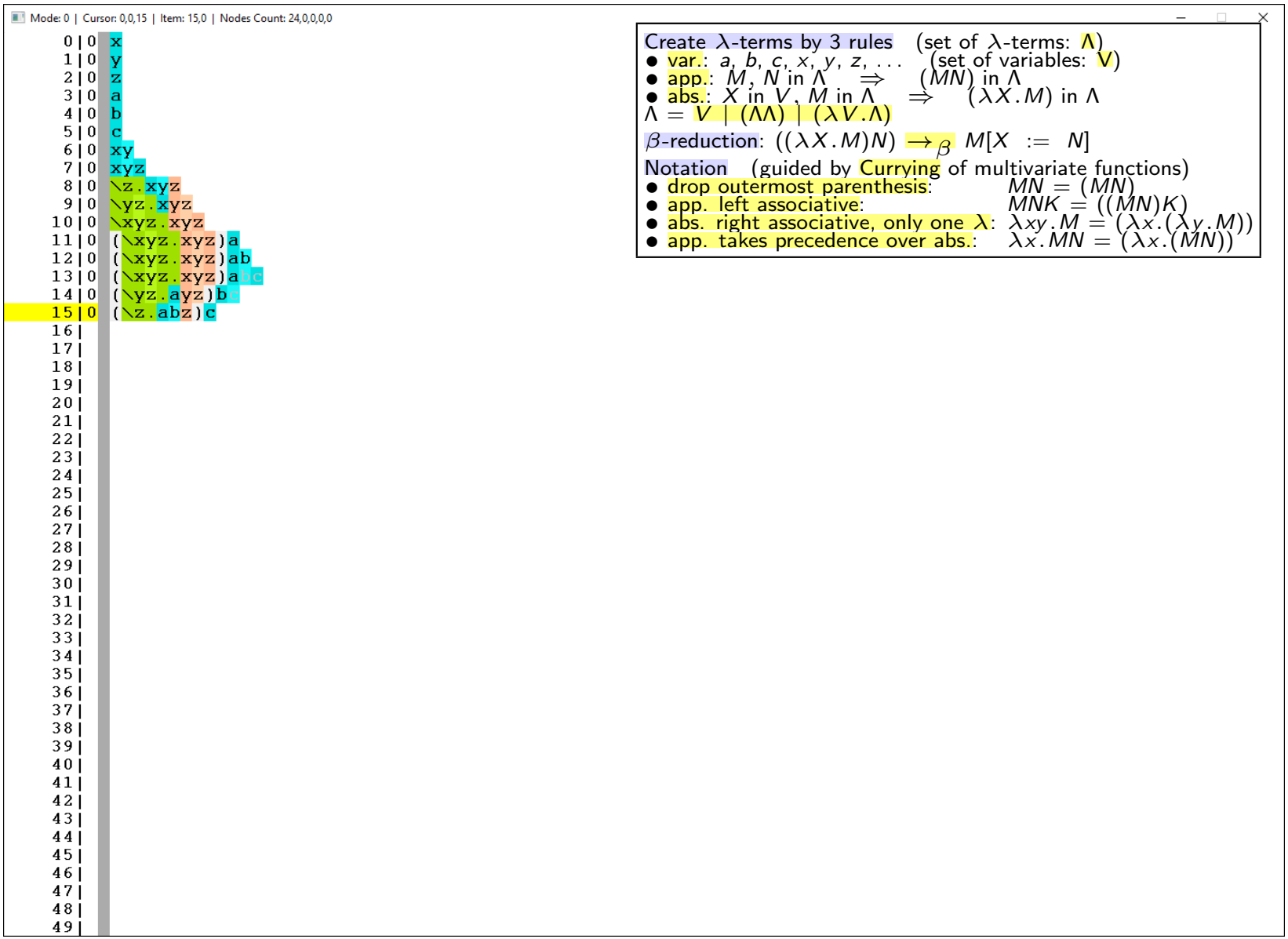
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,14 | Item: 14,0 | Nodes Count: 20,0,0,0,0
var.
var.
var.
var.
var.
var.
app. 0, 1
app. 6, 2
abs. 2, 7
abs. 1, 8
abs. 0, 9
app. 10, 3
app. 11, 4
app. 12, 5
β-red 13
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 a
4 | 0 b
5 | 0 c
6 | 0 xy
7 | 0 xyz
8 | 0 \z.xyz
9 | 0 \yz.xyz
10 | 0 \xyz.xyz
11 | 0 (\xyz.xyz)a
12 | 0 (\xyz.xyz)ab
13 | 0 (\xyz.xyz)abc
14 | 0 (\yz.ayz)bc
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))

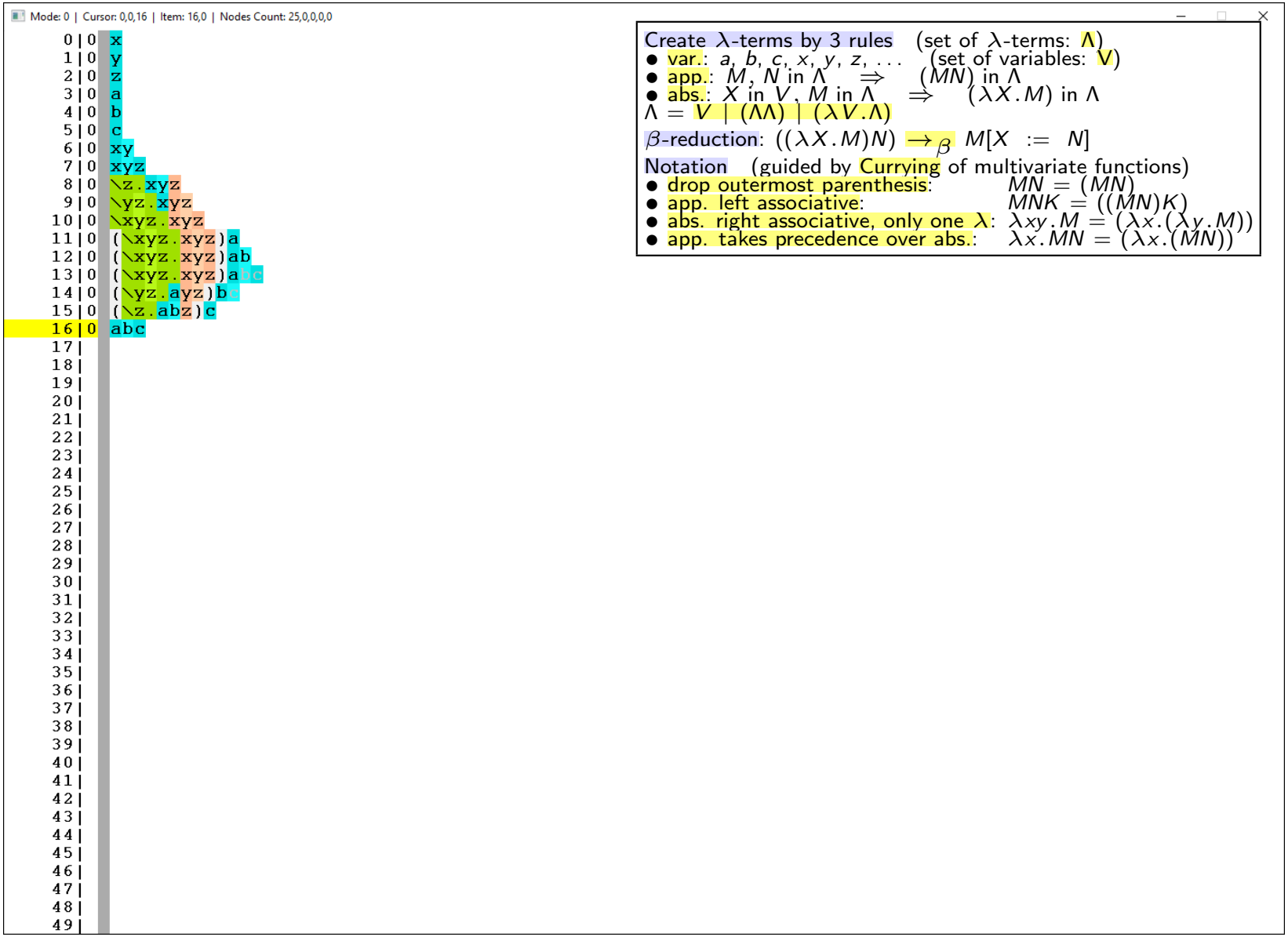


Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

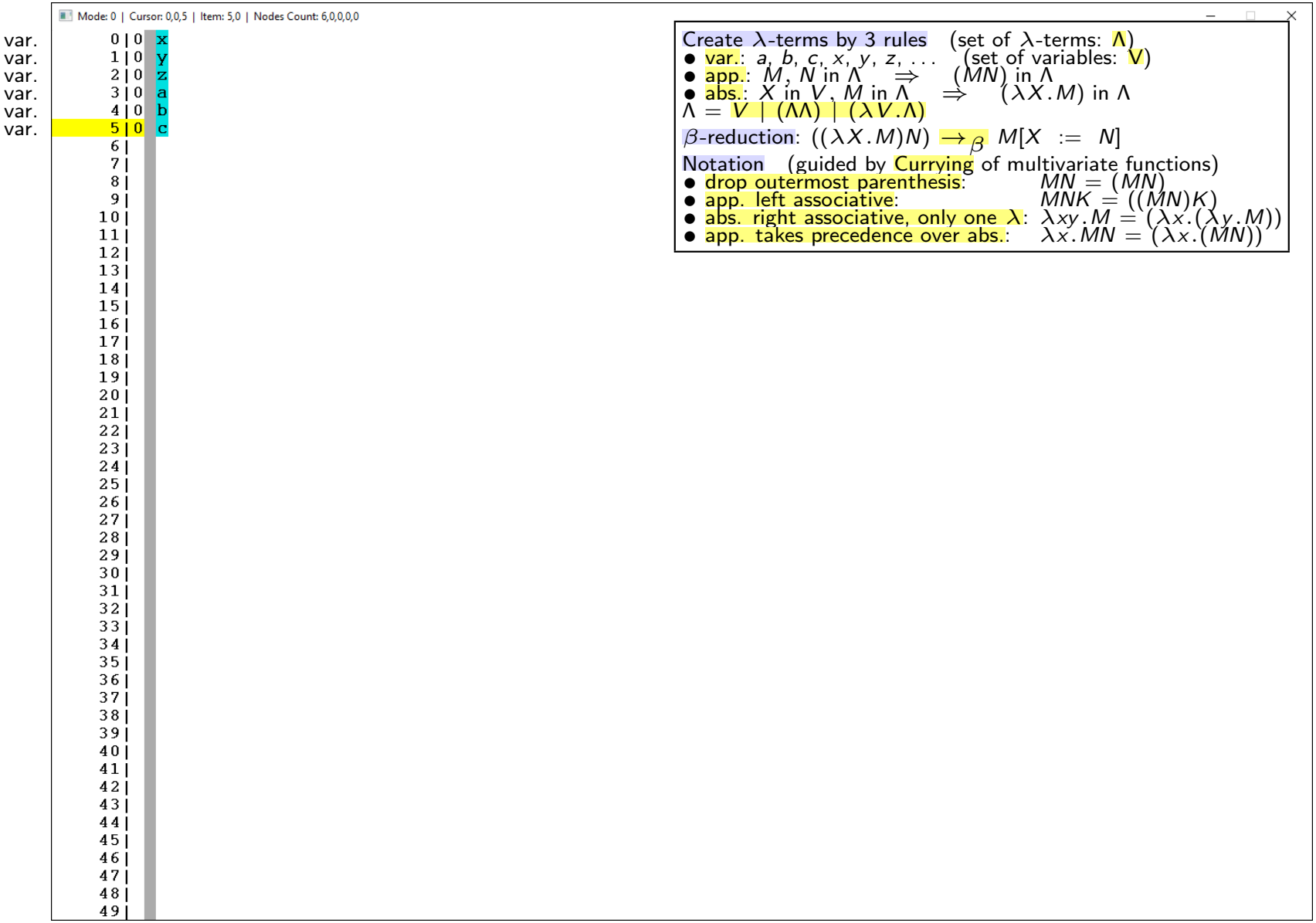


Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

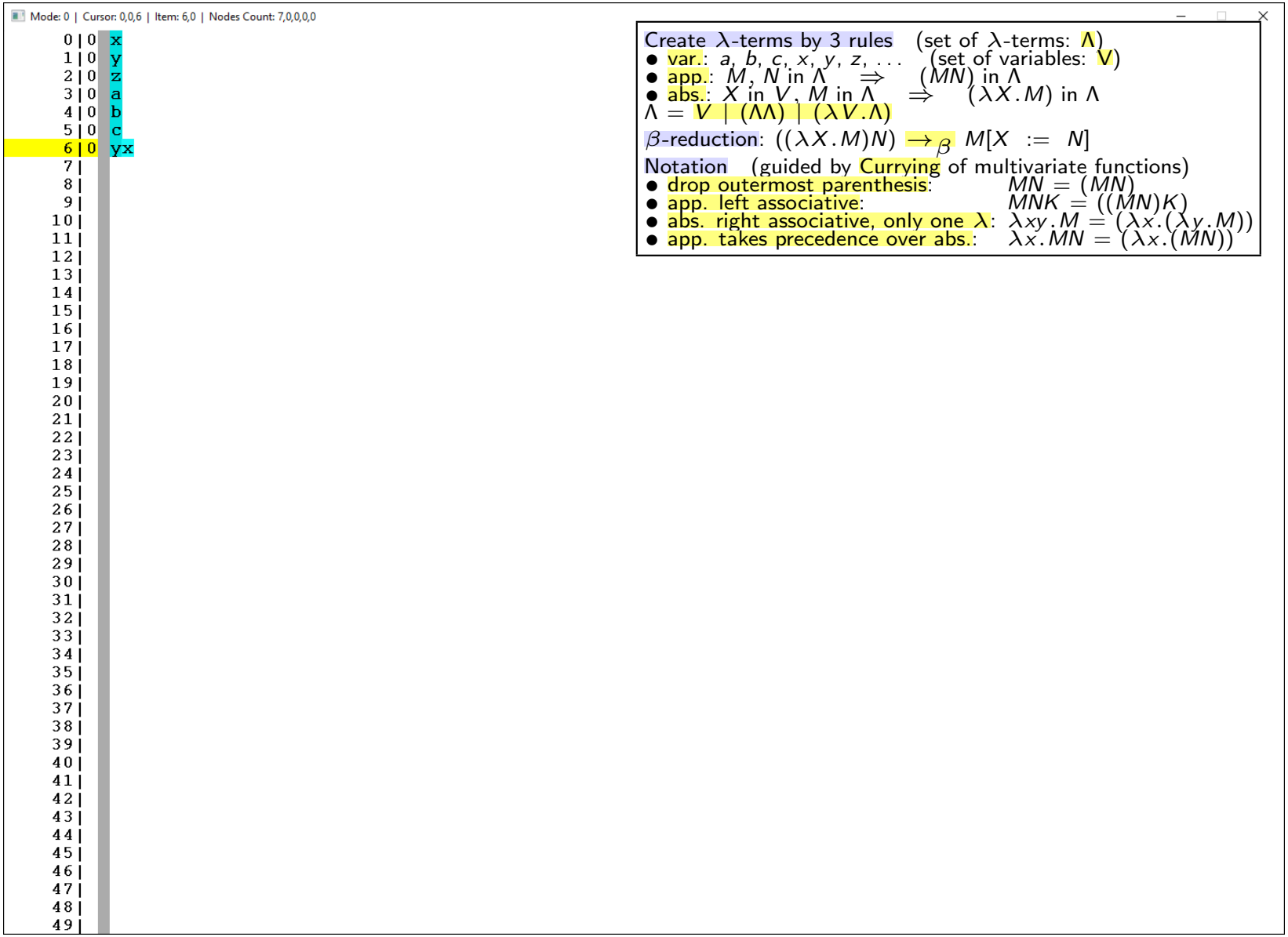
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

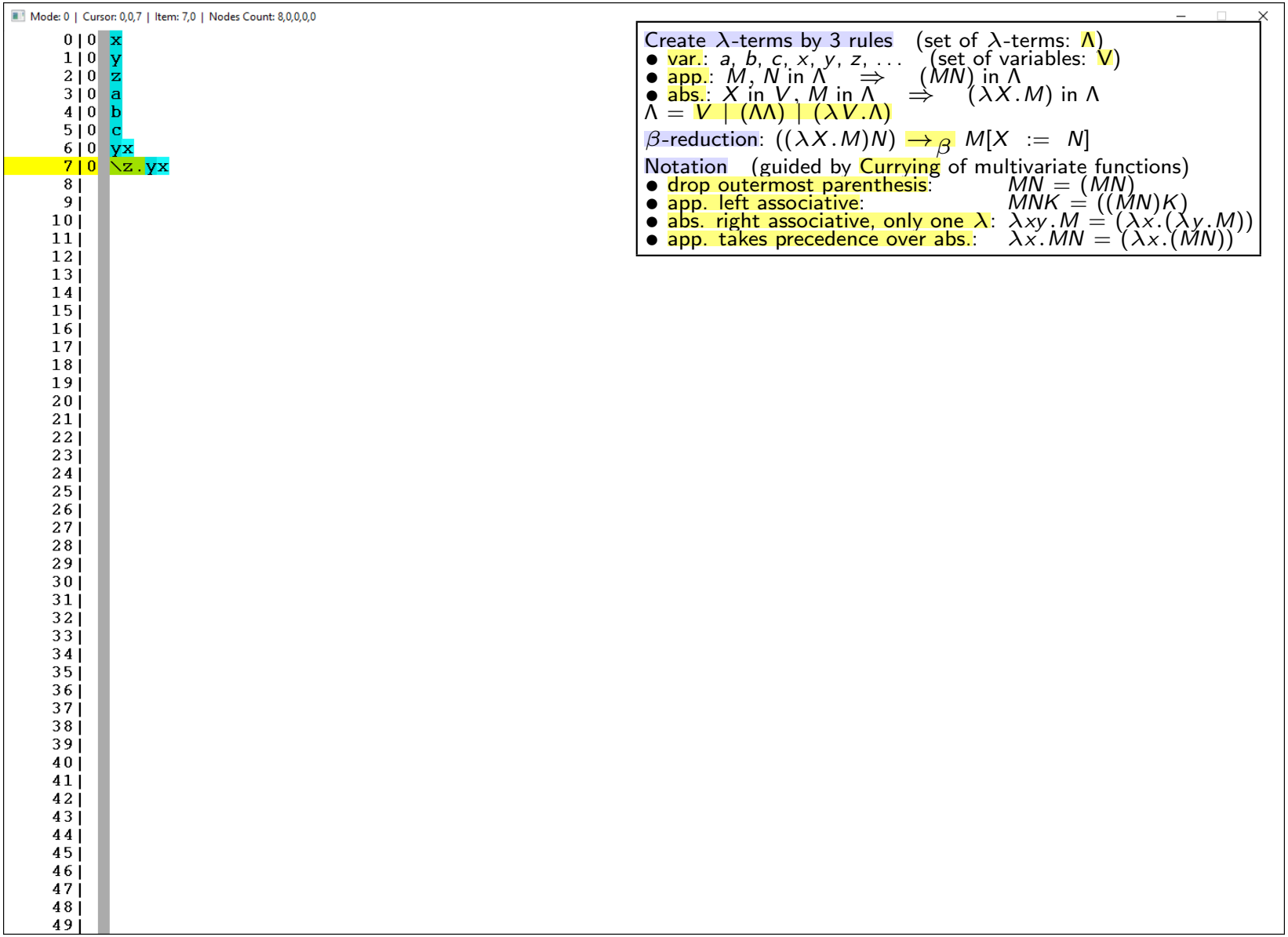
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

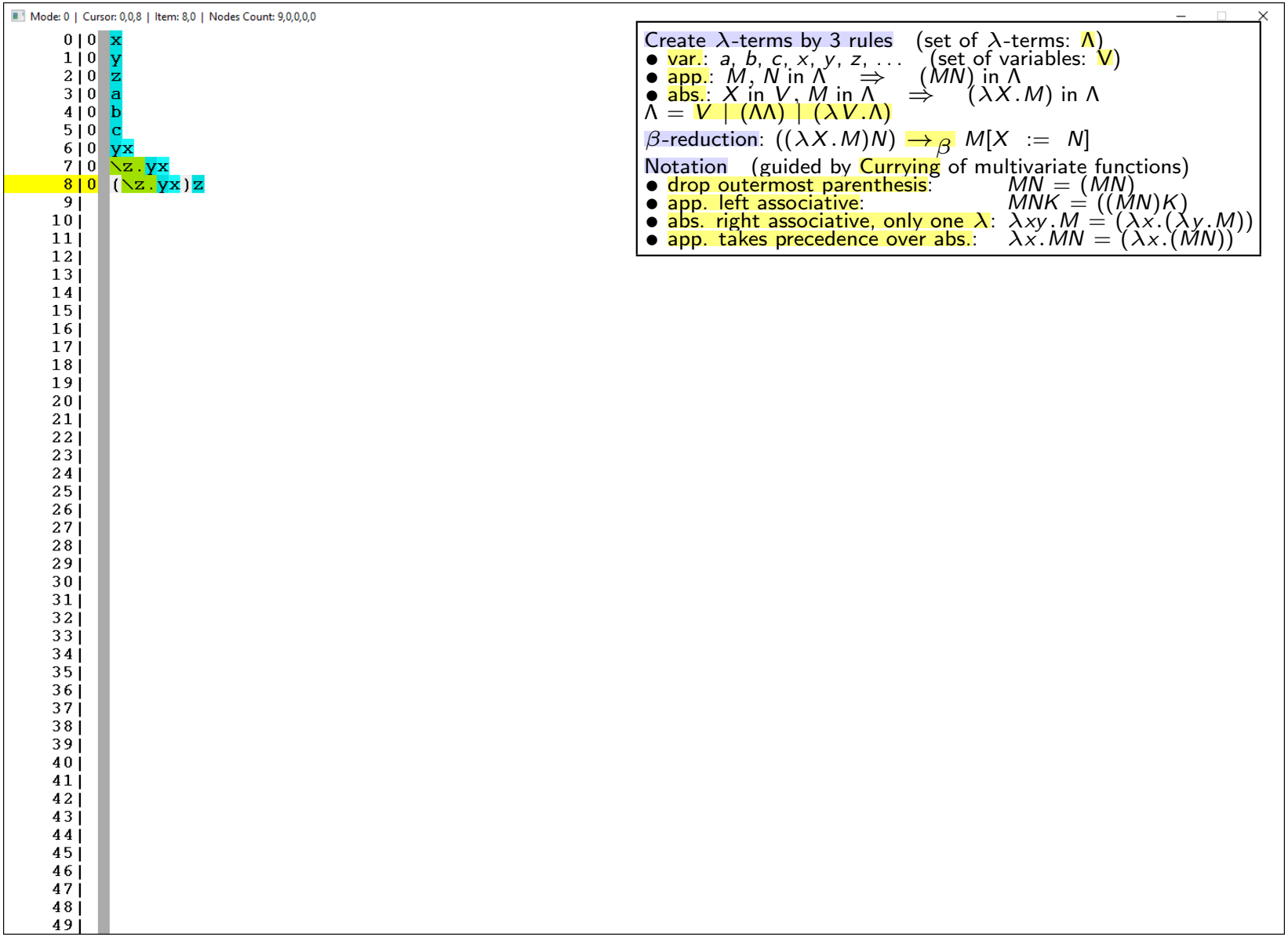
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

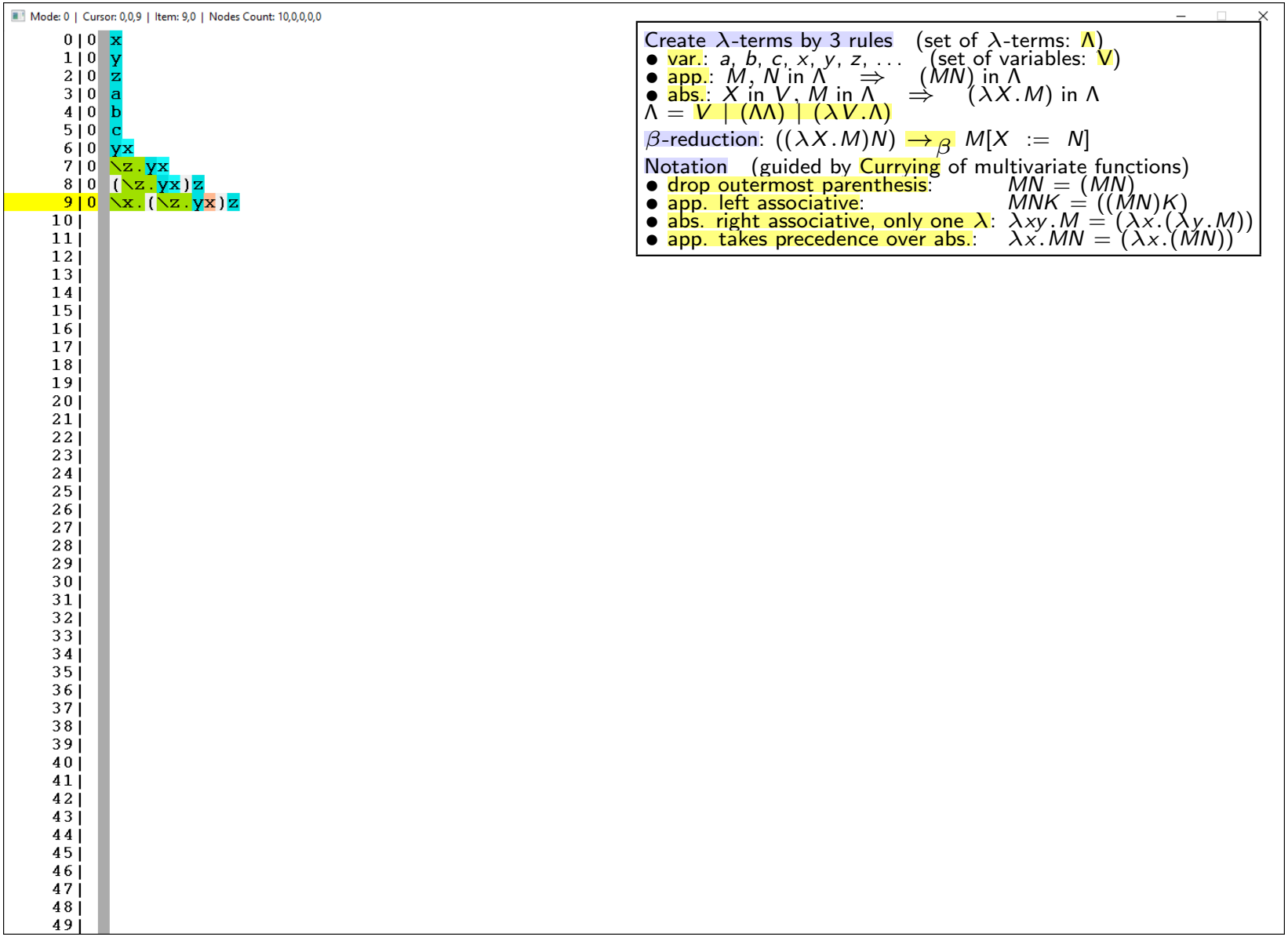
Notation (guided by Currying of multivariate functions)

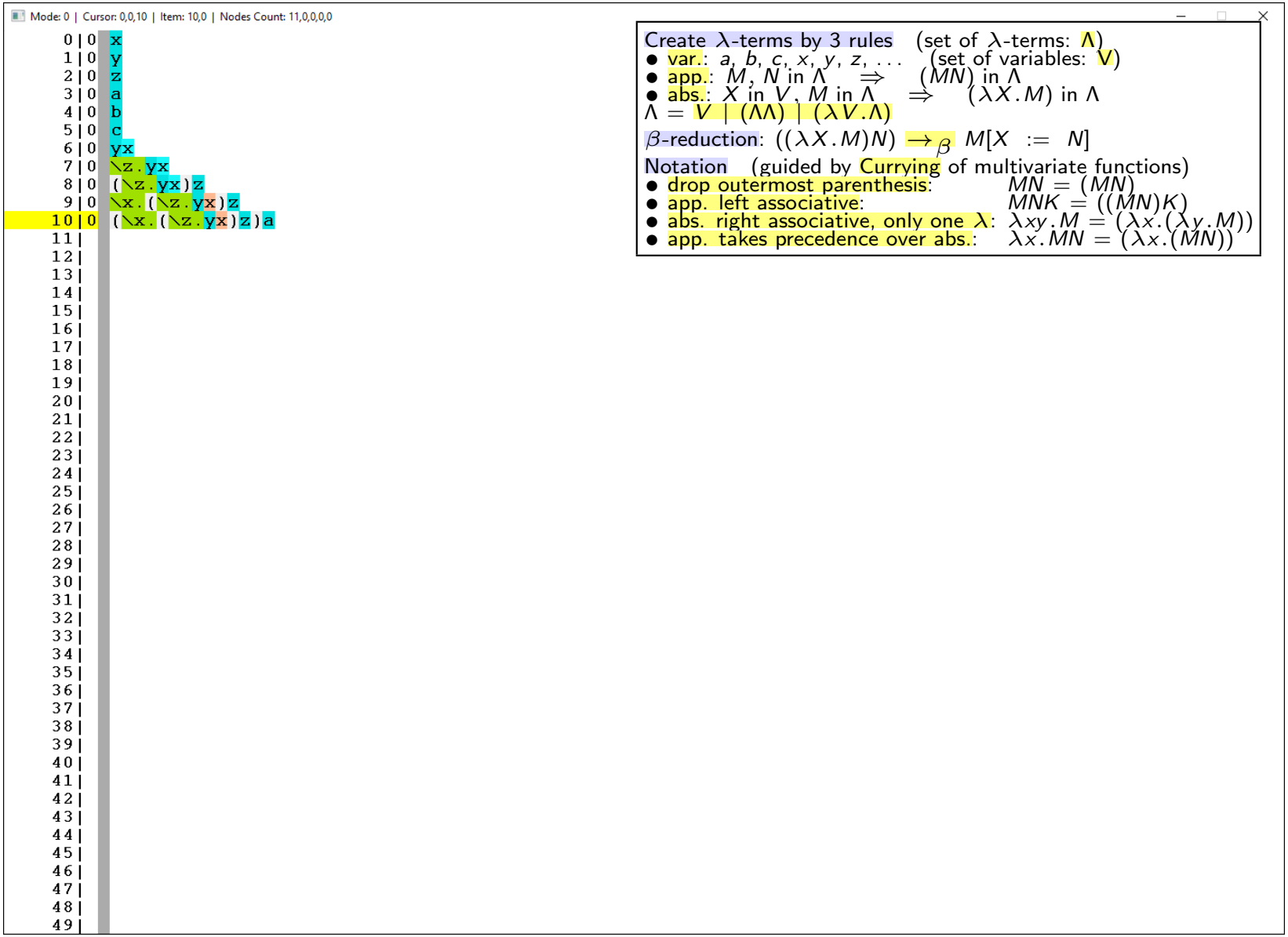
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

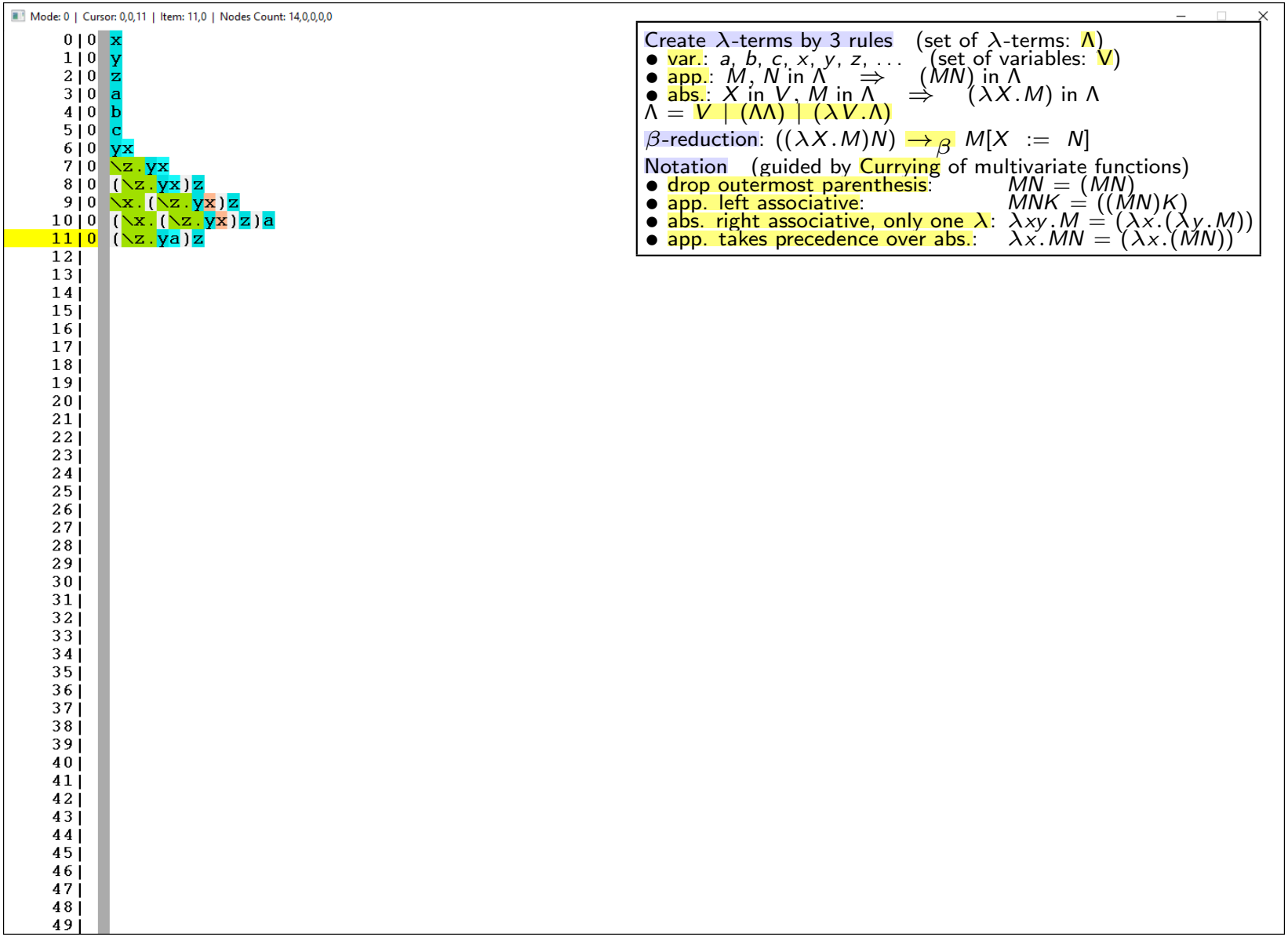

Mode: 0 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 7,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 1, 0 6 | 0 yx
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
• app.: M, N in Λ $\Rightarrow$ (MN) in Λ
• abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: $MN = (MN)$
• app. left associative: $MNK = ((MN)K)$
• abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
• app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

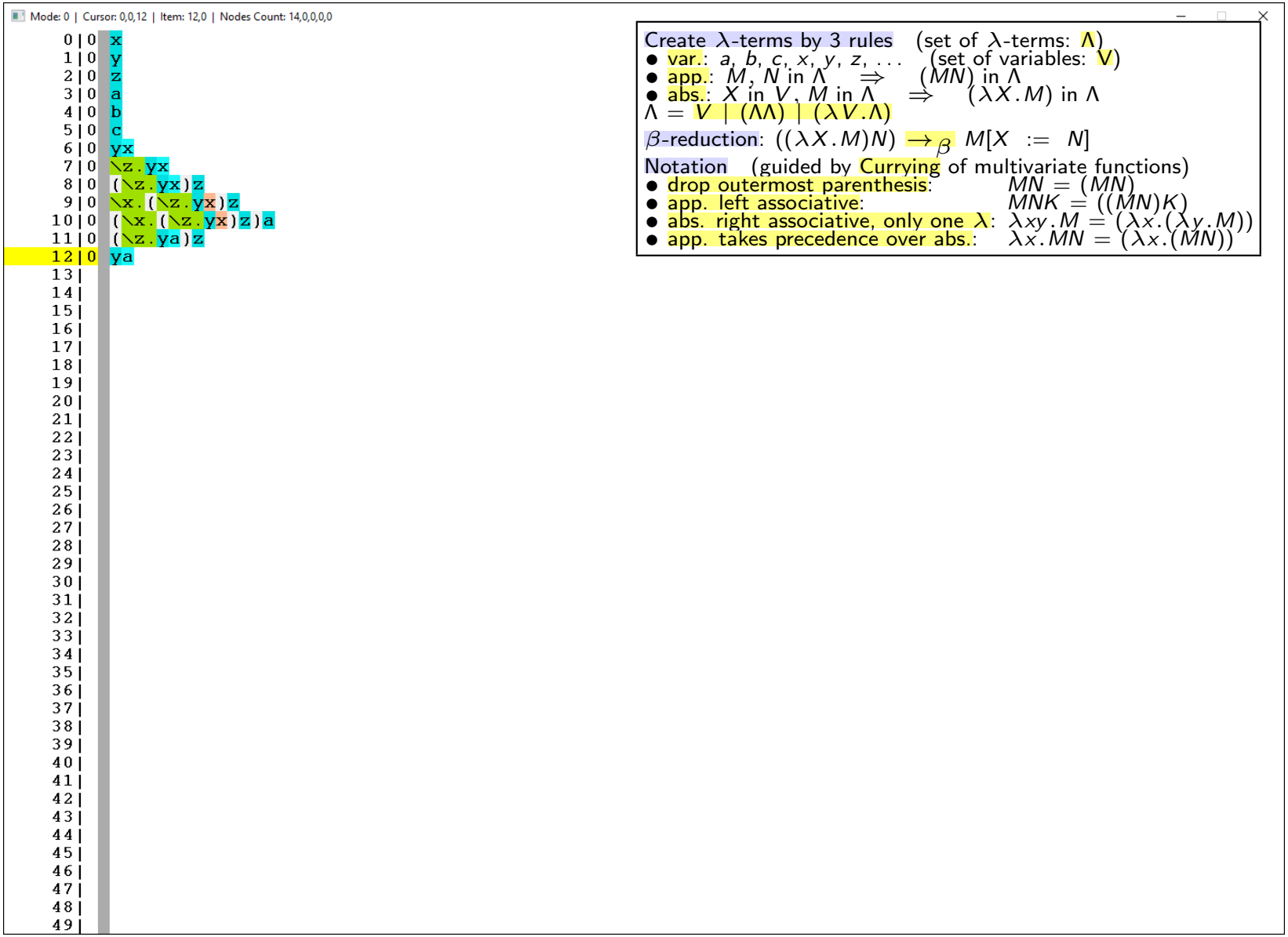

Mode: 0 | Cursor: 0,0,7 | Item: 7,0 | Nodes Count: 8,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 1, 0 6 | 0 yx
abs. 2, 6 7 | 0 \z.yx
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))


Mode: 0 | Cursor: 0,0,8 | Item: 8,0 | Nodes Count: 9,0,0,0,0
var.
var.
var.
var.
var.
var.
app. 1, 0
abs. 2, 6
app. 7, 2
x
y
z
a
b
c
yx
\z.yx
(\z.yx)z
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))


Mode: 0 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 10,0,0,0,0
var.
var.
var.
var.
var.
var.
app. 1, 0
abs. 2, 6
app. 7, 2
abs. 0, 8
0 | 0 x
1 | 0 y
2 | 0 z
3 | 0 a
4 | 0 b
5 | 0 c
6 | 0 yx
7 | 0 \z.yx
8 | 0 (\z.yx)z
9 | 0 \x.(\z.yx)z
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))


Mode: 0 | Cursor: 0,0,10 | Item: 10,0 | Nodes Count: 11,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 1, 0 6 | 0 yx
abs. 2, 6 7 | 0 \z.yx
app. 7, 2 8 | 0 (\z.yx)z
abs. 0, 8 9 | 0 \x.(\z.yx)z
app. 9, 3 10 | 0 (\x.(\z.yx)z)a
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))


Mode: 0 | Cursor: 0,0,11 | Item: 11,0 | Nodes Count: 14,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 1, 0 6 | 0 yx
abs. 2, 6 7 | 0 \z.yx
app. 7, 2 8 | 0 (\z.yx)z
abs. 0, 8 9 | 0 \x.(\z.yx)z
app. 9, 3 10 | 0 (\x.(\z.yx)z)a
β-red 10 11 | 0 (\z.ya)z
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))



Create λ-terms by 3 rules (set of λ-terms: Λ)

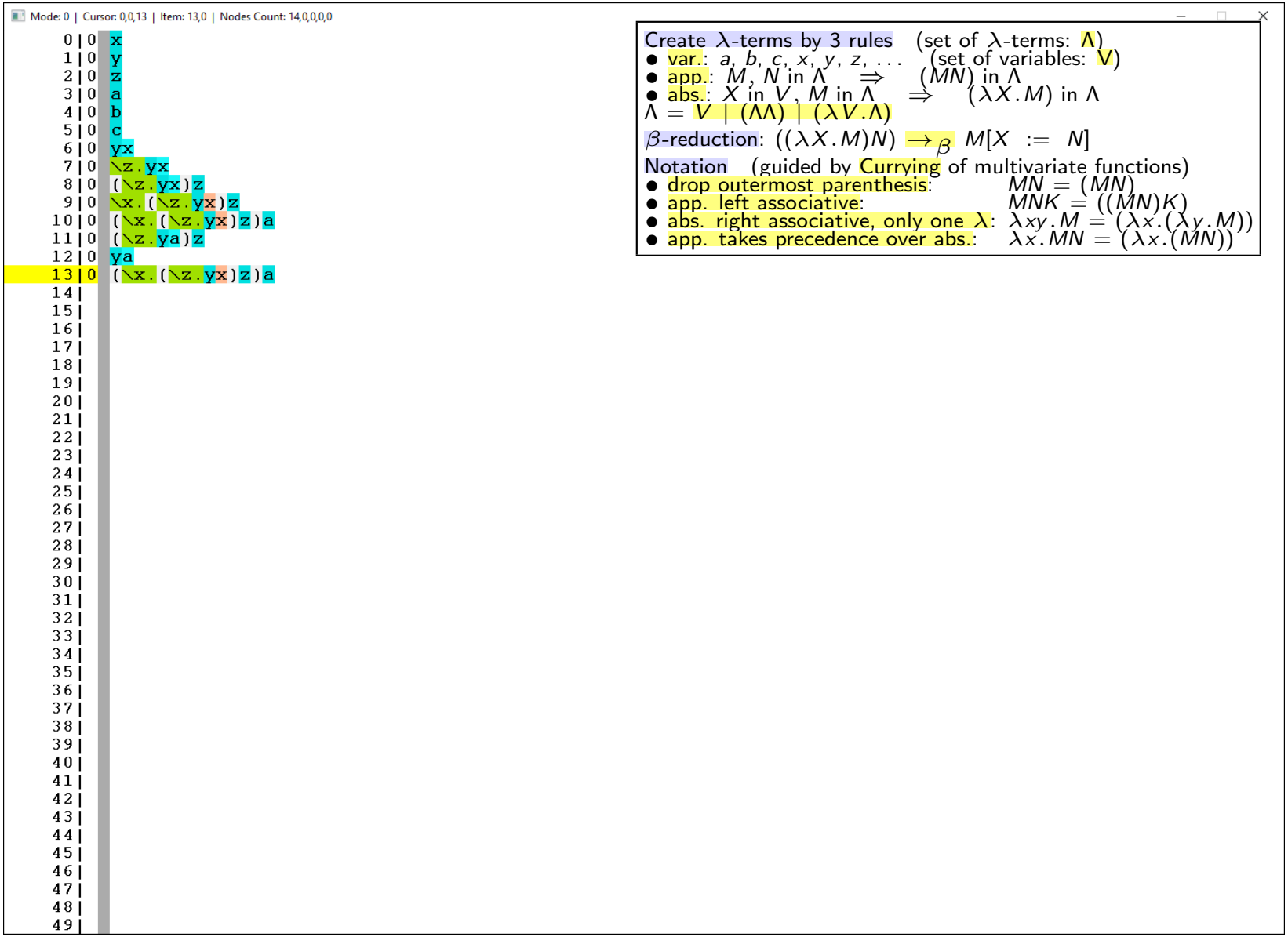
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

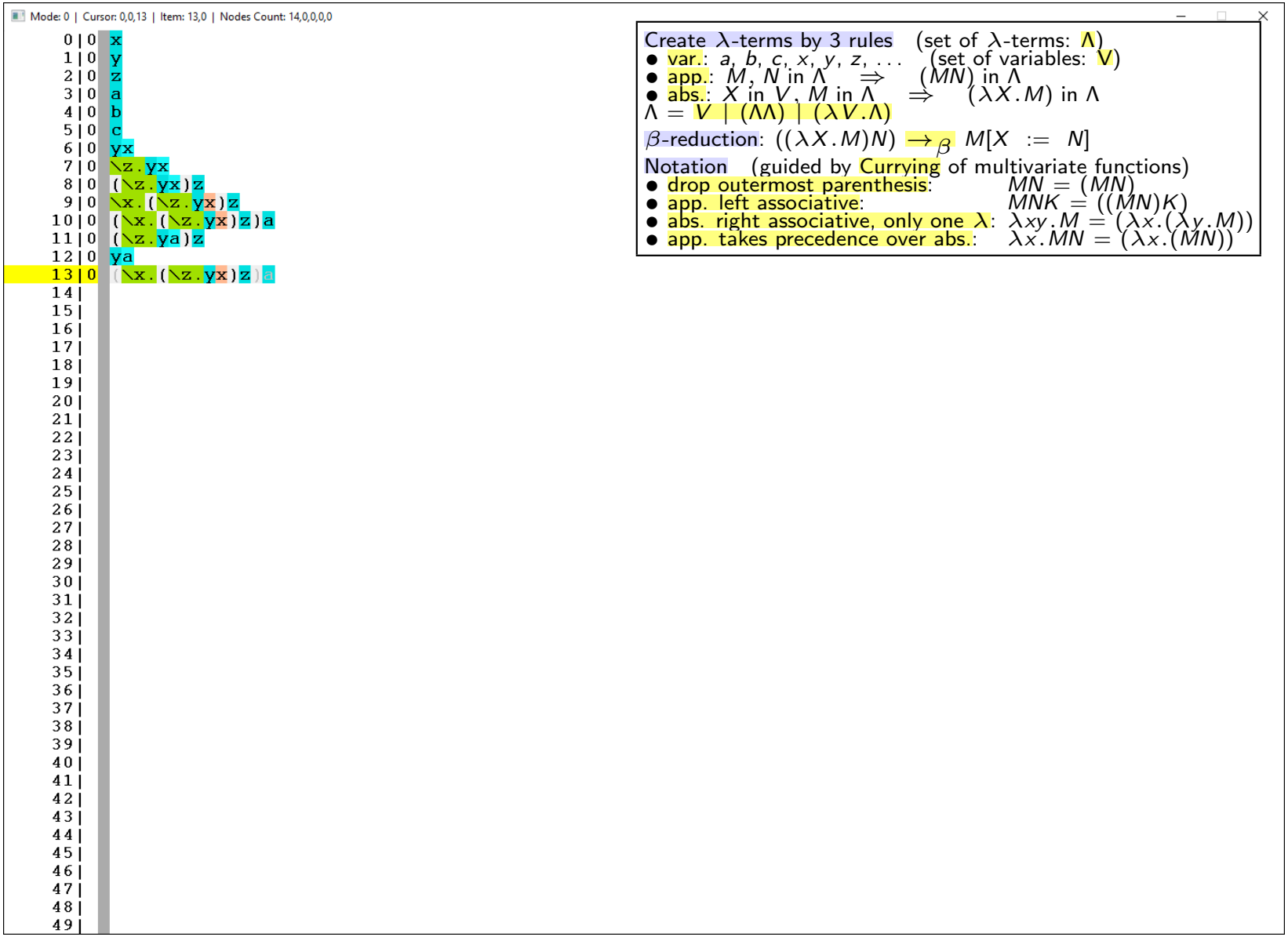
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

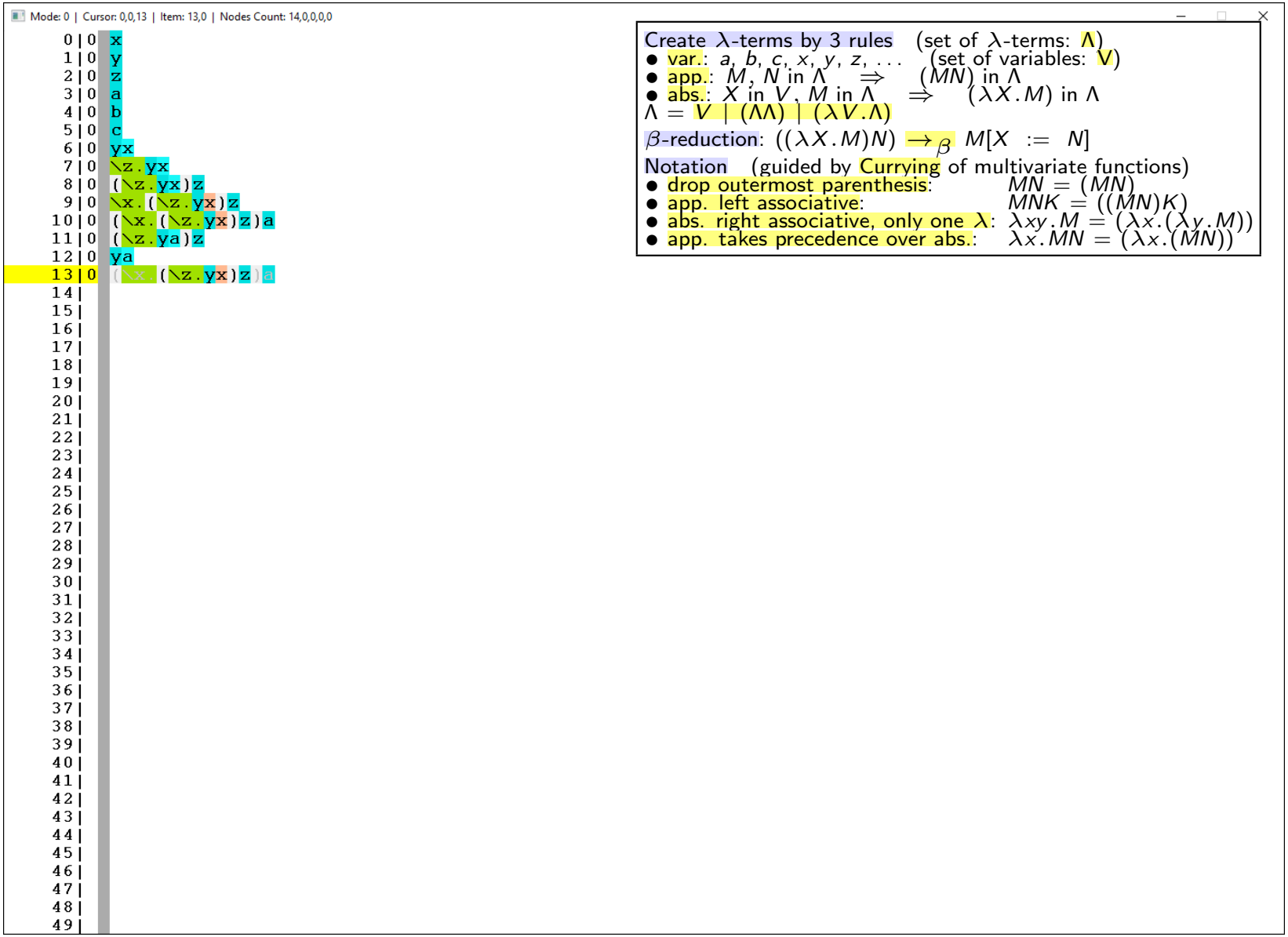
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$



Create λ-terms by 3 rules (set of λ-terms: Λ)

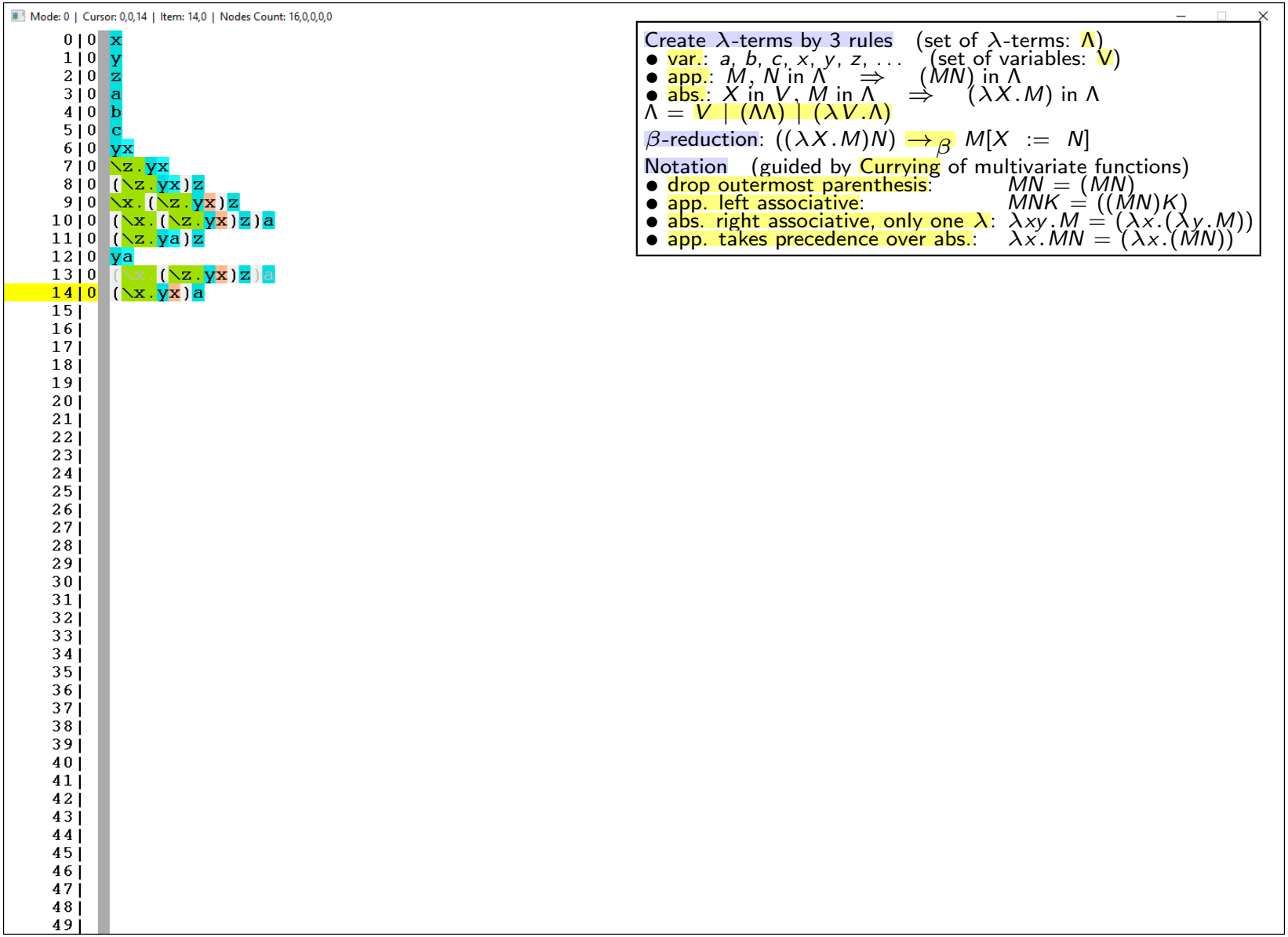
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

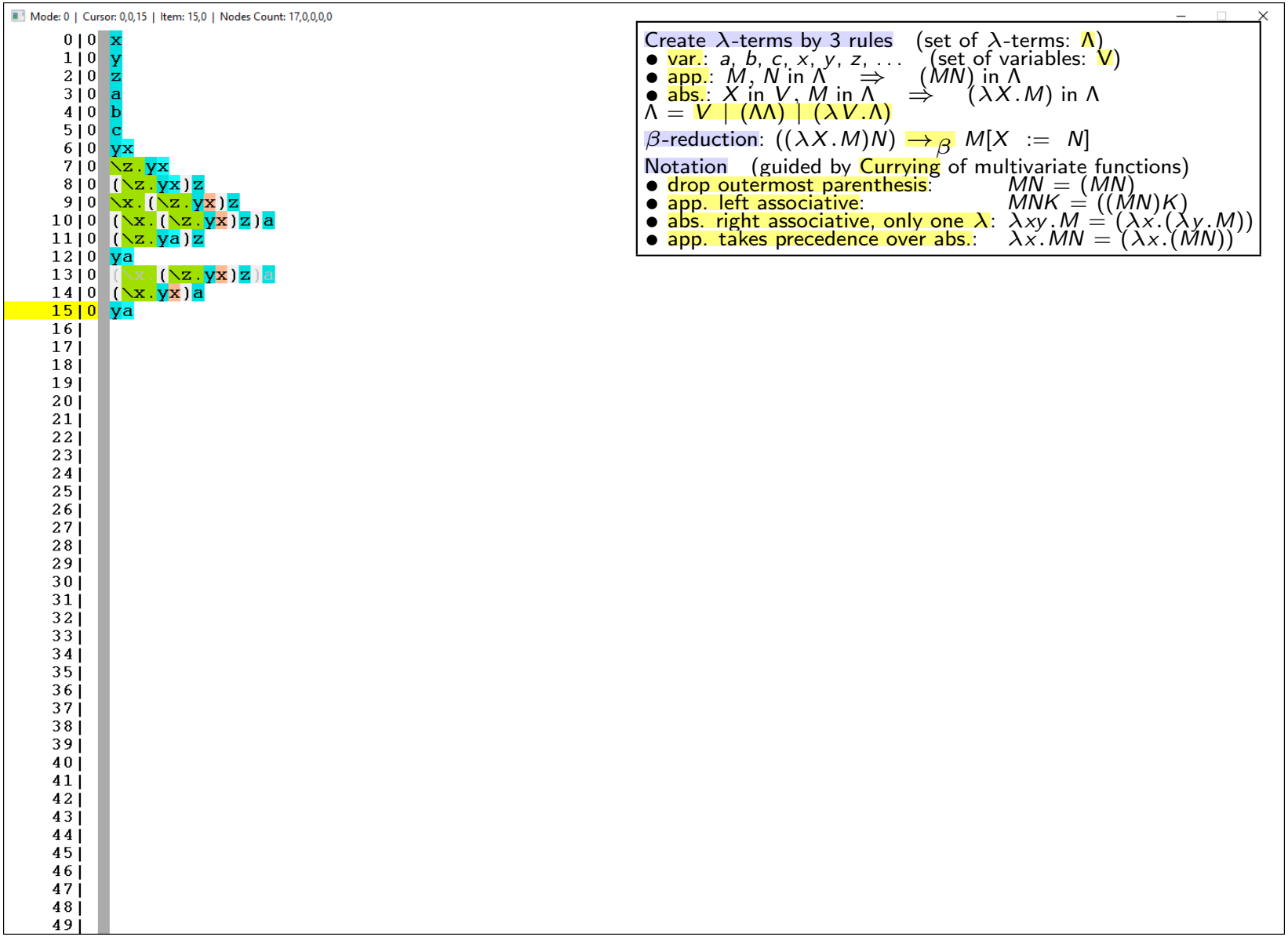
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$


Mode: 0 | Cursor: 0,0,14 | Item: 14,0 | Nodes Count: 16,0,0,0,0
var. 0 | 0 x
var. 1 | 0 y
var. 2 | 0 z
var. 3 | 0 a
var. 4 | 0 b
var. 5 | 0 c
app. 1, 0 6 | 0 yx
abs. 2, 6 7 | 0 \z.yx
app. 7, 2 8 | 0 (\z.yx)z
abs. 0, 8 9 | 0 \x.(\z.yx)z
app. 9, 3 10 | 0 (\x.(\z.yx)z)a
β-red 10 11 | 0 (\z.ya)z
β-red 11 12 | 0 ya
copy 10 13 | 0 (\x.(\z.yx)z)a
β-red 13 14 | 0 (\x.yx)a
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))



Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,24 | Item: 21,0 | Nodes Count: 139,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
	21 |	
	22 |	
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,24 | Item: 21,0 | Nodes Count: 143,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
succ zero, $\beta x2$	21 | 0	\fx.f((\x.x)x)
	22 |	
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,24 | Item: 21,0 | Nodes Count: 142,0,0,0,0

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
	22 |	
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,25 | Item: 22,0 | Nodes Count: 143,0,0,0,0

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
succ one	22 | 0	(\mfx.f(mfx))(\fx.fx)
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,25 | Item: 22,0 | Nodes Count: 147,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

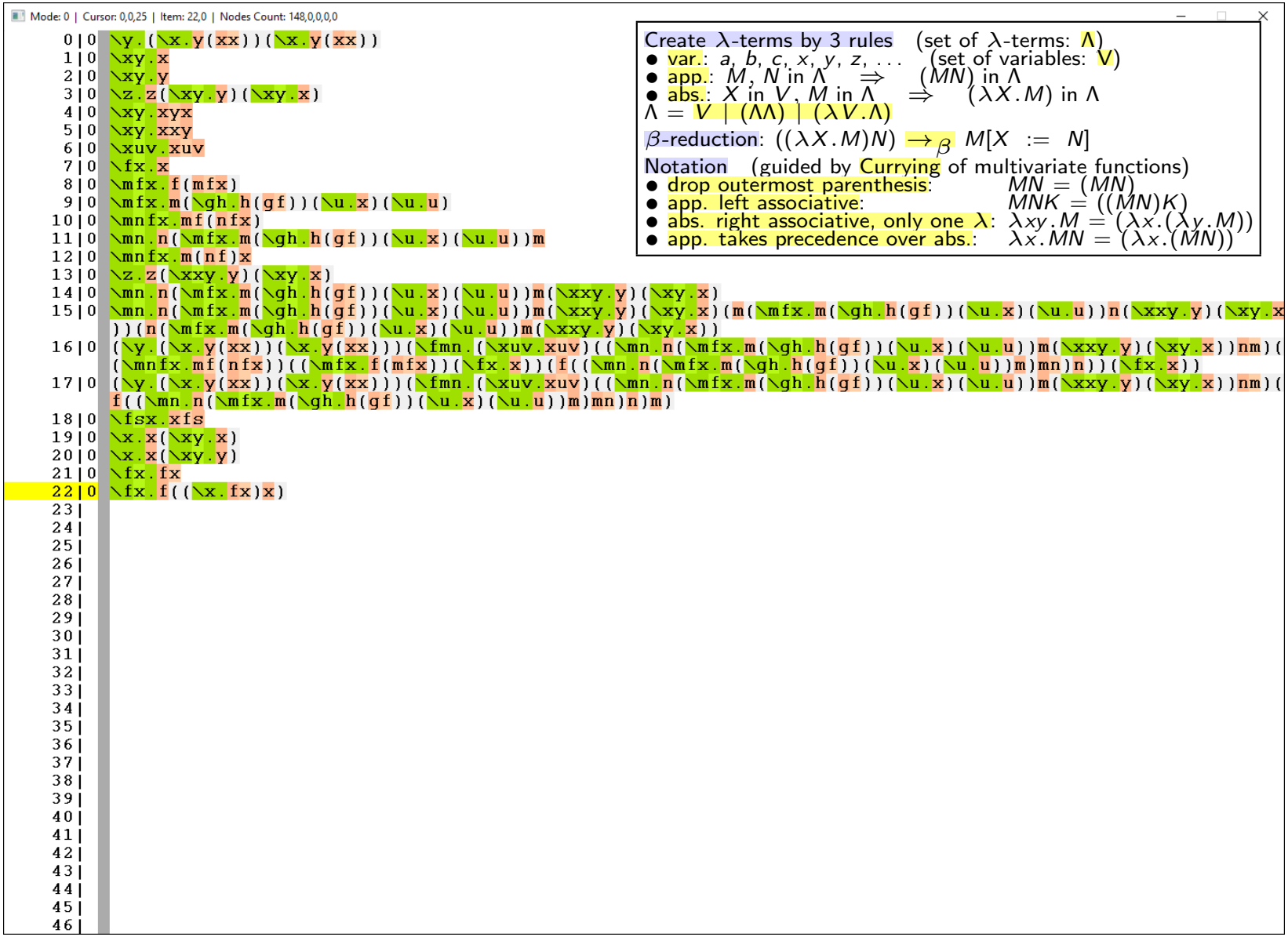
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
succ one, $\beta x1$	22 | 0	\fx.f((\fx.fx)fx)
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,25 | Item: 22,0 | Nodes Count: 146,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
succ one, $\beta x3$	22 | 0	\fx.f(fx)
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,25 | Item: 22,0 | Nodes Count: 146,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

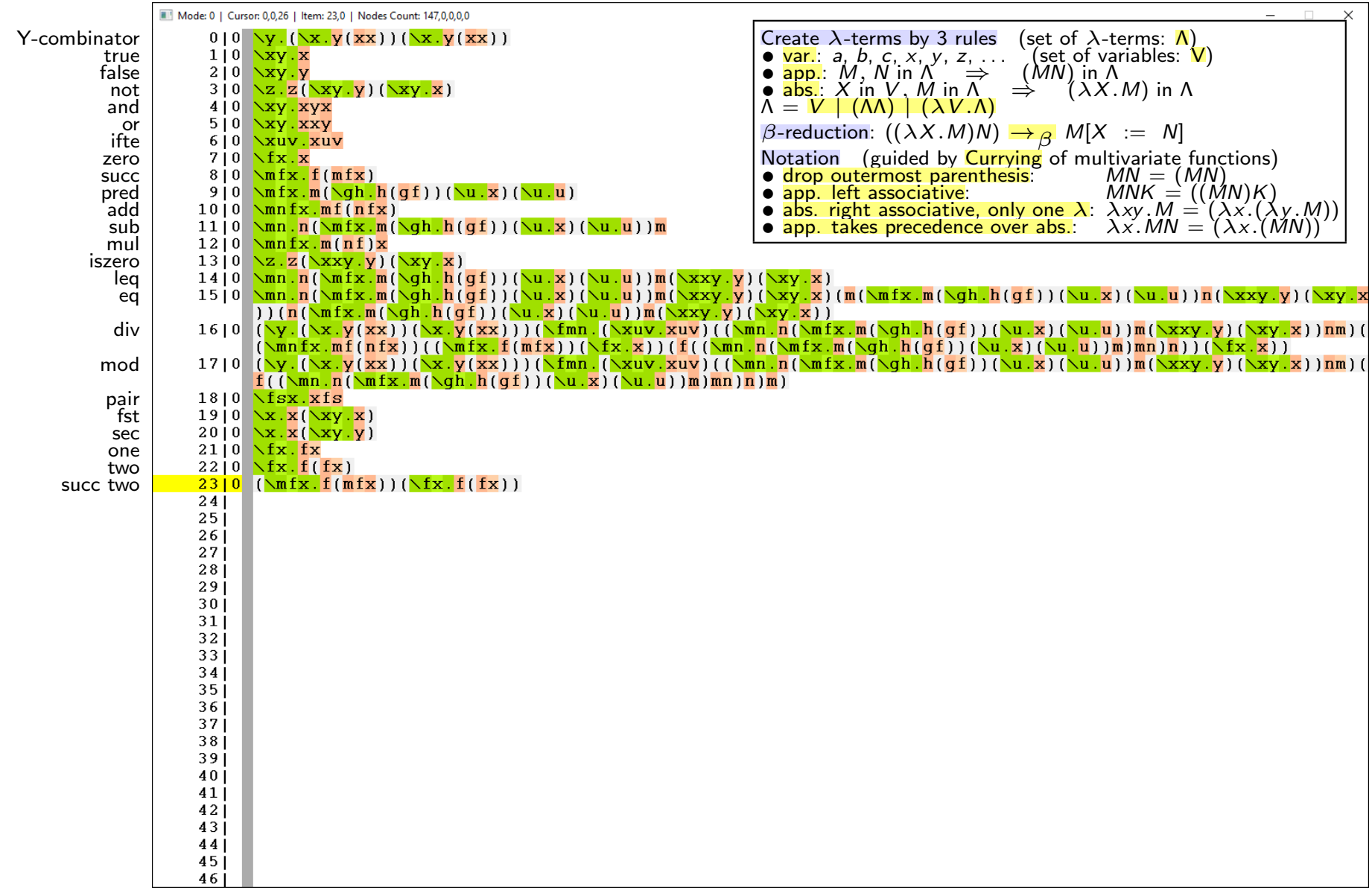
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
             23|
             24|
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,26 | Item: 23,0 | Nodes Count: 151,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

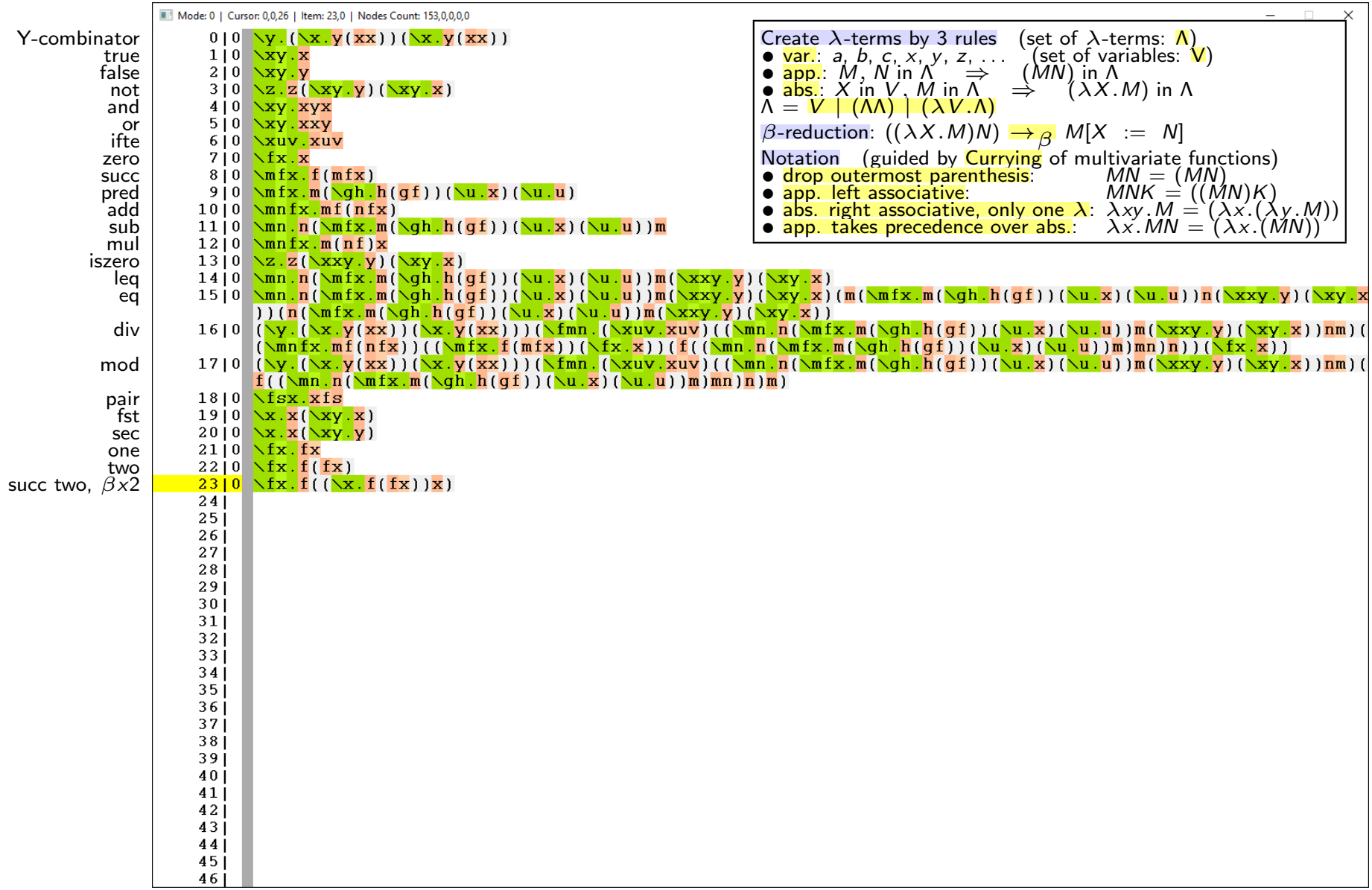
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
succ two, βx1 23|0  \fx.f((\fx.f(fx))fx)
             24|
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,26 | Item: 23,0 | Nodes Count: 151,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

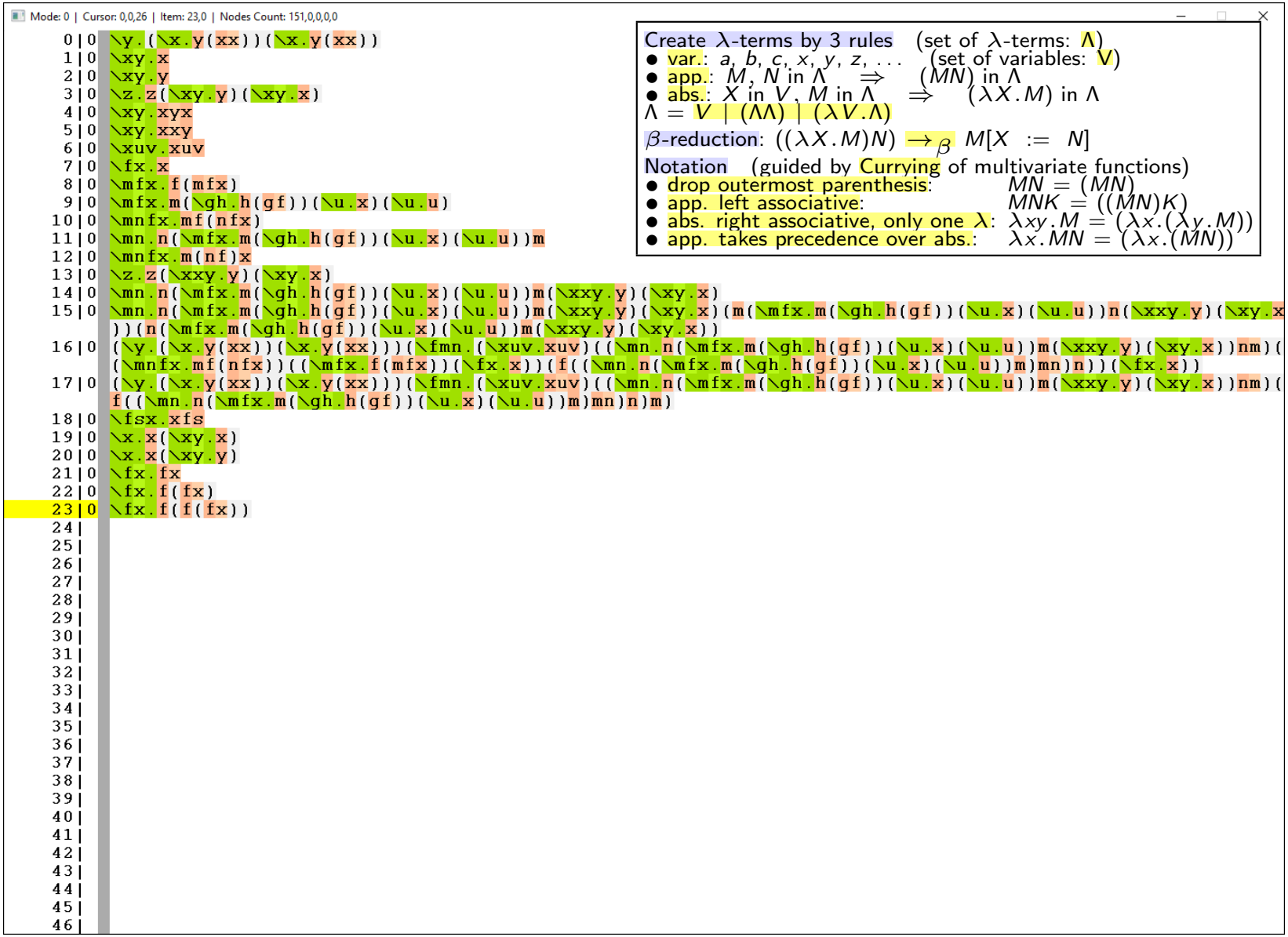
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
succ two, βx3 23|0  \fx.f(f(fx))
             24|
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus



A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 153,0,0,0,0

```
Y-combinator    0|0  \y.(\x.y(xx))(\x.y(xx))
true            1|0  \xy.x
false           2|0  \xy.y
not             3|0  \z.z(\xy.y)(\xy.x)
and             4|0  \xy.xyx
or              5|0  \xy.xxy
ifte            6|0  \xuv.xuv
zero            7|0  \fx.x
succ            8|0  \mfx.f(mfx)
pred            9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add            10|0  \mnfx.mf(nfx)
sub            11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul            12|0  \mnfx.m(nf)x
iszero         13|0  \z.z(\xxy.y)(\xy.x)
leq            14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq             15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                     ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div            16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                     (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod            17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                     f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair           18|0  \fsx.xfs
fst            19|0  \x.x(\xy.x)
sec            20|0  \x.x(\xy.y)
one            21|0  \fx.fx
two            22|0  \fx.f(fx)
three          23|0  \fx.f(f(fx))
add two three  24|0  (\mnfx.mf(nfx))(\fx.f(fx))(\fx.f(f(fx)))
               25|
               26|
               27|
               28|
               29|
               30|
               31|
               32|
               33|
               34|
               35|
               36|
               37|
               38|
               39|
               40|
               41|
               42|
               43|
               44|
               45|
               46|
```

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 157,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
add two three, βx1 24|0 (\nfx.(\fx.f(fx))f(nfx))(\fx.f(f(fx)))
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 157,0,0,0,0

```
Y-combinator   0|0  \y.(\x.y(xx))(\x.y(xx))
true           1|0  \xy.x
false          2|0  \xy.y
not            3|0  \z.z(\xy.y)(\xy.x)
and            4|0  \xy.xyx
or             5|0  \xy.xxy
ifte           6|0  \xuv.xuv
zero           7|0  \fx.x
succ           8|0  \mfx.f(mfx)
pred           9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add           10|0  \mnfx.mf(nfx)
sub           11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul           12|0  \mnfx.m(nf)x
iszero        13|0  \z.z(\xxy.y)(\xy.x)
leq           14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq            15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                    ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div           16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod           17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair          18|0  \fsx.xfs
fst           19|0  \x.x(\xy.x)
sec           20|0  \x.x(\xy.y)
one           21|0  \fx.fx
two           22|0  \fx.f(fx)
three         23|0  \fx.f(f(fx))
add two three, βx2  24|0  \fx.(\fx.f(fx))f((\fx.f(f(fx)))fx)
                25|
                26|
                27|
                28|
                29|
                30|
                31|
                32|
                33|
                34|
                35|
                36|
                37|
                38|
                39|
                40|
                41|
                42|
                43|
                44|
                45|
                46|
```

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 159,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
add two three, $\beta x3$	24 | 0	\fx.(\x.f(fx))((\fx.f(f(fx)))fx)
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 157,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
add two three, $\beta x4$	24 | 0	\fx.f(f((\fx.f(f(fx)))fx))
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 160,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
add two three, $\beta x5$	24 | 0	\fx.f(f((\x.f(f(fx)))x))
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 158,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
three        23|0  \fx.f(f(fx))
add two three, βx6 24|0  \fx.f(f(f(f(fx))))
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 158,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 159,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
mul three	25 | 0	(\mnfx.m(nf)x)(\fx.f(f(fx)))
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 160,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
mul three five 25|0 (\mnfx.m(nf)x)(\fx.f(f(fx)))(\fx.f(f(f(f(fx)))))
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 164,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
mul three five, βx1 25|0 (\nfx.(\fx.f(f(fx)))(nf)x)(\fx.f(f(f(f(fx)))))
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 163,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

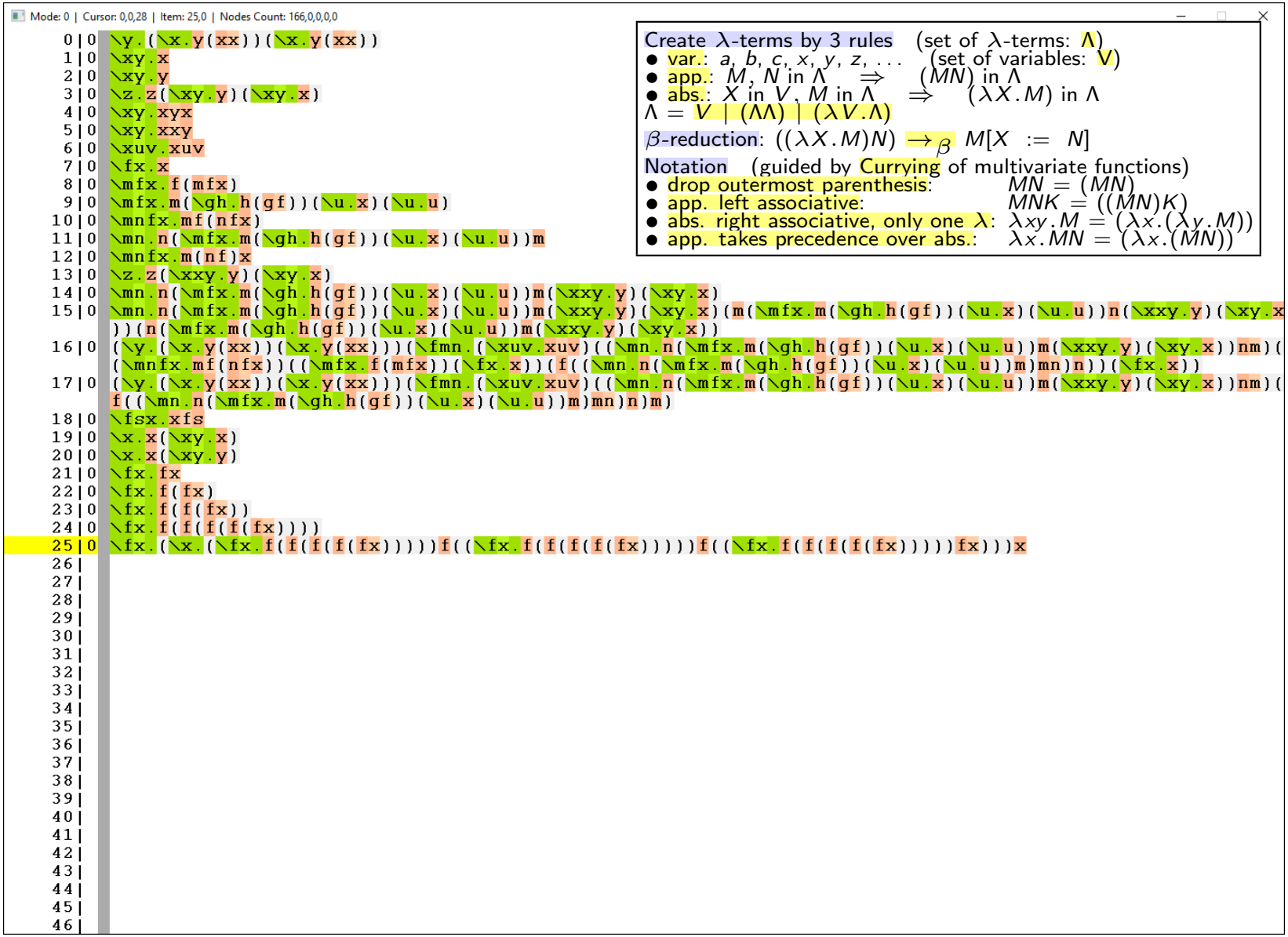
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
mul three five, $\beta x2$	25 | 0	\fx.(\fx.f(f(fx)))((\fx.f(f(f(f(fx)))))f)x
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 164,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

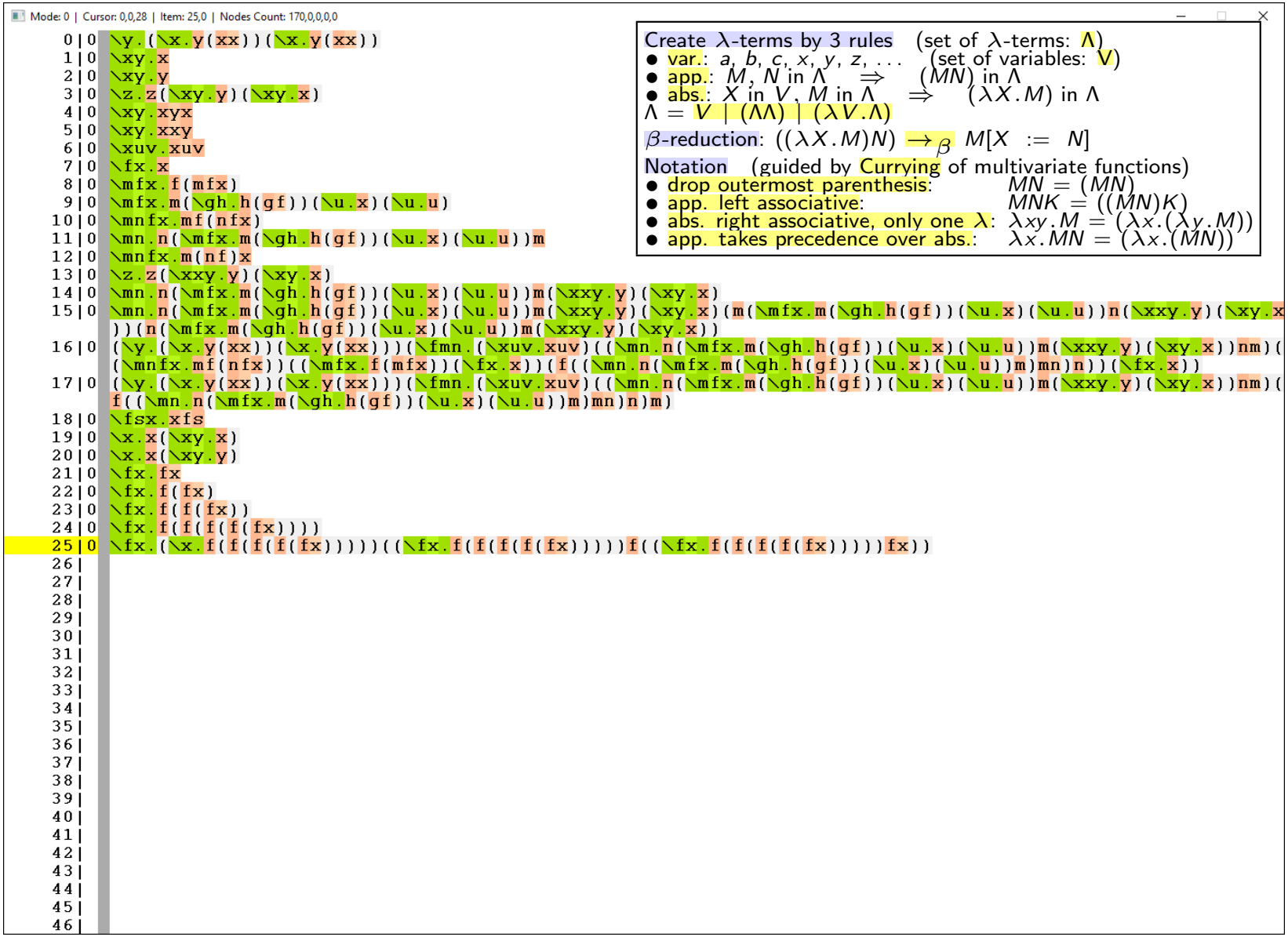
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

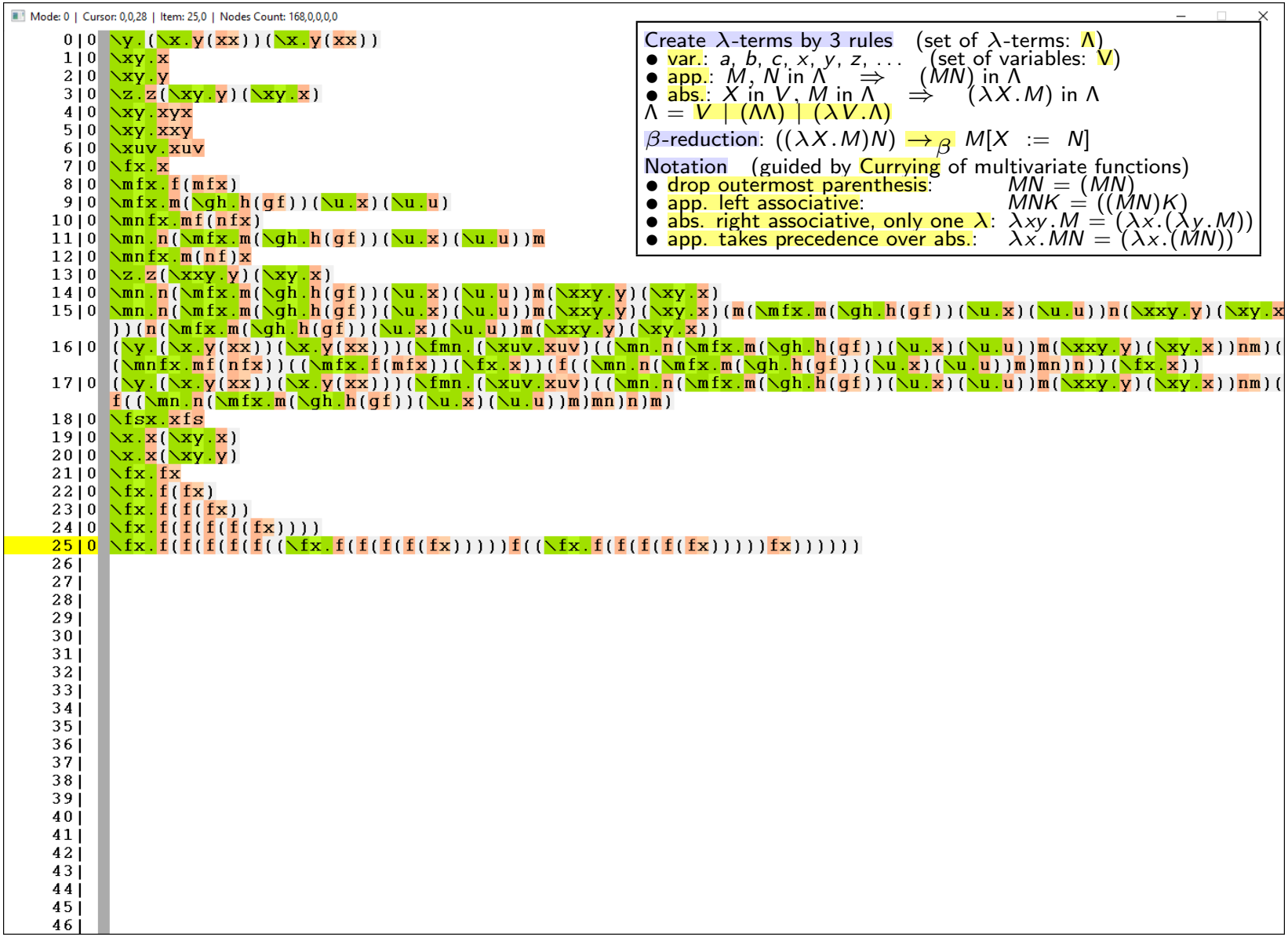
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
mul three five, $\beta x4$	25 | 0	\fx.(\fx.f(f(f(f(fx)))))f((\fx.f(f(f(f(fx)))))f((\fx.f(f(f(f(fx)))))fx))
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus



A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 174,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
mul three five, βx7 25|0 \fx.f(f(f(f(f((\x.f(f(f(f(fx))))))((\fx.f(f(f(f(fx)))))fx))))))
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 172,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

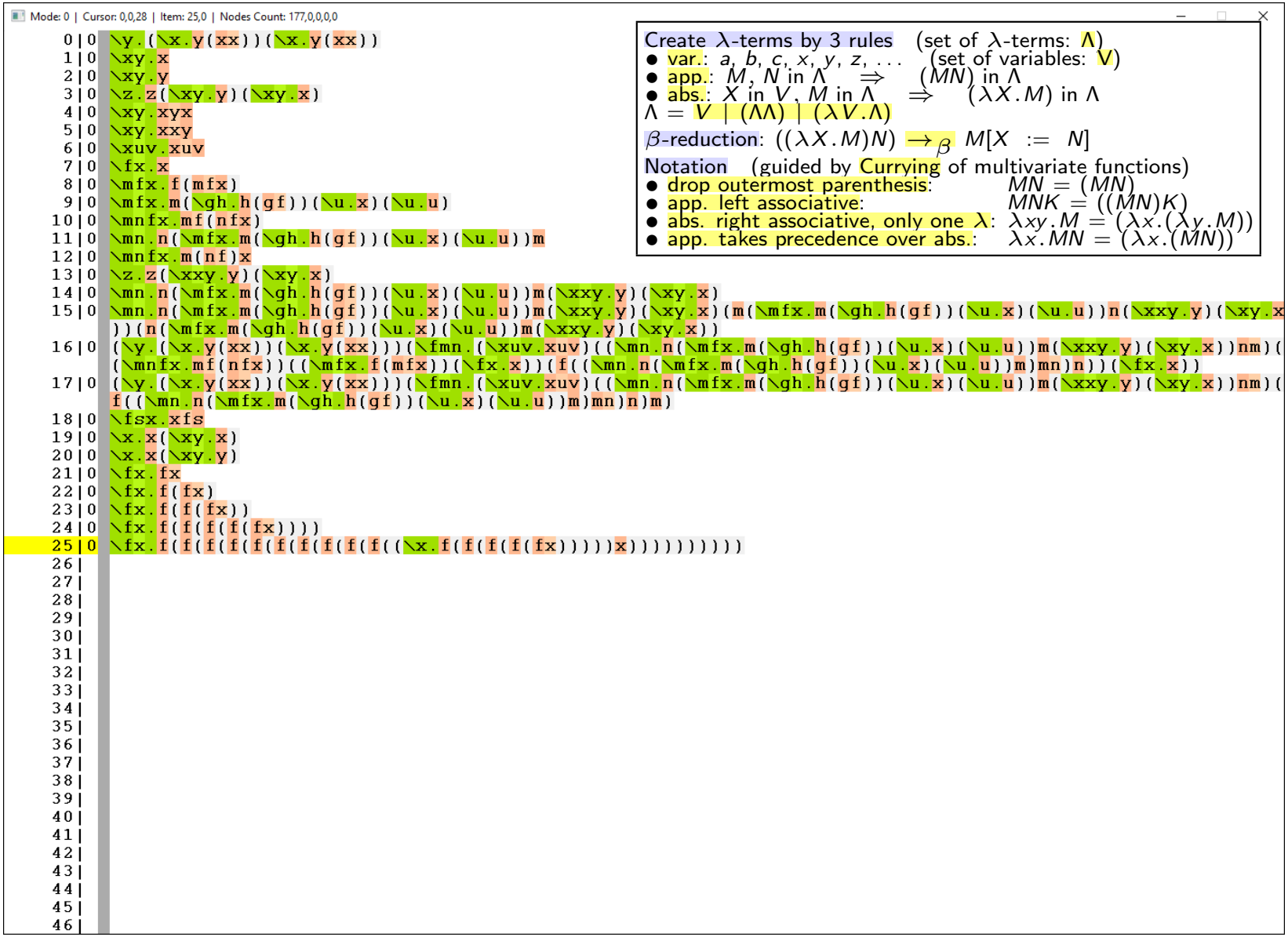
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

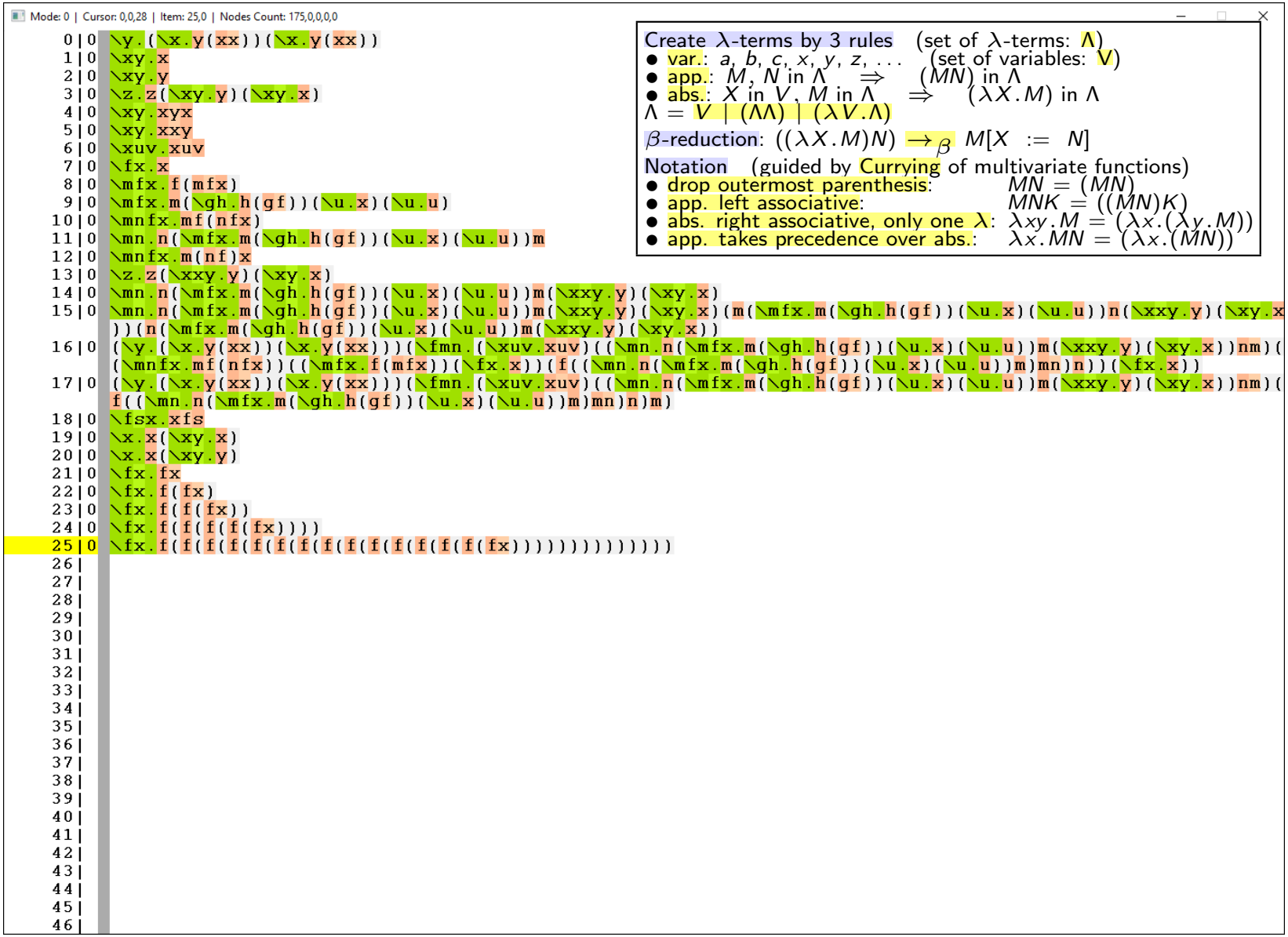
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
mul three five, $\beta x8$	25 | 0	\fx.f(f(f(f(f(f(f(f(f(f((\fx.f(f(f(f(fx)))))fx))))))))))
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 175,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

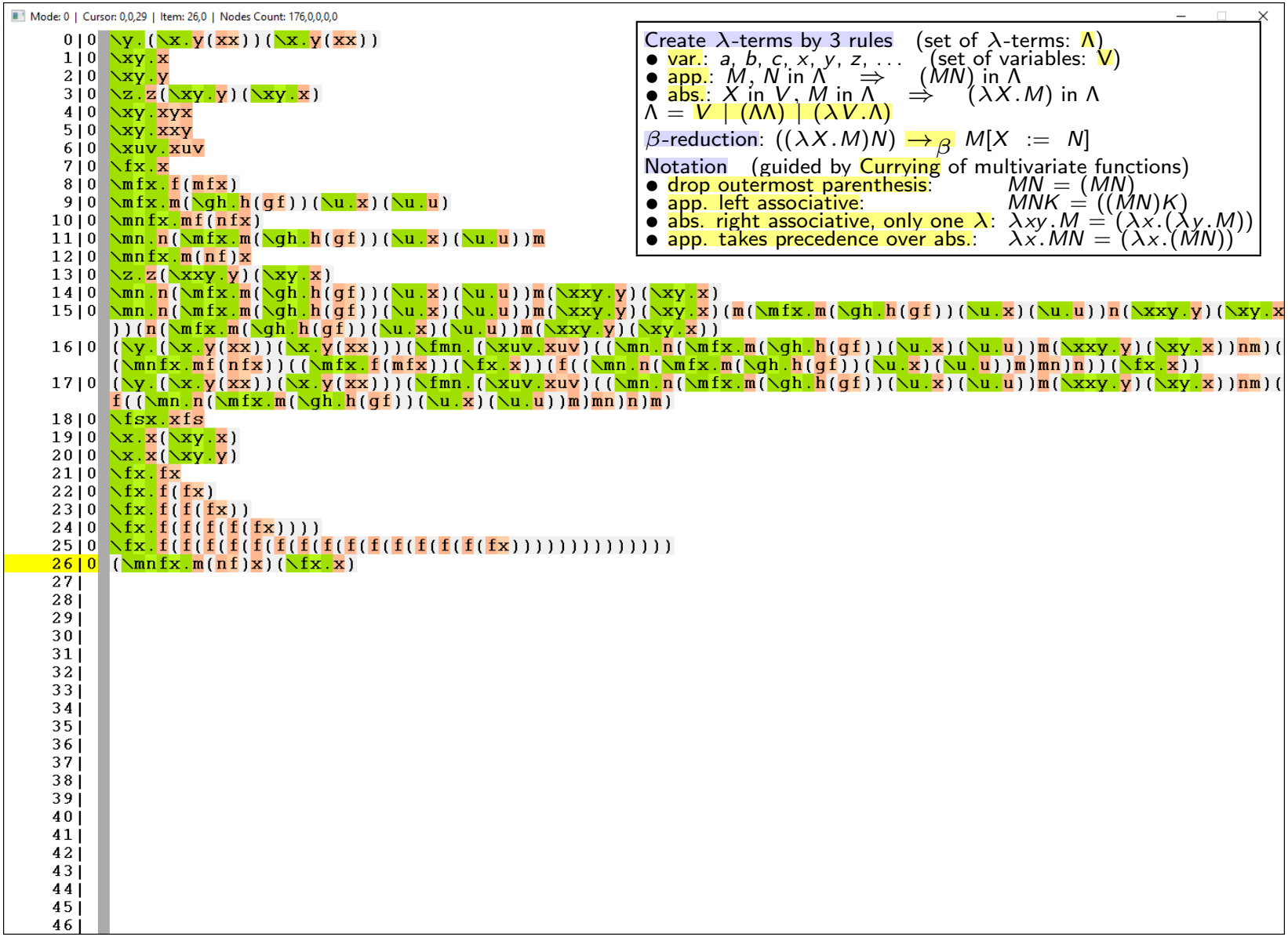
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

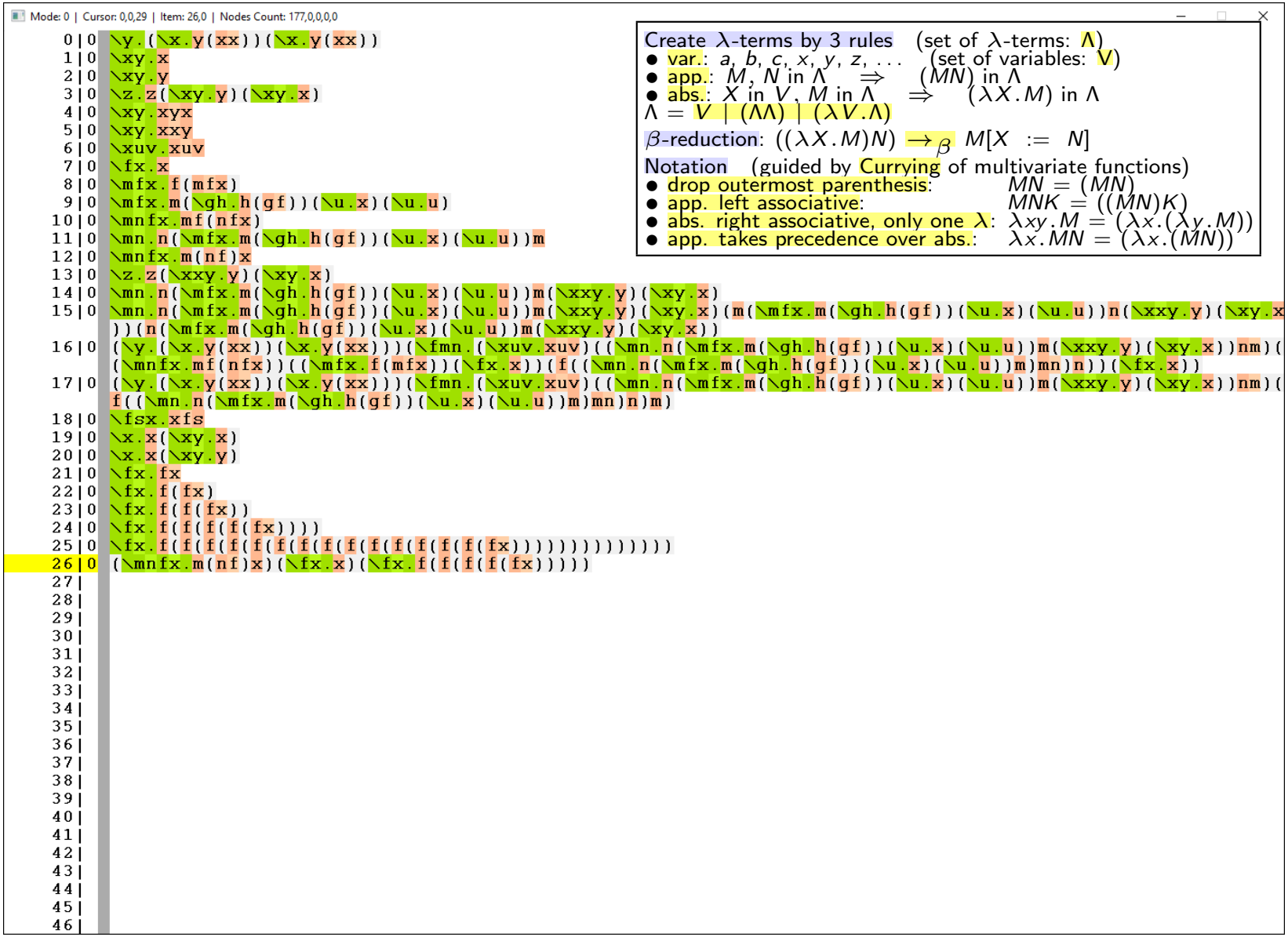
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
fifteen	25 | 0	\fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx))))))))))))))))
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

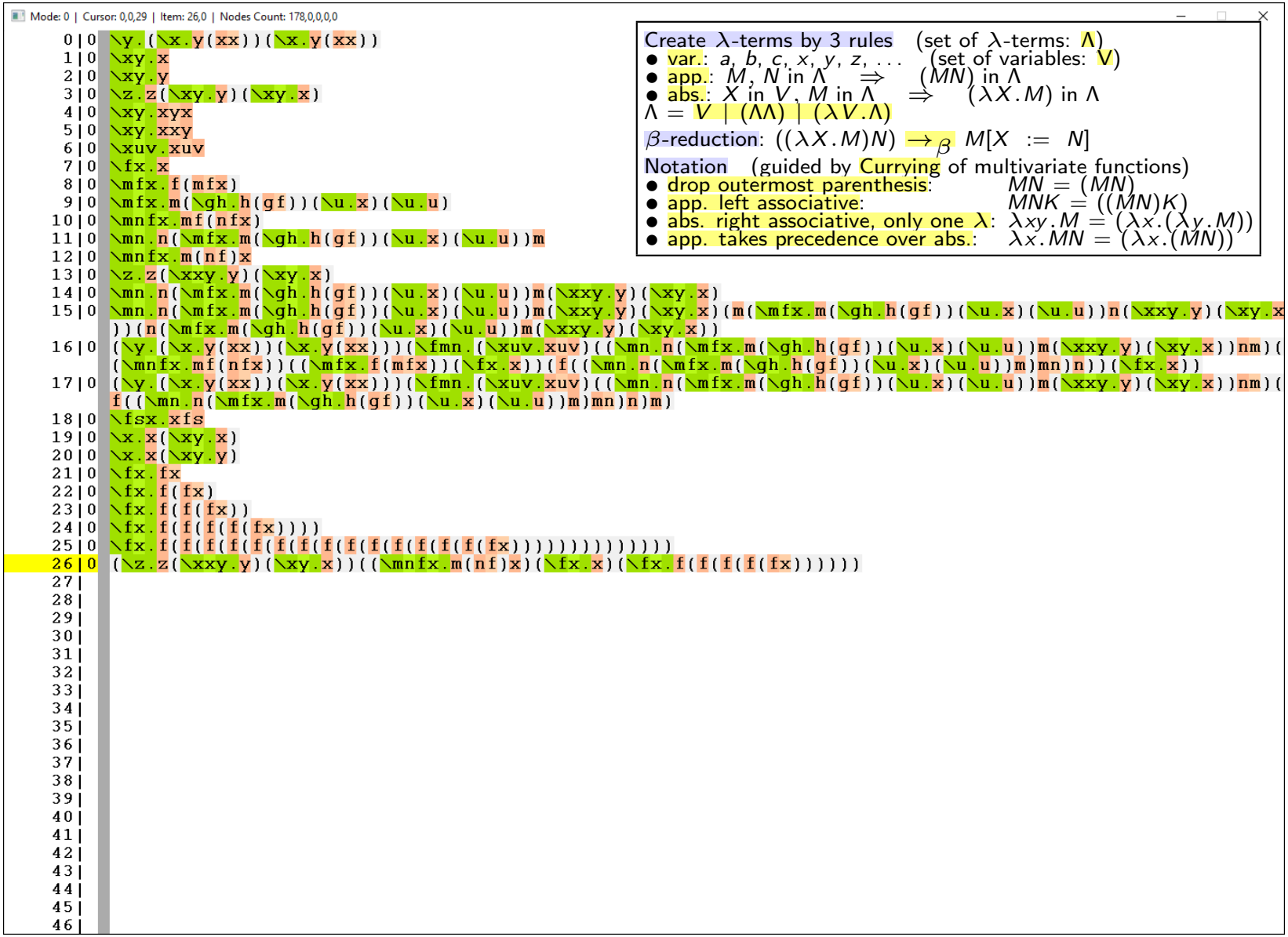
A crash course in λ-calculus



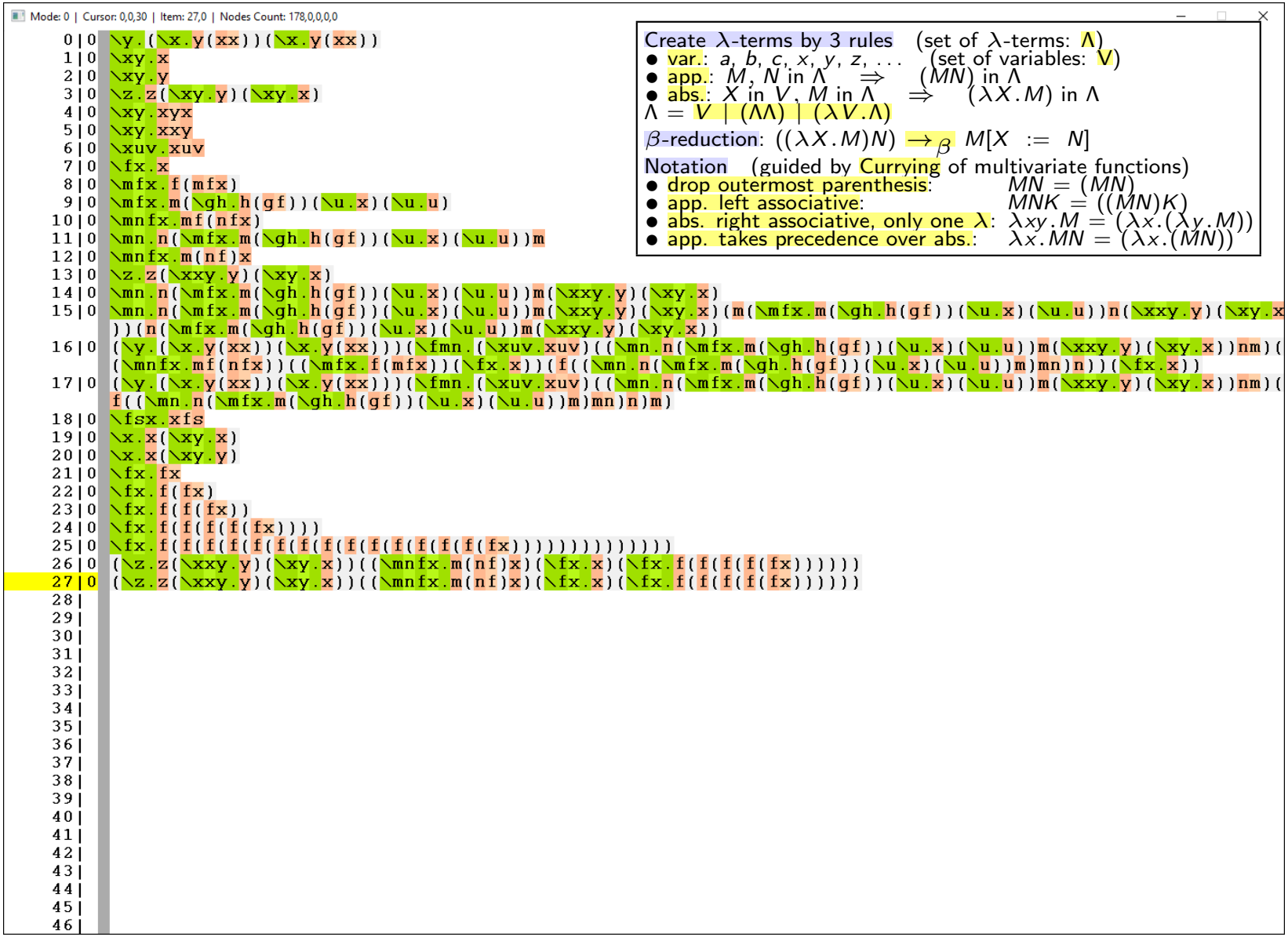
A crash course in λ-calculus



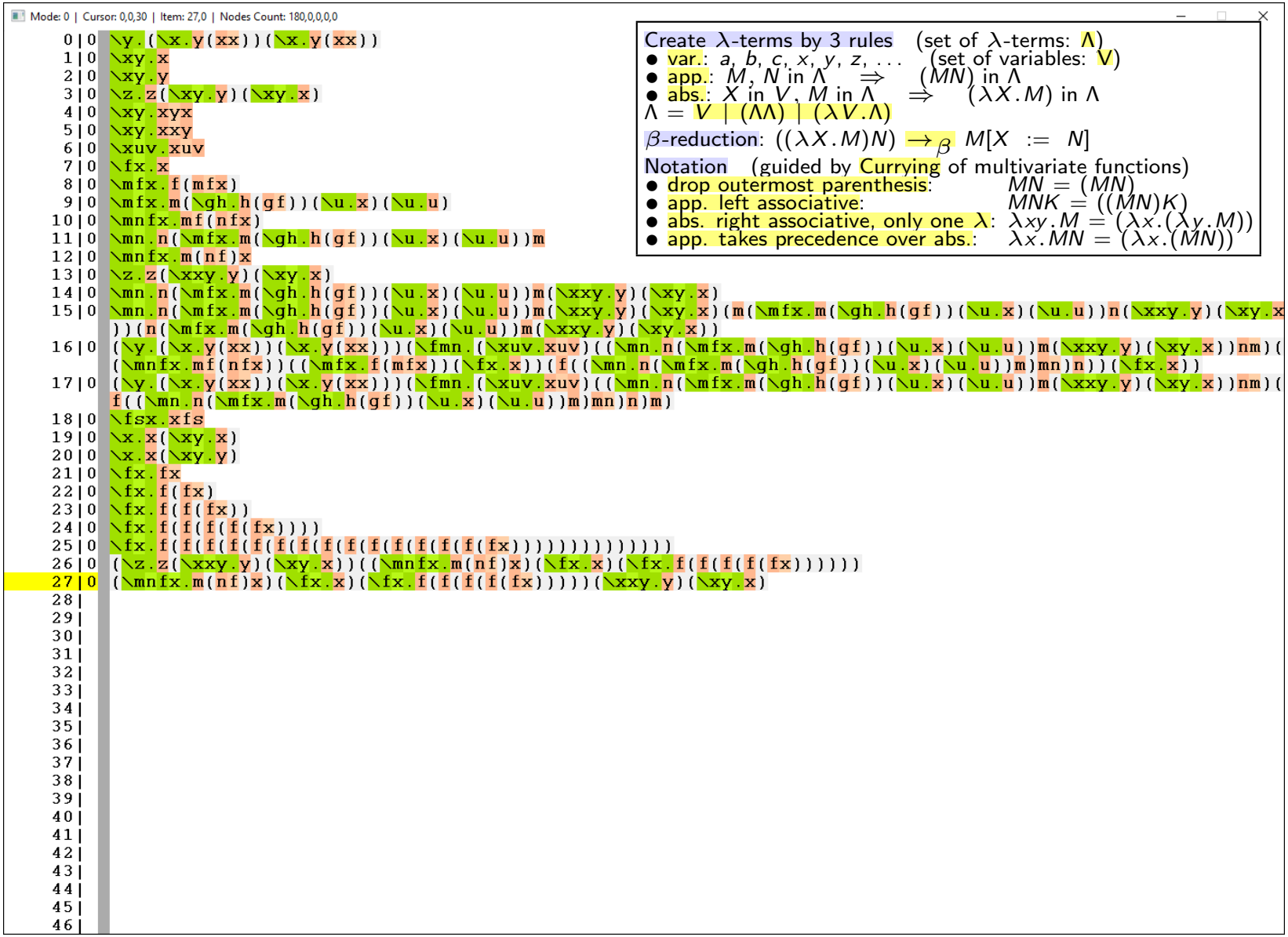
A crash course in λ-calculus



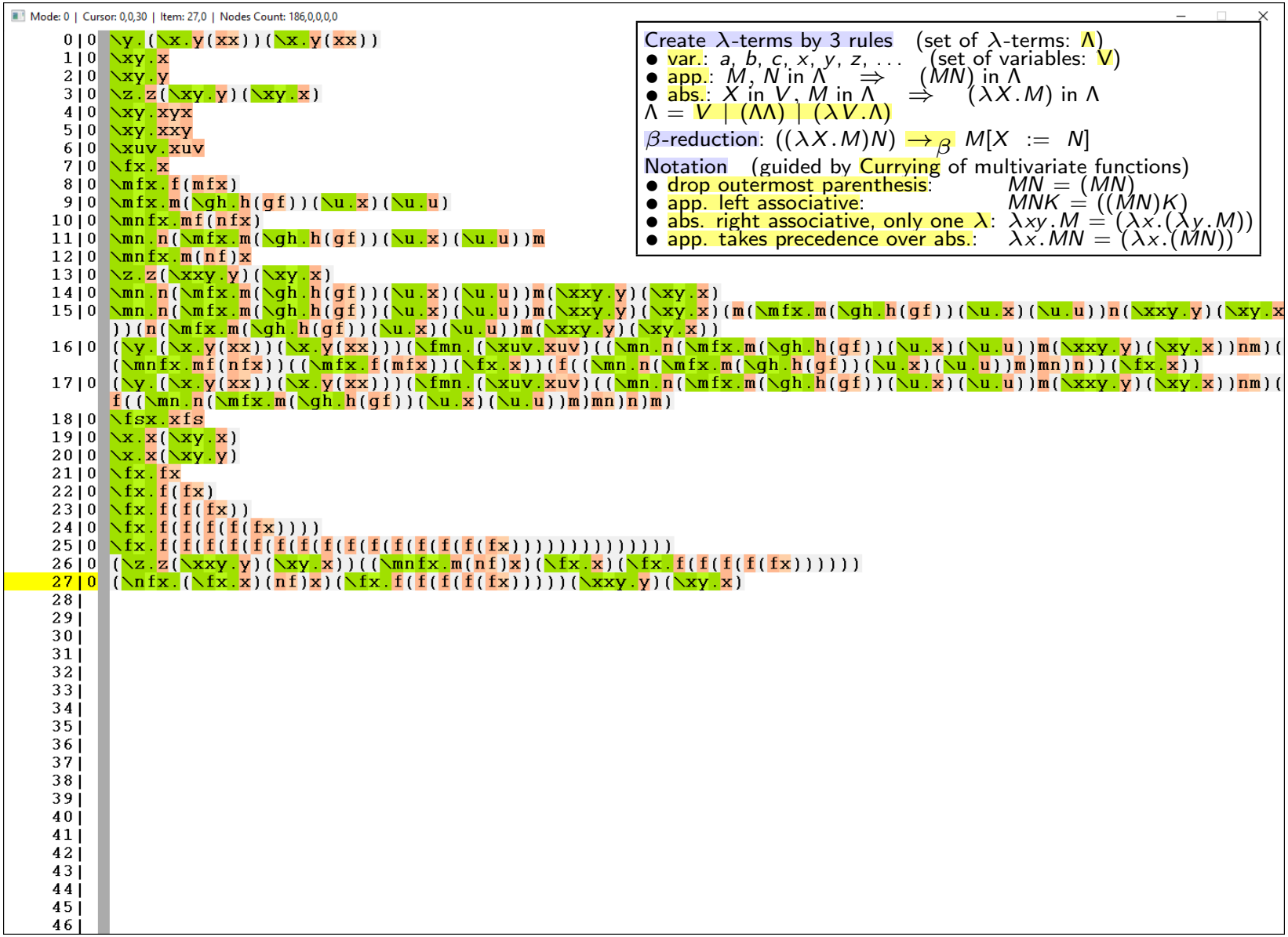
A crash course in λ-calculus

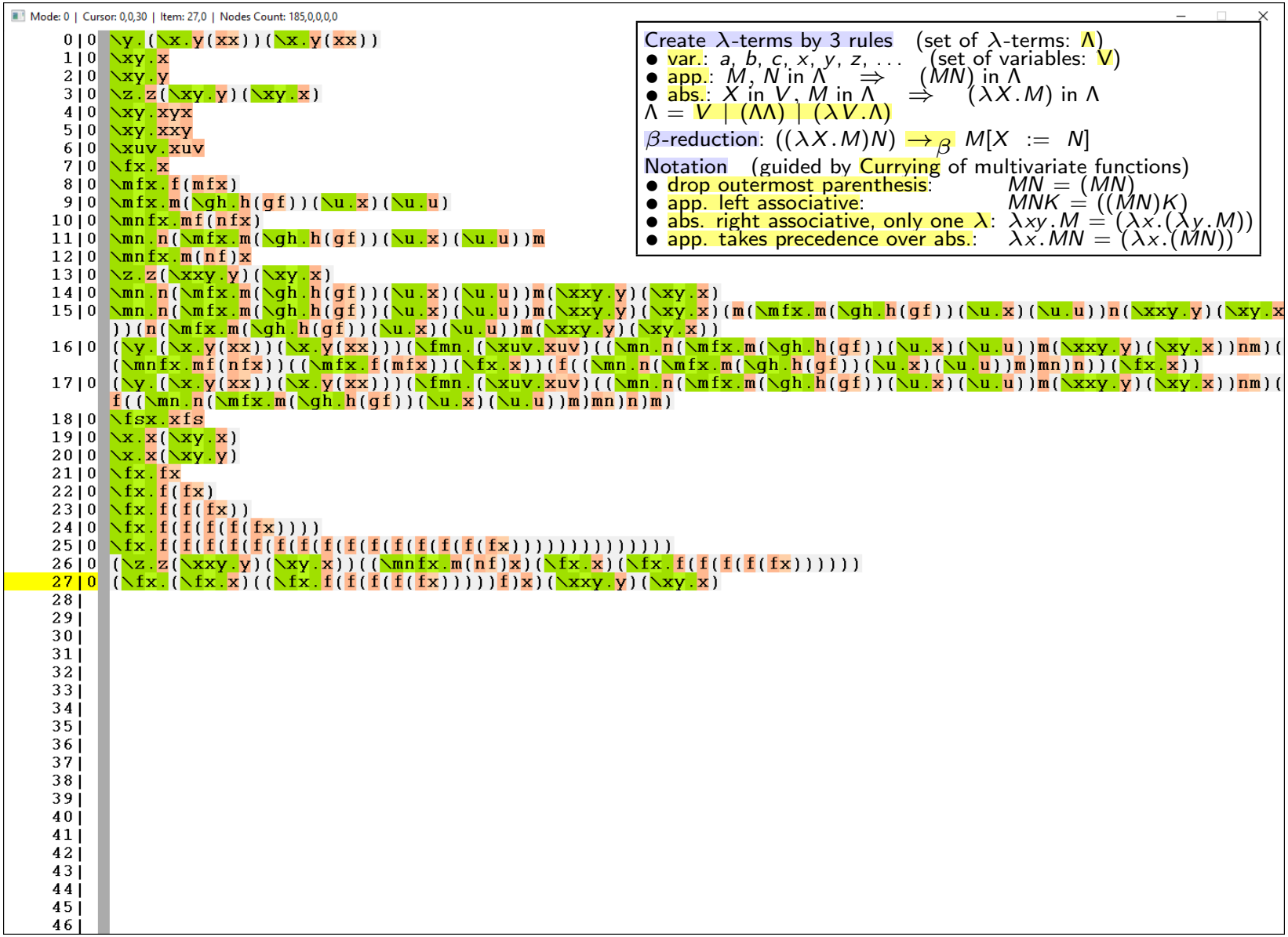


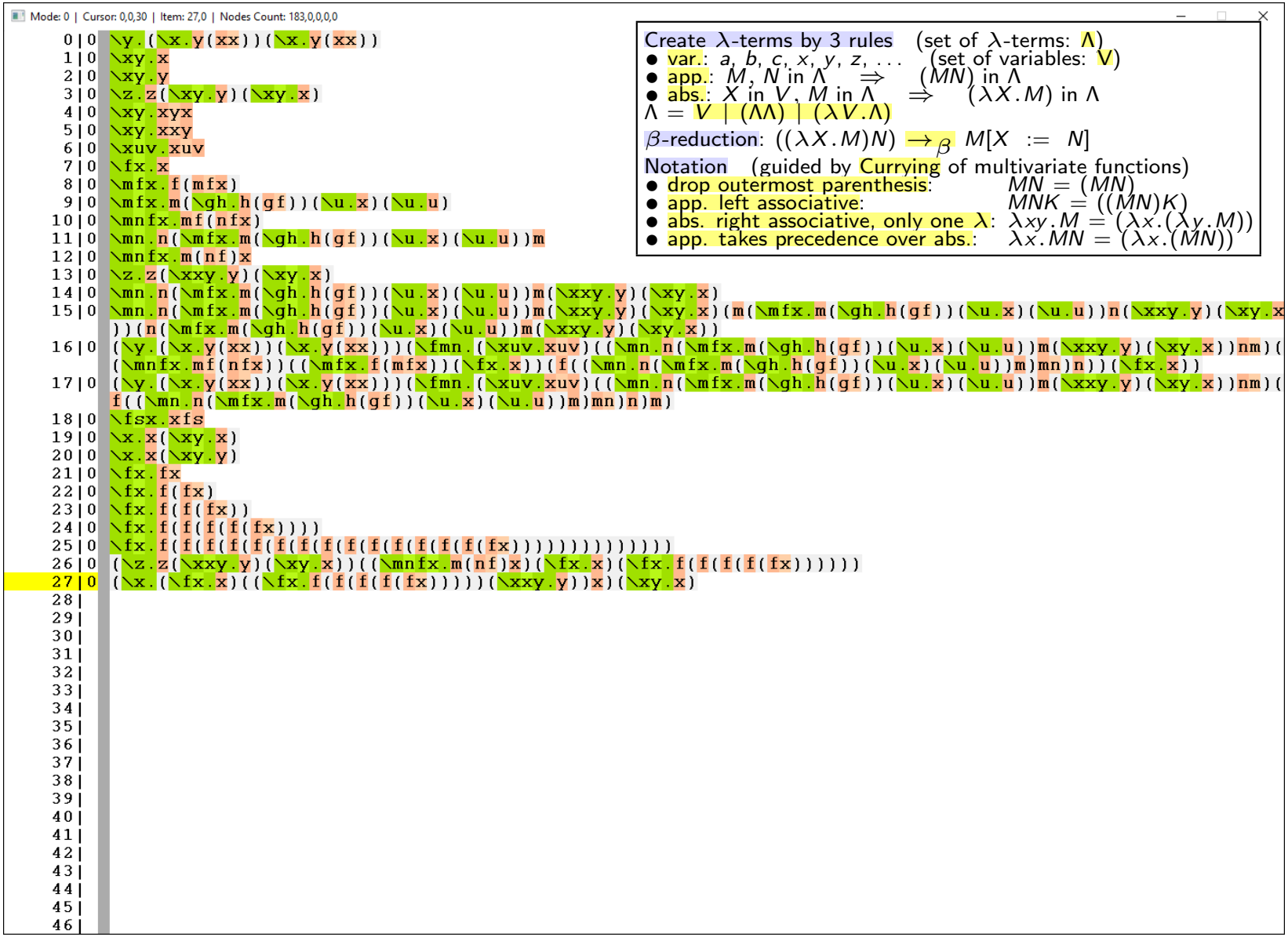
A crash course in λ-calculus



A crash course in λ-calculus




Mode: 0 | Cursor: 0,0,30 | Item: 27,0 | Nodes Count: 185,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
fifteen
iszero (mul zero five)
copy 26, βx3


Mode: 0 | Cursor: 0,0,30 | Item: 27,0 | Nodes Count: 183,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
fifteen
iszero (mul zero five)
copy 26, βx4

Mode: 0 | Cursor: 0,0,30 | Item: 27,0 | Nodes Count: 181,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

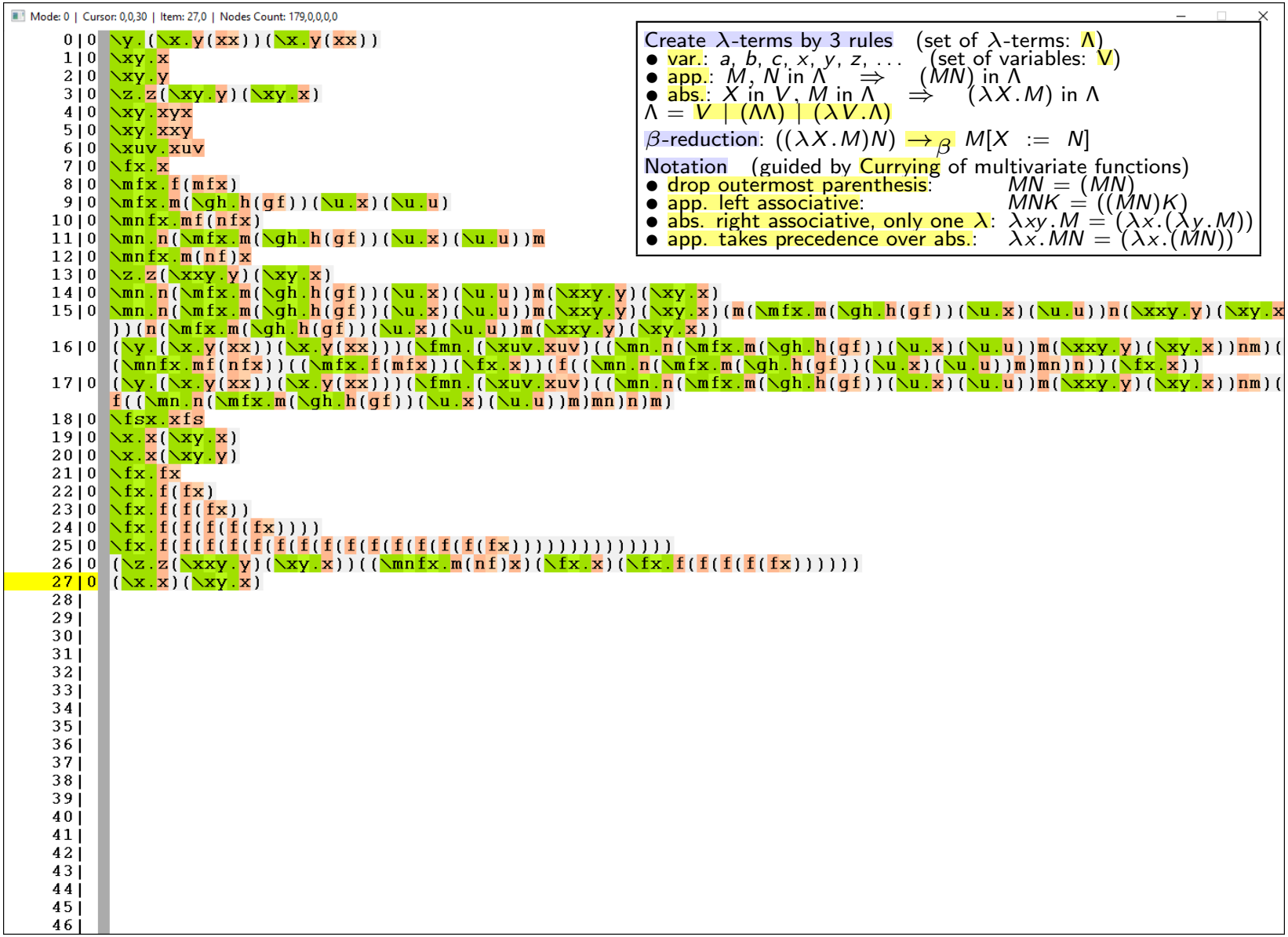
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

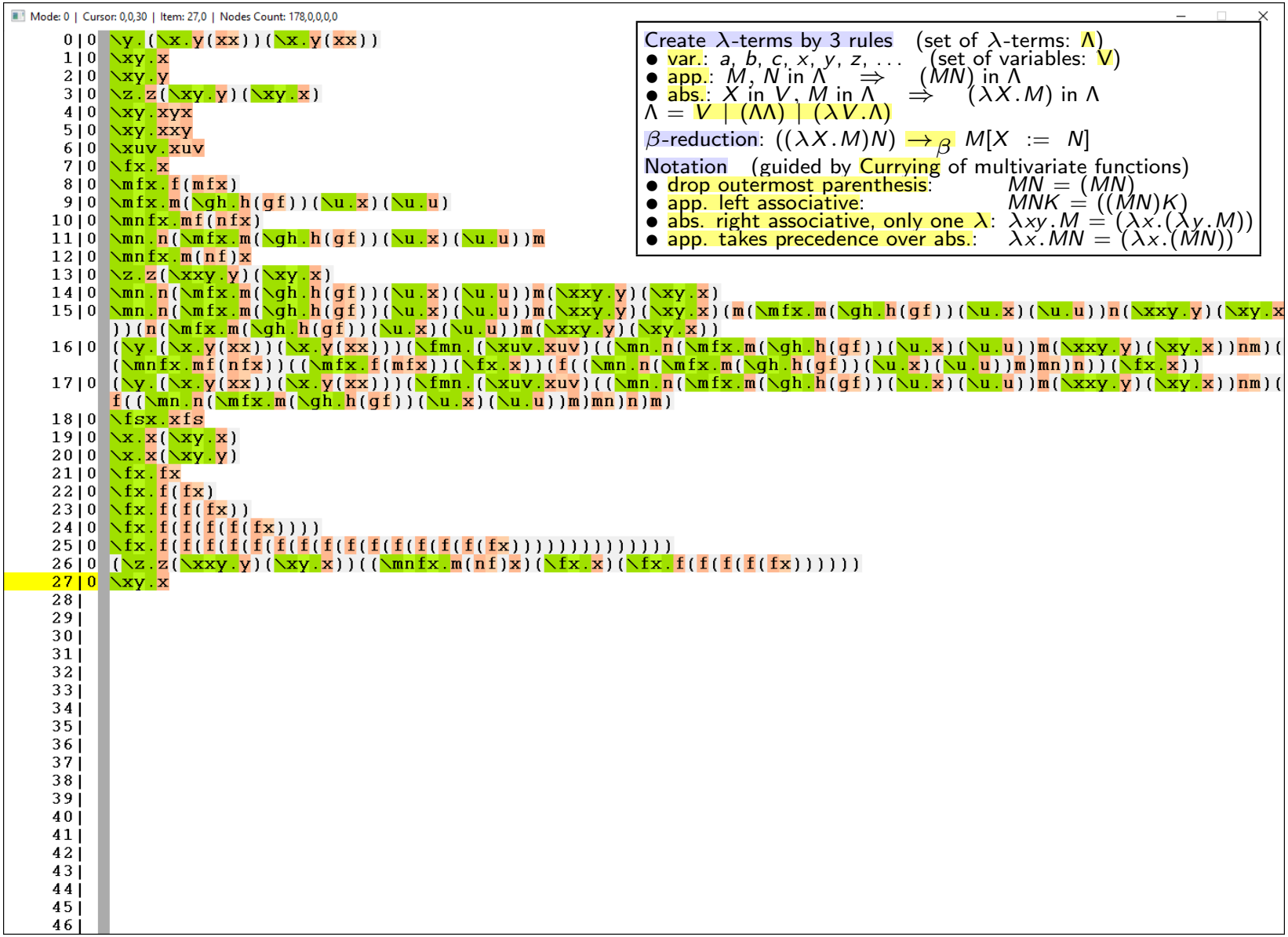
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator   0|0  \y.(\x.y(xx))(\x.y(xx))
true           1|0  \xy.x
false          2|0  \xy.y
not            3|0  \z.z(\xy.y)(\xy.x)
and            4|0  \xy.xyx
or             5|0  \xy.xxy
ifte           6|0  \xuv.xuv
zero           7|0  \fx.x
succ           8|0  \mfx.f(mfx)
pred           9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add           10|0  \mnfx.mf(nfx)
sub           11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul           12|0  \mnfx.m(nf)x
iszero        13|0  \z.z(\xxy.y)(\xy.x)
leq           14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq            15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                    ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div           16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod           17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair          18|0  \fsx.xfs
fst           19|0  \x.x(\xy.x)
sec           20|0  \x.x(\xy.y)
one           21|0  \fx.fx
two           22|0  \fx.f(fx)
three         23|0  \fx.f(f(fx))
five          24|0  \fx.f(f(f(f(fx))))
fifteen       25|0  \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))))
iszero (mul zero five)
              26|0  (\z.z(\xxy.y)(\xy.x))((\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx))))))
copy 26, βx5  27|0  (\fx.x)((\fx.f(f(f(f(fx)))))(\xxy.y))(\xy.x)
              28|
              29|
              30|
              31|
              32|
              33|
              34|
              35|
              36|
              37|
              38|
              39|
              40|
              41|
              42|
              43|
              44|
              45|
              46|
```

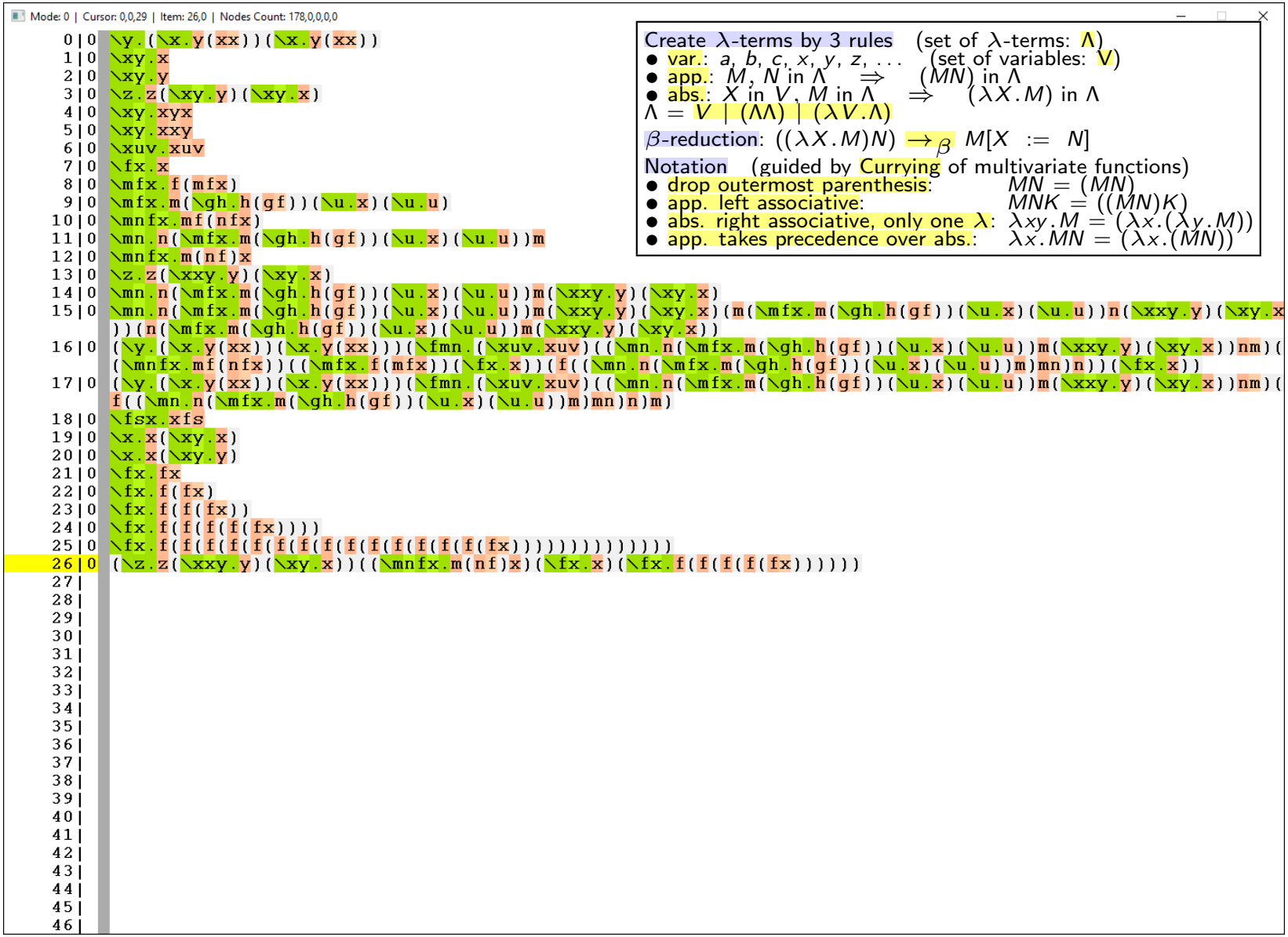
A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus



Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 179,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\xuv.xuv)((\z.z(\xxy.y)(\xy.x))((\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx)))))))
             27|
             28|     ifte (iszero (mul zero five))
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 180,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\xuv.xuv)((\z.z(\xxy.y)(\xy.x))((\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx)))))))(\fx.f(fx))
             27|
             28|     ifte (iszero (mul zero five)) two
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 181,0,0,0,0

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\xuv.xuv)((\z.z(\xxy.y)(\xy.x))((\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx)))))))(\fx.f(fx))(\fx.fx)
             27|
             28|     ifte (iszero (mul zero five)) two one
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 180,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\z.z(\xxy.y)(\xy.x))((\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx))))))(\fx.f(fx))(\fx.fx)
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

ifte (iszero (mul zero five)) two one, $\beta x3$

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 181,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
fifteen	25 | 0	\fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx))))))))))))))
	26 | 0	(\mnfx.m(nf)x)(\fx.x)(\fx.f(f(f(f(fx)))))(\xxy.y)(\xy.x)(\fx.f(fx))(\fx.fx)
	27 |	
	28 |	ifte (iszero (mul zero five)) two one, $\beta x4$
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 185,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
fifteen	25 | 0	\fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx))))))))))))))
	26 | 0	(\nfx.(\fx.x)(nf)x)(\fx.f(f(f(f(fx)))))(\xxy.y)(\xy.x)(\fx.f(fx))(\fx.fx)
	27 |	
	28 |	ifte (iszero (mul zero five)) two one, $\beta x5$
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 184,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\fx.(\fx.x)((\fx.f(f(f(f(fx)))))f)x)(\xxy.y)(\xy.x)(\fx.f(fx))(\fx.fx)
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

ifte (iszero (mul zero five)) two one, $\beta x6$

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 182,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))))
             26|0 (\x.(\fx.x)((\fx.f(f(f(f(fx)))))(\xxy.y))x)(\xy.x)(\fx.f(fx))(\fx.fx)
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

ifte (iszero (mul zero five)) two one, $\beta x7$

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 178,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\x.x)(\xy.x)(\fx.f(fx))(\fx.fx)
             27|
             28|     ifte (iszero (mul zero five)) two one, βx9
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 177,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
three        23|0  \fx.f(f(fx))
five         24|0  \fx.f(f(f(f(fx))))
fifteen      25|0  \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0  (\xy.x)(\fx.f(fx))(\fx.fx)
             27|
             28|      ifte (iszero (mul zero five)) two one, βx10
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 177,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
fifteen      25|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))))
             26|0 (\yfx.f(fx))(\fx.fx)
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

ifte (iszero (mul zero five)) two one, $\beta x11$

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 175,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
fifteen	25 | 0	\fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(fx))))))))))))))
	26 | 0	\fx.f(fx)
	27 |	
	28 |	ifte (iszero (mul zero five)) two one, $\beta x12$
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Mode: 0 | Cursor: 0,0,27 | Item: 24,0 | Nodes Count: 158,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
             25|
             26|
             27|
             28|
             29|
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

Mode: 0 | Cursor: 0,0,28 | Item: 25,0 | Nodes Count: 159,0,0,0,0

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
var.	25 | 0	f
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,29 | Item: 26,0 | Nodes Count: 160,0,0,0,0

Y-combinator	0 | 0	\y.(\x.y(xx))(\x.y(xx))
true	1 | 0	\xy.x
false	2 | 0	\xy.y
not	3 | 0	\z.z(\xy.y)(\xy.x)
and	4 | 0	\xy.xyx
or	5 | 0	\xy.xxy
ifte	6 | 0	\xuv.xuv
zero	7 | 0	\fx.x
succ	8 | 0	\mfx.f(mfx)
pred	9 | 0	\mfx.m(\gh.h(gf))(\u.x)(\u.u)
add	10 | 0	\mnfx.mf(nfx)
sub	11 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul	12 | 0	\mnfx.m(nf)x
iszero	13 | 0	\z.z(\xxy.y)(\xy.x)
leq	14 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq	15 | 0	\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div	16 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)((\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod	17 | 0	(\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair	18 | 0	\fsx.xfs
fst	19 | 0	\x.x(\xy.x)
sec	20 | 0	\x.x(\xy.y)
one	21 | 0	\fx.fx
two	22 | 0	\fx.f(fx)
three	23 | 0	\fx.f(f(fx))
five	24 | 0	\fx.f(f(f(f(fx))))
var.	25 | 0	f
var.	26 | 0	n
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	
	41 |	
	42 |	
	43 |	
	44 |	
	45 |	
	46 |	

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

Mode: 0 | Cursor: 0,0,30 | Item: 27,0 | Nodes Count: 161,0,0,0,0

```
Y-combinator   0|0  \y.(\x.y(xx))(\x.y(xx))
true           1|0  \xy.x
false          2|0  \xy.y
not            3|0  \z.z(\xy.y)(\xy.x)
and            4|0  \xy.xyx
or             5|0  \xy.xxy
ifte           6|0  \xuv.xuv
zero           7|0  \fx.x
succ           8|0  \mfx.f(mfx)
pred           9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add           10|0  \mnfx.mf(nfx)
sub           11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul           12|0  \mnfx.m(nf)x
iszero        13|0  \z.z(\xxy.y)(\xy.x)
leq           14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq            15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                    ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div           16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod           17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair          18|0  \fsx.xfs
fst           19|0  \x.x(\xy.x)
sec           20|0  \x.x(\xy.y)
one           21|0  \fx.fx
two           22|0  \fx.f(fx)
three         23|0  \fx.f(f(fx))
five          24|0  \fx.f(f(f(f(fx))))
var.          25|0  f
var.          26|0  n
iszero n      27|0  (\z.z(\xxy.y)(\xy.x))n
              28|
              29|
              30|
              31|
              32|
              33|
              34|
              35|
              36|
              37|
              38|
              39|
              40|
              41|
              42|
              43|
              44|
              45|
              46|
```

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in $\Lambda \Rightarrow (MN)$ in Λ
- abs.: X in V, M in $\Lambda \Rightarrow (\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,32 | Item: 29,0 | Nodes Count: 164,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
var.         25|0 f
var.         26|0 n
iszero n     27|0 (\z.z(\xxy.y)(\xy.x))n
f (pred n)   28|0 f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n        29|0 (\mnfx.m(nf)x)n
             30|
             31|
             32|
             33|
             34|
             35|
             36|
             37|
             38|
             39|
             40|
             41|
             42|
             43|
             44|
             45|
             46|
```

A crash course in λ-calculus

A crash course in λ-calculus

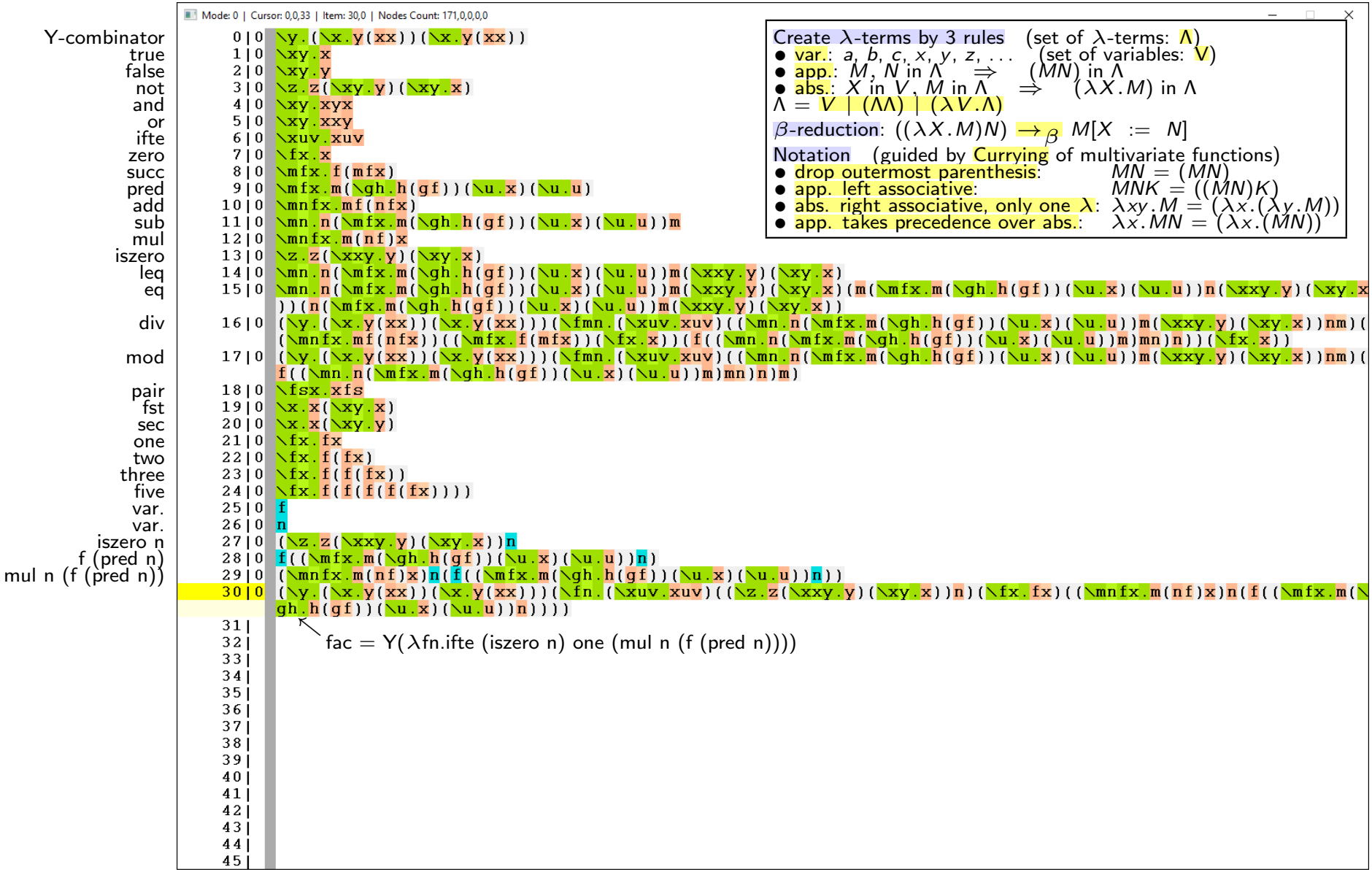

```
 0|0 \y.(\x.y(xx))(\x.y(xx))
 1|0 \xy.x
 2|0 \xy.y
 3|0 \z.z(\xy.y)(\xy.x)
 4|0 \xy.xyx
 5|0 \xy.xxy
 6|0 \xuv.xuv
 7|0 \fx.x
 8|0 \mfx.f(mfx)
 9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
10|0 \mnfx.mf(nfx)
11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
12|0 \mnfx.m(nf)x
13|0 \z.z(\xxy.y)(\xy.x)
14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
     ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
     (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
     f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
18|0 \fsx.xfs
19|0 \x.x(\xy.x)
20|0 \x.x(\xy.y)
21|0 \fx.fx
22|0 \fx.f(fx)
23|0 \fx.f(f(fx))
24|0 \fx.f(f(f(f(fx))))
25|0 f
26|0 n
27|0 (\z.z(\xxy.y)(\xy.x))n
28|0 f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
29|0 (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
30|0 (\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)
31|
32|
33|
34|
35|
36|
37|
38|
39|
40|
41|
42|
43|
44|
45|
46|
```


Mode: 0 | Cursor: 0,0,33 | Item: 30,0 | Nodes Count: 167,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
ifte (iszero n) one


Mode: 0 | Cursor: 0,0,33 | Item: 30,0 | Nodes Count: 168,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
ifte (iszero n) one (mul n (f (pred n)))

A crash course in λ-calculus

A crash course in λ-calculus


Mode: 0 | Cursor: 0,0,33 | Item: 30,0 | Nodes Count: 171,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac = Y(λfn.ifte (iszero n) one (mul n (f (pred n))))

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus


```
 0|0 \y.(\x.y(xx))(\x.y(xx))
 1|0 \xy.x
 2|0 \xy.y
 3|0 \z.z(\xy.y)(\xy.x)
 4|0 \xy.xyx
 5|0 \xy.xxy
 6|0 \xuv.xuv
 7|0 \fx.x
 8|0 \mfx.f(mfx)
 9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
10|0 \mnfx.mf(nfx)
11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
12|0 \mnfx.m(nf)x
13|0 \z.z(\xxy.y)(\xy.x)
14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
     ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
     (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
     f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
18|0 \fsx.xfs
19|0 \x.x(\xy.x)
20|0 \x.x(\xy.y)
21|0 \fx.fx
22|0 \fx.f(fx)
23|0 \fx.f(f(fx))
24|0 \fx.f(f(f(f(fx))))
25|0 f
26|0 n
27|0 (\z.z(\xxy.y)(\xy.x))n
28|0 f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
29|0 (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
30|0 (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
     gh.h(gf))(\u.x)(\u.u))n))))
31|0 (\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u)
     )n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf)))(
     \u.x)(\u.u))n))))(xx))(\fx.f(f(f(f(fx)))))
32|
33|
34|
35|
36|
37|
38|
39|
40|
41|
42|
43|
```

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus


Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 181,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx7

A crash course in λ-calculus

A crash course in λ-calculus

A crash course in λ-calculus

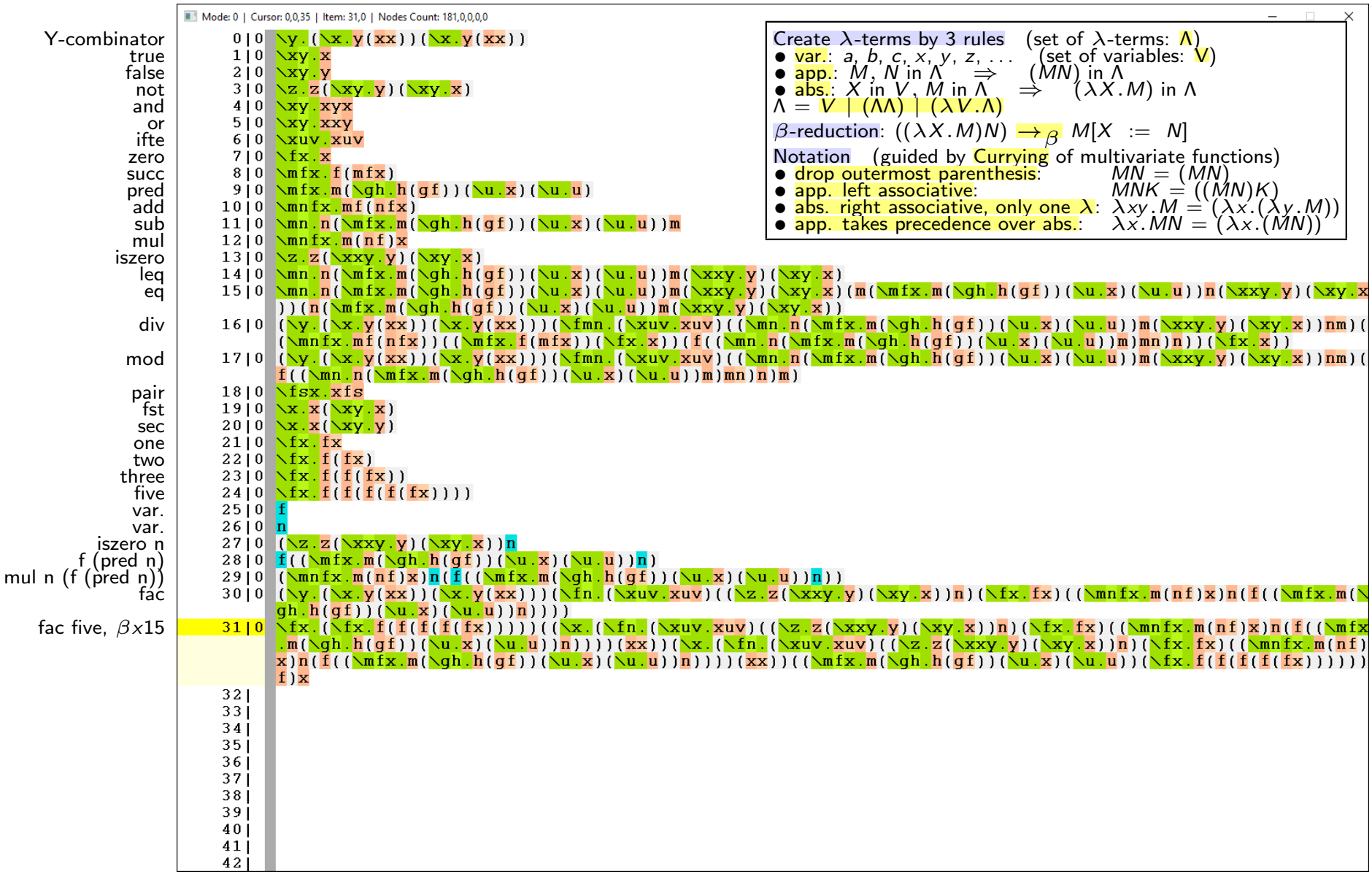
A crash course in λ-calculus


Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 179,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx12

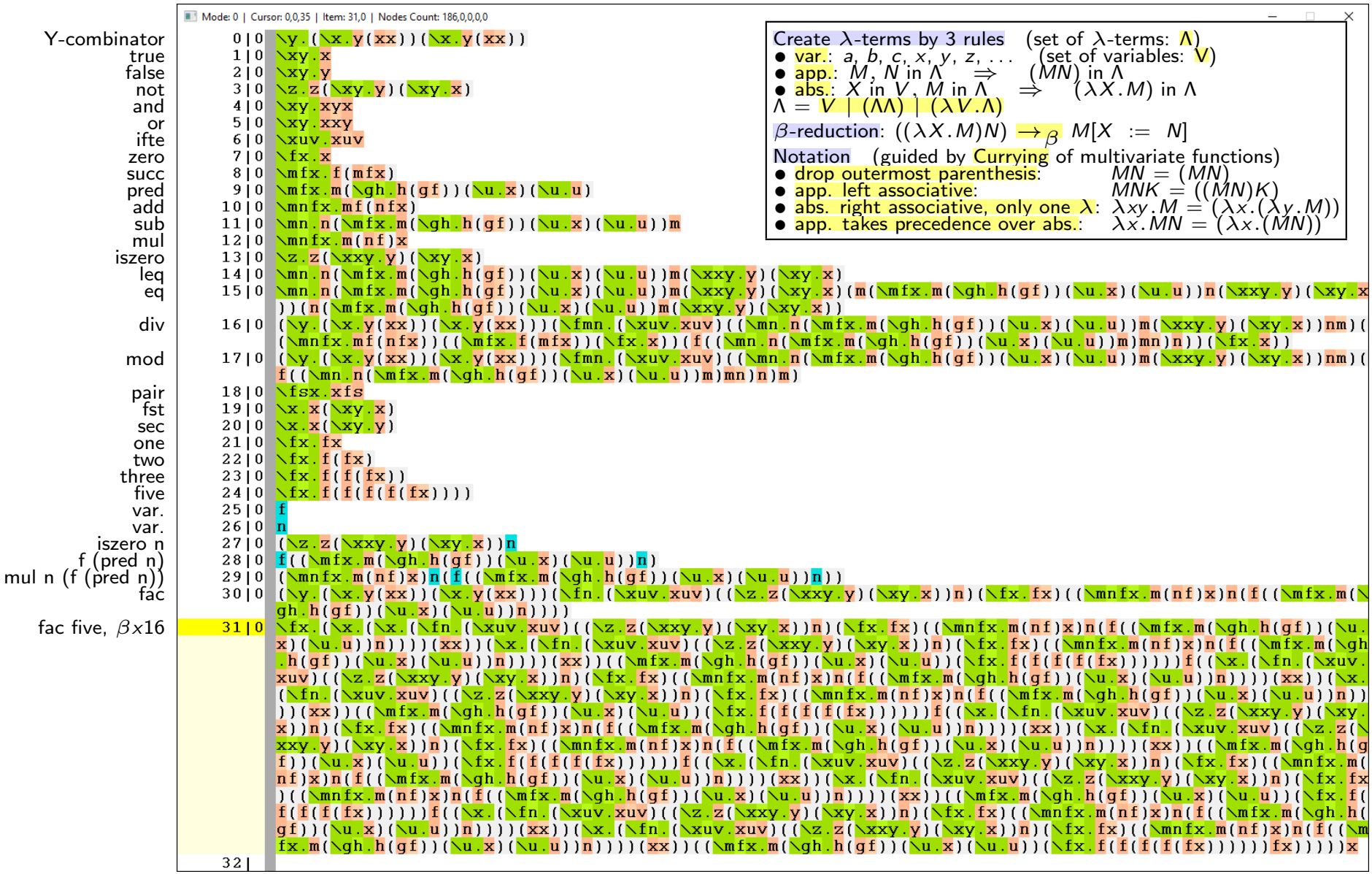
A crash course in λ-calculus

A crash course in λ-calculus

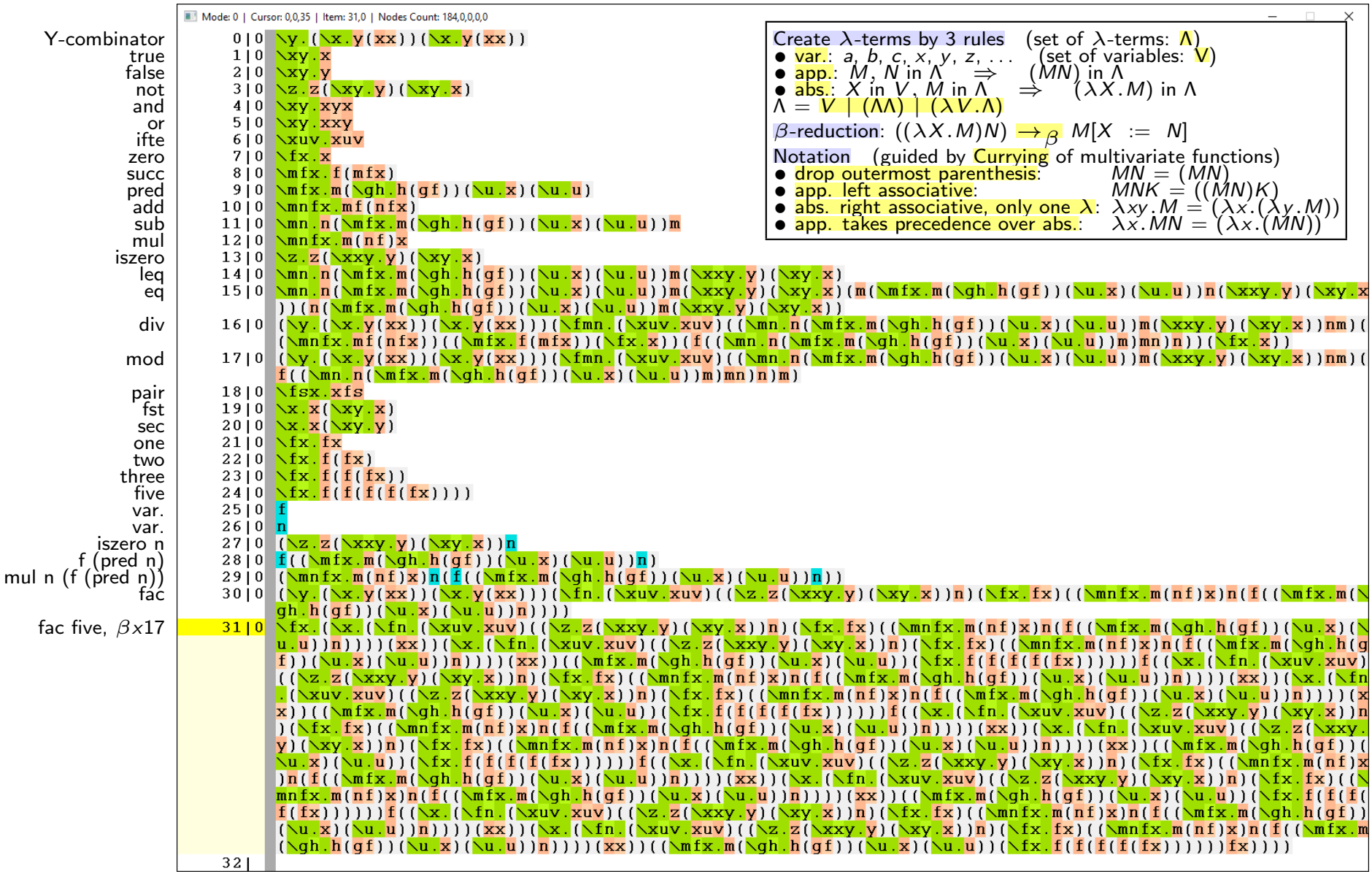
A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 188,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

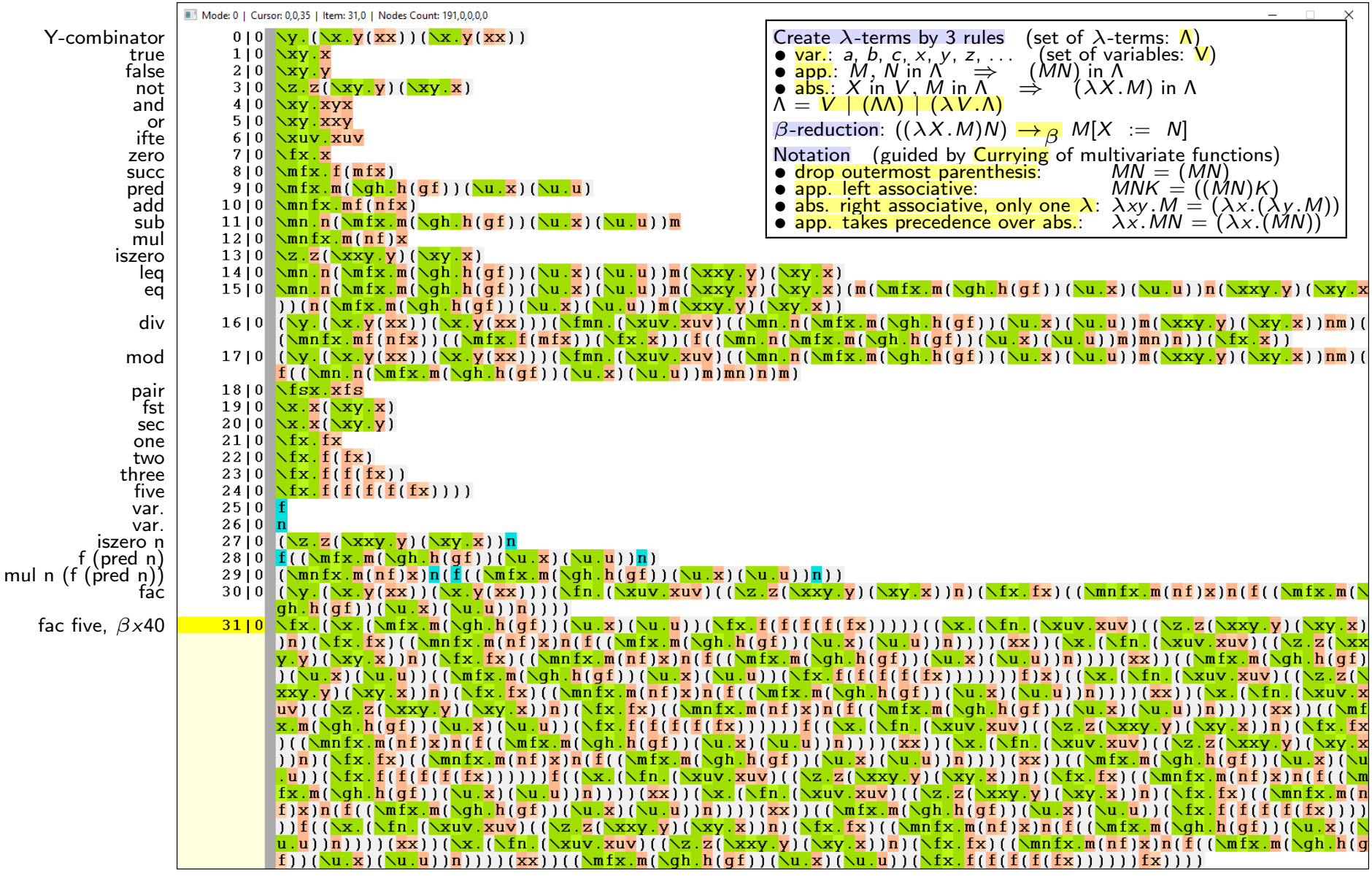
```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x))(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
var.         25|0 f
var.         26|0 n
iszero n     27|0 (\z.z(\xxy.y)(\xy.x))n
f (pred n)   28|0 f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n (f (pred n)) 29|0 (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
fac          30|0 (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
                  gh.h(gf))(\u.x)(\u.u))n))))
fac five, βx18 31|0 \fx.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u)
                  )n))))((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x
                  )(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.
                  h(gf))(\u.x)(\u.u))n))))(xx)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))f((\x.(\fn.(\xuv.
                  xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\x.
                  (\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))
                  ))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))f((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.
                  x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))f((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(
                  nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx
                  )((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(
                  f(f(f(fx))))))f((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(
                  gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\m
                  fx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))fx))))
```

A crash course in λ-calculus

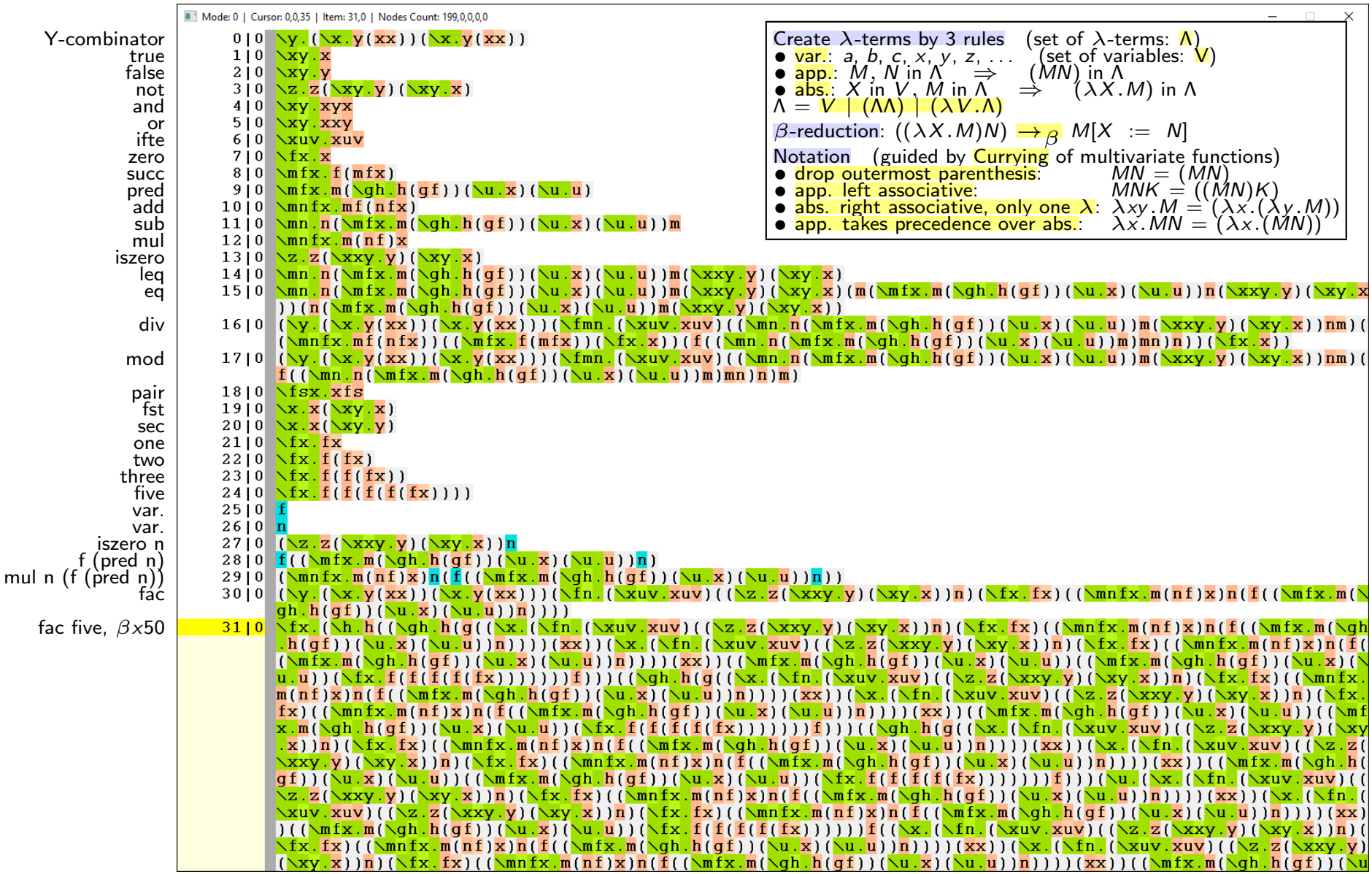

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 194,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx20

A crash course in λ-calculus

A crash course in λ-calculus



A crash course in λ-calculus



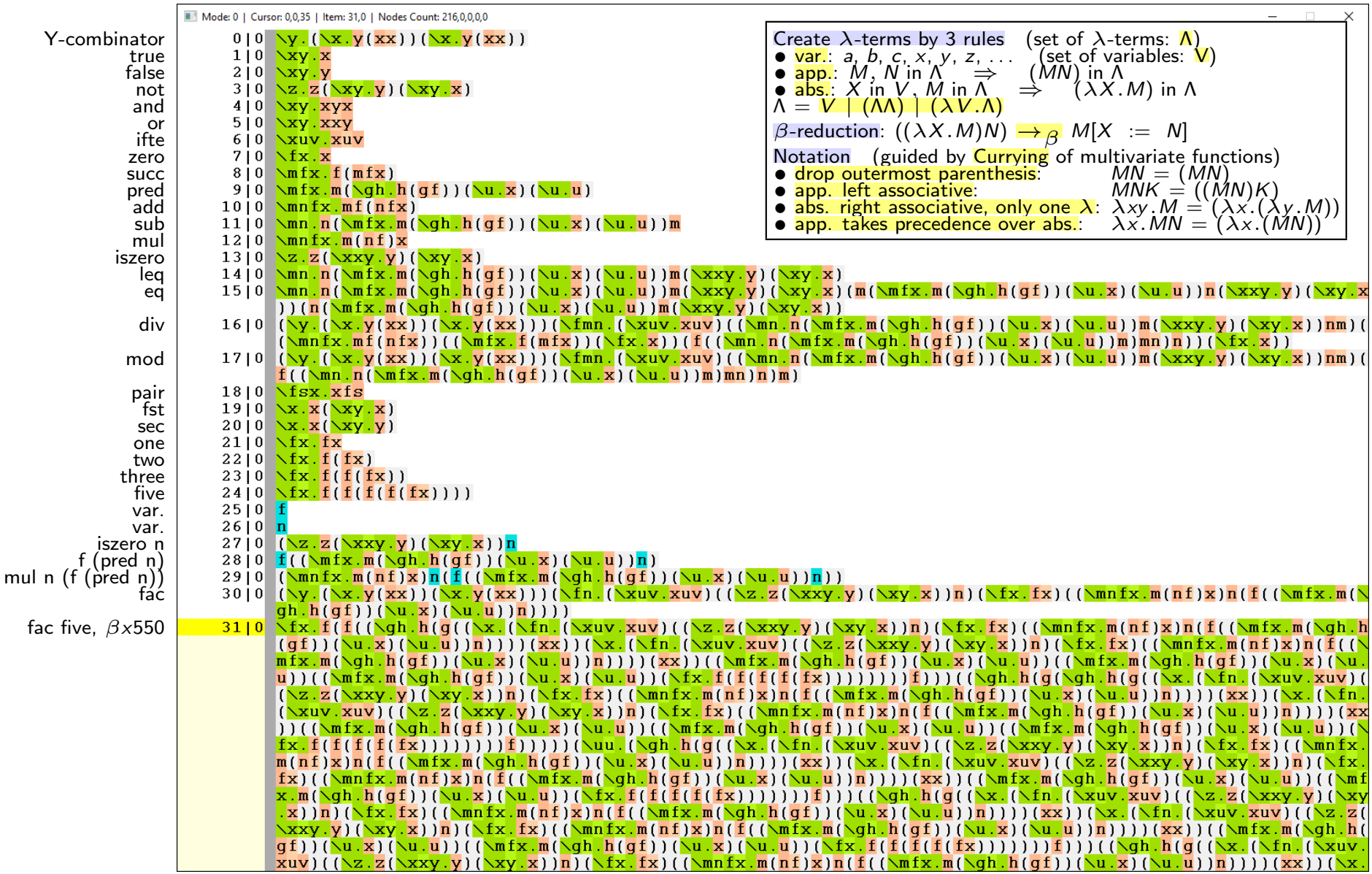
A crash course in λ-calculus

A crash course in λ-calculus

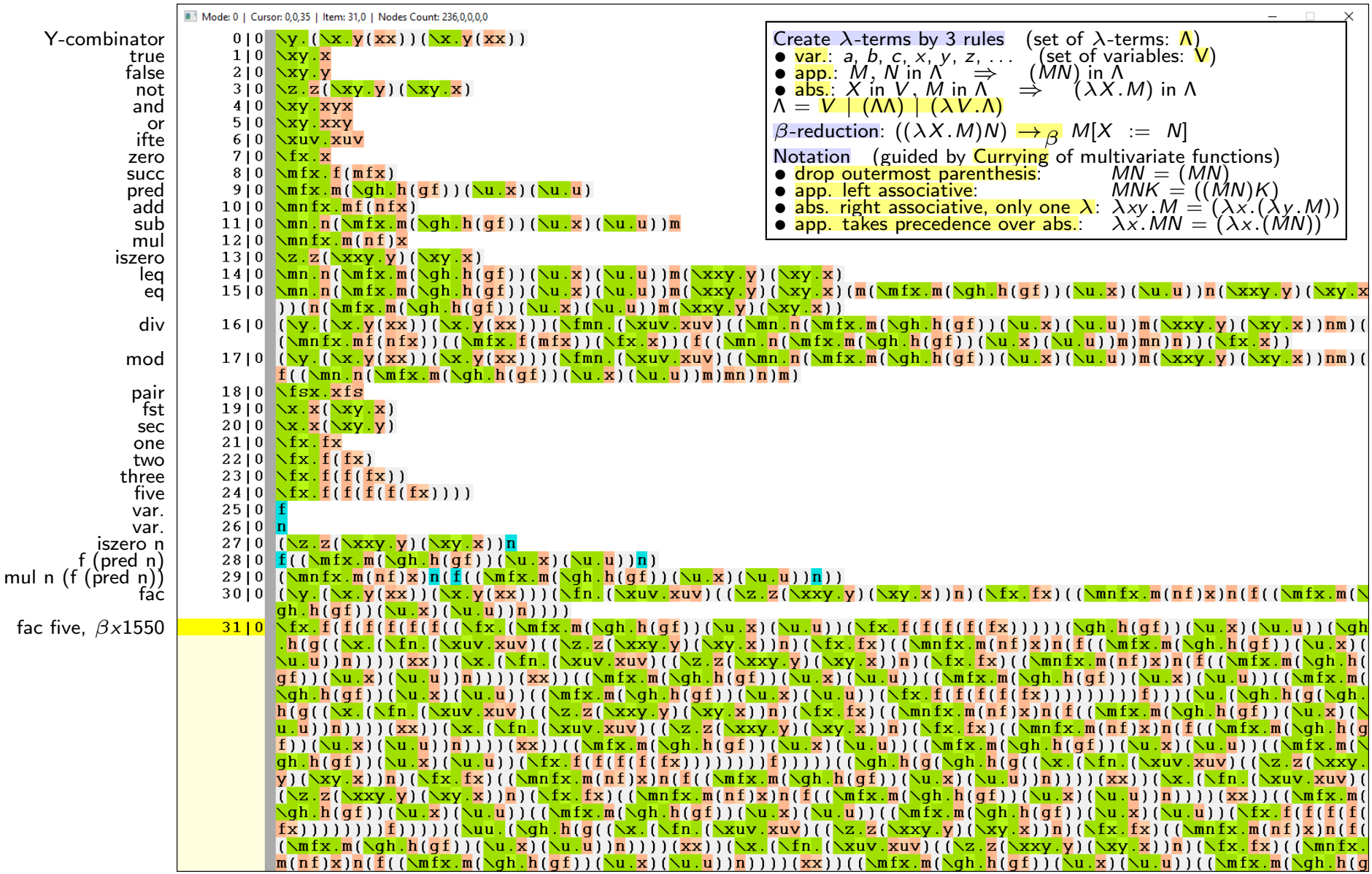
A crash course in λ-calculus

A crash course in λ-calculus

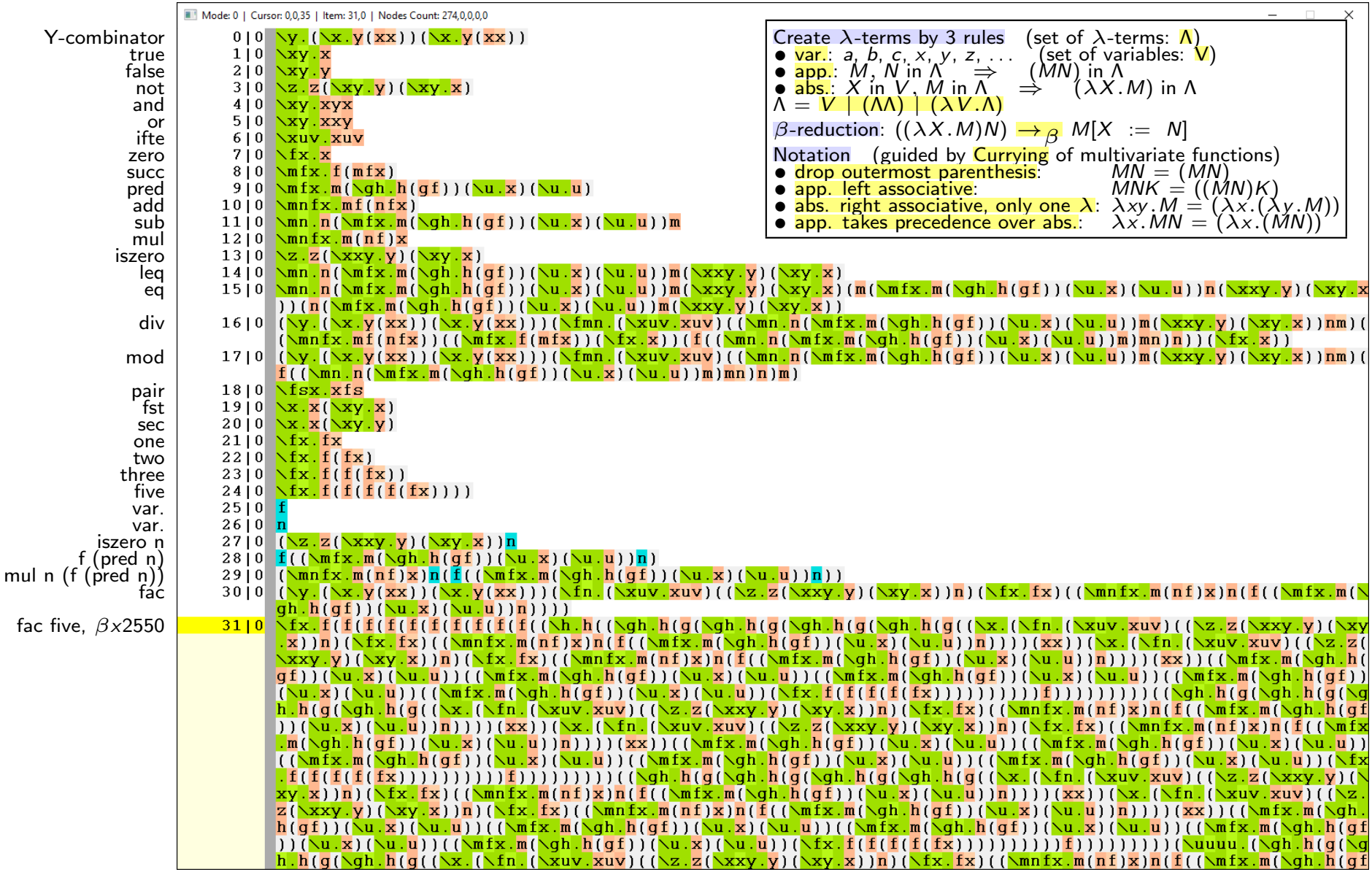
A crash course in λ-calculus



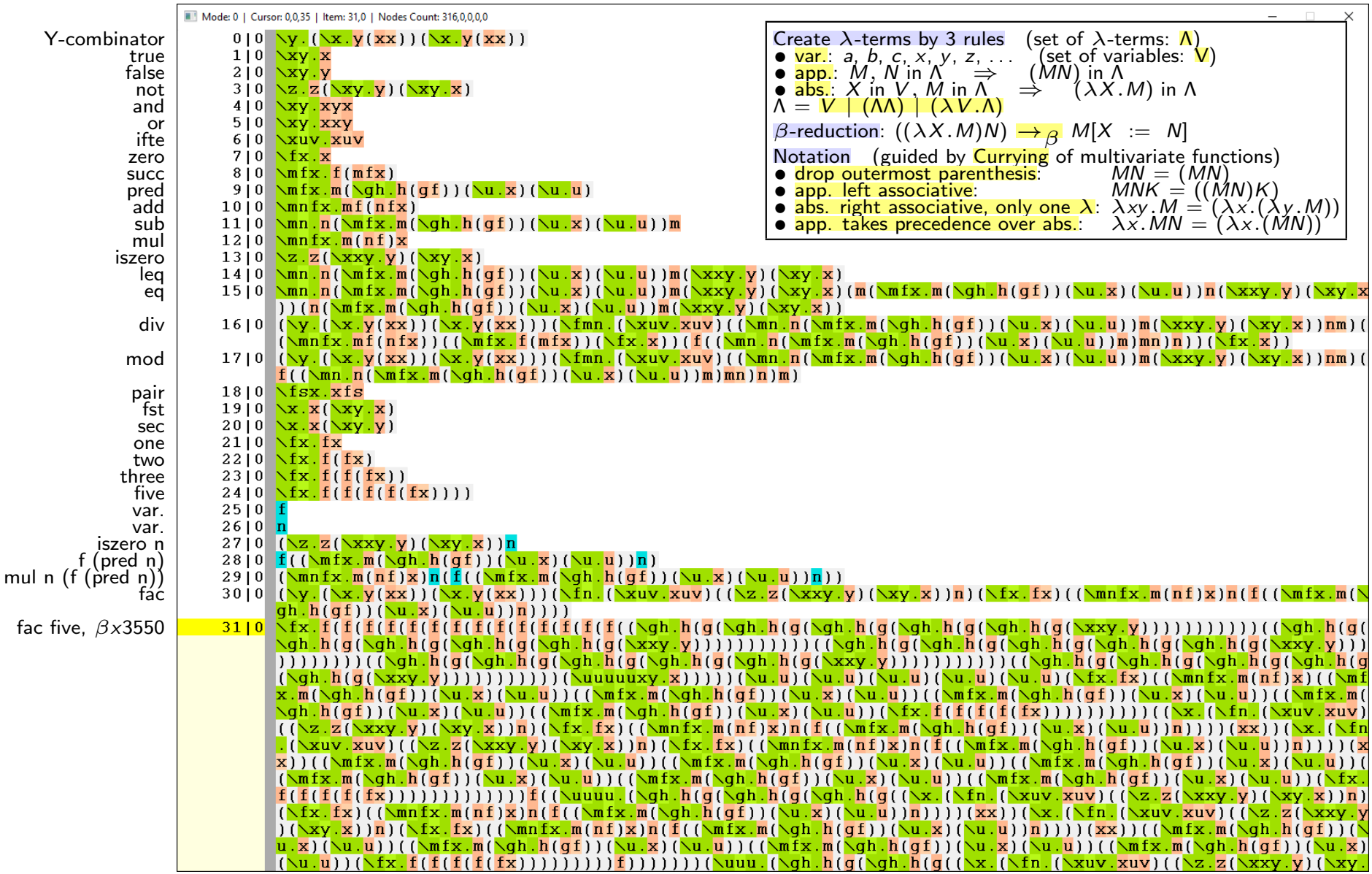
A crash course in λ-calculus



A crash course in λ-calculus

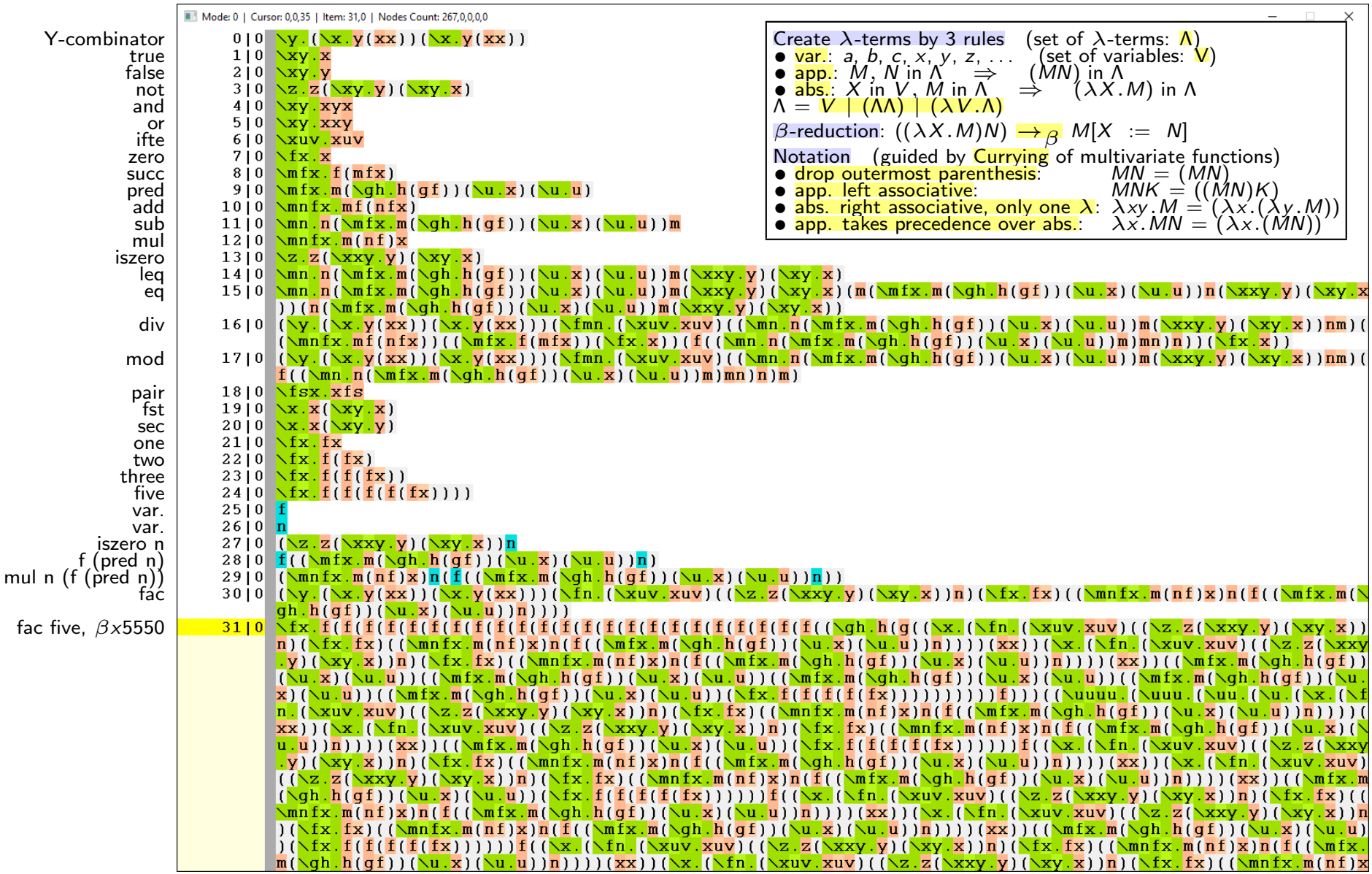


A crash course in λ-calculus

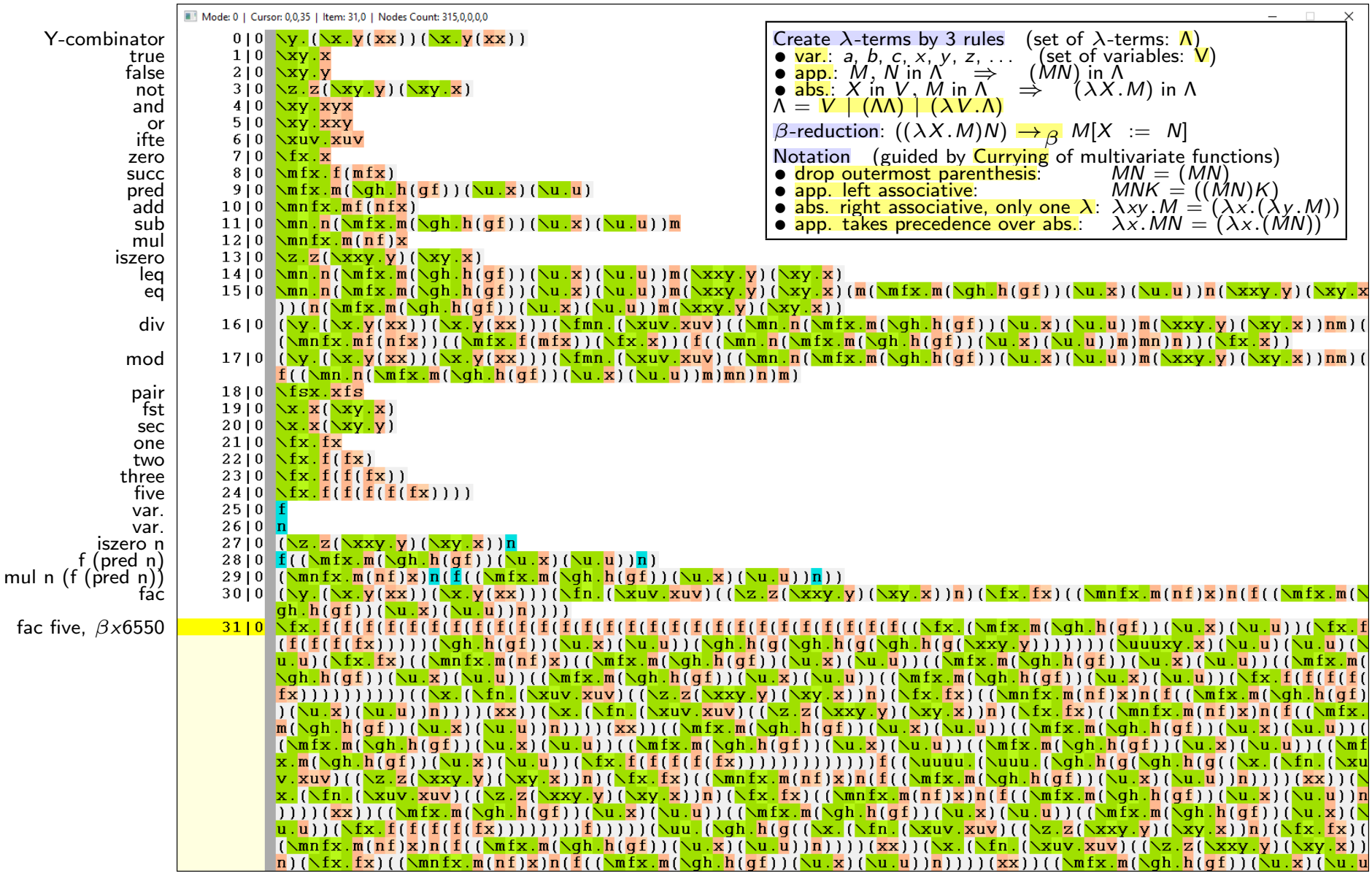


A crash course in λ-calculus

A crash course in λ-calculus

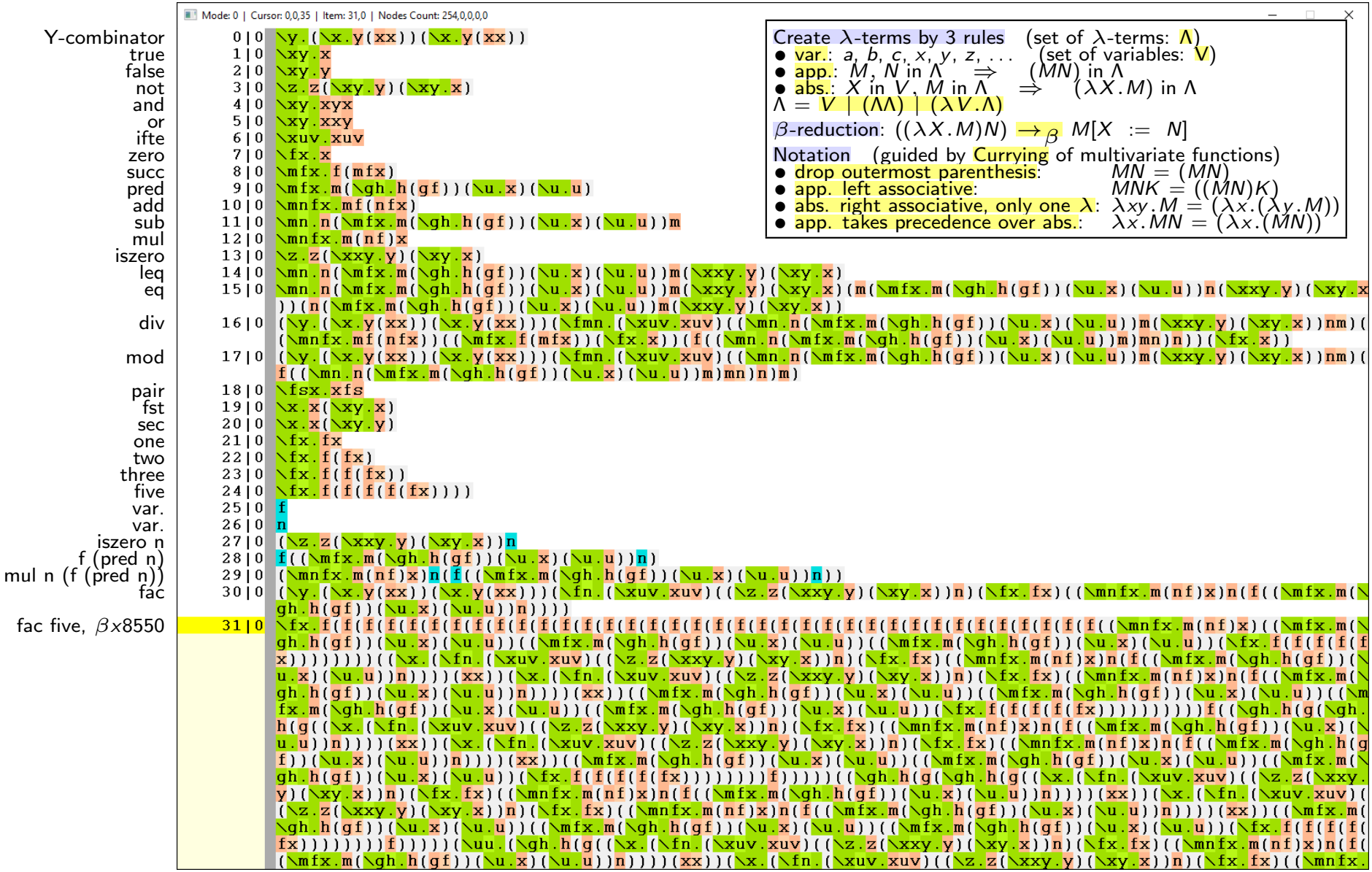


A crash course in λ-calculus

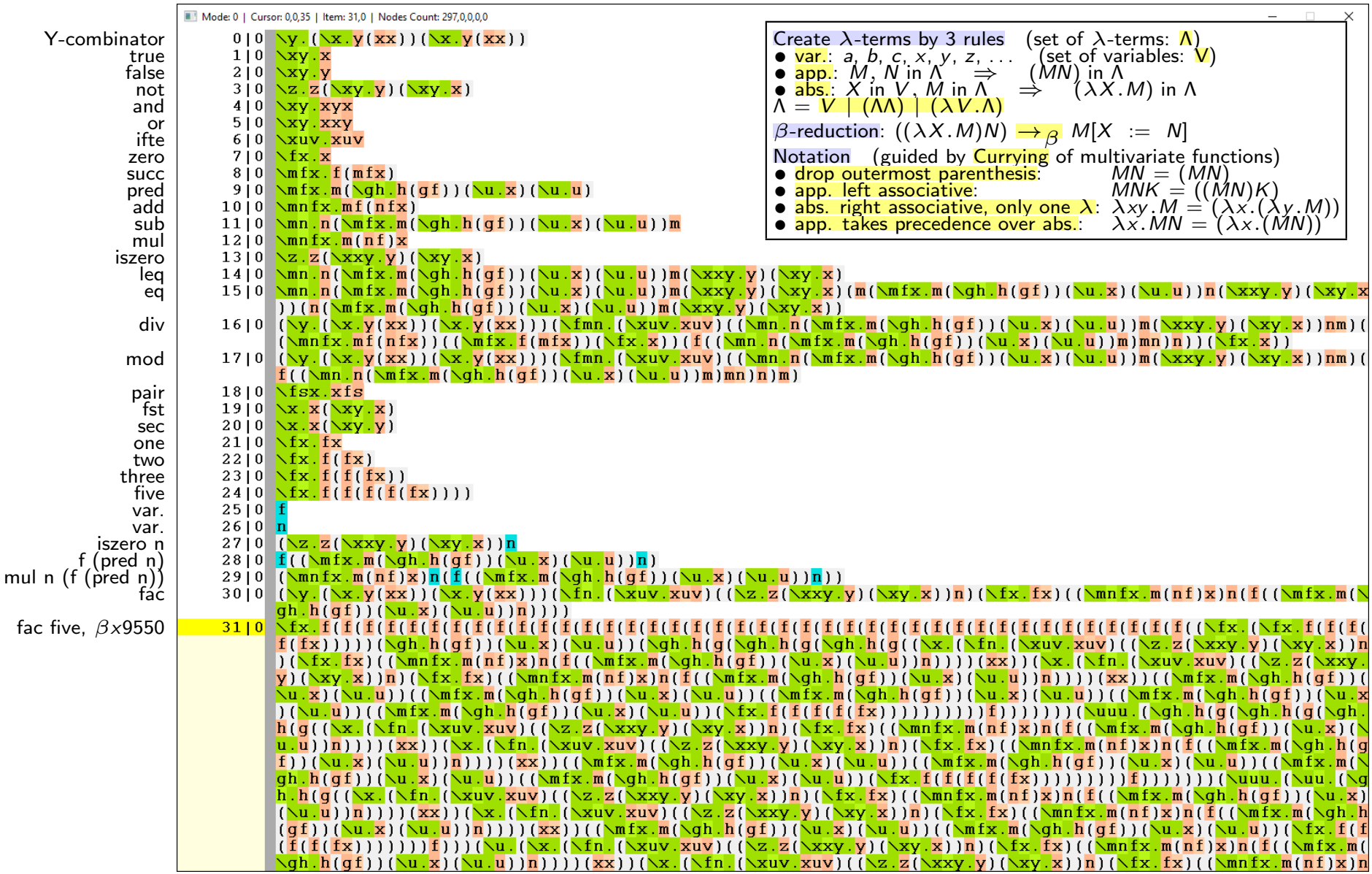


A crash course in λ-calculus

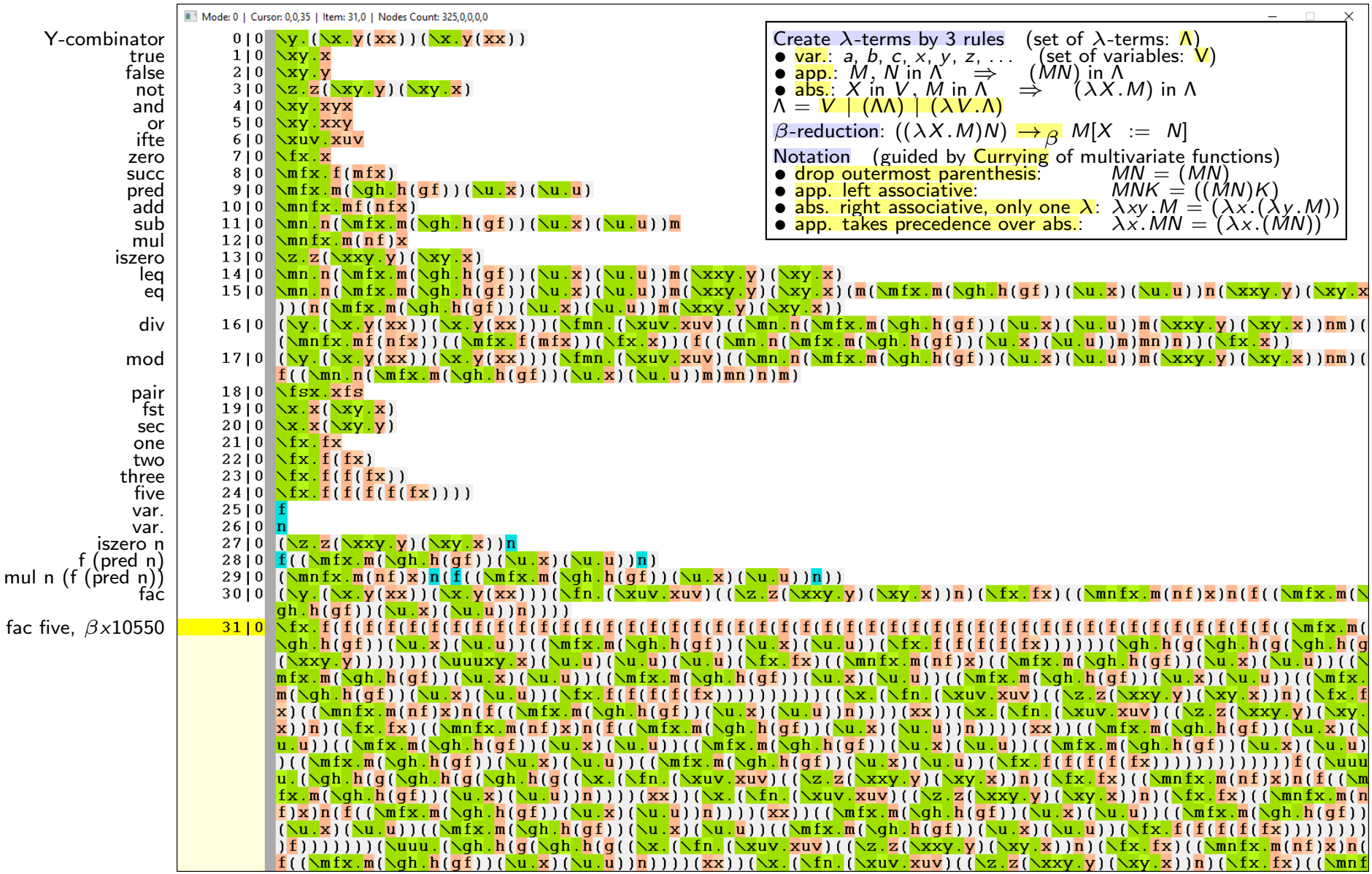
A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 351,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

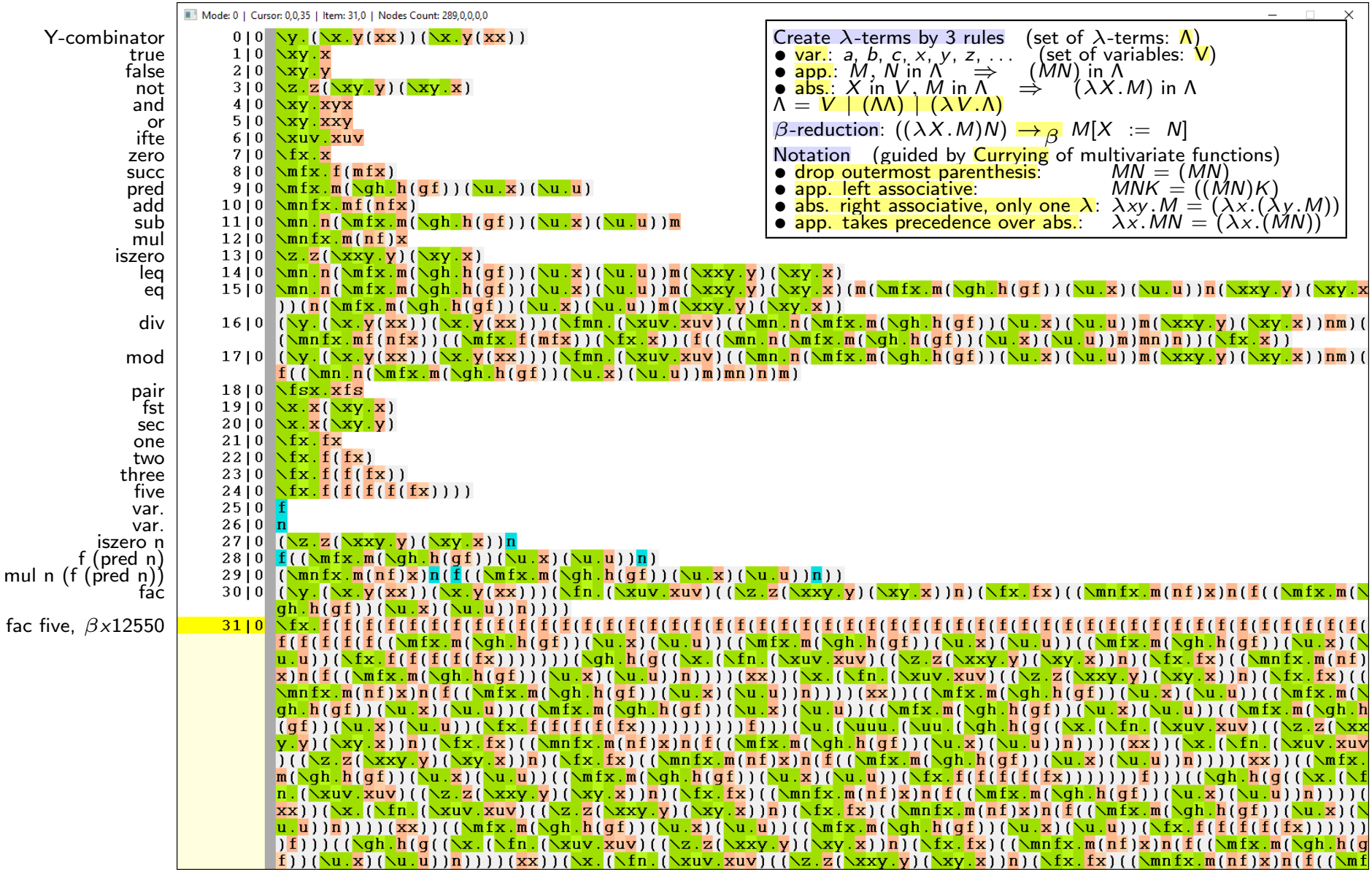
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

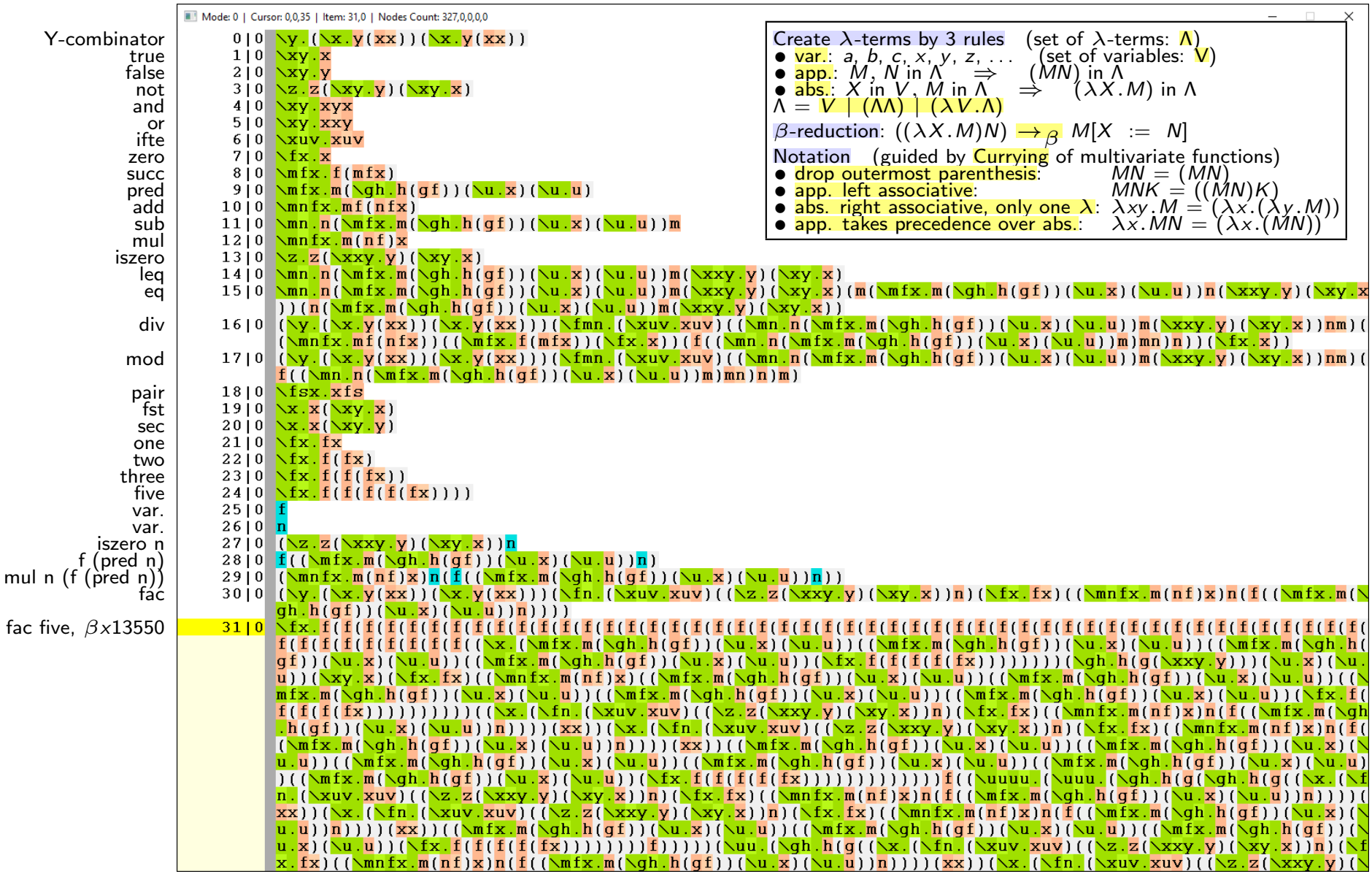
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0 \y.(\x.y(xx))(\x.y(xx))
true          1|0 \xy.x
false         2|0 \xy.y
not           3|0 \z.z(\xy.y)(\xy.x)
and           4|0 \xy.xyx
or            5|0 \xy.xxy
ifte          6|0 \xuv.xuv
zero          7|0 \fx.x
succ          8|0 \mfx.f(mfx)
pred          9|0 \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0 \mnfx.mf(nfx)
sub          11|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0 \mnfx.m(nf)x
iszero       13|0 \z.z(\xxy.y)(\xy.x)
leq          14|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0 \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                  ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0 (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                  f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0 \fsx.xfs
fst          19|0 \x.x(\xy.x)
sec          20|0 \x.x(\xy.y)
one          21|0 \fx.fx
two          22|0 \fx.f(fx)
three        23|0 \fx.f(f(fx))
five         24|0 \fx.f(f(f(f(fx))))
var.         25|0 f
var.         26|0 n
iszero n     27|0 (\z.z(\xxy.y)(\xy.x))n
f (pred n)   28|0 f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n (f (pred n)) 29|0 (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
fac          30|0 (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
                  gh.h(gf))(\u.x)(\u.u))n))))
fac five, βx11550 31|0 \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                  (\h.h((\gh.h(g(\gh.h(g(\gh.h(g(\gh.h(g(\xxy.y))))))))((\gh.h(g(\gh.h(g(\gh.h(g(\gh.h(g(\g
                  h.h(g(\xxy.y)))))))))(\uuuuuxy.x))(\gh.h(g(\gh.h(g(\gh.h(g(\gh.h(g(\xxy.y)))))))))(\gh.h(g(\gh.h(g
                  (\gh.h(g(\xxy.y)))))))))(\gh.h(g(\gh.h(g(\gh.h(g(\xxy.y)))))))(\u.u)(\u.u)(\u.u)(\fx.fx)((\mnfx.m(nf
                  )x)((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(
                  (\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))))))((\x.(\fn.(\
                  xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))
                  (\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u)
                  )n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)
                  (\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.
                  u))(\fx.f(f(f(f(fx))))))))))))f((\uuuu.(\gh.h(g(\gh.h(g(\gh.h(g((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\
                  xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.
                  z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.
                  h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf
                  ))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))))))f)))))))(\uuu.(\gh.h(g(\gh.h(g((\x.(\fn.(\xuv.xuv)((\z.z(\xxy
```

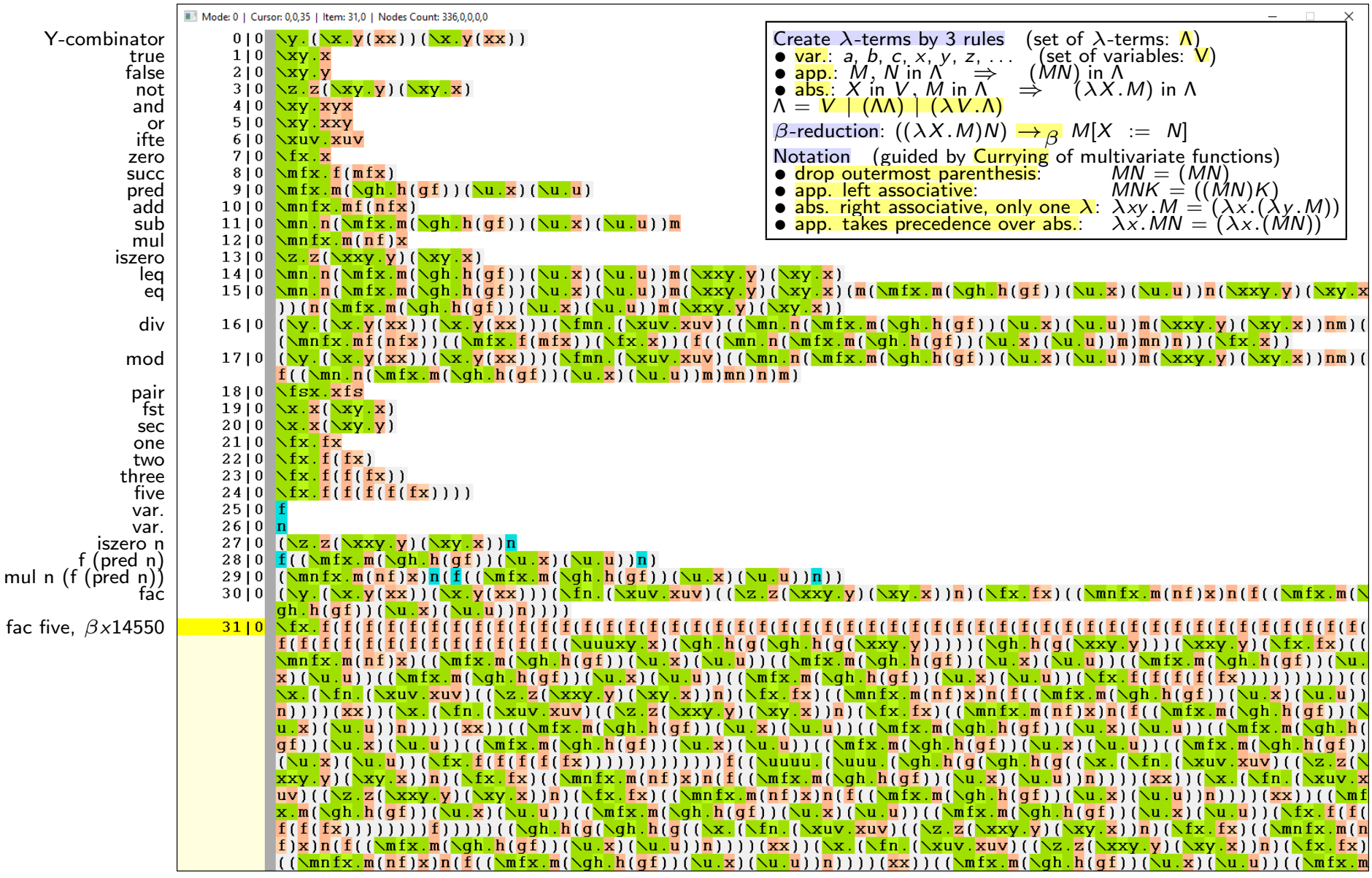
A crash course in λ-calculus



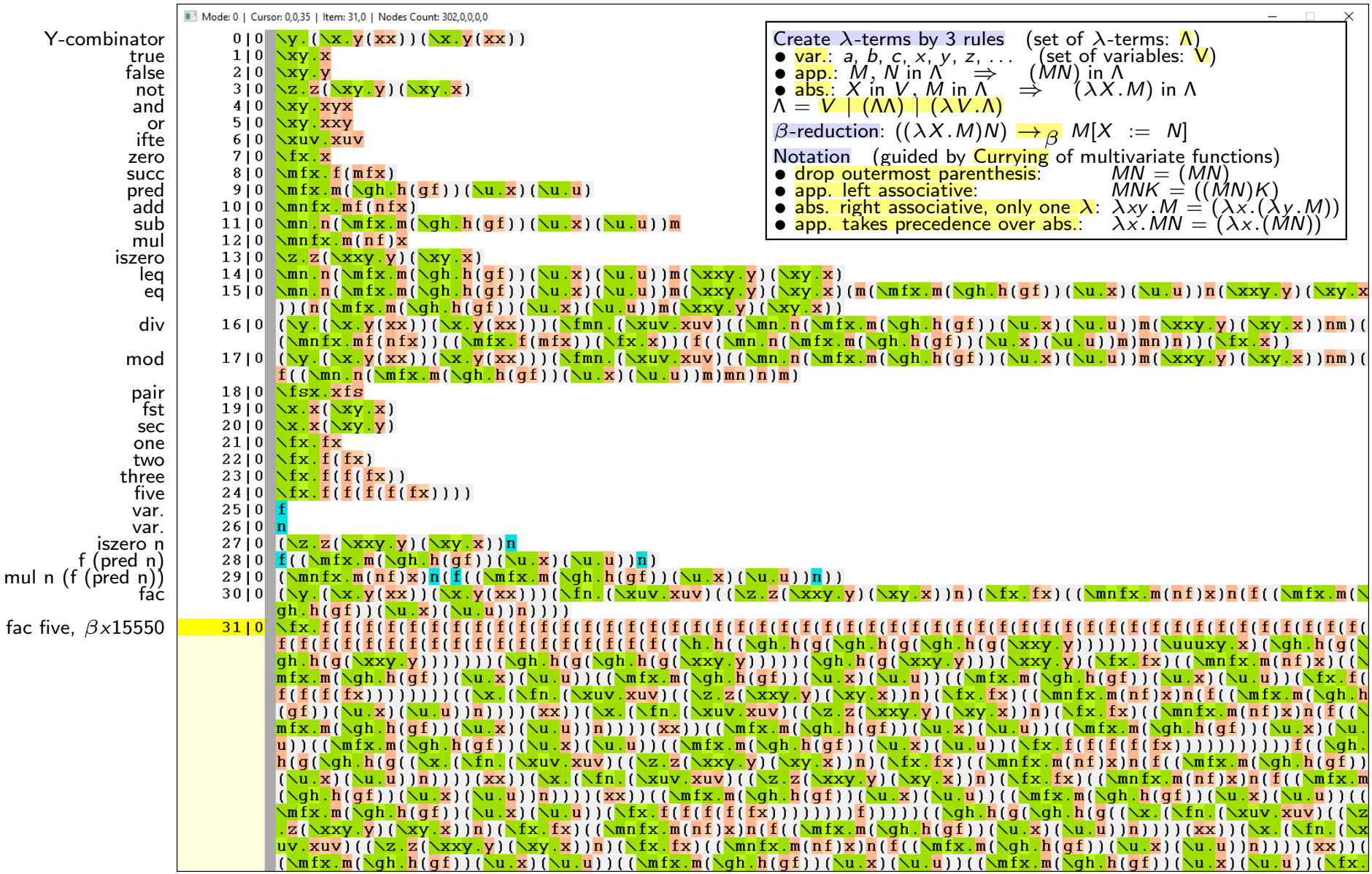
A crash course in λ-calculus



A crash course in λ-calculus

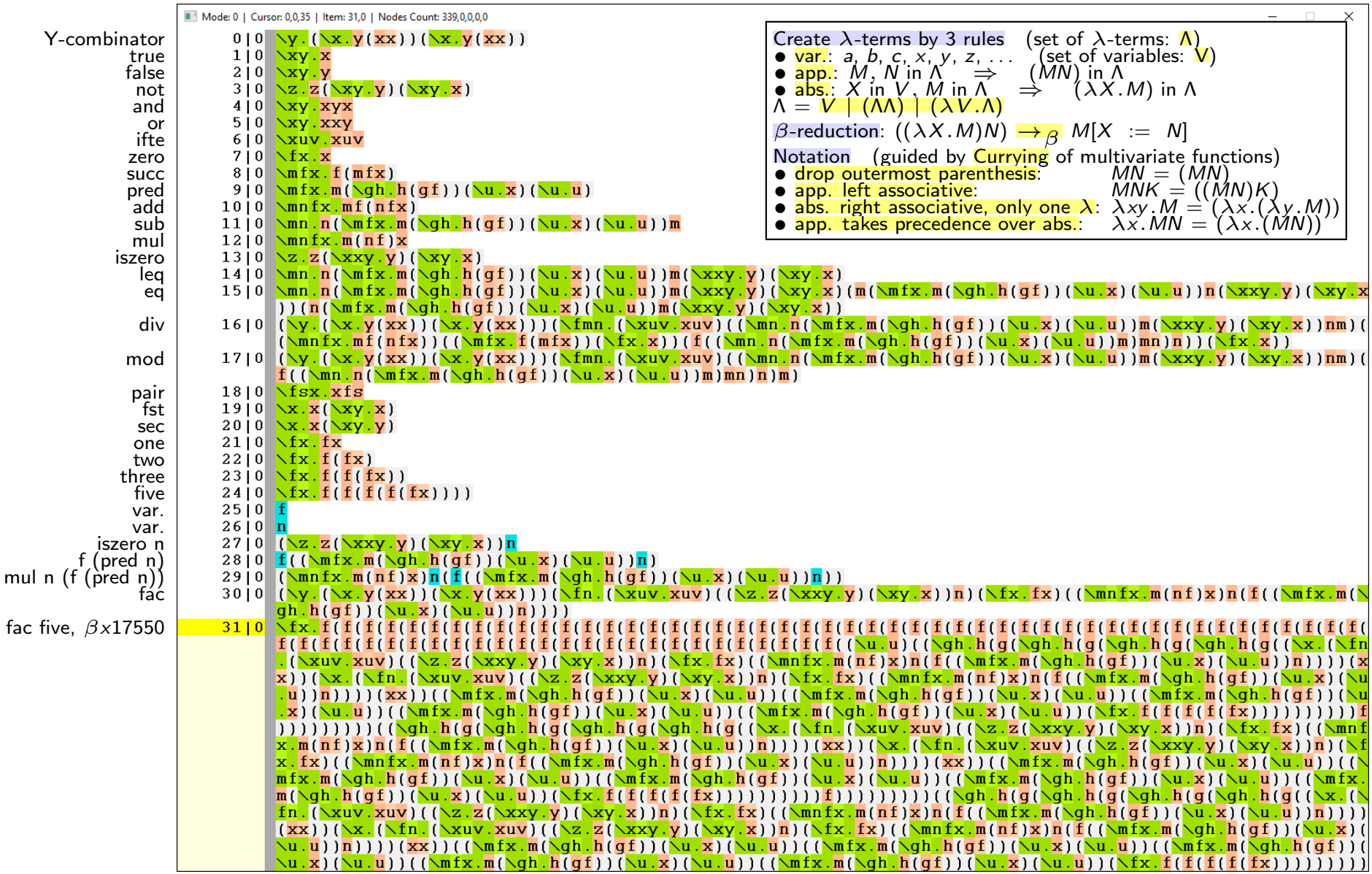


A crash course in λ-calculus

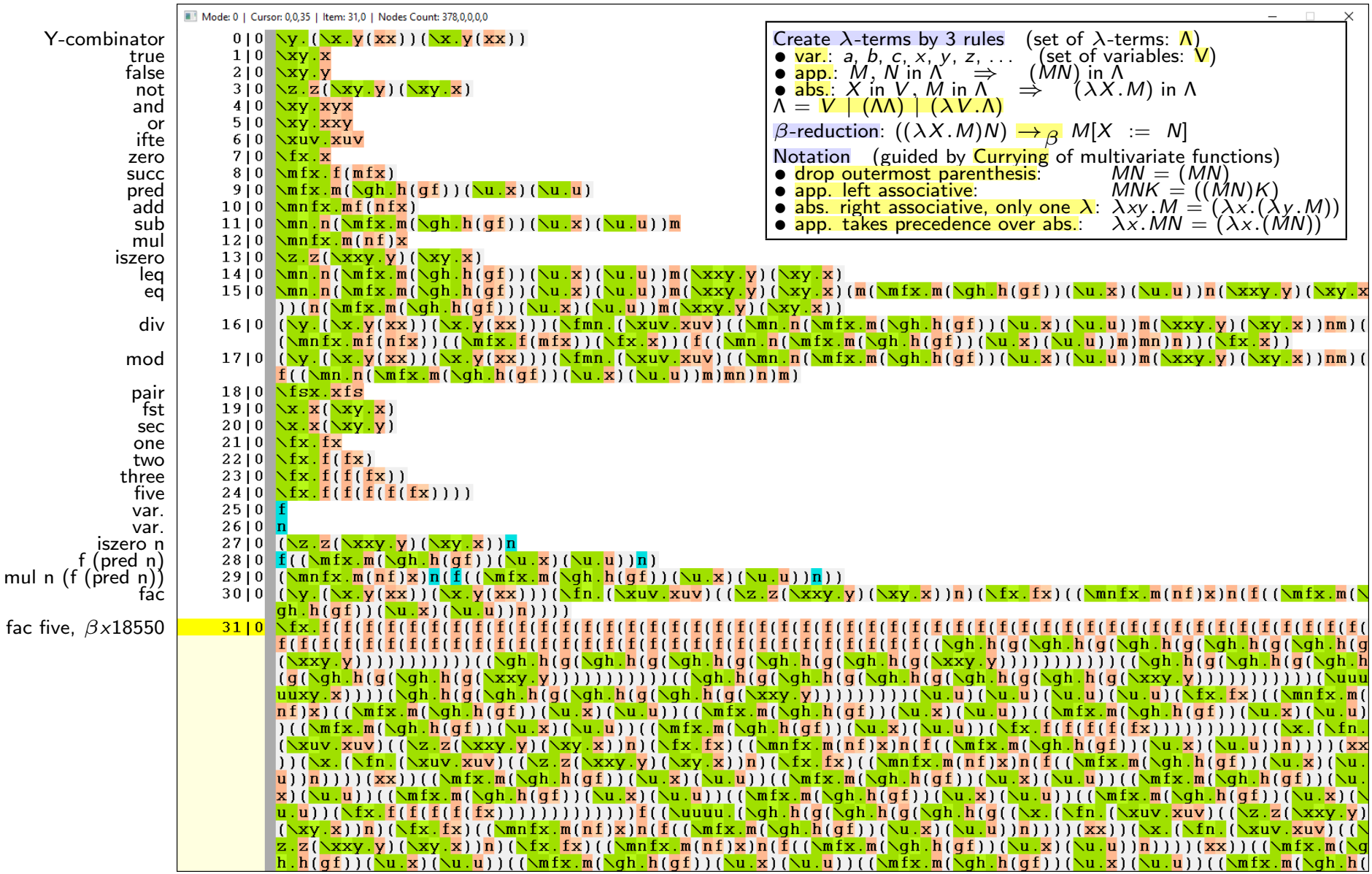


A crash course in λ-calculus

A crash course in λ-calculus



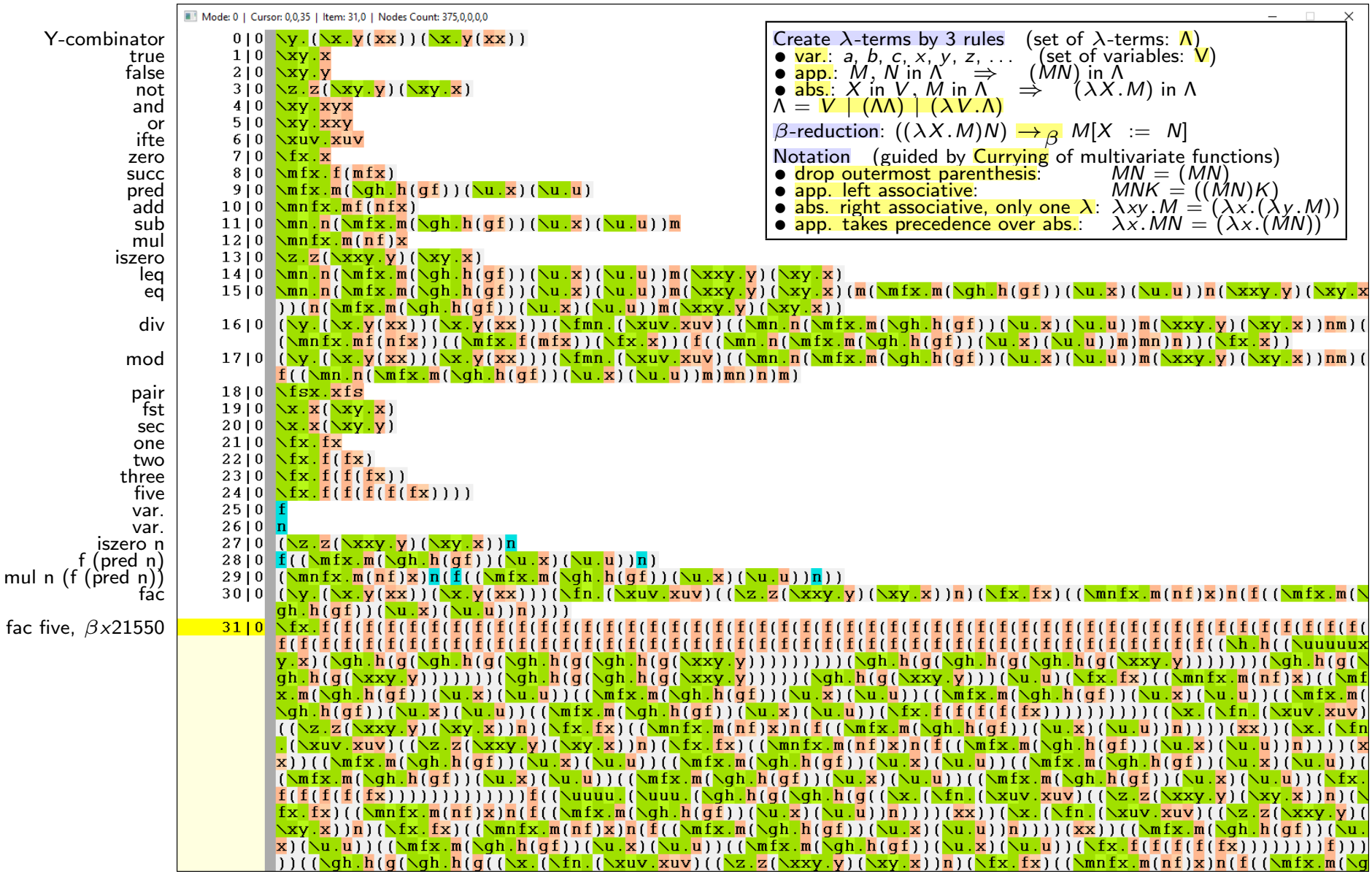
A crash course in λ-calculus



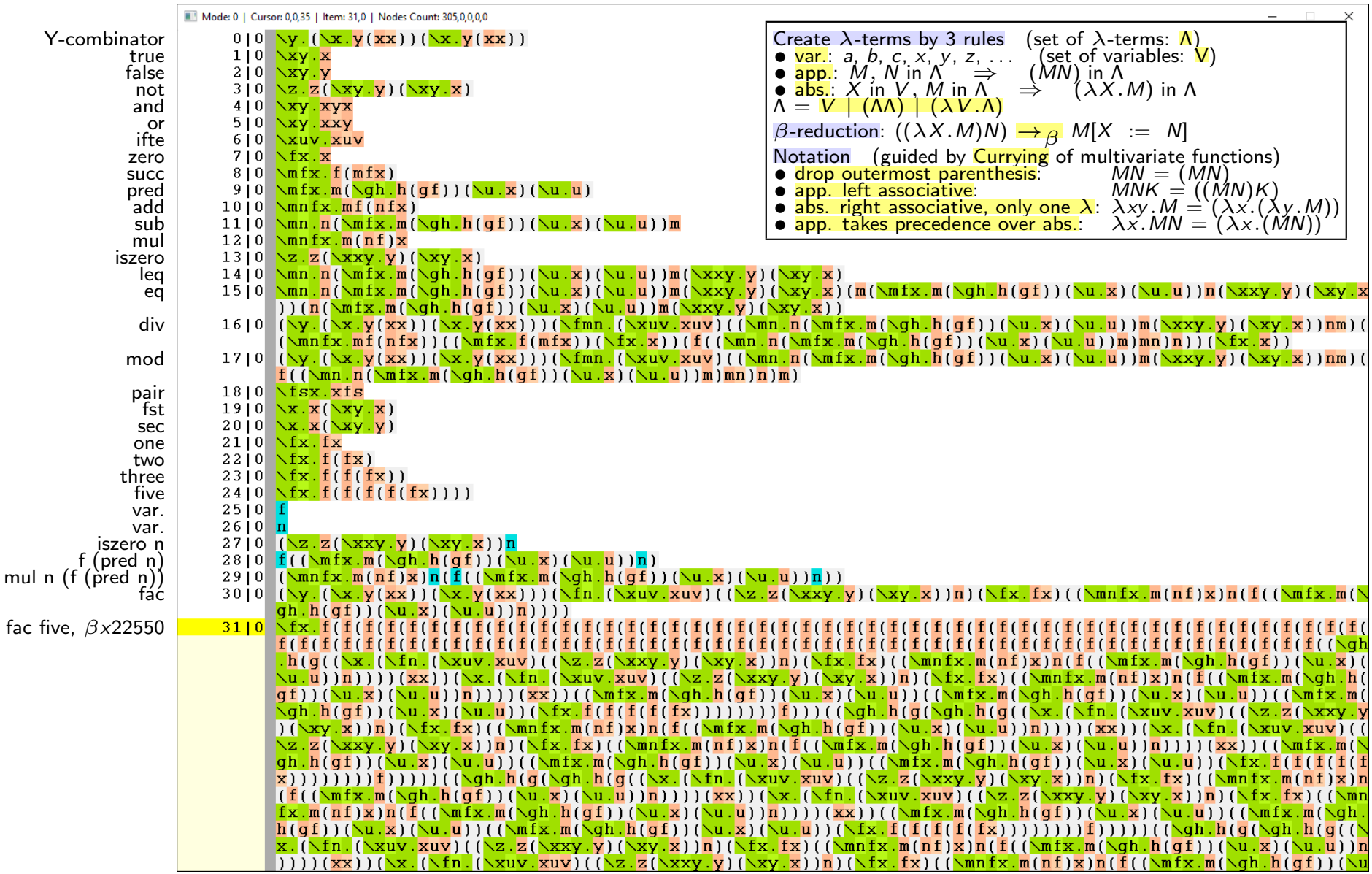
A crash course in λ-calculus

A crash course in λ-calculus

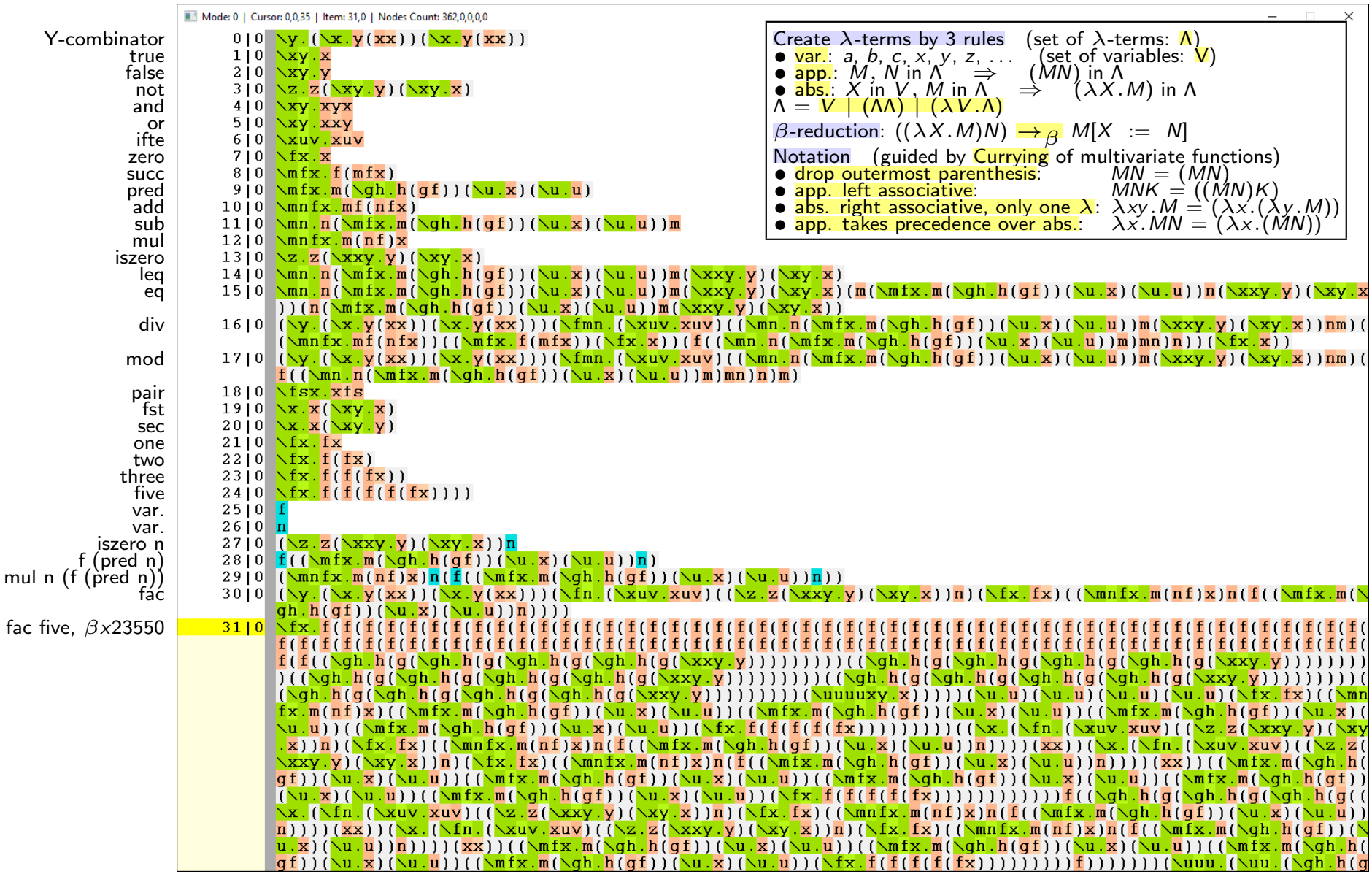
A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus

A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 395,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

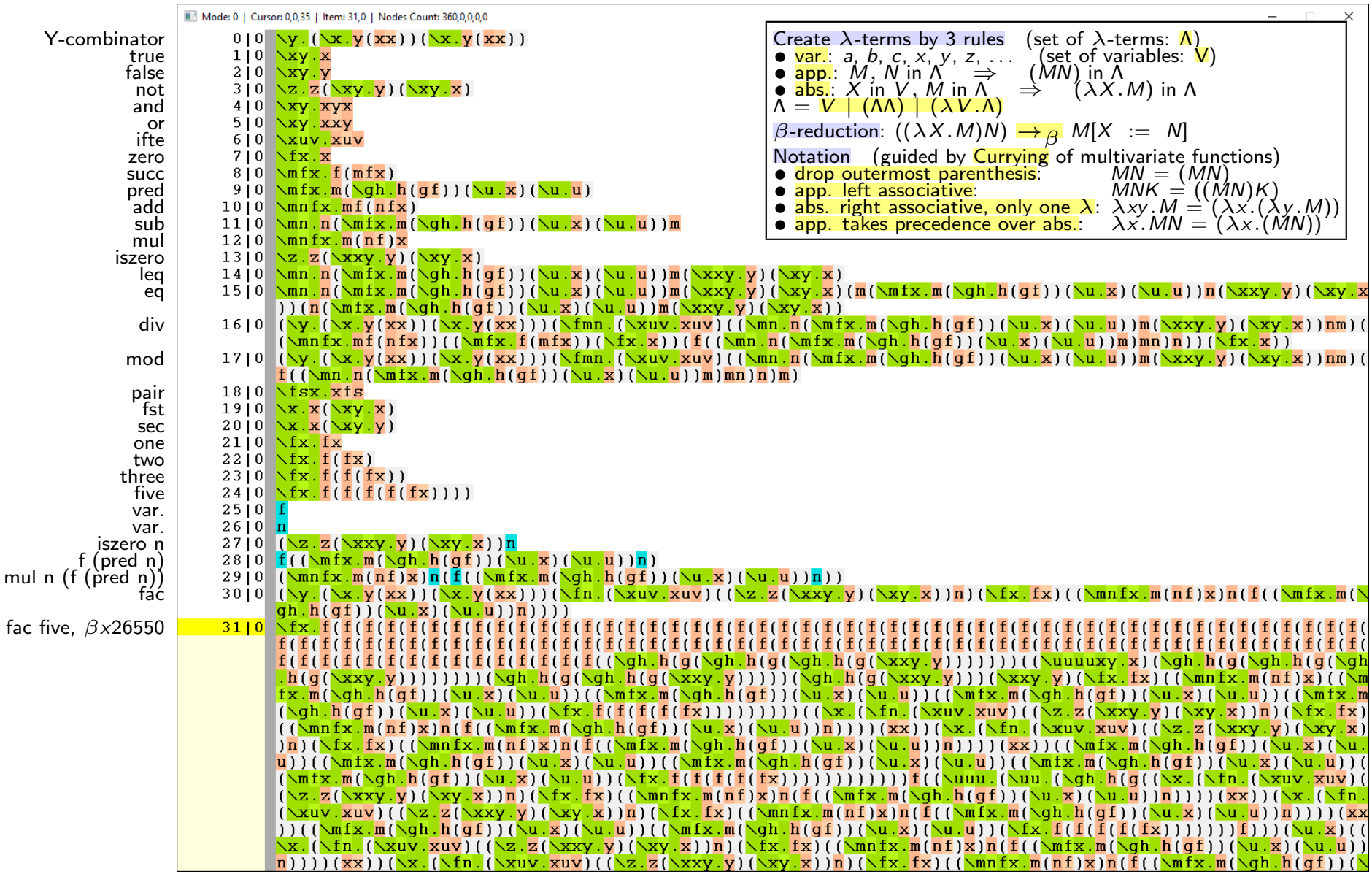
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

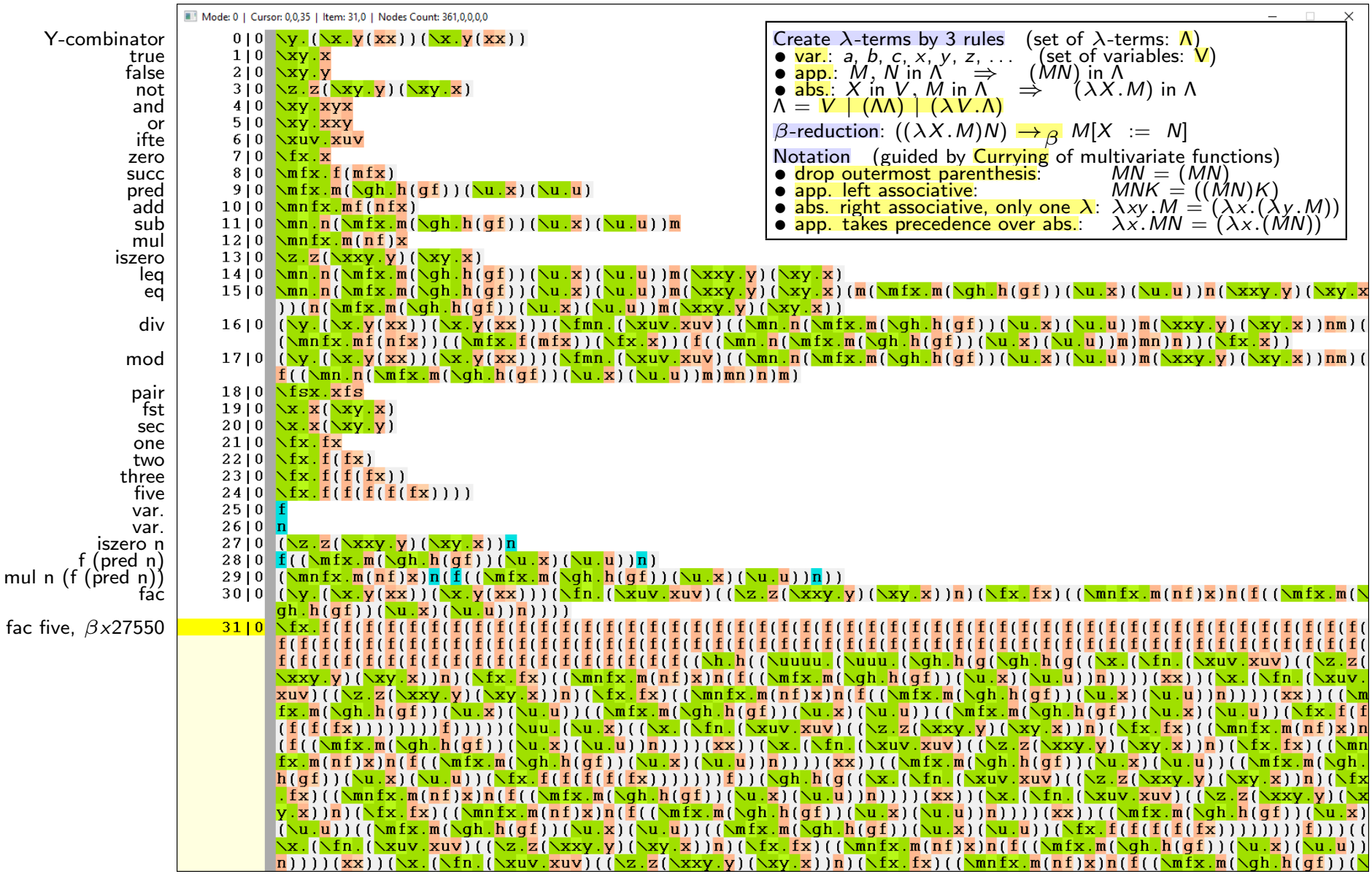
Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

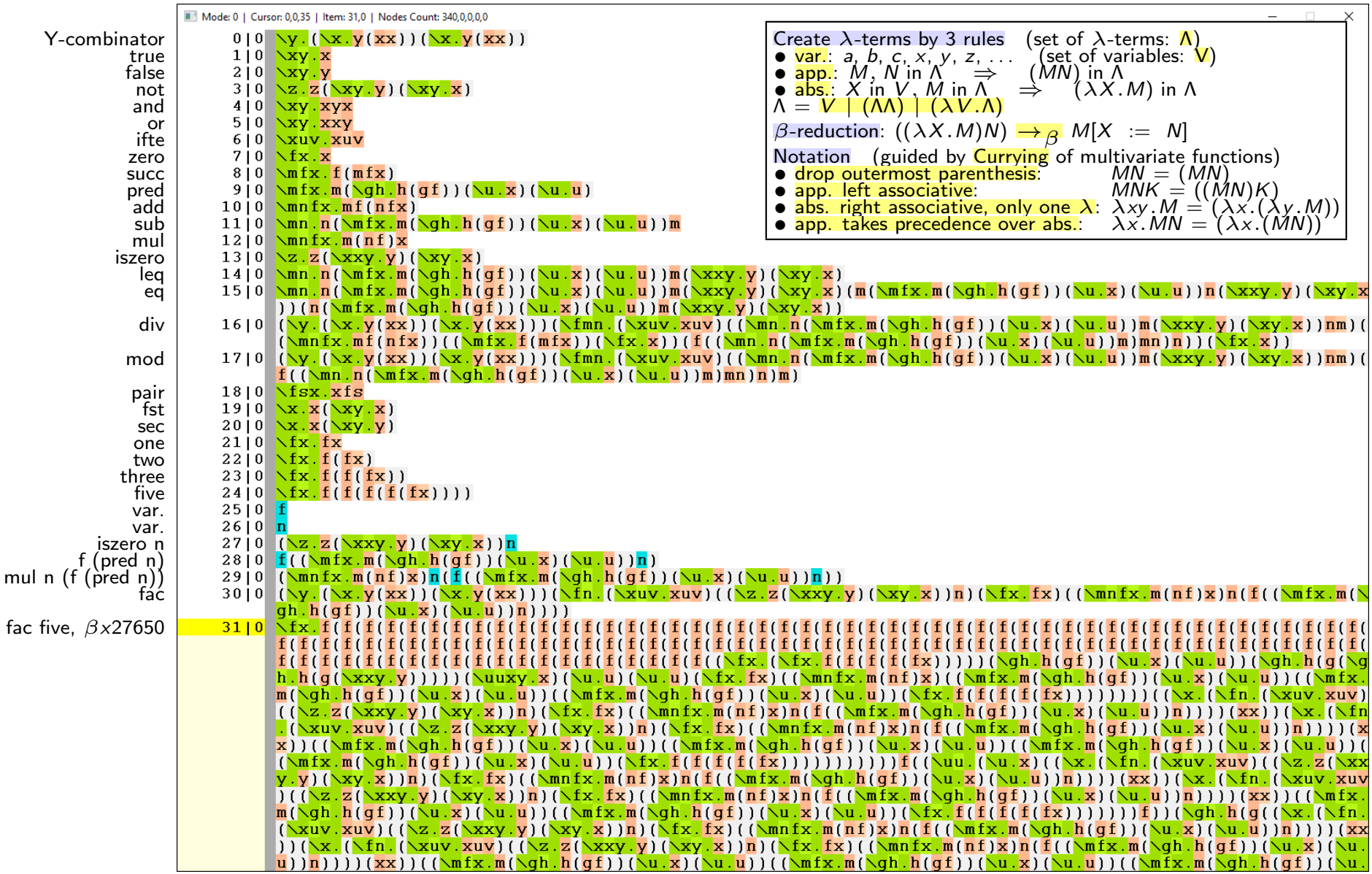
```
Y-combinator   0|0  \y.(\x.y(xx))(\x.y(xx))
true           1|0  \xy.x
false          2|0  \xy.y
not            3|0  \z.z(\xy.y)(\xy.x)
and            4|0  \xy.xyx
or             5|0  \xy.xxy
ifte           6|0  \xuv.xuv
zero           7|0  \fx.x
succ           8|0  \mfx.f(mfx)
pred           9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add           10|0  \mnfx.mf(nfx)
sub           11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul           12|0  \mnfx.m(nf)x
iszero        13|0  \z.z(\xxy.y)(\xy.x)
leq           14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq            15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                    ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div           16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod           17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                    f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair          18|0  \fsx.xfs
fst           19|0  \x.x(\xy.x)
sec           20|0  \x.x(\xy.y)
one           21|0  \fx.fx
two           22|0  \fx.f(fx)
three         23|0  \fx.f(f(fx))
five          24|0  \fx.f(f(f(f(fx))))
var.          25|0  f
var.          26|0  n
iszero n      27|0  (\z.z(\xxy.y)(\xy.x))n
f (pred n)    28|0  f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n (f (pred n))
              29|0  (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
fac           30|0  (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
                    gh.h(gf))(\u.x)(\u.u))n))))
fac five, βx25550
              31|0  \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                    f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                    f(f(f(f(f(f(f(f(f(f(f(\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))\gh.h(g(\gh.h(g(\gh.h(g(\gh.
                    h(g(\xxy.y)))))))))(\uuuuxy.x)(\u.u)(\u.u)(\u.u)(\u.u)(\fx.fx)((\mnfx.m(nf)x)((\mfx.m(\gh.h(gf))(\u.
                    x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\
                    u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx)))))))))))((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy
                    .x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\x.(\fn.(\xuv.xuv)((\z.z(
                    \xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(
                    gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))
                    (\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))))
                    )))))f((\uuuu.(\gh.h(g(\gh.h(g(\gh.h(g((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx))((\mnfx.m(
                    nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx
                    ))((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.
                    m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))((\mfx.m(\gh.h(gf))(\u.x)(\u.u)))(\fx.f(f(f(
                    f(fx))))))))))f))))))))(\uuu.(\gh.h(g(\gh.h(g((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx))((\mn
```

A crash course in λ-calculus

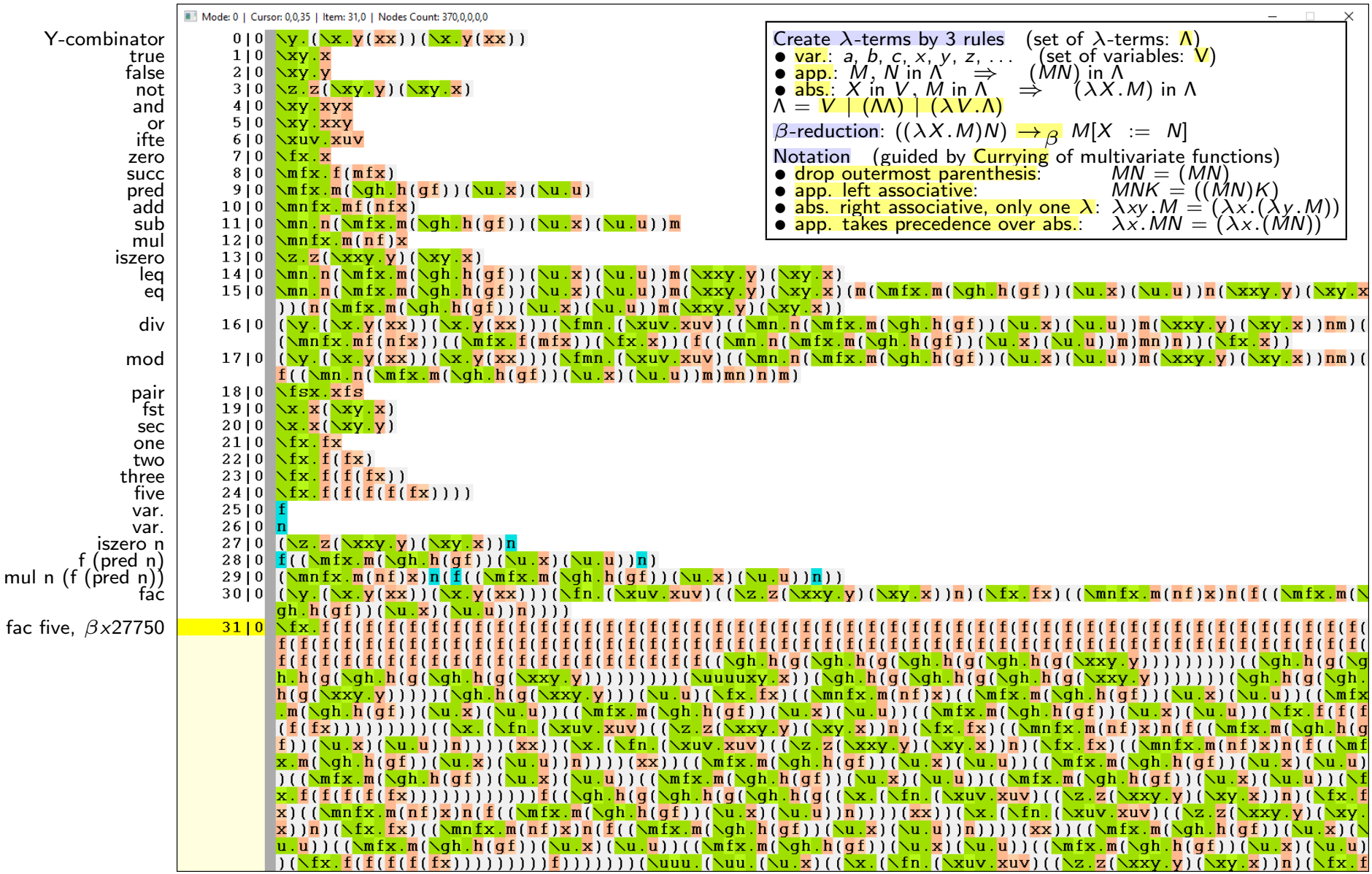


A crash course in λ-calculus

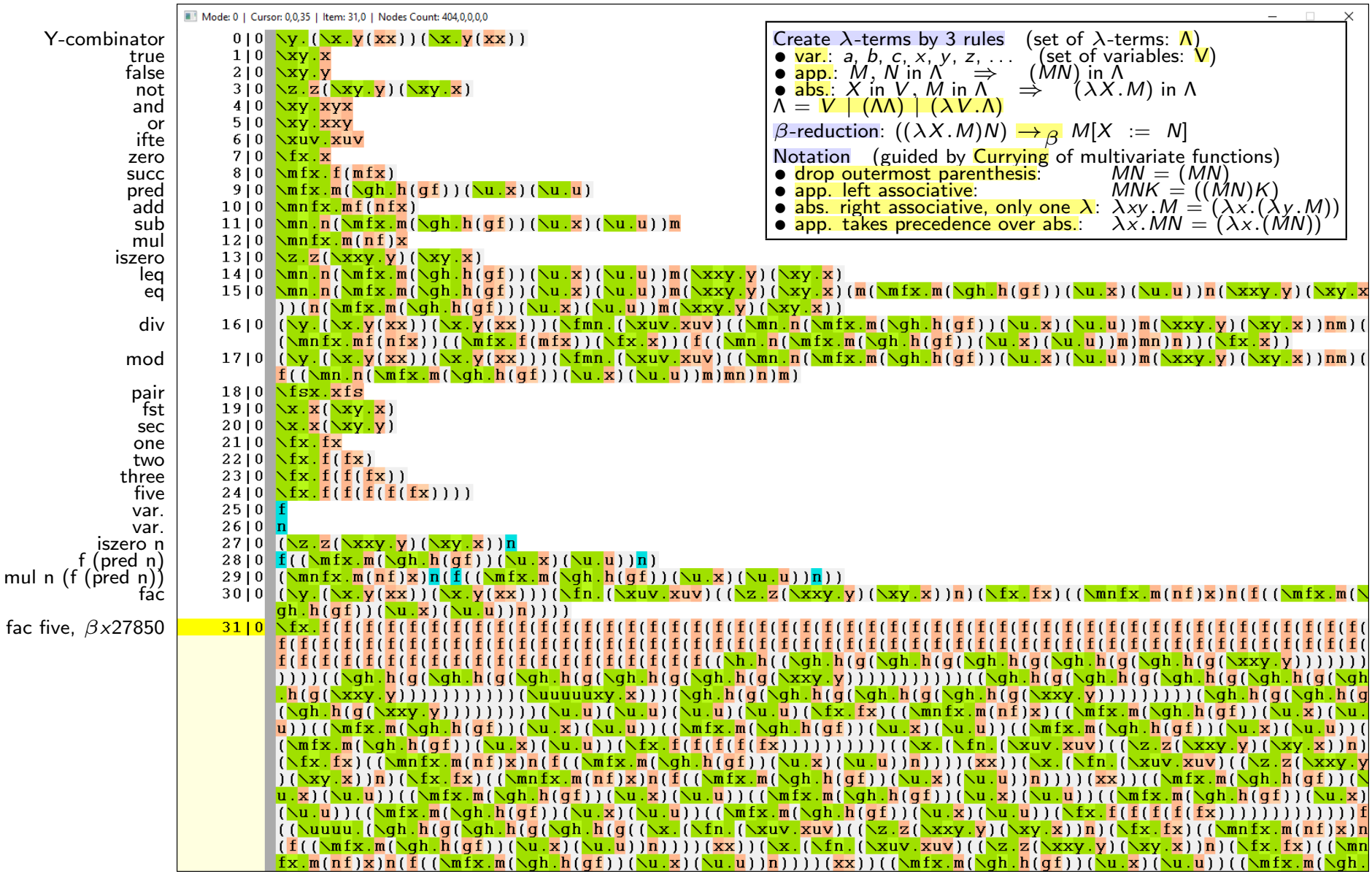



Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 340,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx27650

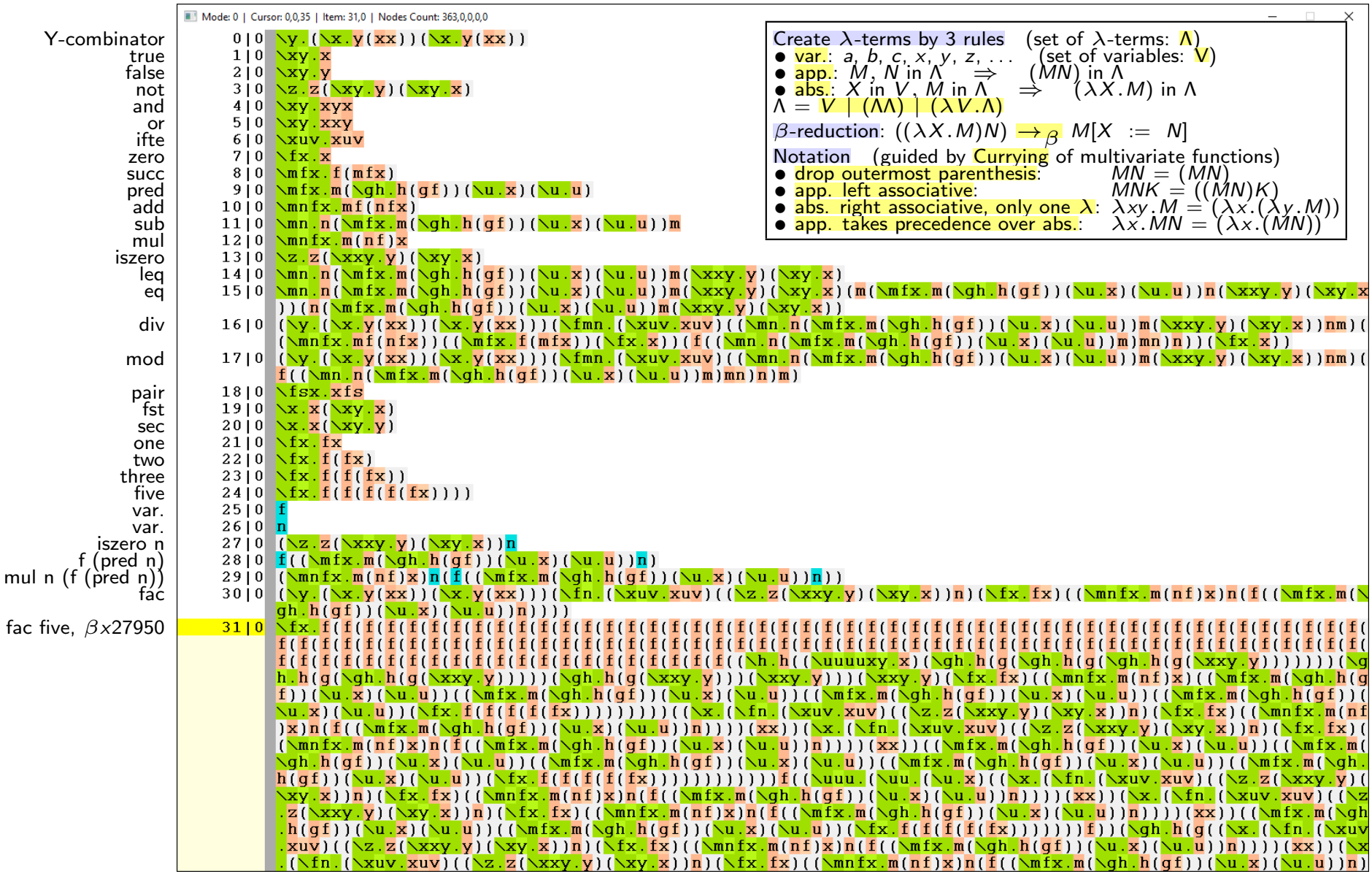
A crash course in λ-calculus



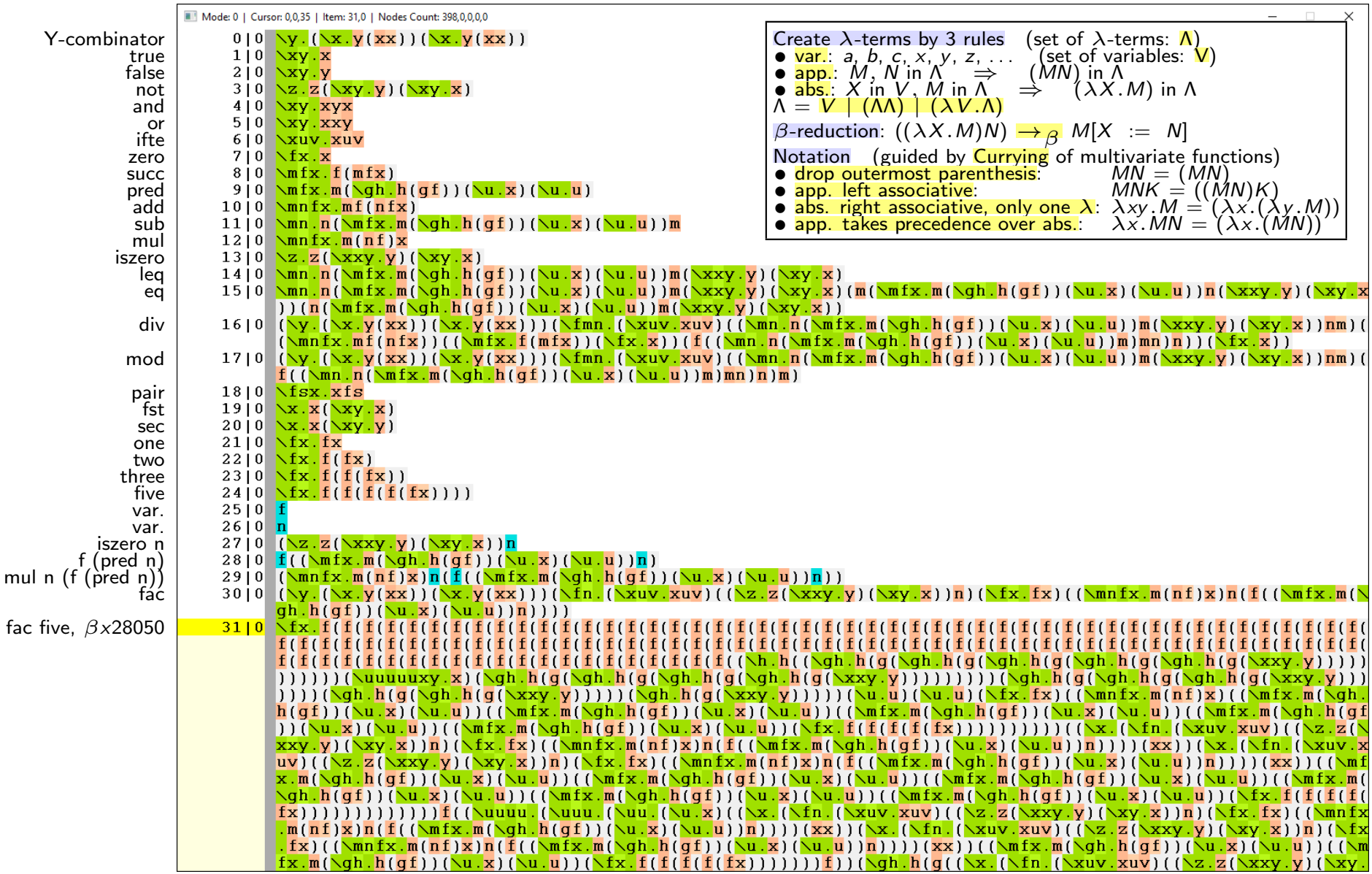
A crash course in λ-calculus

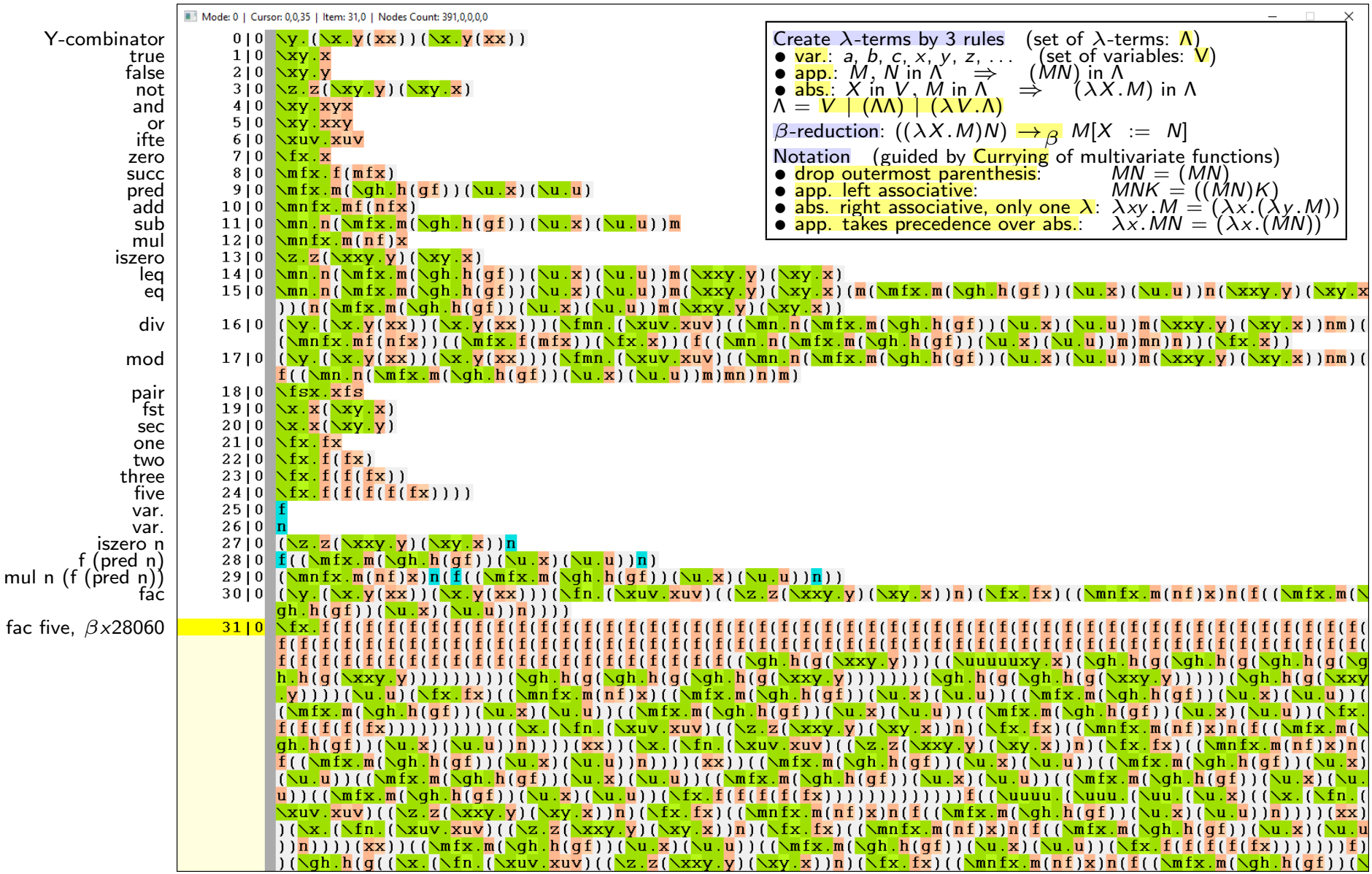


A crash course in λ-calculus

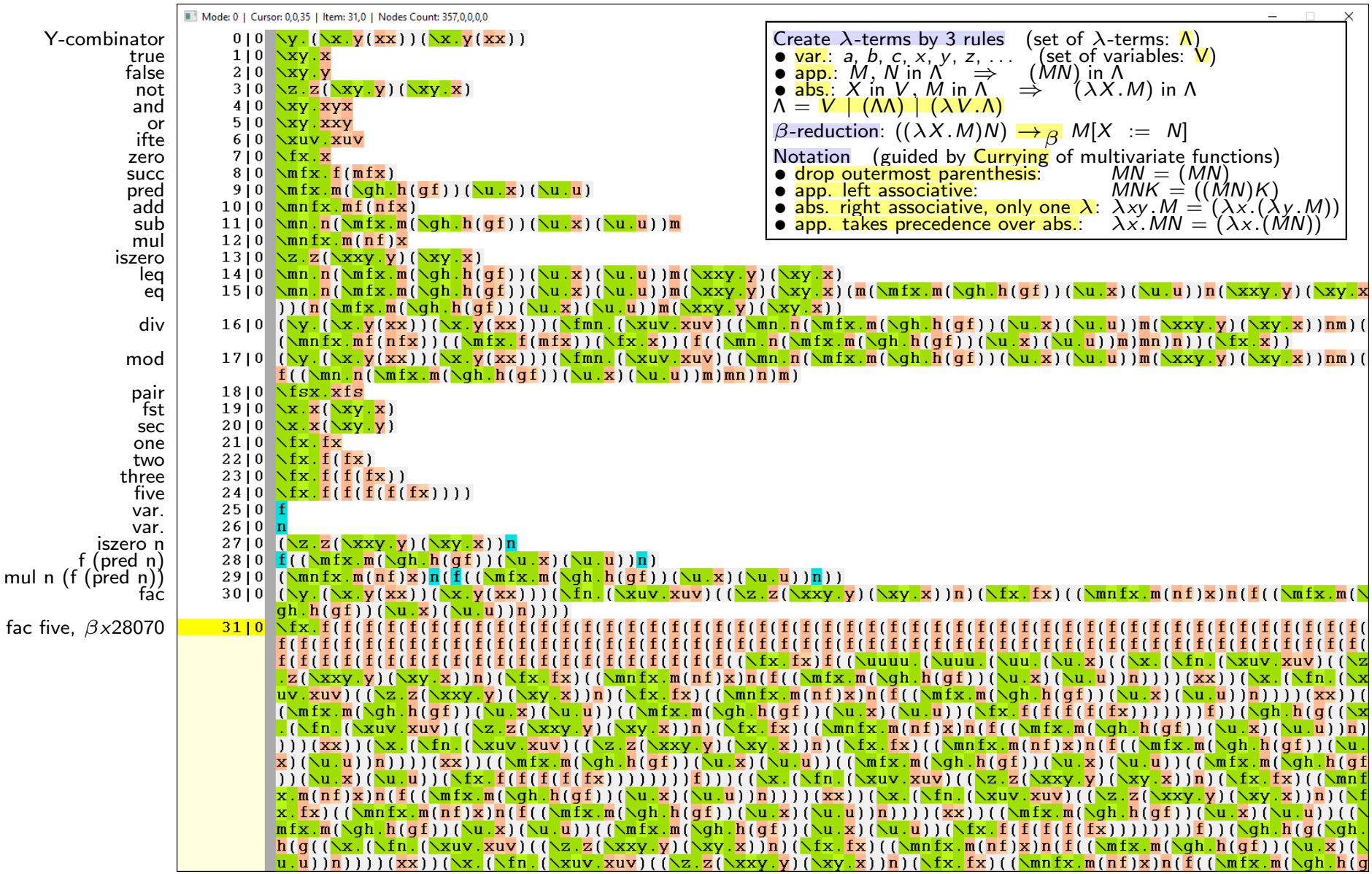


A crash course in λ-calculus



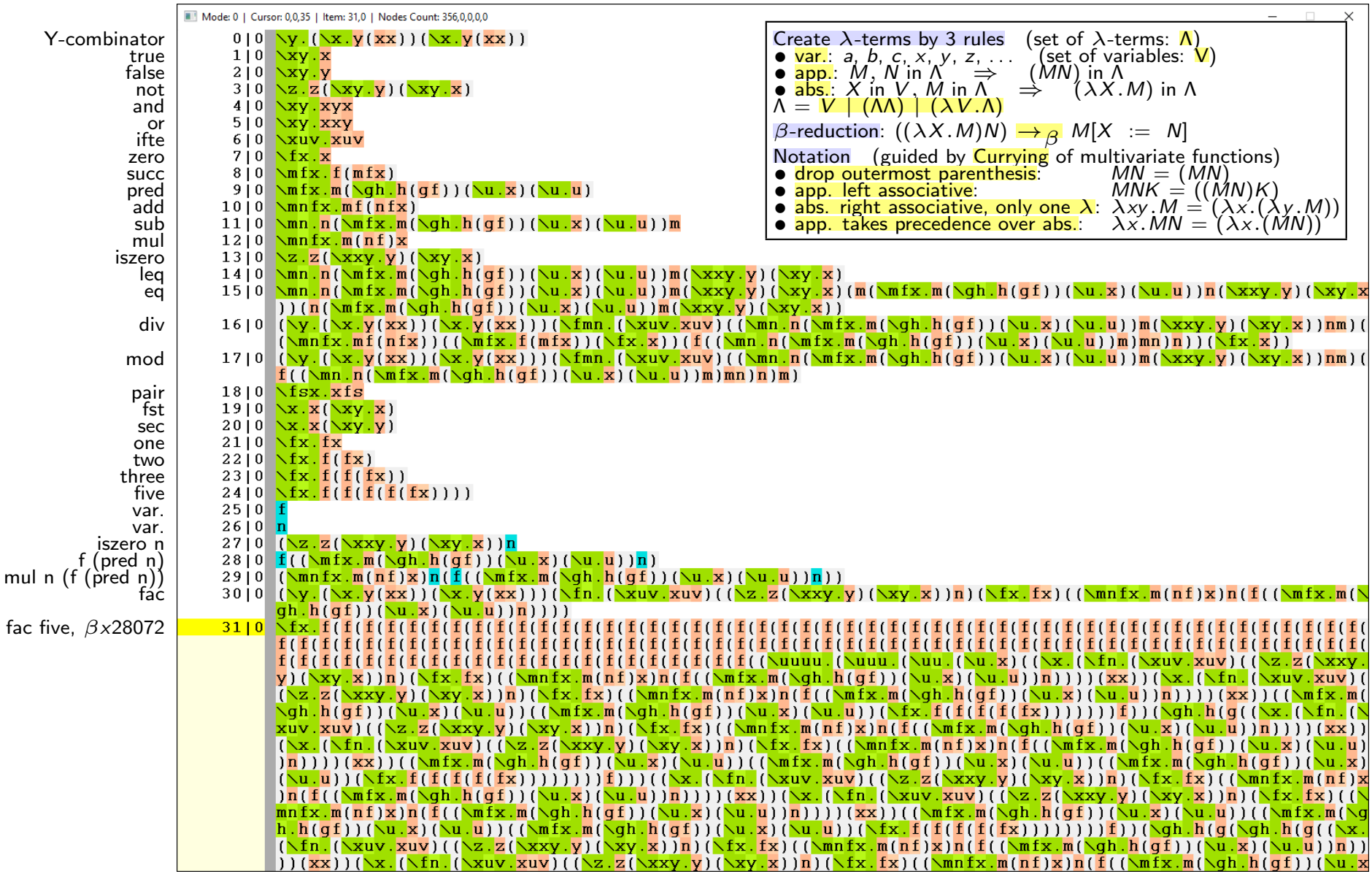

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 391,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx28060

A crash course in λ-calculus

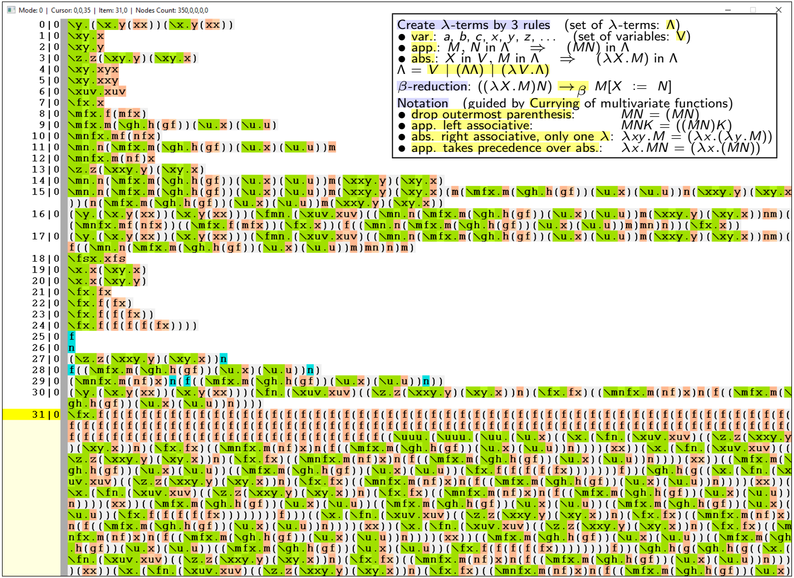


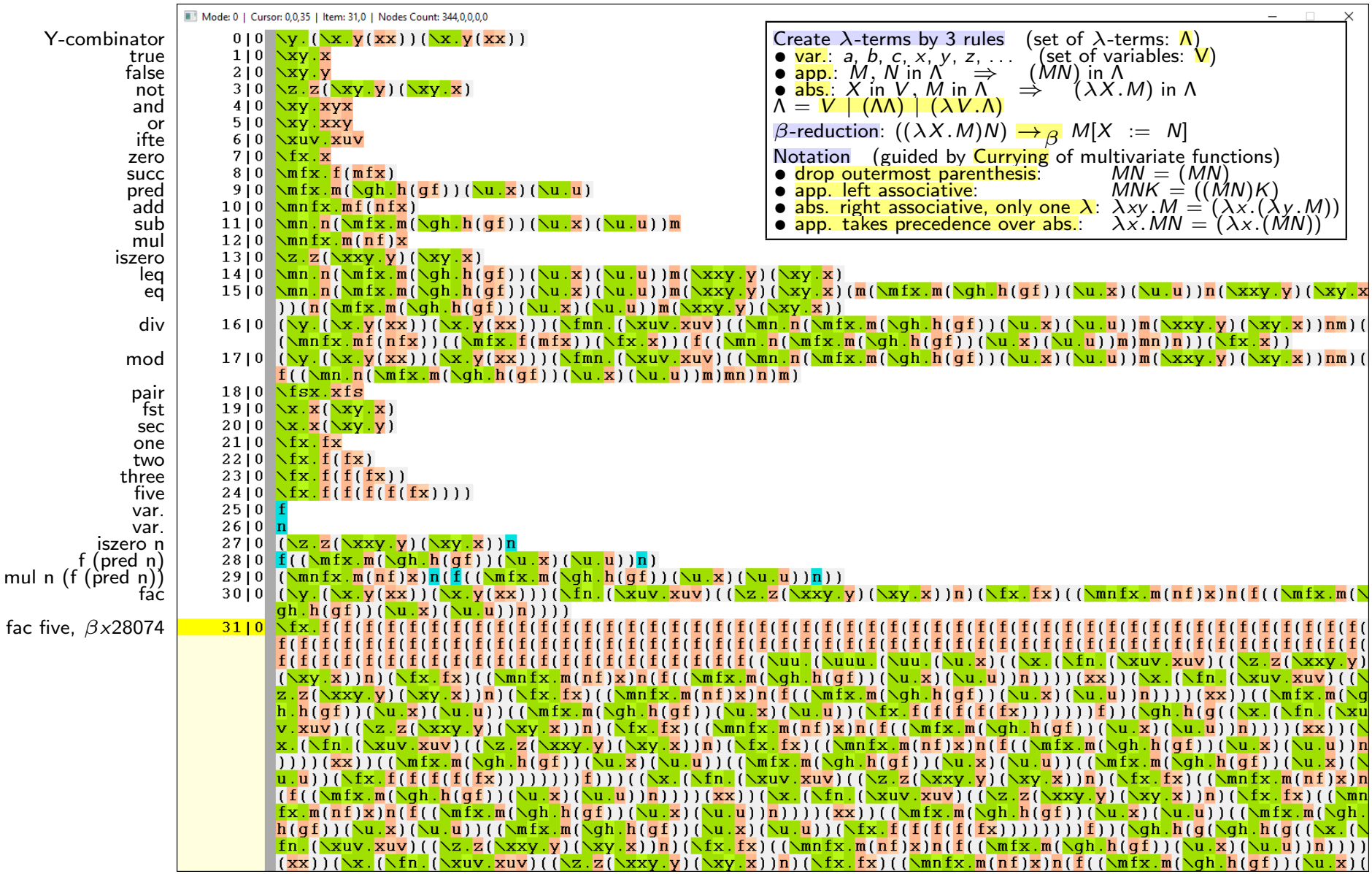
A crash course in λ-calculus

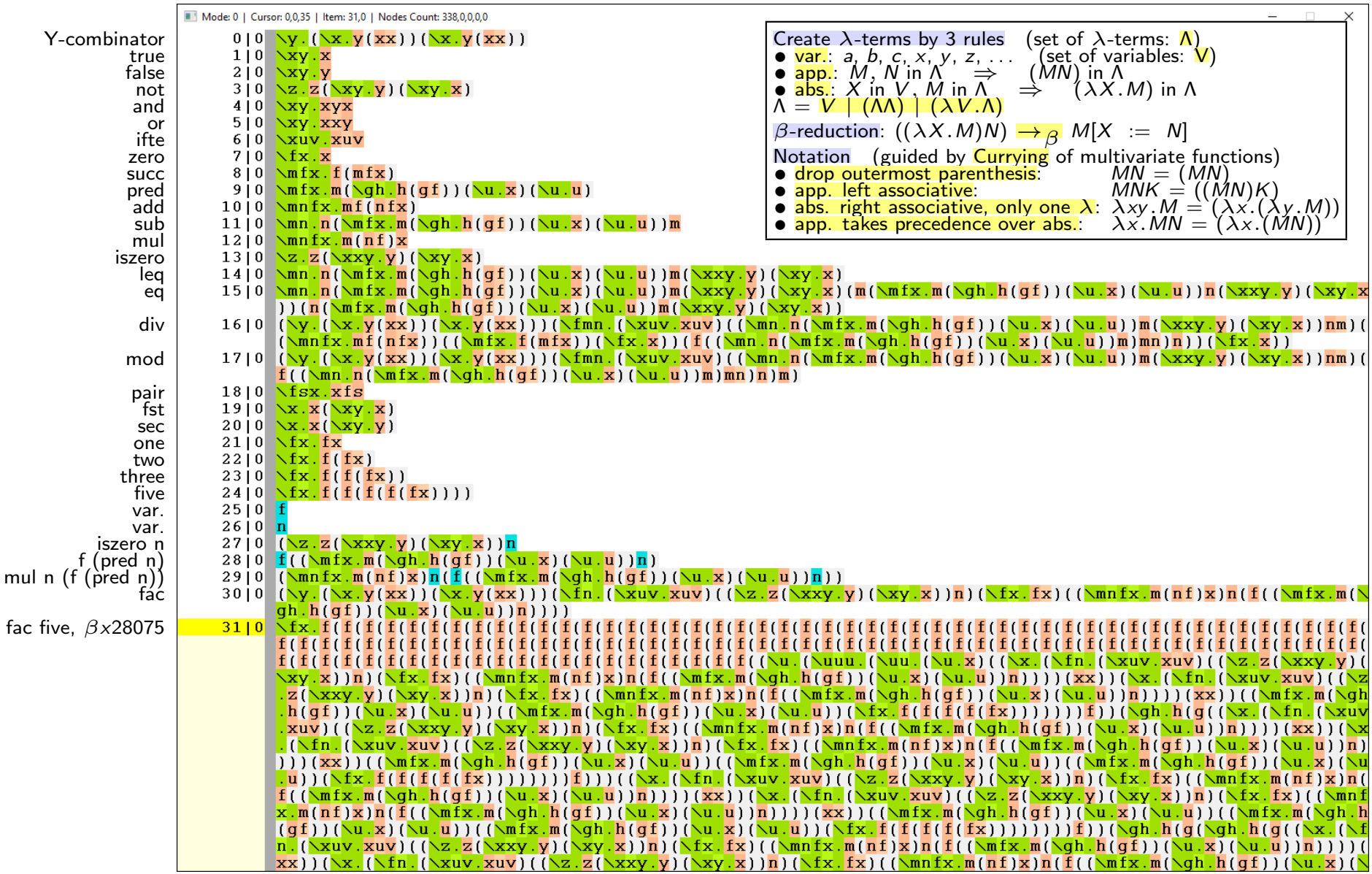
A crash course in λ-calculus



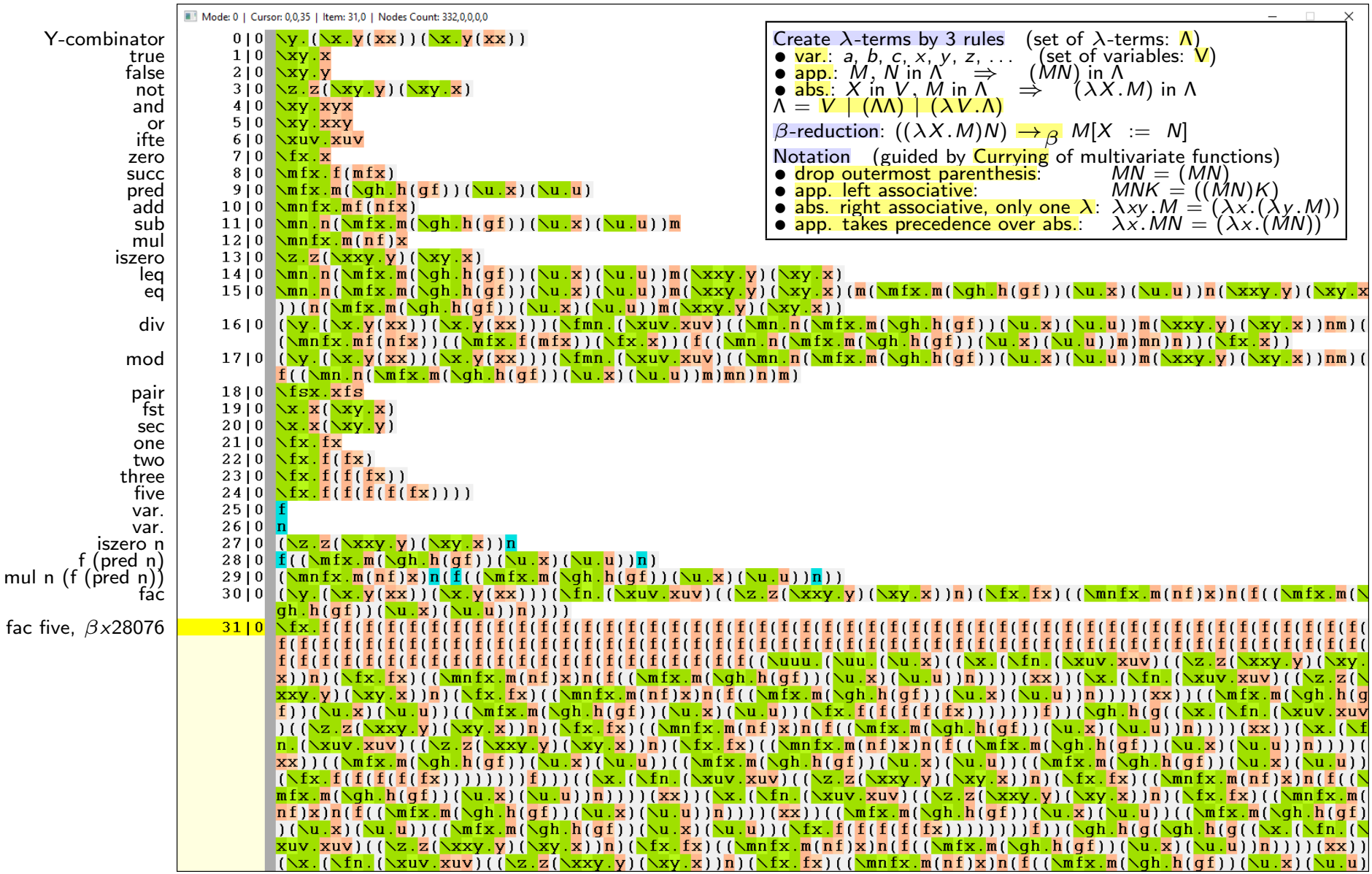
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, $\beta x28073$



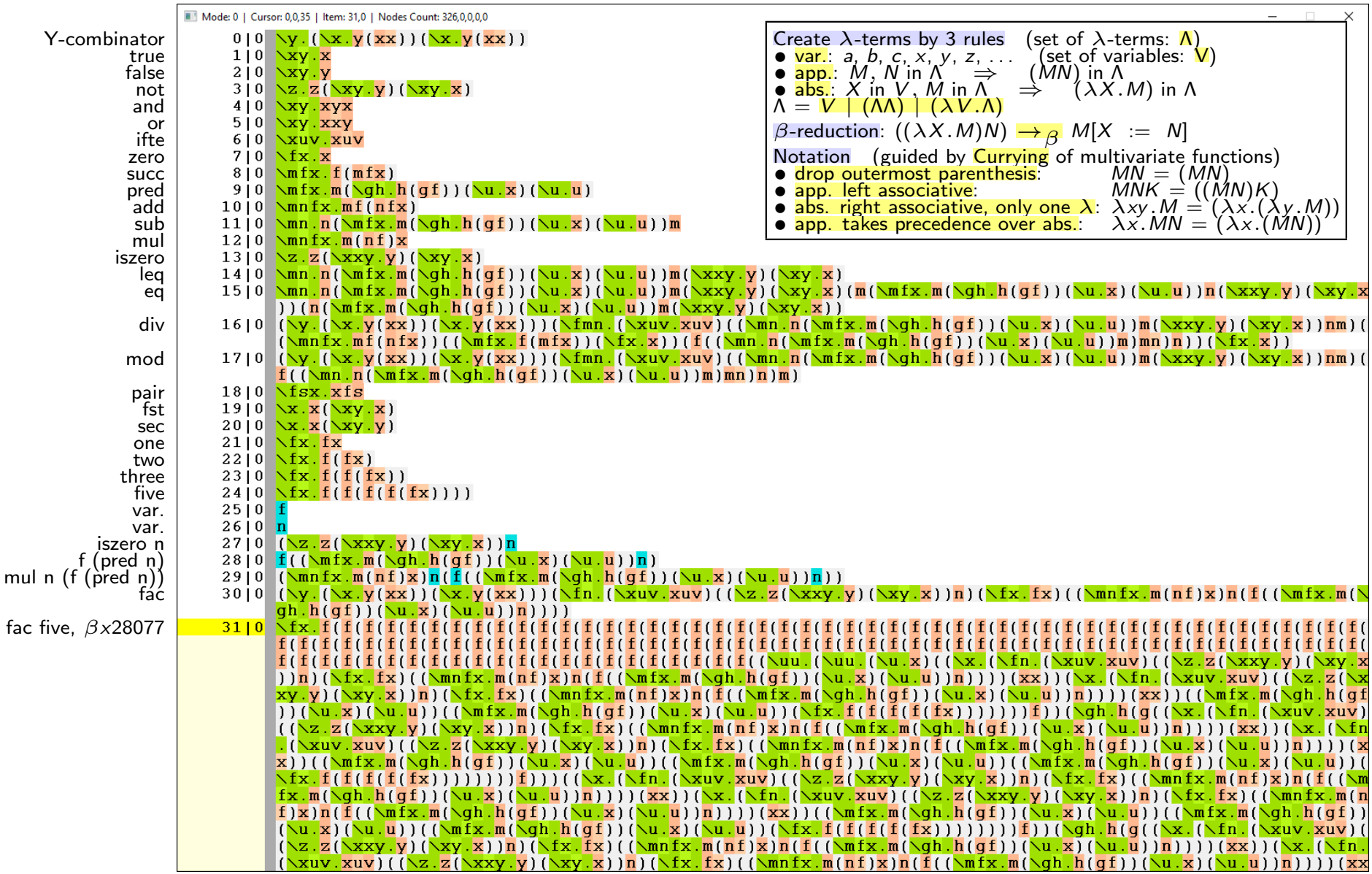

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 344,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx28074


Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 338,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx28075

A crash course in λ-calculus



A crash course in λ-calculus



A crash course in λ-calculus

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 320,0,0,0,0

Create λ-terms by 3 rules (set of λ-terms: Λ)
- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

Notation (guided by Currying of multivariate functions)
- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

```
Y-combinator  0|0  \y.(\x.y(xx))(\x.y(xx))
true          1|0  \xy.x
false         2|0  \xy.y
not           3|0  \z.z(\xy.y)(\xy.x)
and           4|0  \xy.xyx
or            5|0  \xy.xxy
ifte          6|0  \xuv.xuv
zero          7|0  \fx.x
succ          8|0  \mfx.f(mfx)
pred          9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add          10|0  \mnfx.mf(nfx)
sub          11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul          12|0  \mnfx.m(nf)x
iszero       13|0  \z.z(\xxy.y)(\xy.x)
leq          14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq           15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                   ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div          16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)))(\fx.x))
mod          17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                   f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair         18|0  \fsx.xfs
fst          19|0  \x.x(\xy.x)
sec          20|0  \x.x(\xy.y)
one          21|0  \fx.fx
two          22|0  \fx.f(fx)
three        23|0  \fx.f(f(fx))
five         24|0  \fx.f(f(f(f(fx))))
var.         25|0  f
var.         26|0  n
iszero n     27|0  (\z.z(\xxy.y)(\xy.x))n
f (pred n)   28|0  f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n (f (pred n))
             29|0  (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
fac          30|0  (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
                   gh.h(gf))(\u.x)(\u.u))n))))
fac five, βx28078
             31|0  \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                   f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                   f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                   )n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xx
                   y.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf)
                   )(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))f))(\gh.h(g((\x.(\fn.(\xuv.xuv)(
                   (\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.
                   (\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx
                   ))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\
                   fx.f(f(f(f(fx))))))))f))((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mf
                   x.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf
                   )x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(
                   \u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx))))))))f))((\x.(\fn.(\xuv.xuv)((\z.z(\xx
                   y.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\x.(\fn.(\xuv.xuv
                   )((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))))(xx))((\mfx.
```

Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 314,0,0,0,0

```
Y-combinator        0|0  \y.(\x.y(xx))(\x.y(xx))
true                1|0  \xy.x
false               2|0  \xy.y
not                 3|0  \z.z(\xy.y)(\xy.x)
and                 4|0  \xy.xyx
or                  5|0  \xy.xxy
ifte                6|0  \xuv.xuv
zero                7|0  \fx.x
succ                8|0  \mfx.f(mfx)
pred                9|0  \mfx.m(\gh.h(gf))(\u.x)(\u.u)
add                10|0  \mnfx.mf(nfx)
sub                11|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m
mul                12|0  \mnfx.m(nf)x
iszero             13|0  \z.z(\xxy.y)(\xy.x)
leq                14|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)
eq                 15|0  \mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x)(m(\mfx.m(\gh.h(gf))(\u.x)(\u.u))n(\xxy.y)(\xy.x
                         ))(n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))
div                16|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                         (\mnfx.mf(nfx))((\mfx.f(mfx))(\fx.x))(f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n))(\fx.x))
mod                17|0  (\y.(\x.y(xx))(\x.y(xx)))(\fmn.(\xuv.xuv)((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m(\xxy.y)(\xy.x))nm)(
                         f((\mn.n(\mfx.m(\gh.h(gf))(\u.x)(\u.u))m)mn)n)m)
pair               18|0  \fsx.xfs
fst                19|0  \x.x(\xy.x)
sec                20|0  \x.x(\xy.y)
one                21|0  \fx.fx
two                22|0  \fx.f(fx)
three              23|0  \fx.f(f(fx))
five               24|0  \fx.f(f(f(f(fx))))
var.               25|0  f
var.               26|0  n
iszero n           27|0  (\z.z(\xxy.y)(\xy.x))n
f (pred n)         28|0  f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)
mul n (f (pred n)) 29|0  (\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n))
fac                30|0  (\y.(\x.y(xx))(\x.y(xx)))(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\
                         gh.h(gf))(\u.x)(\u.u))n))))
fac five, βx28079  31|0  \fx.f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                         f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                         f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f((\uu.(\u.x)((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(
                         \fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)
                         (\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))))(xx))((\mfx.m(\gh.h(gf))(\u
                         .x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx)))))))f))(\gh.h(g((\x.(\fn.(\xuv.xuv)((\z.
                         z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))))(xx))(\x.(\fn.(\xu
                         v.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))))(xx))((
                         \mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f
                         (f(f(f(fx)))))))f)))((\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n(f((\mfx.m(
                         \gh.h(gf))(\u.x)(\u.u))n)))))(xx))(\x.(\fn.(\xuv.xuv)((\z.z(\xxy.y)(\xy.x))n)(\fx.fx)((\mnfx.m(nf)x)n
                         (f((\mfx.m(\gh.h(gf))(\u.x)(\u.u))n)))))(xx))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))((\mfx.m(\gh.h(gf))(\u.x
                         )(\u.u))((\mfx.m(\gh.h(gf))(\u.x)(\u.u))(\fx.f(f(f(f(fx)))))))f))))))))))))))))))))))))))))))))))))
                         ))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
                   32|
```

Create λ-terms by 3 rules (set of λ-terms: Λ)

- var.: $a, b, c, x, y, z, \ldots$ (set of variables: V)
- app.: M, N in Λ $\Rightarrow$ (MN) in Λ
- abs.: X in V, M in Λ $\Rightarrow$ $(\lambda X.M)$ in Λ

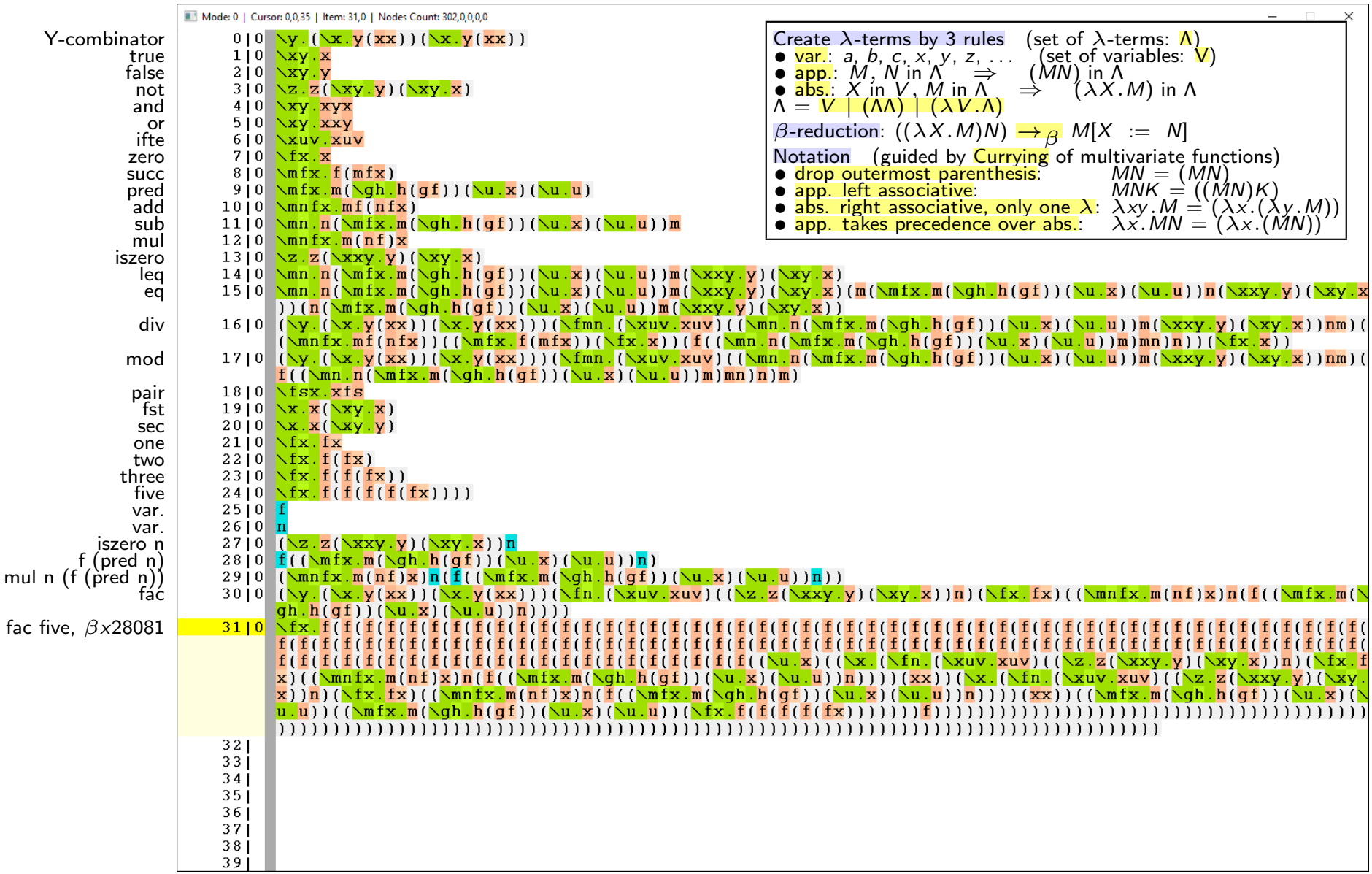
$\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$

β-reduction: $((\lambda X.M)N) \rightarrow_\beta M[X := N]$

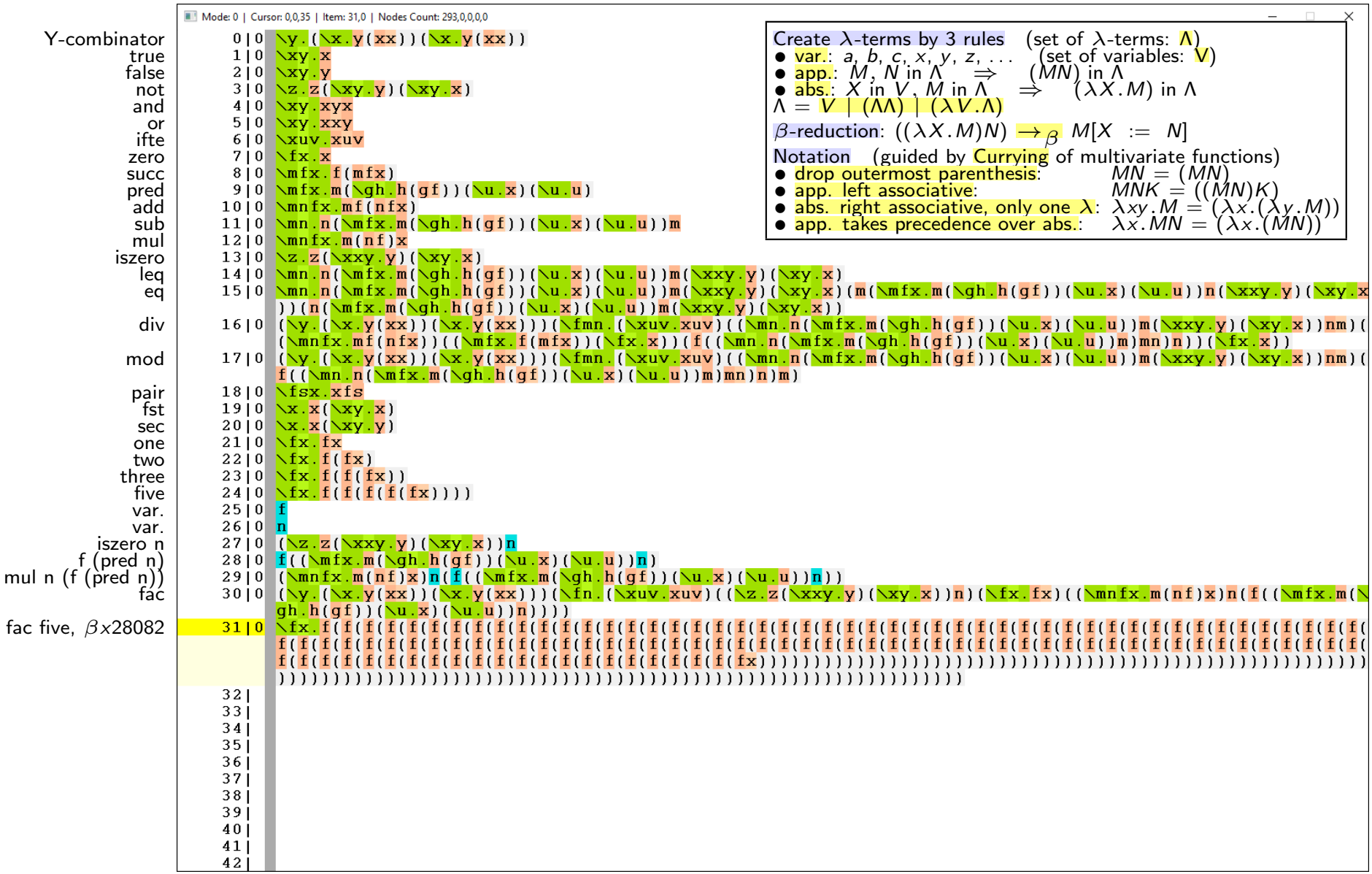
Notation (guided by Currying of multivariate functions)

- drop outermost parenthesis: $MN = (MN)$
- app. left associative: $MNK = ((MN)K)$
- abs. right associative, only one λ: $\lambda xy.M = (\lambda x.(\lambda y.M))$
- app. takes precedence over abs.: $\lambda x.MN = (\lambda x.(MN))$

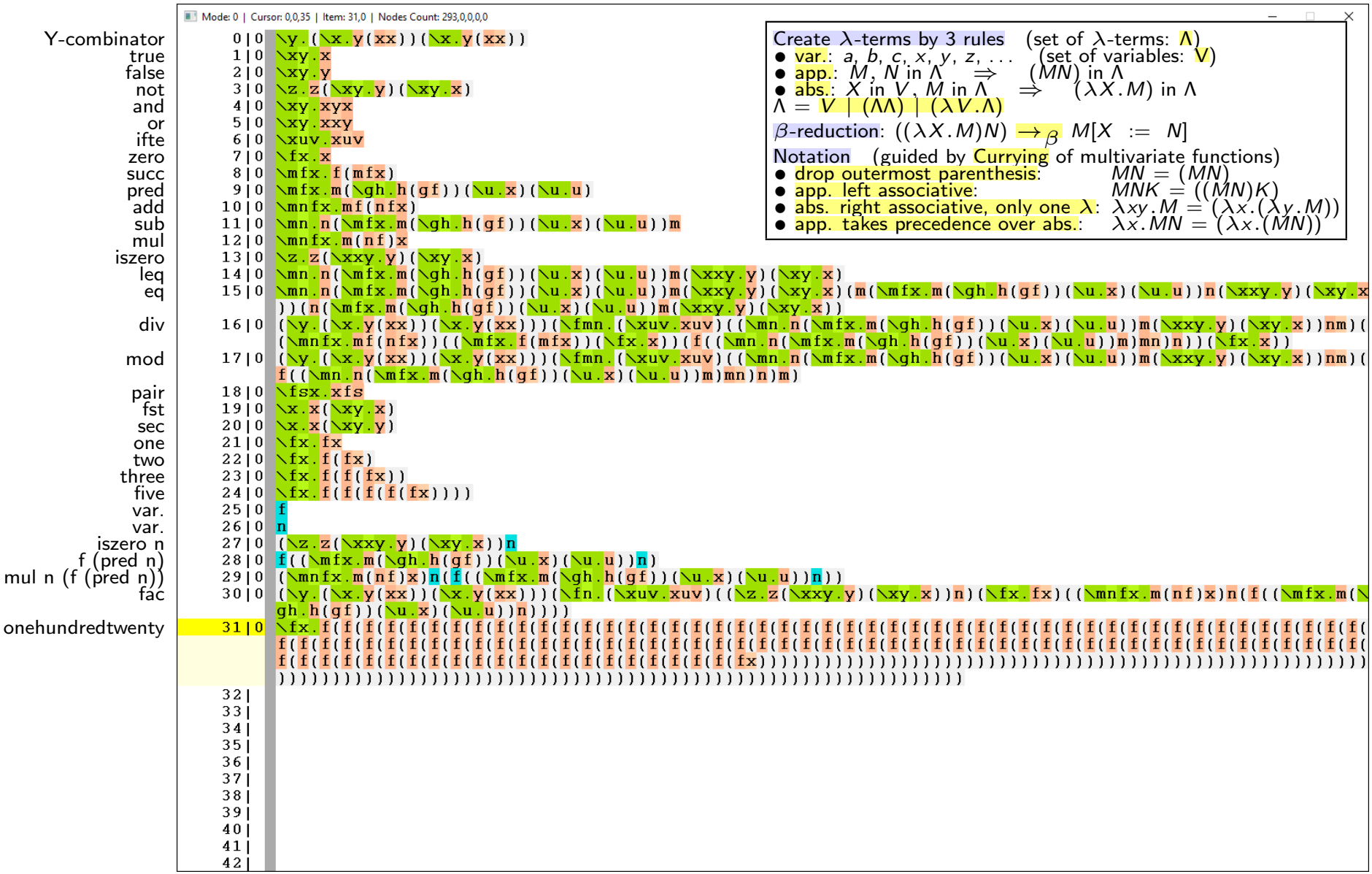
A crash course in λ-calculus


Mode: 0 | Cursor: 0,0,35 | Item: 31,0 | Nodes Count: 302,0,0,0,0
Create λ-terms by 3 rules (set of λ-terms: Λ)
• var.: a, b, c, x, y, z, ... (set of variables: V)
• app.: M, N in Λ ⇒ (MN) in Λ
• abs.: X in V, M in Λ ⇒ (λX.M) in Λ
Λ = V | (ΛΛ) | (λV.Λ)
β-reduction: ((λX.M)N) →β M[X := N]
Notation (guided by Currying of multivariate functions)
• drop outermost parenthesis: MN = (MN)
• app. left associative: MNK = ((MN)K)
• abs. right associative, only one λ: λxy.M = (λx.(λy.M))
• app. takes precedence over abs.: λx.MN = (λx.(MN))
Y-combinator
true
false
not
and
or
ifte
zero
succ
pred
add
sub
mul
iszero
leq
eq
div
mod
pair
fst
sec
one
two
three
five
var.
var.
iszero n
f (pred n)
mul n (f (pred n))
fac
fac five, βx28081

A crash course in λ-calculus



A crash course in λ-calculus



Alonzo Church
(1903 - 1995)
Lambda Calculus: 1930s

Alan Turing
(1912 - 1954)
Turing Machine: 1936

Alonzo Church
(1903 - 1995)
Lambda Calculus: 1930s

Alan Turing
(1912 - 1954)
Turing Machine: 1936

- Church was Turing's doctoral advisor

Alonzo Church
(1903 - 1995)
Lambda Calculus: 1930s

Alan Turing
(1912 - 1954)
Turing Machine: 1936

- Church was Turing's doctoral advisor
- Church: the λ-calculus is Turing complete

Alonzo Church
(1903 - 1995)
Lambda Calculus: 1930s

Alan Turing
(1912 - 1954)
Turing Machine: 1936

- Church was Turing's doctoral advisor
- Church: the λ-calculus is Turing complete
- Turing: the λ-calculus can be emulated on a Turing machine

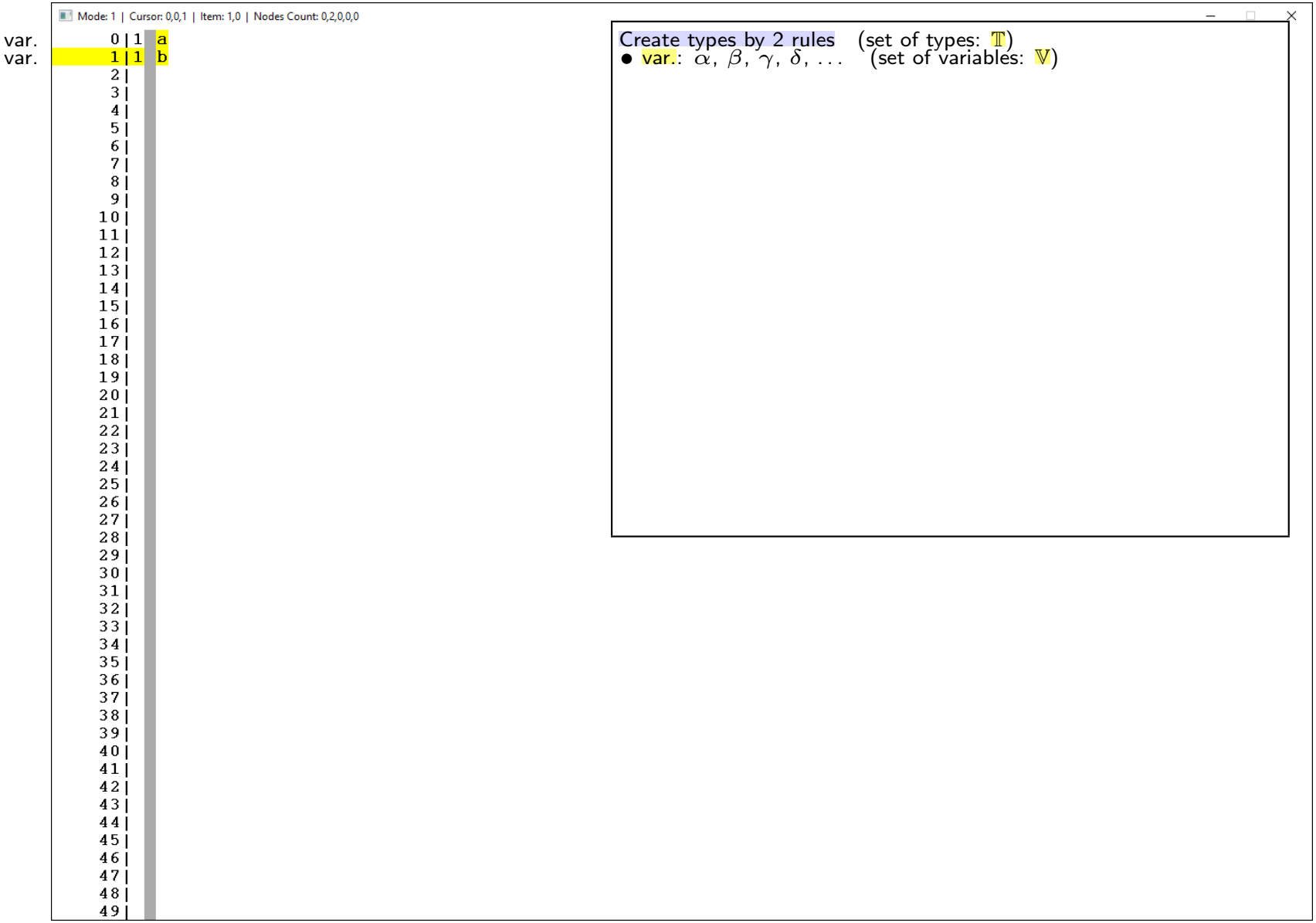
Alonzo Church
(1903 - 1995)
Lambda Calculus: 1930s

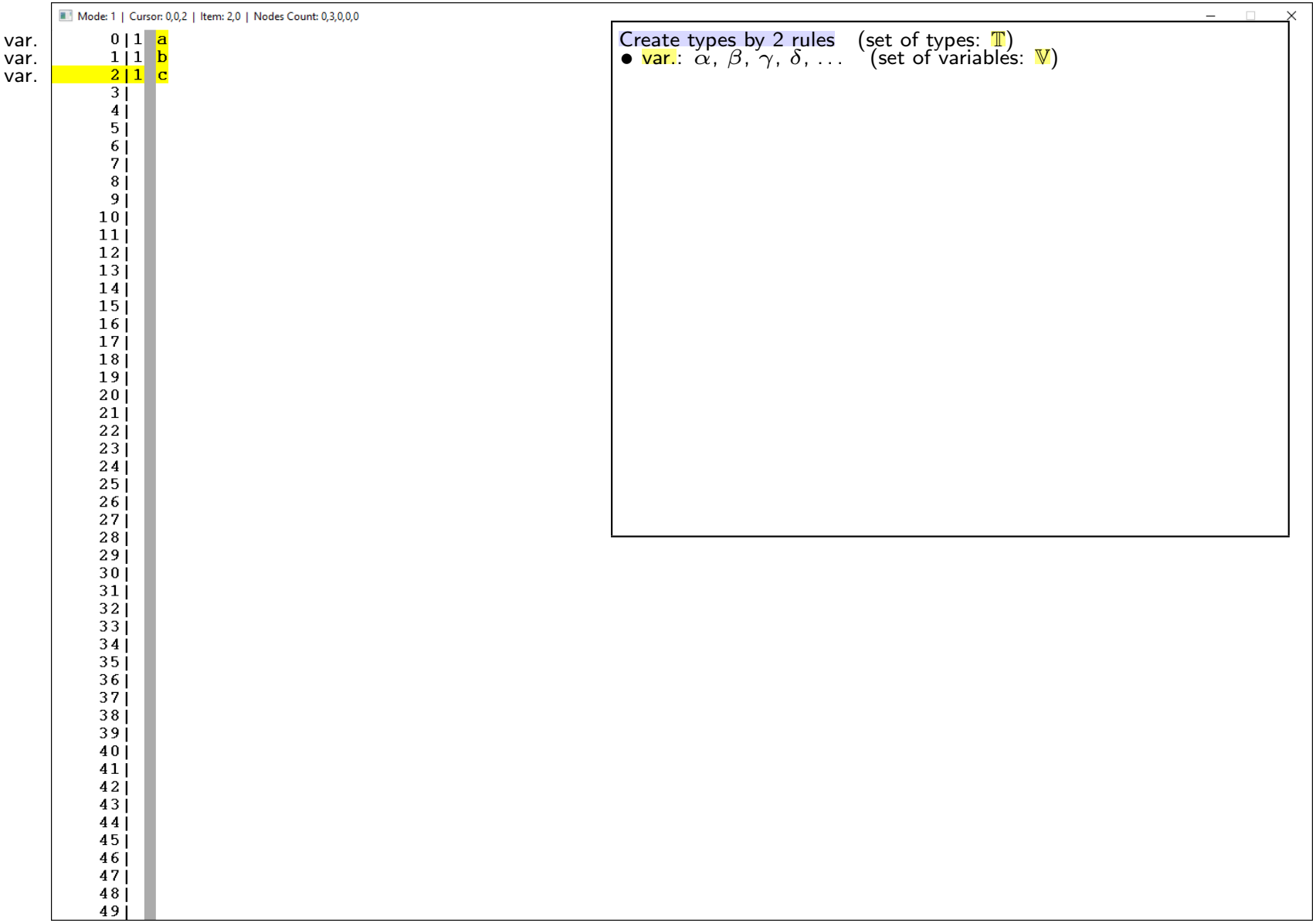
Alan Turing
(1912 - 1954)
Turing Machine: 1936

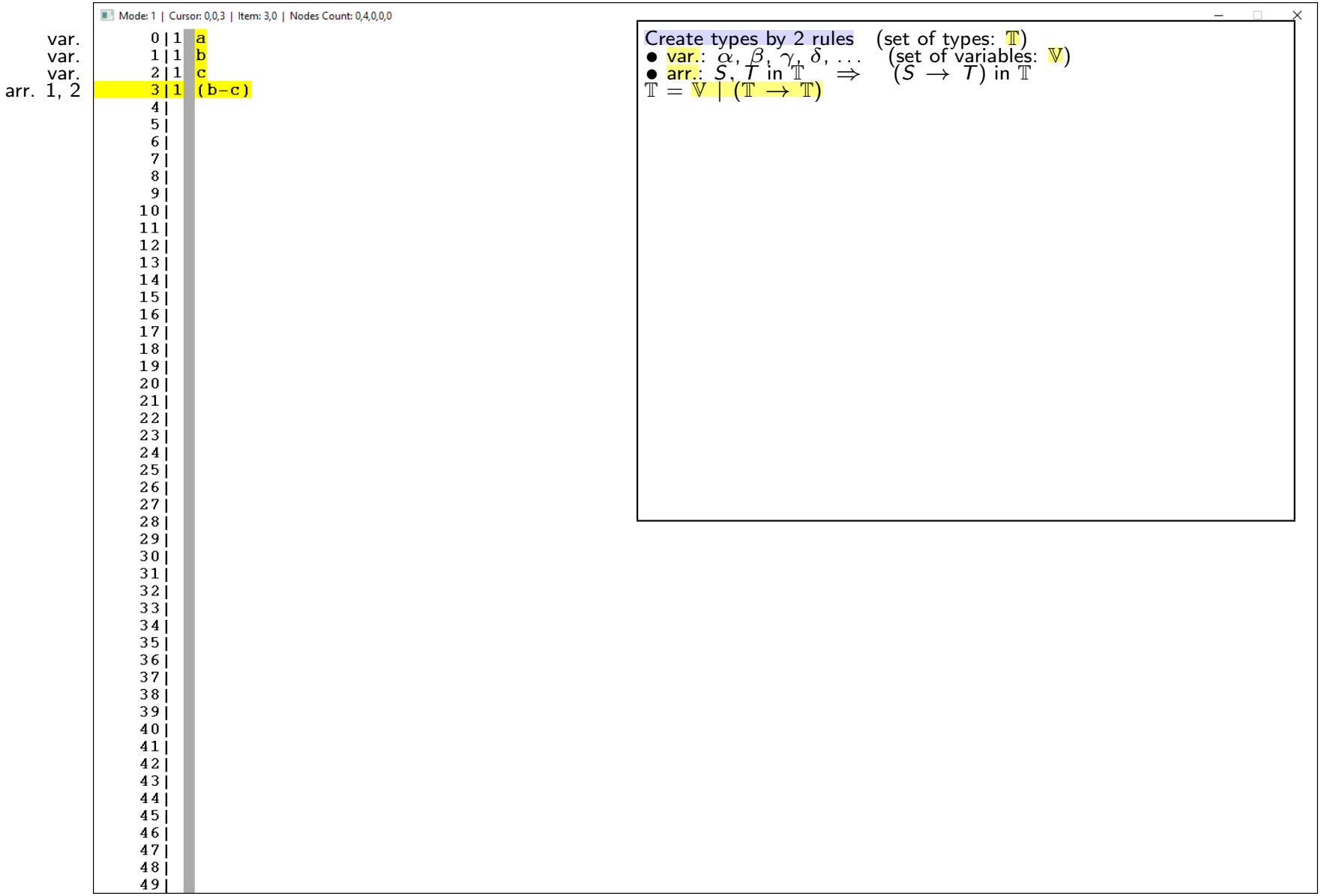
- Church was Turing's doctoral advisor
- Church: the λ-calculus is Turing complete
- Turing: the λ-calculus can be emulated on a Turing machine
- Church-Turing Thesis: in our universe all "effectively computable functions" can be computed by the λ-calculus/Turing machines

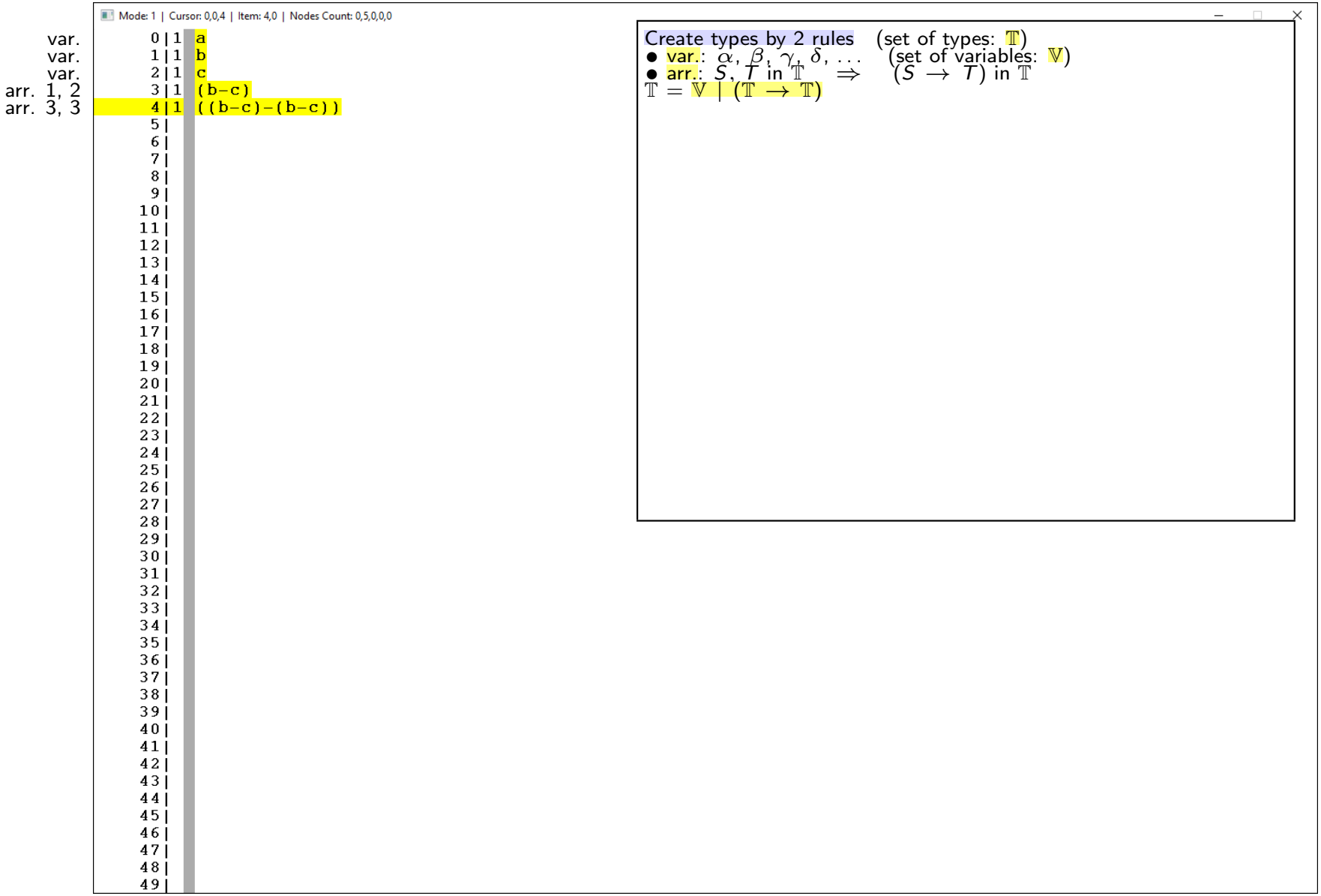
var.

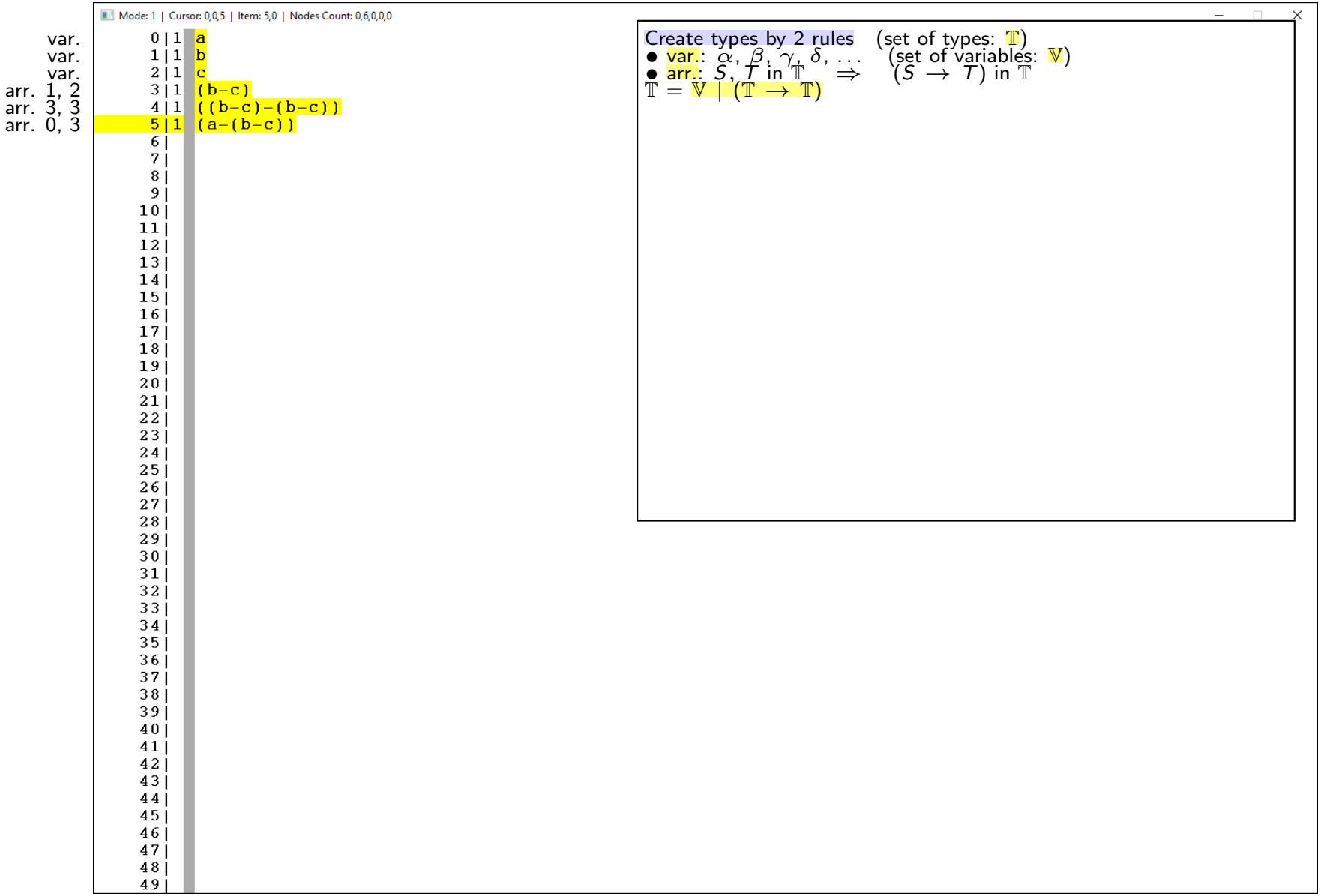

Mode: 1 | Cursor: 0,0,0 | Item: 0,0 | Nodes Count: 0,1,0,0,0
0 | 1 a
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: α, β, γ, δ, ... (set of variables: $\mathbb{V}$)

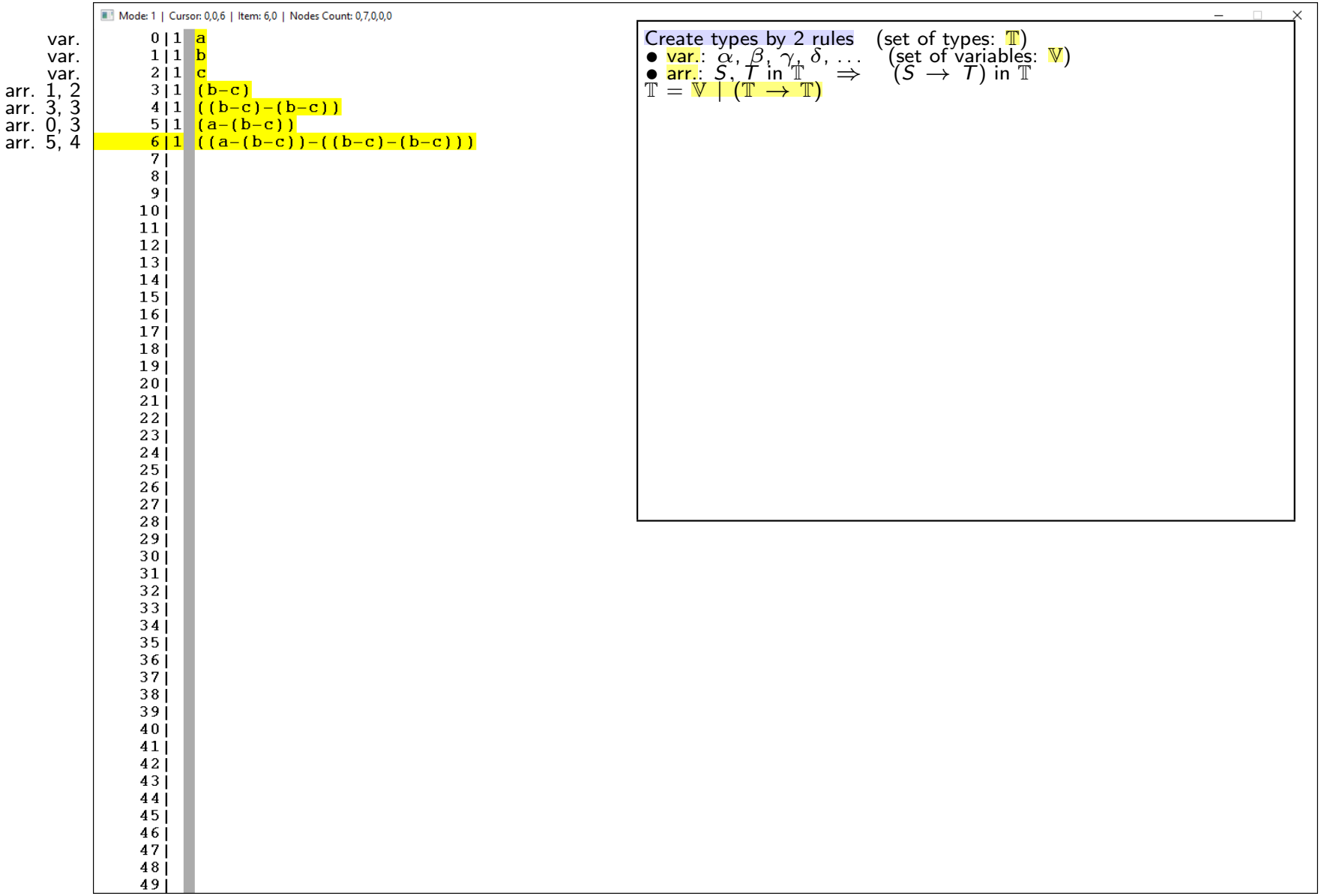

Mode: 1 | Cursor: 0,0,1 | Item: 1,0 | Nodes Count: 0,2,0,0,0
var.
var.
0 | 1 a
1 | 1 b
Create types by 2 rules (set of types: 𝕋)
• var.: α, β, γ, δ, ... (set of variables: 𝕍)

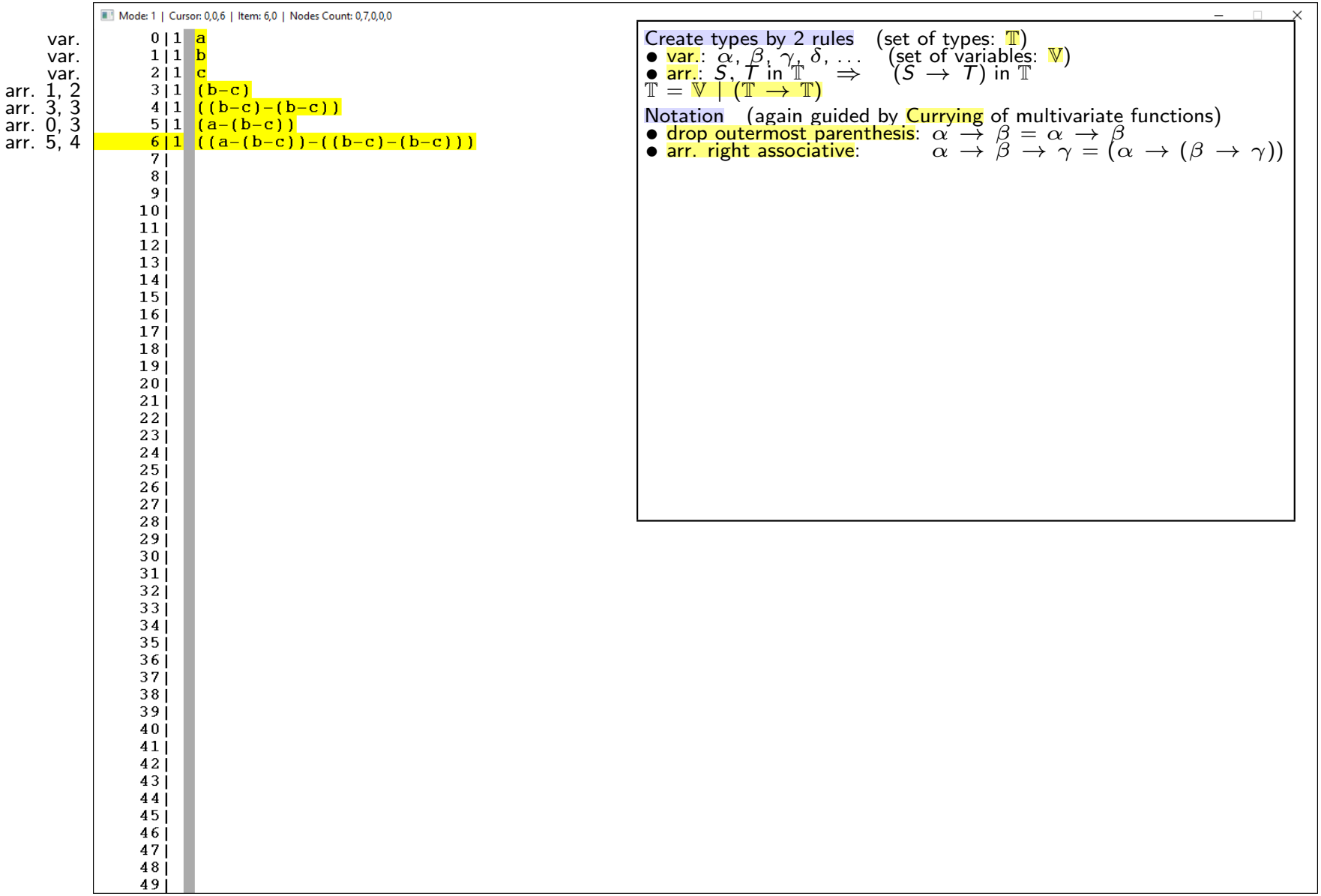

Mode: 1 | Cursor: 0,0,2 | Item: 2,0 | Nodes Count: 0,3,0,0,0
var.
var.
var.
0 | 1 a
1 | 1 b
2 | 1 c
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: α, β, γ, δ, ... (set of variables: $\mathbb{V}$)

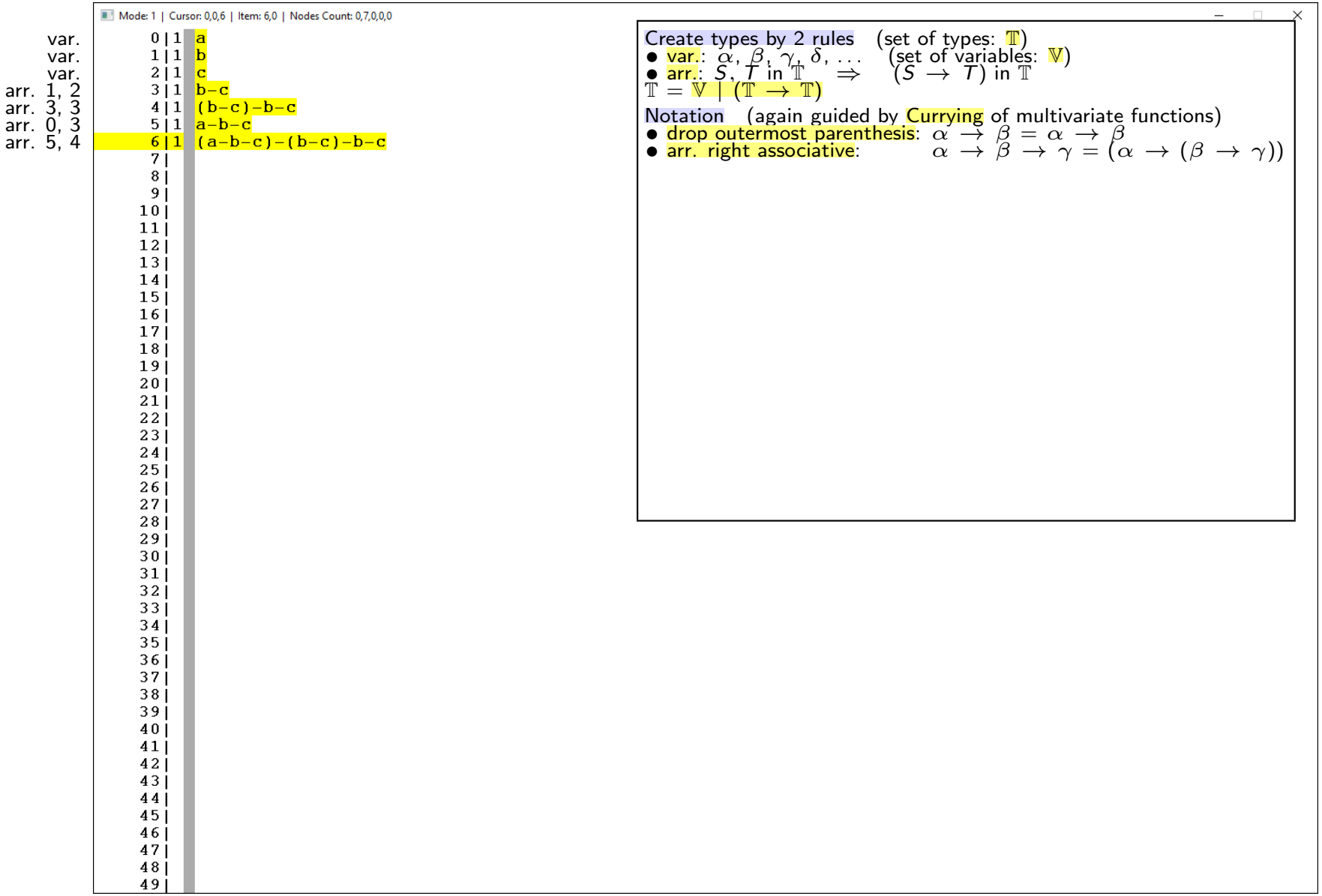

Mode: 1 | Cursor: 0,0,3 | Item: 3,0 | Nodes Count: 0,4,0,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 (b-c)
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: α, β, γ, δ, ... (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

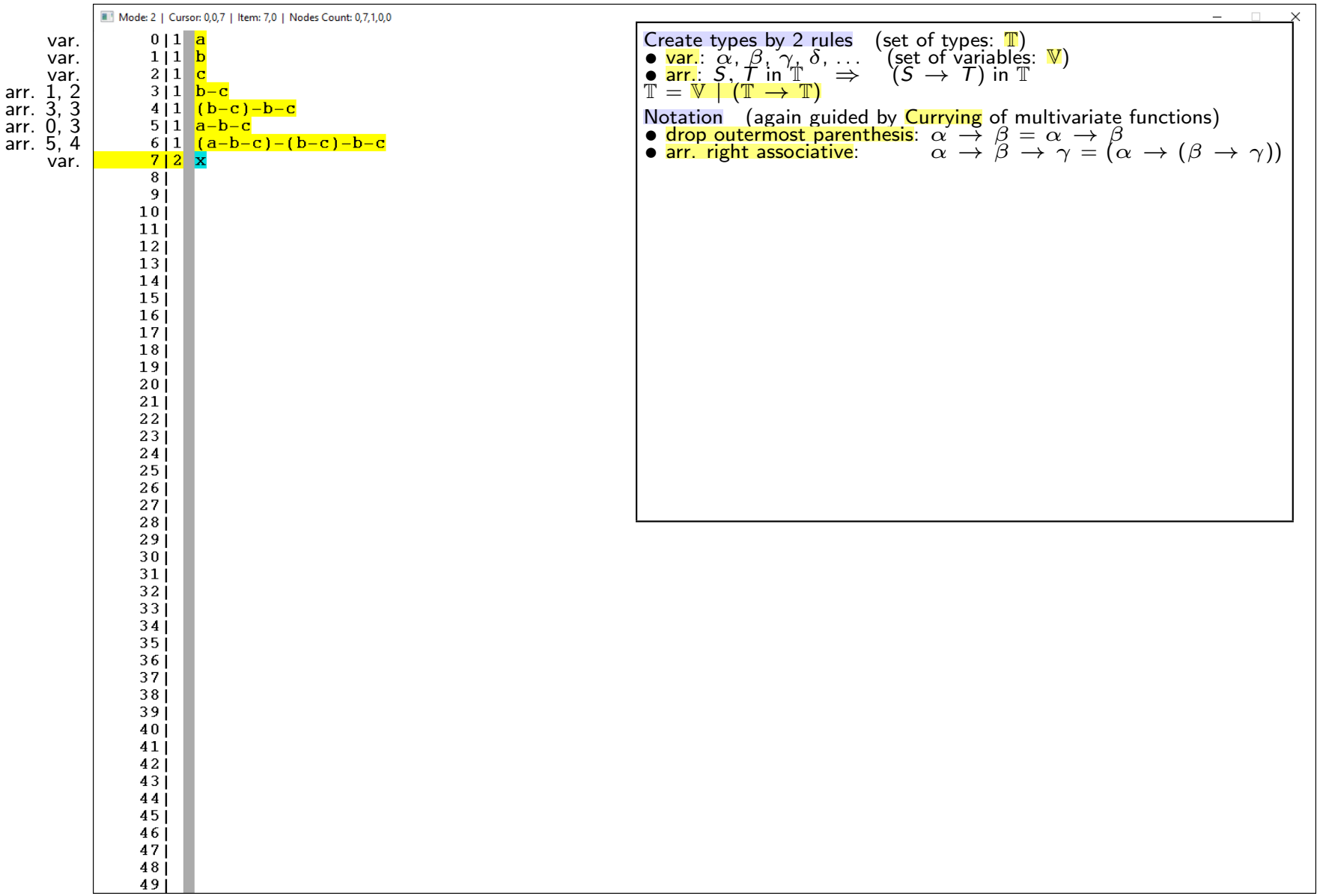

Mode: 1 | Cursor: 0,0,4 | Item: 4,0 | Nodes Count: 0,5,0,0,0
var.
var.
var.
arr. 1, 2
arr. 3, 3
0 | 1 a
1 | 1 b
2 | 1 c
3 | 1 (b-c)
4 | 1 ((b-c)-(b-c))
Create types by 2 rules (set of types: 𝕋)
• var.: α, β, γ, δ, ... (set of variables: 𝕍)
• arr.: S, T in 𝕋 ⇒ (S → T) in 𝕋
𝕋 = 𝕍 | (𝕋 → 𝕋)

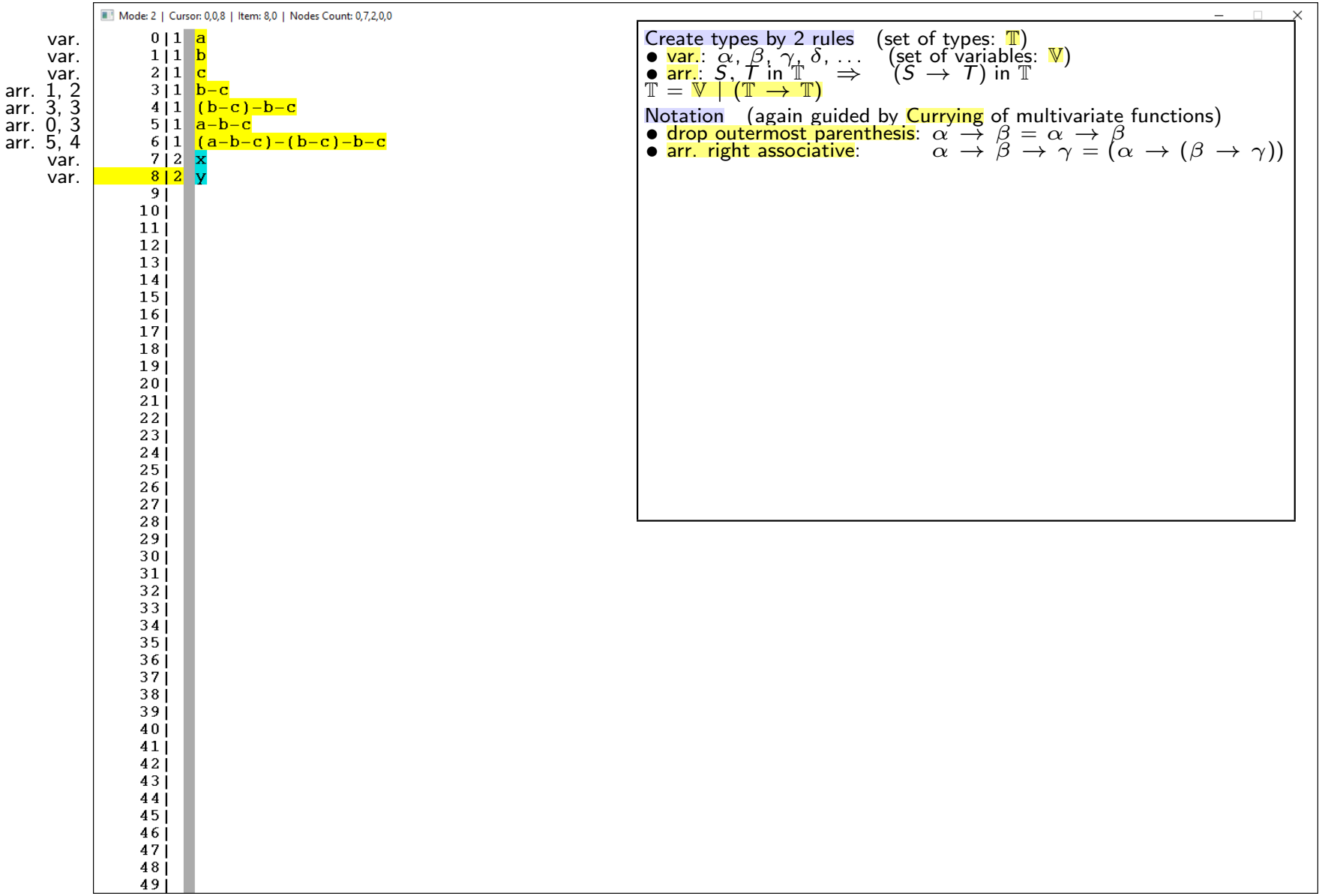

Mode: 1 | Cursor: 0,0,5 | Item: 5,0 | Nodes Count: 0,6,0,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 (b-c)
arr. 3, 3 4 | 1 ((b-c)-(b-c))
arr. 0, 3 5 | 1 (a-(b-c))
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

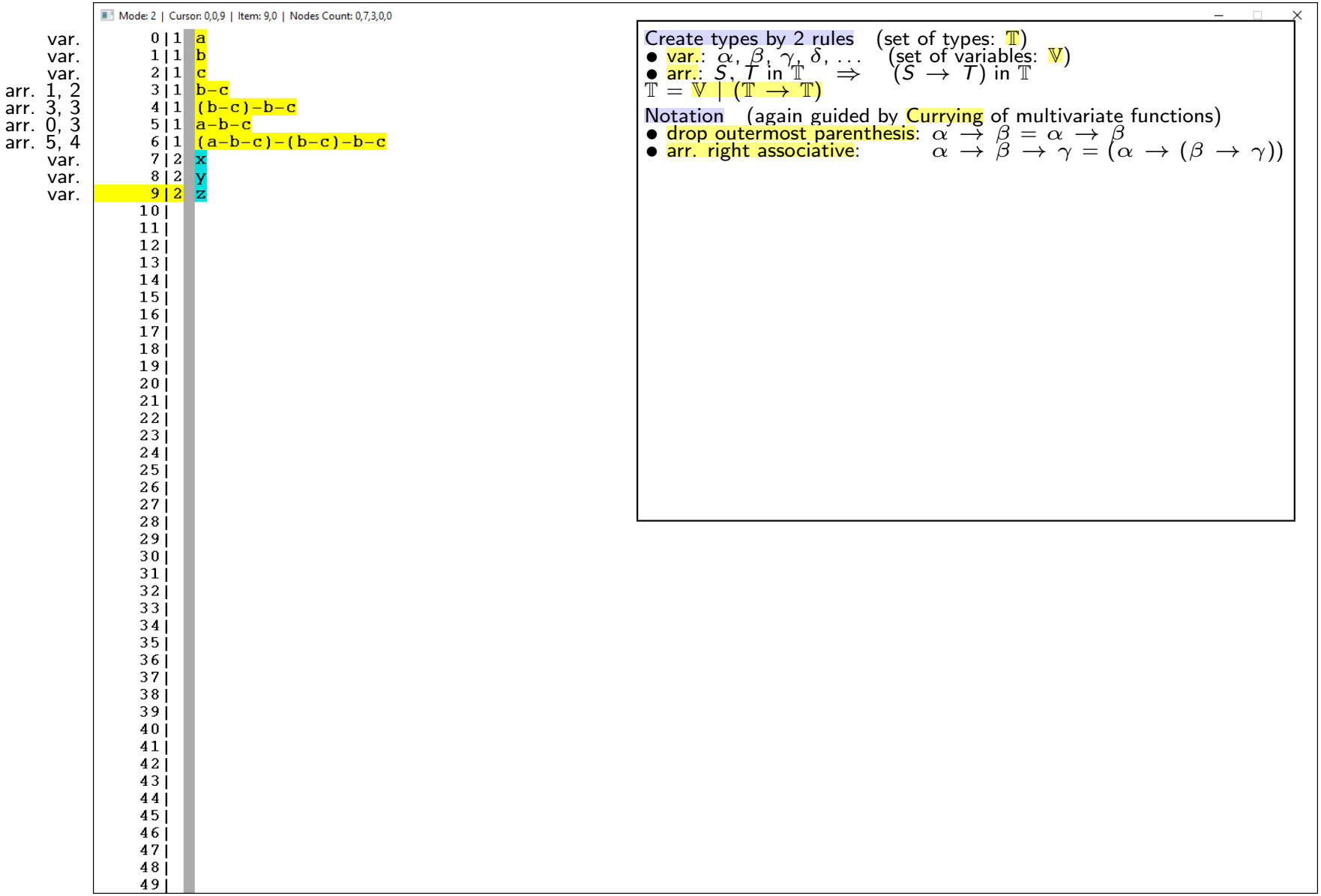

Mode: 1 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 0,7,0,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 (b−c)
arr. 3, 3 4 | 1 ((b−c)−(b−c))
arr. 0, 3 5 | 1 (a−(b−c))
arr. 5, 4 6 | 1 ((a−(b−c))−((b−c)−(b−c)))
Create types by 2 rules (set of types: 𝕋)
• var.: α, β, γ, δ, ... (set of variables: 𝕍)
• arr.: S, T in 𝕋 ⇒ (S → T) in 𝕋
𝕋 = 𝕍 | (𝕋 → 𝕋)

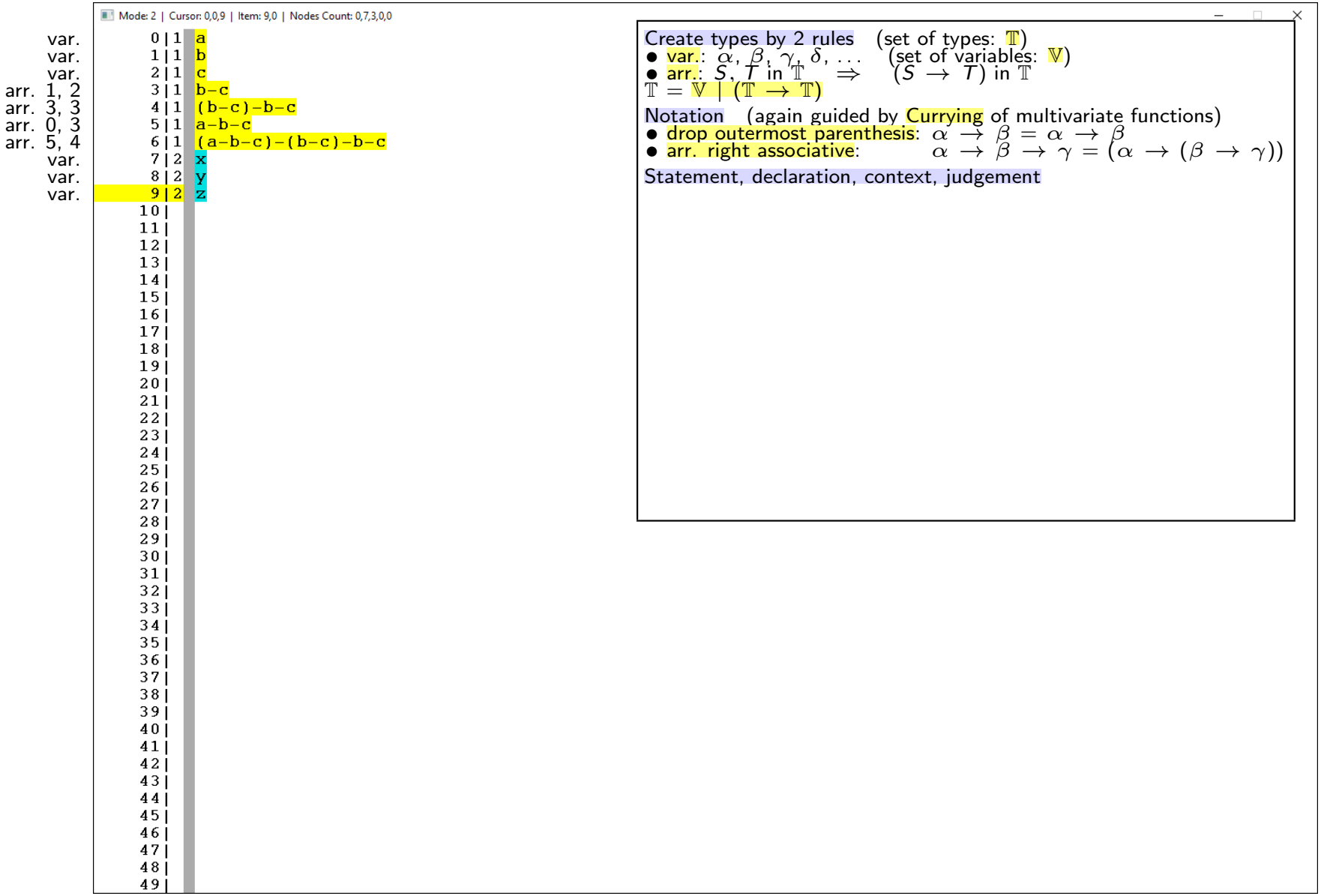

Mode: 1 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 0,7,0,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 (b-c)
arr. 3, 3 4 | 1 ((b-c)-(b-c))
arr. 0, 3 5 | 1 (a-(b-c))
arr. 5, 4 6 | 1 ((a-(b-c))-((b-c)-(b-c)))
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

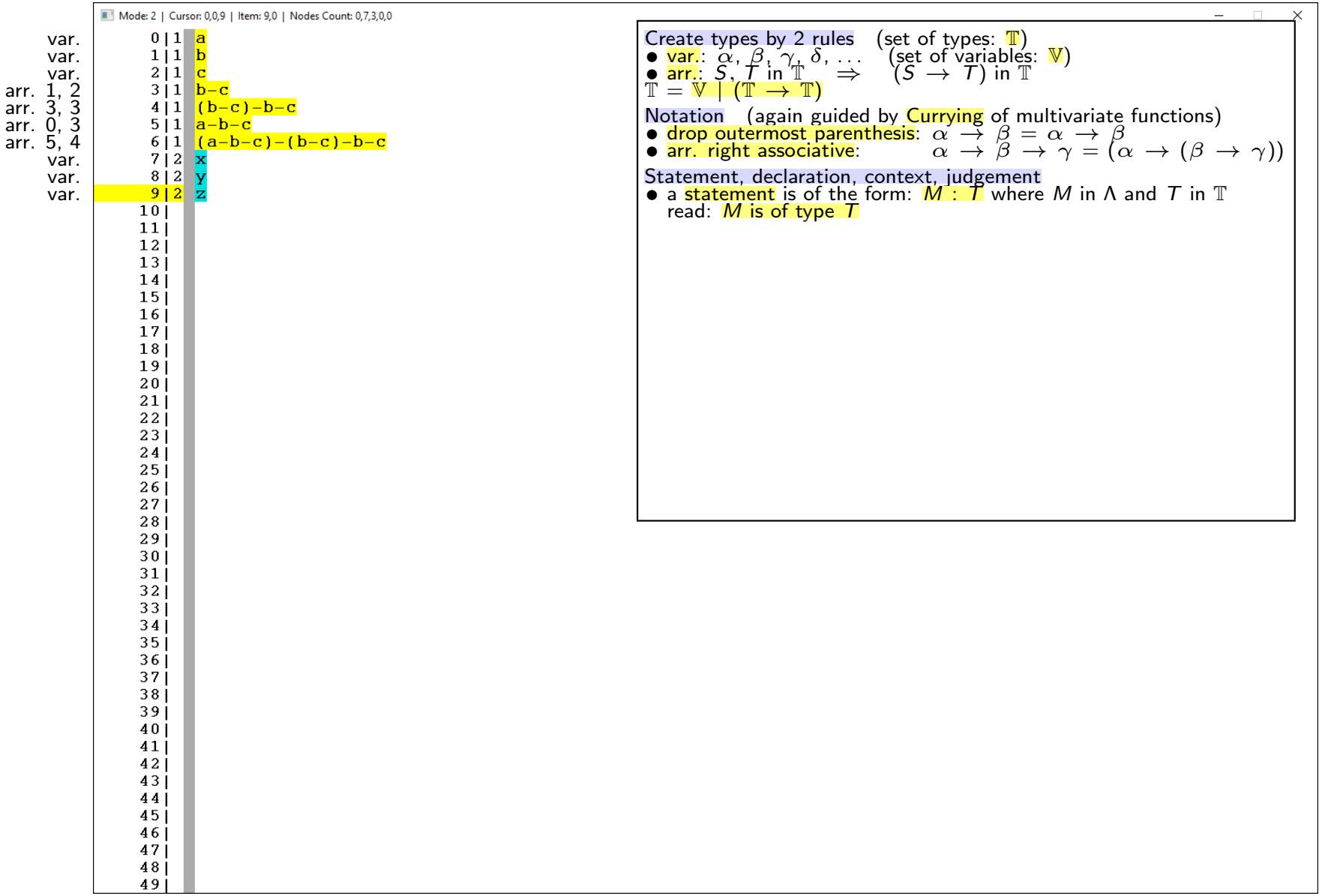

Mode: 1 | Cursor: 0,0,6 | Item: 6,0 | Nodes Count: 0,7,0,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

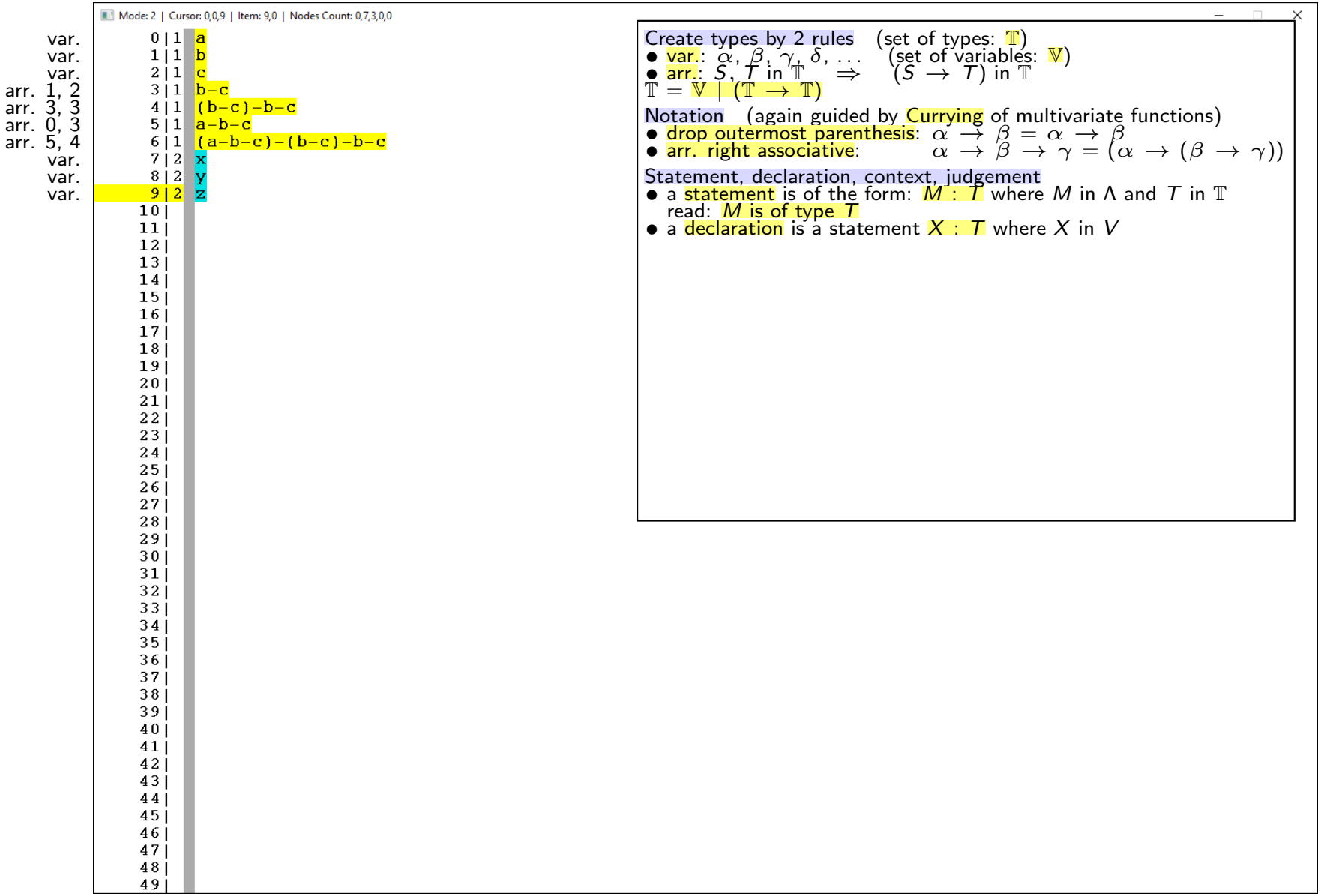

Mode: 2 | Cursor: 0,0,7 | Item: 7,0 | Nodes Count: 0,7,1,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

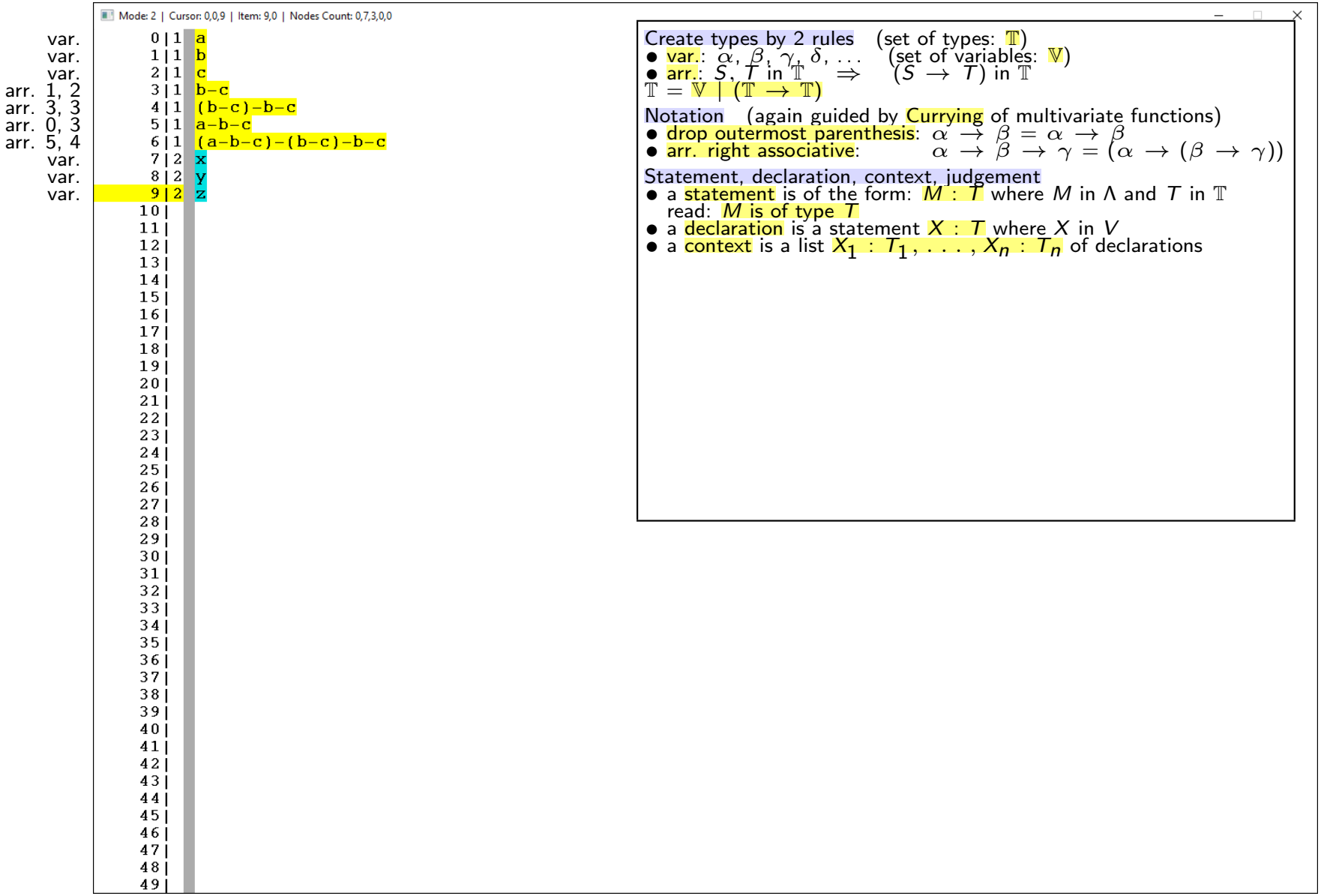

Mode: 2 | Cursor: 0,0,8 | Item: 8,0 | Nodes Count: 0,7,2,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$


Mode: 2 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$


Mode: 2 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement


Mode: 2 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T


Mode: 2 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T
• a declaration is a statement $X : T$ where X in V

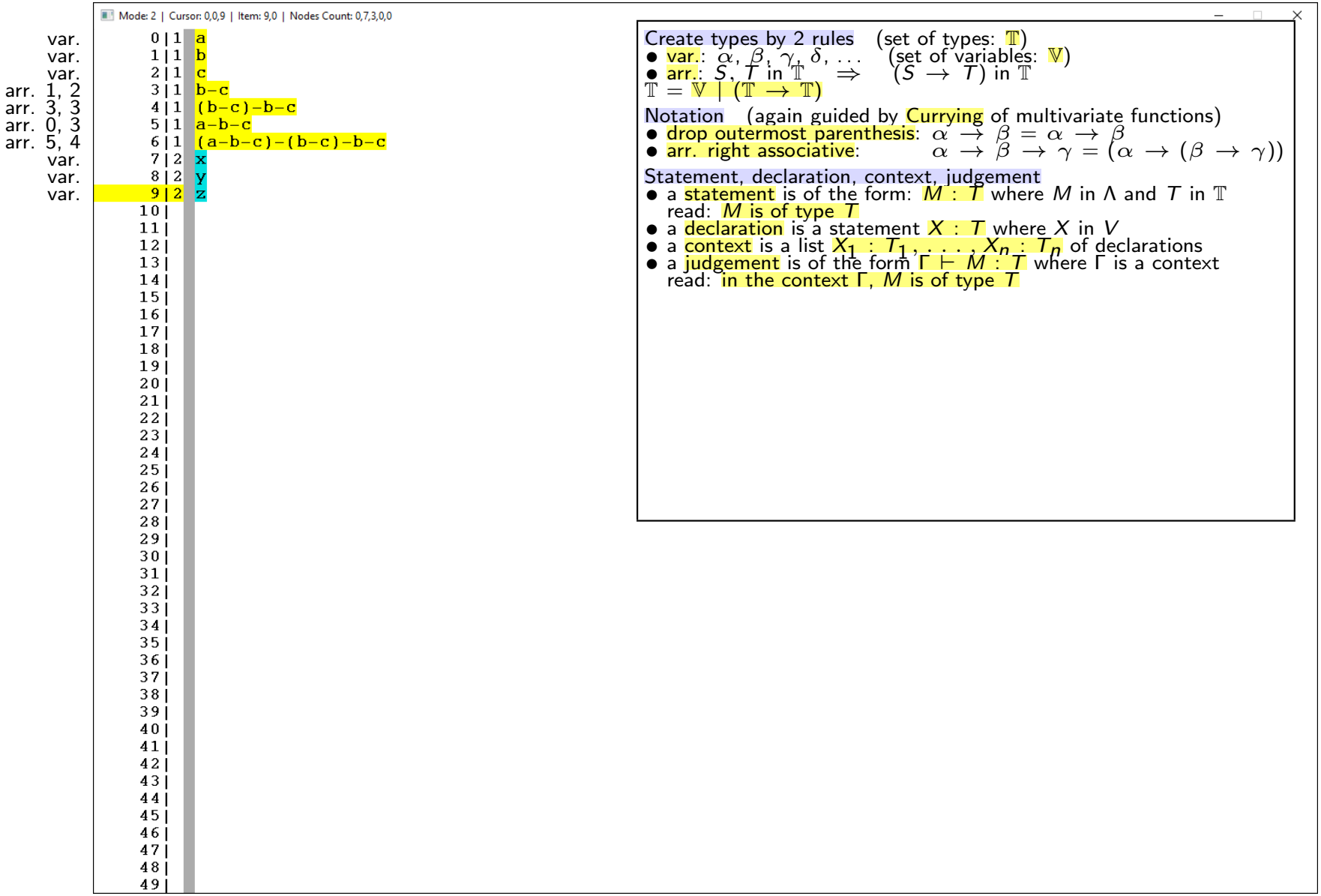


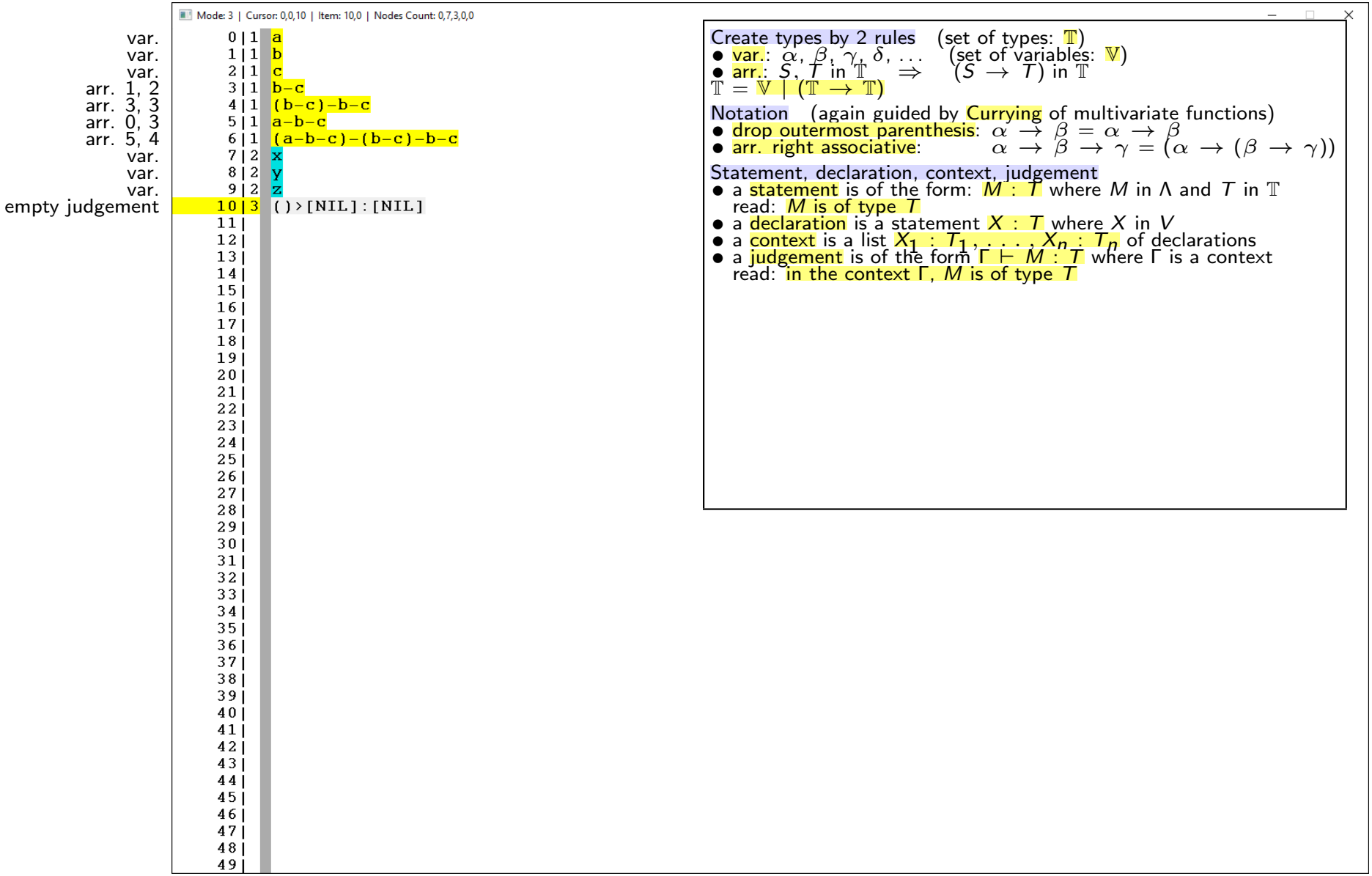
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations


Mode: 2 | Cursor: 0,0,9 | Item: 9,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T
• a declaration is a statement $X : T$ where X in V
• a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
• a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
read: in the context Γ, M is of type T

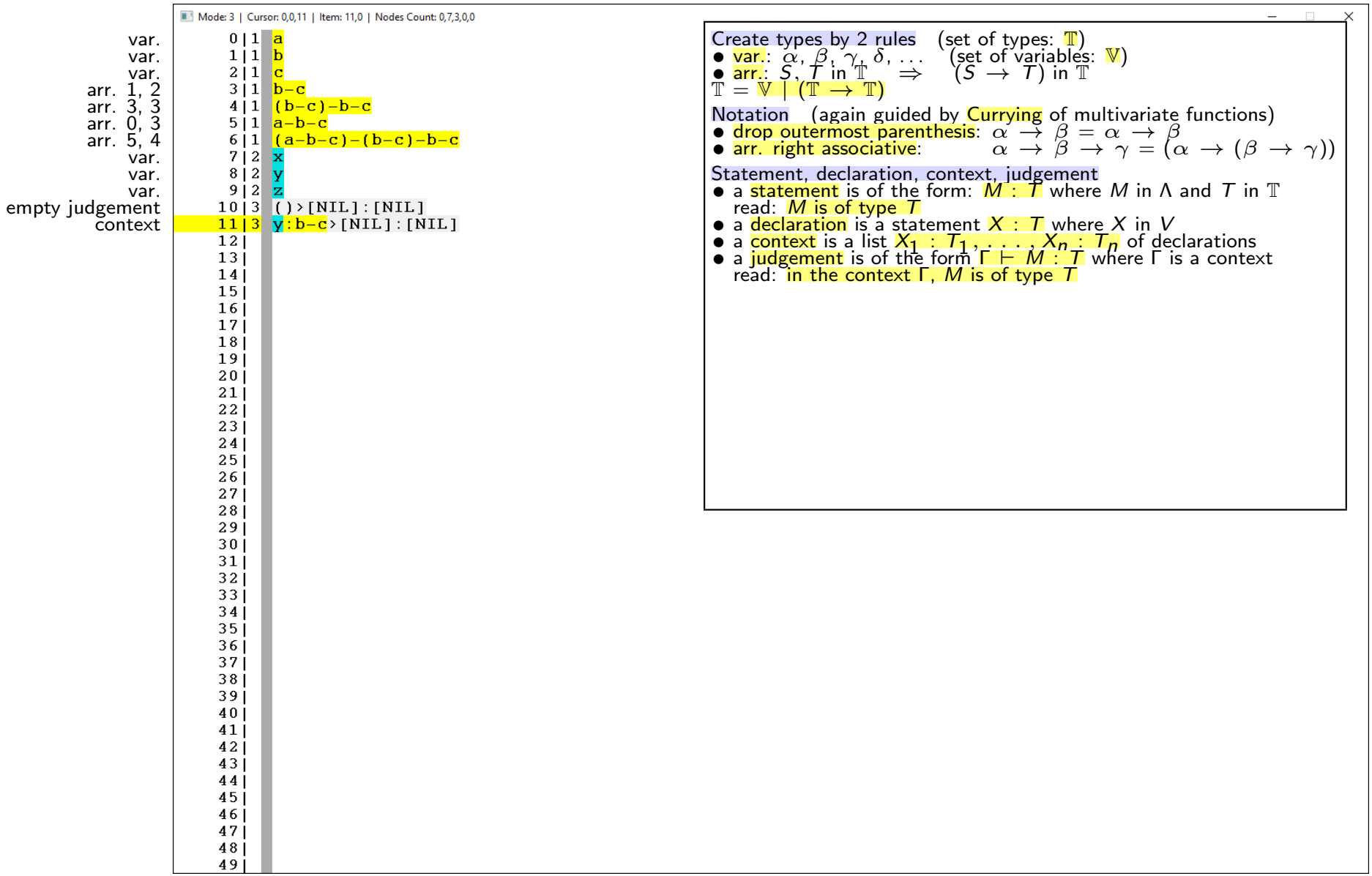


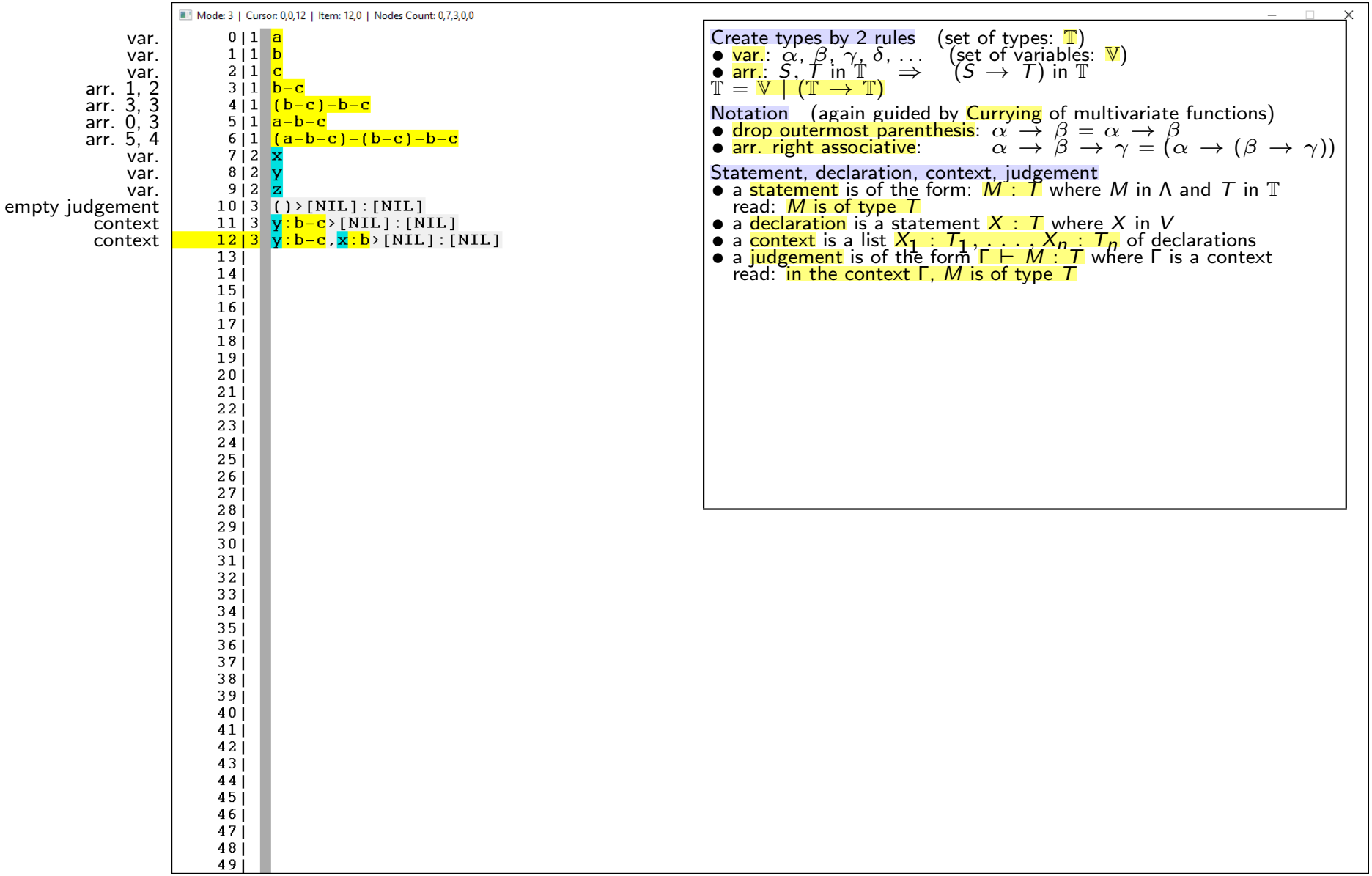
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

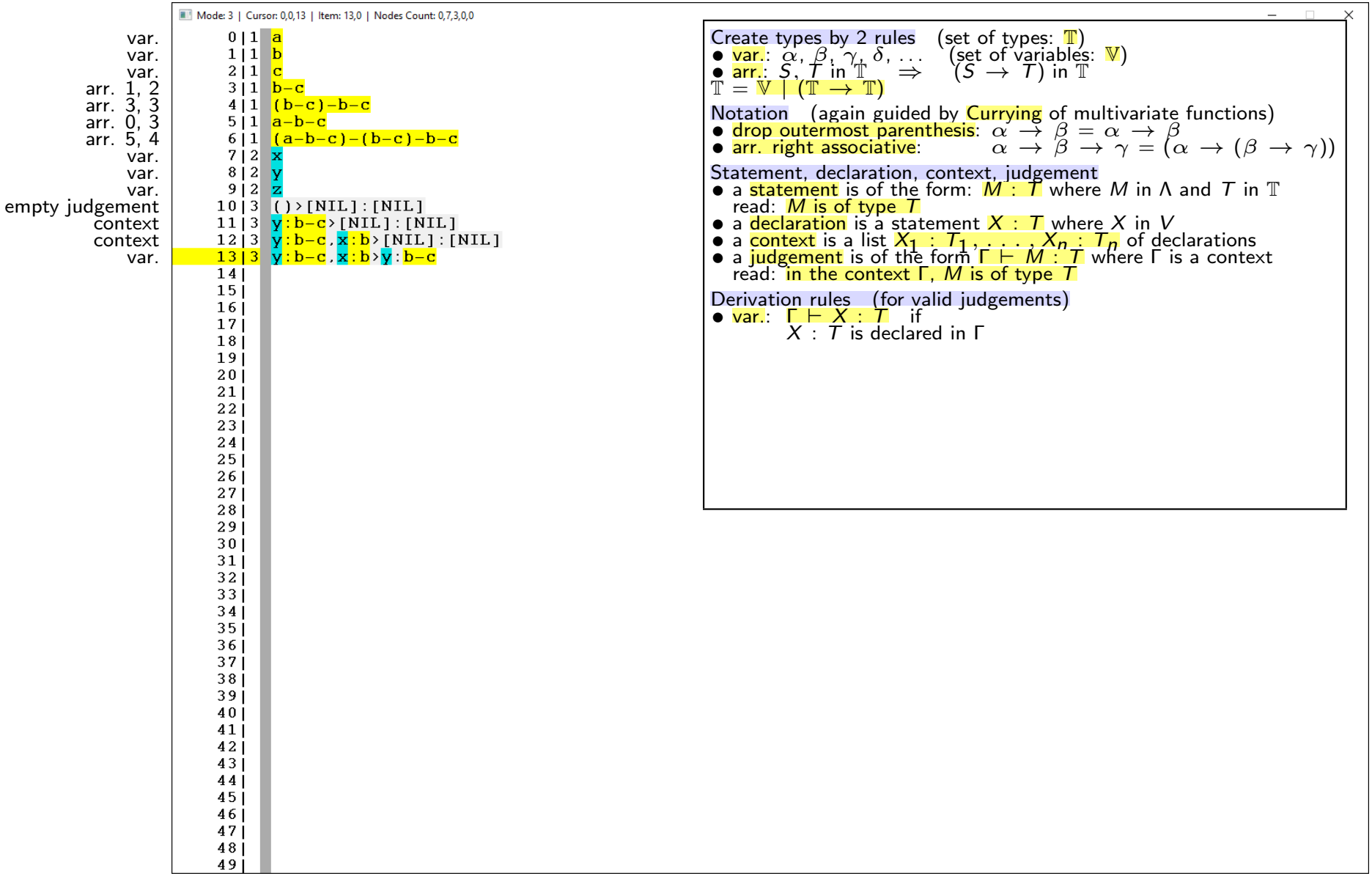
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T


Mode: 3 | Cursor: 0,0,11 | Item: 11,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
empty judgement 10 | 3 ()>[NIL]:[NIL]
context 11 | 3 y:b-c>[NIL]:[NIL]
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T
• a declaration is a statement $X : T$ where X in V
• a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
• a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
read: in the context Γ, M is of type T


Mode: 3 | Cursor: 0,0,12 | Item: 12,0 | Nodes Count: 0,7,3,0,0
var. 0 | 1 a
var. 1 | 1 b
var. 2 | 1 c
arr. 1, 2 3 | 1 b-c
arr. 3, 3 4 | 1 (b-c)-b-c
arr. 0, 3 5 | 1 a-b-c
arr. 5, 4 6 | 1 (a-b-c)-(b-c)-b-c
var. 7 | 2 x
var. 8 | 2 y
var. 9 | 2 z
empty judgement 10 | 3 ()>[NIL]:[NIL]
context 11 | 3 y:b-c>[NIL]:[NIL]
context 12 | 3 y:b-c,x:b>[NIL]:[NIL]
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T
• a declaration is a statement $X : T$ where X in V
• a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
• a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
read: in the context Γ, M is of type T



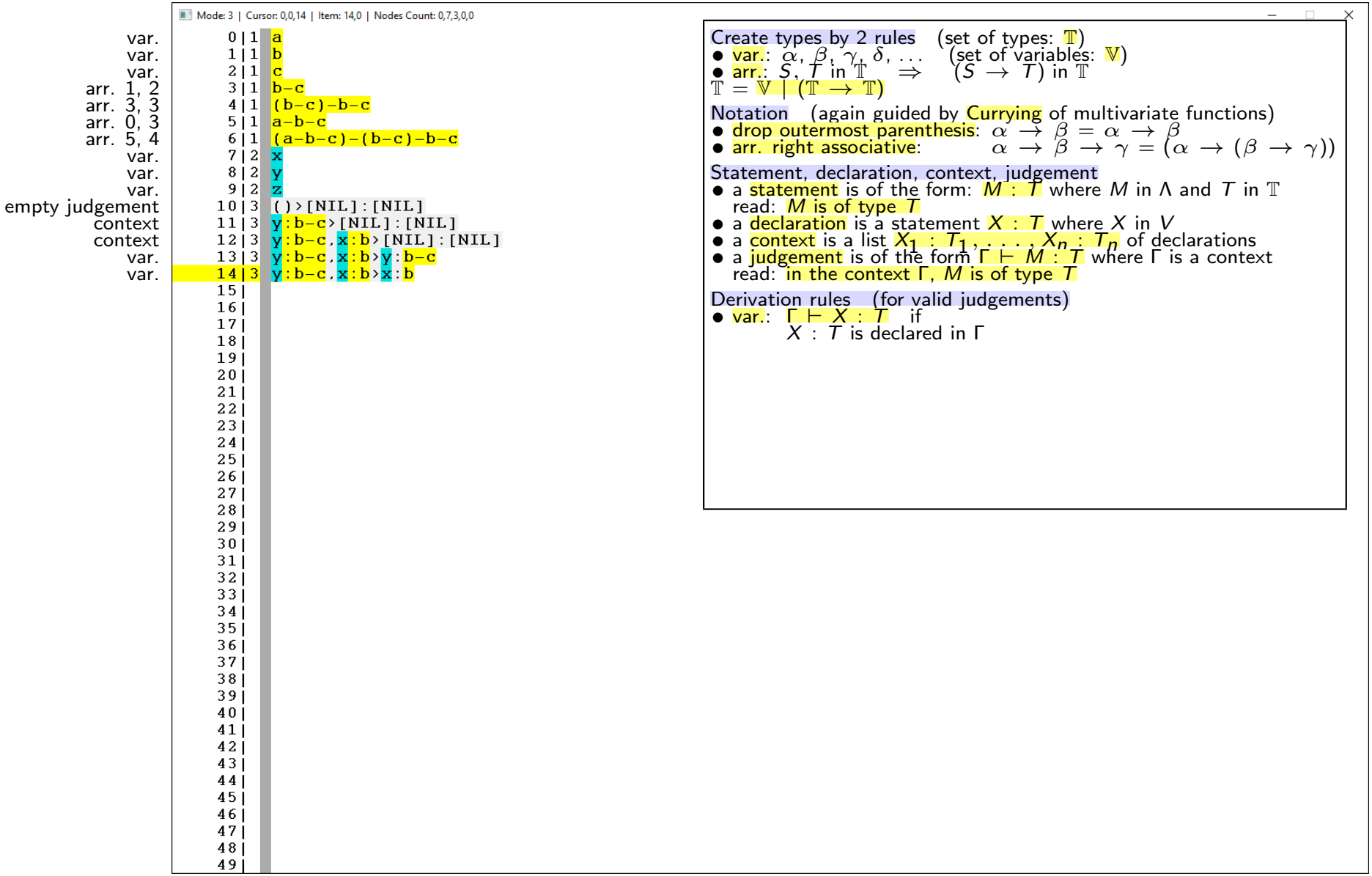
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

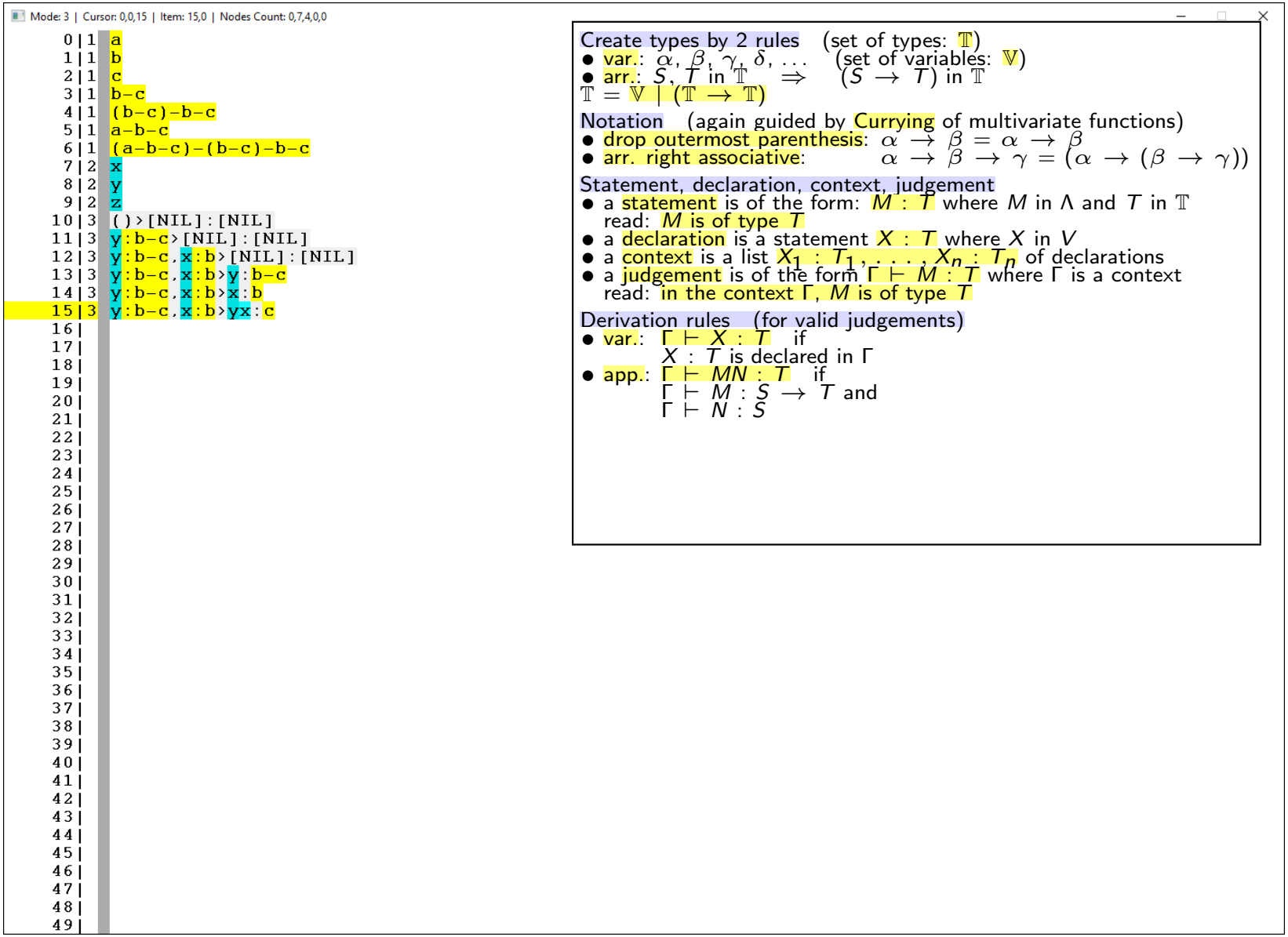
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ


Mode: 3 | Cursor: 0,0,14 | Item: 14,0 | Nodes Count: 0,7,3,0,0
var. 0|1 a
var. 1|1 b
var. 2|1 c
arr. 1, 2 3|1 b-c
arr. 3, 3 4|1 (b-c)-b-c
arr. 0, 3 5|1 a-b-c
arr. 5, 4 6|1 (a-b-c)-(b-c)-b-c
var. 7|2 x
var. 8|2 y
var. 9|2 z
empty judgement 10|3 ()>[NIL]:[NIL]
context 11|3 y:b-c>[NIL]:[NIL]
context 12|3 y:b-c,x:b>[NIL]:[NIL]
var. 13|3 y:b-c,x:b>y:b-c
var. 14|3 y:b-c,x:b>x:b
Create types by 2 rules (set of types: $\mathbb{T}$)
• var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
• arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$
Notation (again guided by Currying of multivariate functions)
• drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
• arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$
Statement, declaration, context, judgement
• a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
read: M is of type T
• a declaration is a statement $X : T$ where X in V
• a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
• a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
read: in the context Γ, M is of type T
Derivation rules (for valid judgements)
• var.: $\Gamma \vdash X : T$ if
$X : T$ is declared in Γ



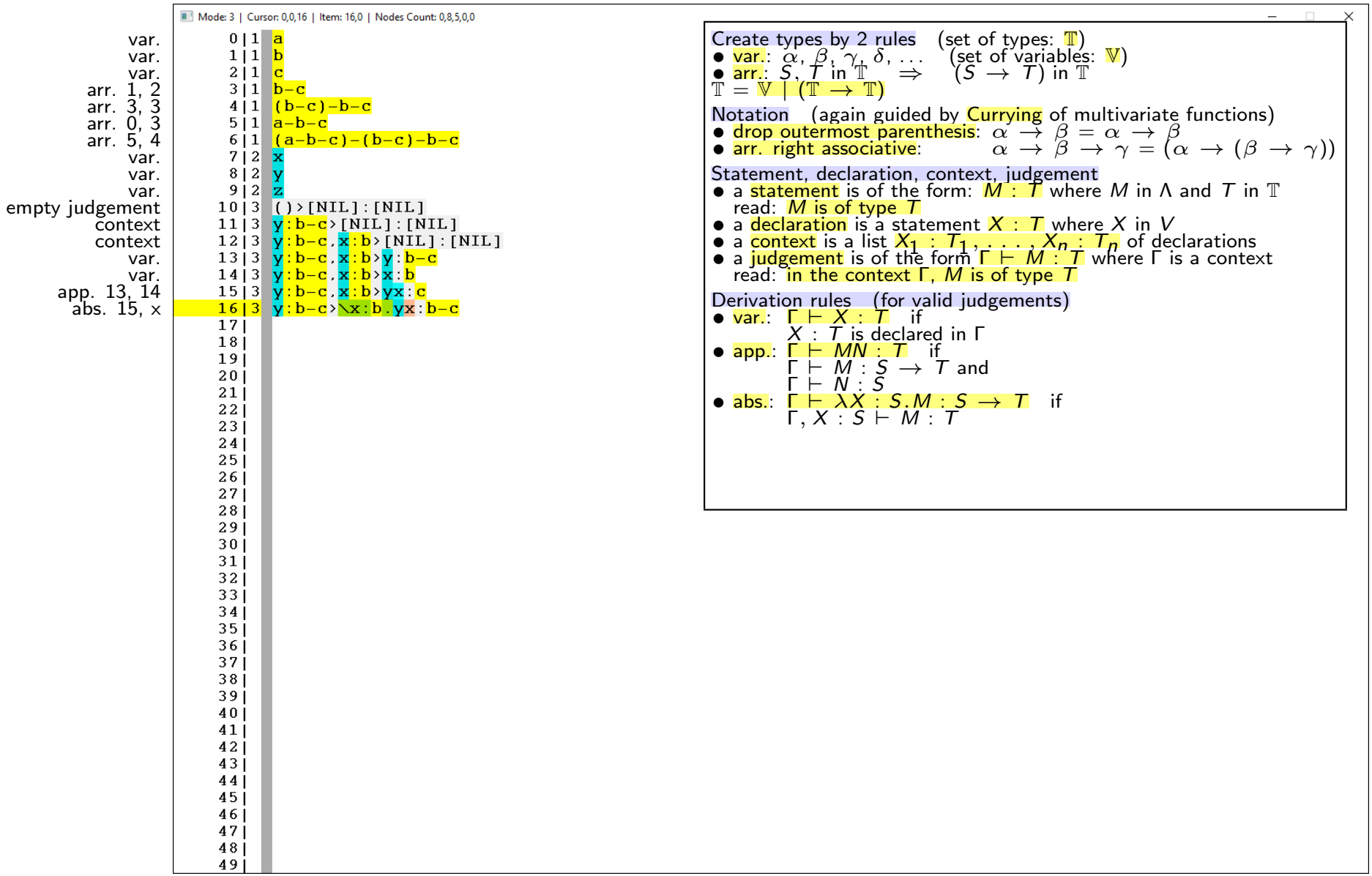
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$



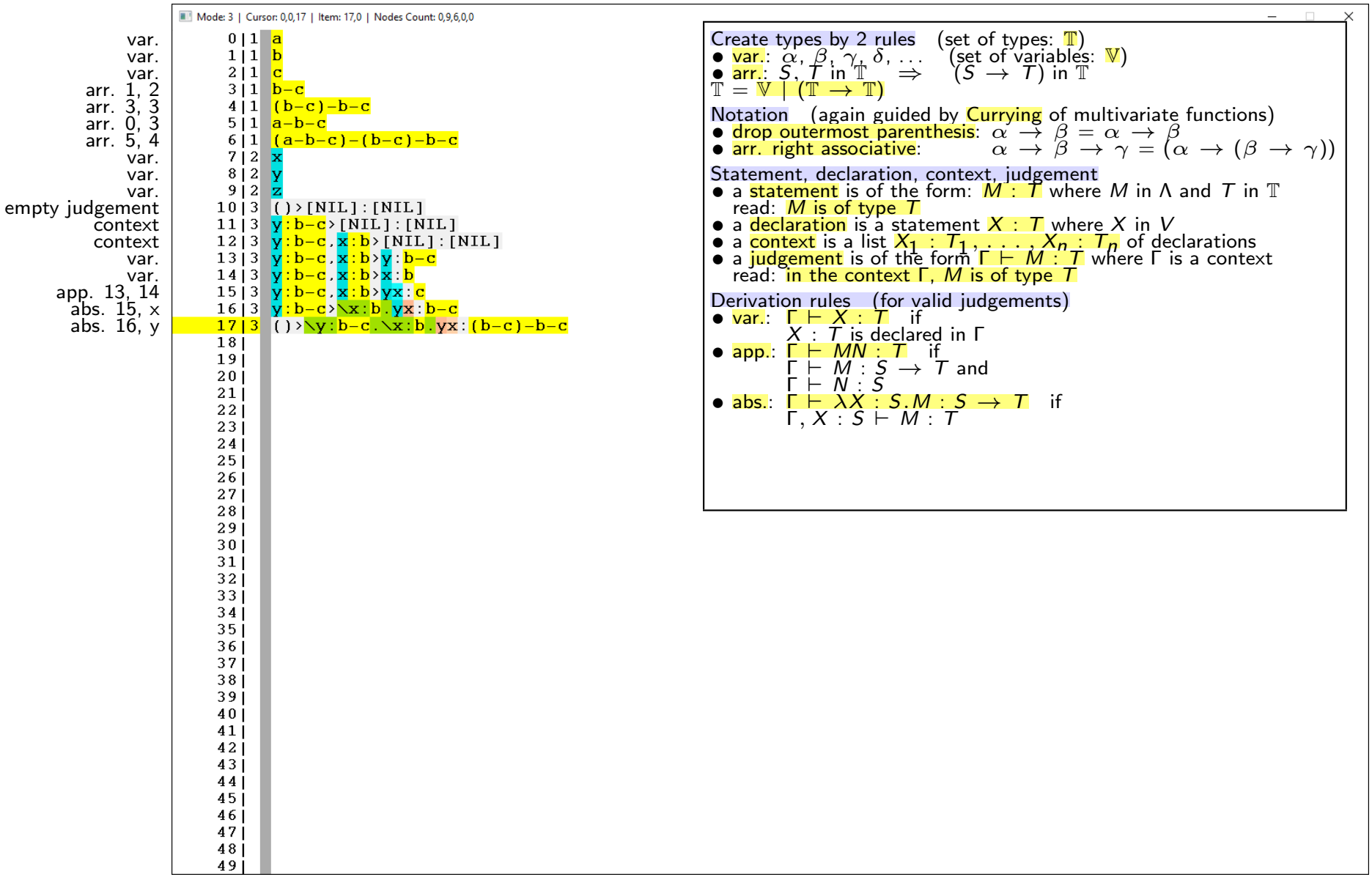
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$



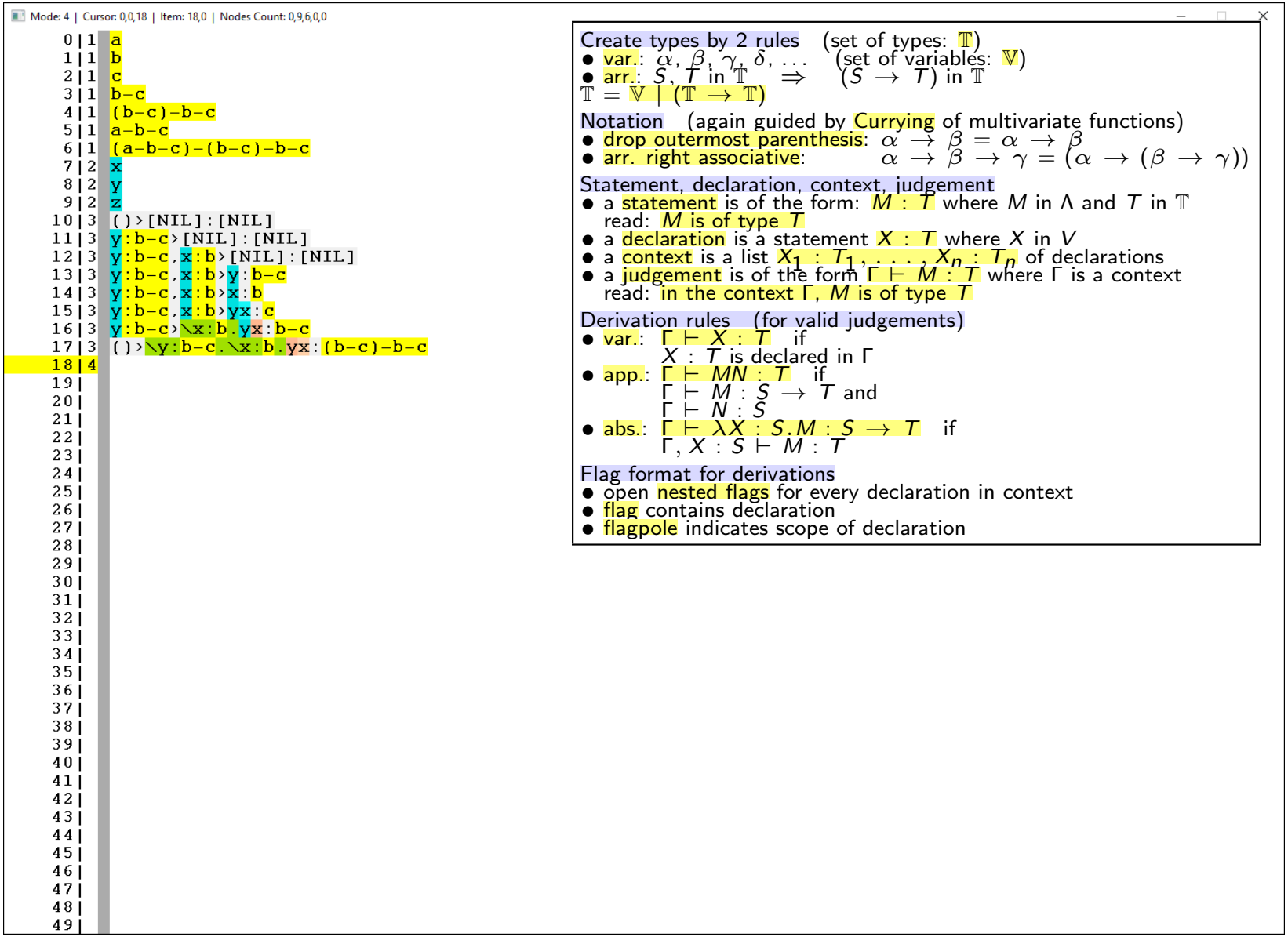
Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

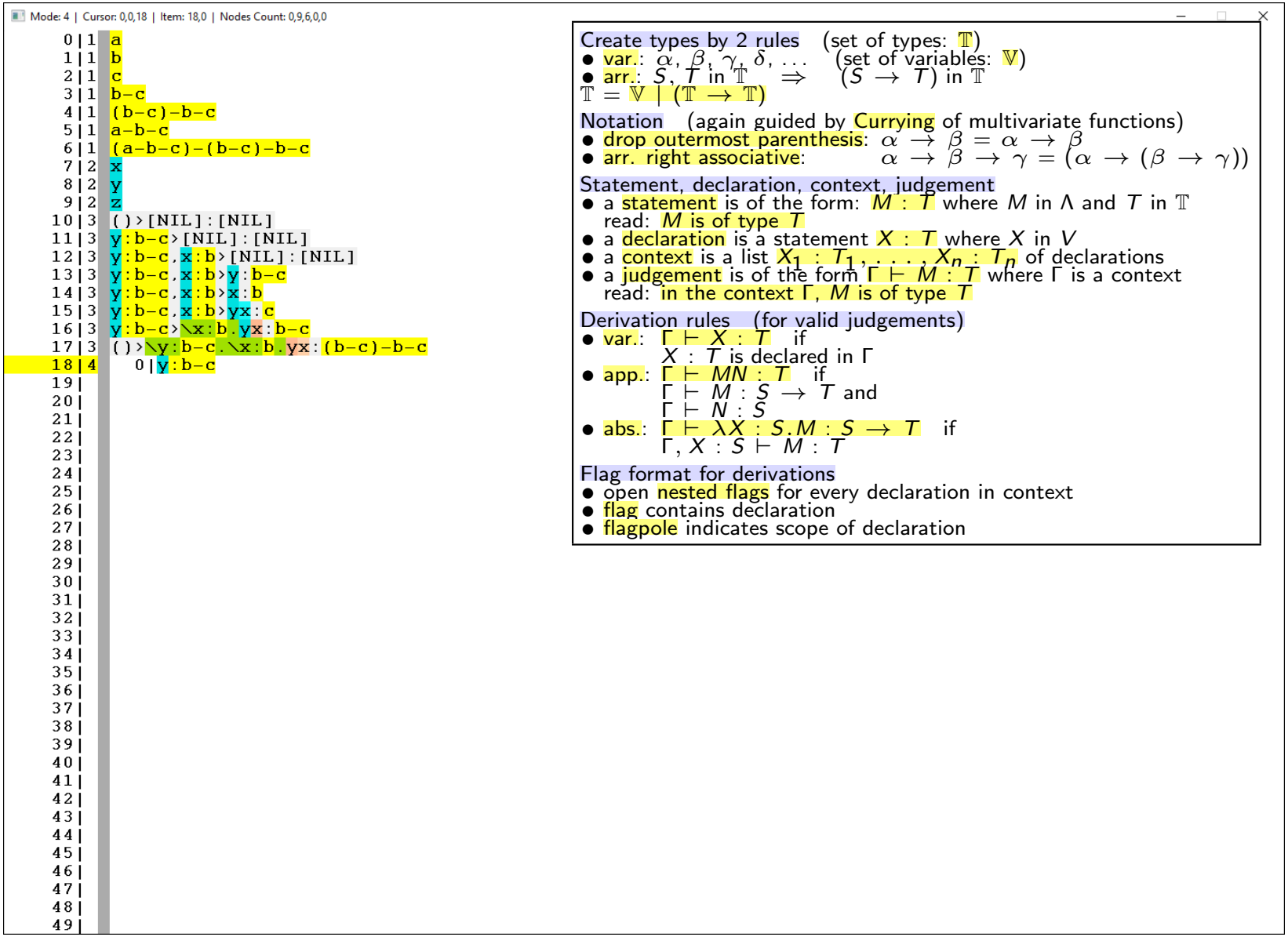
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

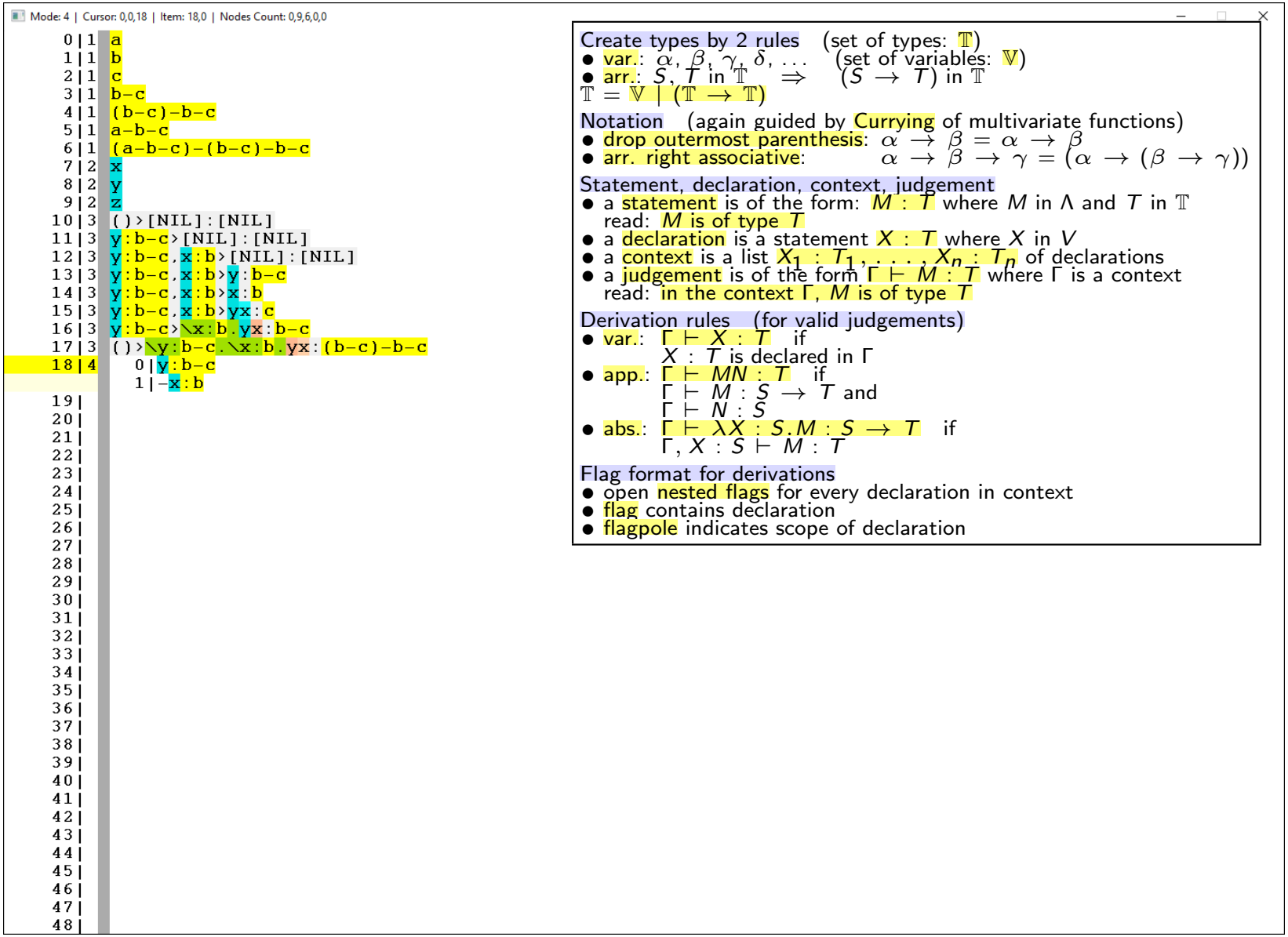
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

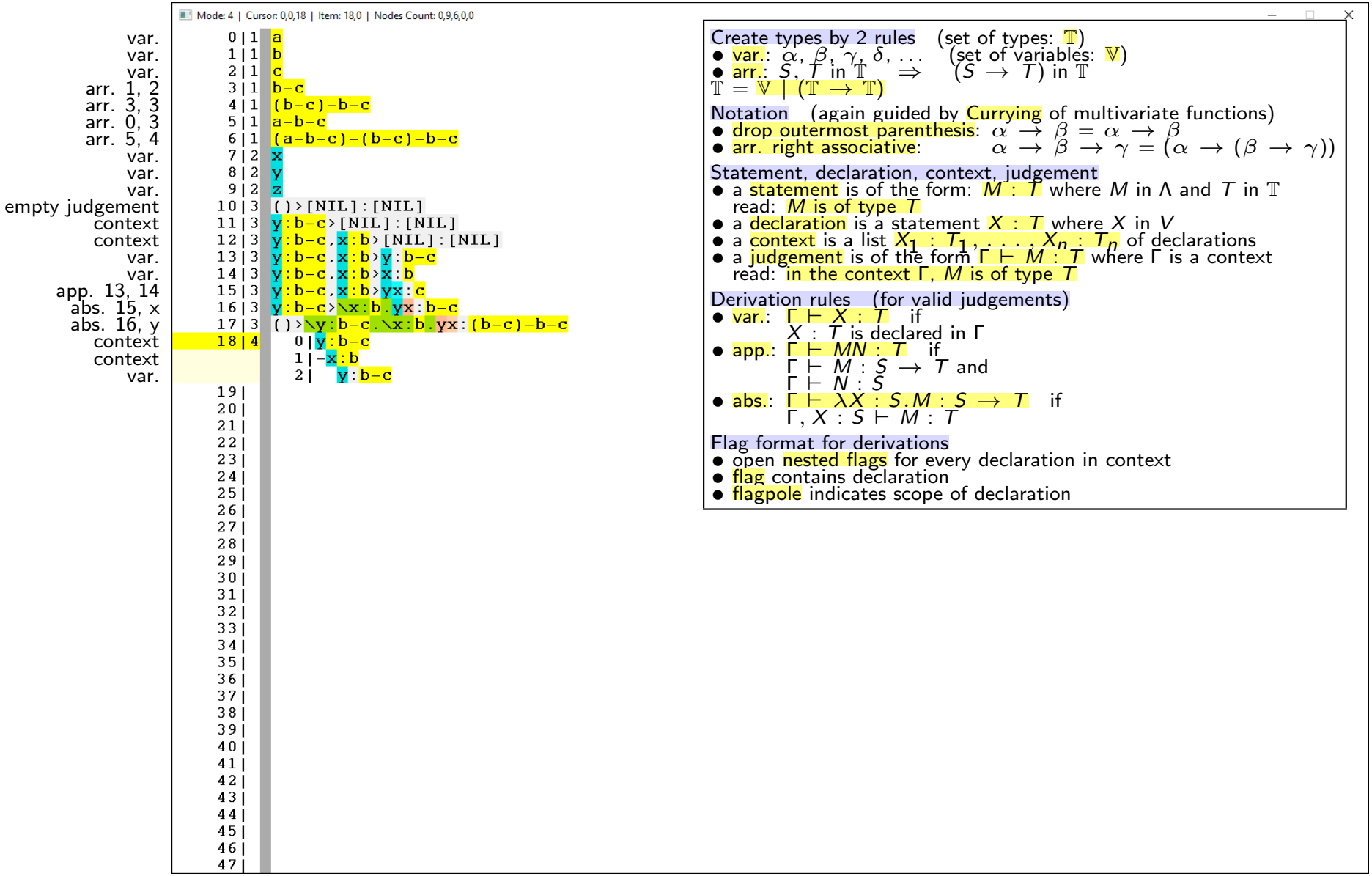
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

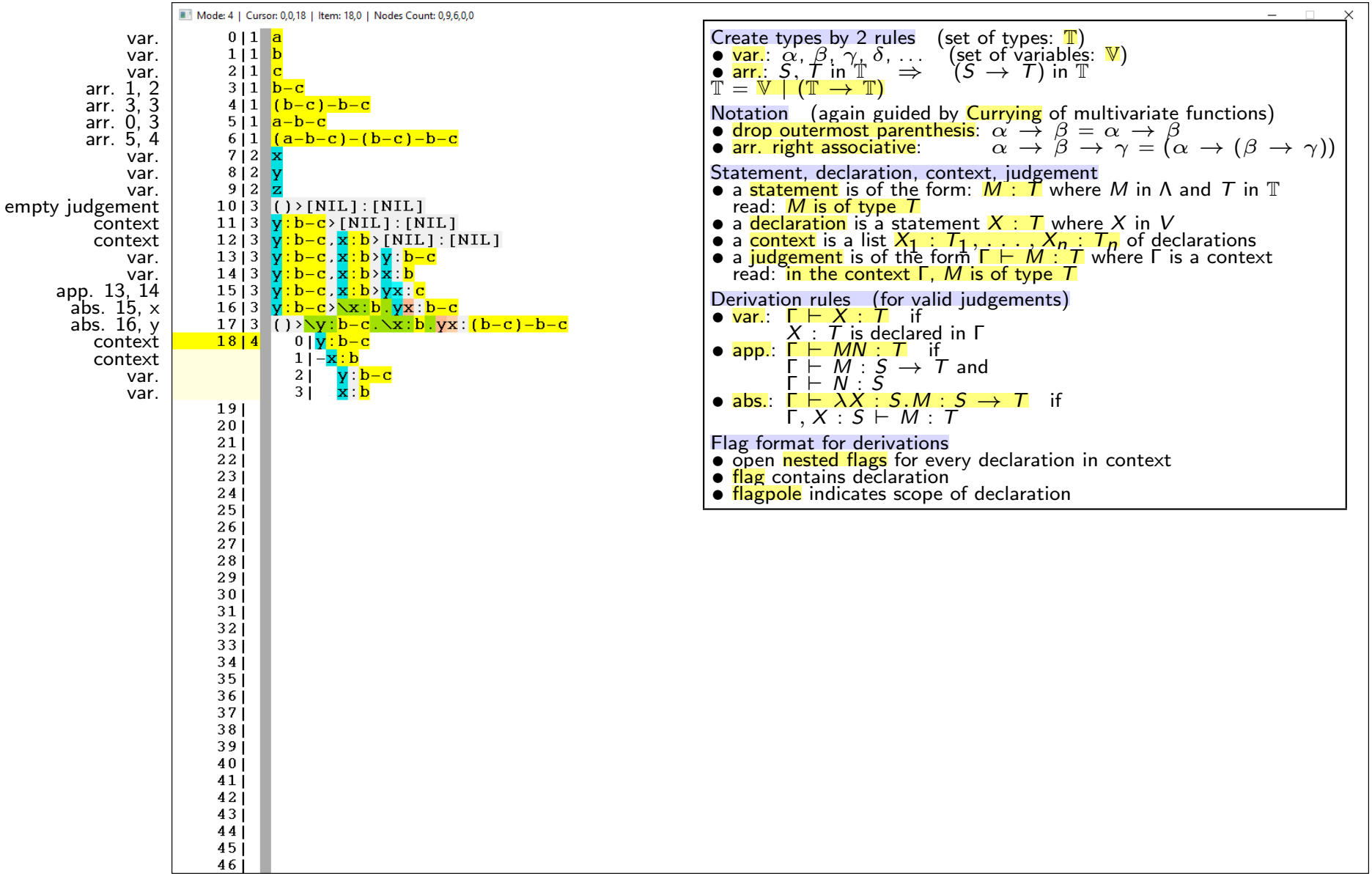
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

```
                                                                                  Mode: 4 | Cursor: 0,0,18 | Item: 18,0 | Nodes Count: 0,9,7,0,0
                                                                                  0|1 a
                                                                                  1|1 b
                                                                                  2|1 c
                                                                                  3|1 b-c
                                                                                  4|1 (b-c)-b-c
                                                                                  5|1 a-b-c
                                                                                  6|1 (a-b-c)-(b-c)-b-c
                                                                                  7|2 x
                                                                                  8|2 y
                                                                                  9|2 z
                                                                                 10|3 ()>[NIL]:[NIL]
                                                                                 11|3 y:b-c>[NIL]:[NIL]
                                                                                 12|3 y:b-c,x:b>[NIL]:[NIL]
                                                                                 13|3 y:b-c,x:b>y:b-c
                                                                                 14|3 y:b-c,x:b>x:b
                                                                                 15|3 y:b-c,x:b>yx:c
                                                                                 16|3 y:b-c>\x:b.yx:b-c
                                                                                 17|3 ()>\y:b-c.\x:b.yx:(b-c)-b-c
                                                                                 18|4   0|y:b-c
                                                                                        1|-x:b
                                                                                        2|  y:b-c
                                                                                        3|  x:b
                                                                                        4|  yx:c
                                                                                 19|
                                                                                 20|
                                                                                 21|
                                                                                 22|
                                                                                 23|
                                                                                 24|
                                                                                 25|
                                                                                 26|
                                                                                 27|
                                                                                 28|
                                                                                 29|
                                                                                 30|
                                                                                 31|
                                                                                 32|
                                                                                 33|
                                                                                 34|
                                                                                 35|
                                                                                 36|
                                                                                 37|
                                                                                 38|
                                                                                 39|
                                                                                 40|
                                                                                 41|
                                                                                 42|
                                                                                 43|
                                                                                 44|
                                                                                 45|
```

var.
var.
var.
arr. 1, 2
arr. 3, 3
arr. 0, 3
arr. 5, 4
var.
var.
var.
empty judgement
context
context
var.
var.
app. 13, 14
abs. 15, x
abs. 16, y
context
context
var.
var.
app. 2, 3

Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

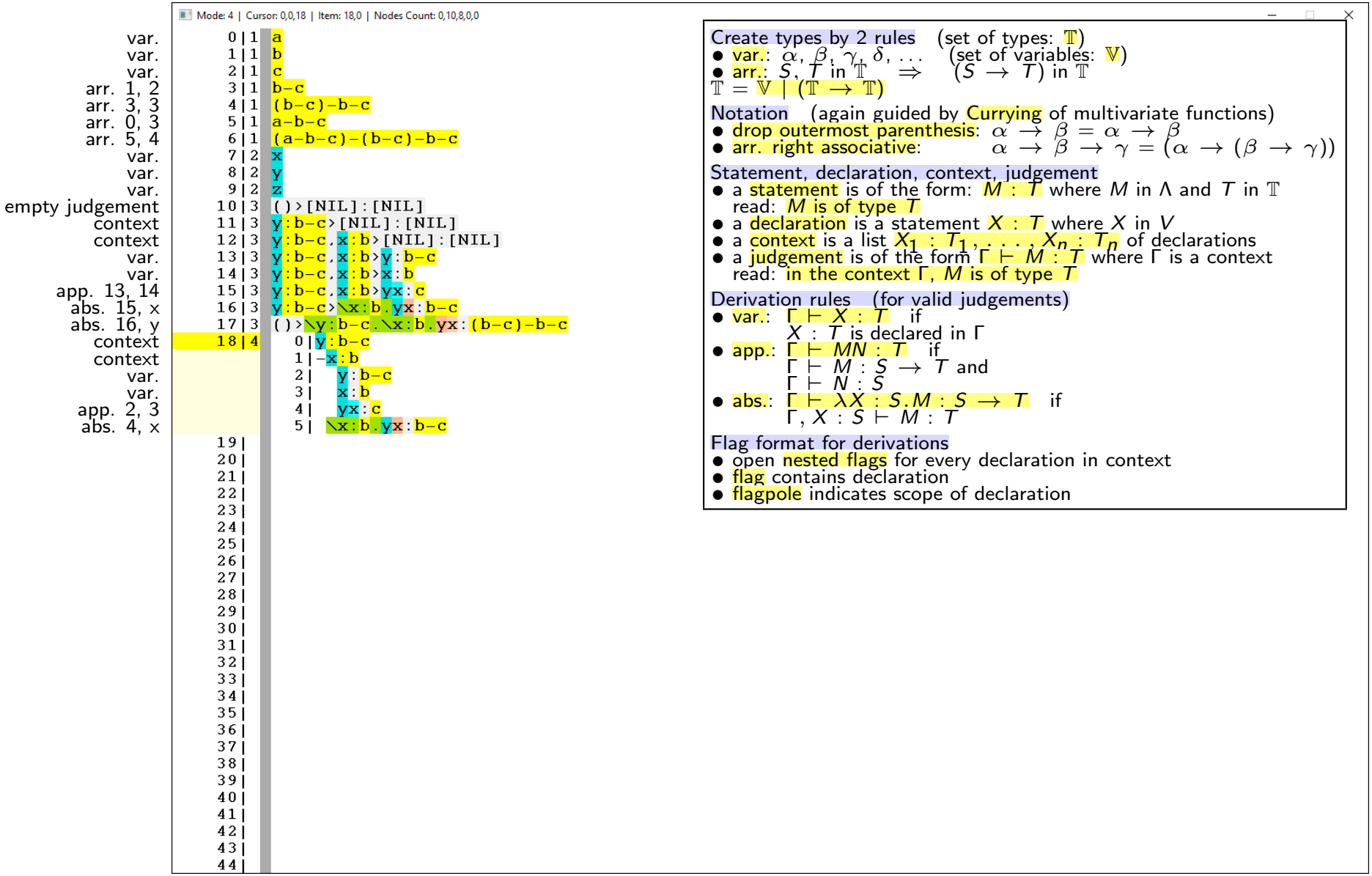
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)

- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)

- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement

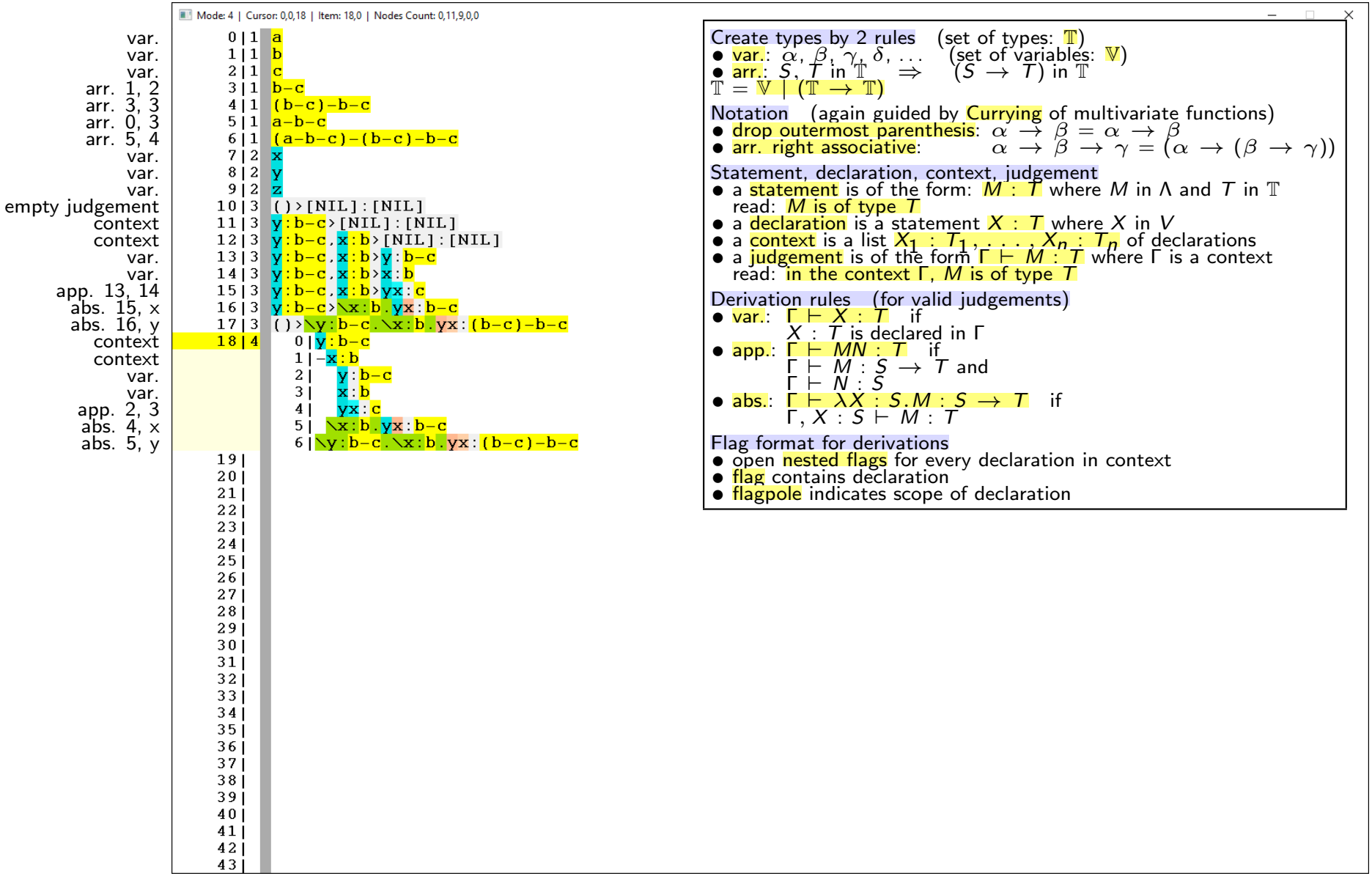
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)

- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations

- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

```
Mode: 1 | Cursor: 0,0,25 | Item: 19,0 | Nodes Count: 0,12,9,0,0
```

var.	0	1	a
var.	1	1	b
var.	2	1	c
arr. 1, 2	3	1	b−c
arr. 3, 3	4	1	(b−c)−b−c
arr. 0, 3	5	1	a−b−c
arr. 5, 4	6	1	(a−b−c)−(b−c)−b−c
var.	7	2	x
var.	8	2	y
var.	9	2	z
empty judgement	10	3	()›[NIL]:[NIL]
context	11	3	y:b−c›[NIL]:[NIL]
context	12	3	y:b−c,x:b›[NIL]:[NIL]
var.	13	3	y:b−c,x:b›y:b−c
var.	14	3	y:b−c,x:b›x:b
app. 13, 14	15	3	y:b−c,x:b›yx:c
abs. 15, x	16	3	y:b−c›\x:b.yx:b−c
abs. 16, y	17	3	()›\y:b−c.\x:b.yx:(b−c)−b−c
context	18	4	0|y:b−c
context			1|−x:b
var.			2| y:b−c
var.			3| x:b
app. 2, 3			4| yx:c
abs. 4, x			5| \x:b.yx:b−c
abs. 5, y			6|\y:b−c.\x:b.yx:(b−c)−b−c
arr. 0, 2	19	1	a−c
	20		
	21		
	22		
	23		
	24		
	25		
	26		
	27		
	28		
	29		
	30		
	31		
	32		
	33		
	34		
	35		
	36		
	37		
	38		
	39		
	40		
	41		
	42		
	43		

Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

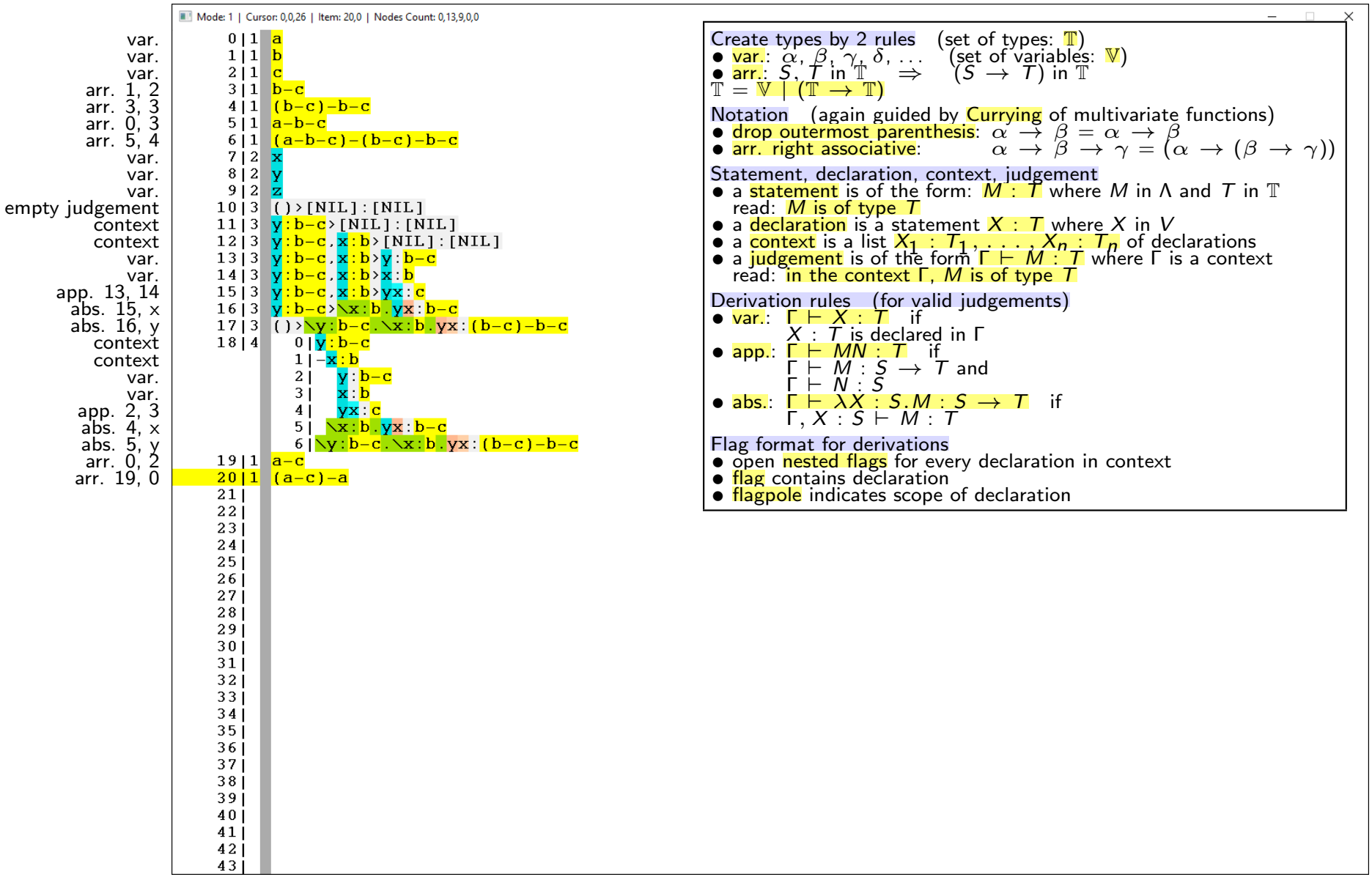
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

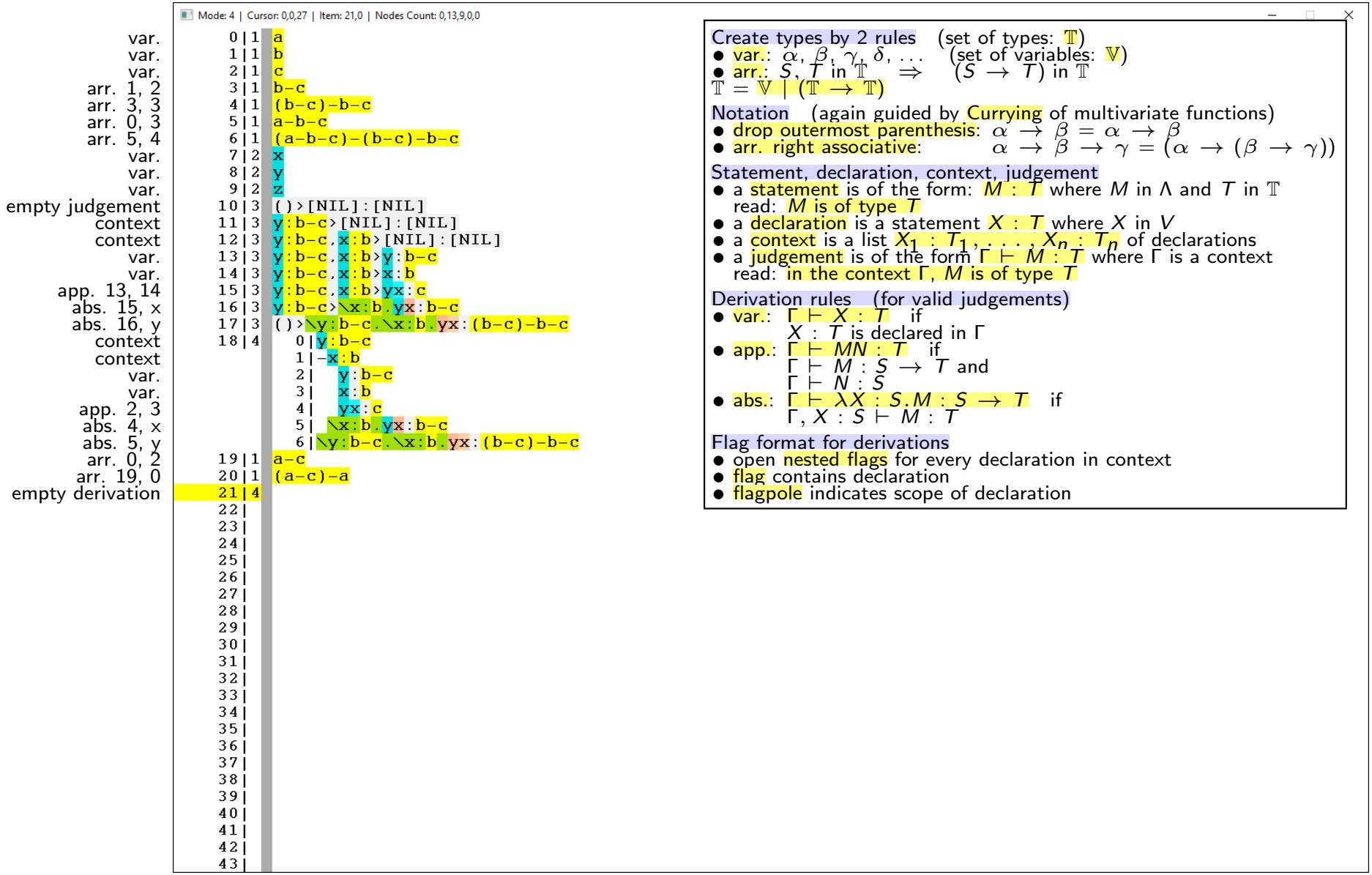
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

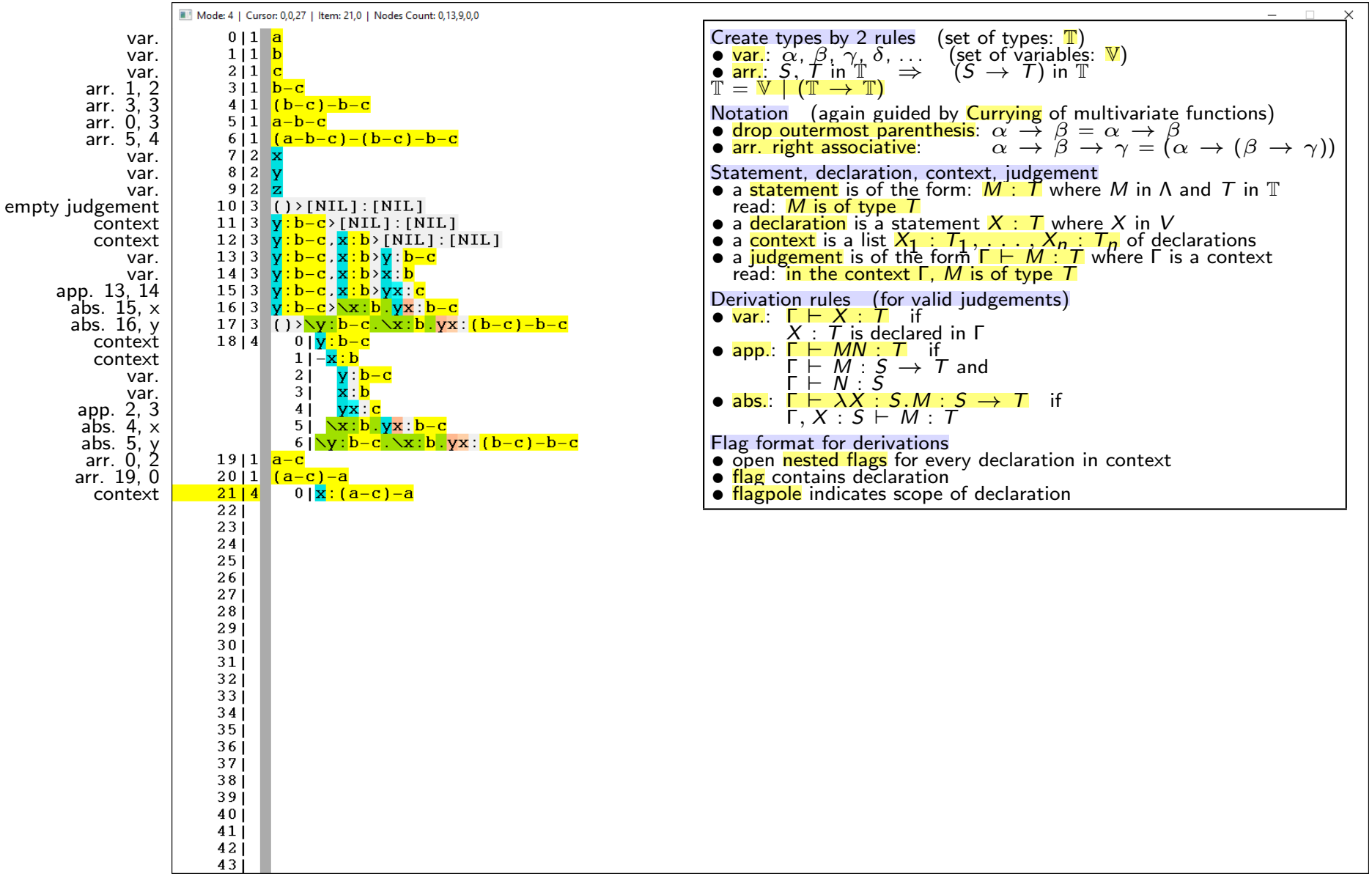
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

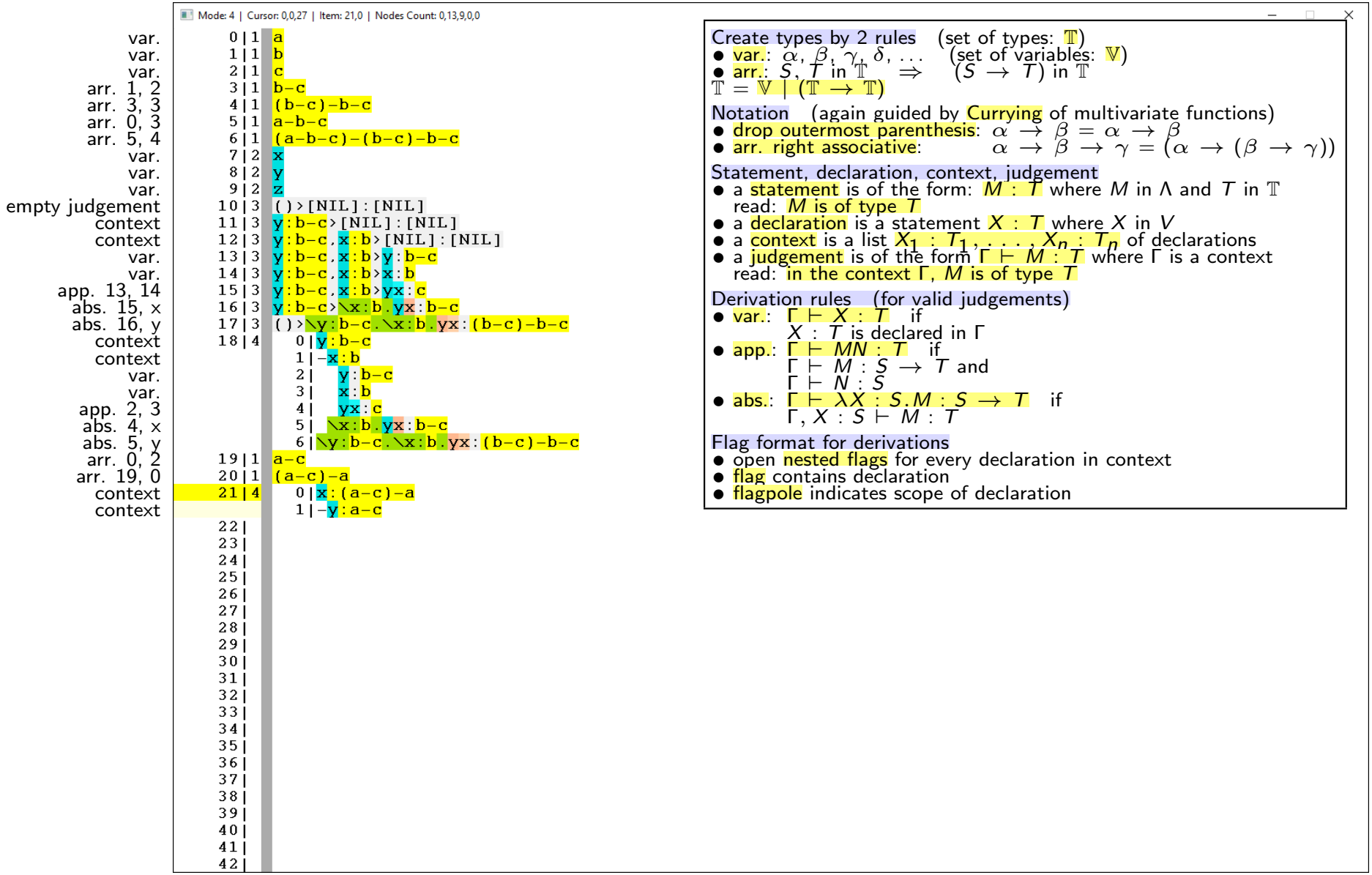
Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

A crash course in type theory



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

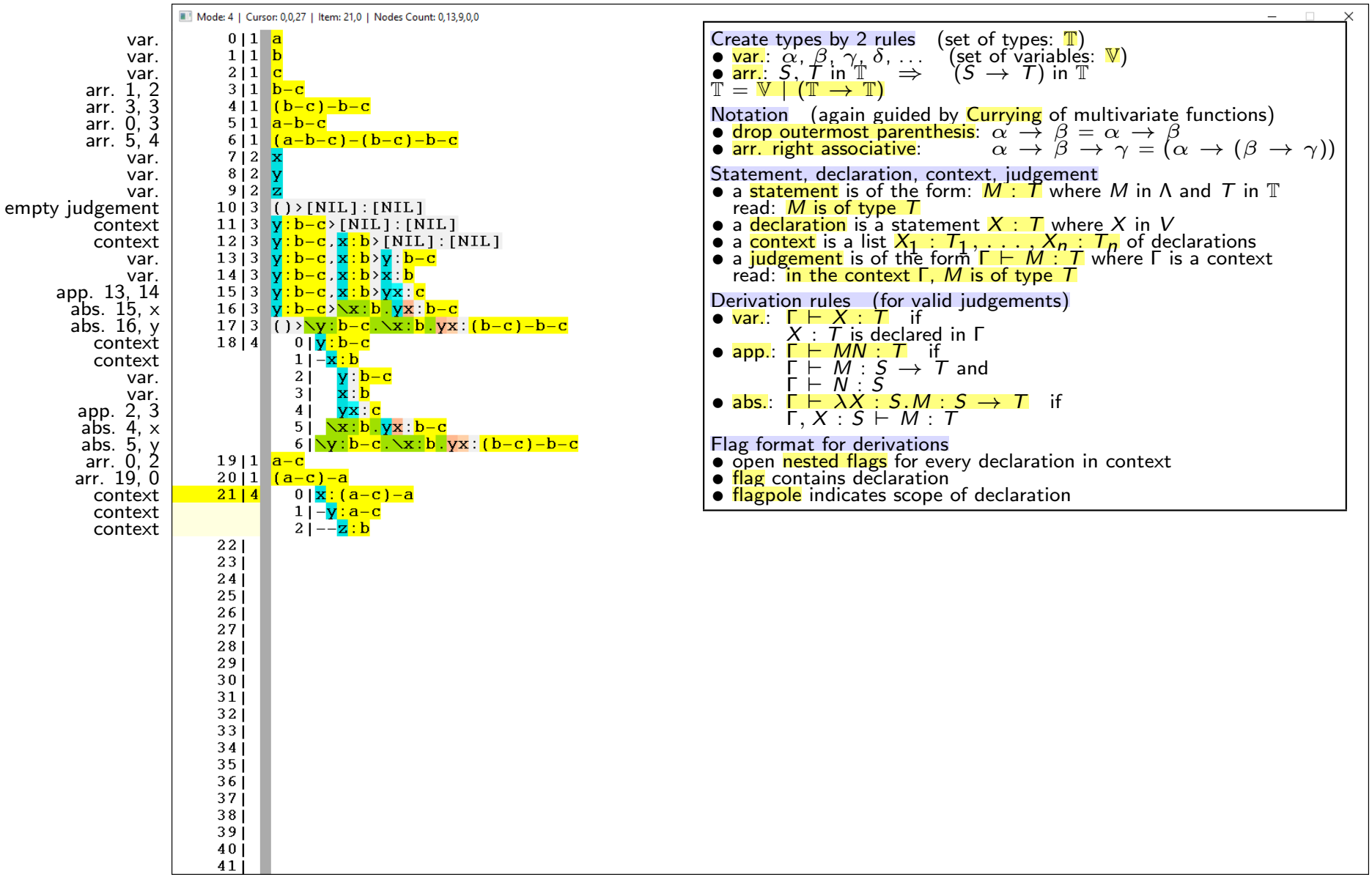
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

```
Mode: 4 | Cursor: 0,0,27 | Item: 21,0 | Nodes Count: 0,13,9,0,0
```

var.	0 | 1	a
var.	1 | 1	b
var.	2 | 1	c
arr. 1, 2	3 | 1	b–c
arr. 3, 3	4 | 1	(b–c)–b–c
arr. 0, 3	5 | 1	a–b–c
arr. 5, 4	6 | 1	(a–b–c)–(b–c)–b–c
var.	7 | 2	x
var.	8 | 2	y
var.	9 | 2	z
empty judgement	10 | 3	()›[NIL]:[NIL]
context	11 | 3	y:b–c›[NIL]:[NIL]
context	12 | 3	y:b–c,x:b›[NIL]:[NIL]
var.	13 | 3	y:b–c,x:b›y:b–c
var.	14 | 3	y:b–c,x:b›x:b
app. 13, 14	15 | 3	y:b–c,x:b›yx:c
abs. 15, x	16 | 3	y:b–c›\x:b.yx:b–c
abs. 16, y	17 | 3	()›\y:b–c.\x:b.yx:(b–c)–b–c
context	18 | 4	0|y:b–c
context		1|–x:b
var.		2| y:b–c
var.		3| x:b
app. 2, 3		4| yx:c
abs. 4, x		5| \x:b.yx:b–c
abs. 5, y		6|\y:b–c.\x:b.yx:(b–c)–b–c
arr. 0, 2	19 | 1	a–c
arr. 19, 0	20 | 1	(a–c)–a
context	21 | 4	0|x:(a–c)–a
context		1|–y:a–c
context		2|––z:b
var.		3| x:(a–c)–a
	22 |	
	23 |	
	24 |	
	25 |	
	26 |	
	27 |	
	28 |	
	29 |	
	30 |	
	31 |	
	32 |	
	33 |	
	34 |	
	35 |	
	36 |	
	37 |	
	38 |	
	39 |	
	40 |	

Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

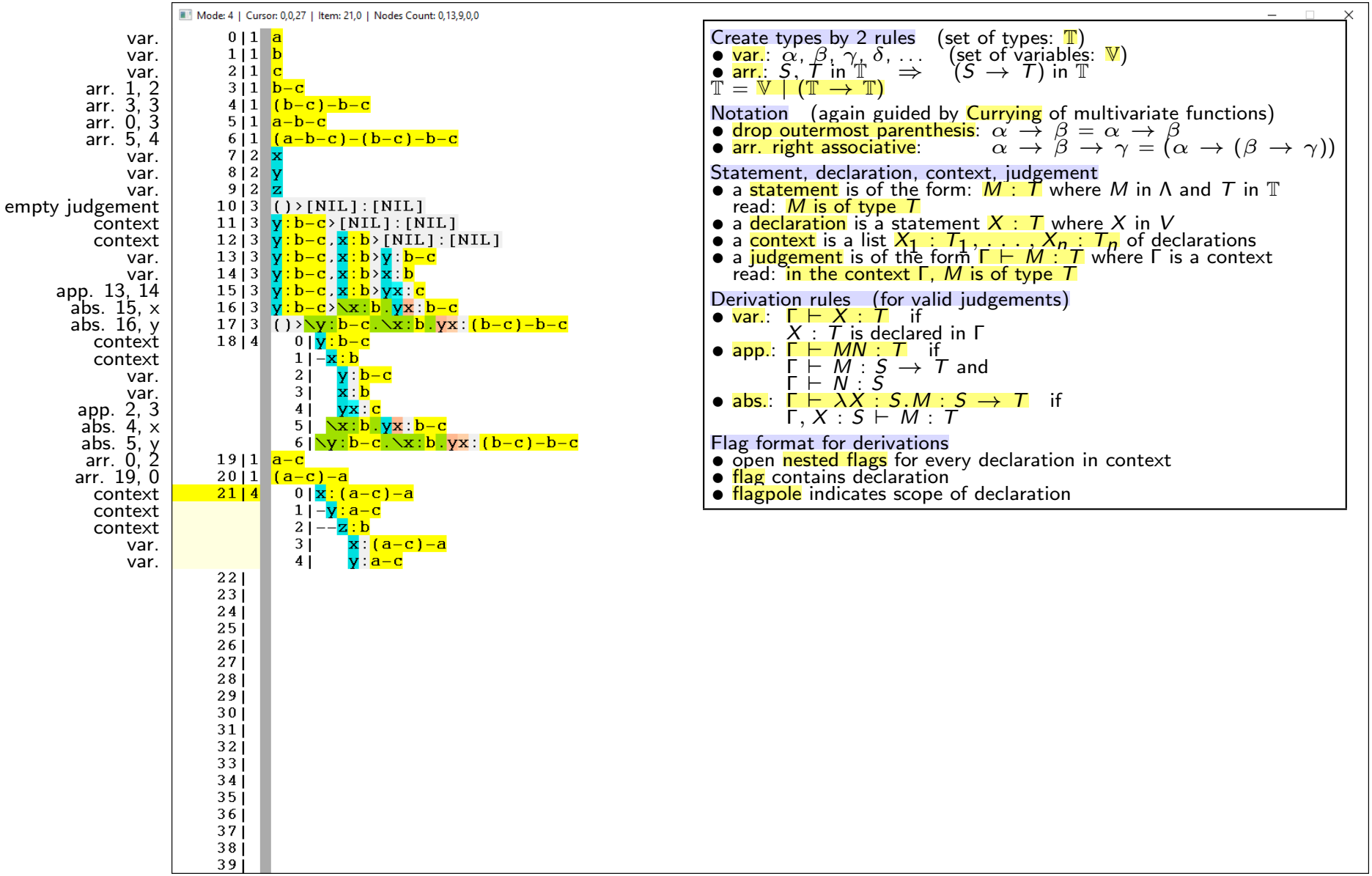
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

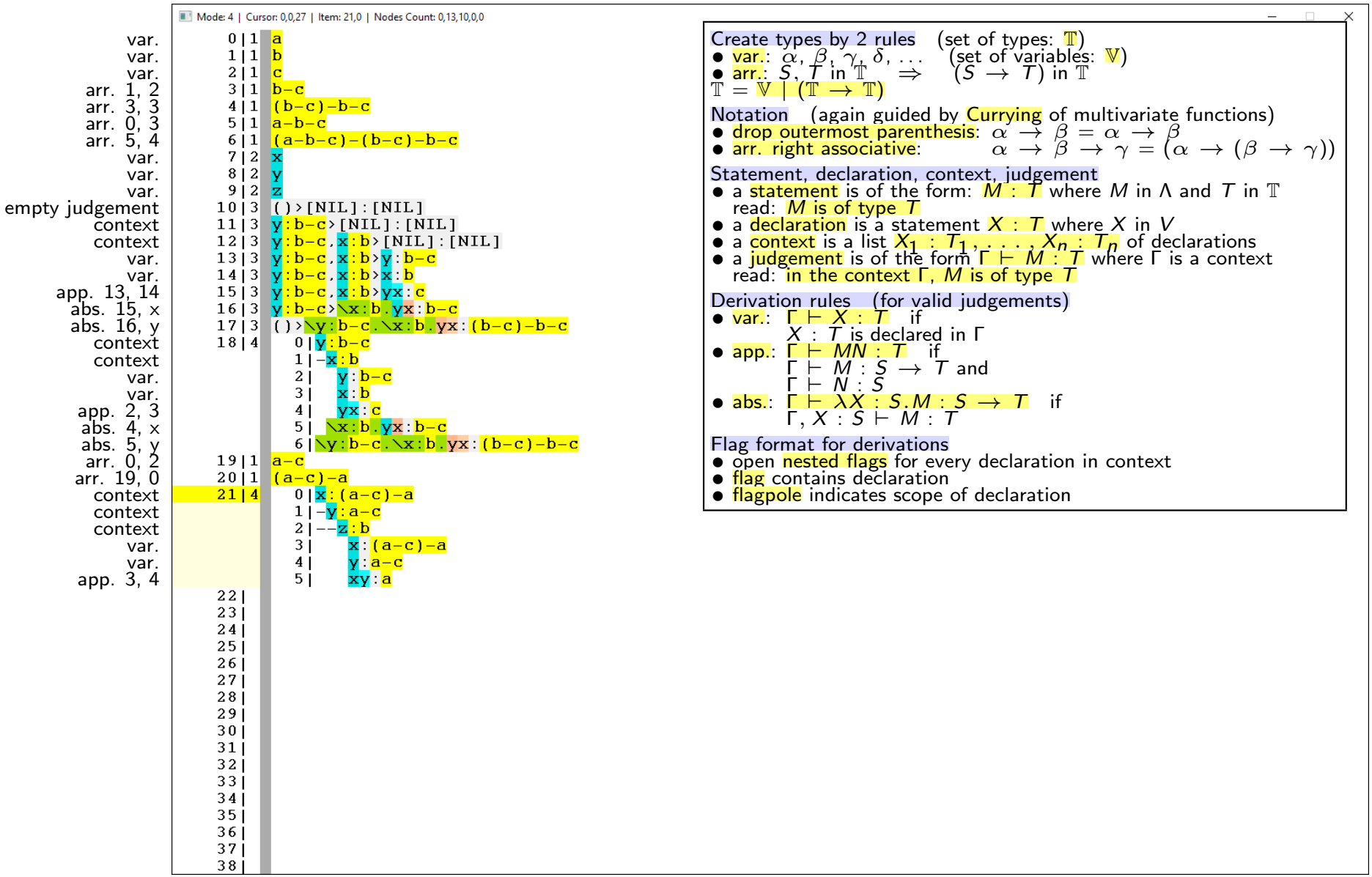
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

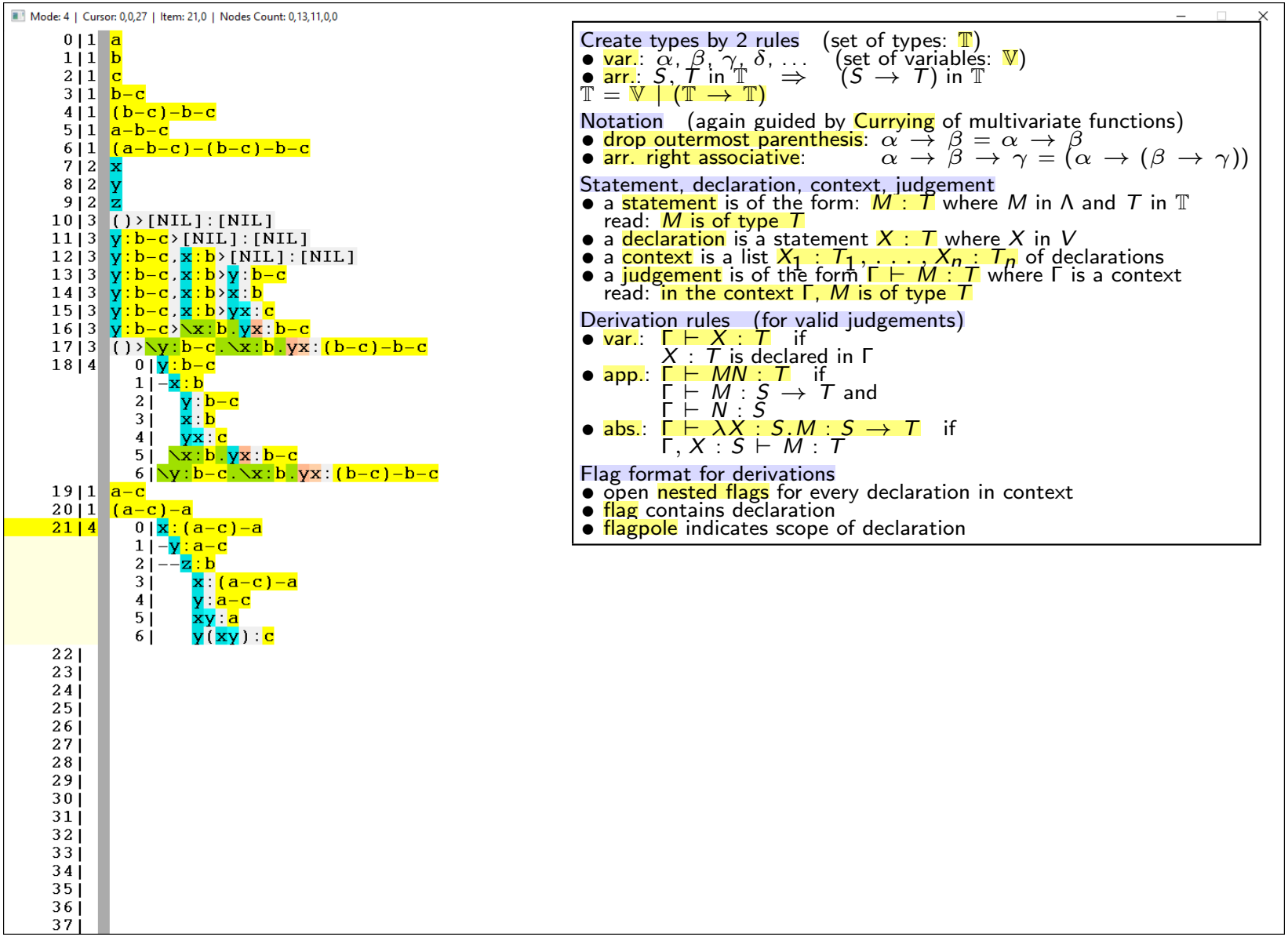
Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

A crash course in type theory



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

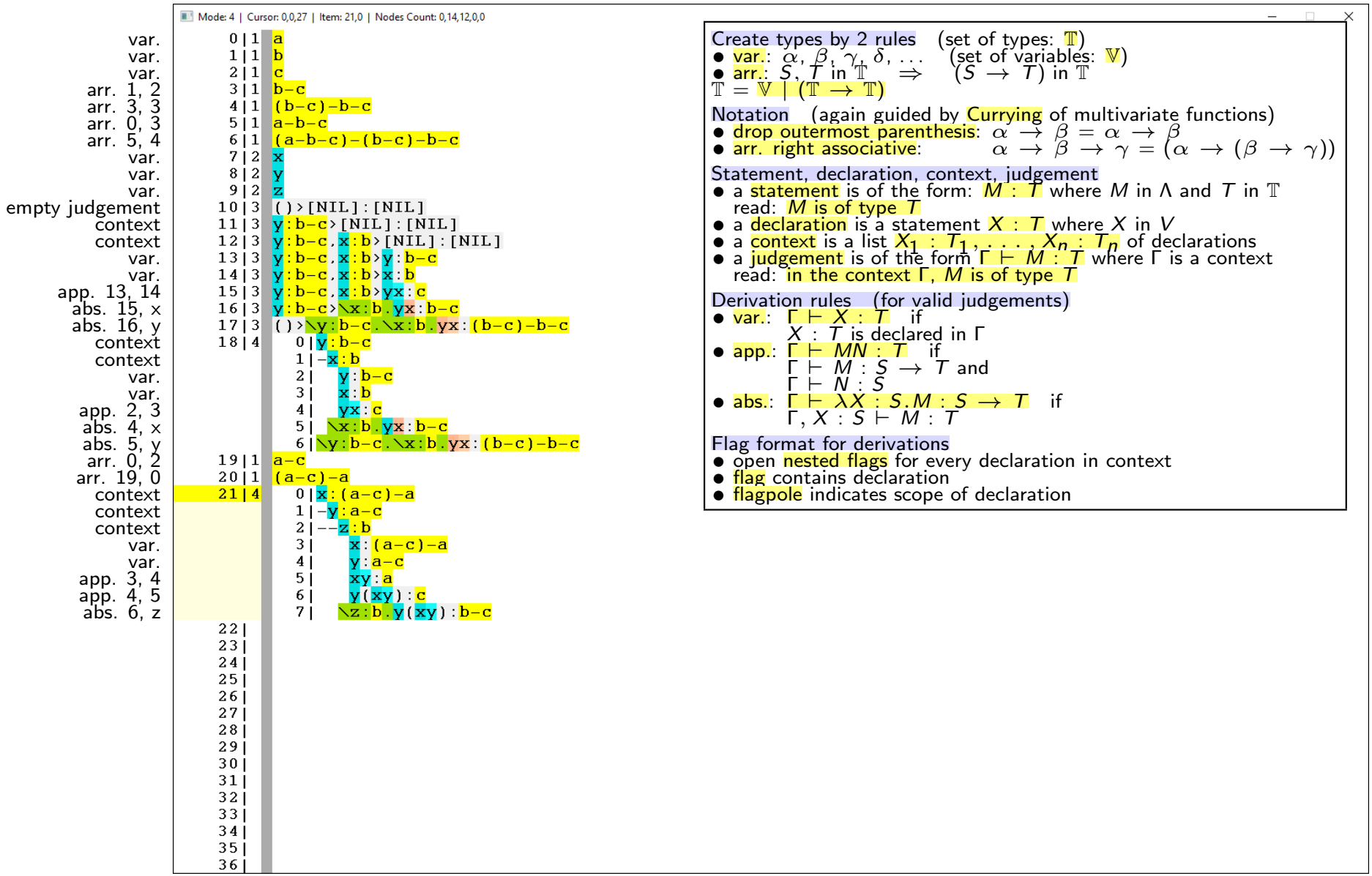
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

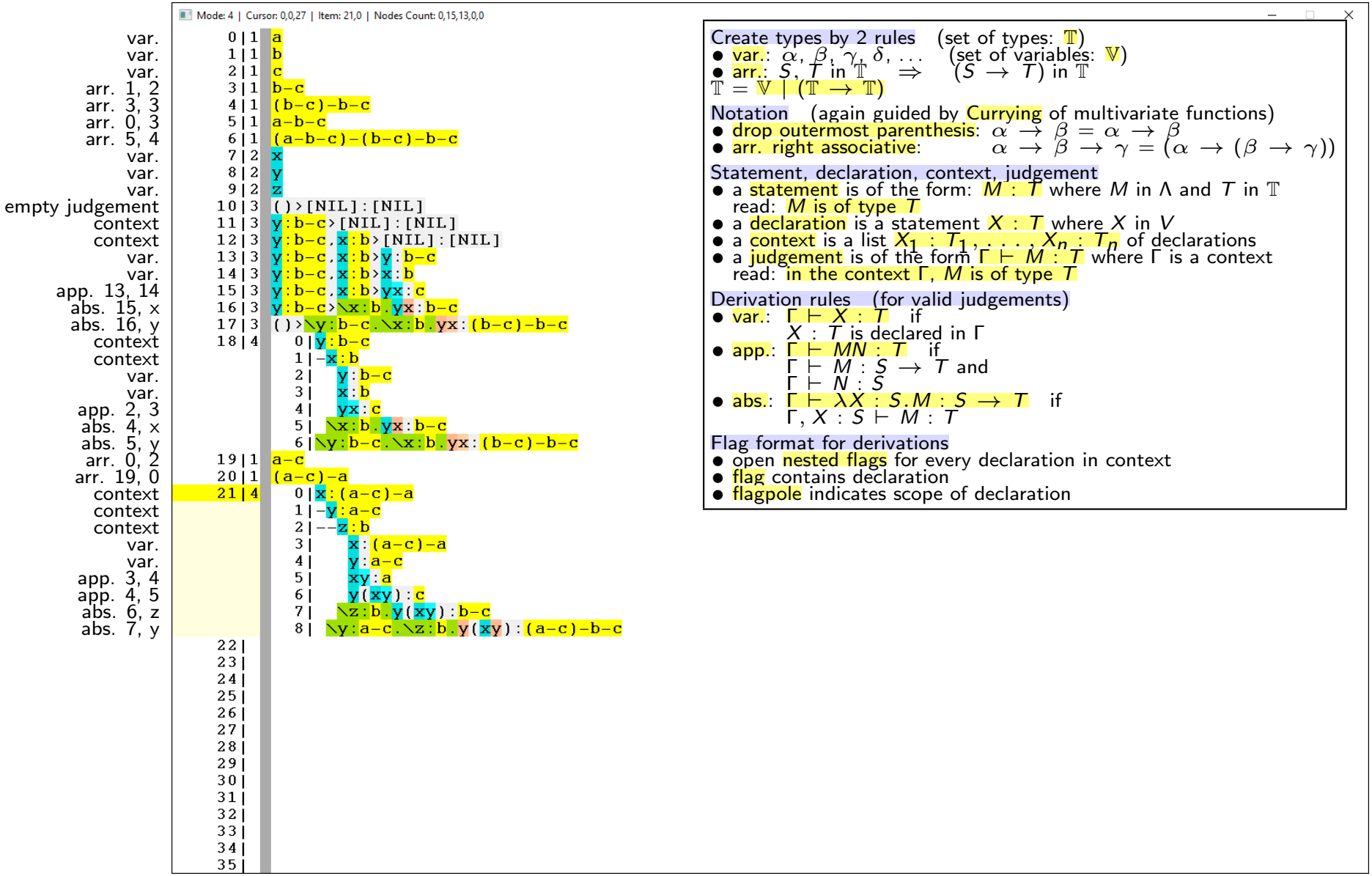
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

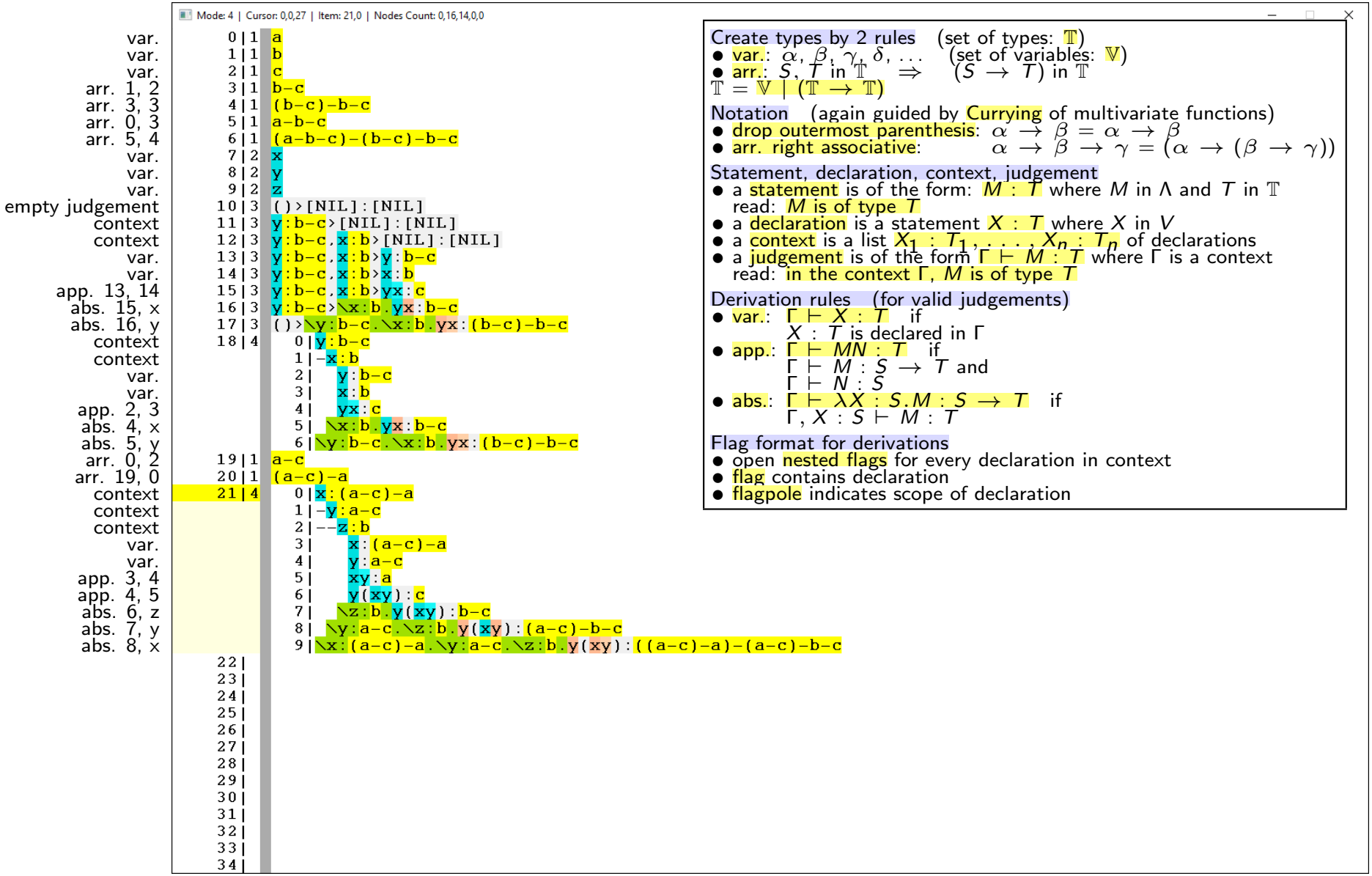
$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration



Create types by 2 rules (set of types: $\mathbb{T}$)
- var.: $\alpha, \beta, \gamma, \delta, \ldots$ (set of variables: $\mathbb{V}$)
- arr.: S, T in $\mathbb{T}$ $\Rightarrow$ $(S \to T)$ in $\mathbb{T}$

$\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$

Notation (again guided by Currying of multivariate functions)
- drop outermost parenthesis: $\alpha \to \beta = \alpha \to \beta$
- arr. right associative: $\alpha \to \beta \to \gamma = (\alpha \to (\beta \to \gamma))$

Statement, declaration, context, judgement
- a statement is of the form: $M : T$ where M in Λ and T in $\mathbb{T}$
  read: M is of type T
- a declaration is a statement $X : T$ where X in V
- a context is a list $X_1 : T_1, \ldots, X_n : T_n$ of declarations
- a judgement is of the form $\Gamma \vdash M : T$ where Γ is a context
  read: in the context Γ, M is of type T

Derivation rules (for valid judgements)
- var.: $\Gamma \vdash X : T$ if
  $X : T$ is declared in Γ
- app.: $\Gamma \vdash MN : T$ if
  $\Gamma \vdash M : S \to T$ and
  $\Gamma \vdash N : S$
- abs.: $\Gamma \vdash \lambda X : S.M : S \to T$ if
  $\Gamma, X : S \vdash M : T$

Flag format for derivations
- open nested flags for every declaration in context
- flag contains declaration
- flagpole indicates scope of declaration

(a) $y : \mathcal{L} \to \mathcal{B}$

(b) $z : \mathcal{L}$

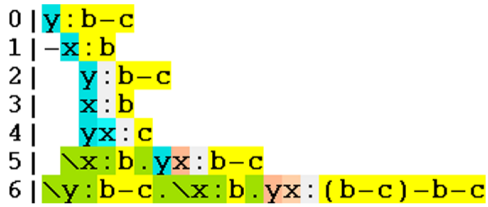
(1) $y : \mathcal{L} \to \mathcal{B}$ (var) on (a)

(2) $z : \mathcal{L}$ (var) on (b)

(3) $yz : \mathcal{B}$ (app) on (1),(2)

(4) $\lambda z : \mathcal{L}.\, yz : \mathcal{L} \to \mathcal{B}$ (abst) on (3)

(5) $\lambda y : \mathcal{L} \to \mathcal{B}.\, \lambda z : \mathcal{L}.\, yz : (\mathcal{L} \to \mathcal{B}) \to \mathcal{L} \to \mathcal{B}$ (abst) on (4)



(a) $x : (\mathcal{L} \to \gamma) \to \mathcal{L}$

(b) $y : \mathcal{L} \to \gamma$

(c) $z : \mathcal{B}$

(v) $xy : \mathcal{L}$ (app) on (a),(b)

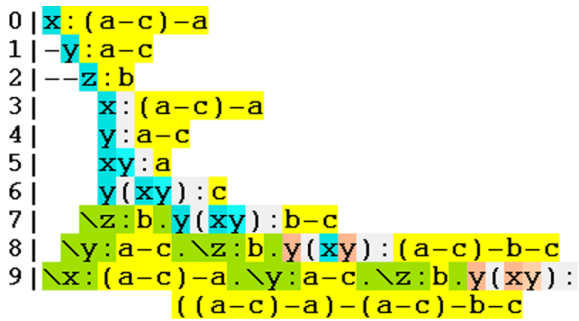
(w) $y(xy) : \gamma$ (app) on (b),(v)

(x) $\lambda z : \mathcal{B}.\, y(xy) : \mathcal{B} \to \gamma$ (abst) on (w)

(y) $\lambda y : \mathcal{L} \to \gamma.\, \lambda z : \mathcal{B}.\, y(xy) : (\mathcal{L} \to \gamma) \to \mathcal{B} \to \gamma$ (abst) on (x)

(z) $\lambda x : (\mathcal{L} \to \gamma) \to \mathcal{L}.\, \lambda y : \mathcal{L} \to \gamma.\, \lambda z : \mathcal{B}.\, y(xy) :$ (abst) on (y)

$((\mathcal{L} \to \gamma) \to \mathcal{L}) \to (\mathcal{L} \to \gamma) \to \mathcal{B} \to \gamma$



(a) $y : \mathcal{L} \to \mathcal{B}$

(b) $z : \mathcal{L}$

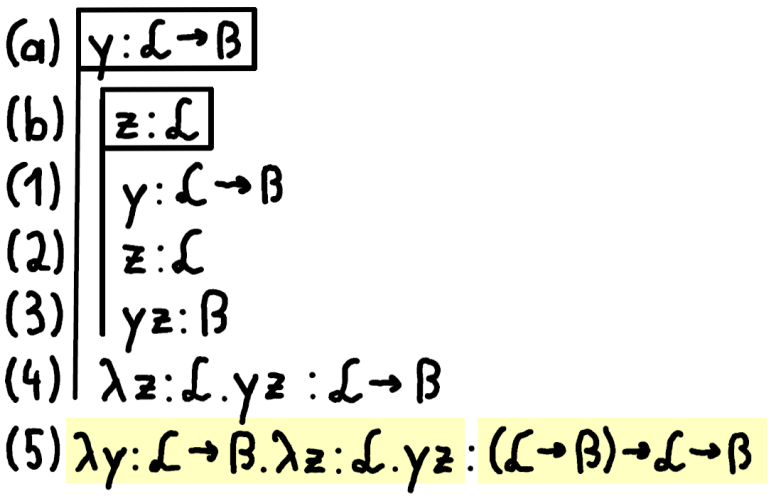
(1) $y : \mathcal{L} \to \mathcal{B}$

(2) $z : \mathcal{L}$

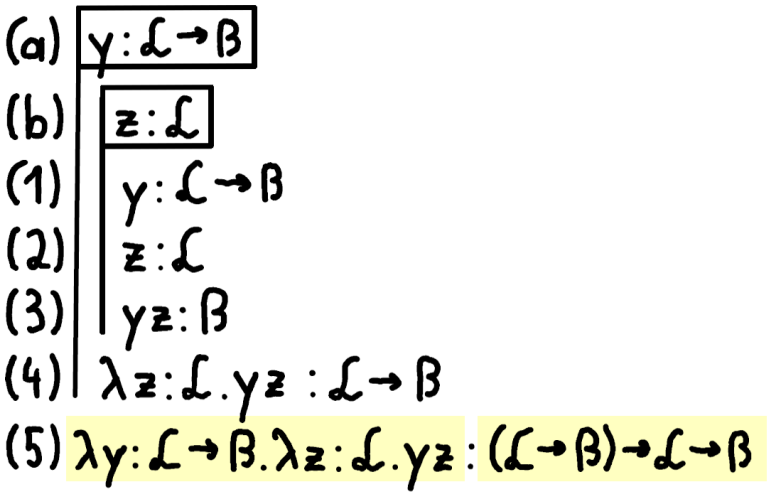
(3) $yz : \mathcal{B}$

(4) $\lambda z : \mathcal{L}.\, yz : \mathcal{L} \to \mathcal{B}$

(5) $\lambda y : \mathcal{L} \to \mathcal{B}.\, \lambda z : \mathcal{L}.\, yz : (\mathcal{L} \to \mathcal{B}) \to \mathcal{L} \to \mathcal{B}$

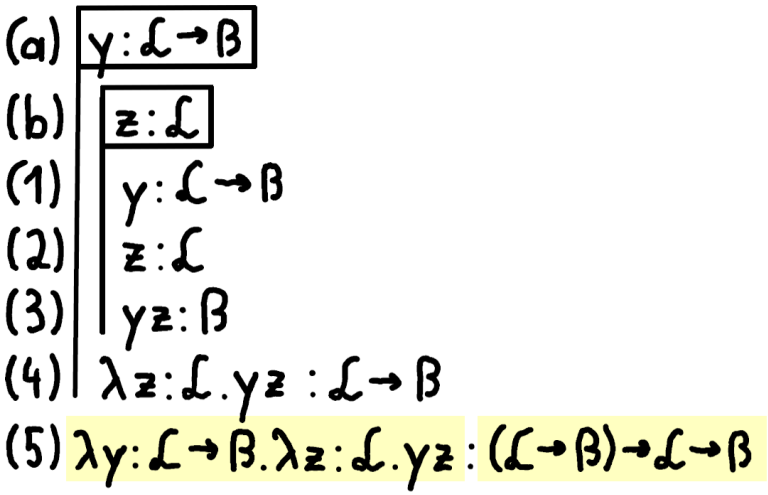


The Curry-Howard Isomorphism



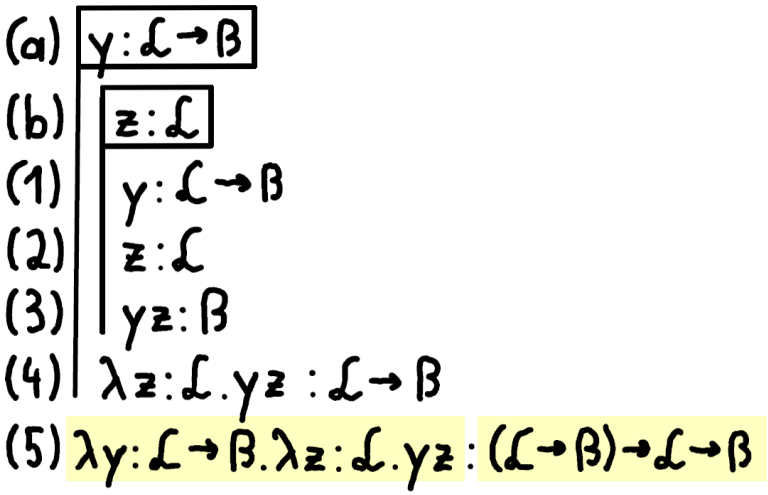
The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms



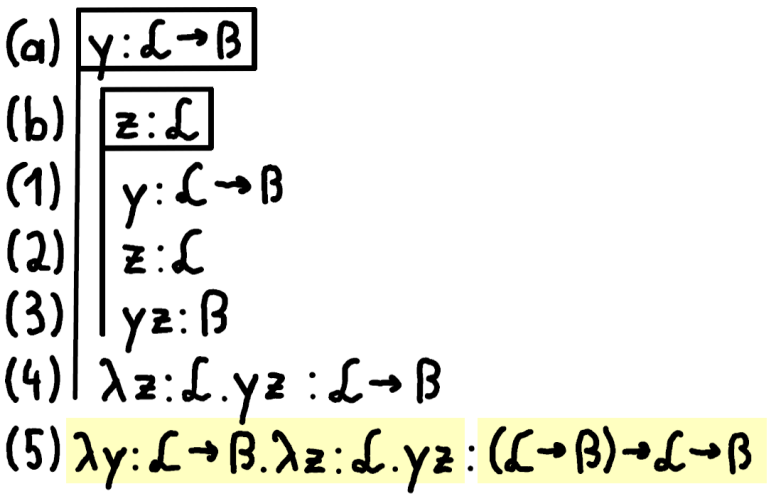
The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\rightarrow$ as $\Rightarrow$



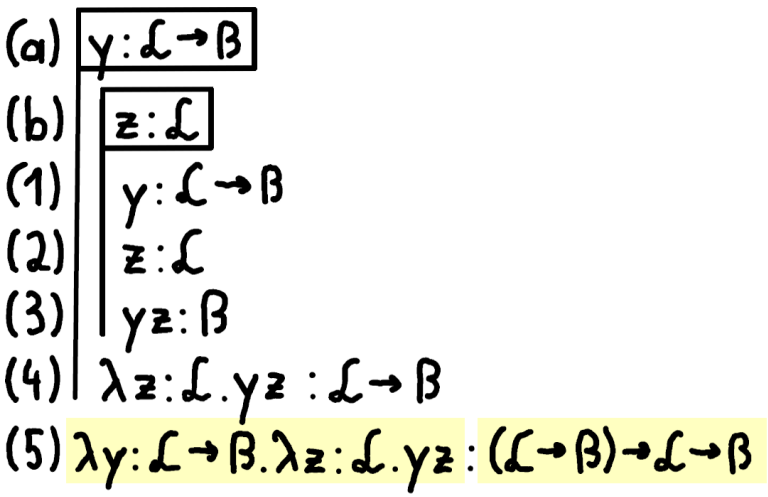
The Curry-Howard Isomorphism

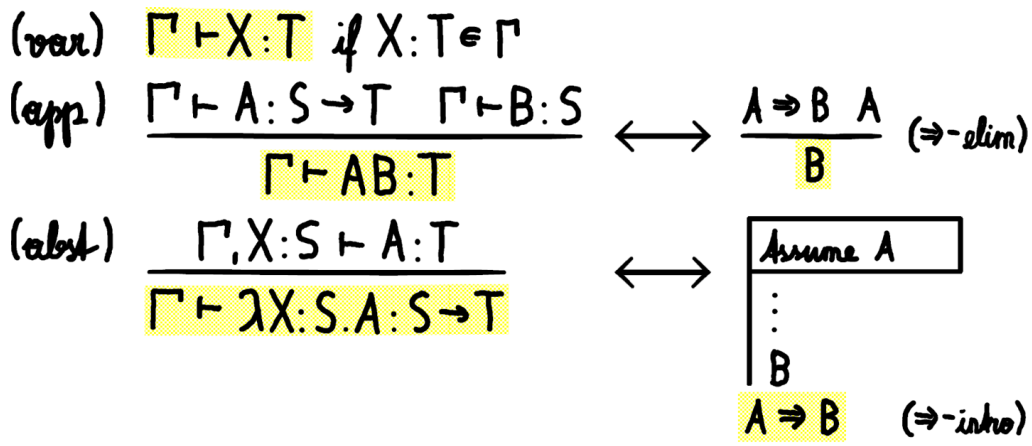
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\rightarrow$ as $\Rightarrow$
  $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$



The Curry-Howard Isomorphism

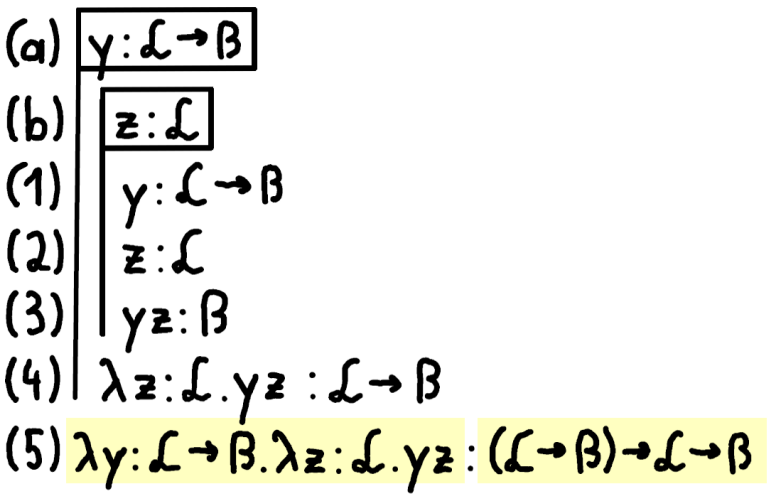
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\rightarrow$ as $\Rightarrow$
  $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \rightarrow \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)

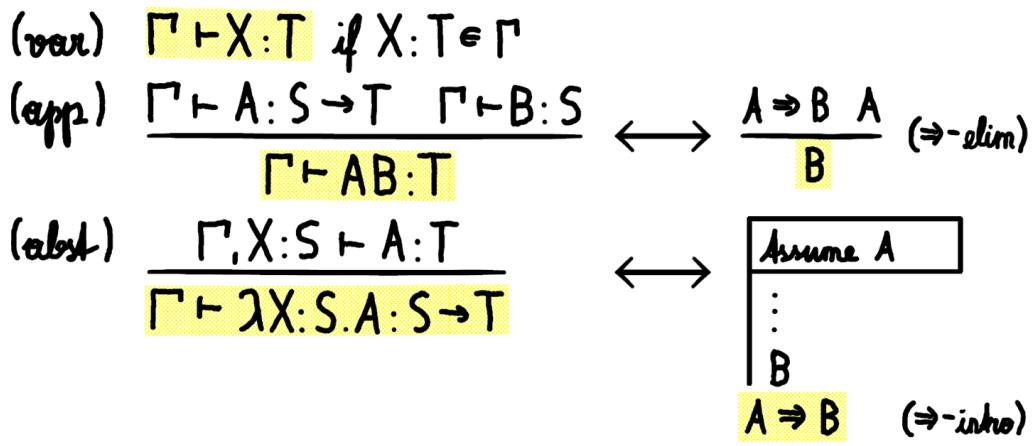




The Curry-Howard Isomorphism

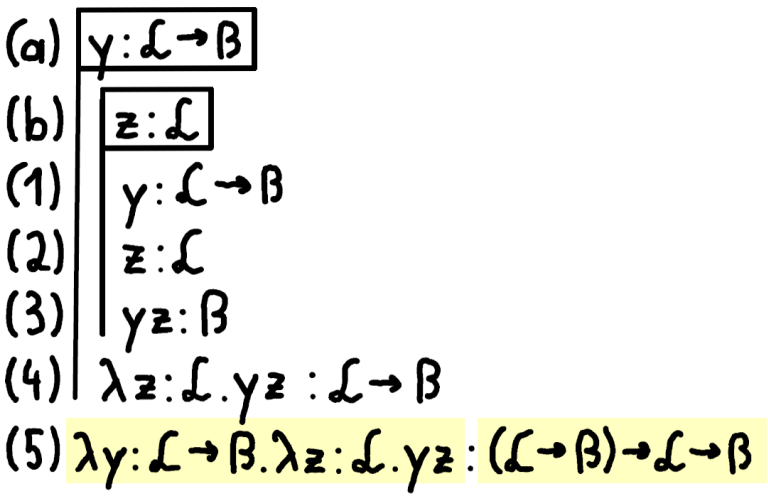
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)

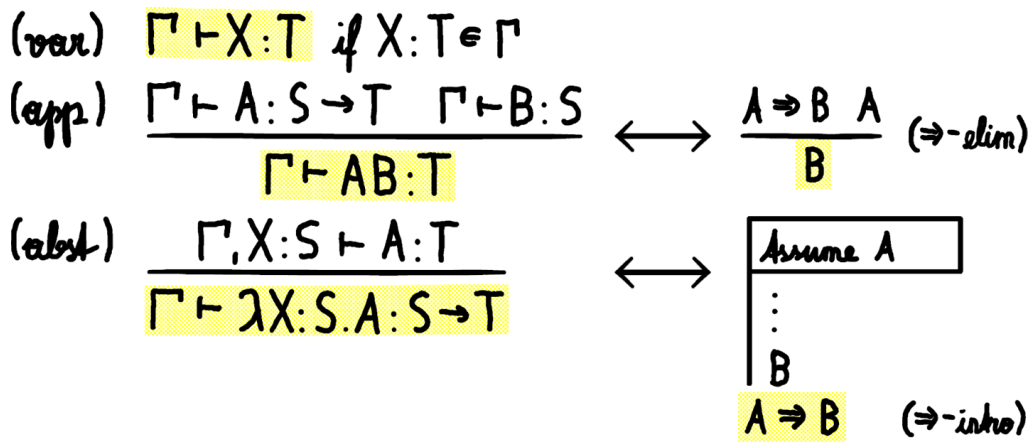




The Curry-Howard Isomorphism

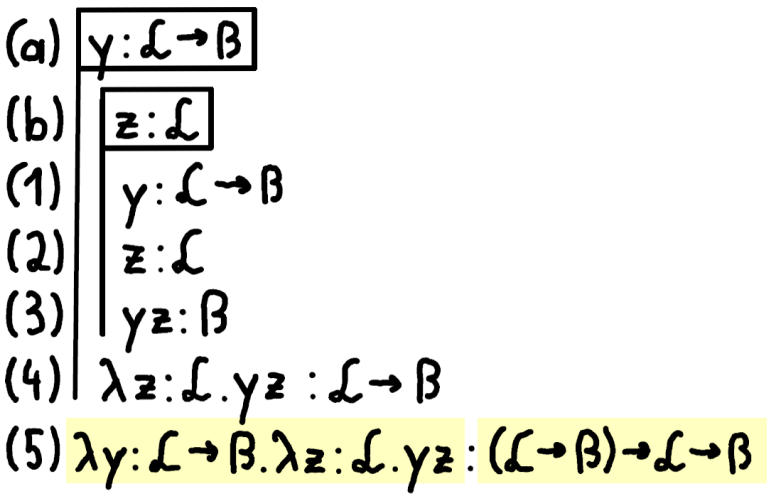
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a proposition is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)

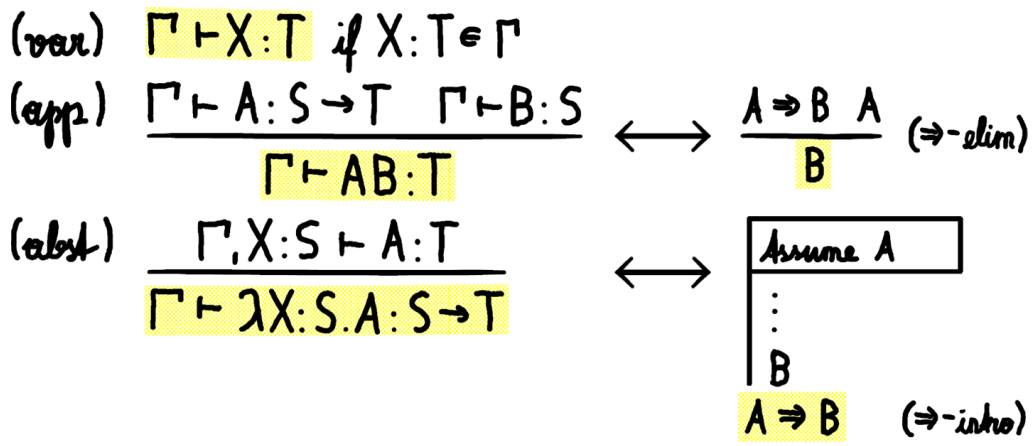




The Curry-Howard Isomorphism

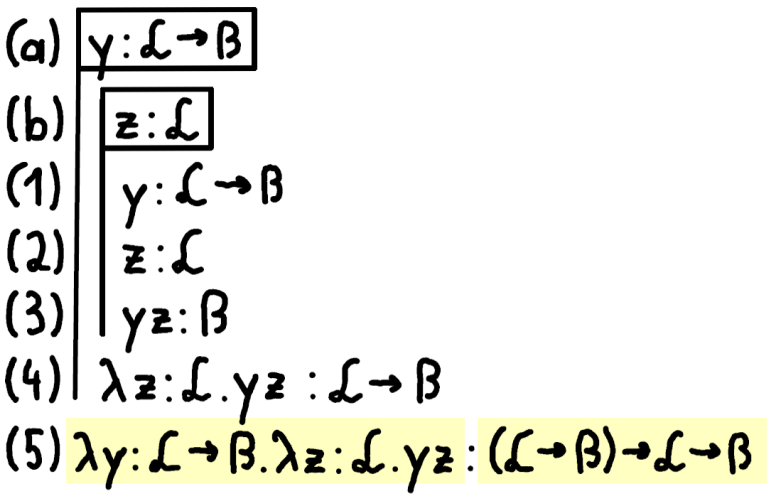
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a proposition is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory

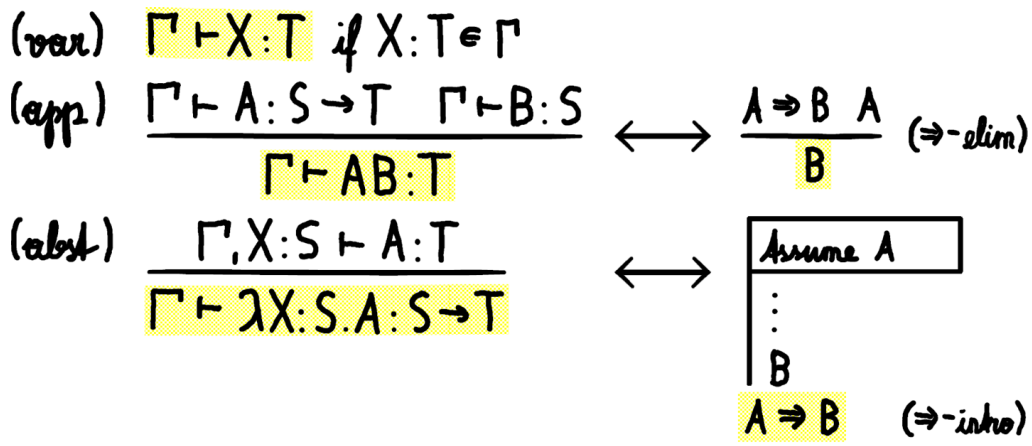




The Curry-Howard Isomorphism

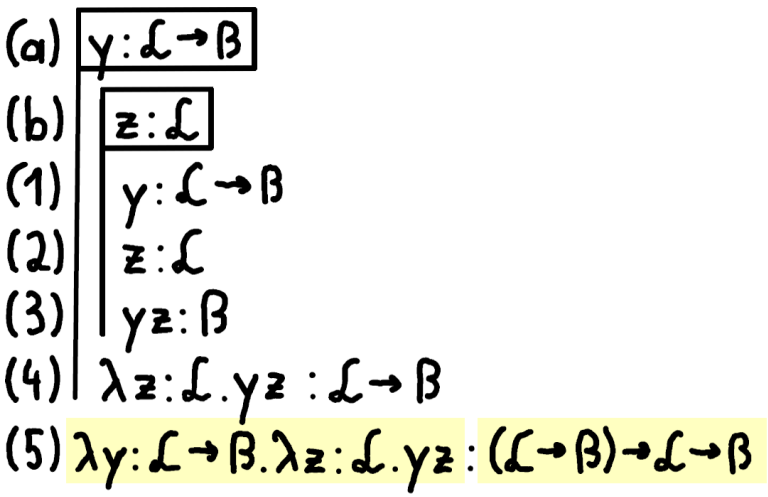
- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a proposition is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!

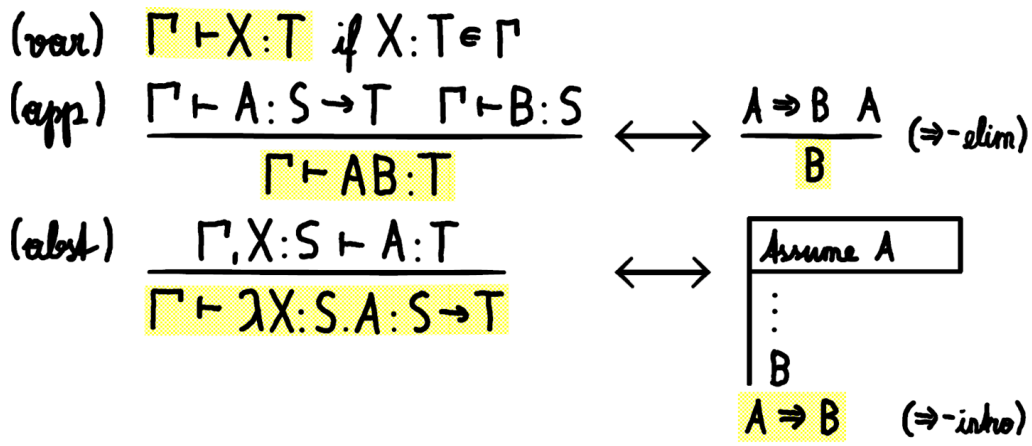




The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a proposition is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!
- philosophical implication: computing and arguing are the same thing!





The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a proposition is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!
- philosophical implication: computing and arguing are the same thing!
  (cf. also: Church-Turing Thesis)

The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a propositions is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!
- philosophical implication: computing and arguing are the same thing!
  (cf. also: Church-Turing Thesis)
- advantage over set theory: proofs are objects of type theory but meta-objects of set theory

The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a propositions is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!
- philosophical implication: computing and arguing are the same thing!
  (cf. also: Church-Turing Thesis)
- advantage over set theory: proofs are objects of type theory but meta-objects of set theory
  λ-terms: data structure for all logical arguments!

The Curry-Howard Isomorphism

- aka PAT: Propositions As Types
  Proofs As Terms
- interpret context as logical assumptions and $\to$ as $\Rightarrow$
  $(\alpha \to \beta) \to \alpha \to \beta$ becomes $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$
  $\lambda y : \alpha \to \beta . \lambda z : \alpha . yz$ becomes a proof of $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ (in the empty context)
- a propositions is true if and only if the corresponding type is inhabited
  (type inhabited: term of given type exists)
- so far: only propositions and $\Rightarrow$ modelled by type theory
  richer type theory: all of logic (and set theory) can be modelled!
- philosophical implication: computing and arguing are the same thing!
  (cf. also: Church-Turing Thesis)
- advantage over set theory: proofs are objects of type theory but meta-objects of set theory
  λ-terms: data structure for all logical arguments!
  basis of Lean: inductive types

Haskell Curry
(1900 - 1982)

William Alvin Howard
(1926)

A richer type theory: the Calculus of Constructions

(sort) $\quad () \vdash * : \square$ — two rules in one: $s = *$ or $s = \square$

(silent) (var) $\quad \dfrac{\Gamma \vdash A : s}{\Gamma, X:A \vdash X:A}$ if $X \notin \mathrm{dom}(\Gamma)$

(silent) (weak) $\quad \dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, X:C \vdash A : B}$ if $X \notin \mathrm{dom}(\Gamma)$

(silent) (form) $\quad \dfrac{\Gamma \vdash A : s_1 \quad \Gamma, X:A \vdash B : s_2}{\Gamma \vdash \Pi X:A.B : s_2}$ — four rules in one

$\leftarrow = A \to B$ if $X \notin FV(B)$

(app) $\quad \dfrac{\Gamma \vdash A : \Pi X:S.T \quad \Gamma \vdash B : S}{\Gamma \vdash AB : T[X:=B]}$ (silent)

(abst) $\quad \dfrac{\Gamma, X:S \vdash A : T \quad \Gamma \vdash \Pi X:S.T : s}{\Gamma \vdash \lambda X:S.A : \Pi X:S.T}$

(conv) $\quad \dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$ if $B =_\beta B'$ (silent)

A richer type theory: the Calculus of Constructions

(1) **Sets** $S : *$.

(2) **Propositions** $P : *$.

(3) **Predicates** $P : S \to *$.

(4) **Implication** $A \Rightarrow B = A \to B$ $(= \Pi x : A.B$ *if* $x \notin FV(B))$.

(5) **Universal q.** $\forall x \in S : P(x) = \Pi x : S.Px$.

(6) **Absurdity** $\bot \equiv \Pi C : *.C$.

(7) **Negation** $\neg \equiv \Pi A : *.A \to \bot$.

(8) **Conjunction** $\wedge \equiv \Pi A : *.\Pi B : *.\Pi C : *.(A \to B \to C) \to C$, $A \wedge B = \wedge AB$.

(9) **Disjunction** $\vee \equiv \Pi A : *.\Pi B : *.\Pi C : *.(A \to C) \to (B \to C) \to C$, $A \vee B = \vee AB$.

(10) **Existential q.** $\exists \equiv \Pi S : *.\Pi P : S \to *.\Pi C : *.((\Pi x : S.(Px \to C)) \to C)$,

$\exists x \in S : P(x) = \exists SP$.

(11) **Axioms** *Add assumption in front of context.*

A richer type theory: the Calculus of Constructions

$(A \vee B) \Rightarrow (\neg A \Rightarrow B)$

$\lambda C : (\Pi C : *.\, (A \to C) \to (B \to C) \to C) \to (A \to \bot) \to B.$

(a) $A : *$

(b) $B : *$

(c) $x : \Pi C : *.\, (A \to C) \to (B \to C) \to C$

(d) $y : A \to \bot$

(1) $xB : (A \to B) \to (B \to B) \to B$ (app) on (c),(b)

(e) $u : A$

(2) $yu : \bot$ (app) on (d),(e)

(3) $yuB : B$ (app) on (2),(b)

(4) $\lambda u : A.\, yuB : A \to B$ (abst) on (3)

(5) $xB(\lambda u : A.\, yuB) : (B \to B) \to B$ (app) on (1),(4)

(f) $v : B$

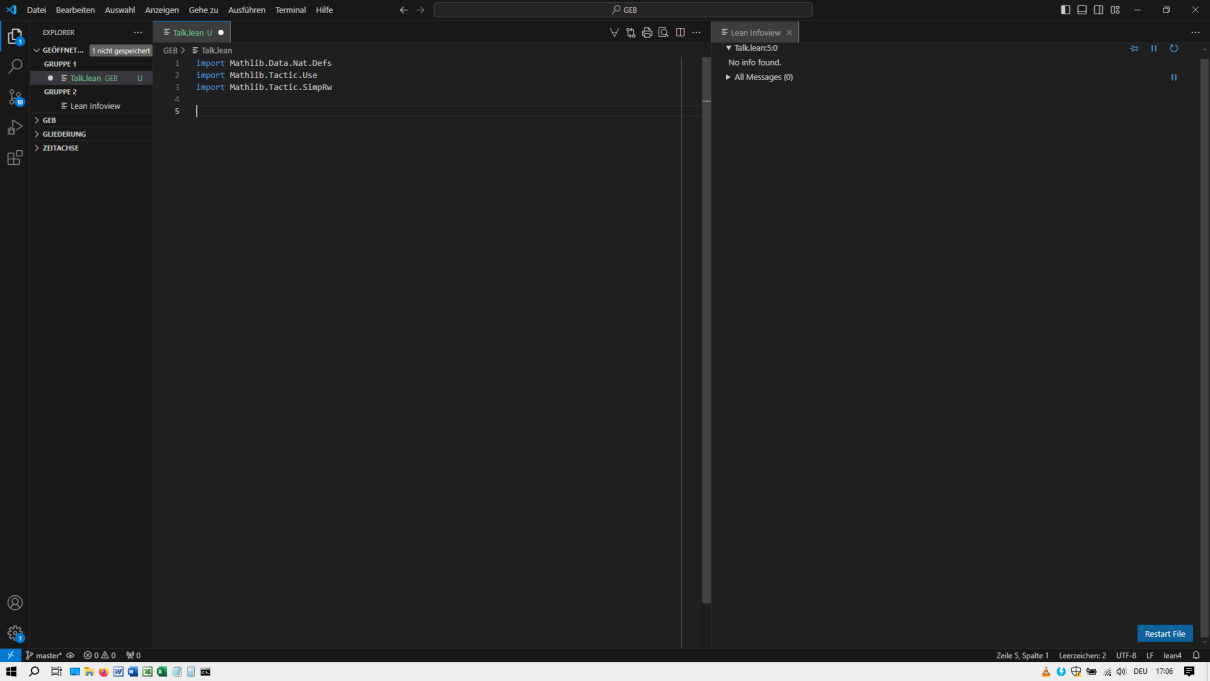
(6) $v : B$ (var) on (f)

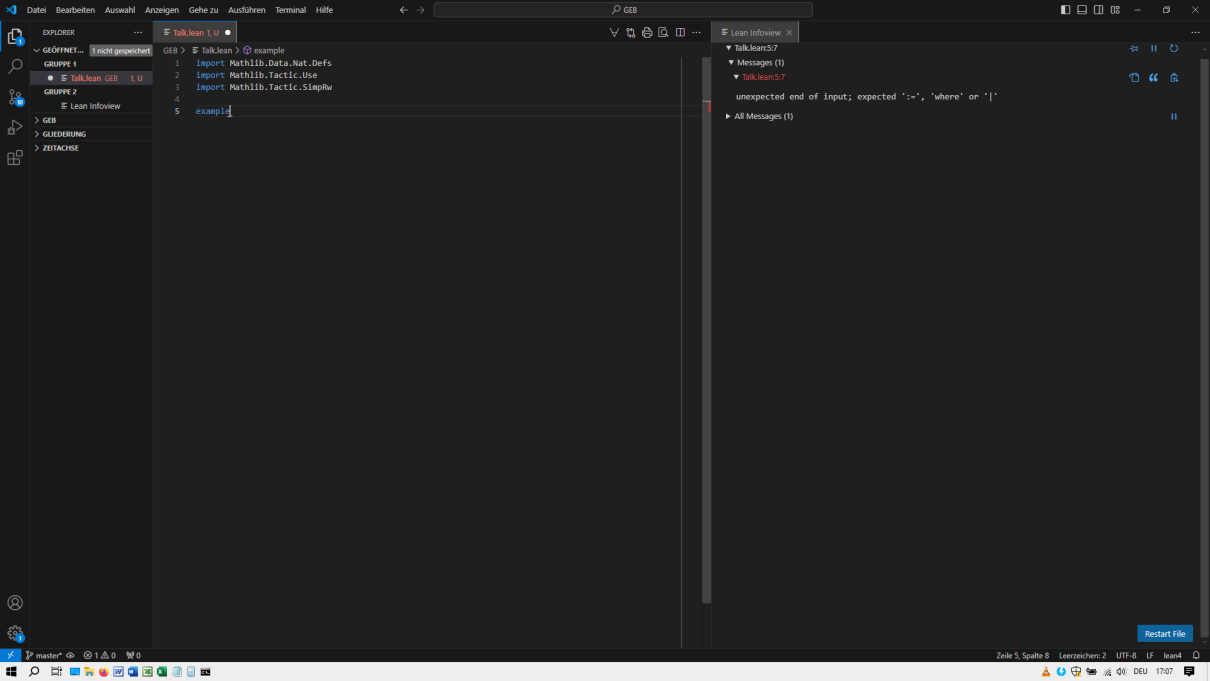
(7) $\lambda v : B.\, v : B \to B$ (abst) on (6)

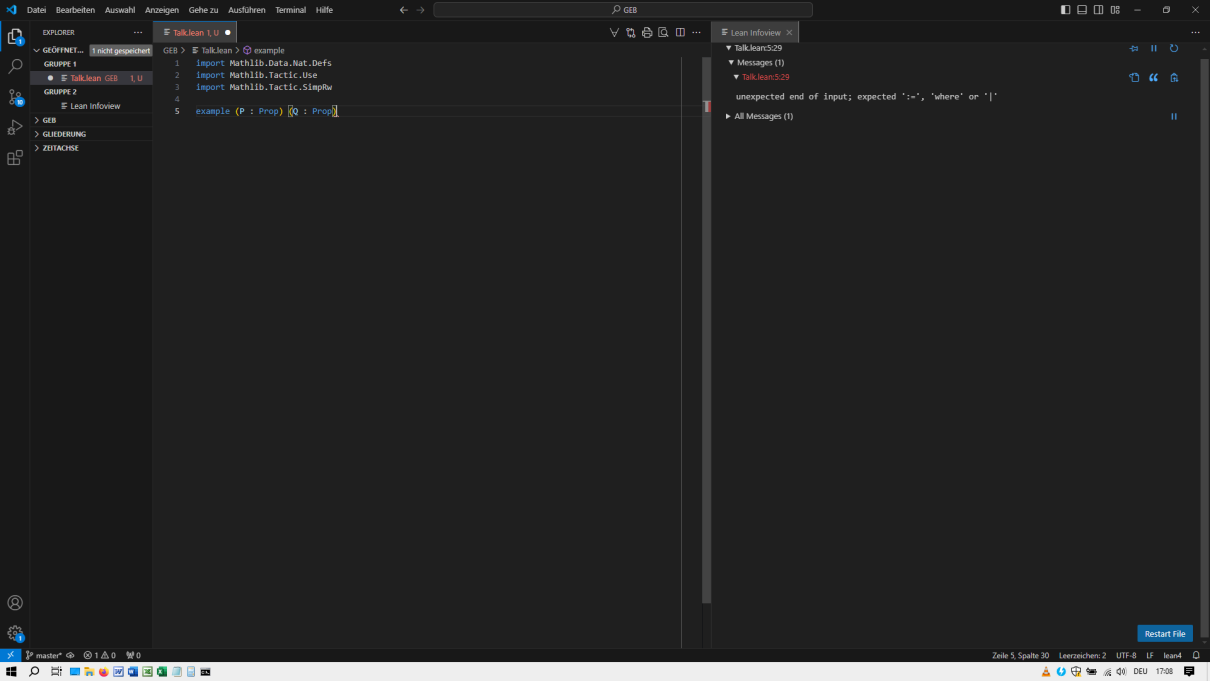
(8) $xB(\lambda u : A.\, yuB)(\lambda v : B.\, v) : B$ (app) on (5),(7)

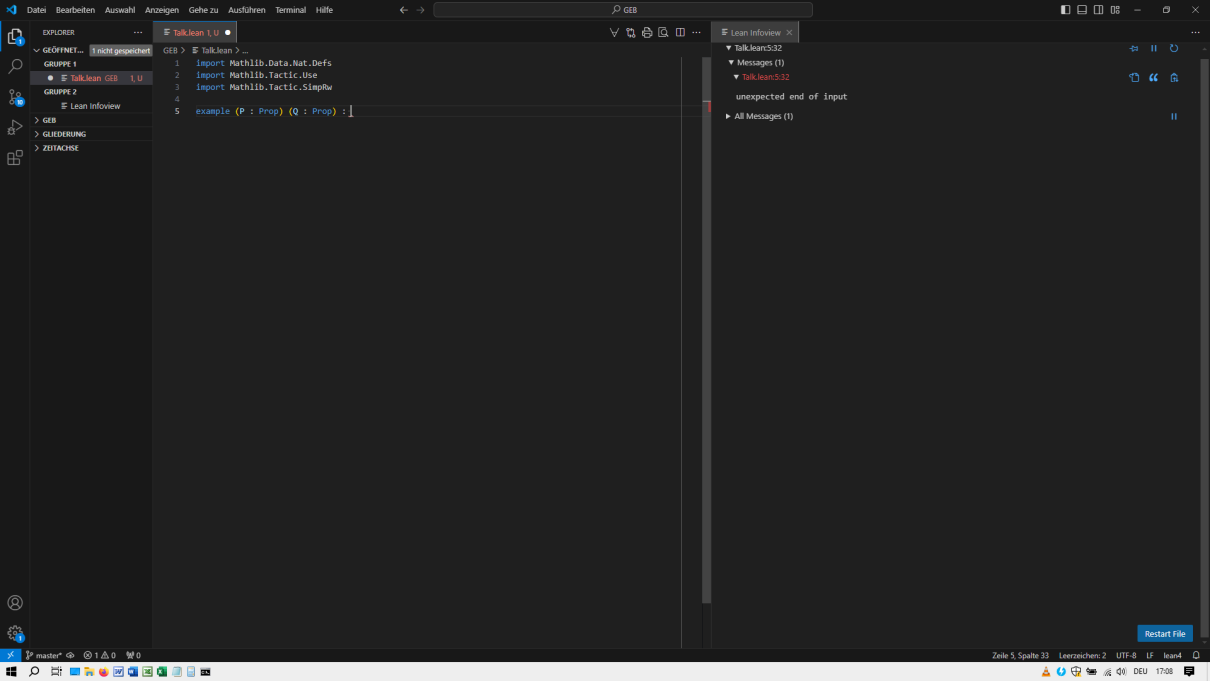
(9) $\lambda y : A \to \bot.\, xB(\lambda u : A.\, yuB)(\lambda v : B.\, v) : (A \to \bot) \to B$ (abst) on (8)

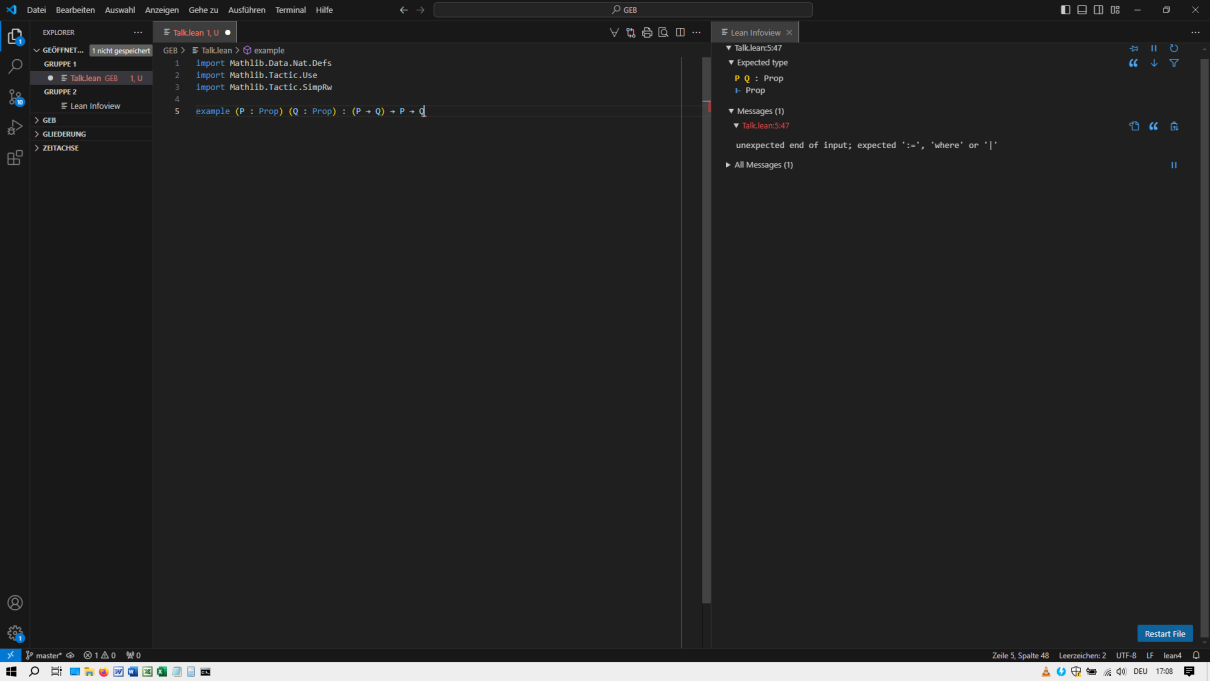
(10) $\lambda x : (\Pi C : *.\, (A \to C) \to (B \to C) \to C).\ \lambda y : A \to \bot.\, xB(\lambda u : A.\, yuB)(\lambda v : B.\, v) : (\Pi C : *.\, (A \to C) \to (B \to C) \to C) \to (A \to \bot) \to B$ (abst) on (9)

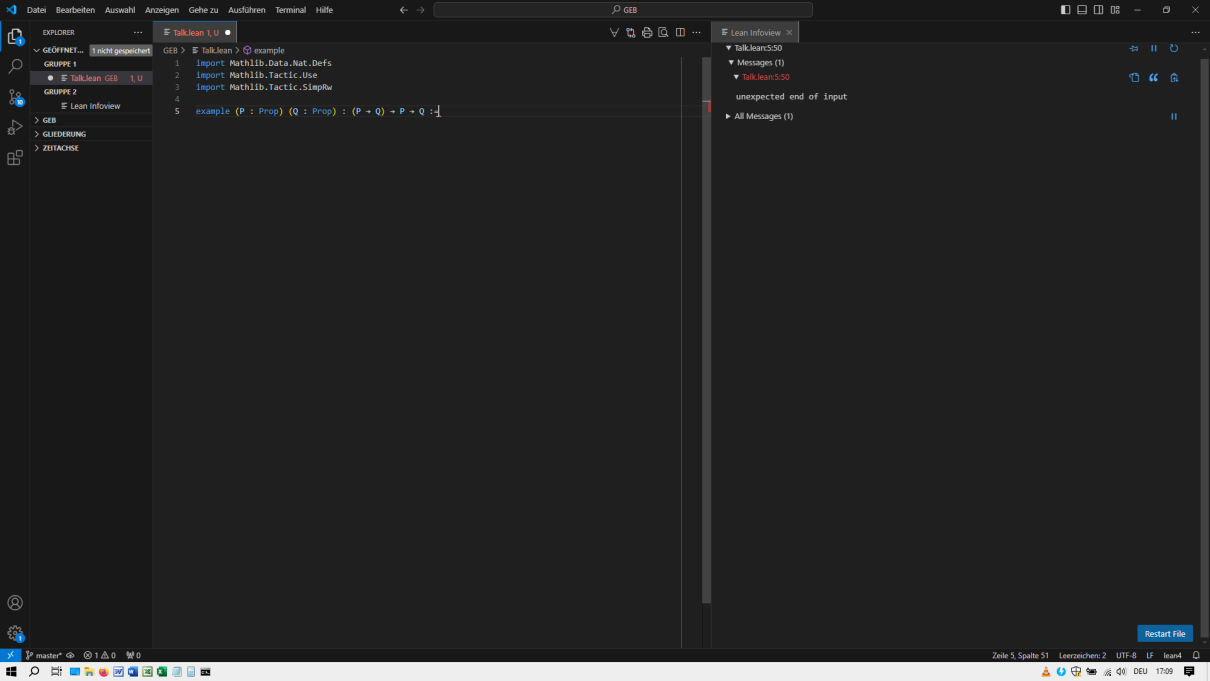

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5
Lean Infoview
Talk.lean:5:0
No info found.
All Messages (0)
Restart File
master*
0 0
0
Zeile 5, Spalte 1 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:06

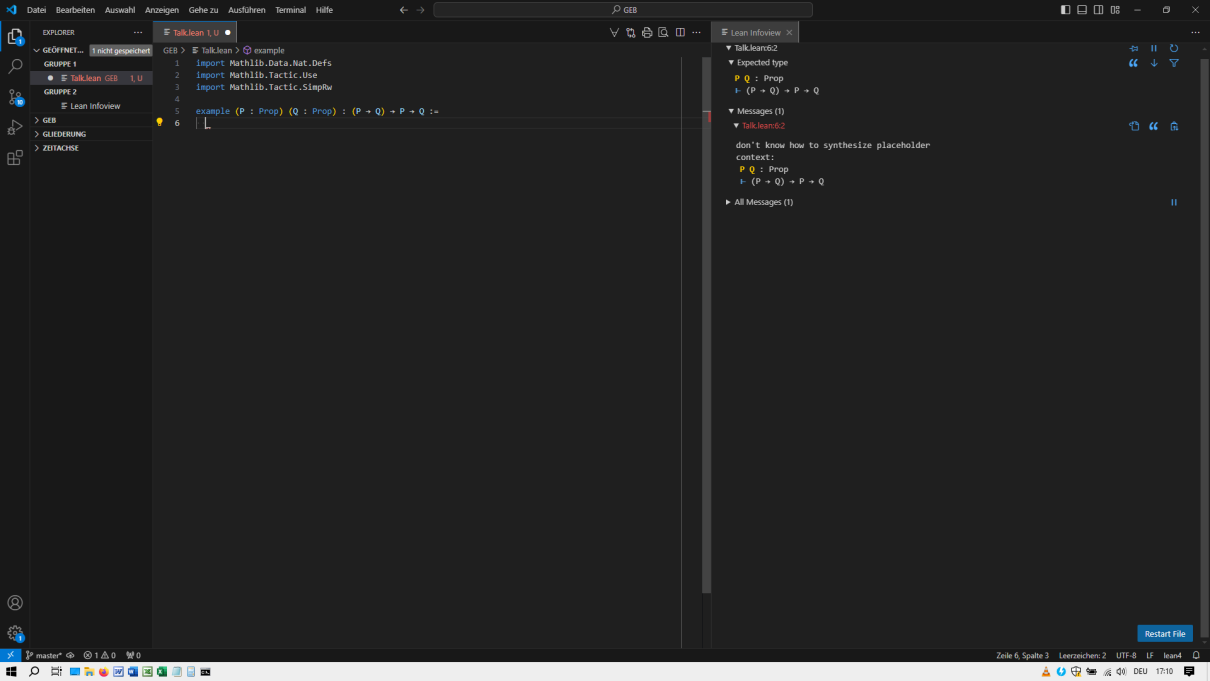

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example
Lean Infoview
Talk.lean:5:7
Messages (1)
Talk.lean:5:7
unexpected end of input; expected ':=', 'where' or '|'
All Messages (1)
Restart File
master*
1 0
0
Zeile 5, Spalte 8
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:07

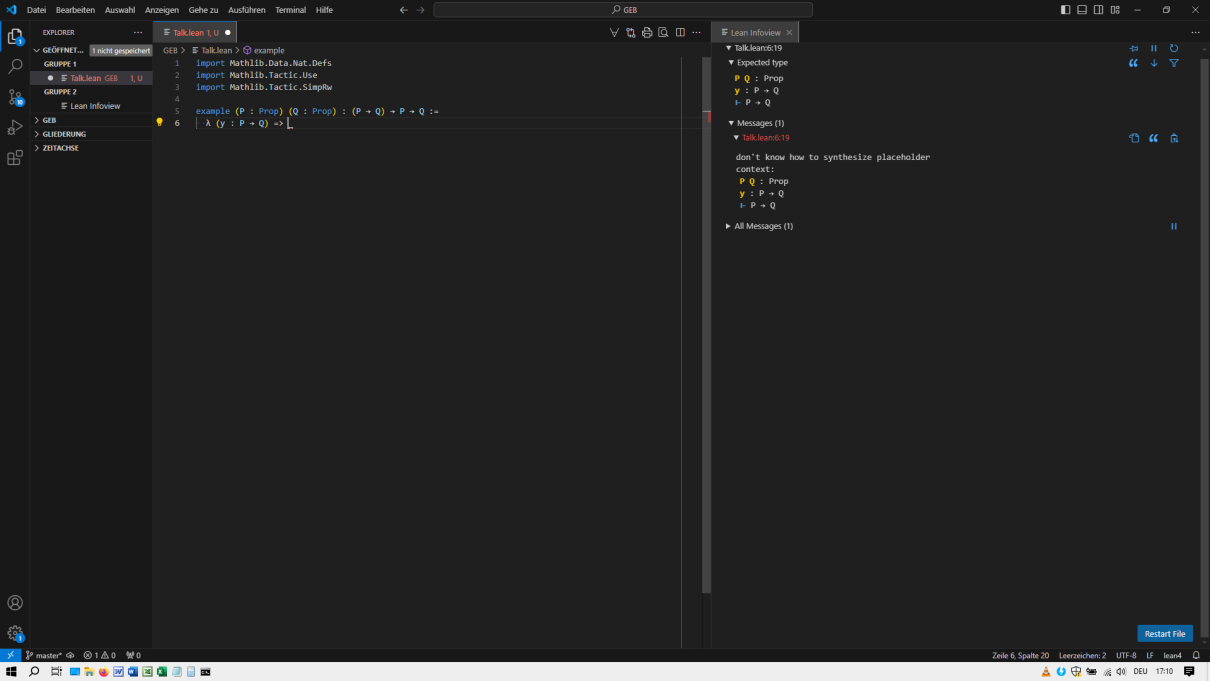

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop)
Lean Infoview
Talk.lean:5:29
Messages (1)
Talk.lean:5:29
unexpected end of input; expected ':=', 'where' or '|'
All Messages (1)
Restart File
master*
1 0
0
Zeile 5, Spalte 30 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:08

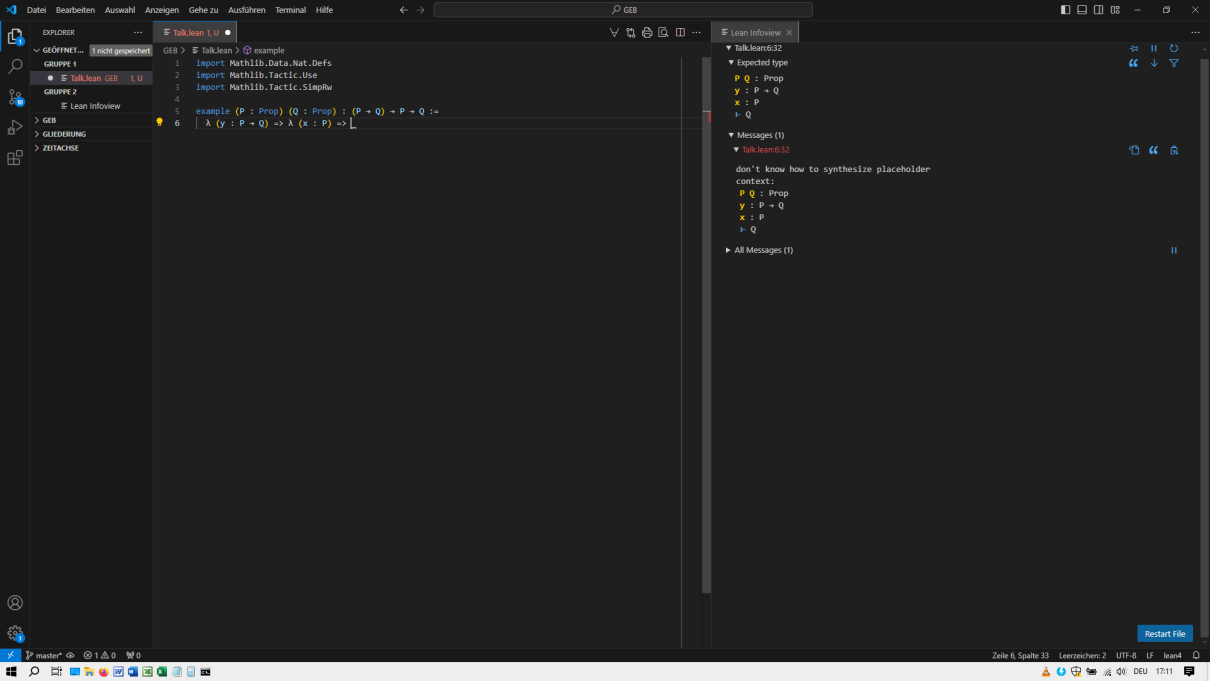

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) :
Lean Infoview
Talk.lean:5:32
Messages (1)
Talk.lean:5:32
unexpected end of input
All Messages (1)
Restart File
master*
1 0
0
Zeile 5, Spalte 33 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:08

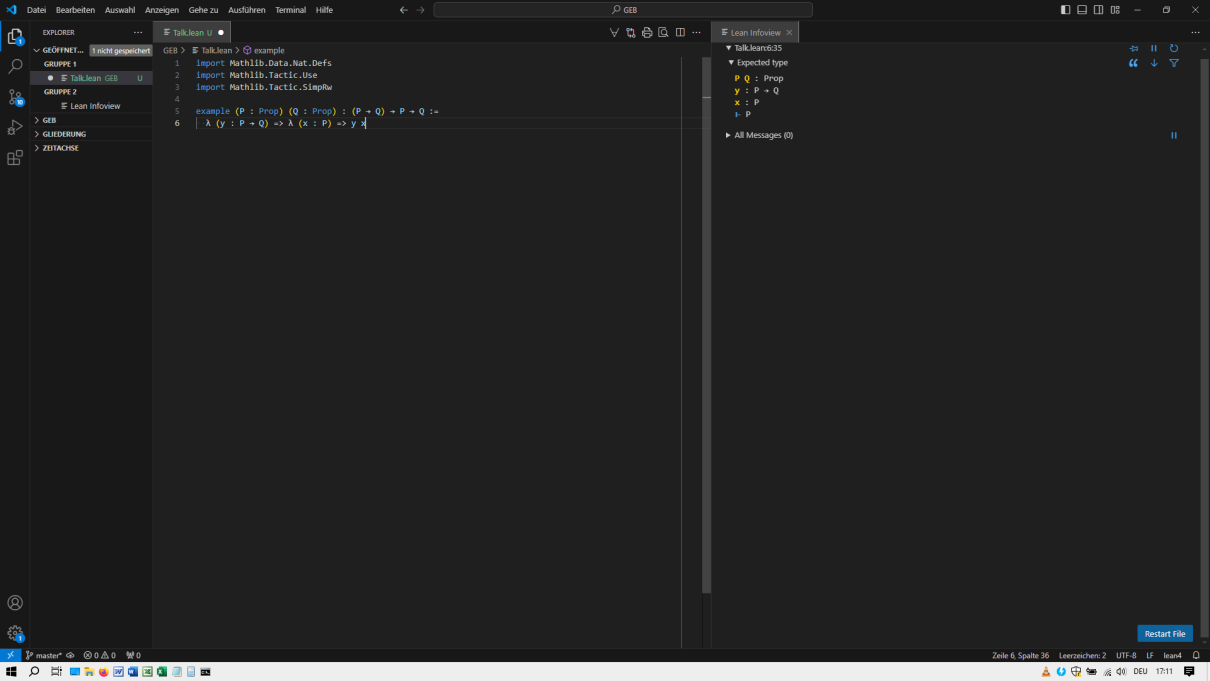

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q
Lean Infoview
Talk.lean:5:47
Expected type
P Q : Prop
⊢ Prop
Messages (1)
Talk.lean:5:47
unexpected end of input; expected ':=', 'where' or '|'
All Messages (1)
Restart File
master*
1 0
0
Zeile 5, Spalte 48 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:08

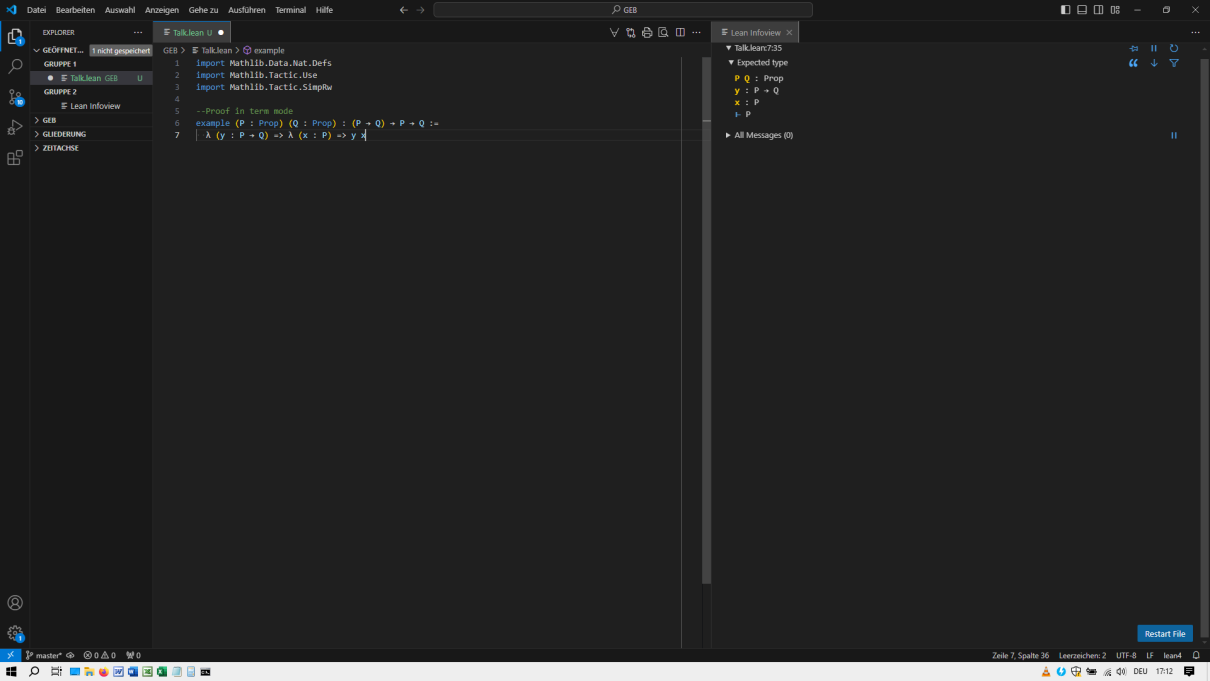

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
Lean Infoview
Talk.lean:5:50
Messages (1)
Talk.lean:5:50
unexpected end of input
All Messages (1)
Restart File
master*
Zeile 5, Spalte 51 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:09

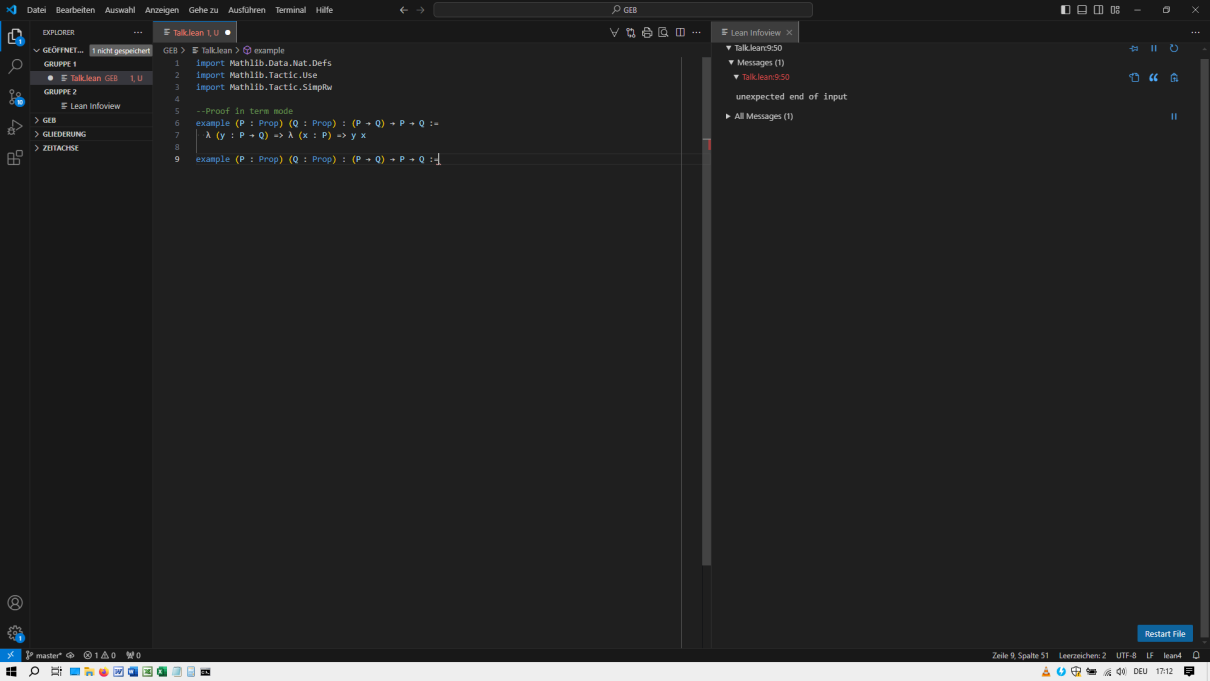

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
6
Lean Infoview
Talk.lean:6:2
Expected type
P Q : Prop
⊢ (P → Q) → P → Q
Messages (1)
Talk.lean:6:2
don't know how to synthesize placeholder
context:
P Q : Prop
⊢ (P → Q) → P → Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 6, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:10

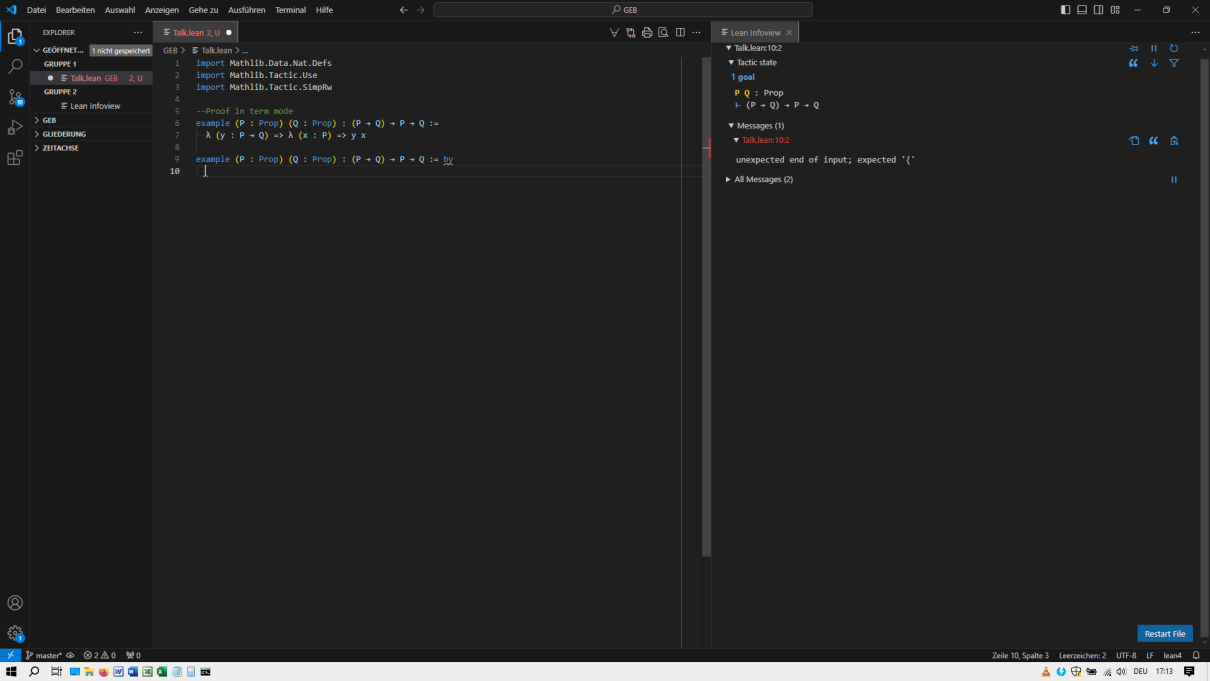

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
6 λ (y : P → Q) => _
Lean Infoview
Talk.lean:6:19
Expected type
P Q : Prop
y : P → Q
⊢ P → Q
Messages (1)
Talk.lean:6:19
don't know how to synthesize placeholder
context:
P Q : Prop
y : P → Q
⊢ P → Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 6, Spalte 20
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:10

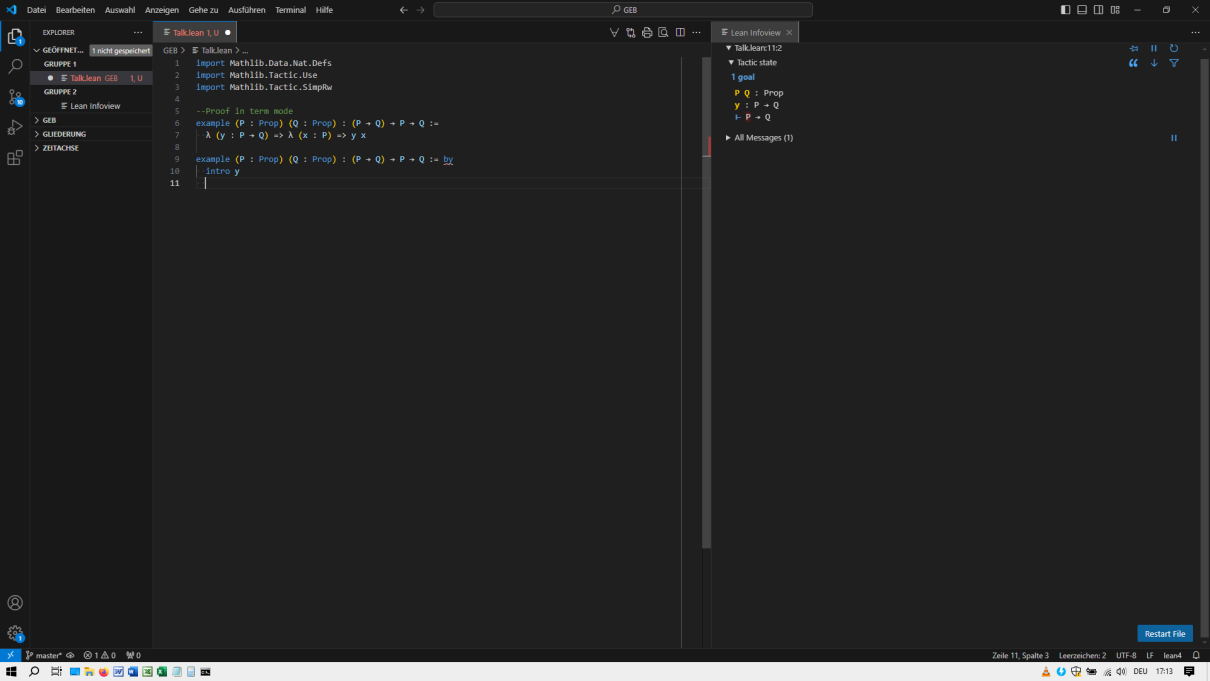

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
6 λ (y : P → Q) => λ (x : P) =>
Lean Infoview
Talk.lean:6:32
Expected type
P Q : Prop
y : P → Q
x : P
⊢ Q
Messages (1)
Talk.lean:6:32
don't know how to synthesize placeholder
context:
P Q : Prop
y : P → Q
x : P
⊢ Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 6, Spalte 33 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:11

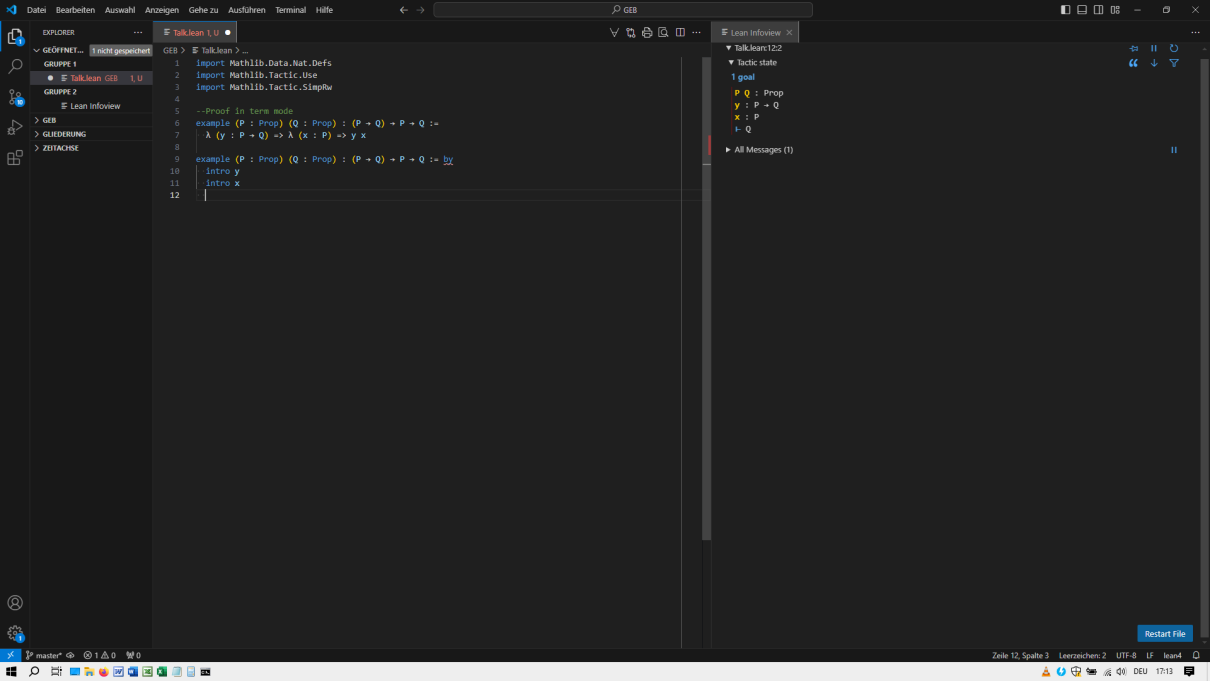

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
6 λ (y : P → Q) => λ (x : P) => y x
Lean Infoview
Talk.lean:6:35
Expected type
P Q : Prop
y : P → Q
x : P
⊢ P
All Messages (0)
Restart File
master*
0 0
0
Zeile 6, Spalte 36 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:11

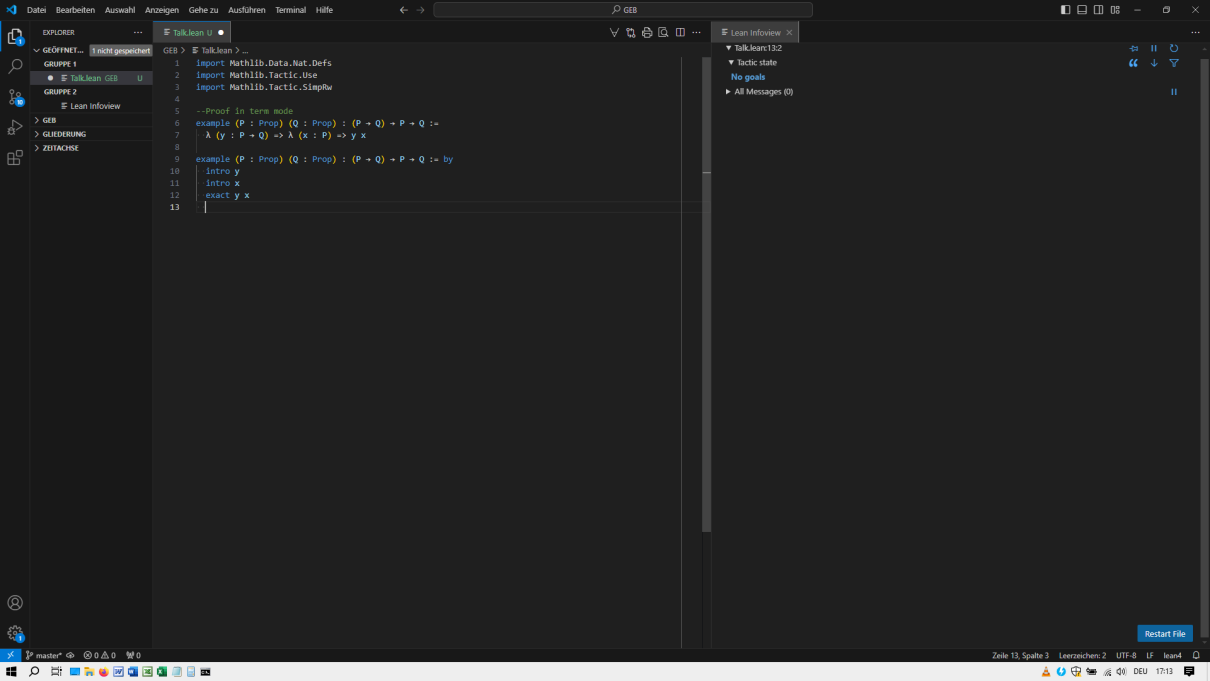

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
Lean Infoview
Talk.lean:7:35
Expected type
P Q : Prop
y : P → Q
x : P
⊢ P
All Messages (0)
Restart File
master*
0 0
0
Zeile 7, Spalte 36 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:12

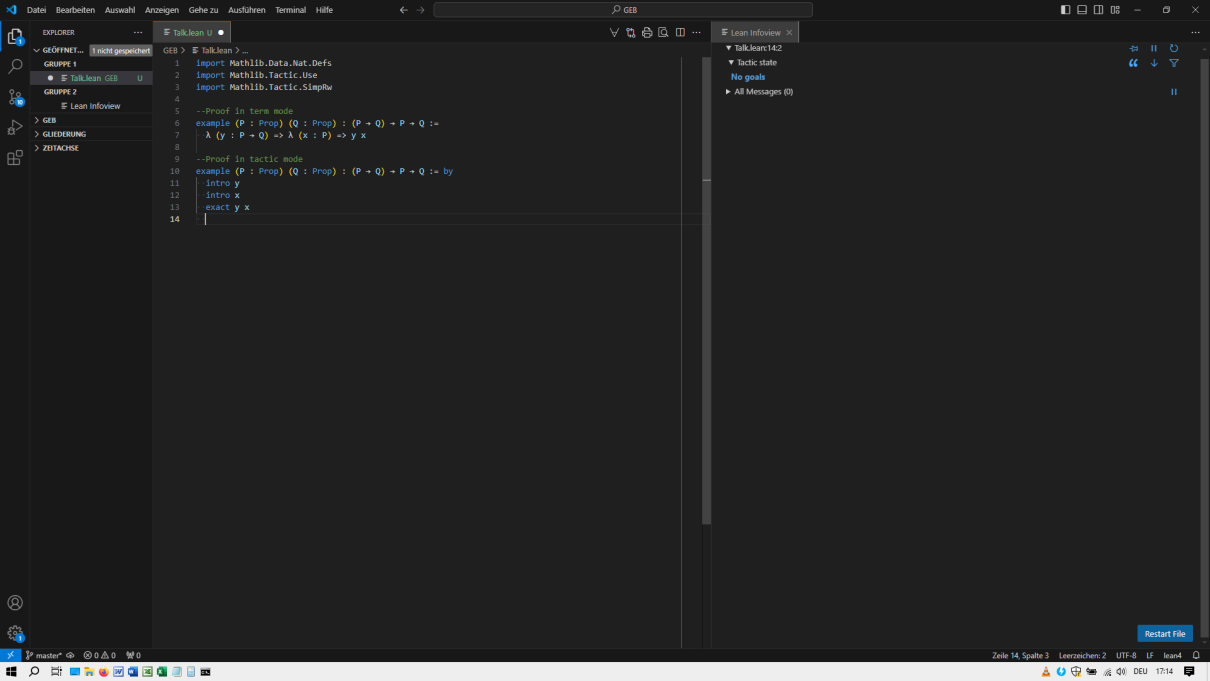

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :
Lean Infoview
Talk.lean:9:50
Messages (1)
Talk.lean:9:50
unexpected end of input
All Messages (1)
Restart File
master*
1 0
0
Zeile 9, Spalte 51 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:12

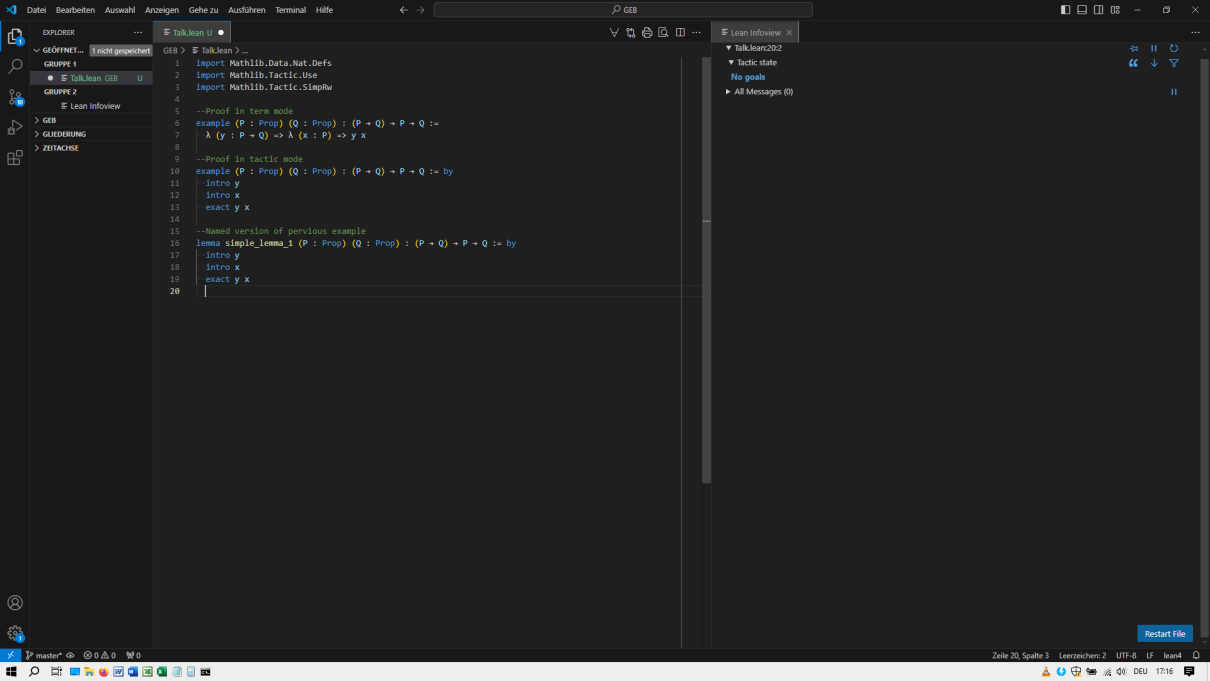

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
10
Lean Infoview
Talk.lean:10:2
Tactic state
1 goal
P Q : Prop
⊢ (P → Q) → P → Q
Messages (1)
Talk.lean:10:2
unexpected end of input; expected '{'
All Messages (2)
Restart File
master*
2 0
0
Zeile 10, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:13

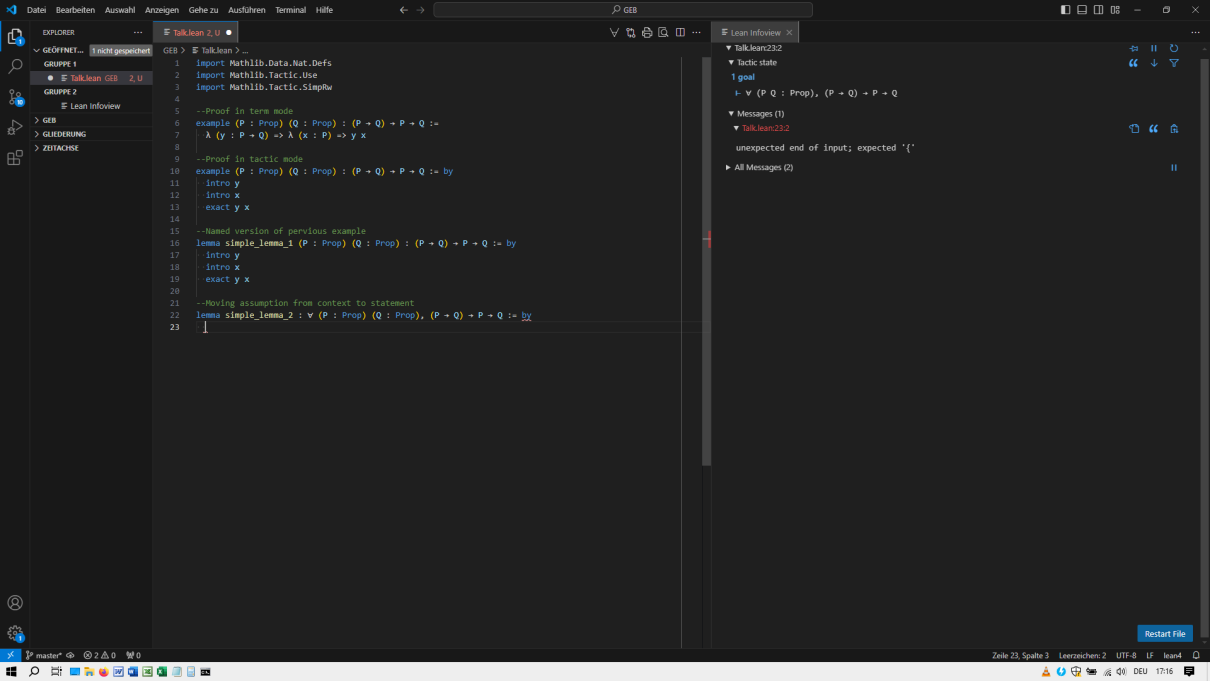

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
import Mathlib.Data.Nat.Defs
import Mathlib.Tactic.Use
import Mathlib.Tactic.SimpRw
--Proof in term mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
λ (y : P → Q) => λ (x : P) => y x
example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
Lean Infoview
Talk.lean:11:2
Tactic state
1 goal
P Q : Prop
y : P → Q
⊢ P → Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 11, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:13

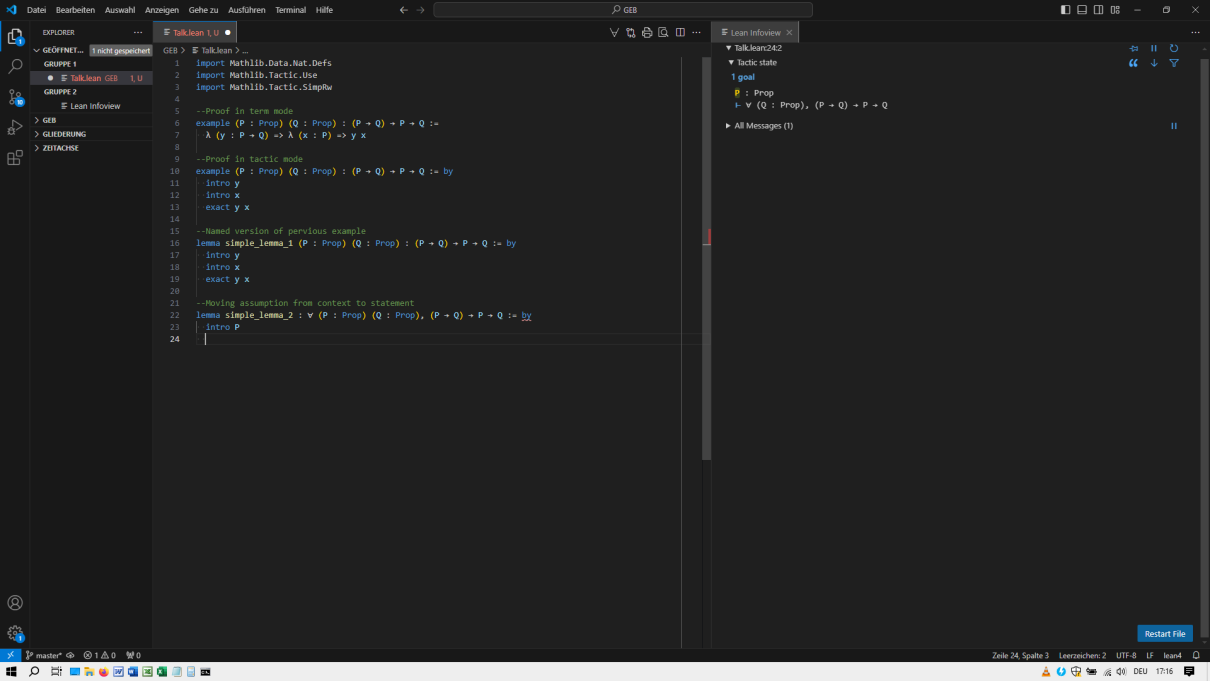

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
10 intro y
11 intro x
12
Lean Infoview
Talk.lean:12:2
Tactic state
1 goal
P Q : Prop
y : P → Q
x : P
⊢ Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 12, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:13

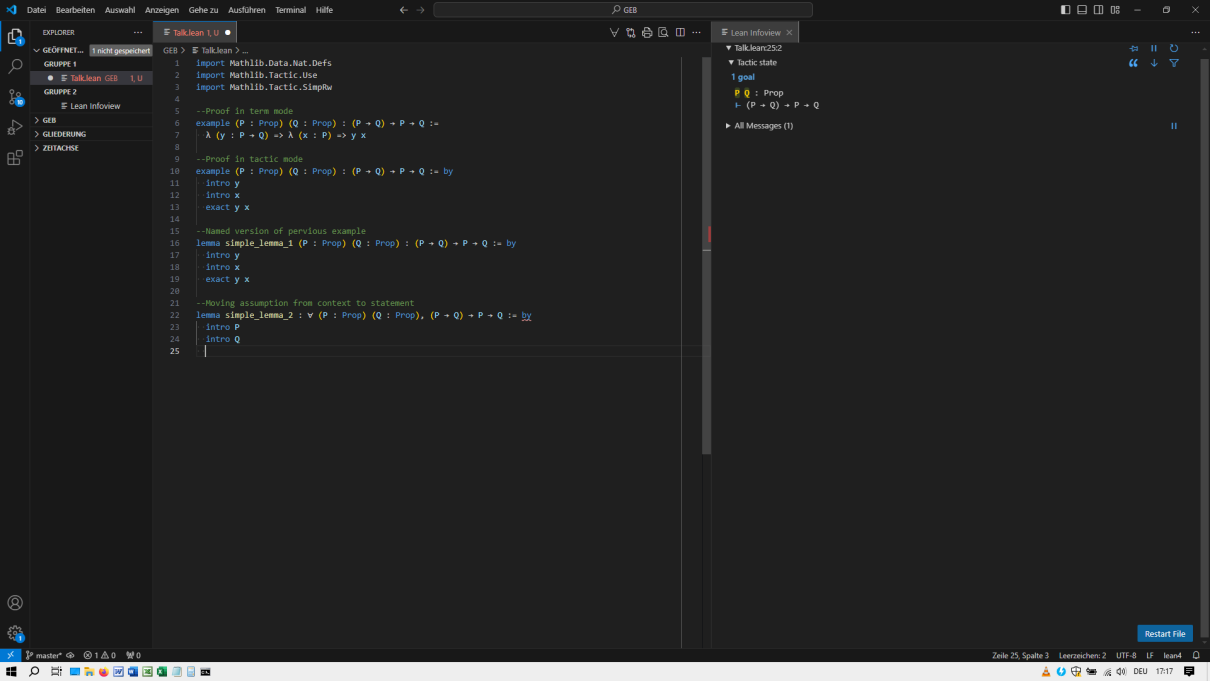

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
10 intro y
11 intro x
12 exact y x
13
Lean Infoview
Talk.lean:13:2
Tactic state
No goals
All Messages (0)
Restart File
master*
0 0
0
Zeile 13, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:13

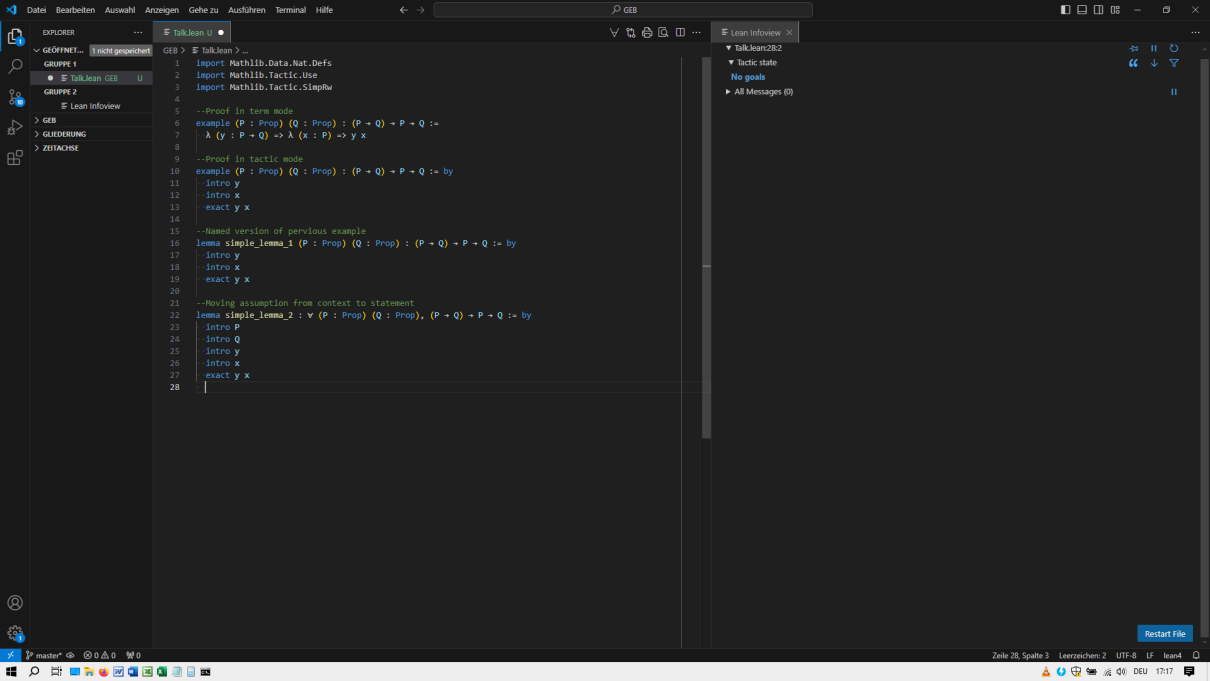

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
Lean Infoview
Talk.lean:14:2
Tactic state
No goals
All Messages (0)
Restart File
master*
0 0
0
Zeile 14, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:14

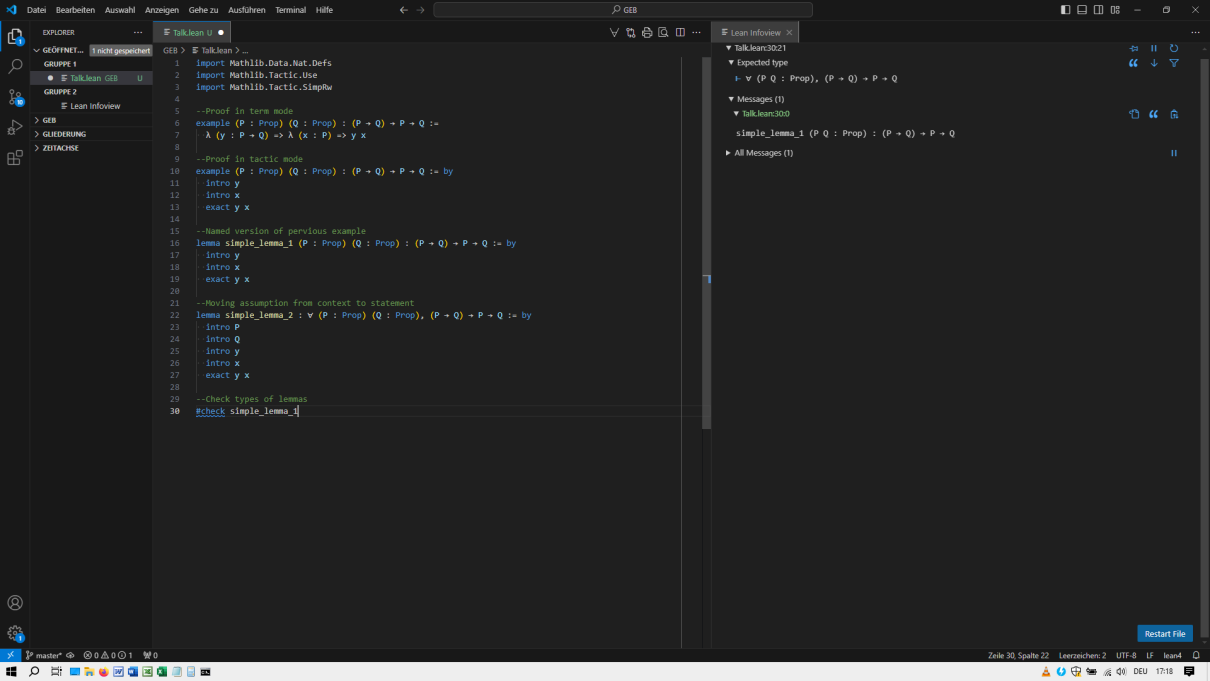

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
Lean Infoview
Talk.lean:20:2
Tactic state
No goals
All Messages (0)
Restart File
master*
Zeile 20, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:16

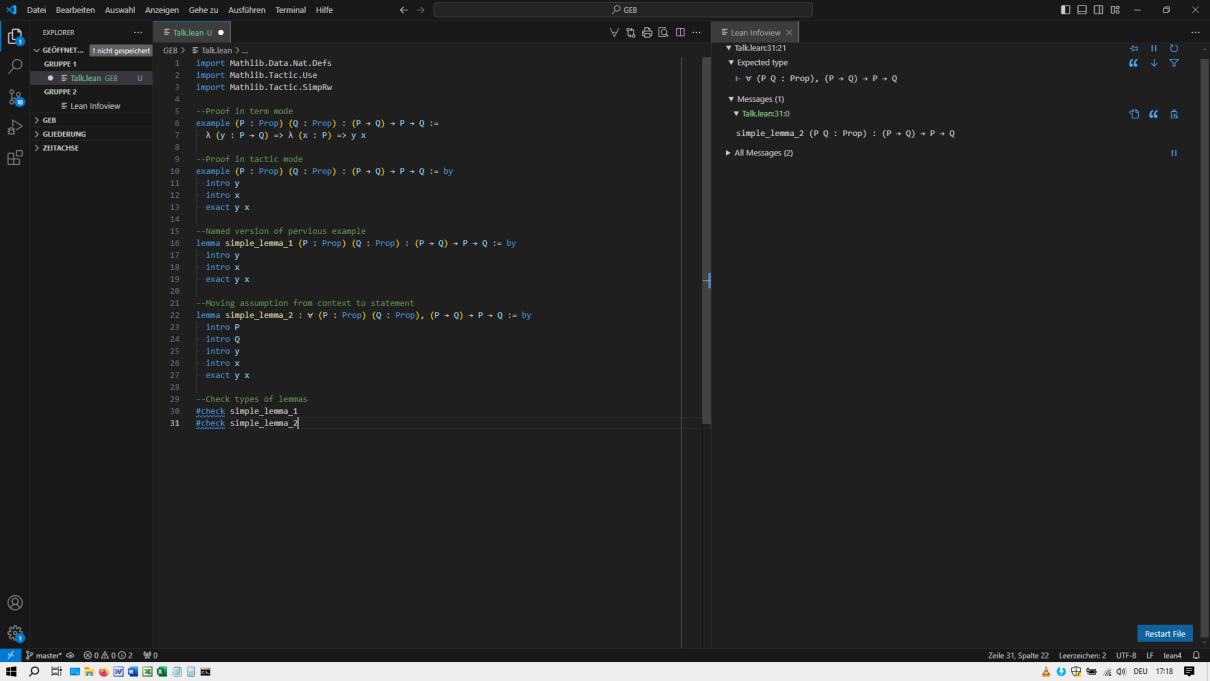

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23
Lean Infoview
Talk.lean:23:2
Tactic state
1 goal
⊢ ∀ (P Q : Prop), (P → Q) → P → Q
Messages (1)
Talk.lean:23:2
unexpected end of input; expected '{'
All Messages (2)
Restart File
master*
2 0
0
Zeile 23, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:16

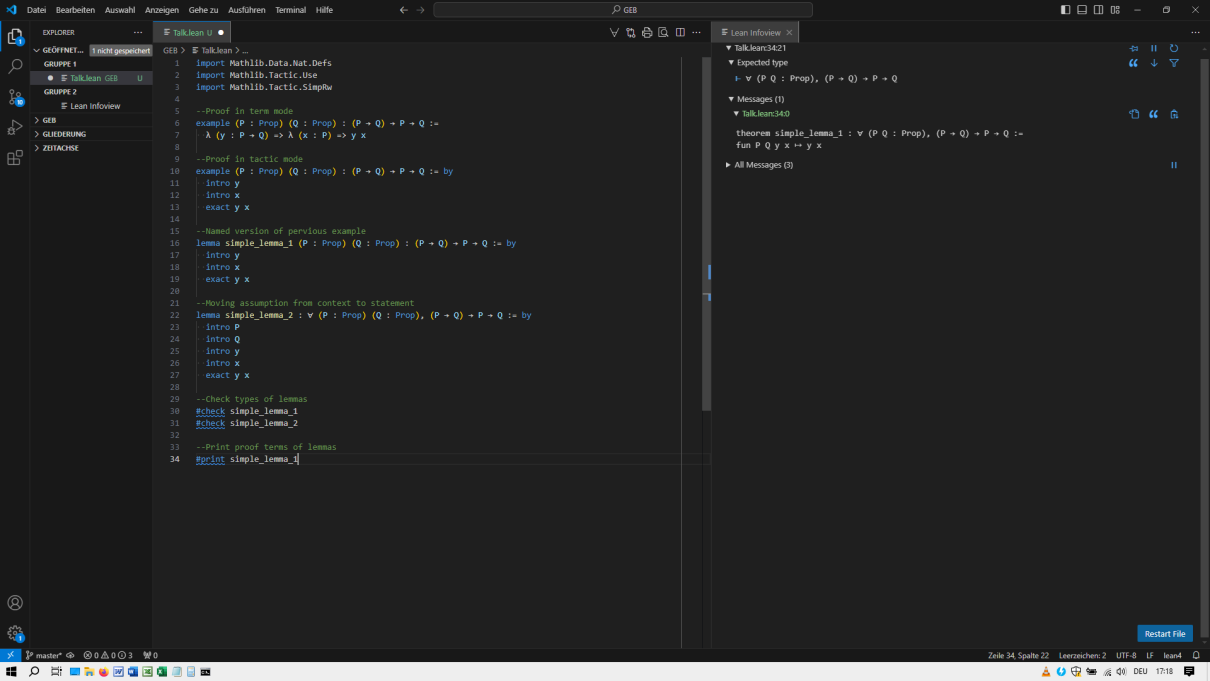

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24
Lean Infoview
Talk.lean:24:2
Tactic state
1 goal
P : Prop
⊢ ∀ (Q : Prop), (P → Q) → P → Q
All Messages (1)
Restart File
master*
1 0
0
Zeile 24, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:16

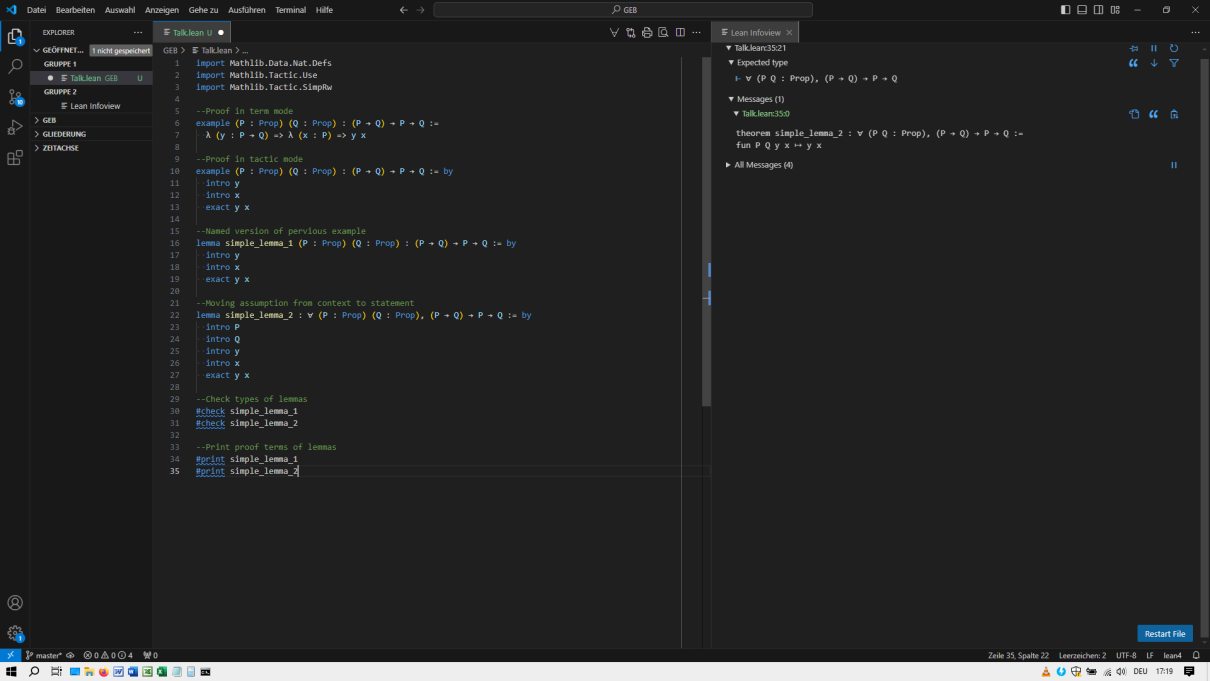

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24 intro Q
25
Lean Infoview
Talk.lean:25:2
Tactic state
1 goal
P Q : Prop
⊢ (P → Q) → P → Q
All Messages (1)
Restart File
master*
Zeile 25, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:17

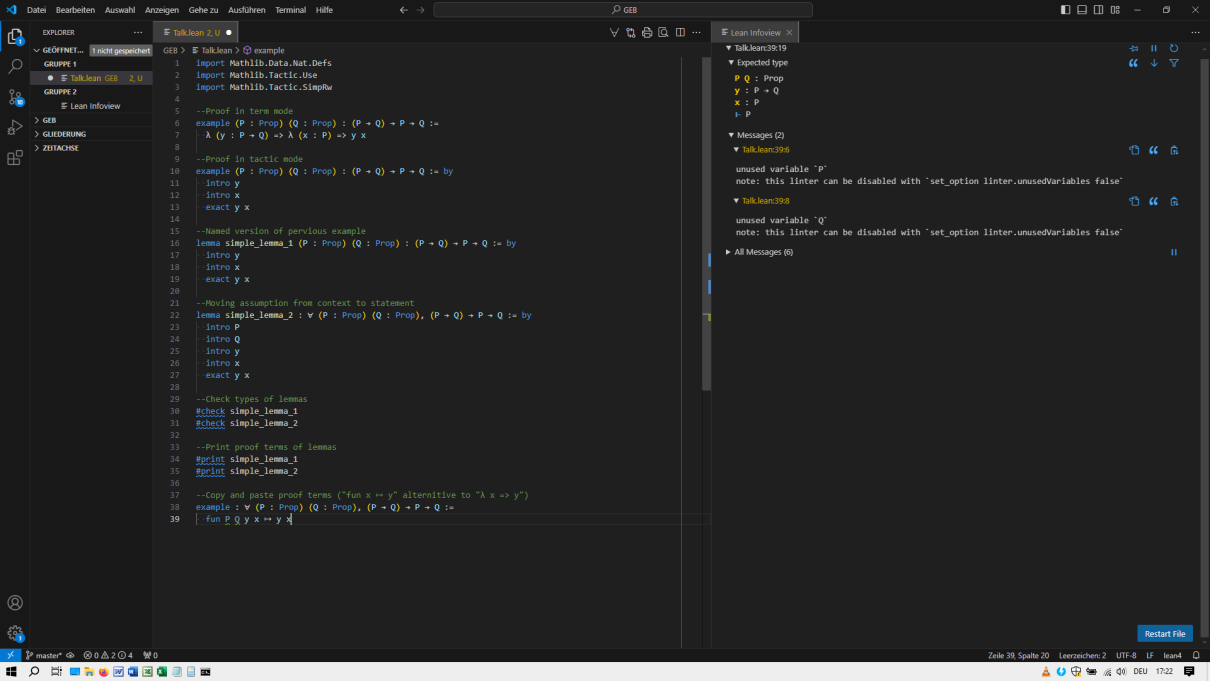

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24 intro Q
25 intro y
26 intro x
27 exact y x
28
Lean Infoview
Talk.lean:28:2
Tactic state
No goals
All Messages (0)
Restart File
master*
0 0
0
Zeile 28, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:17

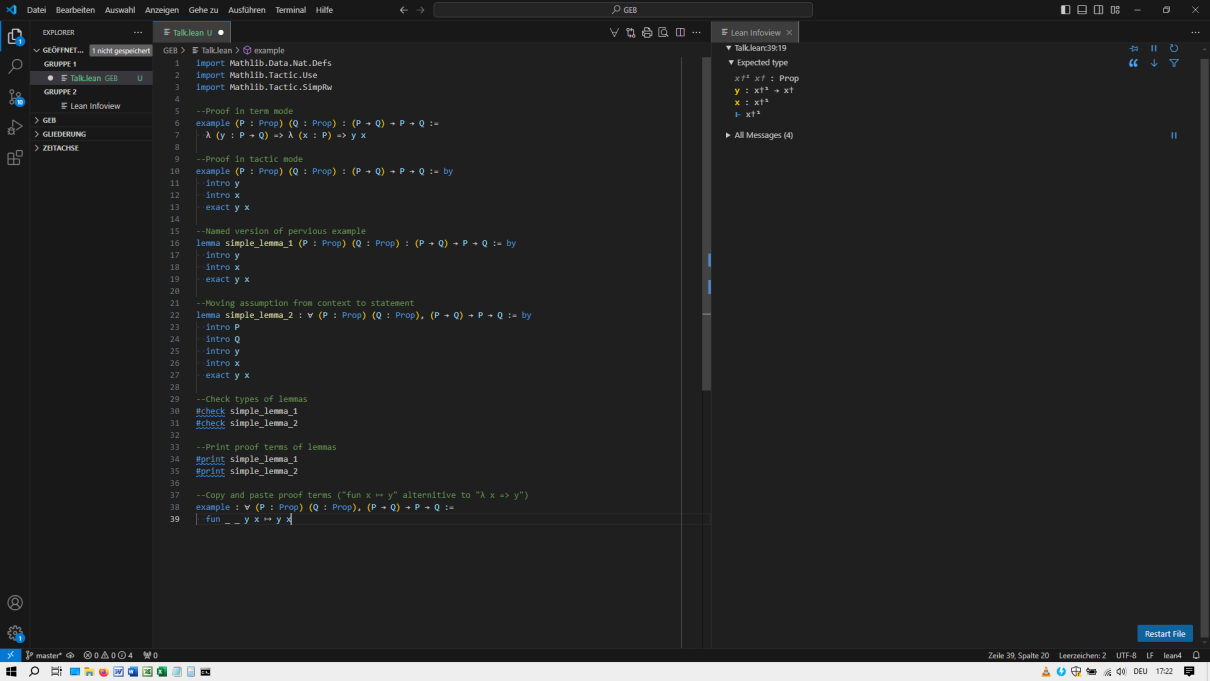

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
import Mathlib.Data.Nat.Defs
import Mathlib.Tactic.Use
import Mathlib.Tactic.SimpRw
--Proof in term mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
λ (y : P → Q) => λ (x : P) => y x
--Proof in tactic mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Named version of pervious example
lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Moving assumption from context to statement
lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
intro P
intro Q
intro y
intro x
exact y x
--Check types of lemmas
#check simple_lemma_1
Lean Infoview
Talk.lean:30:21
Expected type
∀ {P Q : Prop}, (P → Q) → P → Q
Messages (1)
Talk.lean:30:0
simple_lemma_1 (P Q : Prop) : (P → Q) → P → Q
All Messages (1)
Restart File
master*
Zeile 30, Spalte 22 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:18

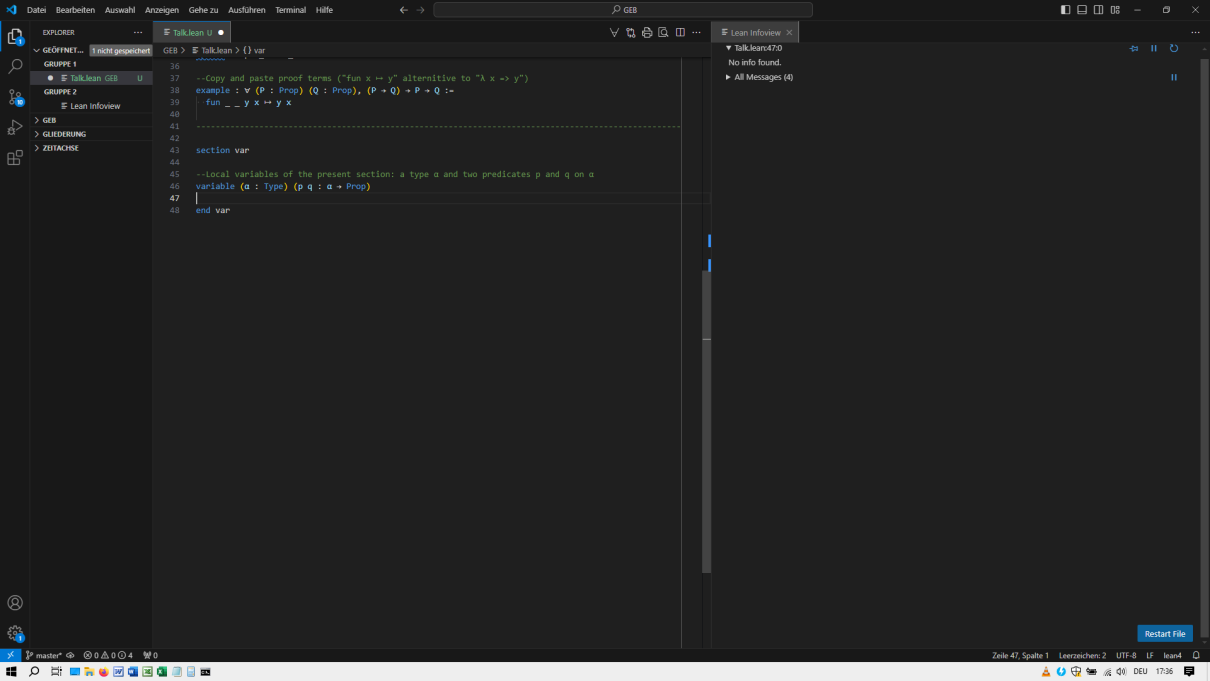

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24 intro Q
25 intro y
26 intro x
27 exact y x
28
29 --Check types of lemmas
30 #check simple_lemma_1
31 #check simple_lemma_2
Lean Infoview
Talk.lean:31:21
Expected type
⊢ ∀ {P Q : Prop}, (P → Q) → P → Q
Messages (1)
Talk.lean:31:0
simple_lemma_2 {P Q : Prop} : (P → Q) → P → Q
All Messages (2)
Restart File
master*
0 0 2
0
Zeile 31, Spalte 22
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:18

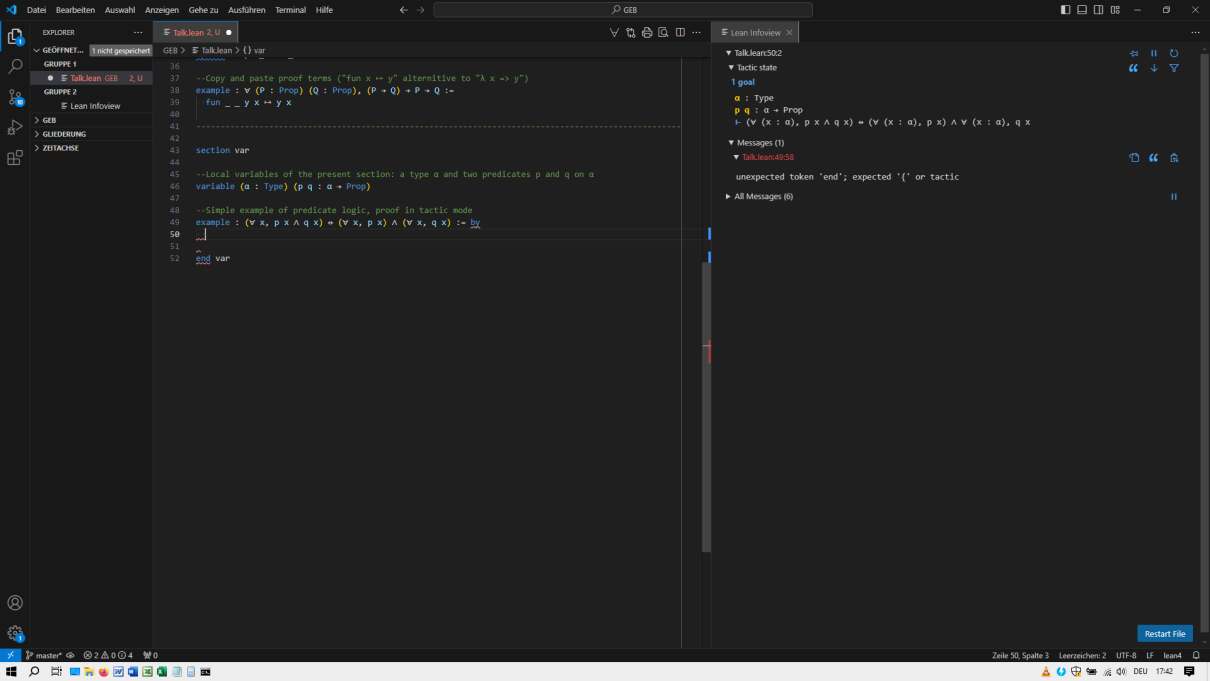

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24 intro Q
25 intro y
26 intro x
27 exact y x
28
29 --Check types of lemmas
30 #check simple_lemma_1
31 #check simple_lemma_2
32
33 --Print proof terms of lemmas
34 #print simple_lemma_1
Lean Infoview
Talk.lean:34:21
Expected type
⊢ ∀ {P Q : Prop}, (P → Q) → P → Q
Messages (1)
Talk.lean:34:0
theorem simple_lemma_1 : ∀ {P Q : Prop}, (P → Q) → P → Q :=
fun P Q y x ↦ y x
All Messages (3)
Restart File
master*
0 0 3
0
Zeile 34, Spalte 22
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:18

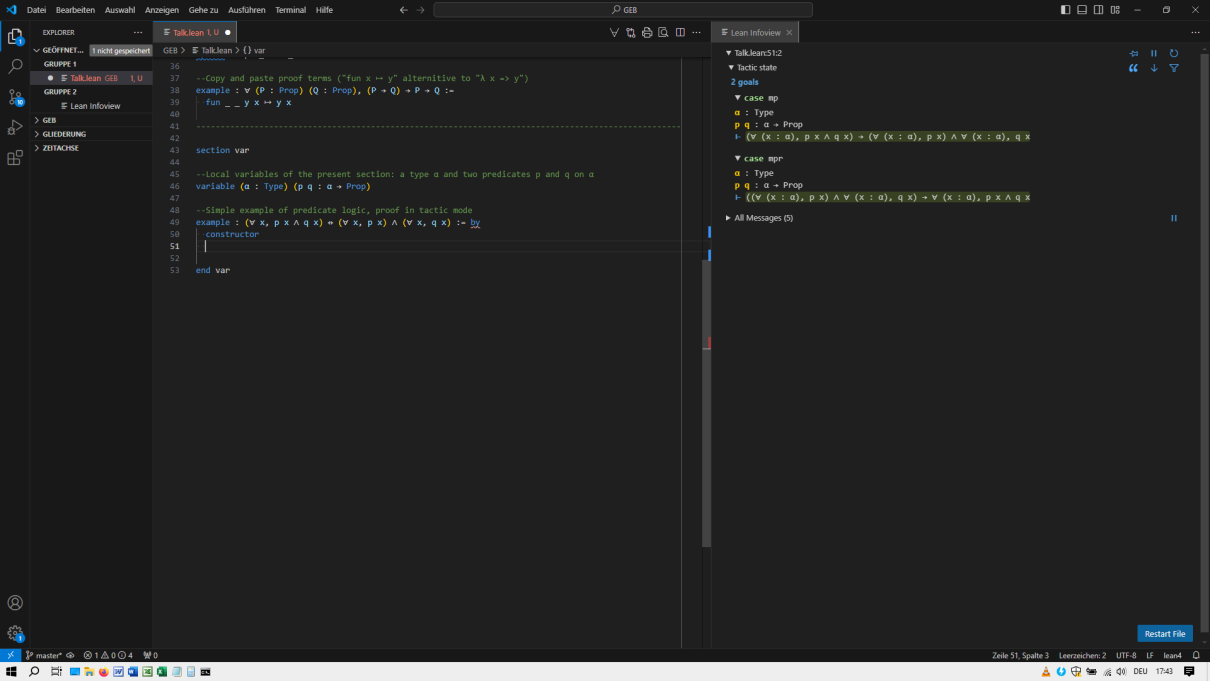

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
import Mathlib.Data.Nat.Defs
import Mathlib.Tactic.Use
import Mathlib.Tactic.SimpRw
--Proof in term mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
λ (y : P → Q) => λ (x : P) => y x
--Proof in tactic mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Named version of pervious example
lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Moving assumption from context to statement
lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
intro P
intro Q
intro y
intro x
exact y x
--Check types of lemmas
#check simple_lemma_1
#check simple_lemma_2
--Print proof terms of lemmas
#print simple_lemma_1
#print simple_lemma_2
Lean Infoview
Talk.lean:35:21
Expected type
⊢ ∀ {P Q : Prop}, (P → Q) → P → Q
Messages (1)
Talk.lean:35:0
theorem simple_lemma_2 : ∀ {P Q : Prop}, (P → Q) → P → Q :=
fun P Q y x ↦ y x
All Messages (4)
Restart File
master*
Zeile 35, Spalte 22
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:19

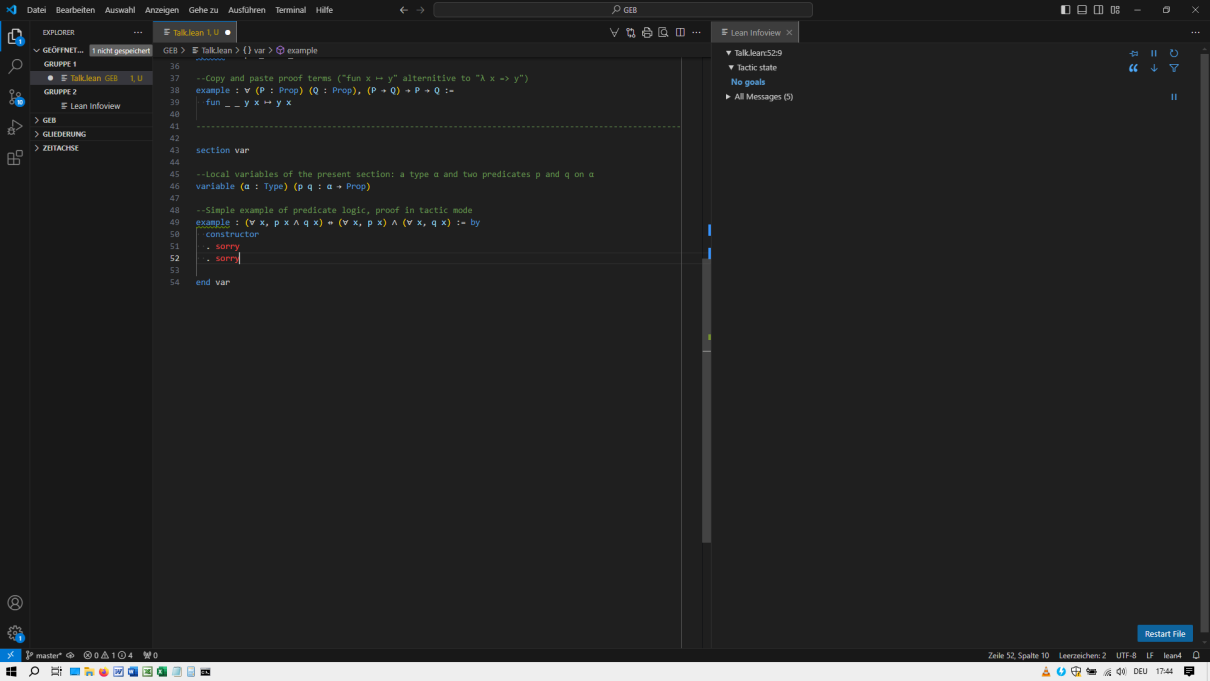

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > example
1 import Mathlib.Data.Nat.Defs
2 import Mathlib.Tactic.Use
3 import Mathlib.Tactic.SimpRw
4
5 --Proof in term mode
6 example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
7 λ (y : P → Q) => λ (x : P) => y x
8
9 --Proof in tactic mode
10 example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
11 intro y
12 intro x
13 exact y x
14
15 --Named version of pervious example
16 lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
17 intro y
18 intro x
19 exact y x
20
21 --Moving assumption from context to statement
22 lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
23 intro P
24 intro Q
25 intro y
26 intro x
27 exact y x
28
29 --Check types of lemmas
30 #check simple_lemma_1
31 #check simple_lemma_2
32
33 --Print proof terms of lemmas
34 #print simple_lemma_1
35 #print simple_lemma_2
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun P Q y x ↦ y x
Lean Infoview
Talk.lean:39:19
Expected type
P Q : Prop
y : P → Q
x : P
⊢ P
Messages (2)
Talk.lean:39:6
unused variable `P`
note: this linter can be disabled with `set_option linter.unusedVariables false`
Talk.lean:39:8
unused variable `Q`
note: this linter can be disabled with `set_option linter.unusedVariables false`
All Messages (6)
Restart File
master*
0 2 4
0
Zeile 39, Spalte 20 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:22

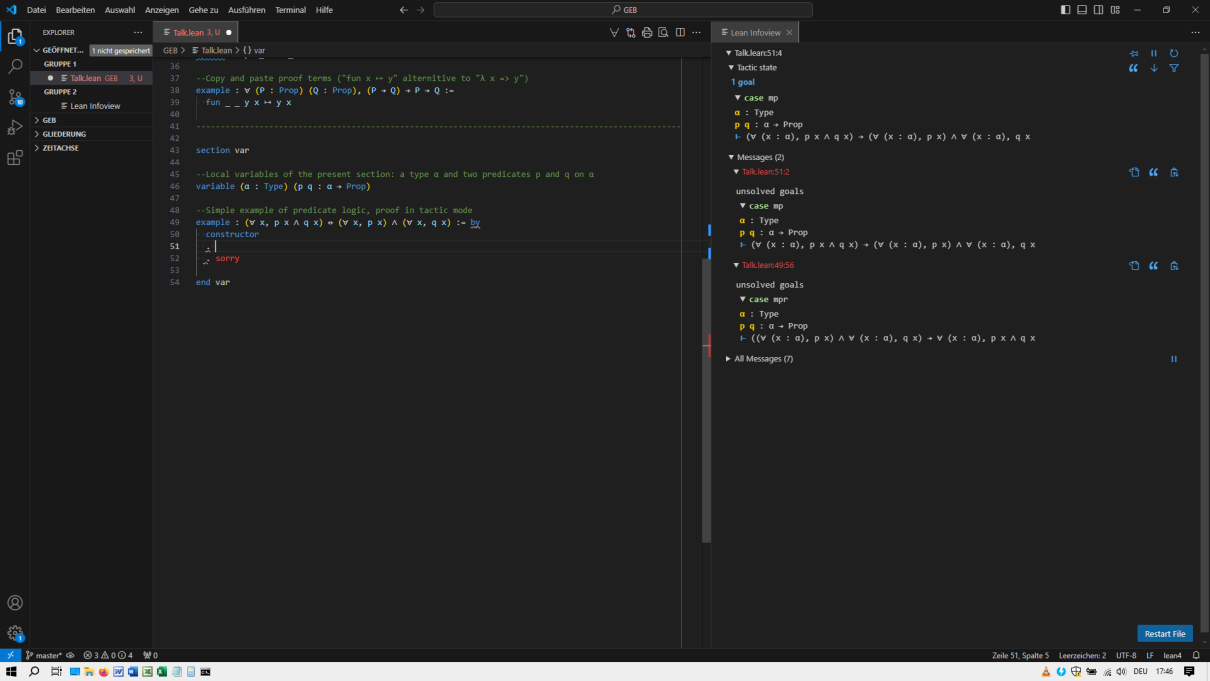

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > example
import Mathlib.Data.Nat.Defs
import Mathlib.Tactic.Use
import Mathlib.Tactic.SimpRw
--Proof in term mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q :=
λ (y : P → Q) => λ (x : P) => y x
--Proof in tactic mode
example (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Named version of pervious example
lemma simple_lemma_1 (P : Prop) (Q : Prop) : (P → Q) → P → Q := by
intro y
intro x
exact y x
--Moving assumption from context to statement
lemma simple_lemma_2 : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q := by
intro P
intro Q
intro y
intro x
exact y x
--Check types of lemmas
#check simple_lemma_1
#check simple_lemma_2
--Print proof terms of lemmas
#print simple_lemma_1
#print simple_lemma_2
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
Lean Infoview
Talk.lean:39:19
Expected type
xf² xf : Prop
y : xf¹ → xf
x : xf¹
⊢ xf¹
All Messages (4)
Restart File
master*
Zeile 39, Spalte 20 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:22

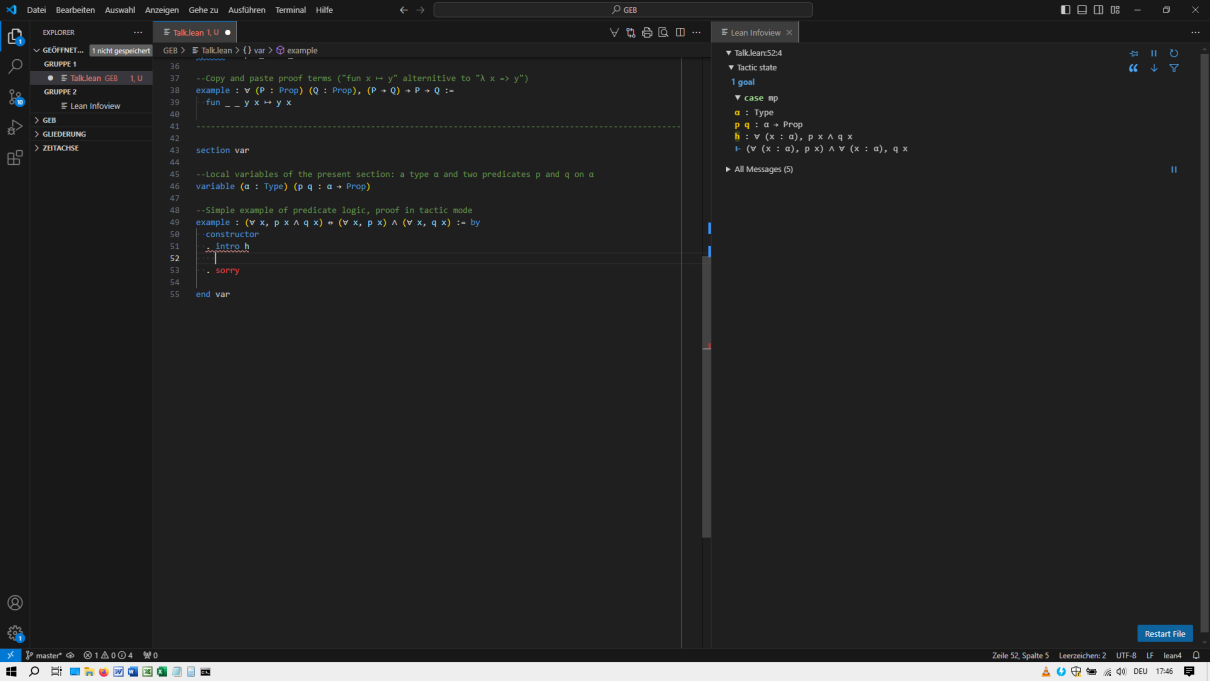

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
end var
Lean Infoview
Talk.lean:47:0
No info found.
All Messages (4)
Restart File
master*
Zeile 47, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:36

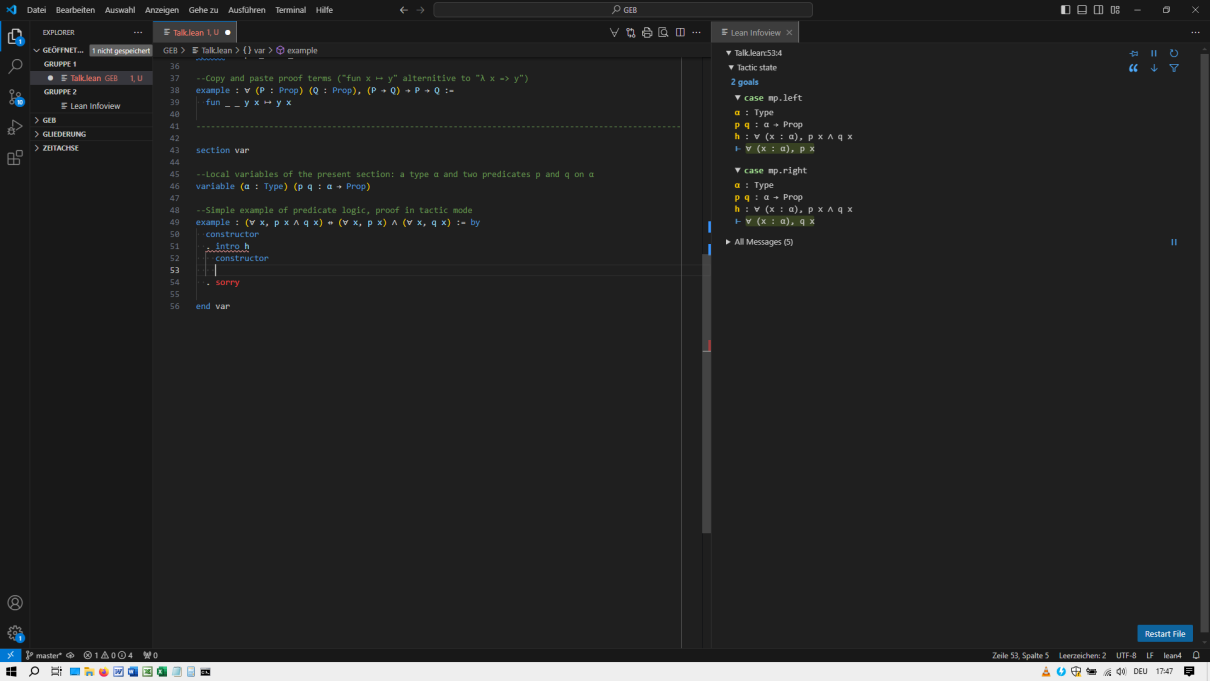

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ⇔ (∀ x, p x) ∧ (∀ x, q x) := by
end var
Lean Infoview
Talk.lean:50:2
Tactic state
1 goal
α : Type
p q : α → Prop
⊢ (∀ (x : α), p x ∧ q x) ⇔ (∀ (x : α), p x) ∧ ∀ (x : α), q x
Messages (1)
Talk.lean:49:58
unexpected token 'end'; expected '{' or tactic
All Messages (6)
Restart File
master*
2 0 4
0
Zeile 50, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:42

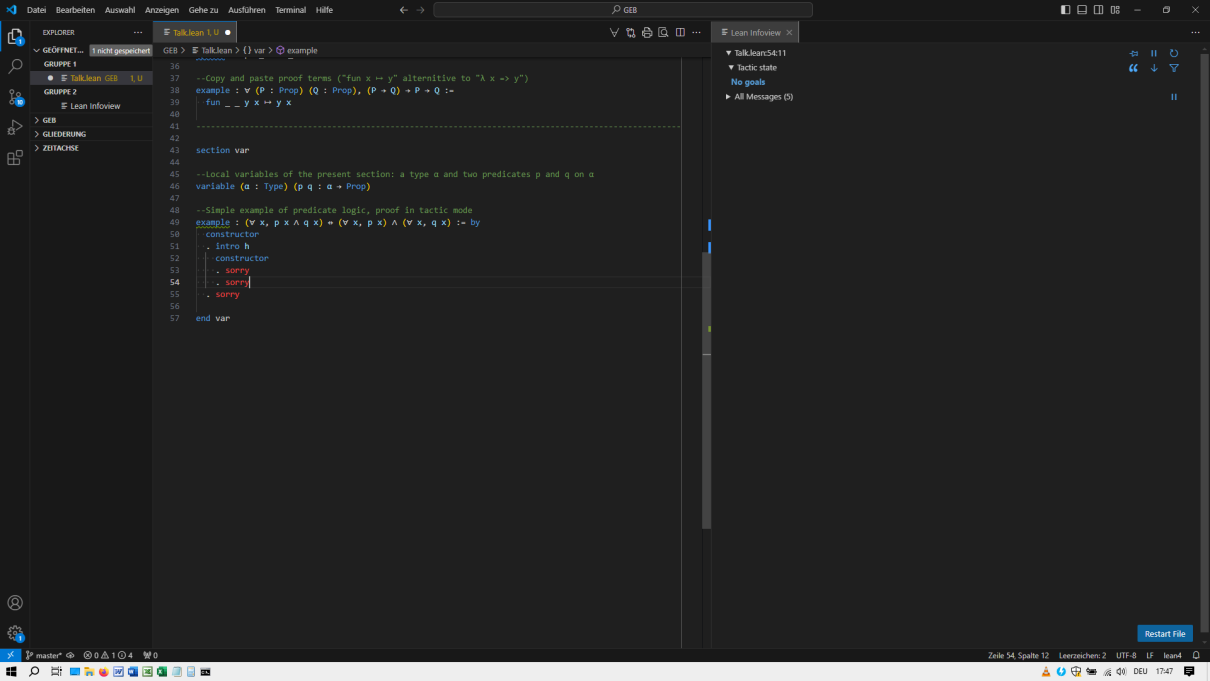

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
end var
Lean Infoview
Talk.lean:51:2
Tactic state
2 goals
case mp
α : Type
p q : α → Prop
⊢ (∀ (x : α), p x ∧ q x) → (∀ (x : α), p x) ∧ ∀ (x : α), q x
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
All Messages (5)
Restart File
master*
Zeile 51, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:43

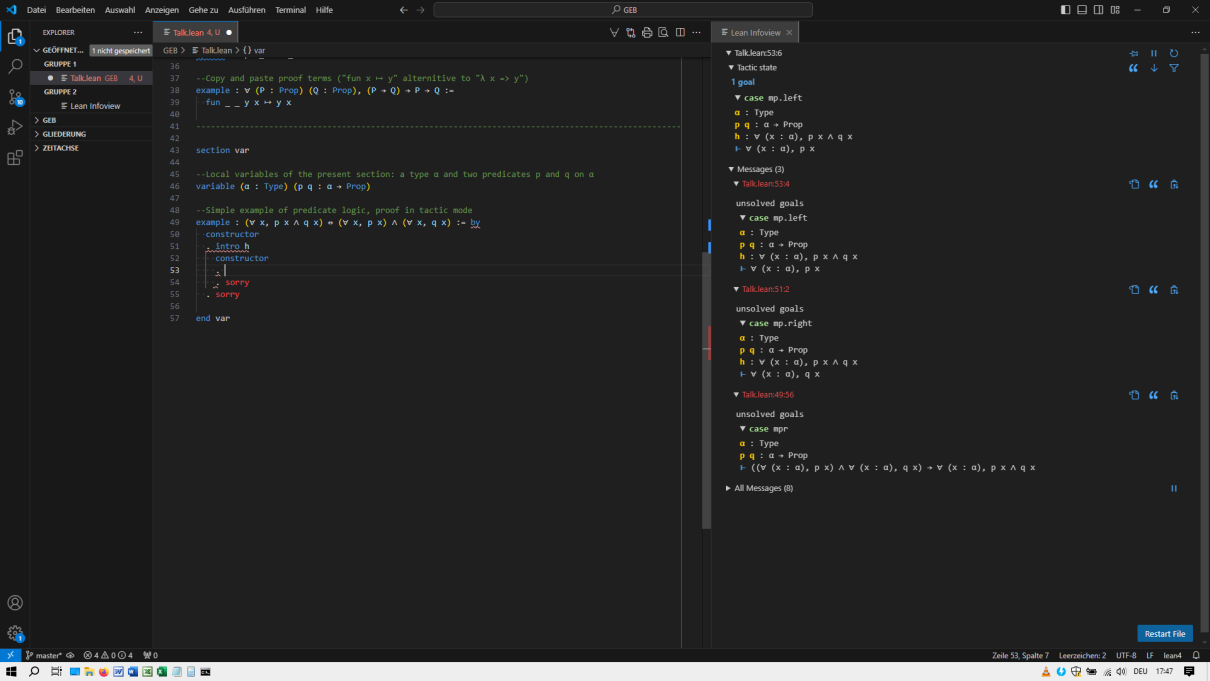

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. sorry
. sorry
end var
Lean Infoview
Talk.lean:52:9
Tactic state
No goals
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 52, Spalte 10
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:44

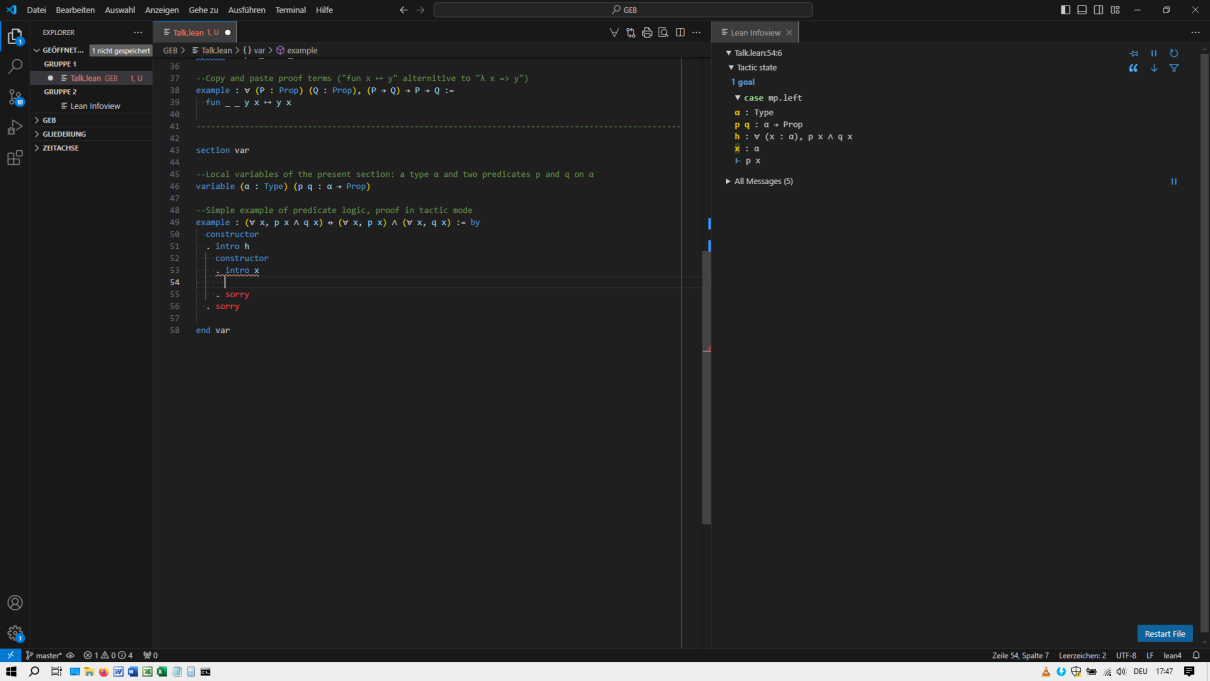

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 3, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 3, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
· sorry
end var
Lean Infoview
Talk.lean:51:4
Tactic state
1 goal
case mp
α : Type
p q : α → Prop
⊢ (∀ (x : α), p x ∧ q x) → (∀ (x : α), p x) ∧ ∀ (x : α), q x
Messages (2)
Talk.lean:51:2
unsolved goals
case mp
α : Type
p q : α → Prop
⊢ (∀ (x : α), p x ∧ q x) → (∀ (x : α), p x) ∧ ∀ (x : α), q x
Talk.lean:49:56
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
All Messages (7)
Restart File
master*
3 0 4
0
Zeile 51, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:46

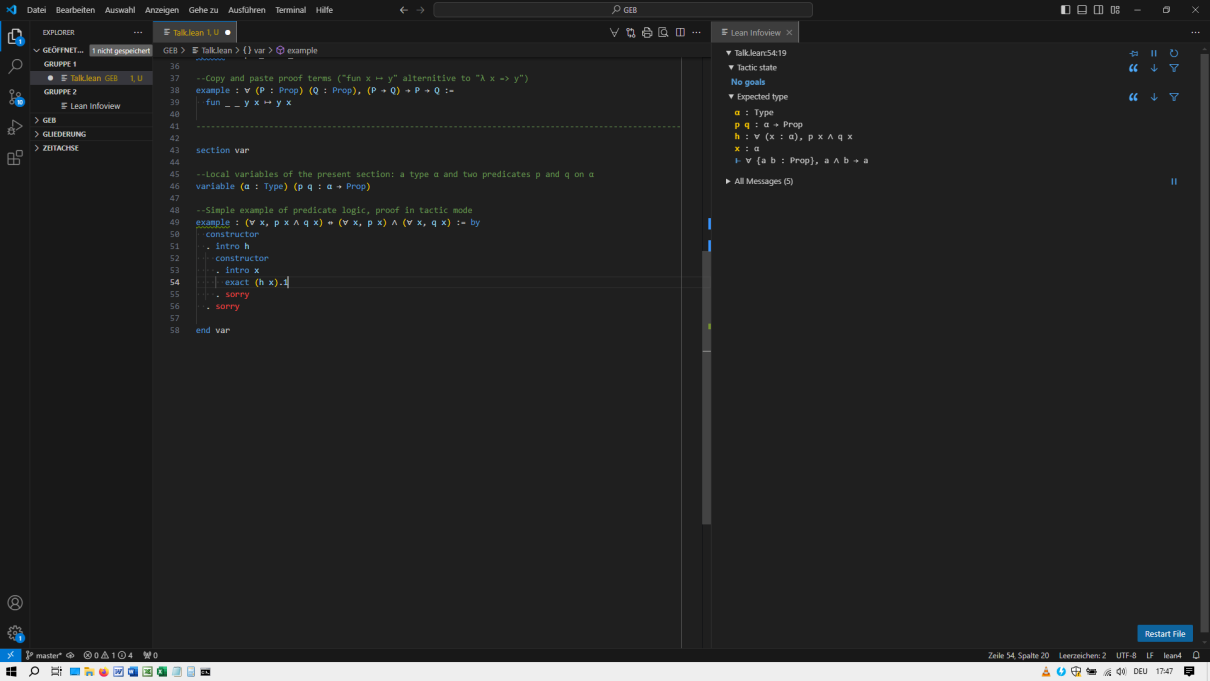

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52
53 . sorry
54
55 end var
Lean Infoview
Talk.lean:52:4
Tactic state
1 goal
case mp
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ (∀ (x : α), p x) ∧ ∀ (x : α), q x
All Messages (5)
Restart File
master*
1 0 4
0
Zeile 52, Spalte 5 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:46

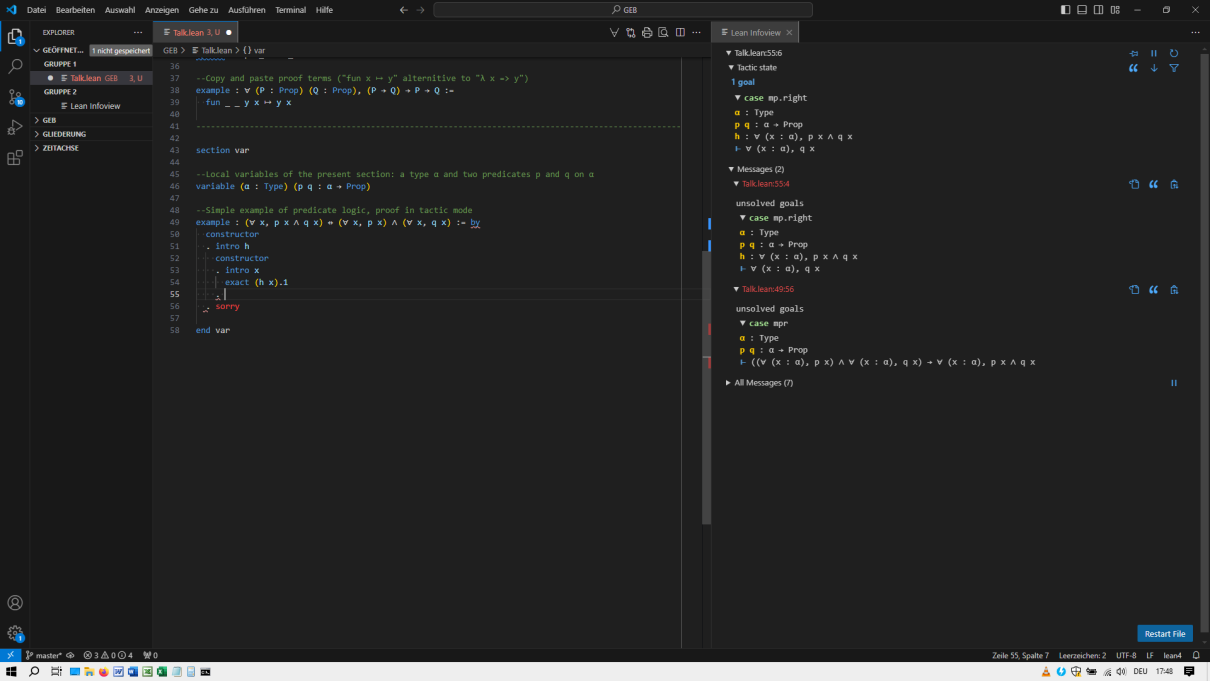

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
· intro h
constructor
· sorry
end var
Lean Infoview
Talk.lean:53:4
Tactic state
2 goals
case mp.left
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), p x
case mp.right
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), q x
All Messages (5)
Restart File
master*
1 0 4
0
Zeile 53, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:47

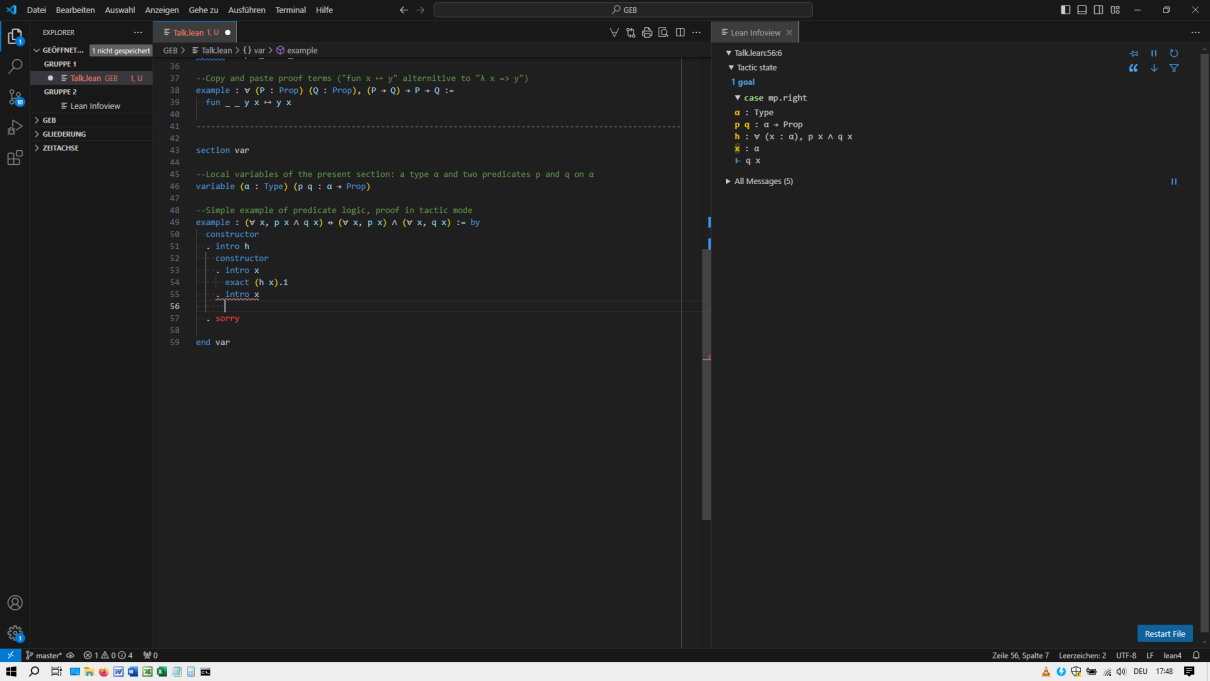

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. sorry
. sorry
. sorry
end var
Lean Infoview
Talk.lean:54:11
Tactic state
No goals
All Messages (5)
Restart File
master*
Zeile 54, Spalte 12 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:47

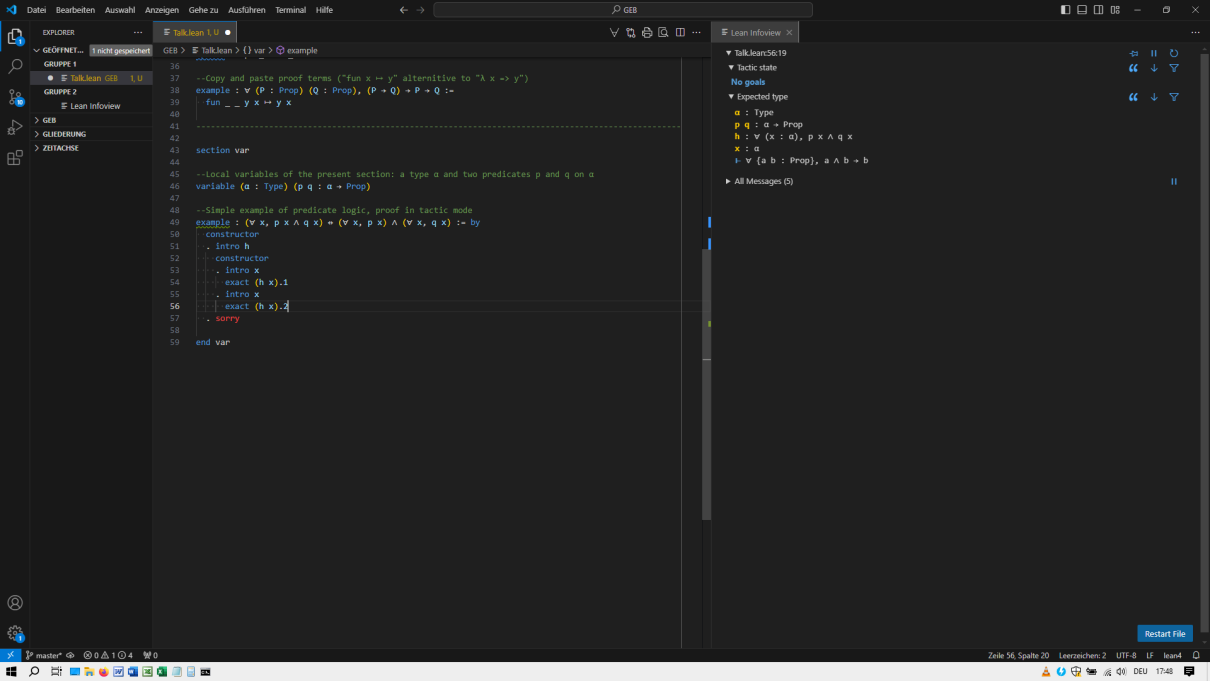

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 4, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 4, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
· intro h
constructor
·
· sorry
· sorry
end var
Lean Infoview
Talk.lean:53:6
Tactic state
1 goal
case mp.left
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), p x
Messages (3)
Talk.lean:53:4
unsolved goals
case mp.left
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), p x
Talk.lean:51:2
unsolved goals
case mp.right
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), q x
Talk.lean:49:56
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
All Messages (8)
Restart File
master*
4 0 4
0
Zeile 53, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:47

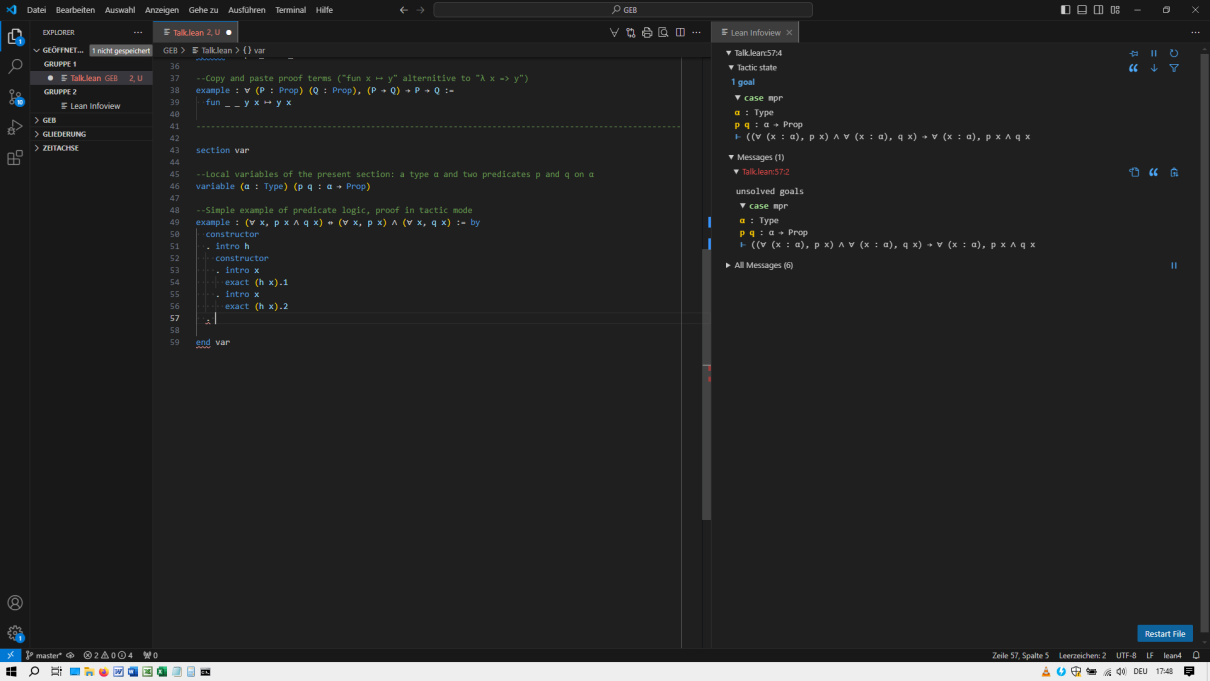

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
. sorry
. sorry
end var
Lean Infoview
Talk.lean:54:6
Tactic state
1 goal
case mp.left
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
x : α
⊢ p x
All Messages (5)
Restart File
master*
Zeile 54, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:47

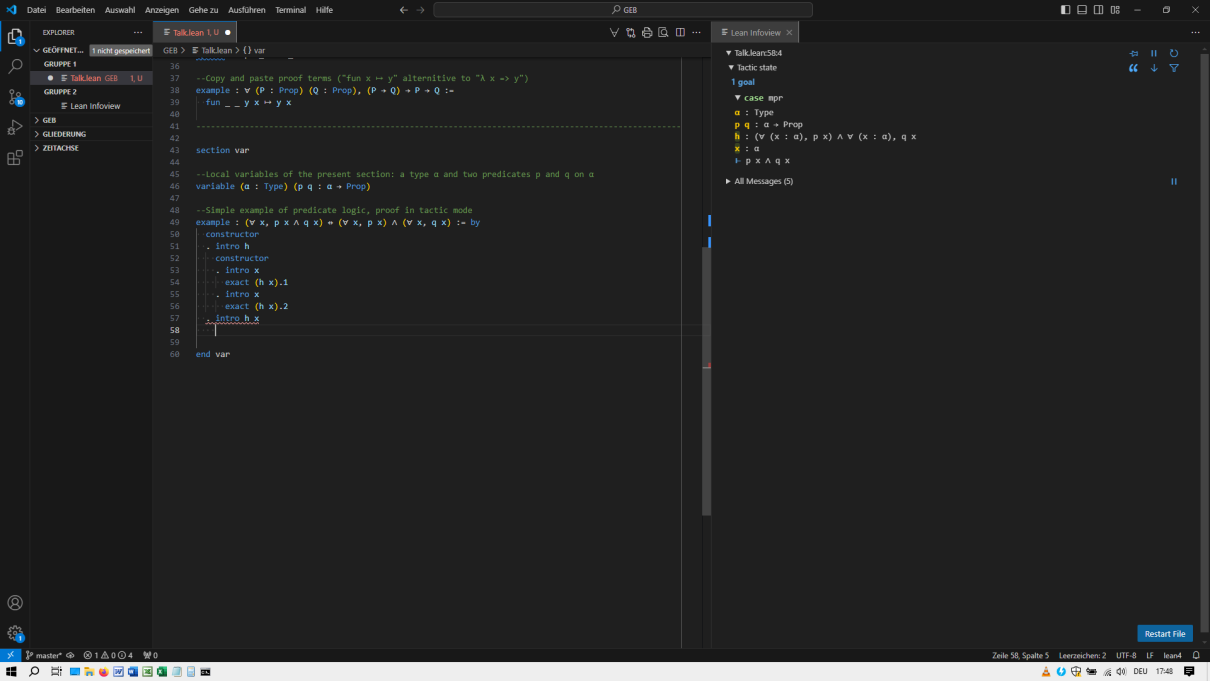

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52 constructor
53 . intro x
54 exact (h x).1
55 . sorry
56 . sorry
57
58 end var
Lean Infoview
Talk.lean:54:19
Tactic state
No goals
Expected type
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
x : α
⊢ ∀ {a b : Prop}, a ∧ b → a
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 54, Spalte 20 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:47

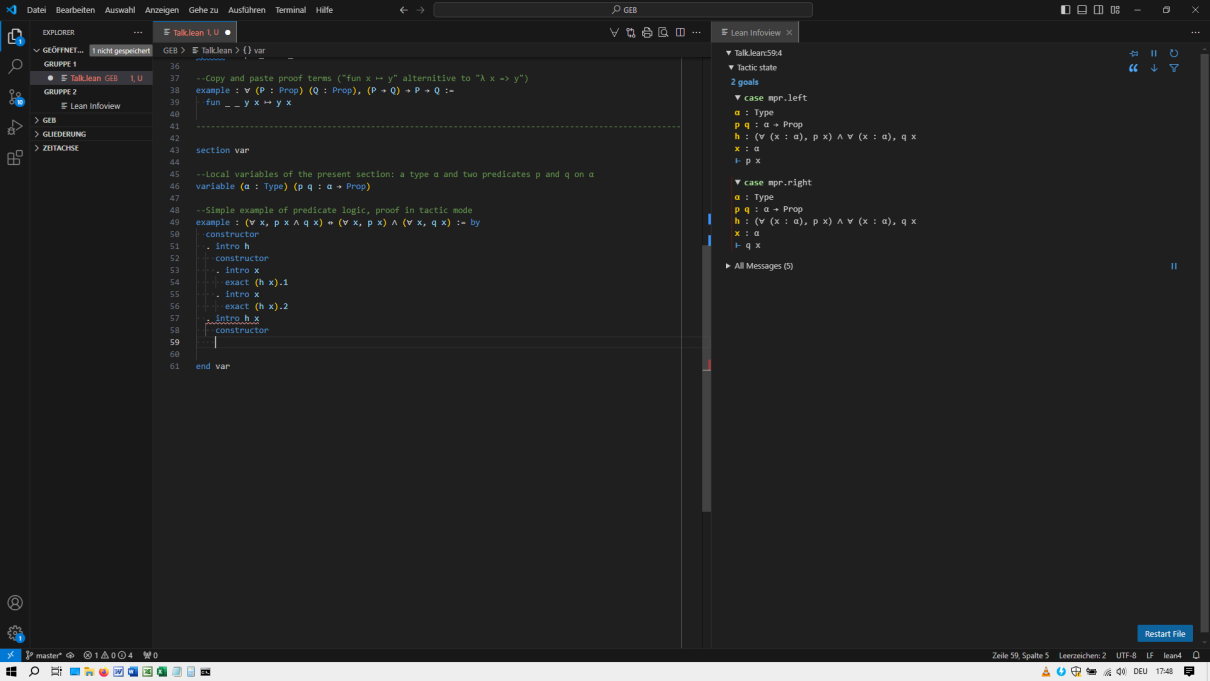

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 3, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 3, U
GEB > Talk.lean > {} var
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52 constructor
53 . intro x
54 exact (h x).1
55 .
56 sorry
57
58 end var
Lean Infoview
Talk.lean:55:6
Tactic state
1 goal
case mp.right
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), q x
Messages (2)
Talk.lean:55:4
unsolved goals
case mp.right
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
⊢ ∀ (x : α), q x
Talk.lean:49:56
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
All Messages (7)
Restart File
master*
3 0 4
0
Zeile 55, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:48

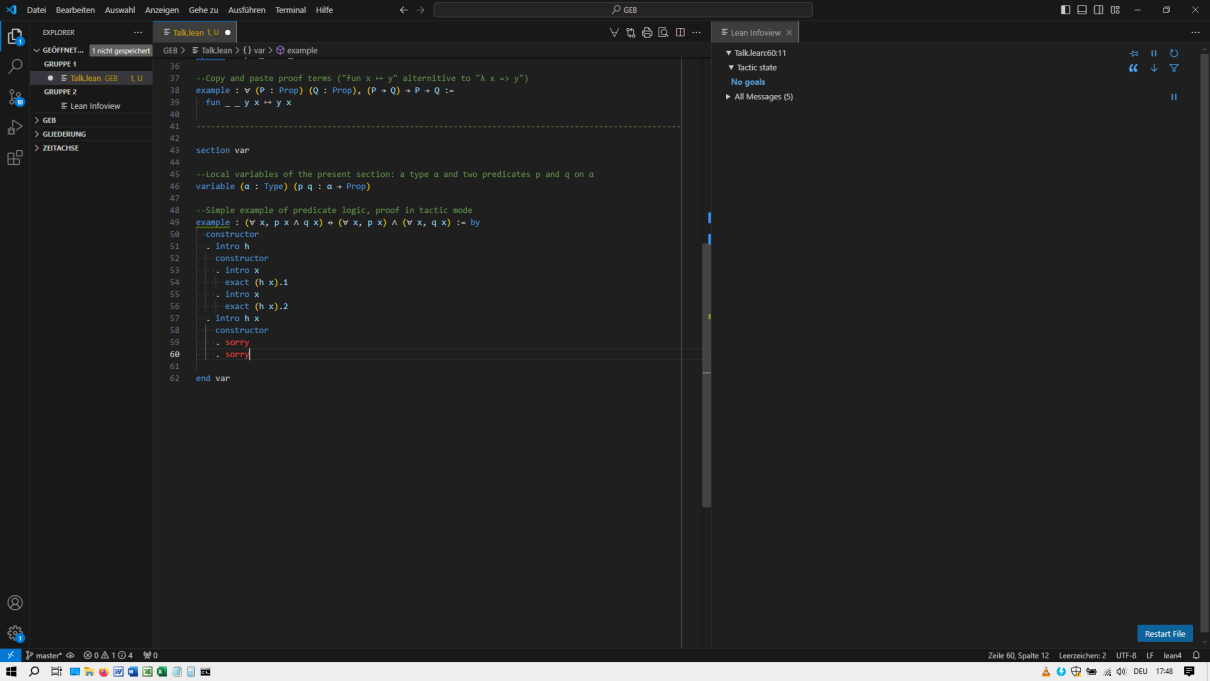

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52 constructor
53 . intro x
54 exact (h x).1
55 . intro x
56
57 . sorry
58
59 end var
Lean Infoview
Talk.lean:56:6
Tactic state
1 goal
case mp.right
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
x : α
⊢ q x
All Messages (5)
Restart File
master*
Zeile 56, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:48

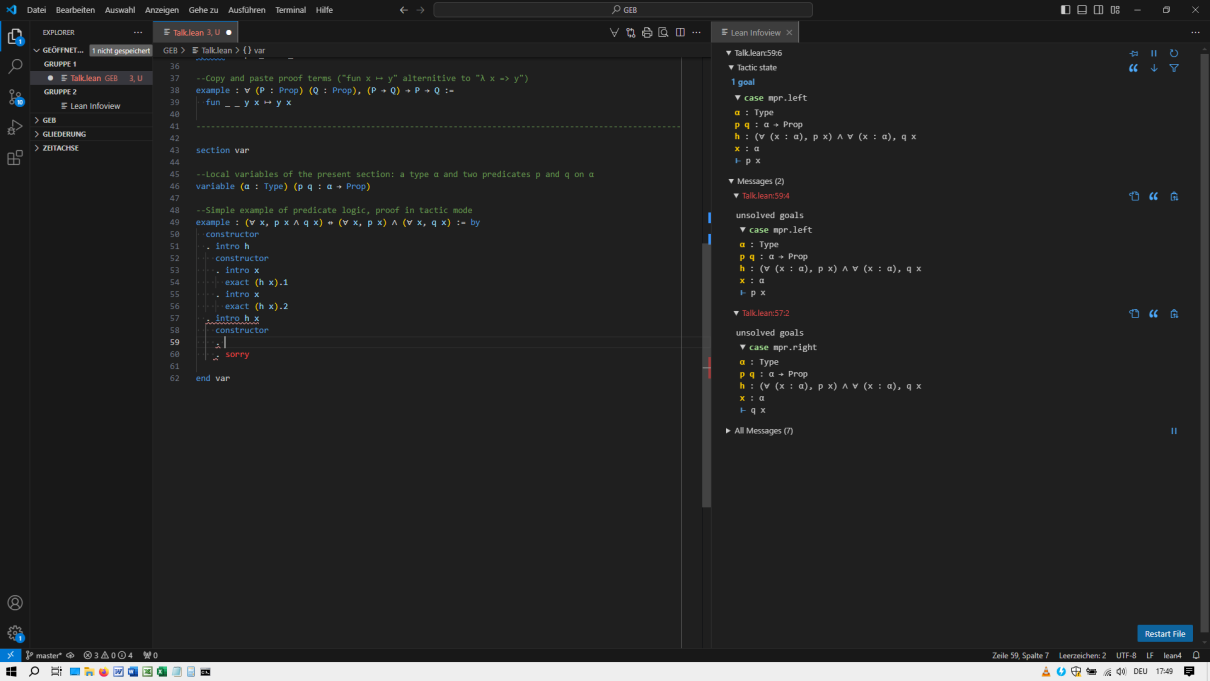

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52 constructor
53 . intro x
54 exact (h x).1
55 . intro x
56 exact (h x).2
57 . sorry
58
59 end var
Lean Infoview
Talk.lean:56:19
Tactic state
No goals
Expected type
α : Type
p q : α → Prop
h : ∀ (x : α), p x ∧ q x
x : α
⊢ ∀ {a b : Prop}, a ∧ b → b
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 56, Spalte 20 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:48

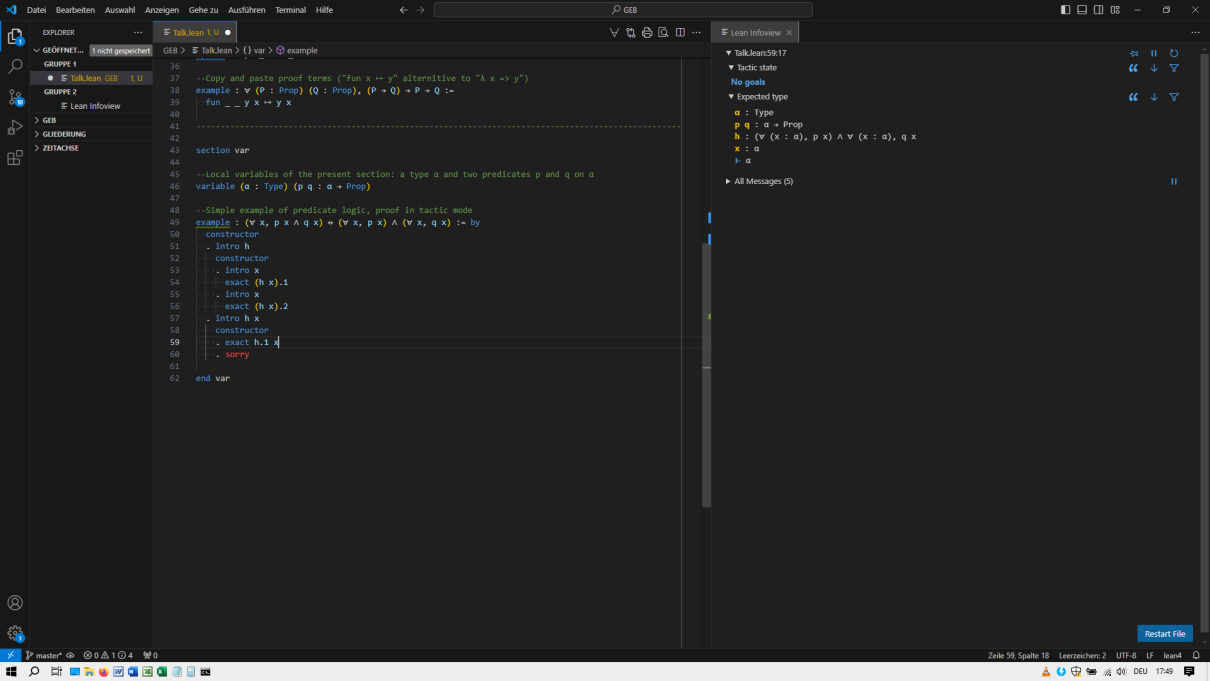

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
.
end var
Lean Infoview
Talk.lean:57:4
Tactic state
1 goal
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
Messages (1)
Talk.lean:57:2
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ ((∀ (x : α), p x) ∧ ∀ (x : α), q x) → ∀ (x : α), p x ∧ q x
All Messages (6)
Restart File
master*
Zeile 57, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:48

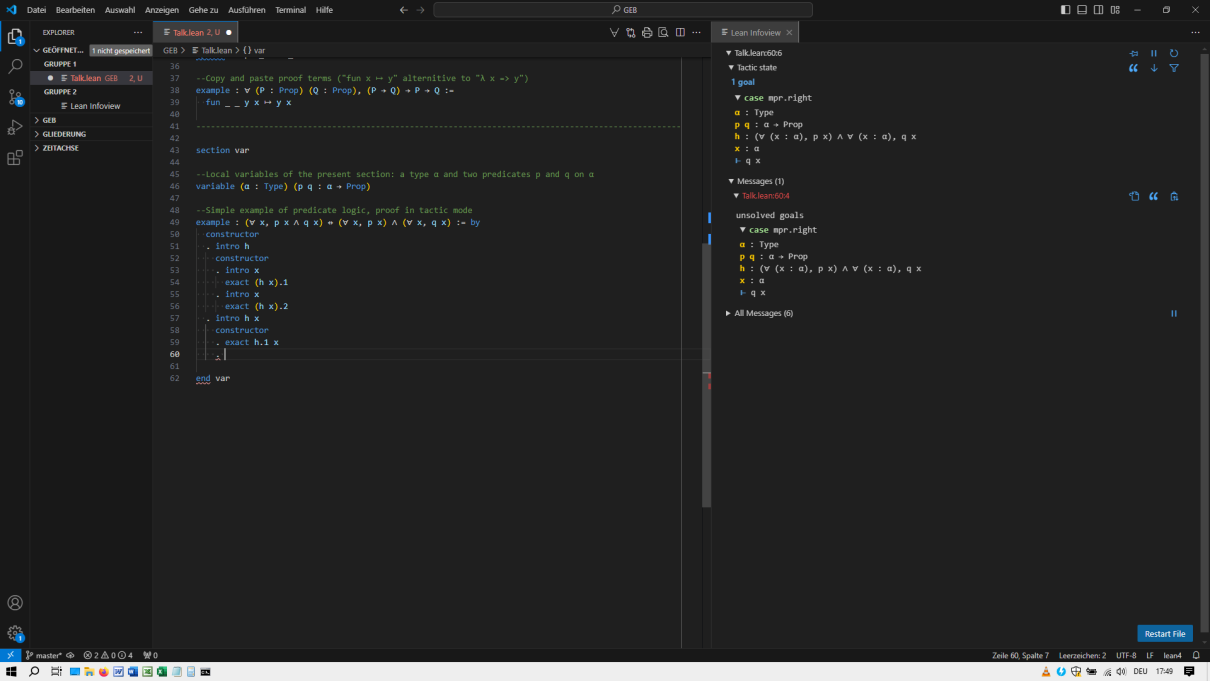

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
end var
Lean Infoview
Talk.lean:58:4
Tactic state
1 goal
case mpr
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ p x ∧ q x
All Messages (5)
Restart File
master*
Zeile 58, Spalte 5 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:48

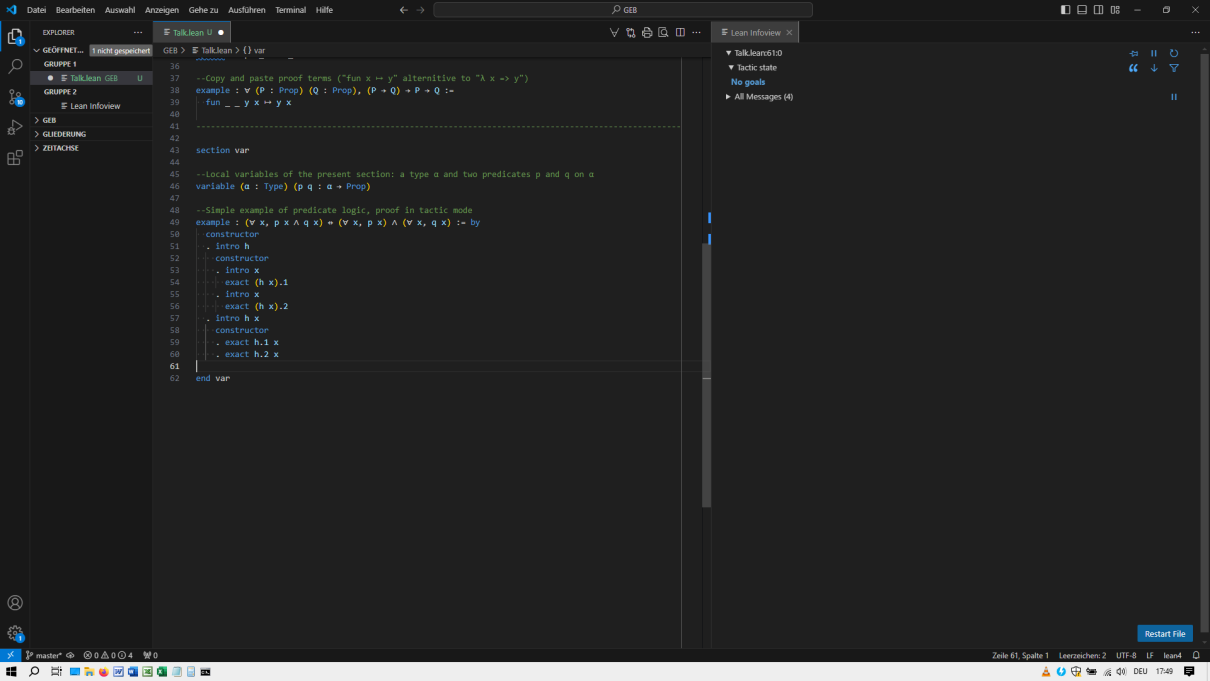

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
end var
Lean Infoview
Talk.lean:59:4
Tactic state
2 goals
case mpr.left
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ p x
case mpr.right
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ q x
All Messages (5)
Restart File
master*
Zeile 59, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:48

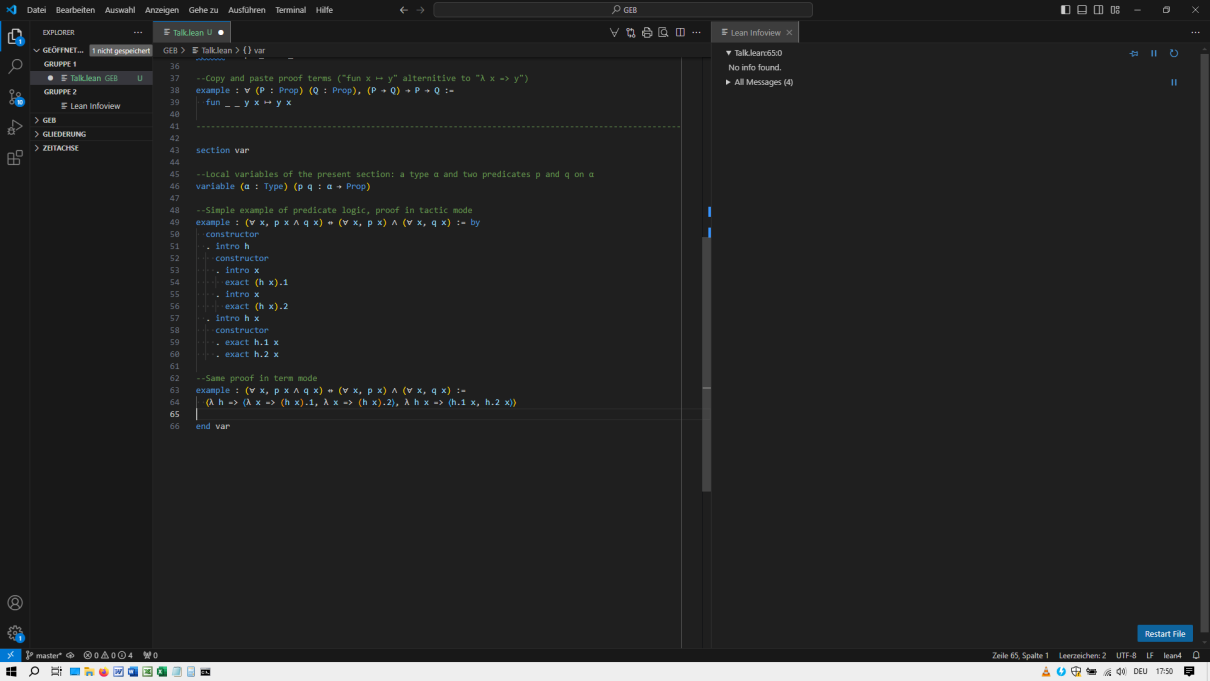

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. sorry
. sorry
end var
Lean Infoview
Talk.lean:60:11
Tactic state
No goals
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 60, Spalte 12
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:48

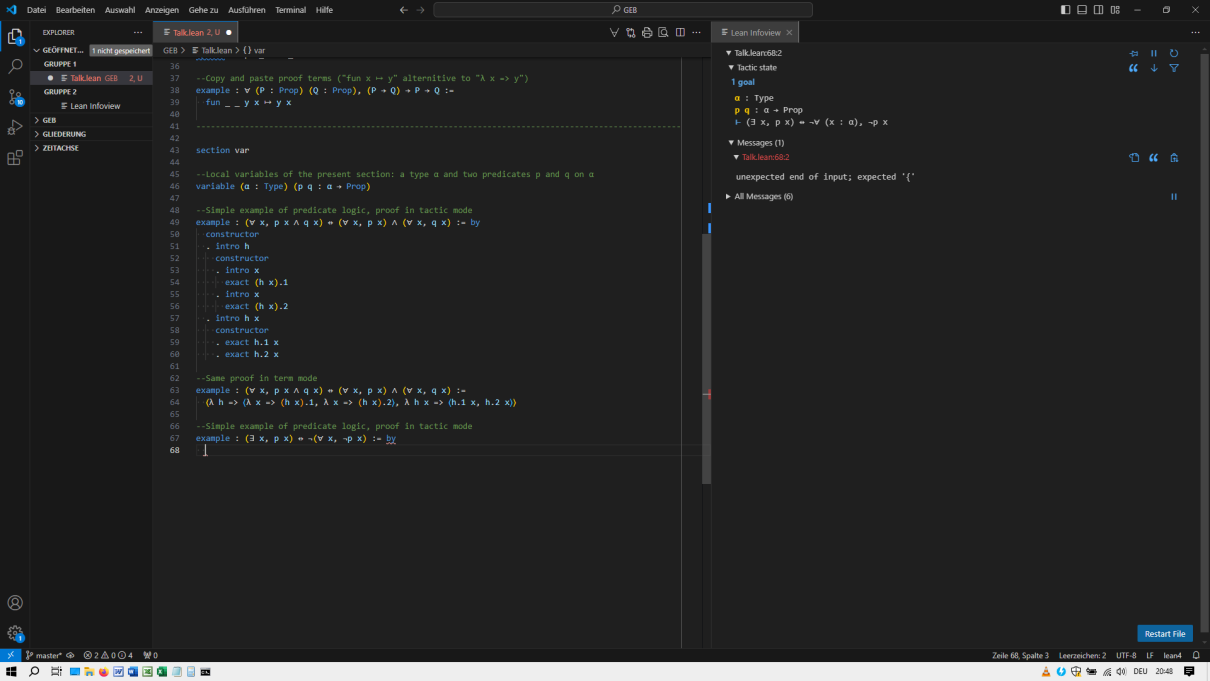

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 3, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 3, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
.
. sorry
end var
Lean Infoview
Talk.lean:59:6
Tactic state
1 goal
case mpr.left
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ p x
Messages (2)
Talk.lean:59:4
unsolved goals
case mpr.left
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ p x
Talk.lean:57:2
unsolved goals
case mpr.right
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ q x
All Messages (7)
Restart File
master*
3 0 4
0
Zeile 59, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:49

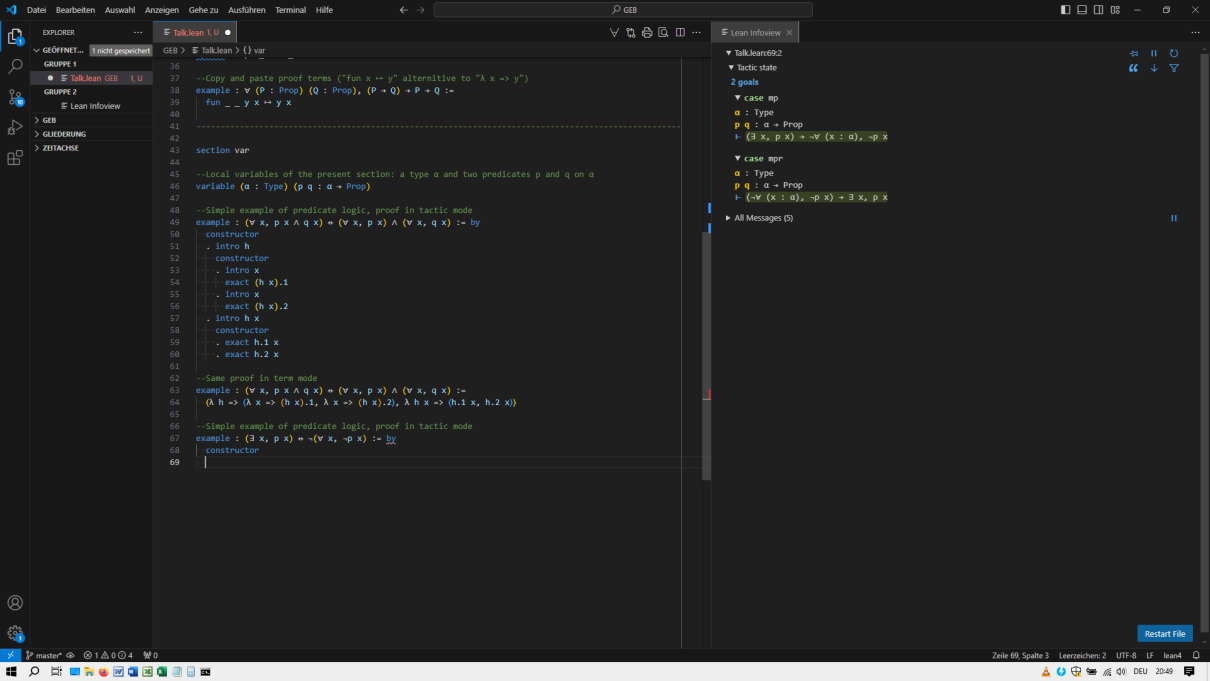

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
36
37 --Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
38 example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
39 fun _ _ y x ↦ y x
40
41
42
43 section var
44
45 --Local variables of the present section: a type α and two predicates p and q on α
46 variable (α : Type) (p q : α → Prop)
47
48 --Simple example of predicate logic, proof in tactic mode
49 example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
50 constructor
51 . intro h
52 constructor
53 . intro x
54 exact (h x).1
55 . intro x
56 exact (h x).2
57 . intro h x
58 constructor
59 . exact h.1 x
60 . sorry
61
62 end var
Lean Infoview
Talk.lean:59:17
Tactic state
No goals
Expected type
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ α
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 59, Spalte 18 Leerzeichen: 2 UTF-8 LF lean4
DEU 17:49

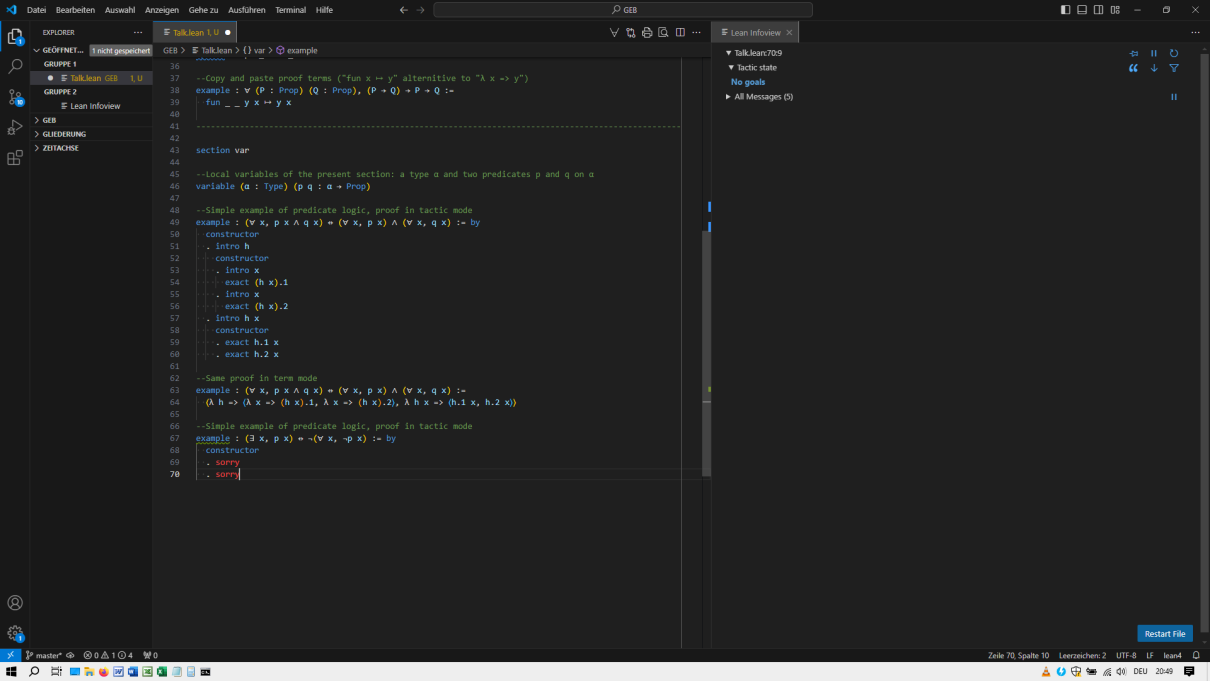

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
.
end var
Lean Infoview
Talk.lean:60:6
Tactic state
1 goal
case mpr.right
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ q x
Messages (1)
Talk.lean:60:4
unsolved goals
case mpr.right
α : Type
p q : α → Prop
h : (∀ (x : α), p x) ∧ ∀ (x : α), q x
x : α
⊢ q x
All Messages (6)
Restart File
master*
Zeile 60, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:49

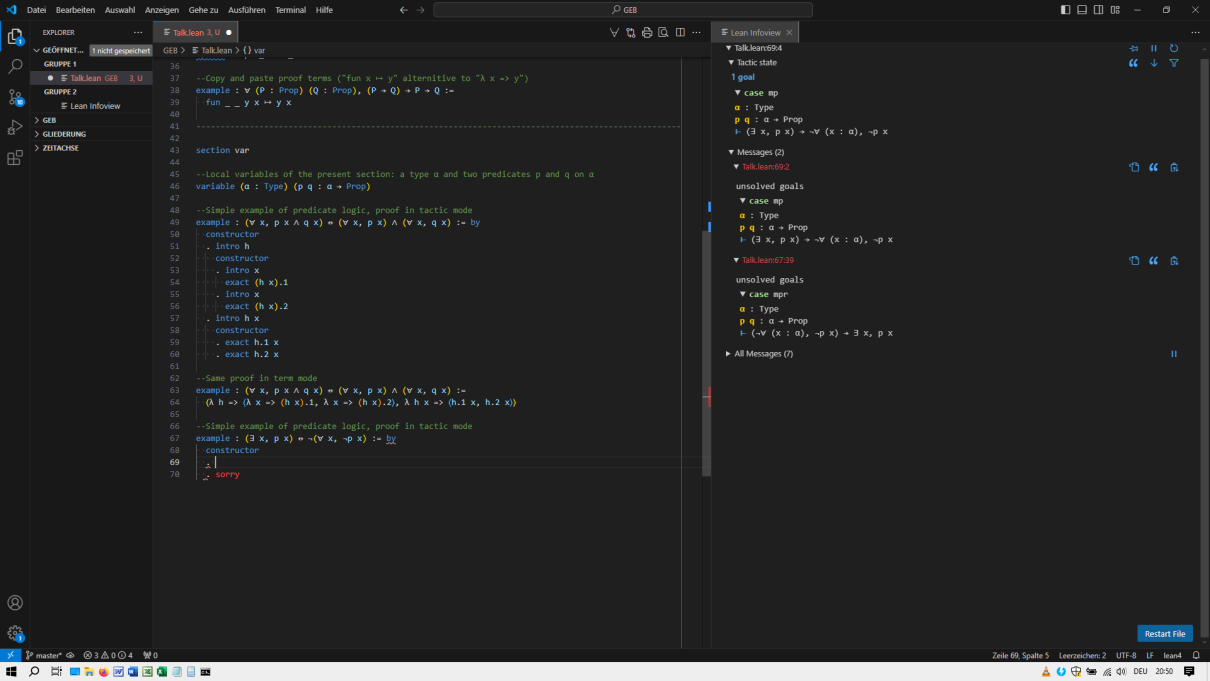

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
end var
Lean Infoview
Talk.lean:61:0
Tactic state
No goals
All Messages (4)
Restart File
master*
Zeile 61, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:49

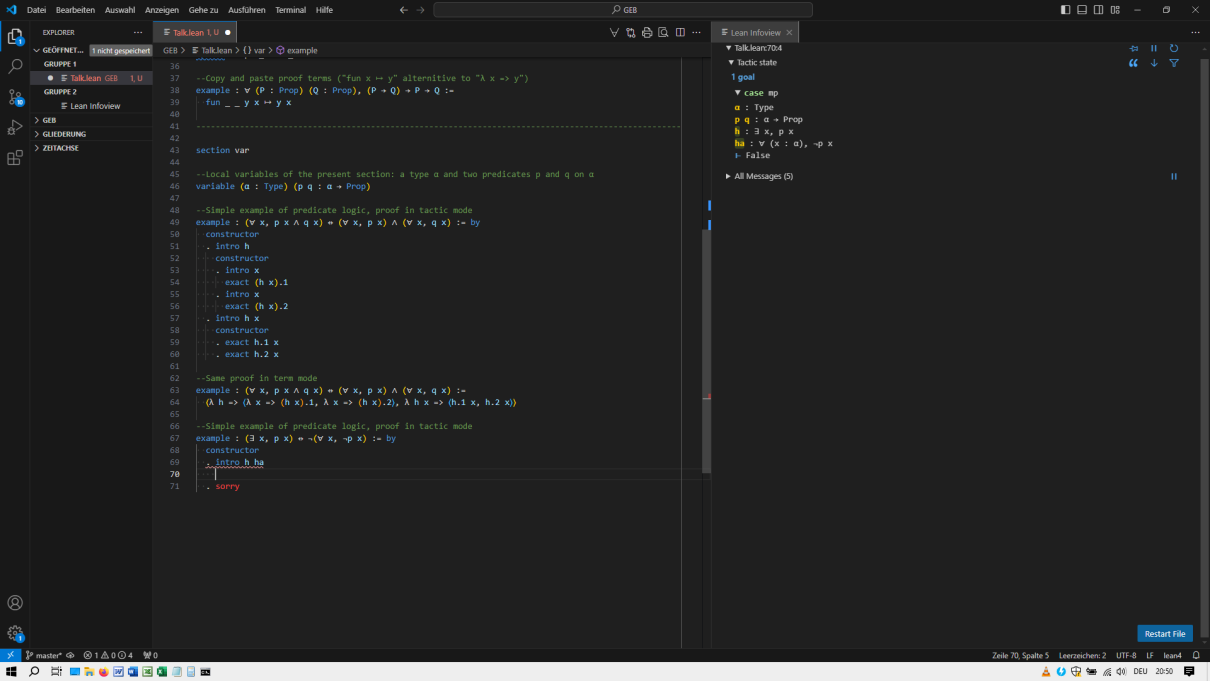

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
end var
Lean Infoview
Talk.lean:65:0
No info found.
All Messages (4)
Restart File
master*
Zeile 65, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
17:50

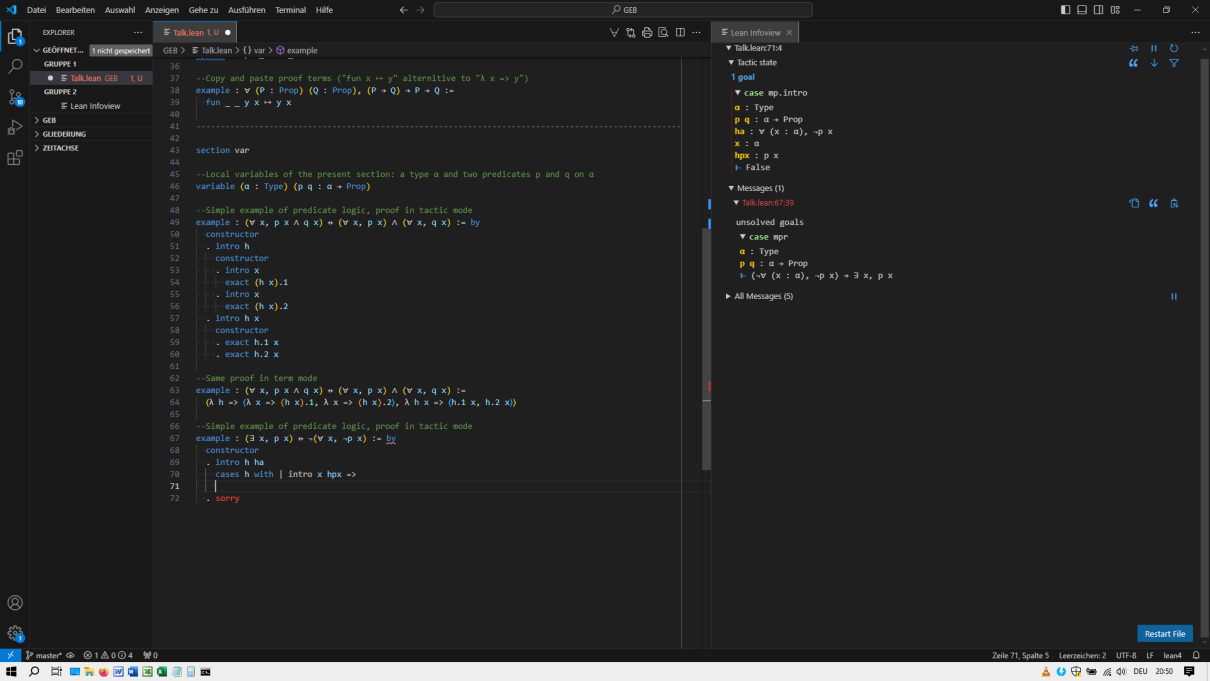

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
(λ h => (λ x => (h x).1, λ x => (h x).2), λ h x => (h.1 x, h.2 x))
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
Lean Infoview
Talk.lean:68:2
Tactic state
1 goal
α : Type
p q : α → Prop
⊢ (∃ x, p x) ↔ ¬∀ (x : α), ¬p x
Messages (1)
Talk.lean:68:2
unexpected end of input; expected '{'
All Messages (6)
Restart File
master*
Zeile 68, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:48

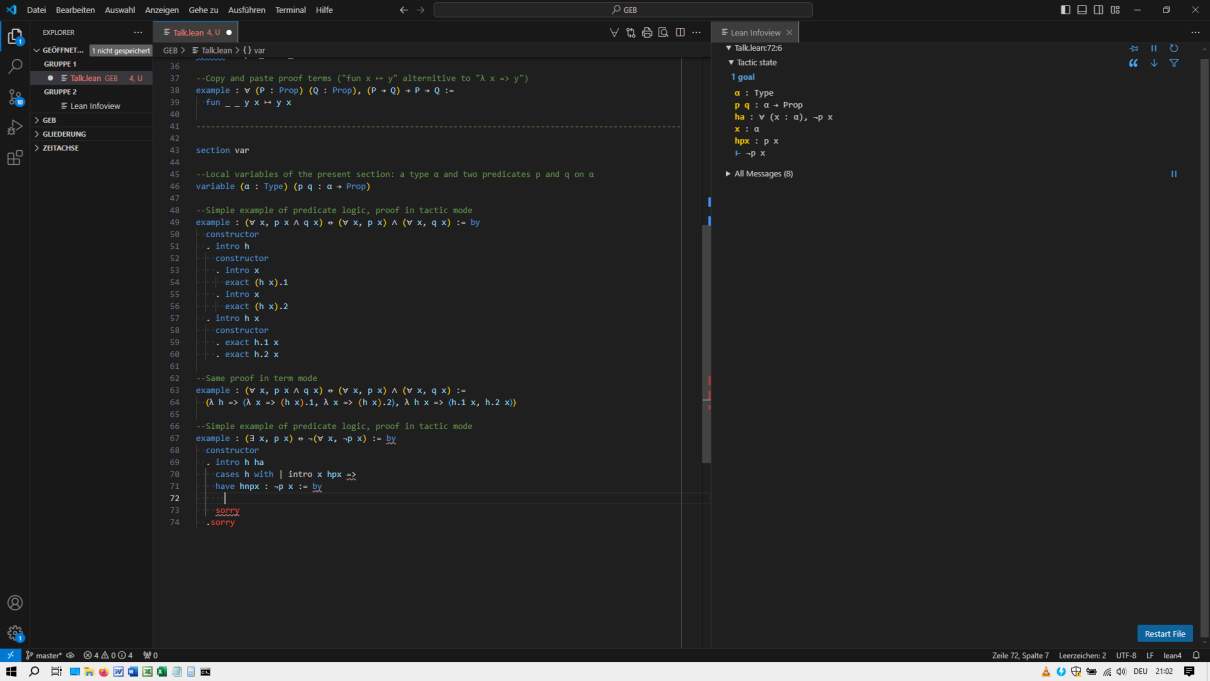

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
(λ h => (λ x => (h x).1, λ x => (h x).2), λ h x => (h.1 x, h.2 x))
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
Lean Infoview
Talk.lean:69:2
Tactic state
2 goals
case mp
α : Type
p q : α → Prop
⊢ (∃ x, p x) → ¬∀ (x : α), ¬p x
case mpr
α : Type
p q : α → Prop
⊢ (¬∀ (x : α), ¬p x) → ∃ x, p x
All Messages (5)
Restart File
master*
Zeile 69, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:49

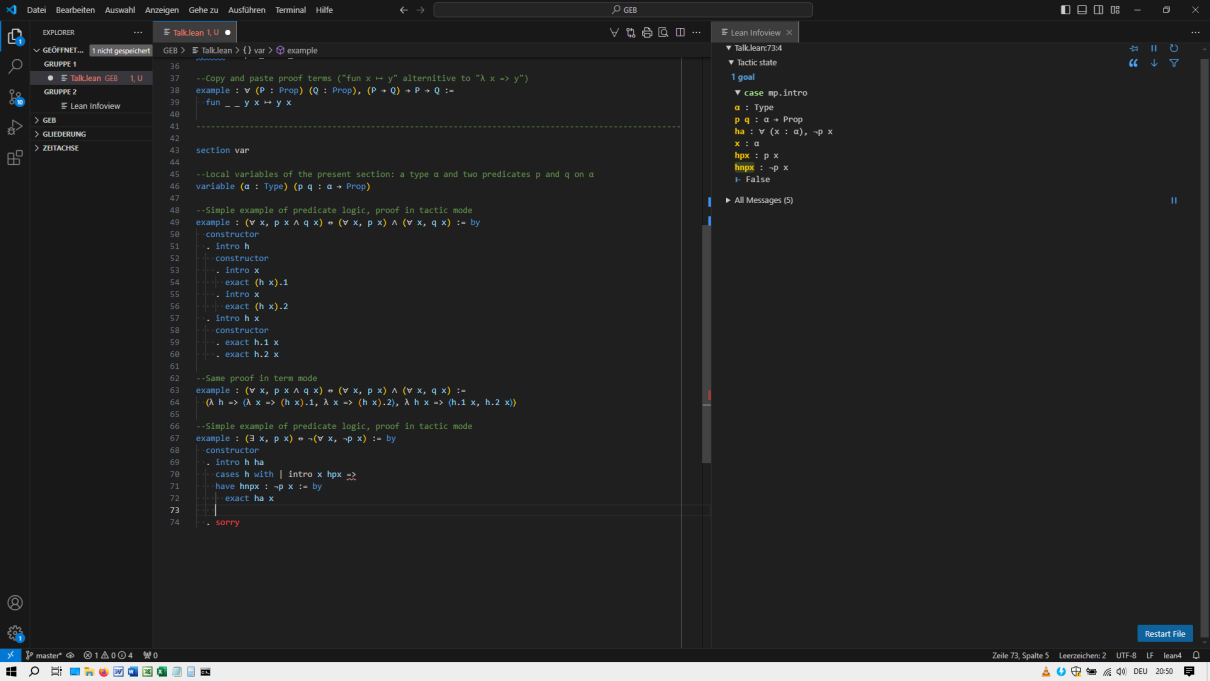

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
(λ h => (λ x => (h x).1, λ x => (h x).2), λ h x => (h.1 x, h.2 x))
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. sorry
. sorry
Lean Infoview
Talk.lean:70:9
Tactic state
No goals
All Messages (5)
Restart File
master*
0 1 4
0
Zeile 70, Spalte 10 Leerzeichen: 2 UTF-8 LF lean4
DEU 20:49

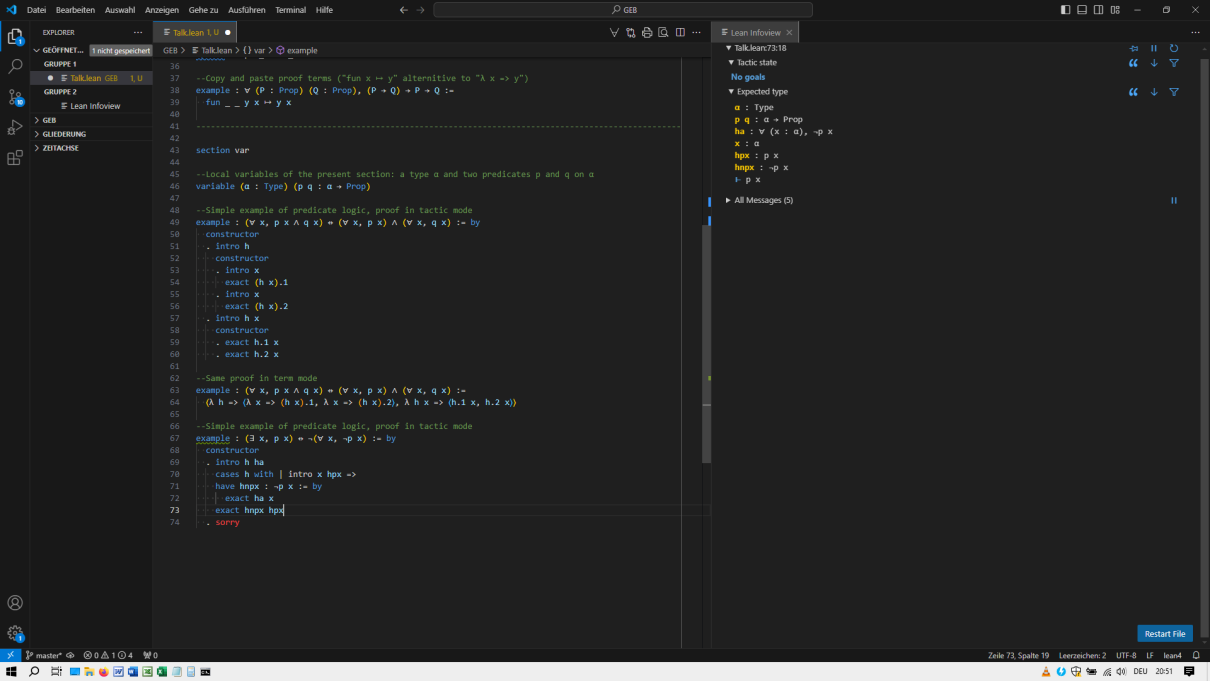

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 3, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 3, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
·
sorry
Lean Infoview
Talk.lean:69:4
Tactic state
1 goal
case mp
α : Type
p q : α → Prop
⊢ (∃ x, p x) → ¬∀ (x : α), ¬p x
Messages (2)
Talk.lean:69:2
unsolved goals
case mp
α : Type
p q : α → Prop
⊢ (∃ x, p x) → ¬∀ (x : α), ¬p x
Talk.lean:67:39
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ (¬∀ (x : α), ¬p x) → ∃ x, p x
All Messages (7)
Restart File
master*
3 0 4
0
Zeile 69, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:50

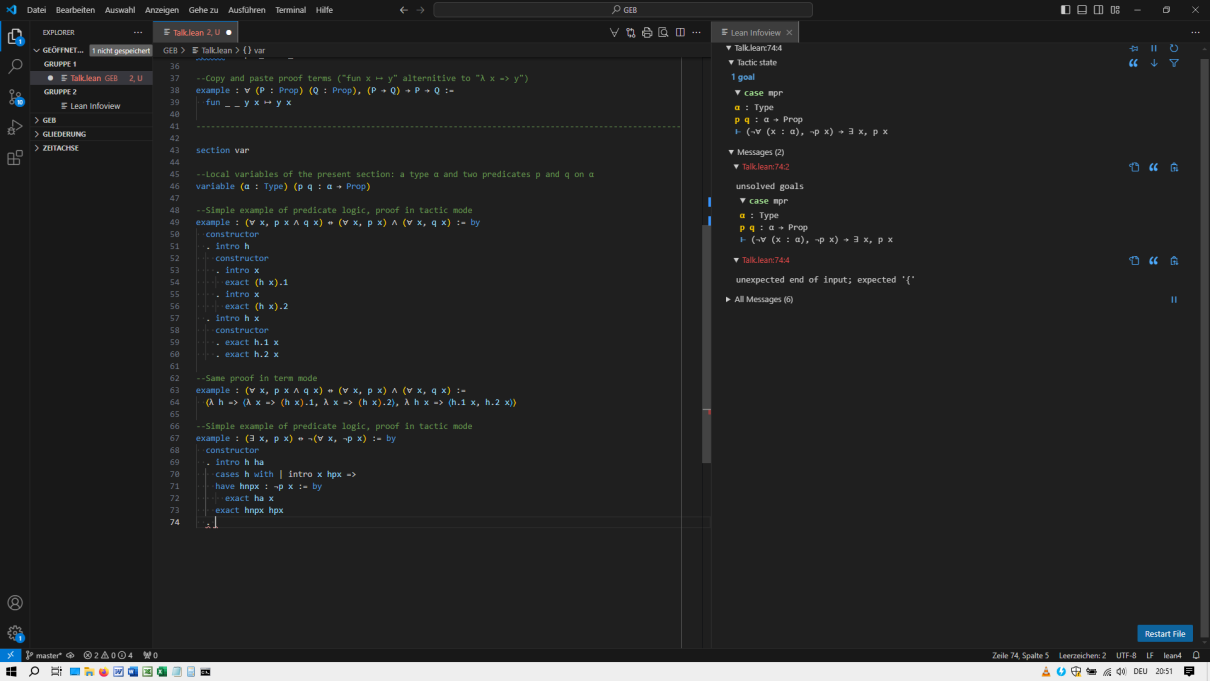

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
. sorry
Lean Infoview
Talk.lean:70:4
Tactic state
1 goal
case mp
α : Type
p q : α → Prop
h : ∃ x, p x
ha : ∀ (x : α), ¬p x
⊢ False
All Messages (5)
Restart File
master*
Zeile 70, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:50

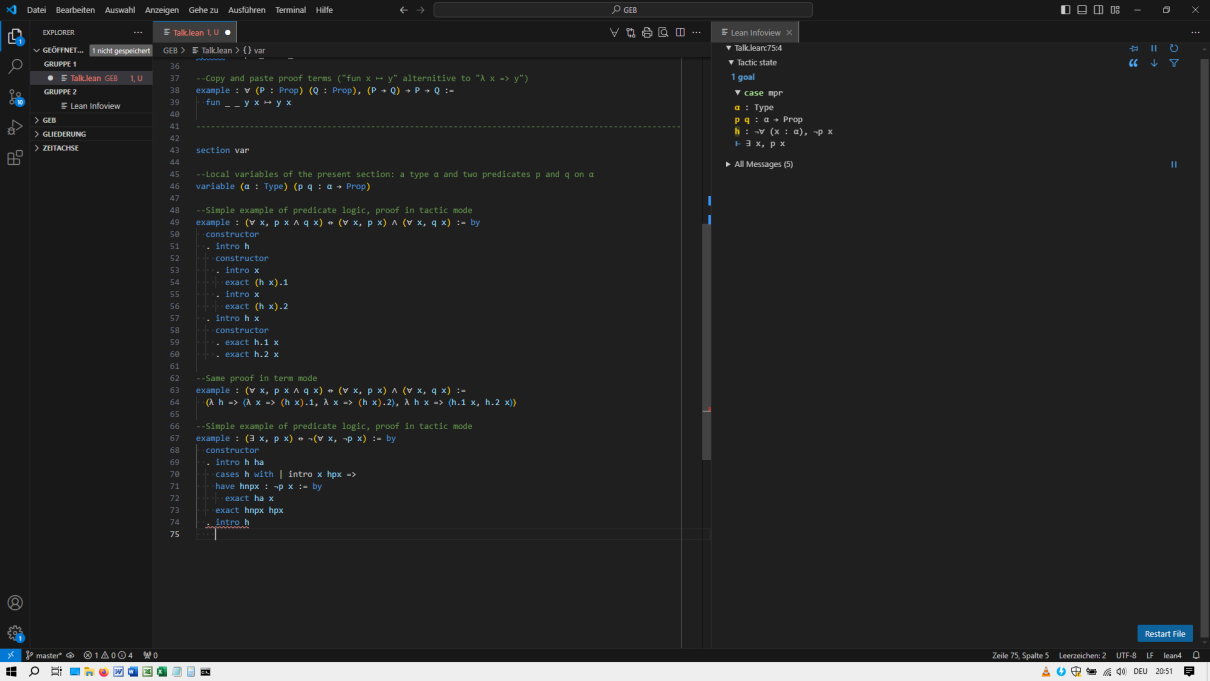

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
. sorry
Lean Infoview
Talk.lean:71:4
Tactic state
1 goal
case mp.intro
α : Type
p q : α → Prop
ha : ∀ (x : α), ¬p x
x : α
hpx : p x
⊢ False
Messages (1)
Talk.lean:67:39
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ (¬∀ (x : α), ¬p x) → ∃ x, p x
All Messages (5)
Restart File
master*
Zeile 71, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:50

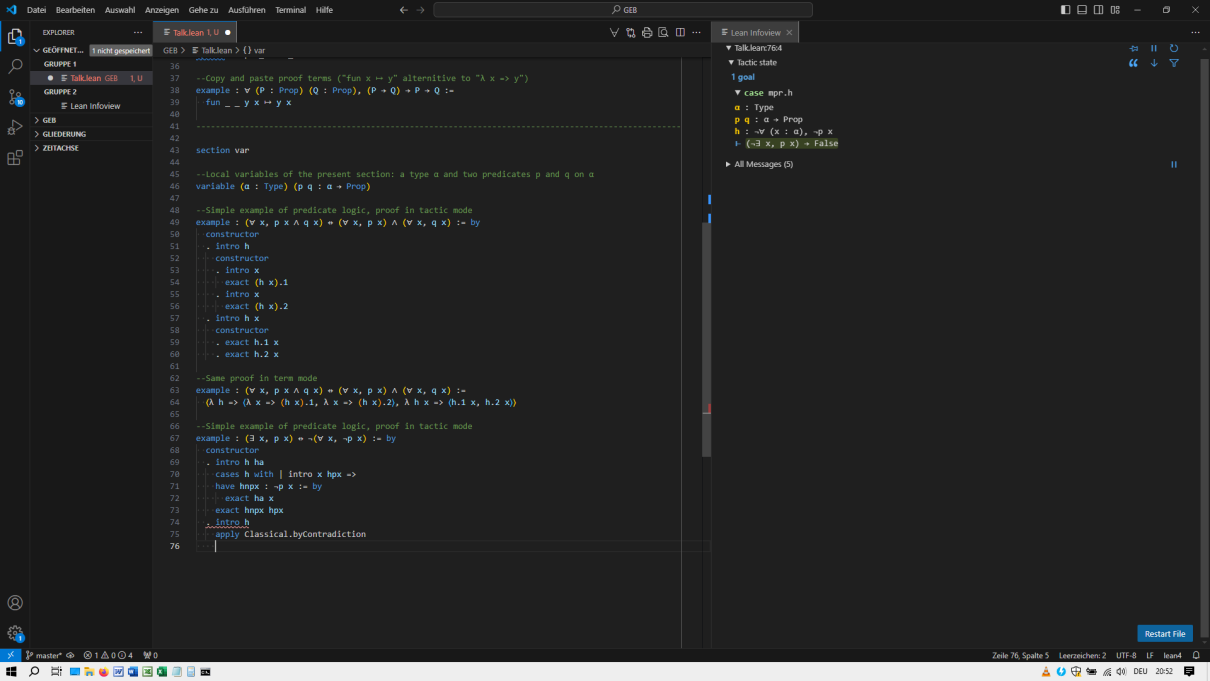

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 4, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 4, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
sorry
.sorry
Lean Infoview
Talk.lean:72:6
Tactic state
1 goal
α : Type
p q : α → Prop
ha : ∀ (x : α), ¬p x
x : α
hpx : p x
⊢ ¬p x
All Messages (8)
Restart File
master*
Zeile 72, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:02

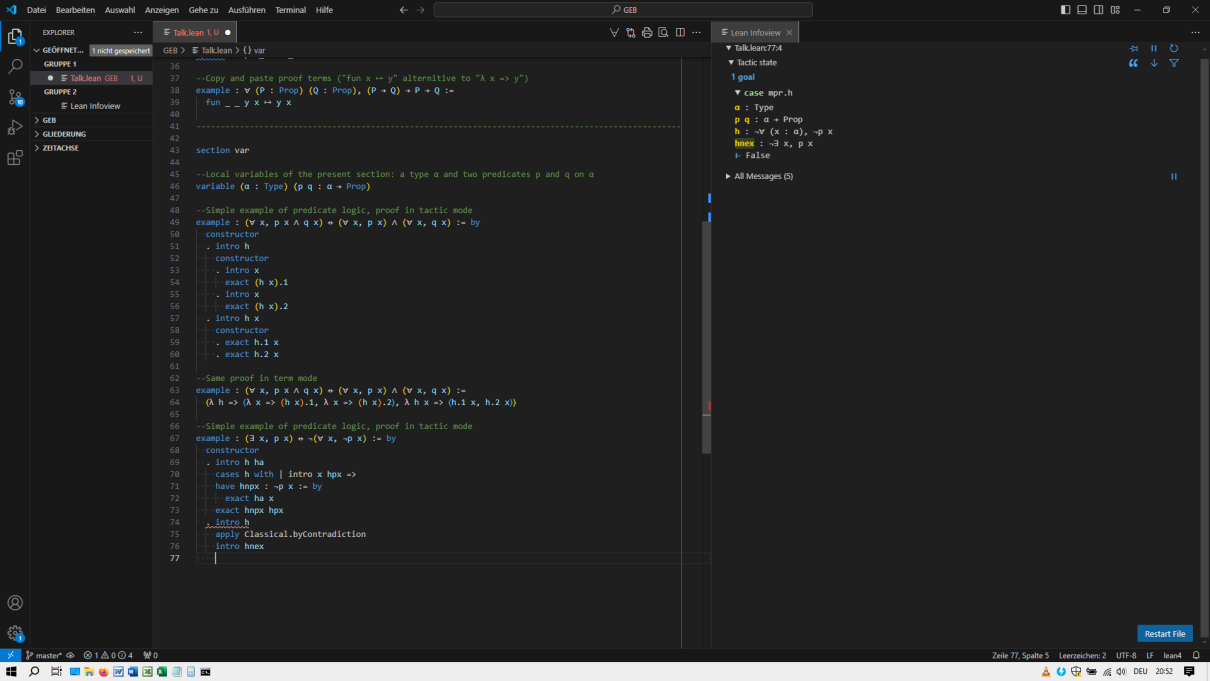

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
(λ h => (λ x => (h x).1, λ x => (h x).2), λ h x => (h.1 x, h.2 x))
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
. sorry
Lean Infoview
Talk.lean:73:4
Tactic state
1 goal
case mp.intro
α : Type
p q : α → Prop
ha : ∀ (x : α), ¬p x
x : α
hpx : p x
hnpx : ¬p x
⊢ False
All Messages (5)
Restart File
master*
Zeile 73, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:50

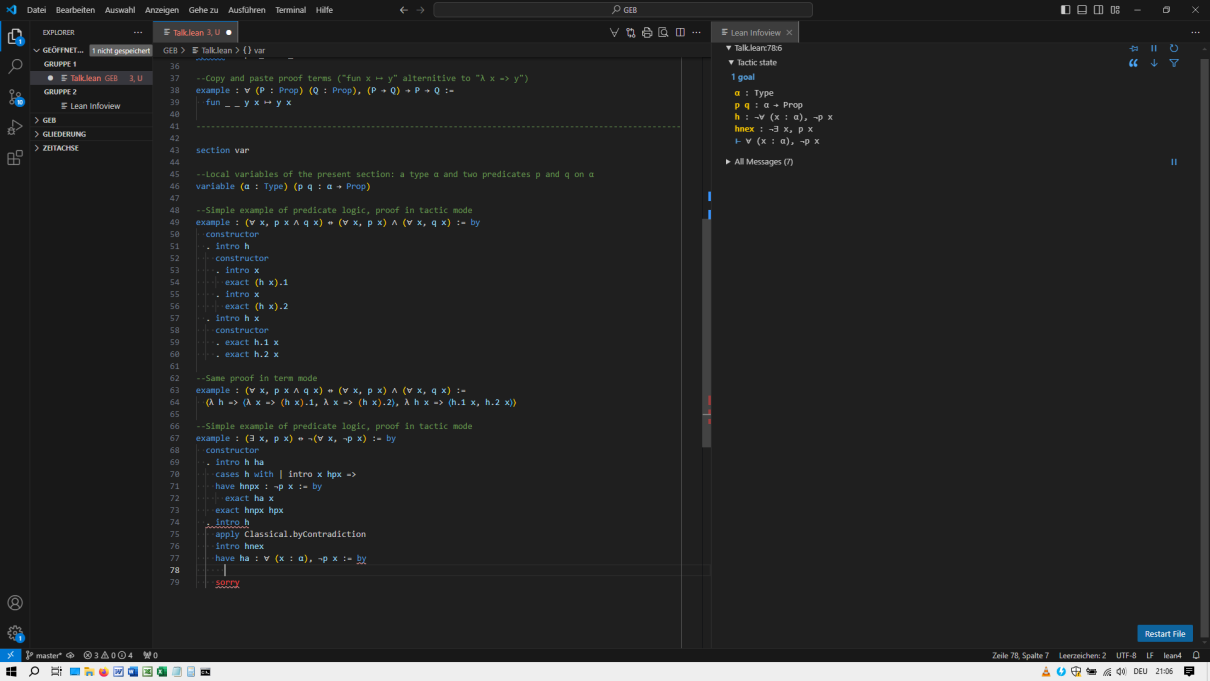

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. sorry
Lean Infoview
Talk.lean:73:18
Tactic state
No goals
Expected type
α : Type
p q : α → Prop
ha : ∀ (x : α), ¬p x
x : α
hpx : p x
hnpx : ¬p x
⊢ p x
All Messages (5)
Restart File
master*
Zeile 73, Spalte 19 Leerzeichen: 2 UTF-8 LF lean4
DEU 20:51

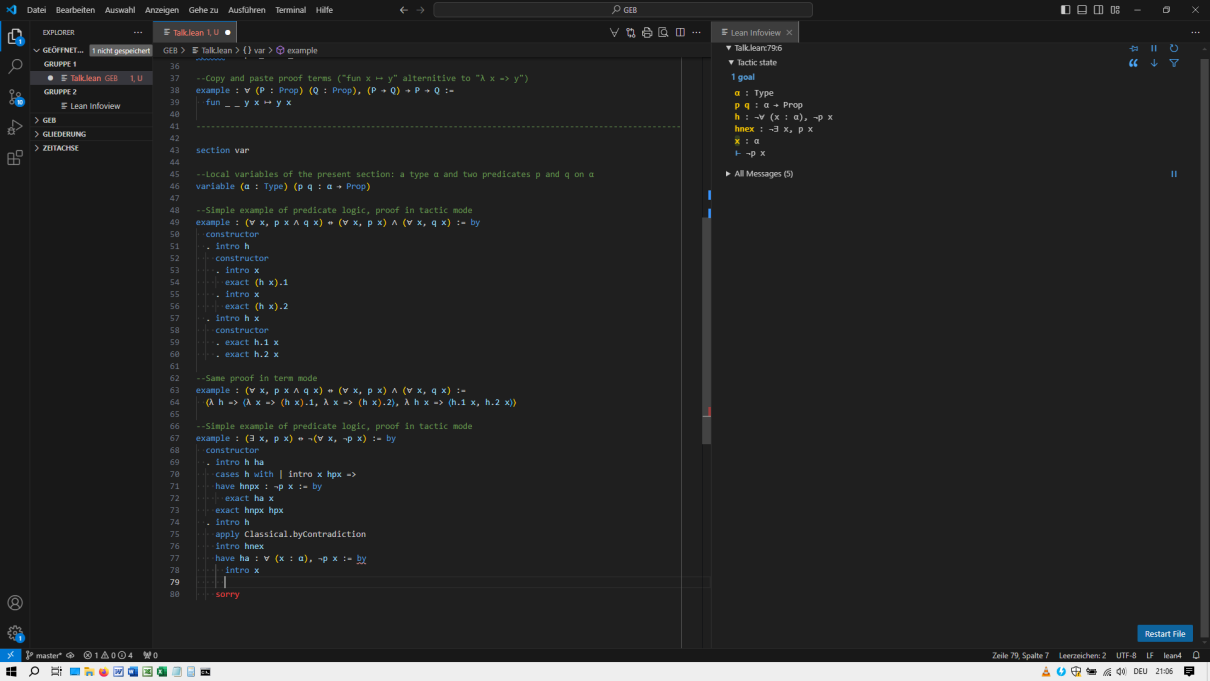

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
Lean Infoview
Talk.lean:74:4
Tactic state
1 goal
case mpr
α : Type
p q : α → Prop
⊢ (¬∀ (x : α), ¬p x) → ∃ x, p x
Messages (2)
Talk.lean:74:2
unsolved goals
case mpr
α : Type
p q : α → Prop
⊢ (¬∀ (x : α), ¬p x) → ∃ x, p x
Talk.lean:74:4
unexpected end of input; expected '{'
All Messages (6)
Restart File
master*
Zeile 74, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:51

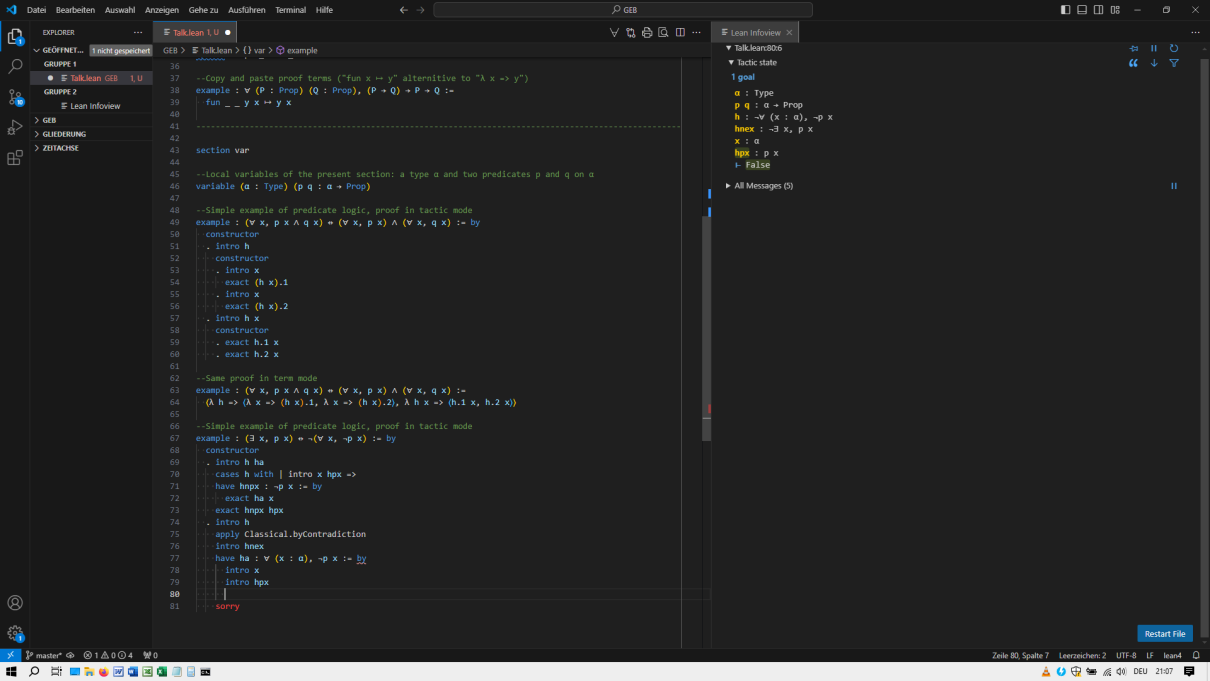

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
Lean Infoview
Talk.lean:75:4
Tactic state
1 goal
case mpr
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
⊢ ∃ x, p x
All Messages (5)
Restart File
master*
Zeile 75, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:51

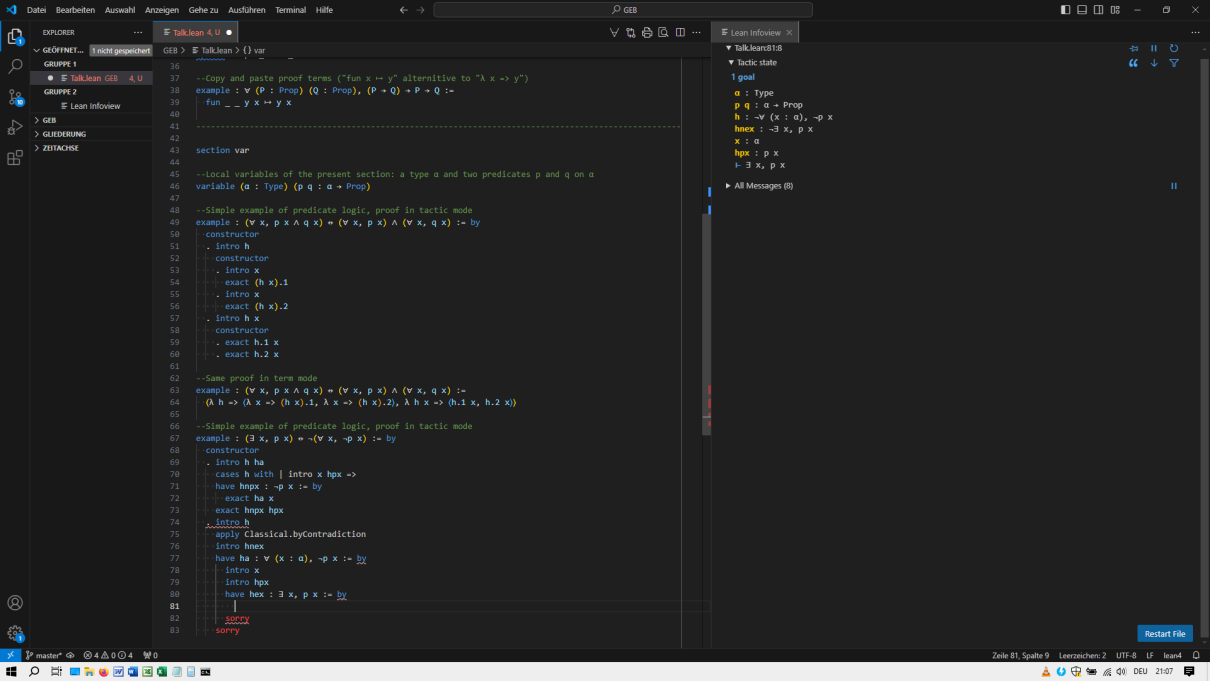

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
Lean Infoview
Talk.lean:76:4
Tactic state
1 goal
case mpr.h
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
⊢ (¬∃ x, p x) → False
All Messages (5)
Restart File
master*
Zeile 76, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:52

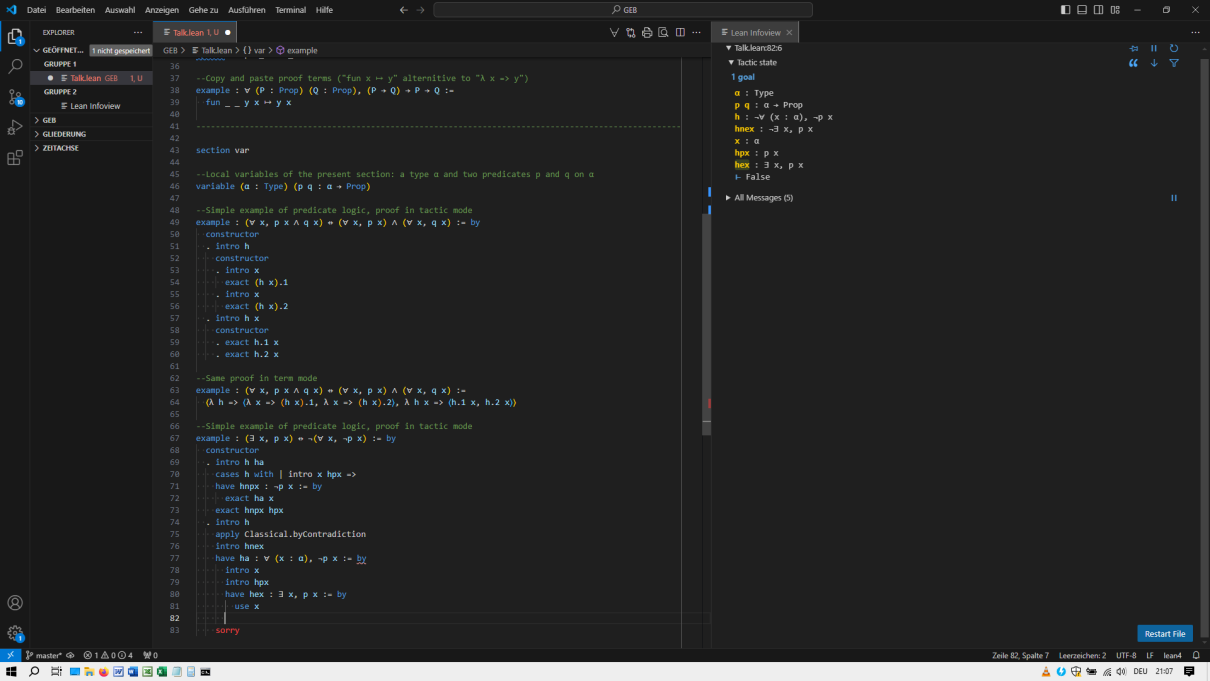

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
Lean Infoview
Talk.lean:77:4
Tactic state
1 goal
case mpr.h
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
⊢ False
All Messages (5)
Restart File
master*
Zeile 77, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
20:52


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 3, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 3, U
GEB > Talk.lean > {} var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
sorry
Lean Infoview
Talk.lean:7:6
Tactic state
1 goal
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
⊢ ∀ (x : α), ¬p x
All Messages (7)
Restart File
master*
Zeile 78, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:06

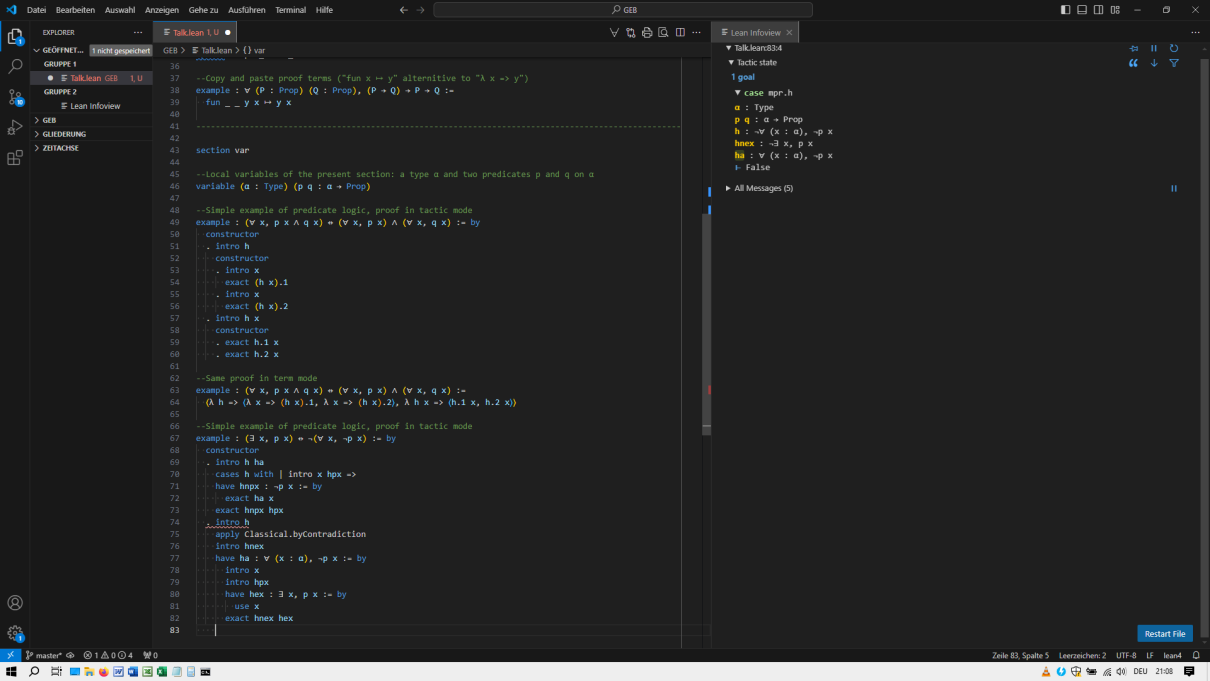

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
intro x
sorry
Lean Infoview
Talk.lean:79:6
Tactic state
1 goal
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
x : α
⊢ ¬p x
All Messages (5)
Restart File
master*
Zeile 79, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:06

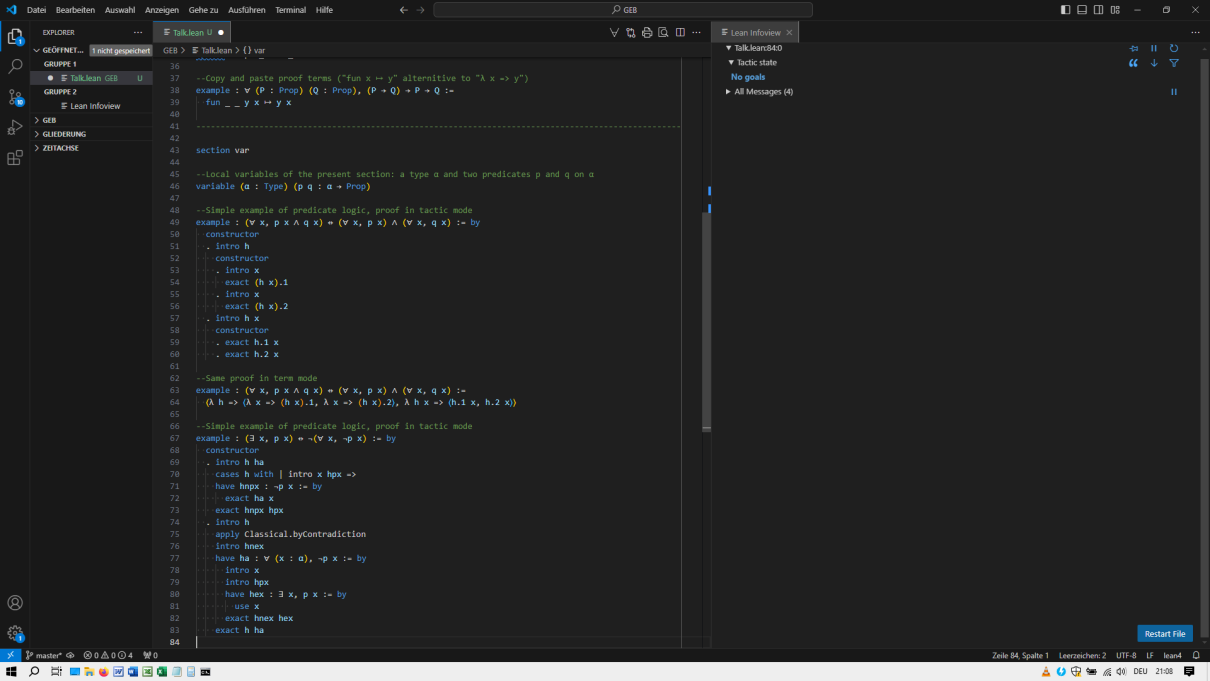

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var > example
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
intro x
intro hpx
sorry
Lean Infoview
Talk.lean:80:6
Tactic state
1 goal
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
x : α
hpx : p x
⊢ False
All Messages (5)
Restart File
master*
Zeile 80, Spalte 7
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:07

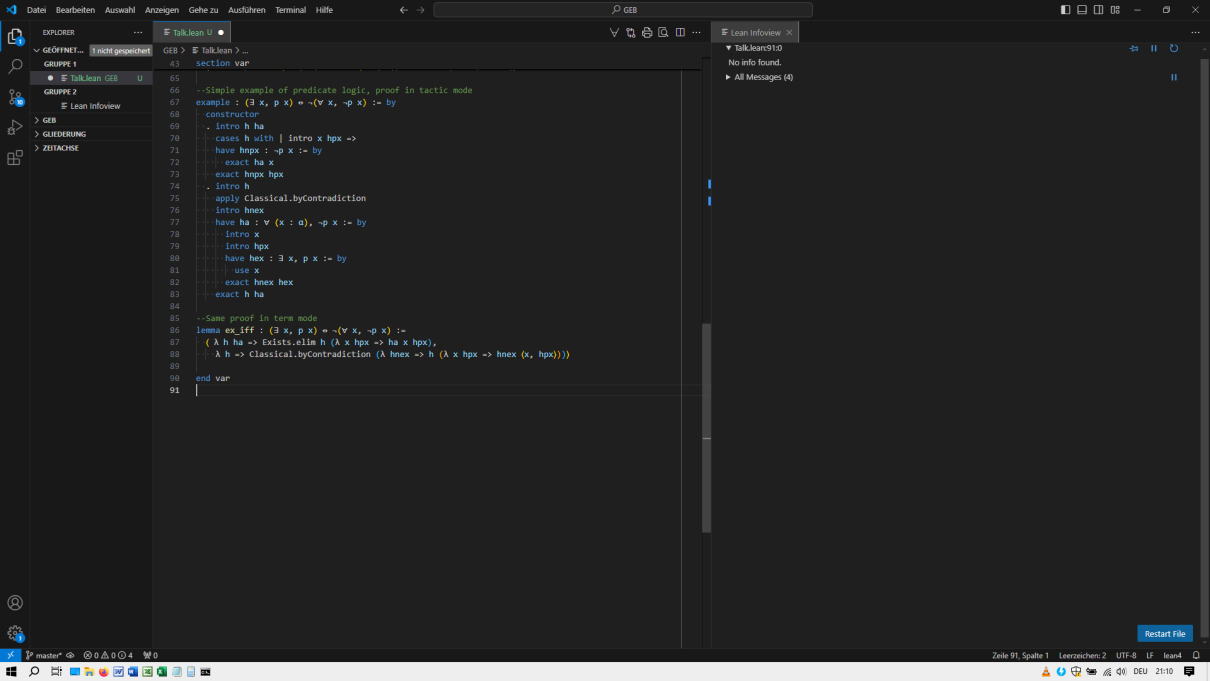

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 4, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 4, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
intro x
intro hpx
have hex : ∃ x, p x := by
sorry
sorry
Lean Infoview
Talk.lean:81:8
Tactic state
1 goal
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
x : α
hpx : p x
⊢ ∃ x, p x
All Messages (8)
Restart File
master*
Zeile 81, Spalte 9
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:07

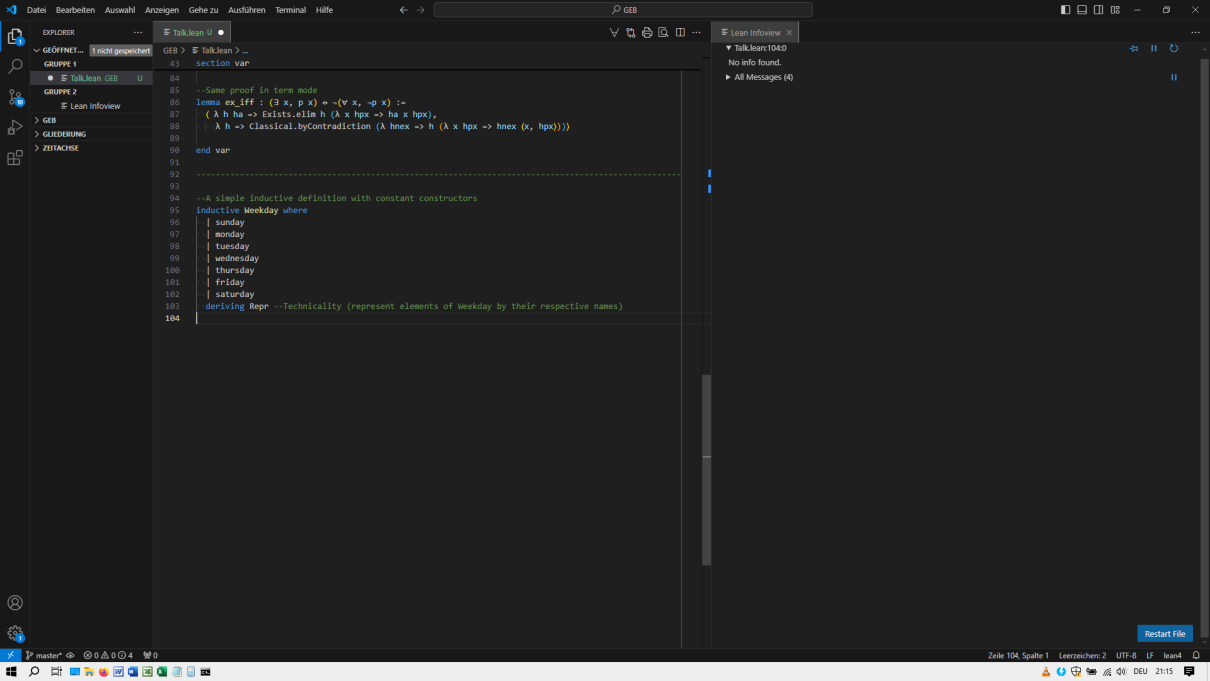


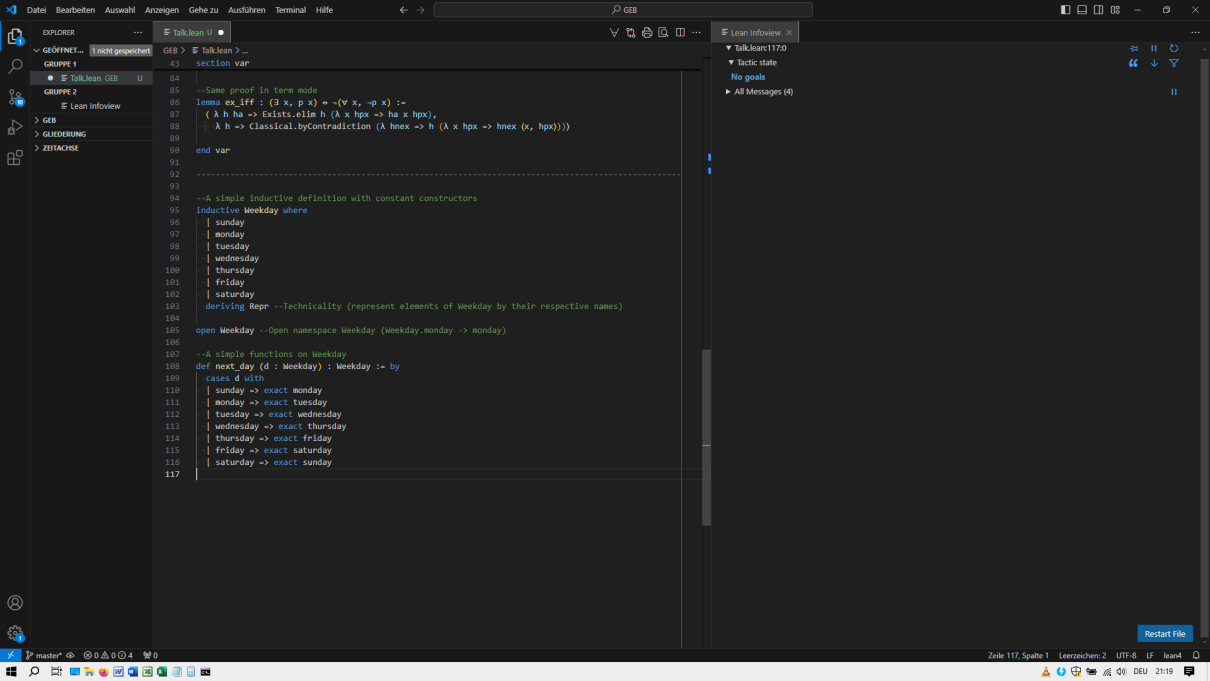

```
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
 fun _ _ y x ↦ y x

-------------------------------------------------------------------------------

section var

--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)

--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
 constructor
 . intro h
   constructor
   . intro x
     exact (h x).1
   . intro x
     exact (h x).2
 . intro h x
   constructor
   . exact h.1 x
   . exact h.2 x

--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
 ⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩

--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
 constructor
 . intro h ha
   cases h with | intro x hpx =>
   have hnpx : ¬p x := by
     exact ha x
   exact hnpx hpx
 . intro h
   apply Classical.byContradiction
   intro hnex
   have ha : ∀ (x : α), ¬p x := by
     intro x
     intro hpx
     have hex : ∃ x, p x := by
       use x

   sorry
```


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
intro x
intro hpx
have hex : ∃ x, p x := by
use x
exact hnex hex
Lean Infoview
Talk.lean:83:4
Tactic state
1 goal
case mpr.h
α : Type
p q : α → Prop
h : ¬∀ (x : α), ¬p x
hnex : ¬∃ x, p x
ha : ∀ (x : α), ¬p x
⊢ False
All Messages (5)
Restart File
master*
Zeile 83, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:08

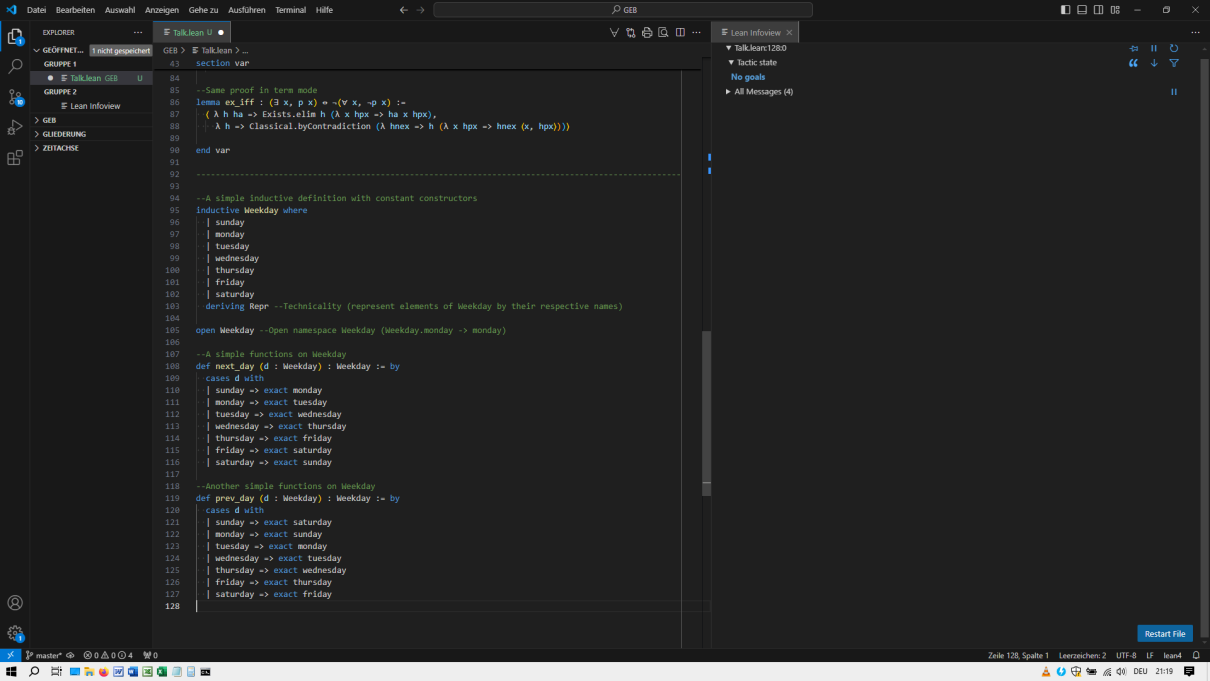

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > { } var
--Copy and paste proof terms ("fun x ↦ y" alternitive to "λ x => y")
example : ∀ (P : Prop) (Q : Prop), (P → Q) → P → Q :=
fun _ _ y x ↦ y x
section var
--Local variables of the present section: a type α and two predicates p and q on α
variable (α : Type) (p q : α → Prop)
--Simple example of predicate logic, proof in tactic mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) := by
constructor
. intro h
constructor
. intro x
exact (h x).1
. intro x
exact (h x).2
. intro h x
constructor
. exact h.1 x
. exact h.2 x
--Same proof in term mode
example : (∀ x, p x ∧ q x) ↔ (∀ x, p x) ∧ (∀ x, q x) :=
⟨λ h => ⟨λ x => (h x).1, λ x => (h x).2⟩, λ h x => ⟨h.1 x, h.2 x⟩⟩
--Simple example of predicate logic, proof in tactic mode
example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
constructor
. intro h ha
cases h with | intro x hpx =>
have hnpx : ¬p x := by
exact ha x
exact hnpx hpx
. intro h
apply Classical.byContradiction
intro hnex
have ha : ∀ (x : α), ¬p x := by
intro x
intro hpx
have hex : ∃ x, p x := by
use x
exact hnex hex
exact h ha
Lean Infoview
Talk.lean:84:0
Tactic state
No goals
All Messages (4)
Restart File
master*
Zeile 84, Spalte 1 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:08

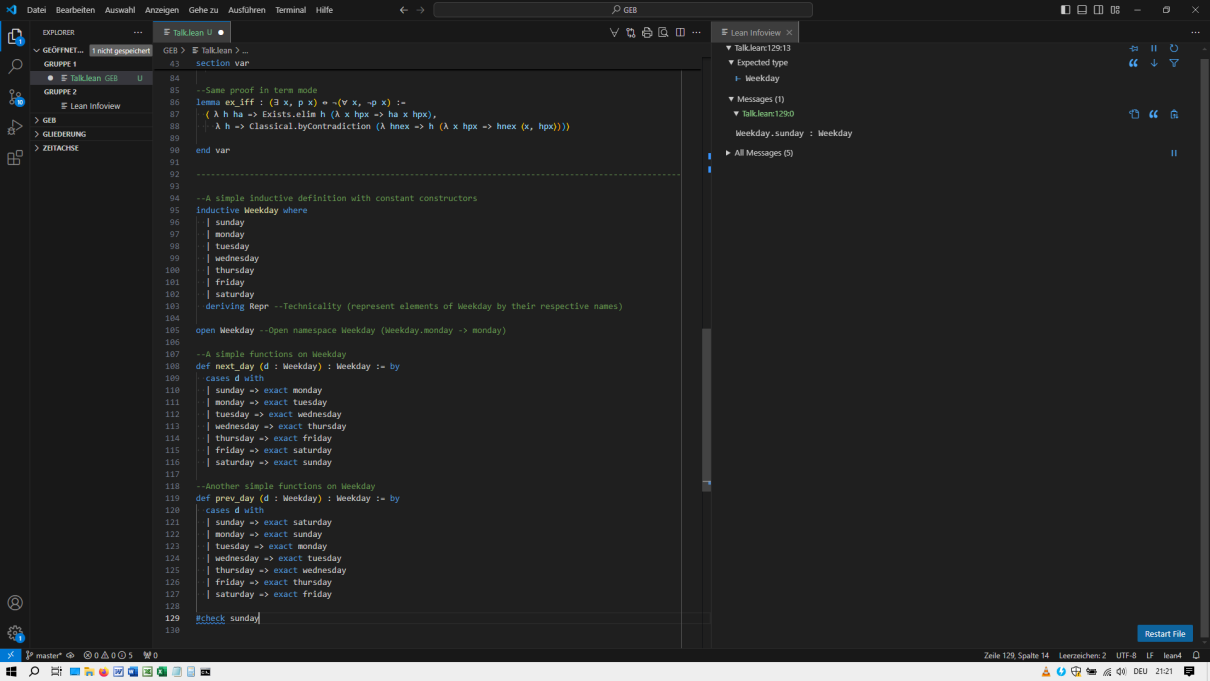

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
43 section var
65
66 --Simple example of predicate logic, proof in tactic mode
67 example : (∃ x, p x) ↔ ¬(∀ x, ¬p x) := by
68 constructor
69 . intro h ha
70 cases h with | intro x hpx =>
71 have hnpx : ¬p x := by
72 exact ha x
73 exact hnpx hpx
74 . intro h
75 apply Classical.byContradiction
76 intro hnex
77 have ha : ∀ (x : α), ¬p x := by
78 intro x
79 intro hpx
80 have hex : ∃ x, p x := by
81 use x
82 exact hnex hex
83 exact h ha
84
85 --Same proof in term mode
86 lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
87 ⟨ λ h ha => Exists.elim h (λ x hpx => ha x hpx),
88 λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩))⟩
89
90 end var
91
Lean Infoview
Talk.lean:91:0
No info found.
All Messages (4)
Restart File
master*
0 0 4
0
Zeile 91, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:10

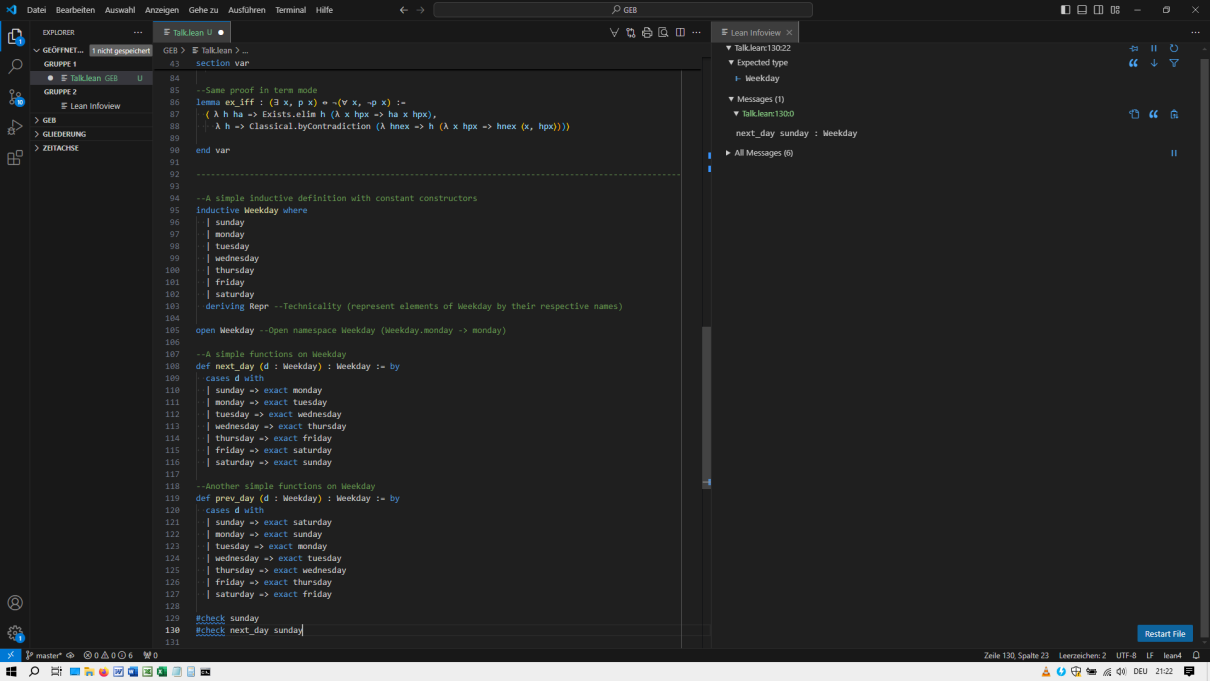

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
43 section var
84
85 --Same proof in term mode
86 lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
87 ⟨ λ h ha => Exists.elim h (λ x hpx => ha x hpx),
88 λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩))⟩
89
90 end var
91
92 ----------
93
94 --A simple inductive definition with constant constructors
95 inductive Weekday where
96 | sunday
97 | monday
98 | tuesday
99 | wednesday
100 | thursday
101 | friday
102 | saturday
103 deriving Repr --Technicality (represent elements of Weekday by their respective names)
104
Lean Infoview
Talk.lean:104:0
No info found.
All Messages (4)
Restart File
master*
Zeile 104, Spalte 1 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:15

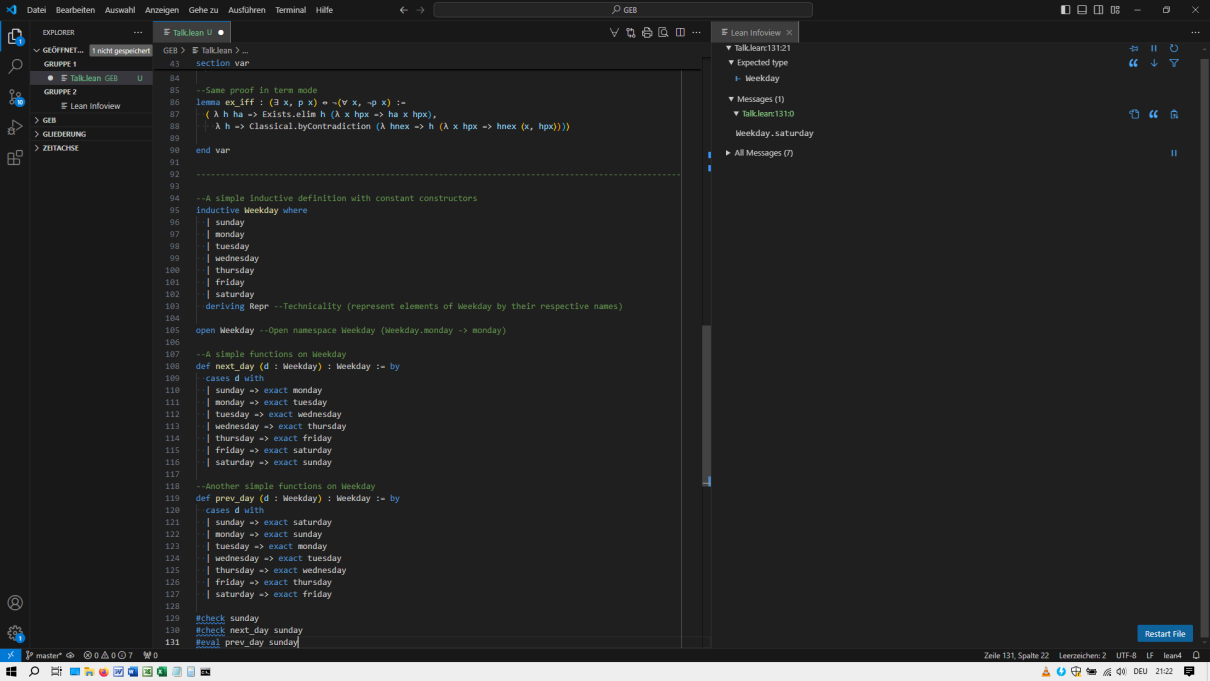


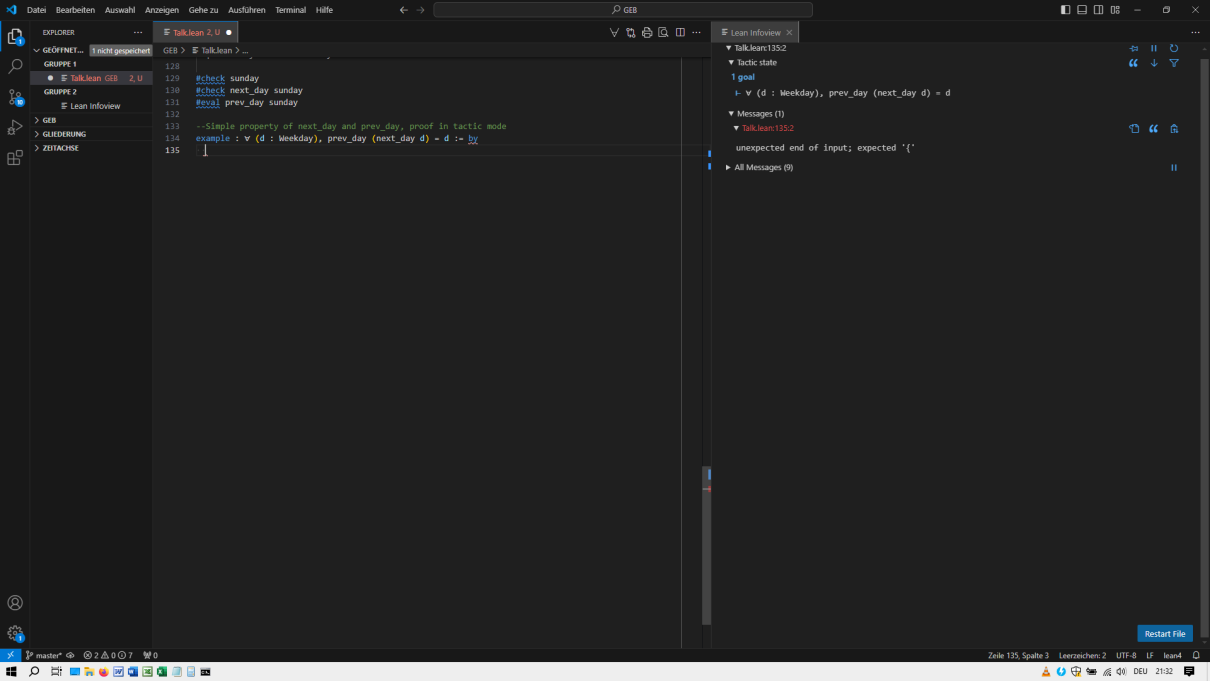

```
43  section var
84
85  --Same proof in term mode
86  lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
87    ⟨ λ h ha => Exists.elim h (λ x hpx => ha x hpx),
88      λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩))⟩
89
90  end var
91
92  ----------------------------------------------------------------------------------------
93
94  --A simple inductive definition with constant constructors
95  inductive Weekday where
96    | sunday
97    | monday
98    | tuesday
99    | wednesday
100   | thursday
101   | friday
102   | saturday
103   deriving Repr --Technicality (represent elements of Weekday by their respective names)
104
105 open Weekday --Open namespace Weekday (Weekday.monday -> monday)
106
107 --A simple functions on Weekday
108 def next_day (d : Weekday) : Weekday := by
109   cases d with
110   | sunday => exact monday
111   | monday => exact tuesday
112   | tuesday => exact wednesday
113   | wednesday => exact thursday
114   | thursday => exact friday
115   | friday => exact saturday
116   | saturday => exact sunday
117
```

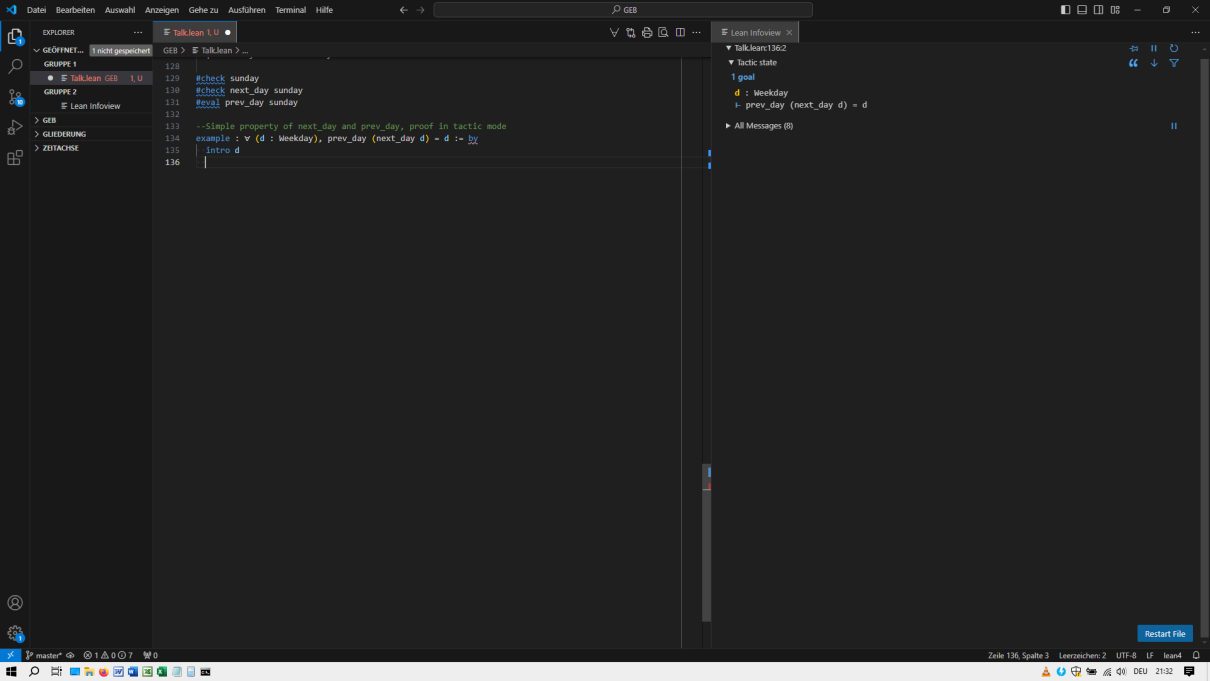

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
43 section var
84
85 --Same proof in term mode
86 lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
87 (λ h ha => Exists.elim h (λ x hpx => ha x hpx),
88 λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩)))
89
90 end var
91
92 ----------
93
94 --A simple inductive definition with constant constructors
95 inductive Weekday where
96 | sunday
97 | monday
98 | tuesday
99 | wednesday
100 | thursday
101 | friday
102 | saturday
103 deriving Repr --Technicality (represent elements of Weekday by their respective names)
104
105 open Weekday --Open namespace Weekday (Weekday.monday -> monday)
106
107 --A simple functions on Weekday
108 def next_day (d : Weekday) : Weekday := by
109 cases d with
110 | sunday => exact monday
111 | monday => exact tuesday
112 | tuesday => exact wednesday
113 | wednesday => exact thursday
114 | thursday => exact friday
115 | friday => exact saturday
116 | saturday => exact sunday
117
118 --Another simple functions on Weekday
119 def prev_day (d : Weekday) : Weekday := by
120 cases d with
121 | sunday => exact saturday
122 | monday => exact sunday
123 | tuesday => exact monday
124 | wednesday => exact tuesday
125 | thursday => exact wednesday
126 | friday => exact thursday
127 | saturday => exact friday
128
Lean Infoview
Talk.lean:128:0
Tactic state
No goals
All Messages (4)
Restart File
master*
Zeile 128, Spalte 1 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:19

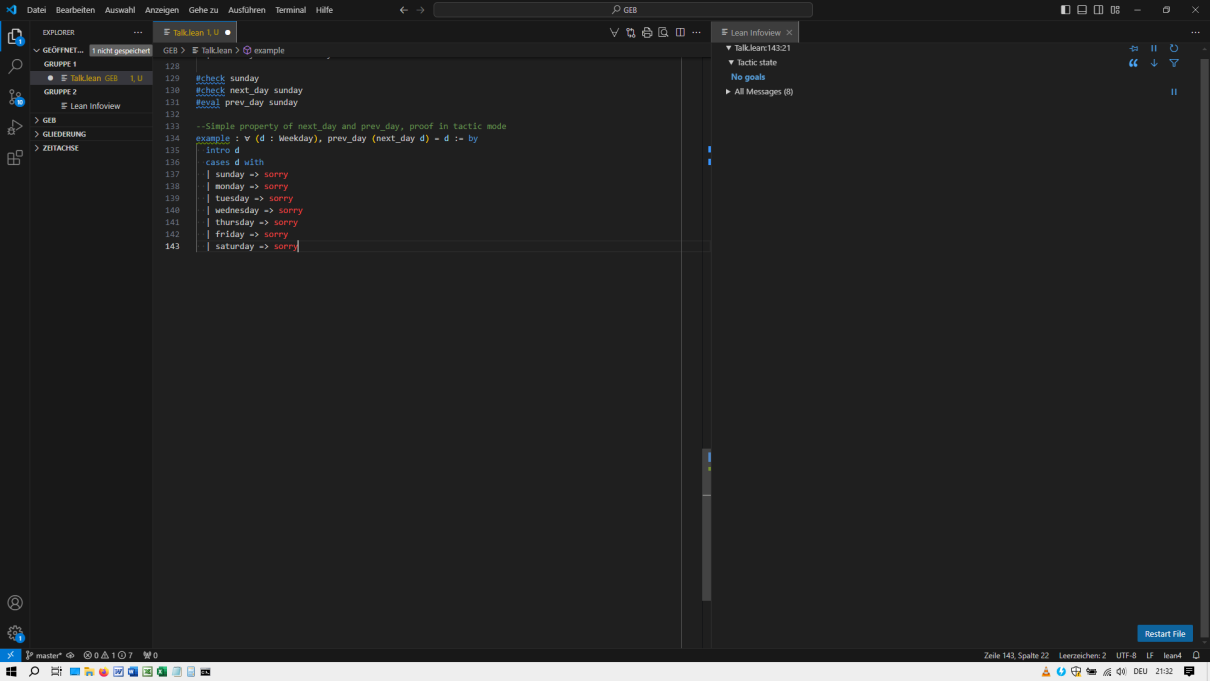

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
section var
--Same proof in term mode
lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
(λ h ha => Exists.elim h (λ x hpx => ha x hpx),
λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex (x, hpx))))
end var
--A simple inductive definition with constant constructors
inductive Weekday where
| sunday
| monday
| tuesday
| wednesday
| thursday
| friday
| saturday
deriving Repr --Technicality (represent elements of Weekday by their respective names)
open Weekday --Open namespace Weekday (Weekday.monday -> monday)
--A simple functions on Weekday
def next_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact monday
| monday => exact tuesday
| tuesday => exact wednesday
| wednesday => exact thursday
| thursday => exact friday
| friday => exact saturday
| saturday => exact sunday
--Another simple functions on Weekday
def prev_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact saturday
| monday => exact sunday
| tuesday => exact monday
| wednesday => exact tuesday
| thursday => exact wednesday
| friday => exact thursday
| saturday => exact friday
#check sunday
Lean Infoview
Talk.lean:129:13
Expected type
Weekday
Messages (1)
Talk.lean:129:0
Weekday.sunday : Weekday
All Messages (5)
Restart File
master*
Zeile 129, Spalte 14 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:21

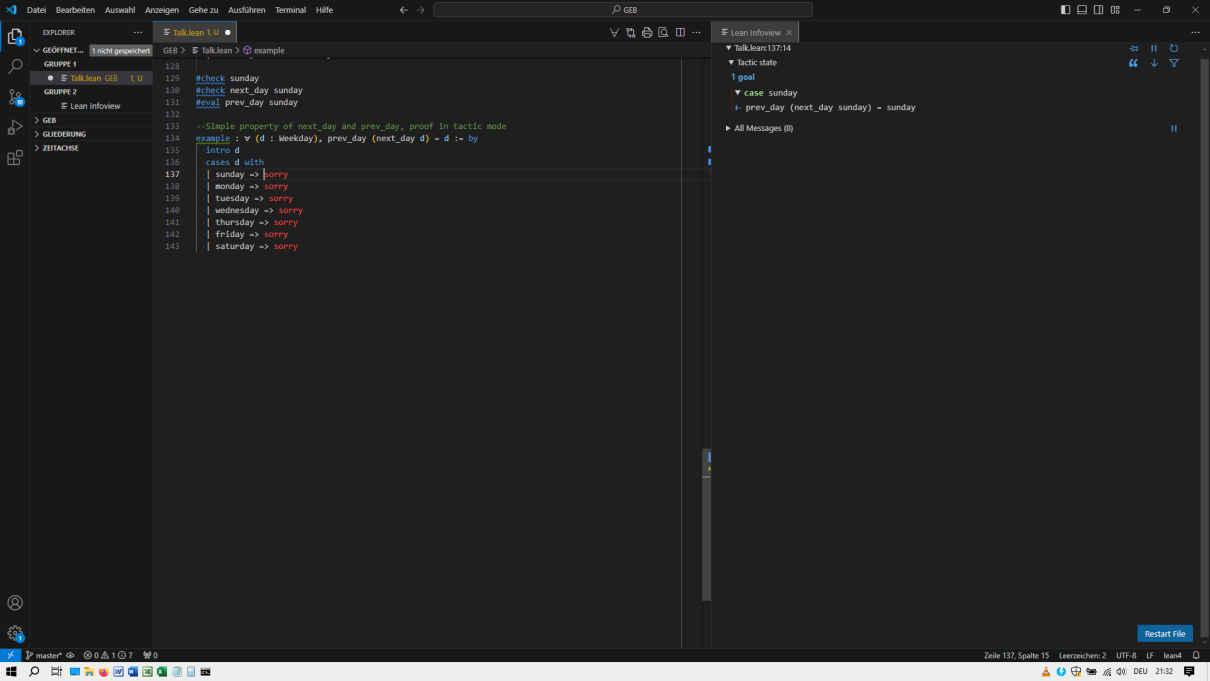

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
section var
--Same proof in term mode
lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
(λ h ha => Exists.elim h (λ x hpx => ha x hpx),
λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩)))
end var
--A simple inductive definition with constant constructors
inductive Weekday where
| sunday
| monday
| tuesday
| wednesday
| thursday
| friday
| saturday
deriving Repr --Technicality (represent elements of Weekday by their respective names)
open Weekday --Open namespace Weekday (Weekday.monday -> monday)
--A simple functions on Weekday
def next_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact monday
| monday => exact tuesday
| tuesday => exact wednesday
| wednesday => exact thursday
| thursday => exact friday
| friday => exact saturday
| saturday => exact sunday
--Another simple functions on Weekday
def prev_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact saturday
| monday => exact sunday
| tuesday => exact monday
| wednesday => exact tuesday
| thursday => exact wednesday
| friday => exact thursday
| saturday => exact friday
#check sunday
#check next_day sunday
Lean Infoview
Talk.lean:130:22
Expected type
Weekday
Messages (1)
Talk.lean:130:0
next_day sunday : Weekday
All Messages (6)
Restart File
master*
Zeile 130, Spalte 23 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:22

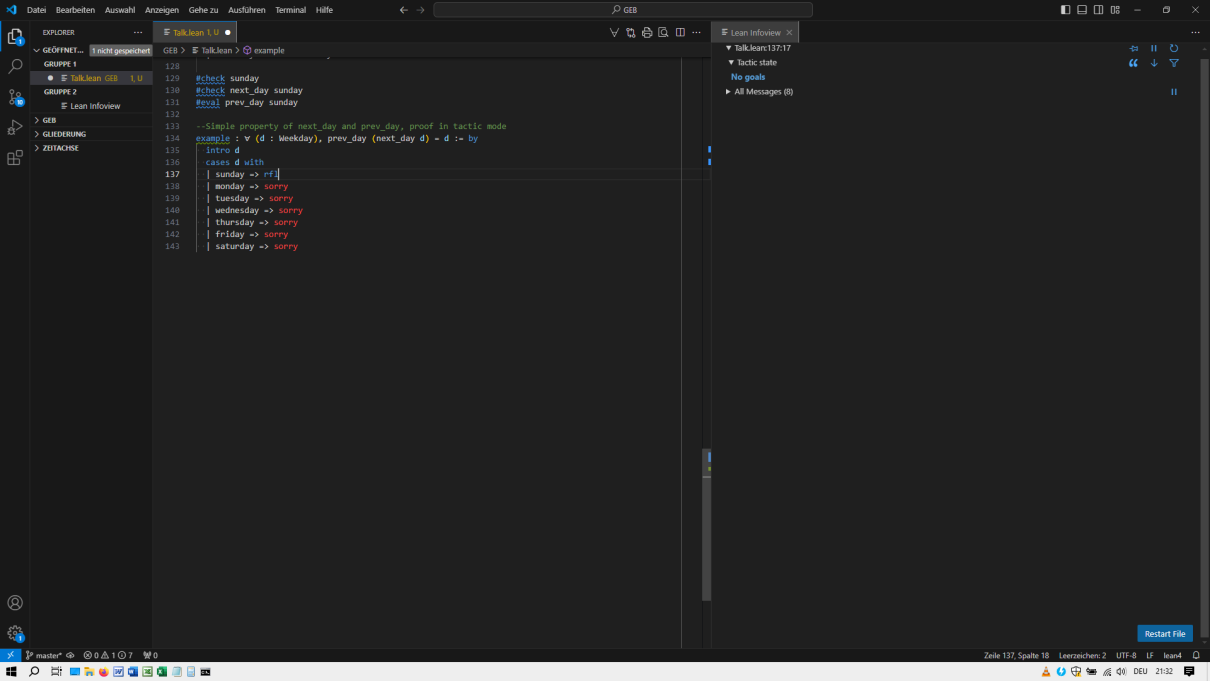

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
section var
--Same proof in term mode
lemma ex_iff : (∃ x, p x) ↔ ¬(∀ x, ¬p x) :=
(λ h ha => Exists.elim h (λ x hpx => ha x hpx),
λ h => Classical.byContradiction (λ hnex => h (λ x hpx => hnex ⟨x, hpx⟩)))
end var
--A simple inductive definition with constant constructors
inductive Weekday where
| sunday
| monday
| tuesday
| wednesday
| thursday
| friday
| saturday
deriving Repr --Technicality (represent elements of Weekday by their respective names)
open Weekday --Open namespace Weekday (Weekday.monday -> monday)
--A simple functions on Weekday
def next_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact monday
| monday => exact tuesday
| tuesday => exact wednesday
| wednesday => exact thursday
| thursday => exact friday
| friday => exact saturday
| saturday => exact sunday
--Another simple functions on Weekday
def prev_day (d : Weekday) : Weekday := by
cases d with
| sunday => exact saturday
| monday => exact sunday
| tuesday => exact monday
| wednesday => exact tuesday
| thursday => exact wednesday
| friday => exact thursday
| saturday => exact friday
#check sunday
#check next_day sunday
#eval prev_day sunday
Lean Infoview
Talk.lean:131:21
Expected type
Weekday
Messages (1)
Talk.lean:131:0
Weekday.saturday
All Messages (7)
Restart File
master*
Zeile 131, Spalte 22 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:22

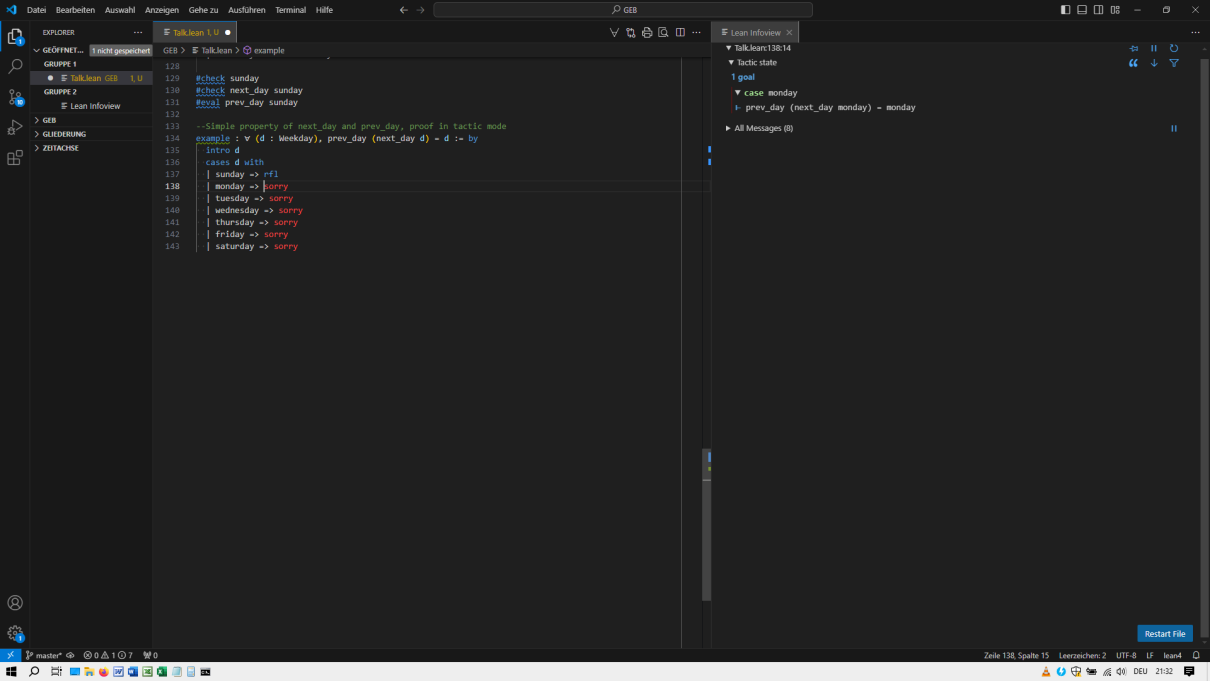

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
128
129 #check sunday
130 #check next_day sunday
131 #eval prev_day sunday
132
133 --Simple property of next_day and prev_day, proof in tactic mode
134 example : ∀ (d : Weekday), prev_day (next_day d) = d := by
135
Lean Infoview
Talk.lean:135:2
Tactic state
1 goal
⊢ ∀ (d : Weekday), prev_day (next_day d) = d
Messages (1)
Talk.lean:135:2
unexpected end of input; expected '{'
All Messages (9)
Restart File
master*
2 0 7
0
Zeile 135, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

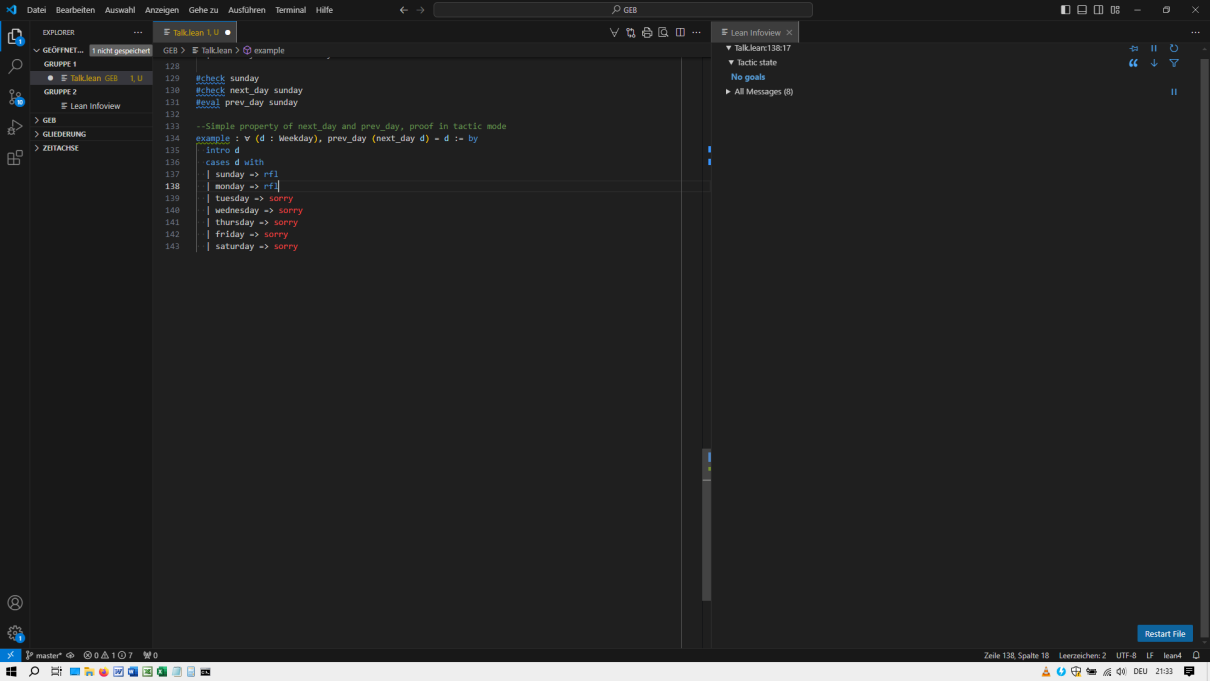

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
128
129 #check sunday
130 #check next_day sunday
131 #eval prev_day sunday
132
133 --Simple property of next_day and prev_day, proof in tactic mode
134 example : ∀ (d : Weekday), prev_day (next_day d) = d := by
135 intro d
136
Lean Infoview
Talk.lean:136:2
Tactic state
1 goal
d : Weekday
⊢ prev_day (next_day d) = d
All Messages (8)
Restart File
master*
Zeile 136, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

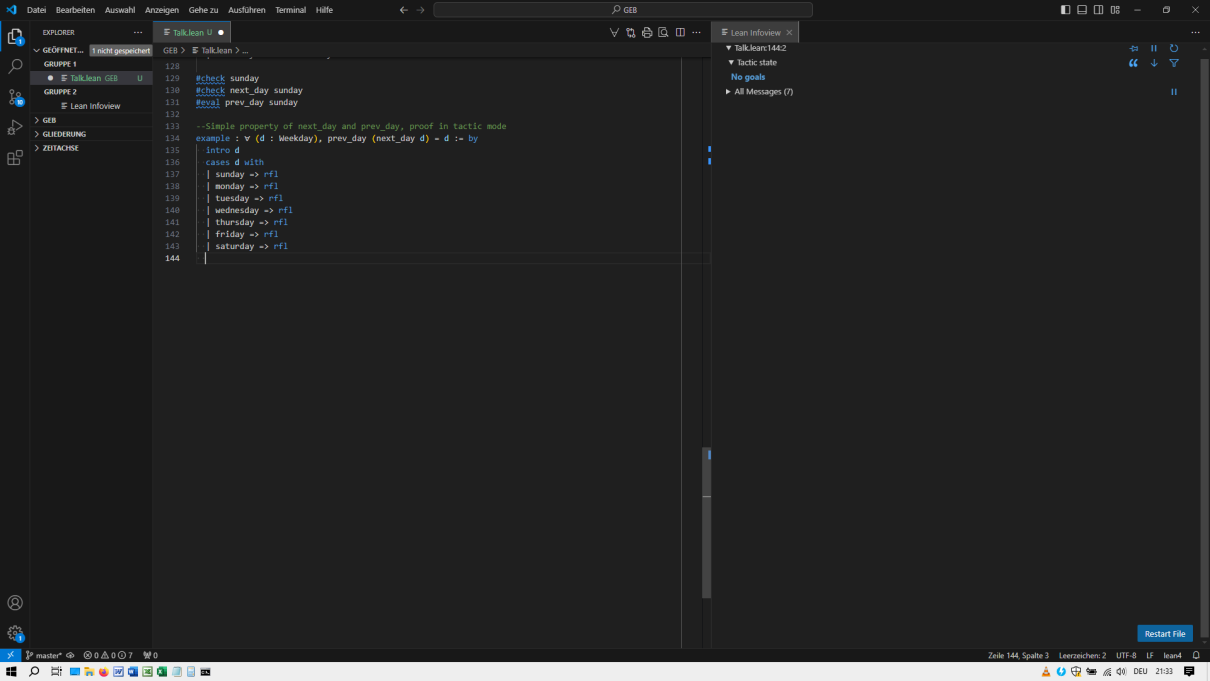

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => sorry
| monday => sorry
| tuesday => sorry
| wednesday => sorry
| thursday => sorry
| friday => sorry
| saturday => sorry
Lean Infoview
Talk.lean:143:21
Tactic state
No goals
All Messages (8)
Restart File
master*
Zeile 143, Spalte 22
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

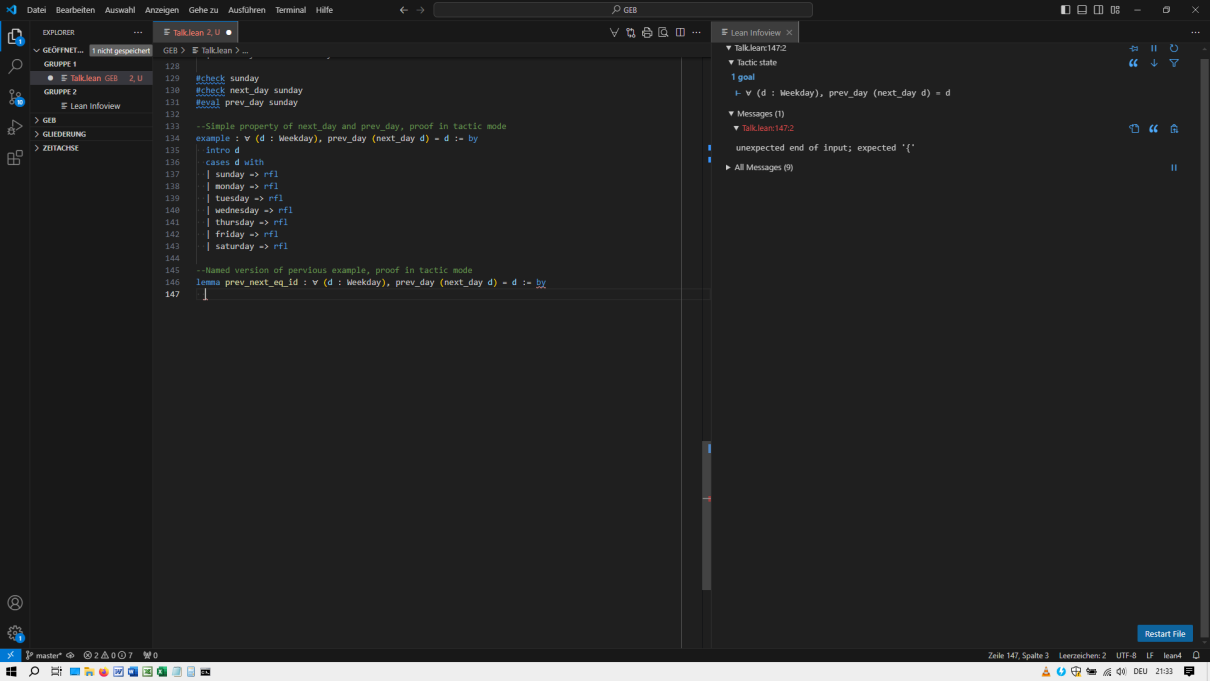

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
128
129 #check sunday
130 #check next_day sunday
131 #eval prev_day sunday
132
133 --Simple property of next_day and prev_day, proof in tactic mode
134 example : ∀ (d : Weekday), prev_day (next_day d) = d := by
135 intro d
136 cases d with
137 | sunday => sorry
138 | monday => sorry
139 | tuesday => sorry
140 | wednesday => sorry
141 | thursday => sorry
142 | friday => sorry
143 | saturday => sorry
Lean Infoview
Talk.lean:137:14
Tactic state
1 goal
case sunday
prev_day (next_day sunday) = sunday
All Messages (8)
Restart File
master*
0 1 7
0
Zeile 137, Spalte 15
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

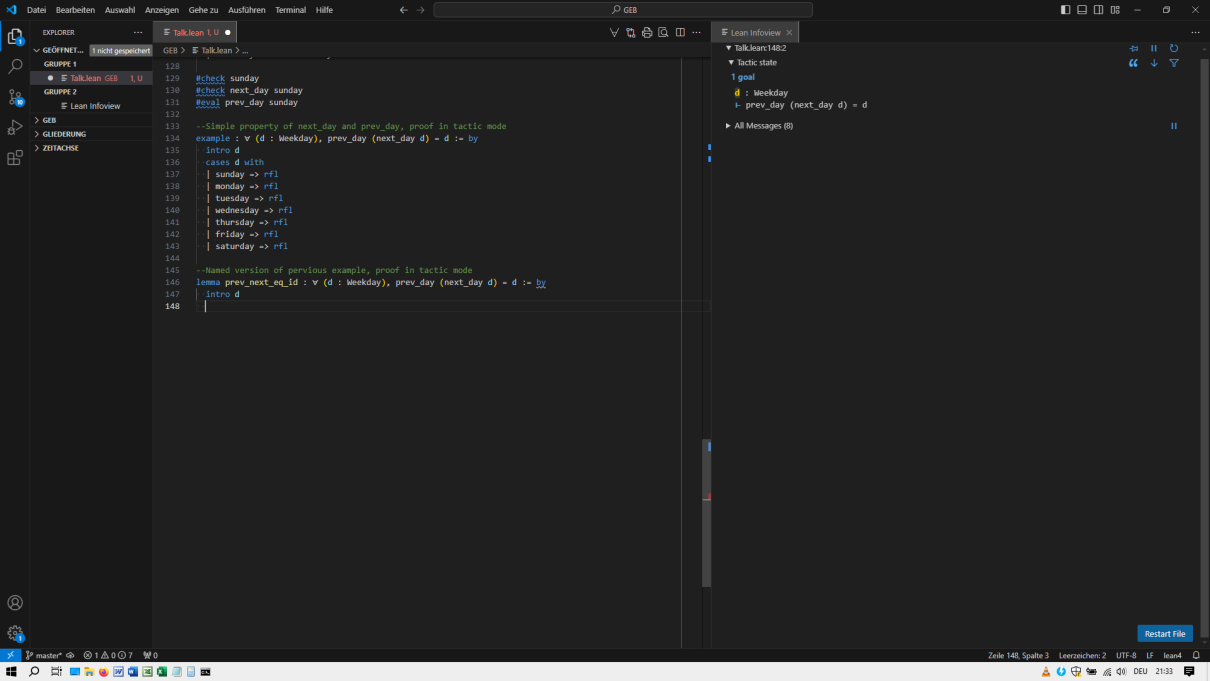

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
128
129 #check sunday
130 #check next_day sunday
131 #eval prev_day sunday
132
133 --Simple property of next_day and prev_day, proof in tactic mode
134 example : ∀ (d : Weekday), prev_day (next_day d) = d := by
135 intro d
136 cases d with
137 | sunday => rfl
138 | monday => sorry
139 | tuesday => sorry
140 | wednesday => sorry
141 | thursday => sorry
142 | friday => sorry
143 | saturday => sorry
Lean Infoview
Talk.lean:137:17
Tactic state
No goals
All Messages (8)
Restart File
master*
0 1 7
0
Zeile 137, Spalte 18
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

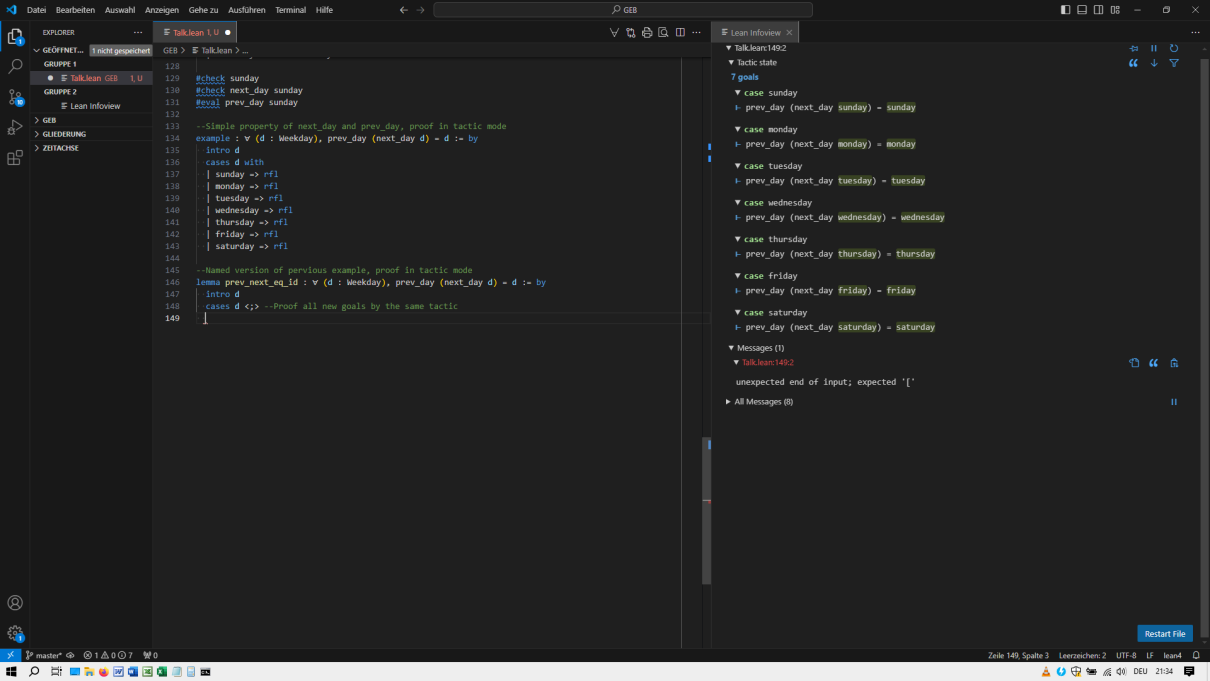

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => sorry
| tuesday => sorry
| wednesday => sorry
| thursday => sorry
| friday => sorry
| saturday => sorry
Lean Infoview
Talk.lean:138:14
Tactic state
1 goal
case monday
prev_day (next_day monday) = monday
All Messages (8)
Restart File
master*
Zeile 138, Spalte 15
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:32

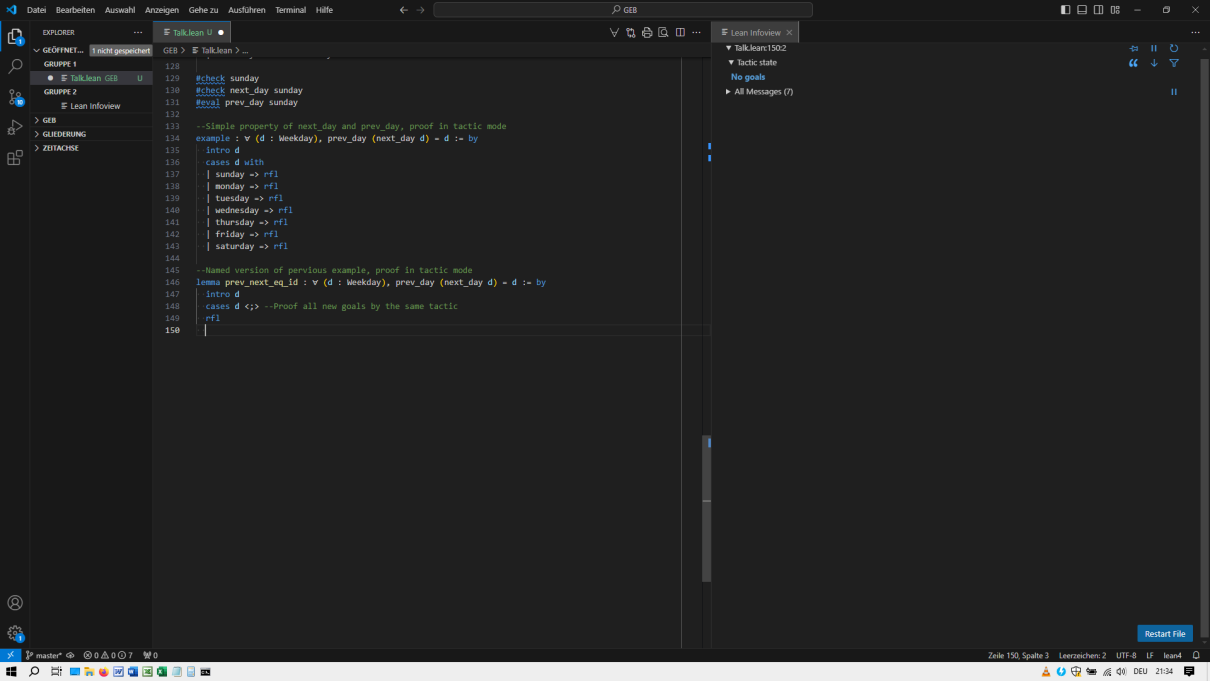

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > example
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => sorry
| wednesday => sorry
| thursday => sorry
| friday => sorry
| saturday => sorry
Lean Infoview
Talk.lean:138:17
Tactic state
No goals
All Messages (8)
Restart File
master*
Zeile 138, Spalte 18 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:33

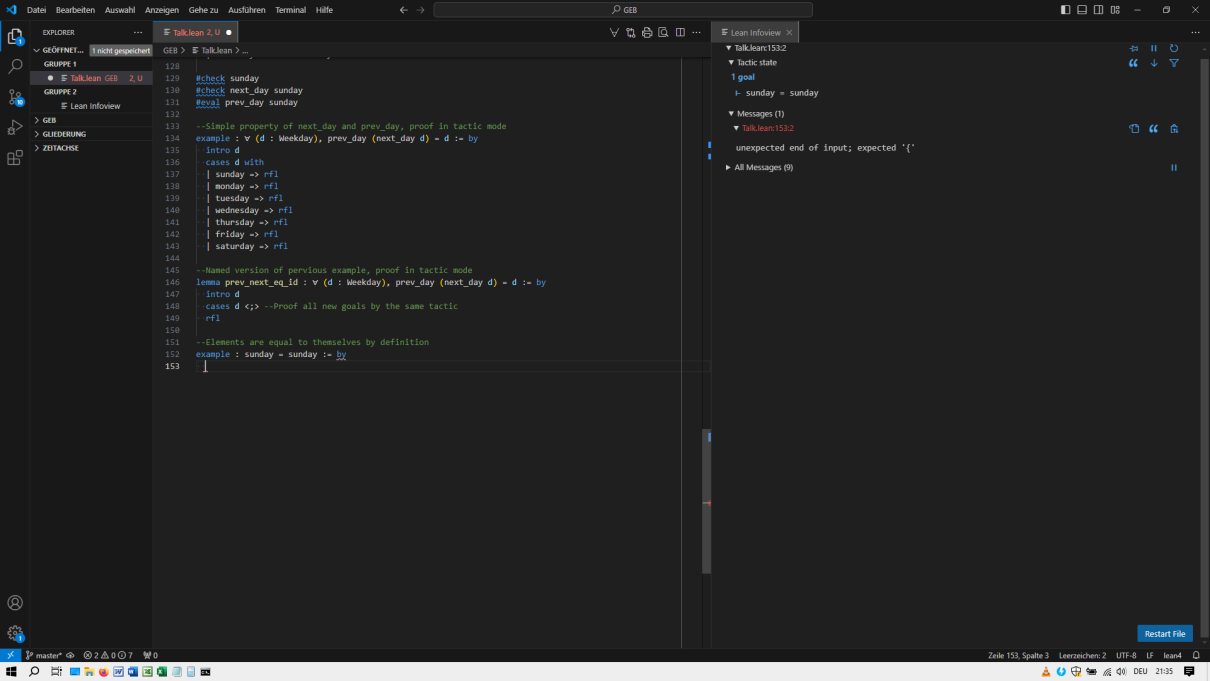

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
Lean Infoview
Talk.lean:144:2
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 144, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:33

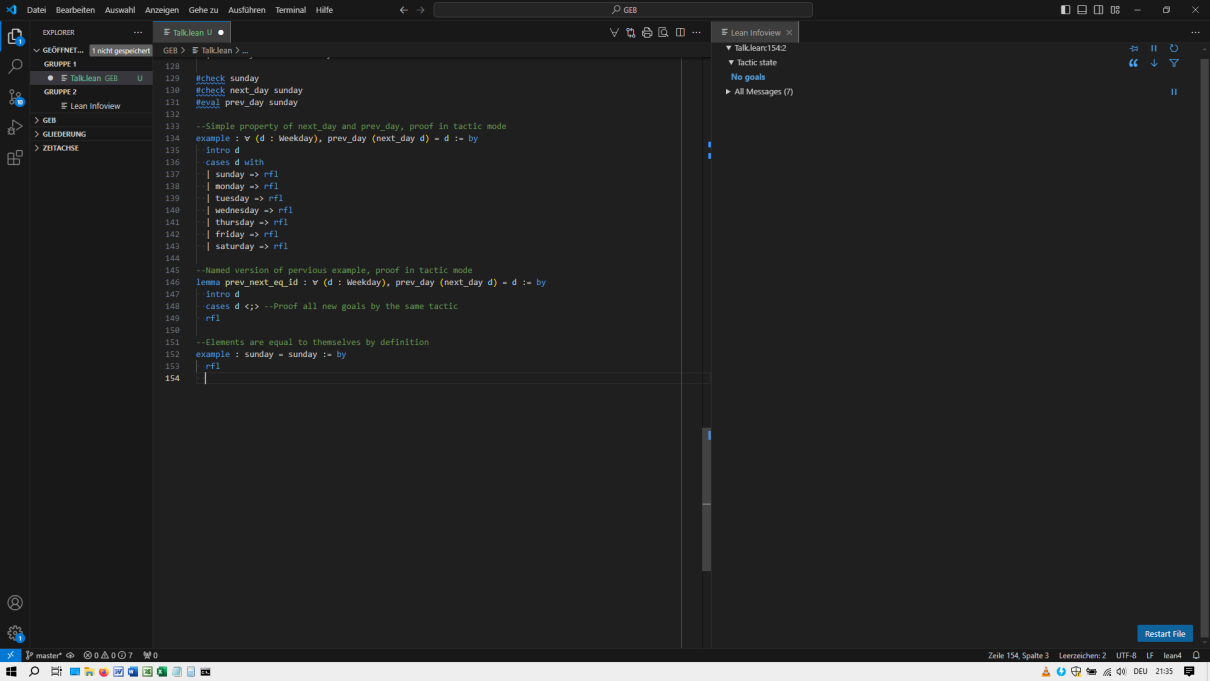

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
Lean Infoview
Talk.lean:147:2
Tactic state
1 goal
⊢ ∀ (d : Weekday), prev_day (next_day d) = d
Messages (1)
Talk.lean:147:2
unexpected end of input; expected '{'
All Messages (9)
Restart File
master*
Zeile 147, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:33

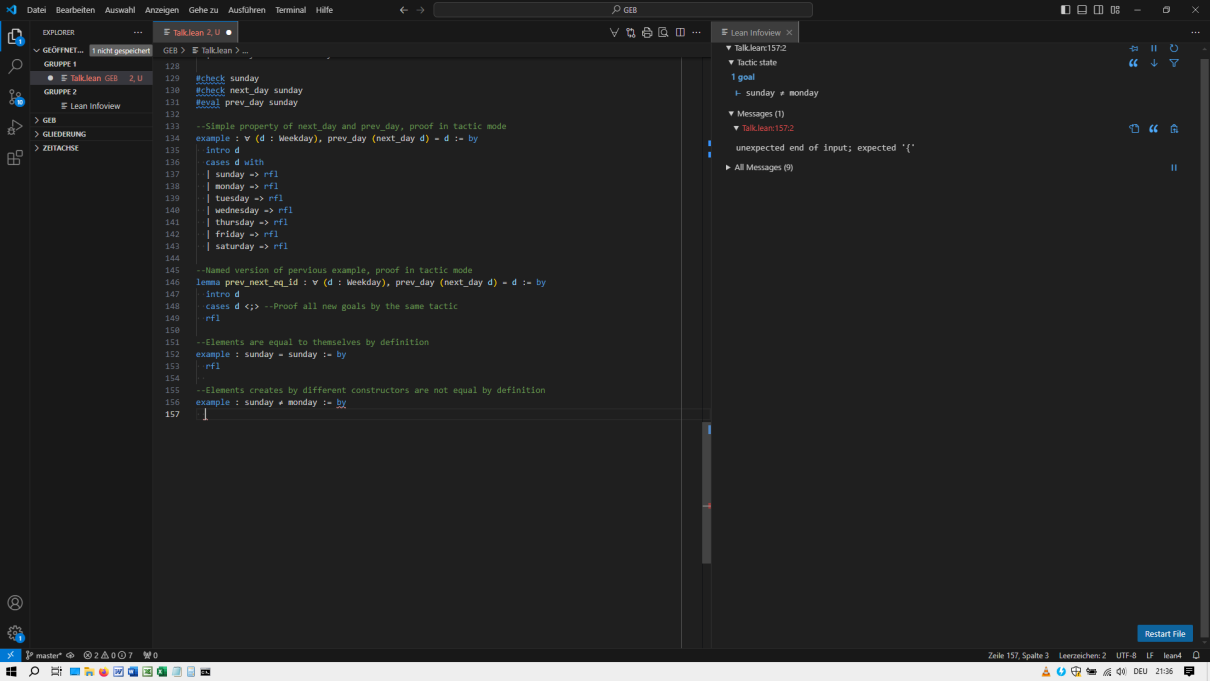

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
Lean Infoview
Talk.lean:148:2
Tactic state
1 goal
d : Weekday
⊢ prev_day (next_day d) = d
All Messages (8)
Restart File
master*
Zeile 148, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:33

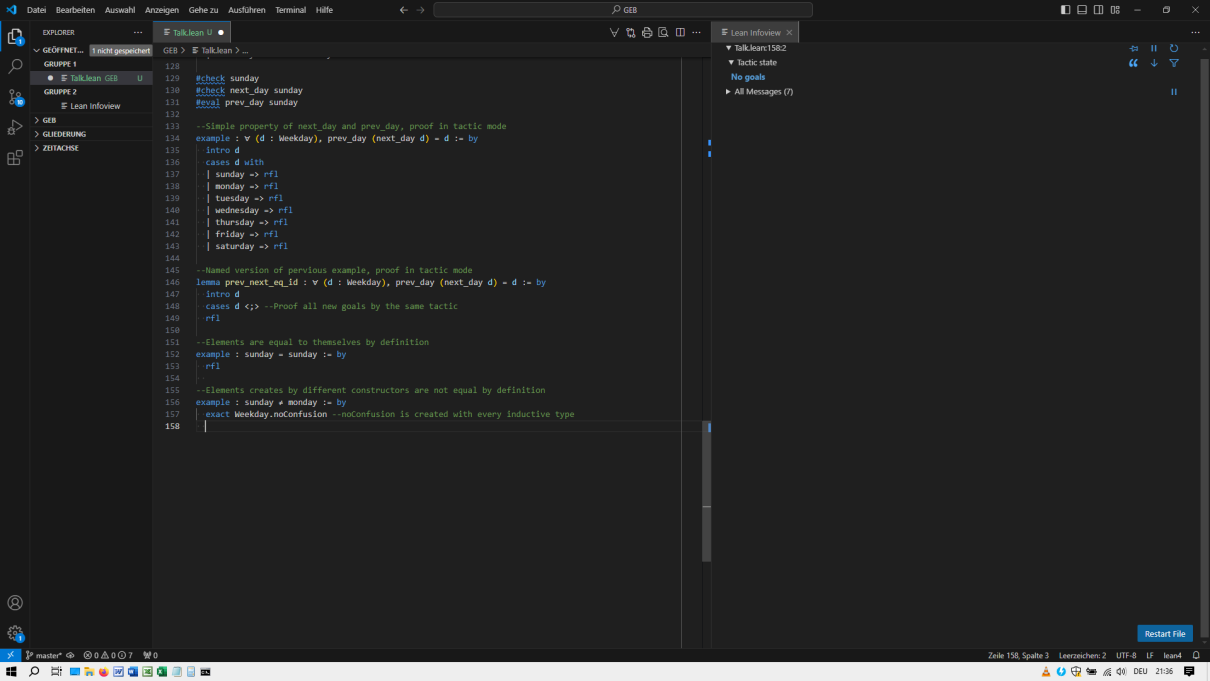

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
Lean Infoview
Talk.lean:149:2
Tactic state
7 goals
case sunday
prev_day (next_day sunday) = sunday
case monday
prev_day (next_day monday) = monday
case tuesday
prev_day (next_day tuesday) = tuesday
case wednesday
prev_day (next_day wednesday) = wednesday
case thursday
prev_day (next_day thursday) = thursday
case friday
prev_day (next_day friday) = friday
case saturday
prev_day (next_day saturday) = saturday
Messages (1)
Talk.lean:149:2
unexpected end of input; expected '['
All Messages (8)
Restart File
master*
Zeile 149, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:34

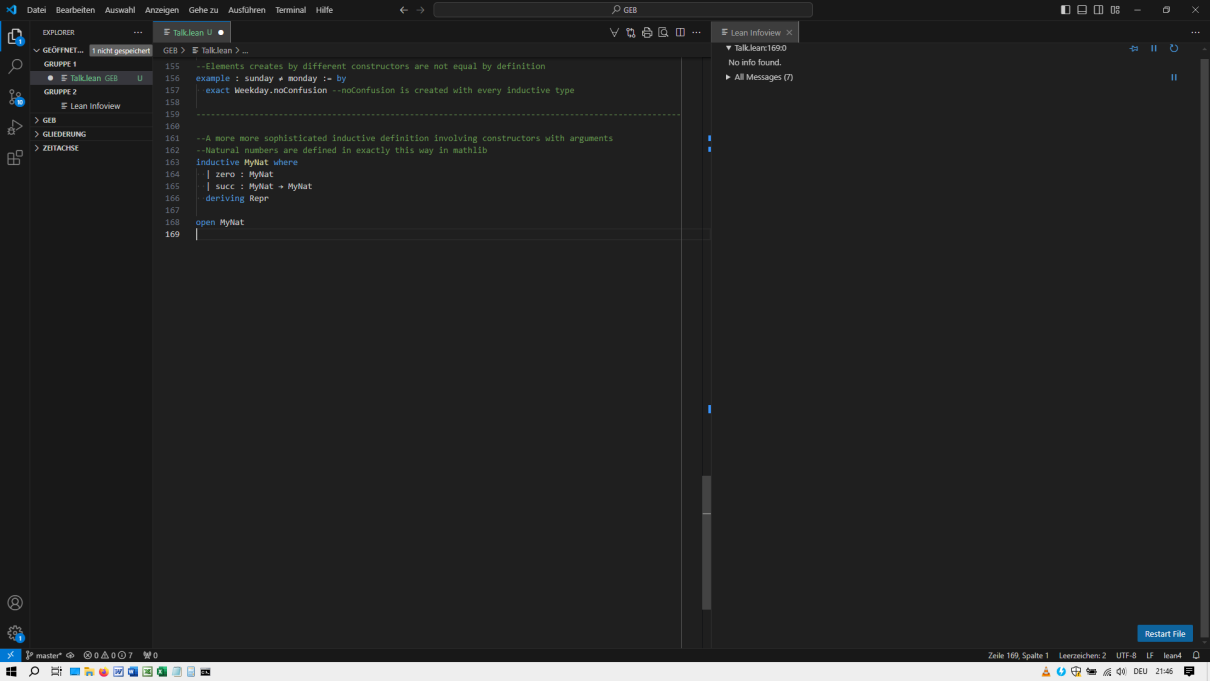

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
rfl
Lean Infoview
Talk.lean:150:2
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 150, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:34

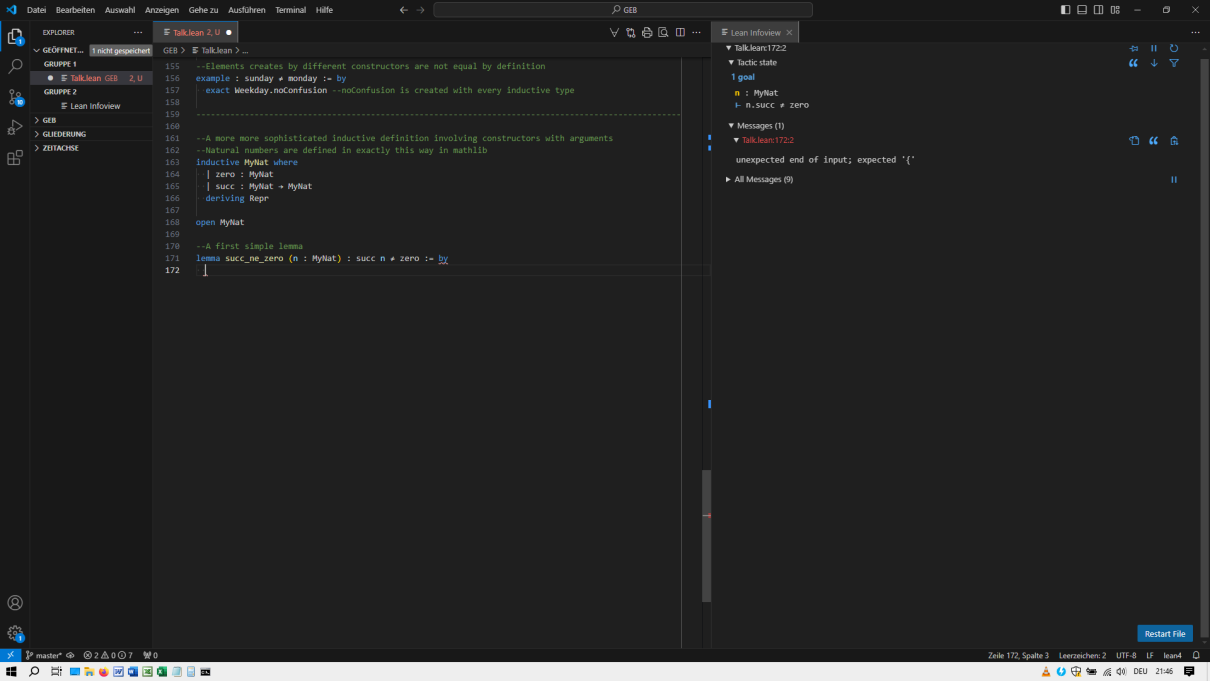

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
rfl
--Elements are equal to themselves by definition
example : sunday = sunday := by
Lean Infoview
Talk.lean:153:2
Tactic state
1 goal
⊢ sunday = sunday
Messages (1)
Talk.lean:153:2
unexpected end of input; expected '{'
All Messages (9)
Restart File
master*
Zeile 153, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:35

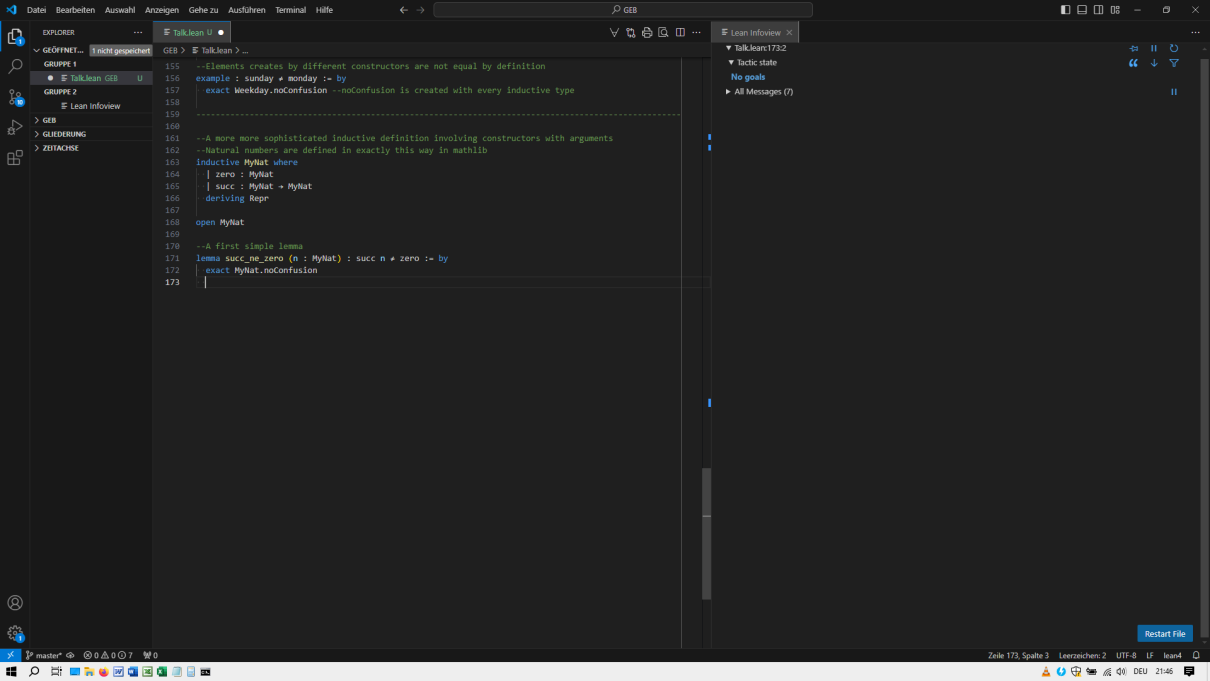

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
rfl
--Elements are equal to themselves by definition
example : sunday = sunday := by
rfl
Lean Infoview
Talk.lean:154:2
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 154, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:35

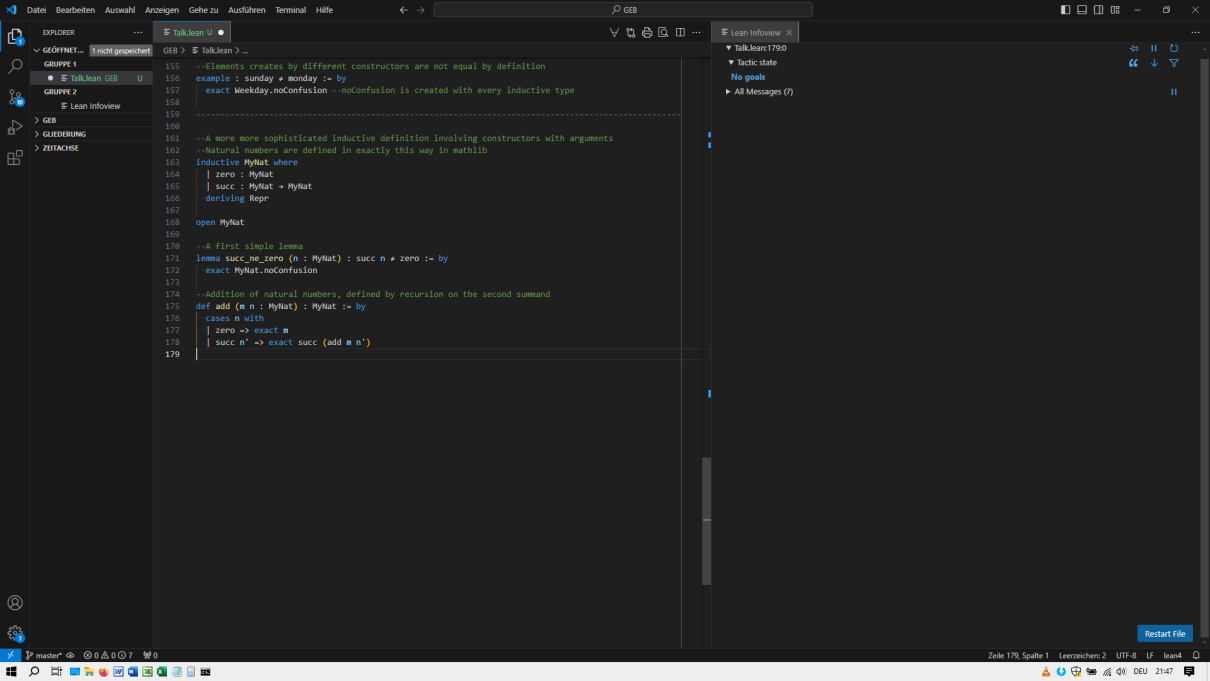

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
rfl
--Elements are equal to themselves by definition
example : sunday = sunday := by
rfl
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
Lean Infoview
Talk.lean:157:2
Tactic state
1 goal
⊢ sunday ≠ monday
Messages (1)
Talk.lean:157:2
unexpected end of input; expected '{'
All Messages (9)
Restart File
master*
Zeile 157, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:36

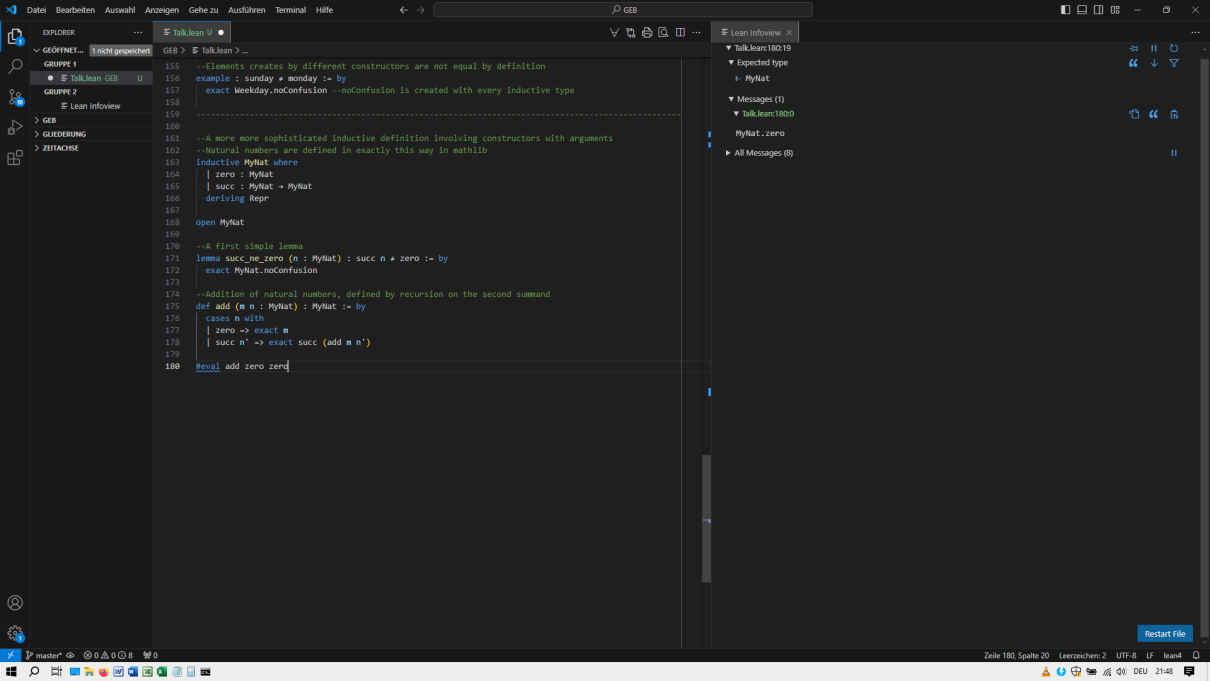

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
#check sunday
#check next_day sunday
#eval prev_day sunday
--Simple property of next_day and prev_day, proof in tactic mode
example : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d with
| sunday => rfl
| monday => rfl
| tuesday => rfl
| wednesday => rfl
| thursday => rfl
| friday => rfl
| saturday => rfl
--Named version of pervious example, proof in tactic mode
lemma prev_next_eq_id : ∀ (d : Weekday), prev_day (next_day d) = d := by
intro d
cases d <;> --Proof all new goals by the same tactic
rfl
--Elements are equal to themselves by definition
example : sunday = sunday := by
rfl
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
Lean Infoview
Talk.lean:158:2
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 158, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:36

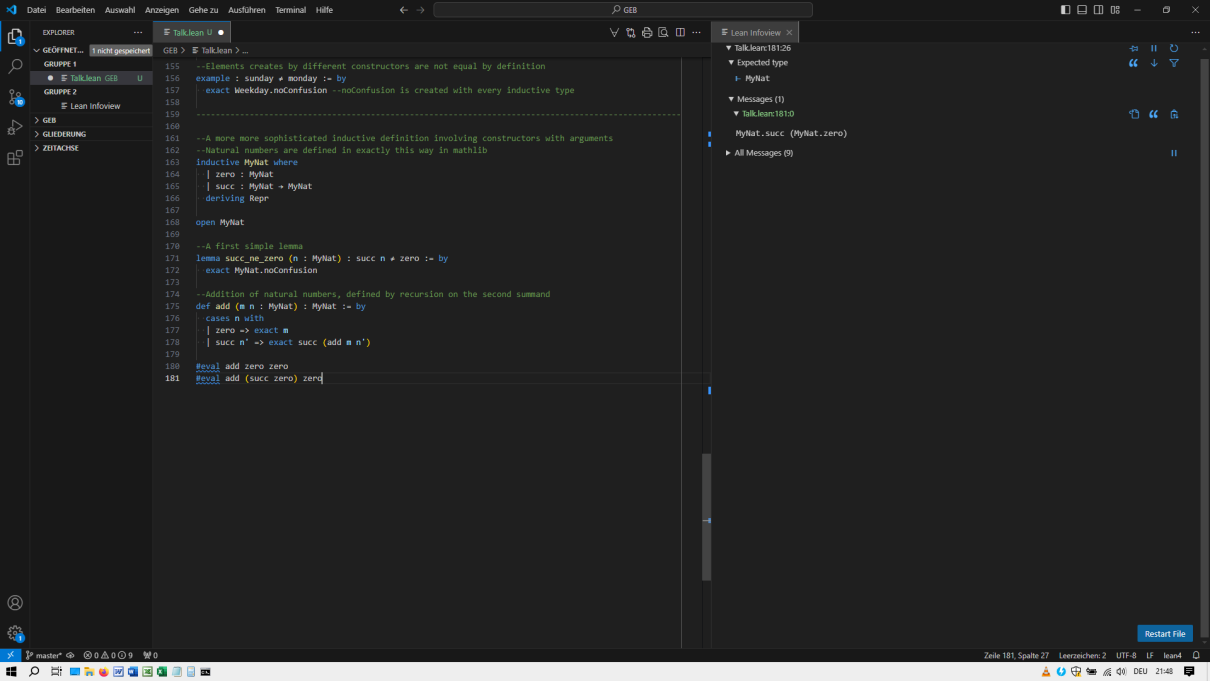

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
155 --Elements creates by different constructors are not equal by definition
156 example : sunday ≠ monday := by
157 exact Weekday.noConfusion --noConfusion is created with every inductive type
158
159 ----------
160
161 --A more more sophisticated inductive definition involving constructors with arguments
162 --Natural numbers are defined in exactly this way in mathlib
163 inductive MyNat where
164 | zero : MyNat
165 | succ : MyNat → MyNat
166 deriving Repr
167
168 open MyNat
169
Lean Infoview
Talk.lean:169:0
No info found.
All Messages (7)
Restart File
master*
Zeile 169, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:46

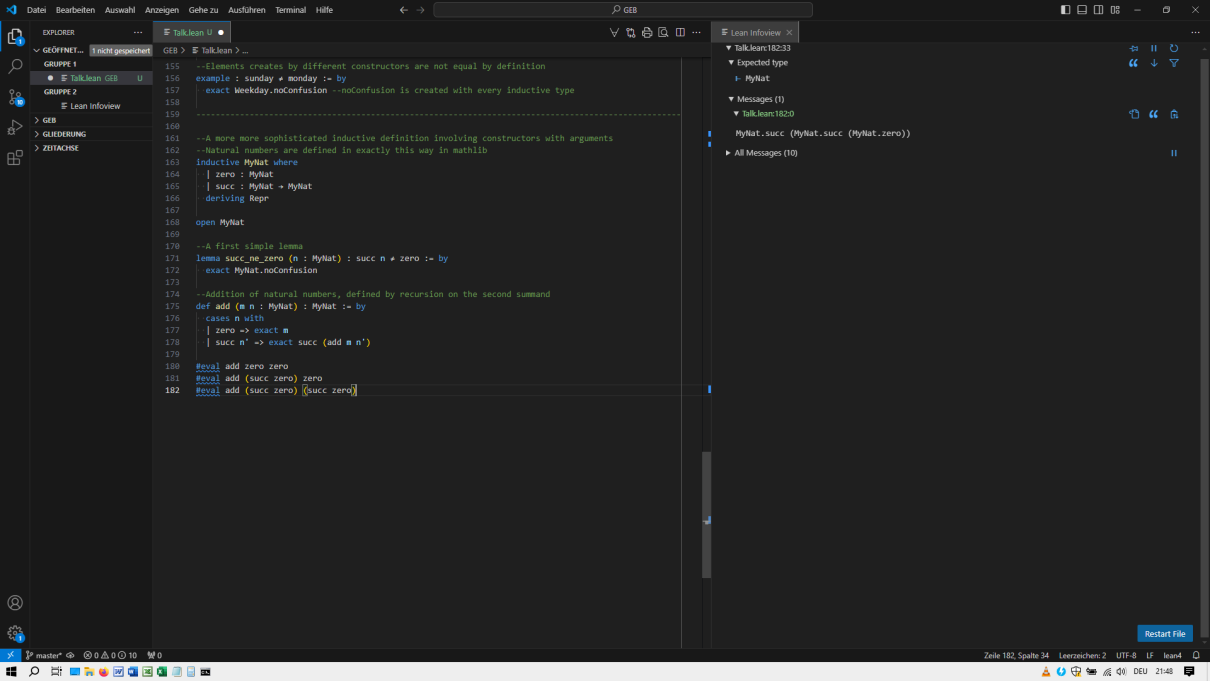

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
155 --Elements creates by different constructors are not equal by definition
156 example : sunday ≠ monday := by
157 exact Weekday.noConfusion --noConfusion is created with every inductive type
158
159
160
161 --A more more sophisticated inductive definition involving constructors with arguments
162 --Natural numbers are defined in exactly this way in mathlib
163 inductive MyNat where
164 | zero : MyNat
165 | succ : MyNat → MyNat
166 deriving Repr
167
168 open MyNat
169
170 --A first simple lemma
171 lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
172
Lean Infoview
Talk.lean:172:2
Tactic state
1 goal
n : MyNat
⊢ n.succ ≠ zero
Messages (1)
Talk.lean:172:2
unexpected end of input; expected '{'
All Messages (9)
Restart File
master*
2 0 7
0
Zeile 172, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:46

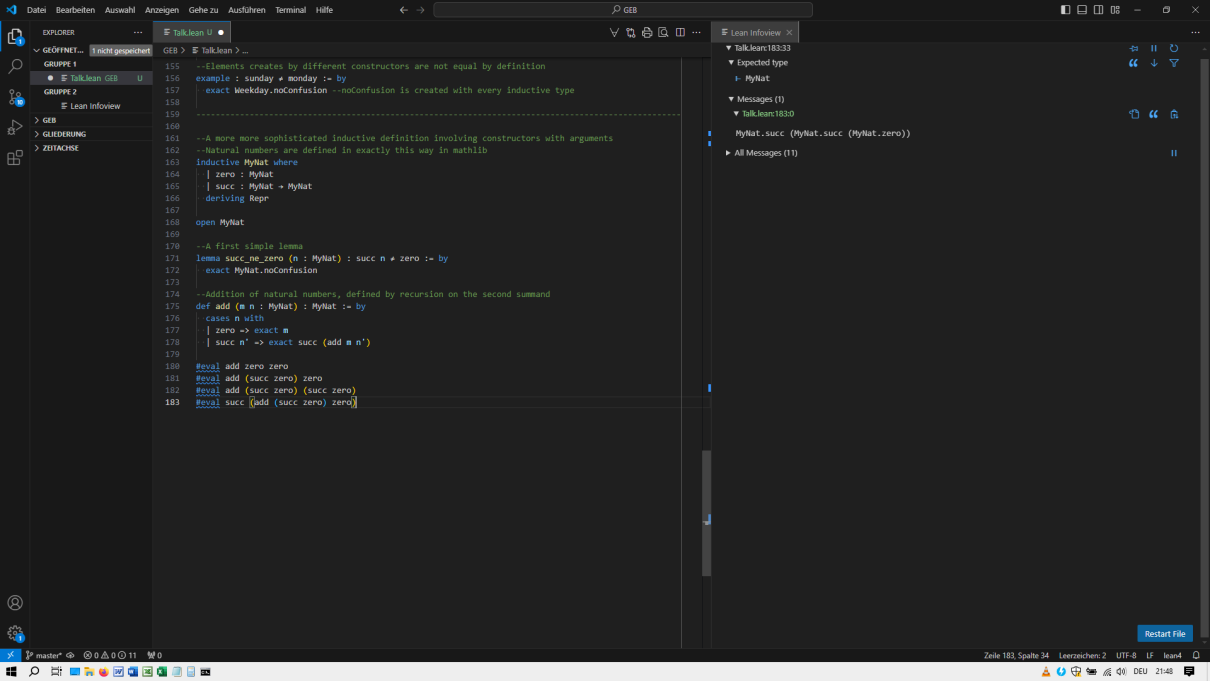

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
Lean Infoview
Talk.lean:173:2
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 173, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:46

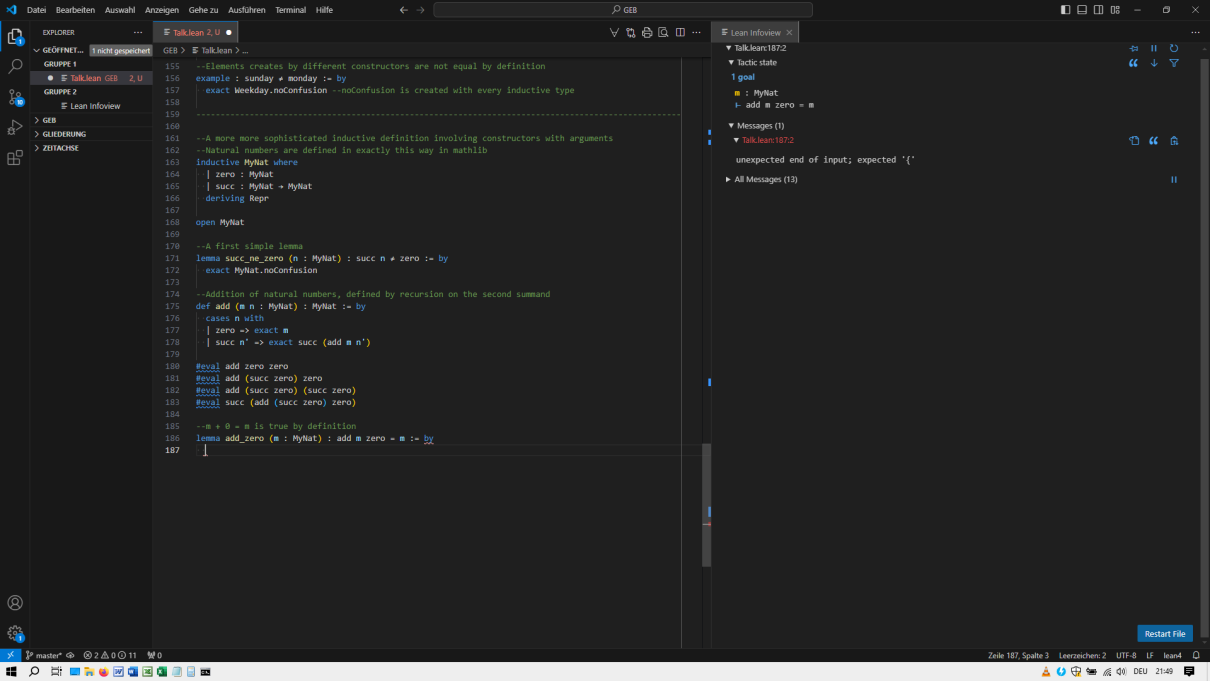

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
Lean Infoview
Talk.lean:179:0
Tactic state
No goals
All Messages (7)
Restart File
master*
Zeile 179, Spalte 1
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:47

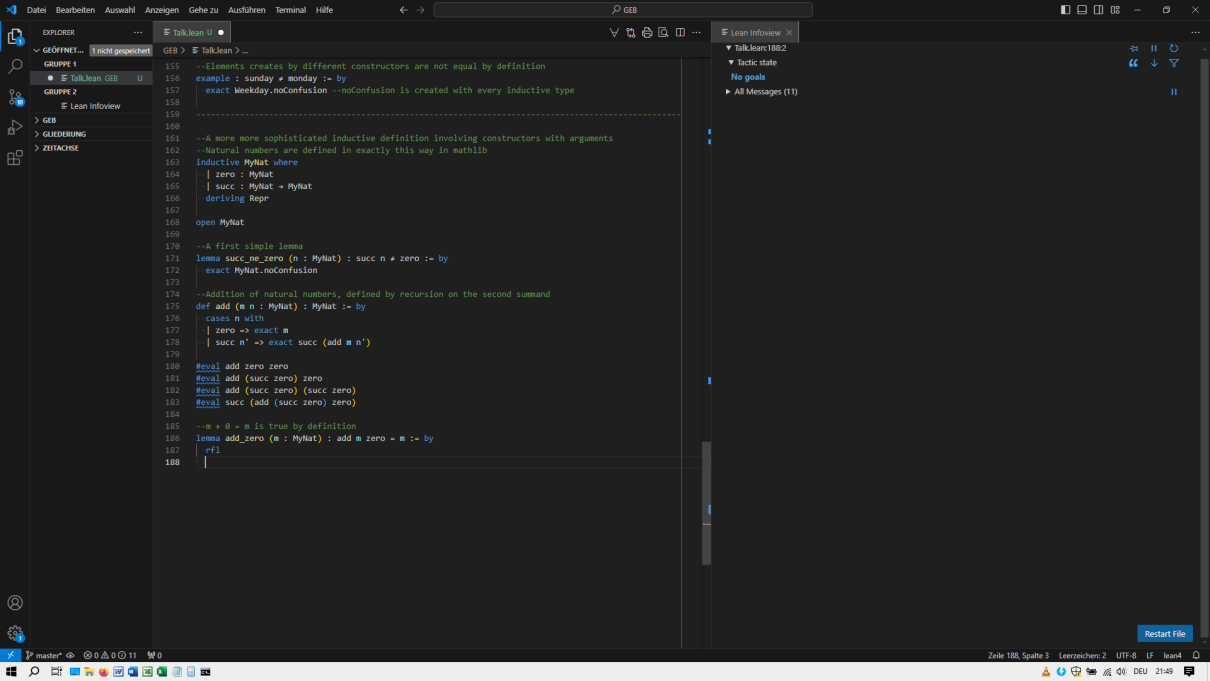

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
Lean Infoview
Talk.lean:180:19
Expected type
MyNat
Messages (1)
Talk.lean:180:0
MyNat.zero
All Messages (8)
Restart File
master*
Zeile 180, Spalte 20 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:48

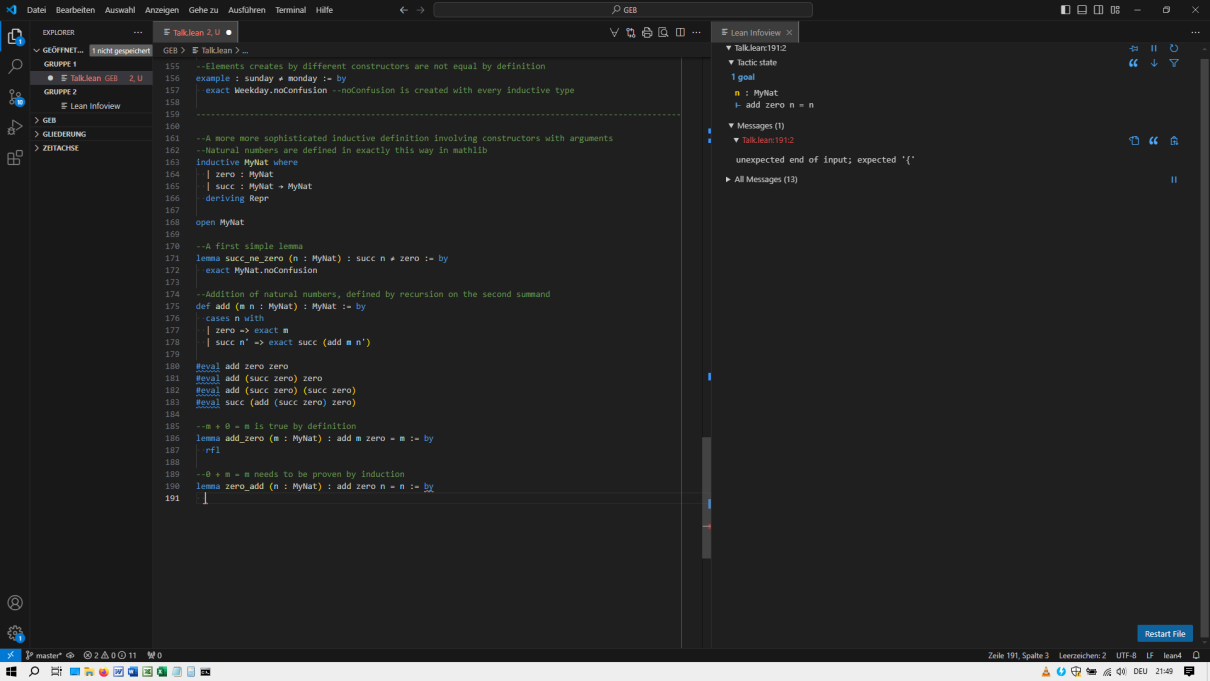

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
155 --Elements creates by different constructors are not equal by definition
156 example : sunday ≠ monday := by
157 exact Weekday.noConfusion --noConfusion is created with every inductive type
158
159 ----------
160
161 --A more more sophisticated inductive definition involving constructors with arguments
162 --Natural numbers are defined in exactly this way in mathlib
163 inductive MyNat where
164 | zero : MyNat
165 | succ : MyNat → MyNat
166 deriving Repr
167
168 open MyNat
169
170 --A first simple lemma
171 lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
172 exact MyNat.noConfusion
173
174 --Addition of natural numbers, defined by recursion on the second summand
175 def add (m n : MyNat) : MyNat := by
176 cases n with
177 | zero => exact m
178 | succ n' => exact succ (add m n')
179
180 #eval add zero zero
181 #eval add (succ zero) zero
Lean Infoview
Talk.lean:181:26
Expected type
⊢ MyNat
Messages (1)
Talk.lean:181:0
MyNat.succ (MyNat.zero)
All Messages (9)
Restart File
master*
0 0 9
0
Zeile 181, Spalte 27 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:48

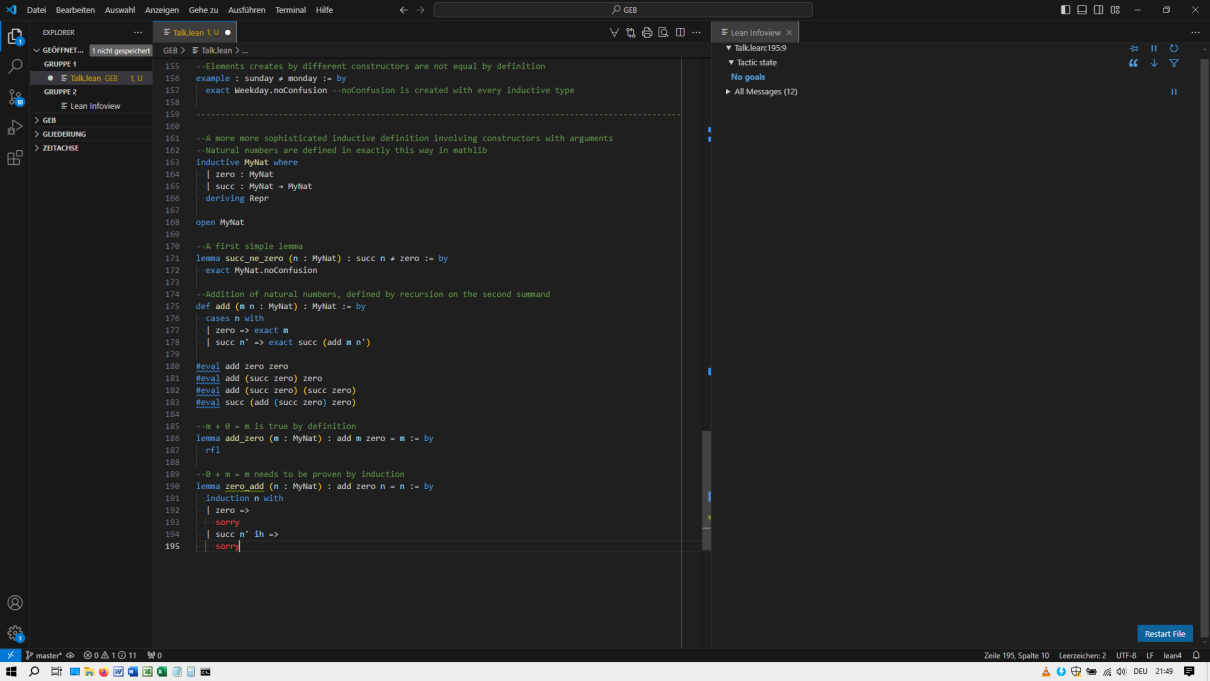

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
155 --Elements creates by different constructors are not equal by definition
156 example : sunday ≠ monday := by
157 exact Weekday.noConfusion --noConfusion is created with every inductive type
158
159 ----------
160
161 --A more more sophisticated inductive definition involving constructors with arguments
162 --Natural numbers are defined in exactly this way in mathlib
163 inductive MyNat where
164 | zero : MyNat
165 | succ : MyNat → MyNat
166 deriving Repr
167
168 open MyNat
169
170 --A first simple lemma
171 lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
172 exact MyNat.noConfusion
173
174 --Addition of natural numbers, defined by recursion on the second summand
175 def add (m n : MyNat) : MyNat := by
176 cases n with
177 | zero => exact m
178 | succ n' => exact succ (add m n')
179
180 #eval add zero zero
181 #eval add (succ zero) zero
182 #eval add (succ zero) (succ zero)
Lean Infoview
Talk.lean:182:33
Expected type
MyNat
Messages (1)
Talk.lean:182:0
MyNat.succ (MyNat.succ (MyNat.zero))
All Messages (10)
Restart File
master*
0 0 10
0
Zeile 182, Spalte 34 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:48

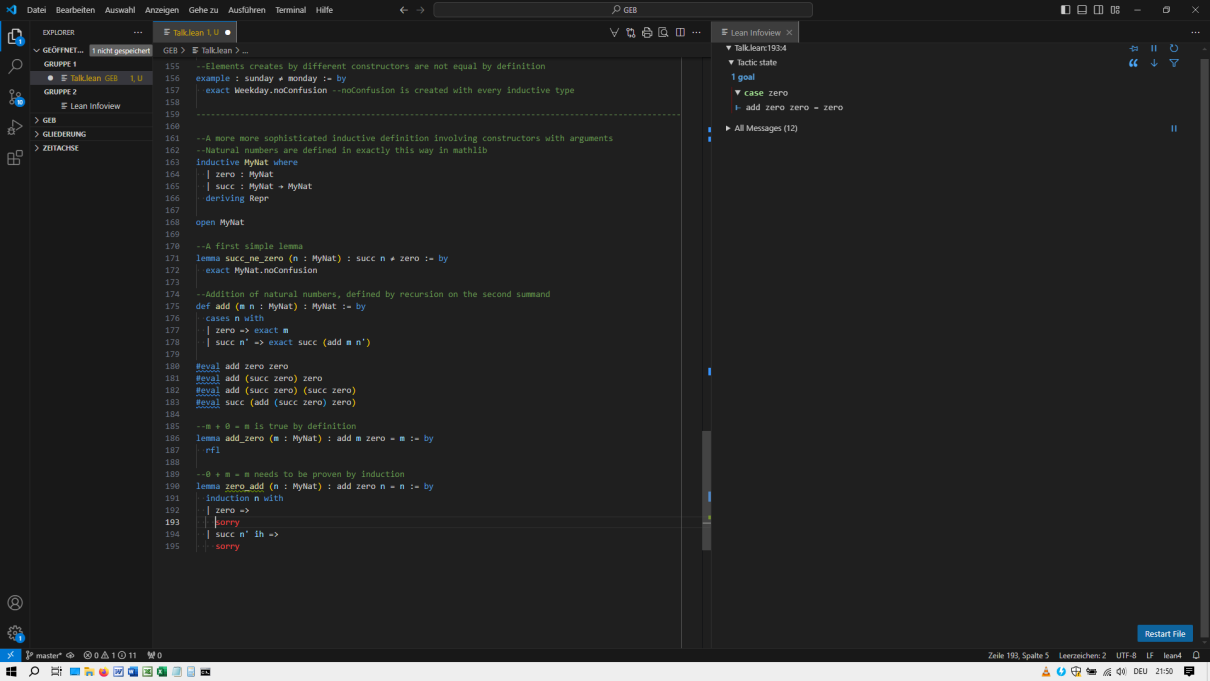

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
155 --Elements creates by different constructors are not equal by definition
156 example : sunday ≠ monday := by
157 exact Weekday.noConfusion --noConfusion is created with every inductive type
158
159 ----------
160
161 --A more more sophisticated inductive definition involving constructors with arguments
162 --Natural numbers are defined in exactly this way in mathlib
163 inductive MyNat where
164 | zero : MyNat
165 | succ : MyNat → MyNat
166 deriving Repr
167
168 open MyNat
169
170 --A first simple lemma
171 lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
172 exact MyNat.noConfusion
173
174 --Addition of natural numbers, defined by recursion on the second summand
175 def add (m n : MyNat) : MyNat := by
176 cases n with
177 | zero => exact m
178 | succ n' => exact succ (add m n')
179
180 #eval add zero zero
181 #eval add (succ zero) zero
182 #eval add (succ zero) (succ zero)
183 #eval succ (add (succ zero) zero)
Lean Infoview
Talk.lean:183:33
Expected type
MyNat
Messages (1)
Talk.lean:183:0
MyNat.succ (MyNat.succ (MyNat.zero))
All Messages (11)
Restart File
master*
0 0 11
0
Zeile 183, Spalte 34 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:48

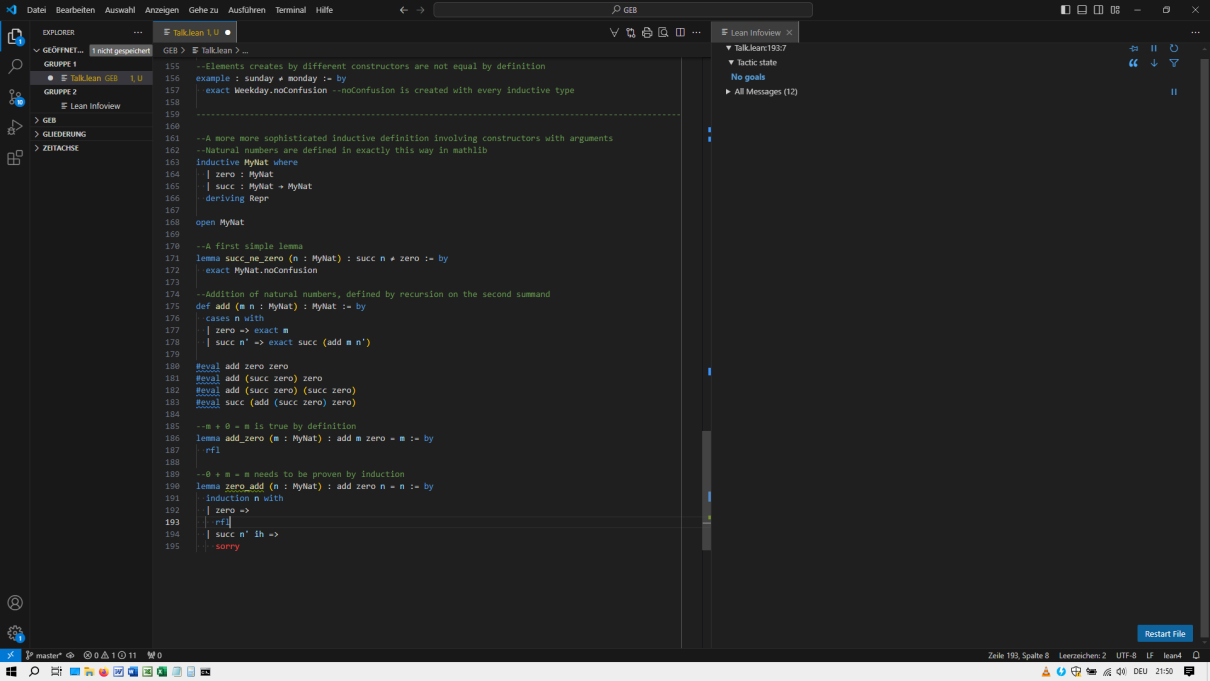

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
Lean Infoview
Talk.lean:187:2
Tactic state
1 goal
m : MyNat
⊢ add m zero = m
Messages (1)
Talk.lean:187:2
unexpected end of input; expected '{'
All Messages (13)
Restart File
master*
Zeile 187, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:49

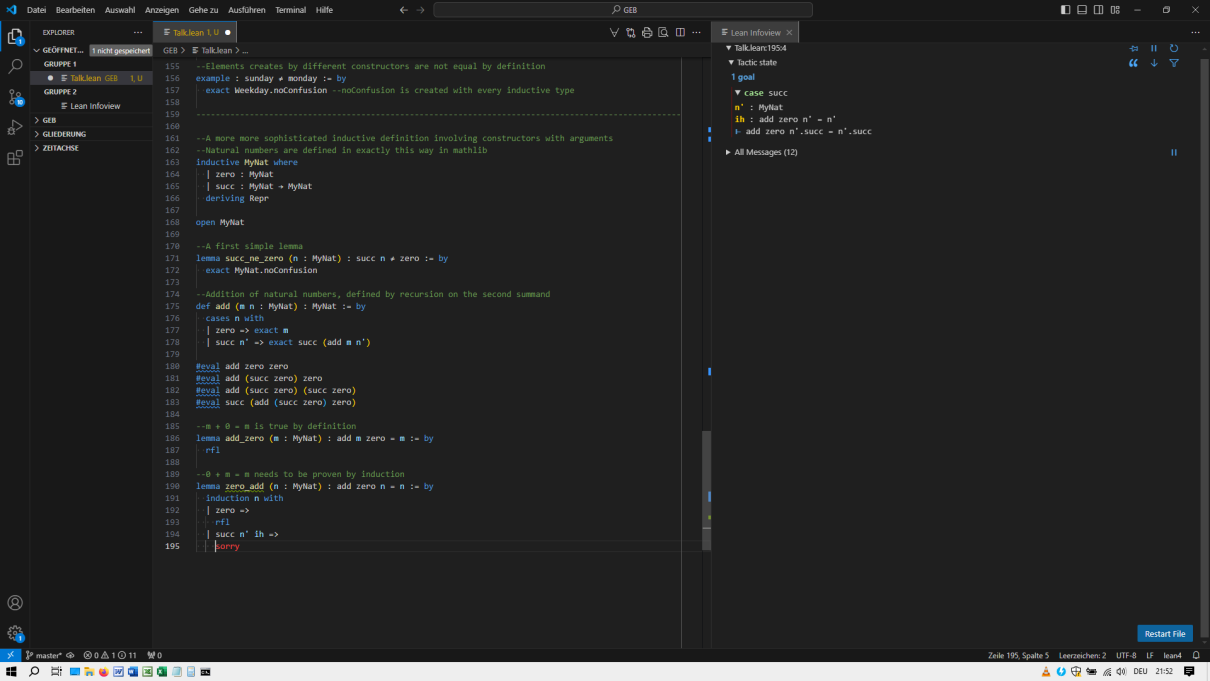

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
Lean Infoview
Talk.lean:188:2
Tactic state
No goals
All Messages (11)
Restart File
master*
Zeile 188, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:49

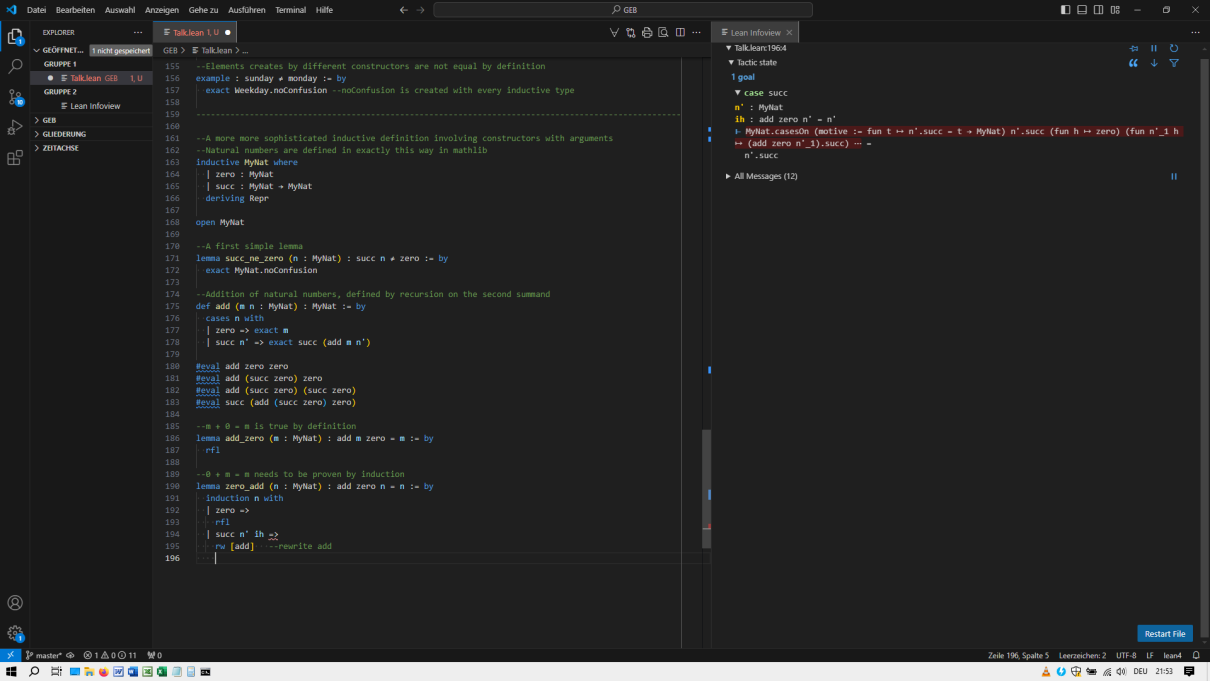

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
Lean Infoview
Talk.lean:191:2
Tactic state
1 goal
n : MyNat
⊢ add zero n = n
Messages (1)
Talk.lean:191:2
unexpected end of input; expected '{'
All Messages (13)
Restart File
master*
2 0 11
0
Zeile 191, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:49

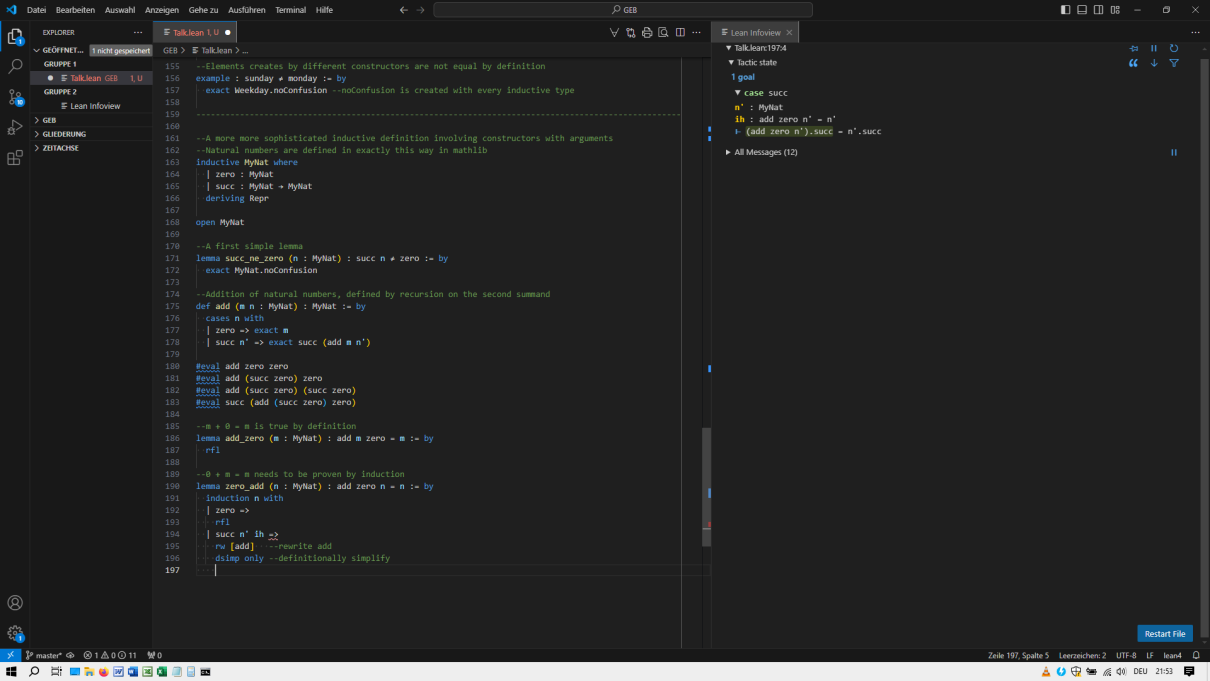

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
sorry
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:195:9
Tactic state
No goals
All Messages (12)
Restart File
master*
Zeile 195, Spalte 10
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:49

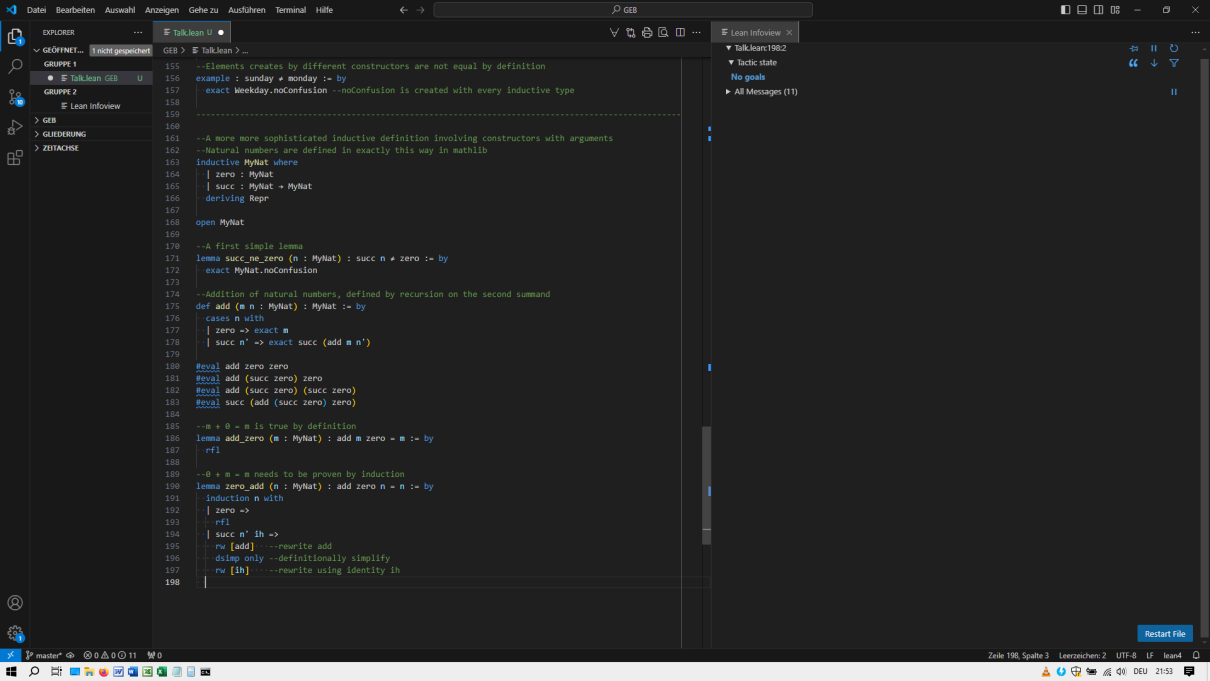

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
sorry
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:193:4
Tactic state
1 goal
case zero
add zero zero = zero
All Messages (12)
Restart File
master*
0 1 11
0
Zeile 193, Spalte 5 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:50

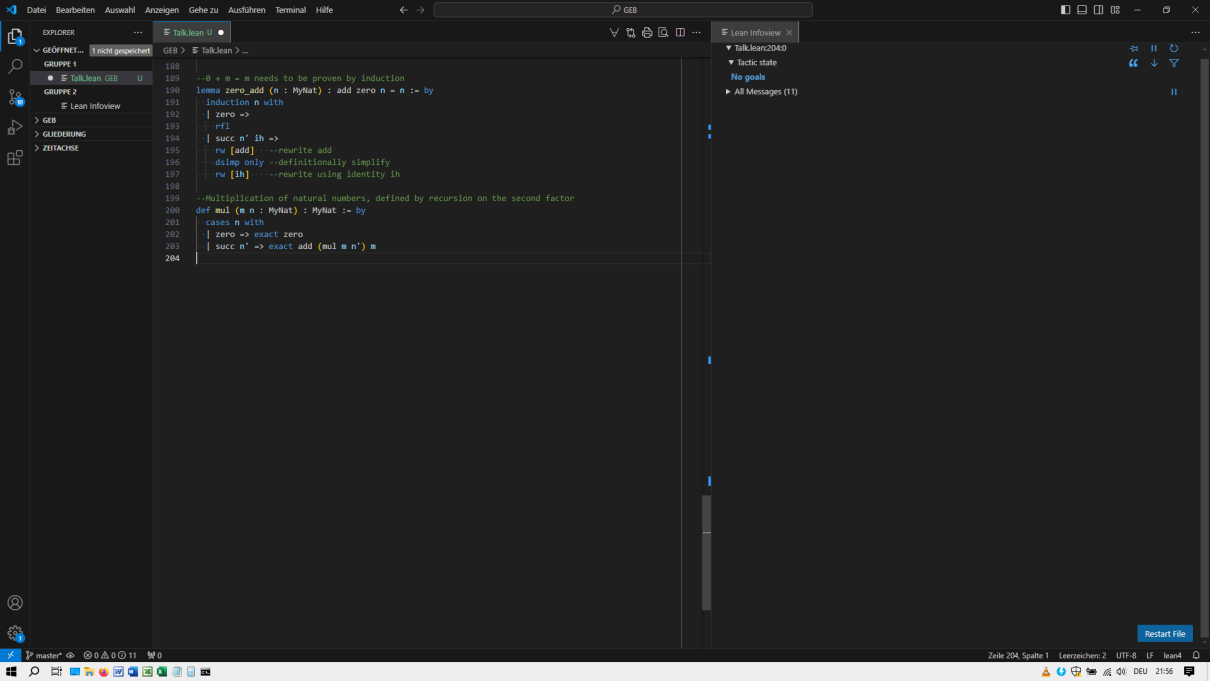

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:193:7
Tactic state
No goals
All Messages (12)
Restart File
master*
0 1 11
0
Zeile 193, Spalte 8
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:50

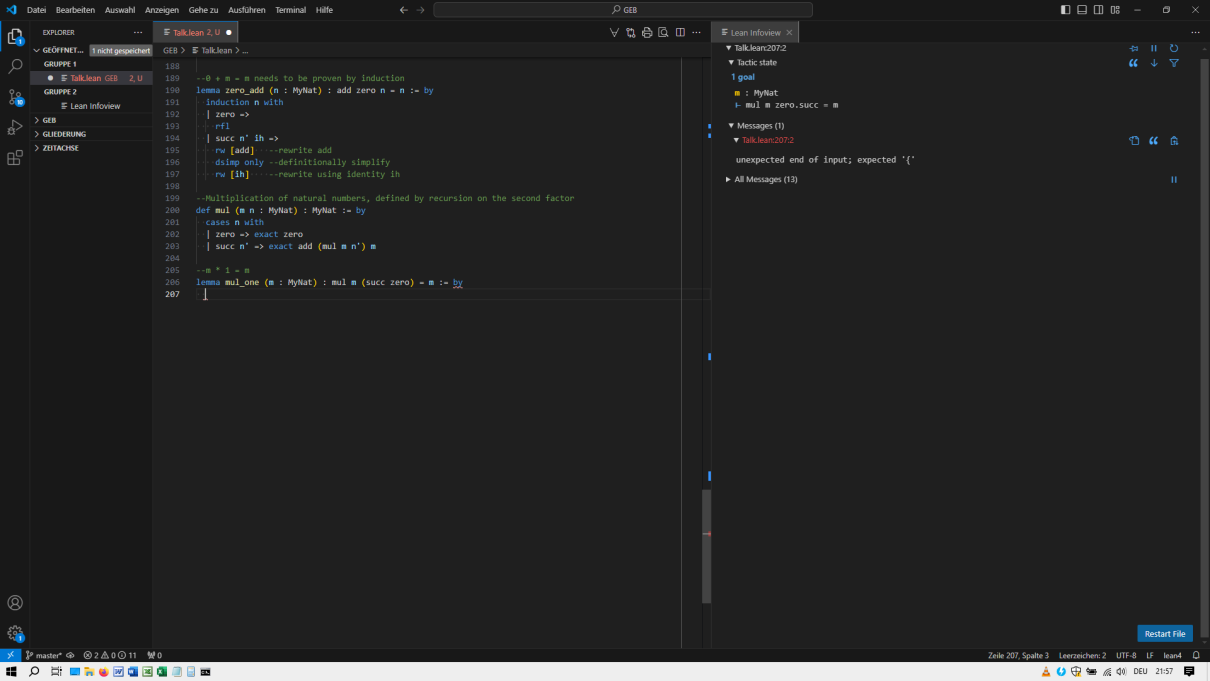

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:195:4
Tactic state
1 goal
case succ
n' : MyNat
ih : add zero n' = n'
⊢ add zero n'.succ = n'.succ
All Messages (12)
Restart File
master*
0 1 11
0
Zeile 195, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:52

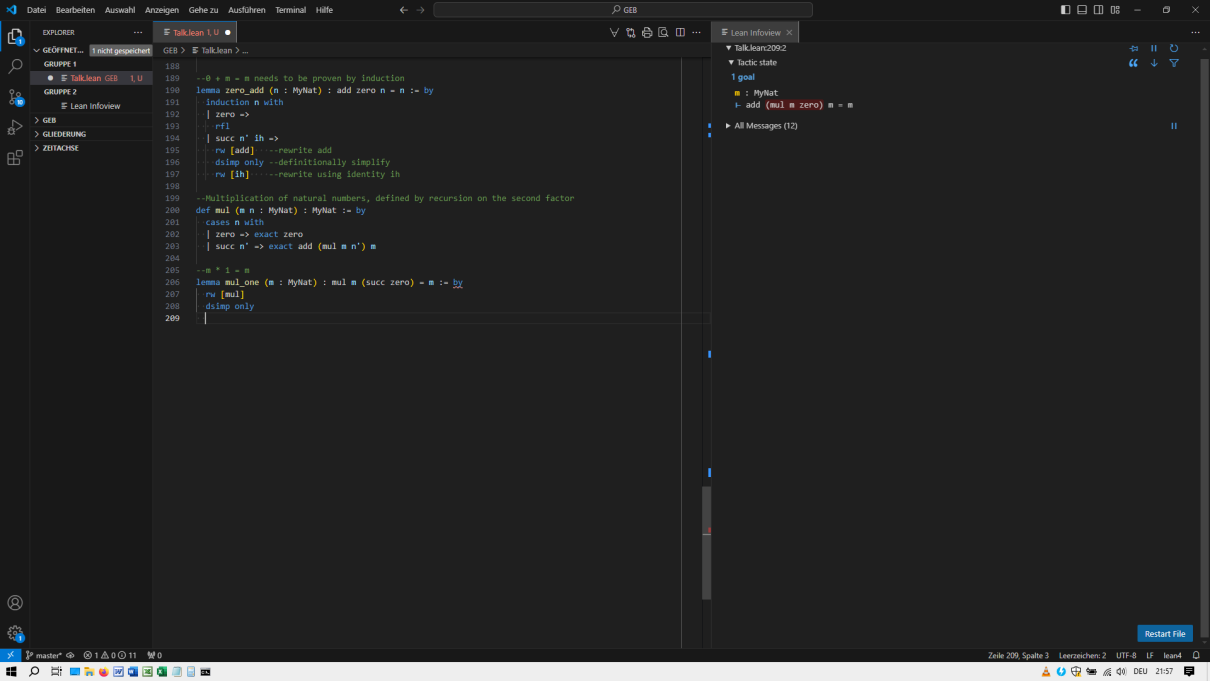

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
Lean Infoview
Talk.lean:196:4
Tactic state
1 goal
case succ
n' : MyNat
ih : add zero n' = n'
⊢ MyNat.casesOn (motive := fun t ↦ n'.succ = t → MyNat) n'.succ (fun h ↦ zero) (fun n'_1 h ↦ (add zero n'_1).succ) ⋯ =
n'.succ
All Messages (12)
Restart File
master*
Zeile 196, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:53

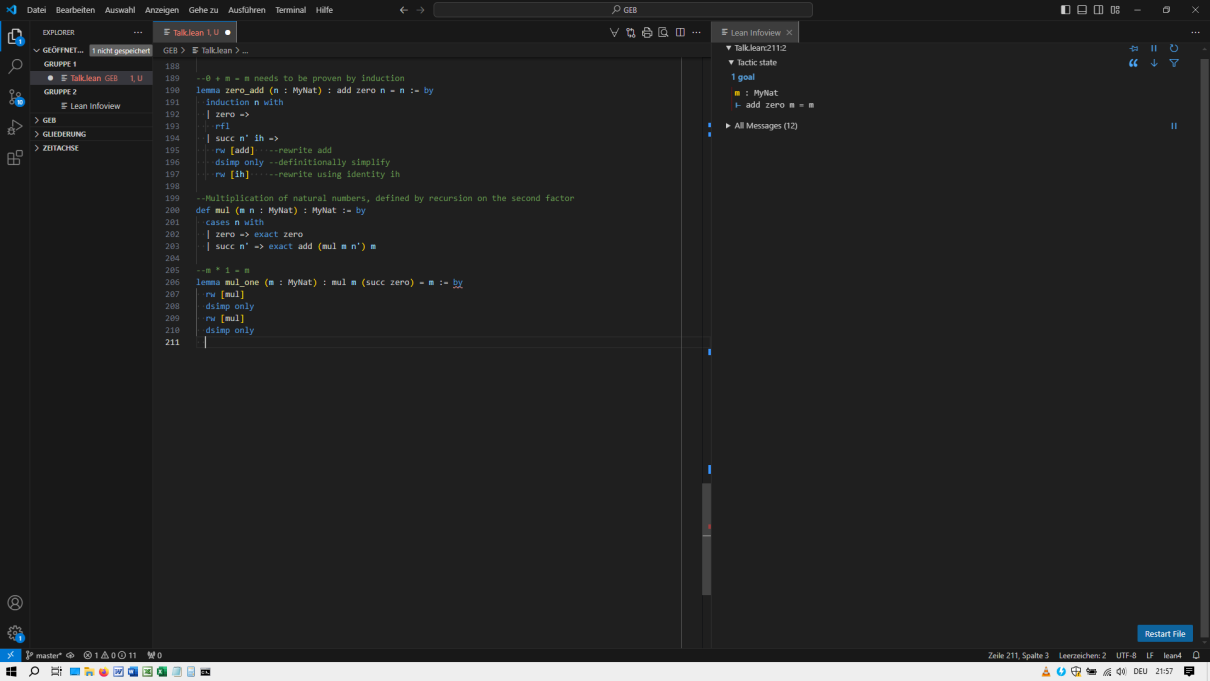

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
Lean Infoview
Talk.lean:197:4
Tactic state
1 goal
case succ
n' : MyNat
ih : add zero n' = n'
⊢ (add zero n').succ = n'.succ
All Messages (12)
Restart File
master*
Zeile 197, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:53

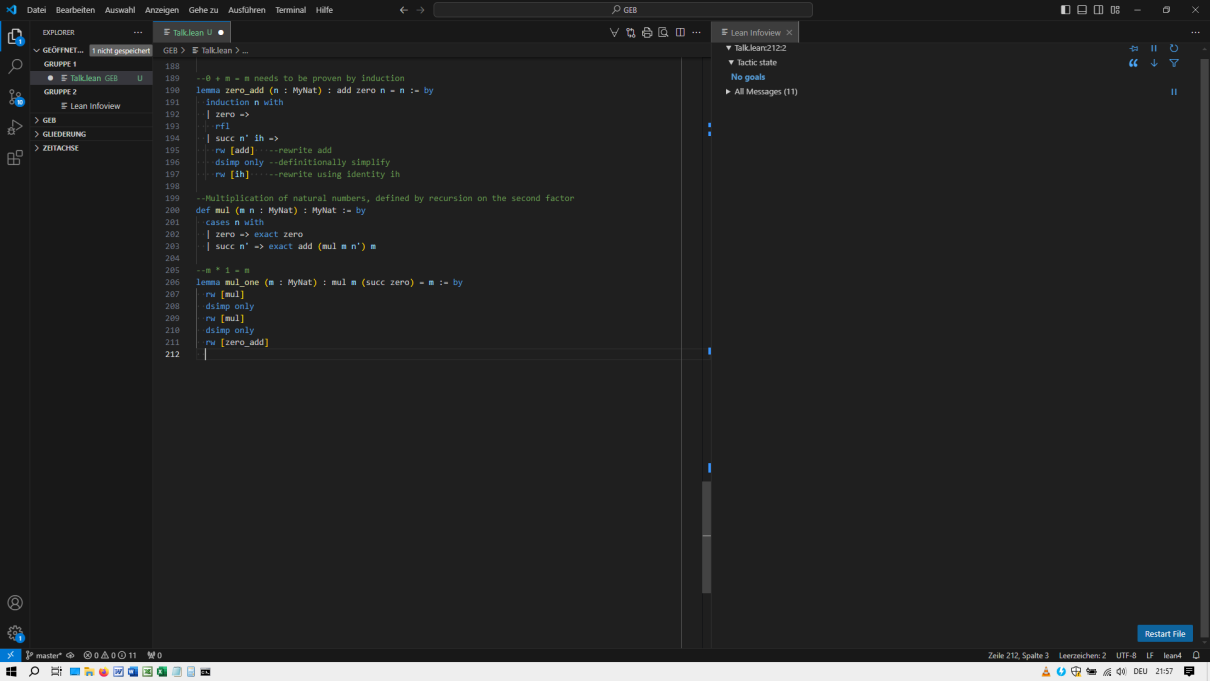

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--Elements creates by different constructors are not equal by definition
example : sunday ≠ monday := by
exact Weekday.noConfusion --noConfusion is created with every inductive type
--A more more sophisticated inductive definition involving constructors with arguments
--Natural numbers are defined in exactly this way in mathlib
inductive MyNat where
| zero : MyNat
| succ : MyNat → MyNat
deriving Repr
open MyNat
--A first simple lemma
lemma succ_ne_zero (n : MyNat) : succ n ≠ zero := by
exact MyNat.noConfusion
--Addition of natural numbers, defined by recursion on the second summand
def add (m n : MyNat) : MyNat := by
cases n with
| zero => exact m
| succ n' => exact succ (add m n')
#eval add zero zero
#eval add (succ zero) zero
#eval add (succ zero) (succ zero)
#eval succ (add (succ zero) zero)
--m + 0 = m is true by definition
lemma add_zero (m : MyNat) : add m zero = m := by
rfl
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] --rewrite using identity ih
Lean Infoview
Talk.lean:198:2
Tactic state
No goals
All Messages (11)
Restart File
master*
0 0 11
0
Zeile 198, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:53

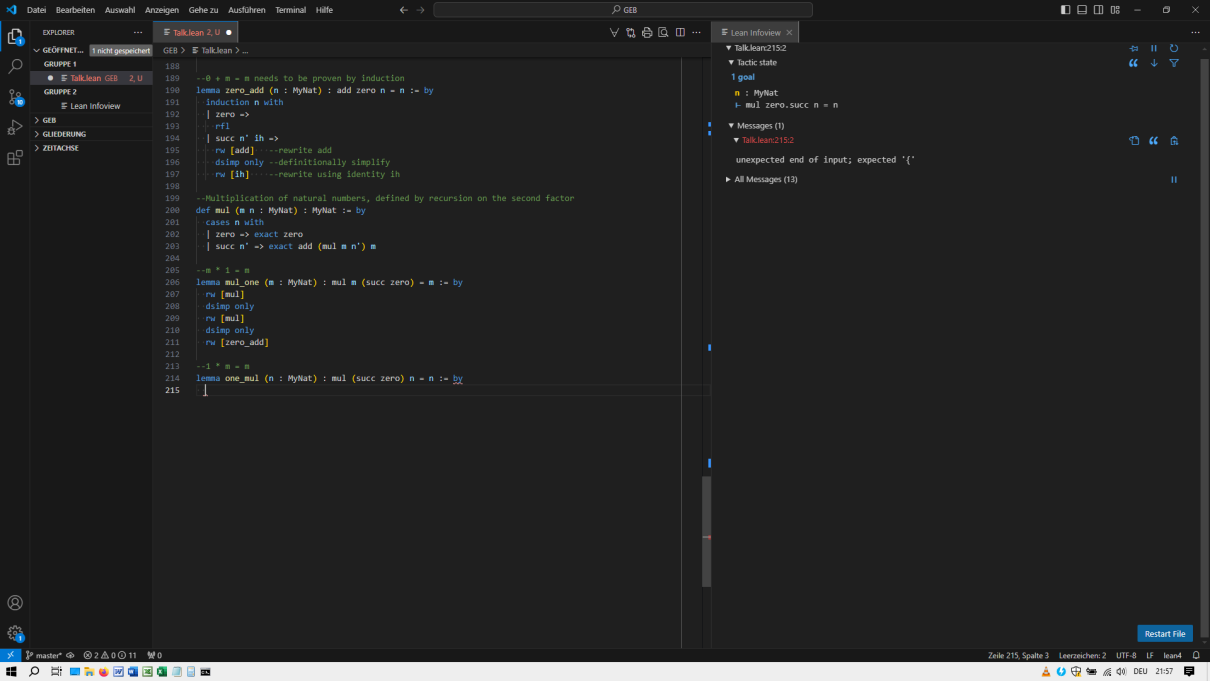

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
188
189 --0 + m = m needs to be proven by induction
190 lemma zero_add (n : MyNat) : add zero n = n := by
191 induction n with
192 | zero =>
193 rfl
194 | succ n' ih =>
195 rw [add] --rewrite add
196 dsimp only --definitionally simplify
197 rw [ih] ----rewrite using identity ih
198
199 --Multiplication of natural numbers, defined by recursion on the second factor
200 def mul (m n : MyNat) : MyNat := by
201 cases n with
202 | zero => exact zero
203 | succ n' => exact add (mul m n') m
204
Lean Infoview
Talk.lean:204:0
Tactic state
No goals
All Messages (11)
Restart File
master*
0 0 11
0
Zeile 204, Spalte 1 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:56

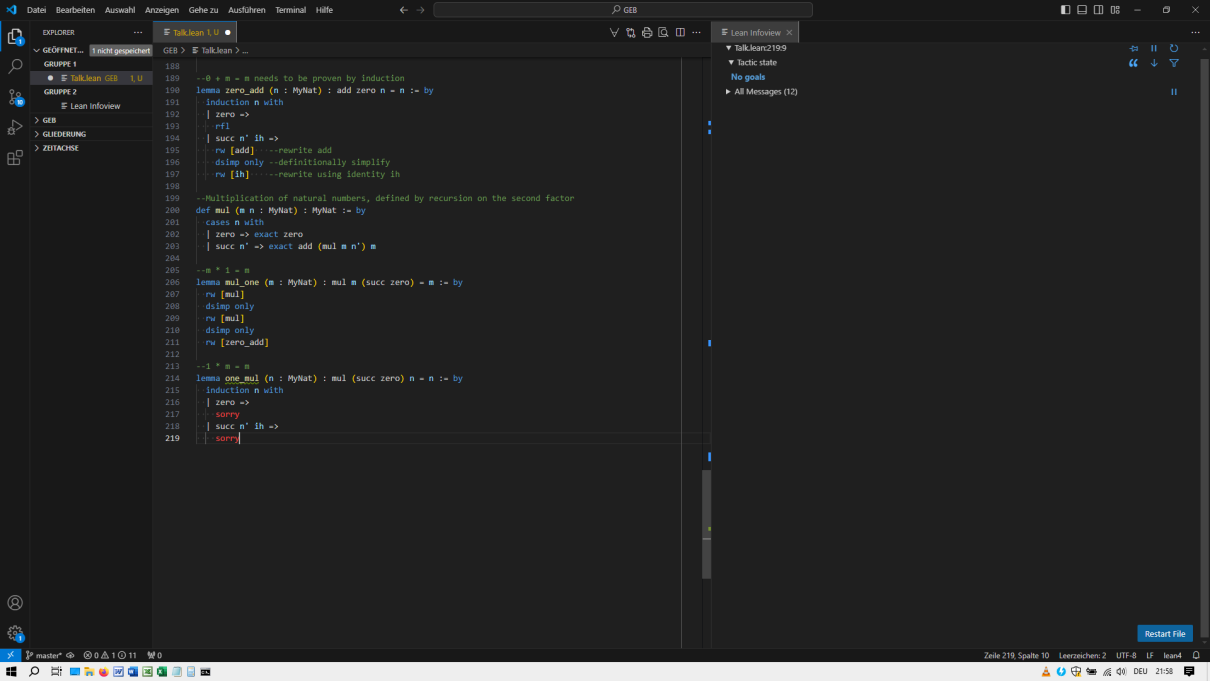

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
188
189 --0 + m = m needs to be proven by induction
190 lemma zero_add (n : MyNat) : add zero n = n := by
191 induction n with
192 | zero =>
193 rfl
194 | succ n' ih =>
195 rw [add] --rewrite add
196 dsimp only --definitionally simplify
197 rw [ih] ----rewrite using identity ih
198
199 --Multiplication of natural numbers, defined by recursion on the second factor
200 def mul (m n : MyNat) : MyNat := by
201 cases n with
202 | zero => exact zero
203 | succ n' => exact add (mul m n') m
204
205 --m * 1 = m
206 lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
207
Lean Infoview
Talk.lean:207:2
Tactic state
1 goal
m : MyNat
⊢ mul m zero.succ = m
Messages (1)
Talk.lean:207:2
unexpected end of input; expected '{'
All Messages (13)
Restart File
master*
2 0 11
0
Zeile 207, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:57

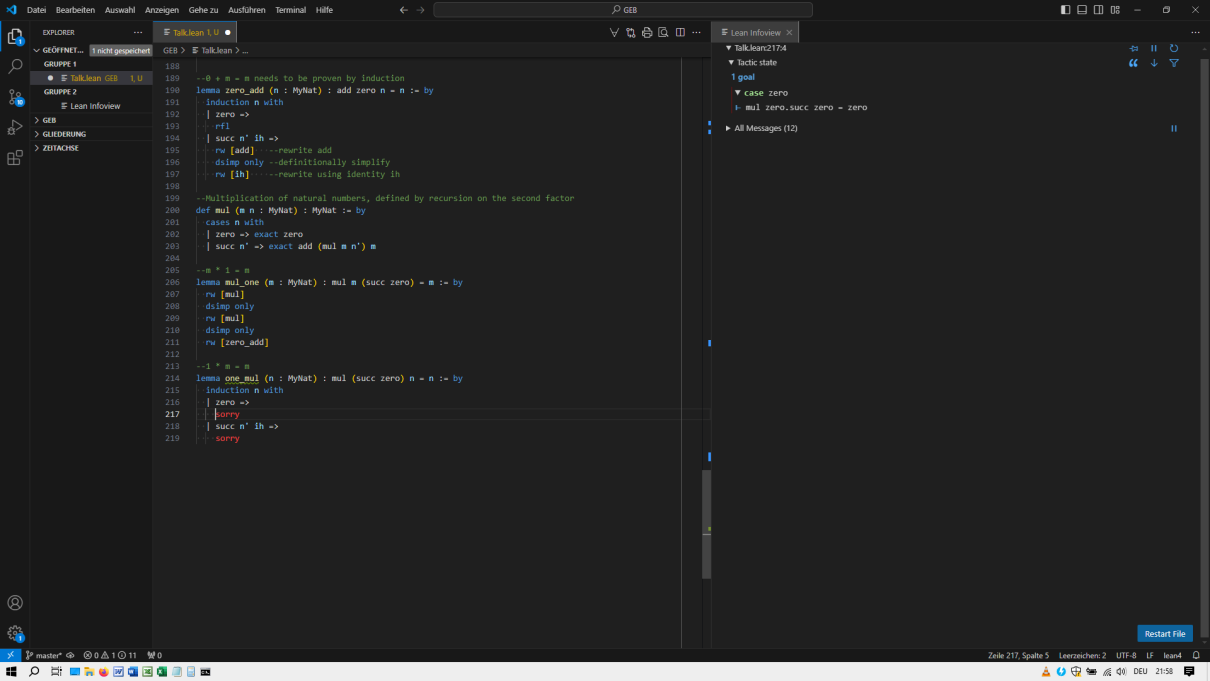

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
188
189 --0 + m = m needs to be proven by induction
190 lemma zero_add (n : MyNat) : add zero n = n := by
191 induction n with
192 | zero =>
193 rfl
194 | succ n' ih =>
195 rw [add] --rewrite add
196 dsimp only --definitionally simplify
197 rw [ih] ----rewrite using identity ih
198
199 --Multiplication of natural numbers, defined by recursion on the second factor
200 def mul (m n : MyNat) : MyNat := by
201 cases n with
202 | zero => exact zero
203 | succ n' => exact add (mul m n') m
204
205 --m * 1 = m
206 lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
207 rw [mul]
208 dsimp only
209
Lean Infoview
Talk.lean:209:2
Tactic state
1 goal
m : MyNat
⊢ add (mul m zero) m = m
All Messages (12)
Restart File
master*
1 0 11
0
Zeile 209, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:57

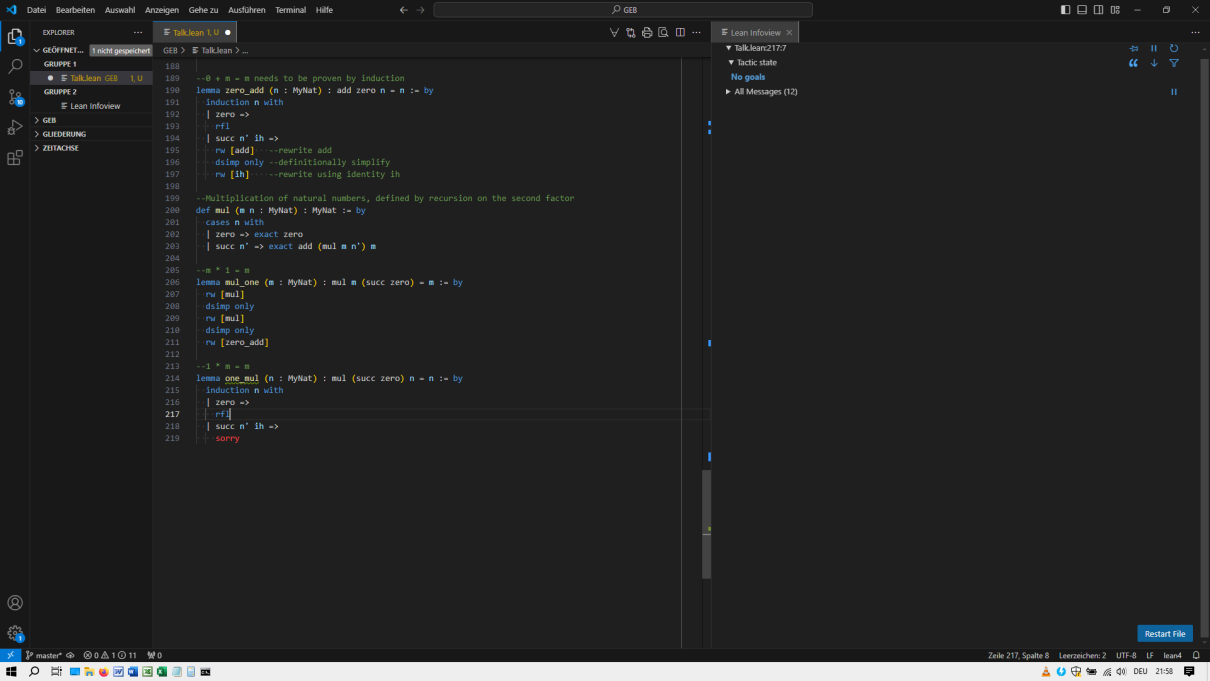

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
188
189 --0 + m = m needs to be proven by induction
190 lemma zero_add (n : MyNat) : add zero n = n := by
191 induction n with
192 | zero =>
193 rfl
194 | succ n' ih =>
195 rw [add] --rewrite add
196 dsimp only --definitionally simplify
197 rw [ih] ----rewrite using identity ih
198
199 --Multiplication of natural numbers, defined by recursion on the second factor
200 def mul (m n : MyNat) : MyNat := by
201 cases n with
202 | zero => exact zero
203 | succ n' => exact add (mul m n') m
204
205 --m * 1 = m
206 lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
207 rw [mul]
208 dsimp only
209 rw [mul]
210 dsimp only
211
Lean Infoview
Talk.lean:211:2
Tactic state
1 goal
m : MyNat
⊢ add zero m = m
All Messages (12)
Restart File
master*
1 0 11
0
Zeile 211, Spalte 3 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:57

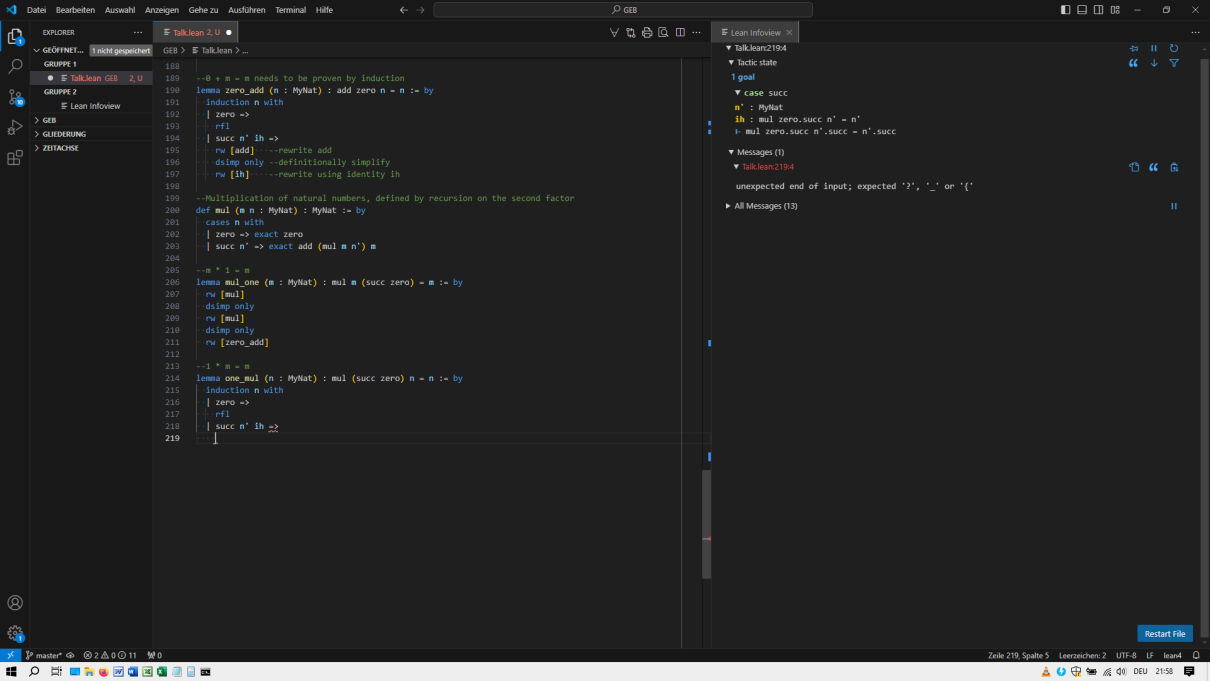

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
Lean Infoview
Talk.lean:212:2
Tactic state
No goals
All Messages (11)
Restart File
master*
Zeile 212, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:57

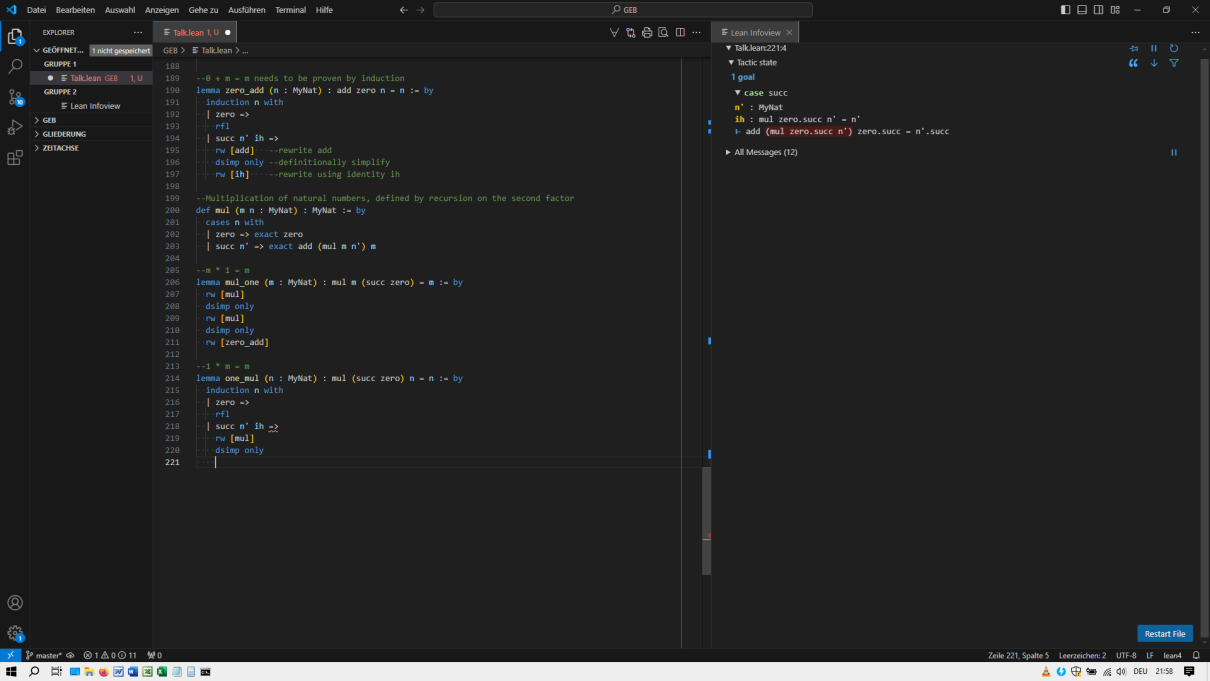

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
Lean Infoview
Talk.lean:215:2
Tactic state
1 goal
n : MyNat
⊢ mul zero.succ n = n
Messages (1)
Talk.lean:215:2
unexpected end of input; expected '{'
All Messages (13)
Restart File
master*
2 0 11
0
Zeile 215, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:57

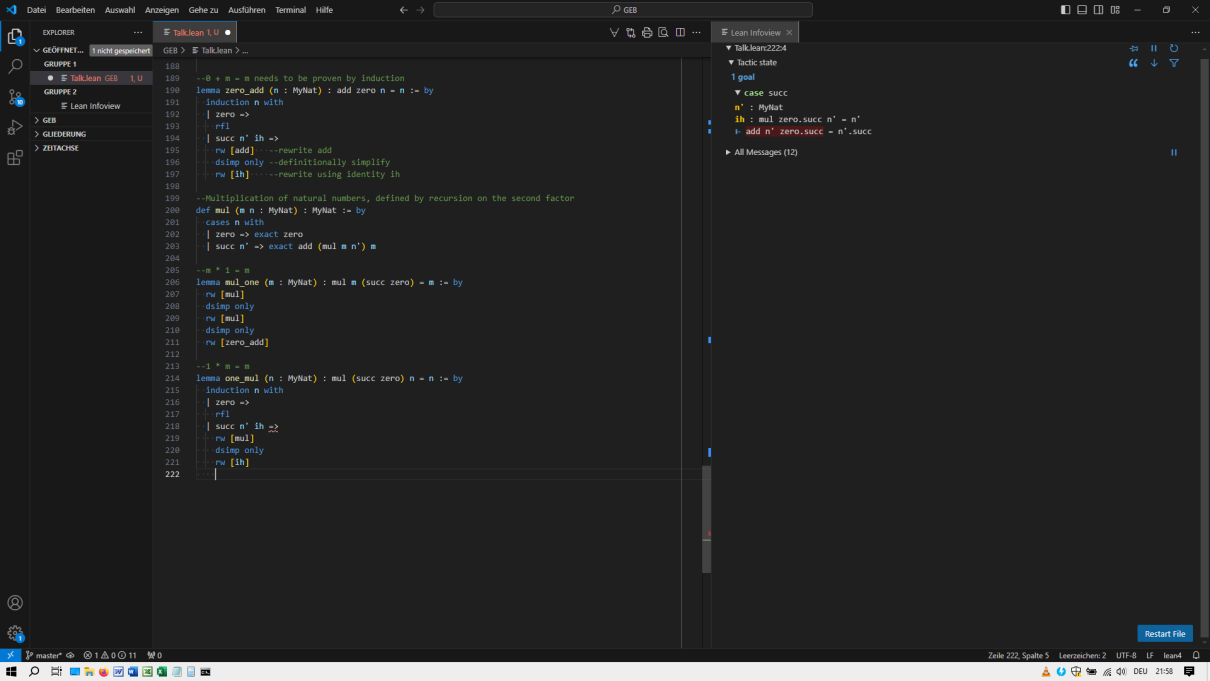

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
sorry
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:219:9
Tactic state
No goals
All Messages (12)
Restart File
master*
Zeile 219, Spalte 10 Leerzeichen: 2 UTF-8 LF lean4
DEU 21:58

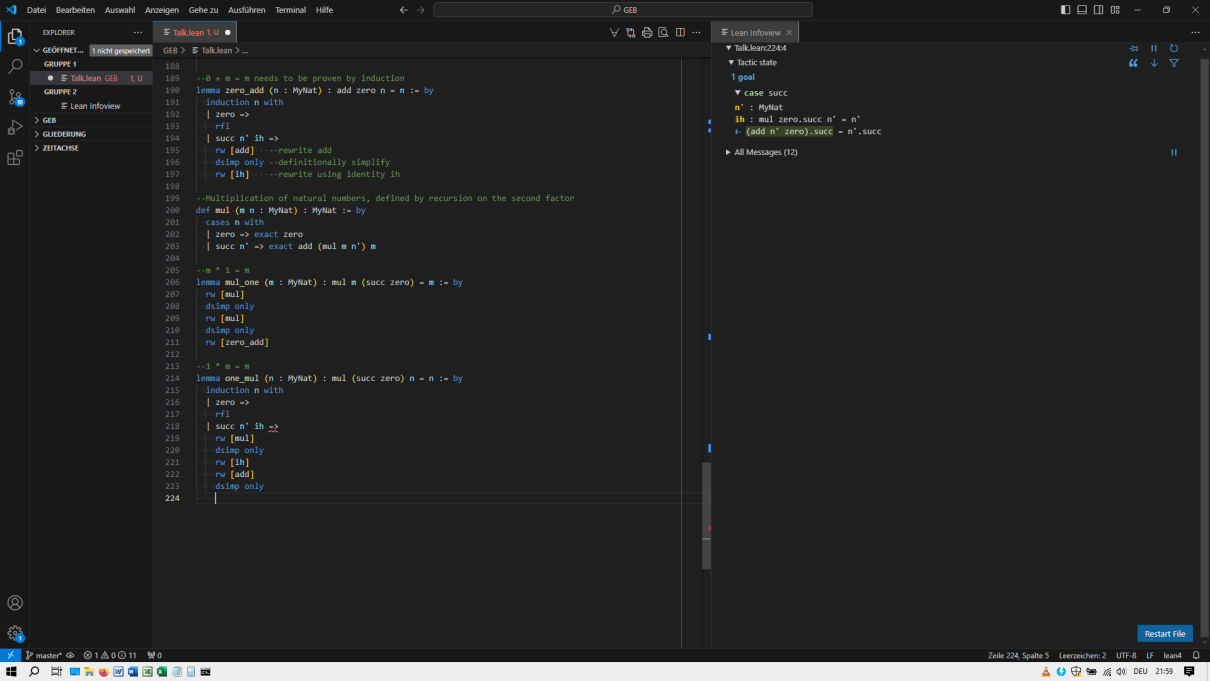

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
sorry
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:217:4
Tactic state
1 goal
case zero
mul zero.succ zero = zero
All Messages (12)
Restart File
master*
0 1 11
0
Zeile 217, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:58

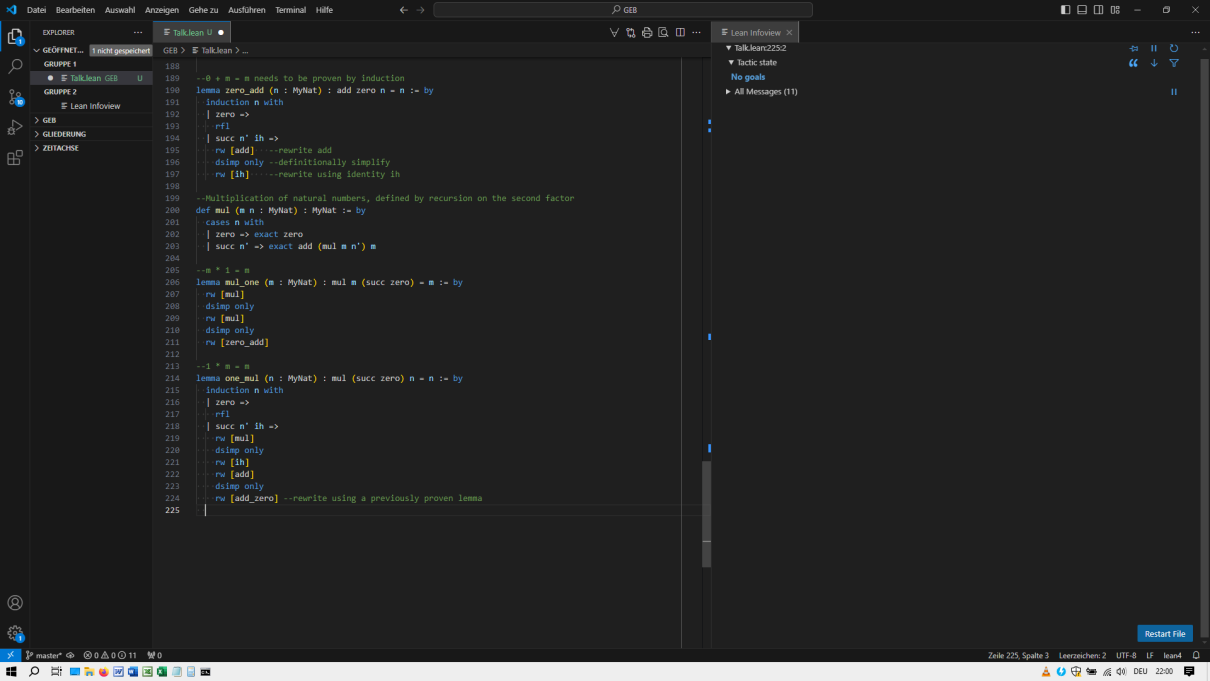

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
sorry
Lean Infoview
Talk.lean:217:7
Tactic state
No goals
All Messages (12)
Restart File
master*
0 1 11
0
Zeile 217, Spalte 8
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:58

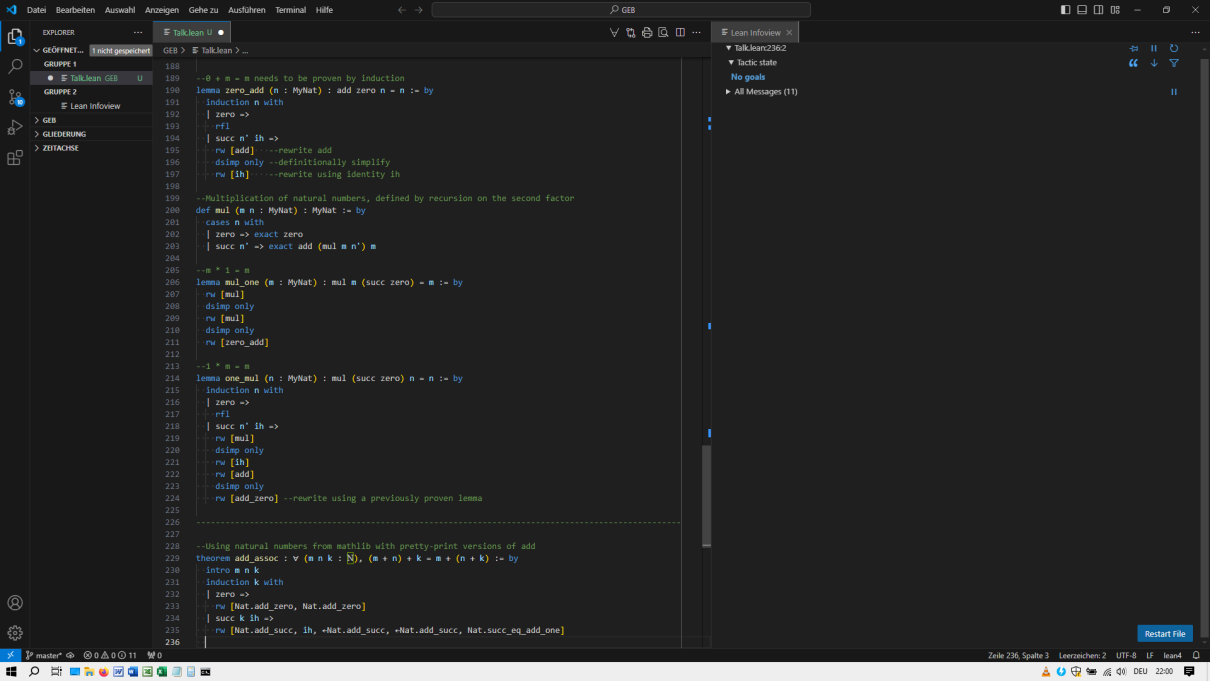

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 2, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 2, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
Lean Infoview
Talk.lean:219:4
Tactic state
1 goal
case succ
n' : MyNat
ih : mul zero.succ n' = n'
⊢ mul zero.succ n'.succ = n'.succ
Messages (1)
Talk.lean:219:4
unexpected end of input; expected '?', '_' or '{'
All Messages (13)
Restart File
master*
Zeile 219, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:58


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [mul]
dsimp only
Lean Infoview
Talk.lean:221:4
Tactic state
1 goal
case succ
n' : MyNat
ih : mul zero.succ n' = n'
⊢ add (mul zero.succ n') zero.succ = n'.succ
All Messages (12)
Restart File
master*
Zeile 221, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:58

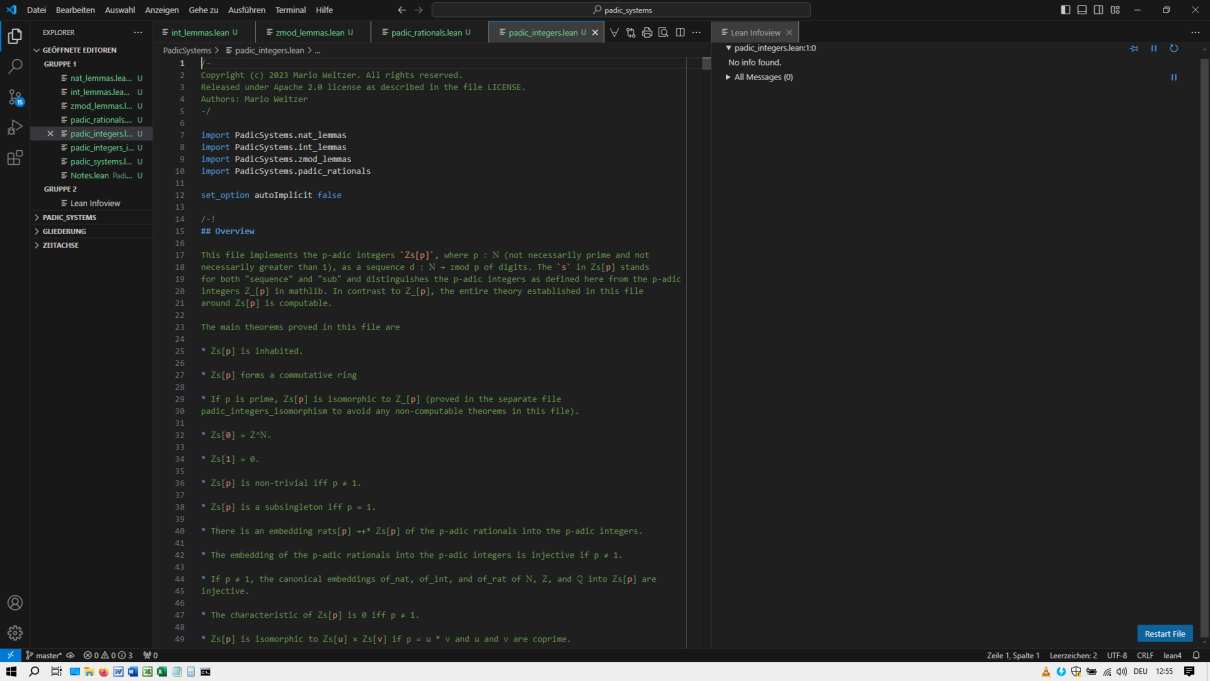

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [mul]
dsimp only
rw [ih]
Lean Infoview
Talk.lean:222:4
Tactic state
1 goal
case succ
n' : MyNat
ih : mul zero.succ n' = n'
⊢ add n' zero.succ = n'.succ
All Messages (12)
Restart File
master*
Zeile 222, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:58


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB 1, U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean 1, U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [mul]
dsimp only
rw [ih]
rw [add]
dsimp only
Lean Infoview
Talk.lean:224:4
Tactic state
1 goal
case succ
n' : MyNat
ih : mul zero.succ n' = n'
⊢ (add n' zero).succ = n'.succ
All Messages (12)
Restart File
master*
1 0 11
0
Zeile 224, Spalte 5
Leerzeichen: 2
UTF-8
LF
lean4
DEU
21:59


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
GEB
EXPLORER
GEÖFFNET... 1 nicht gespeichert
GRUPPE 1
Talk.lean GEB U
GRUPPE 2
Lean Infoview
GEB
GLIEDERUNG
ZEITACHSE
Talk.lean U
GEB > Talk.lean > ...
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [add] --rewrite add
dsimp only --definitionally simplify
rw [ih] ----rewrite using identity ih
--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
cases n with
| zero => exact zero
| succ n' => exact add (mul m n') m
--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
rw [mul]
dsimp only
rw [mul]
dsimp only
rw [zero_add]
--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
induction n with
| zero =>
rfl
| succ n' ih =>
rw [mul]
dsimp only
rw [ih]
rw [add]
dsimp only
rw [add_zero] --rewrite using a previously proven lemma
Lean Infoview
Talk.lean:225:2
Tactic state
No goals
All Messages (11)
Restart File
master*
Zeile 225, Spalte 3
Leerzeichen: 2
UTF-8
LF
lean4
DEU
22:00

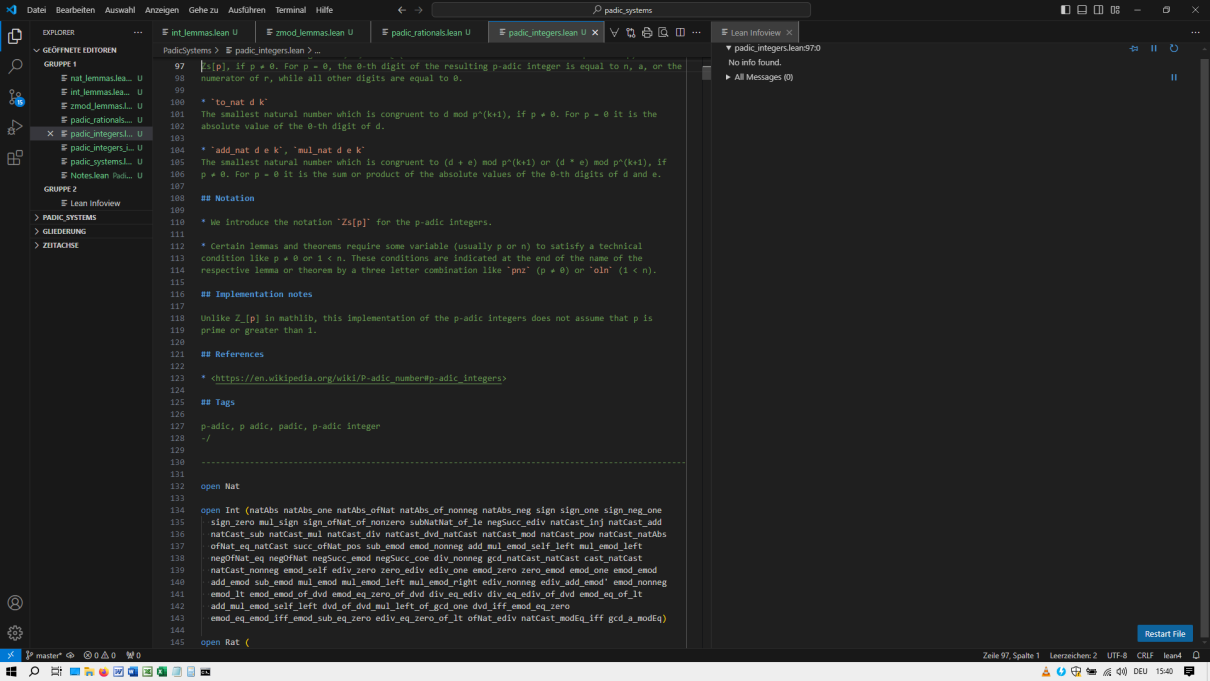


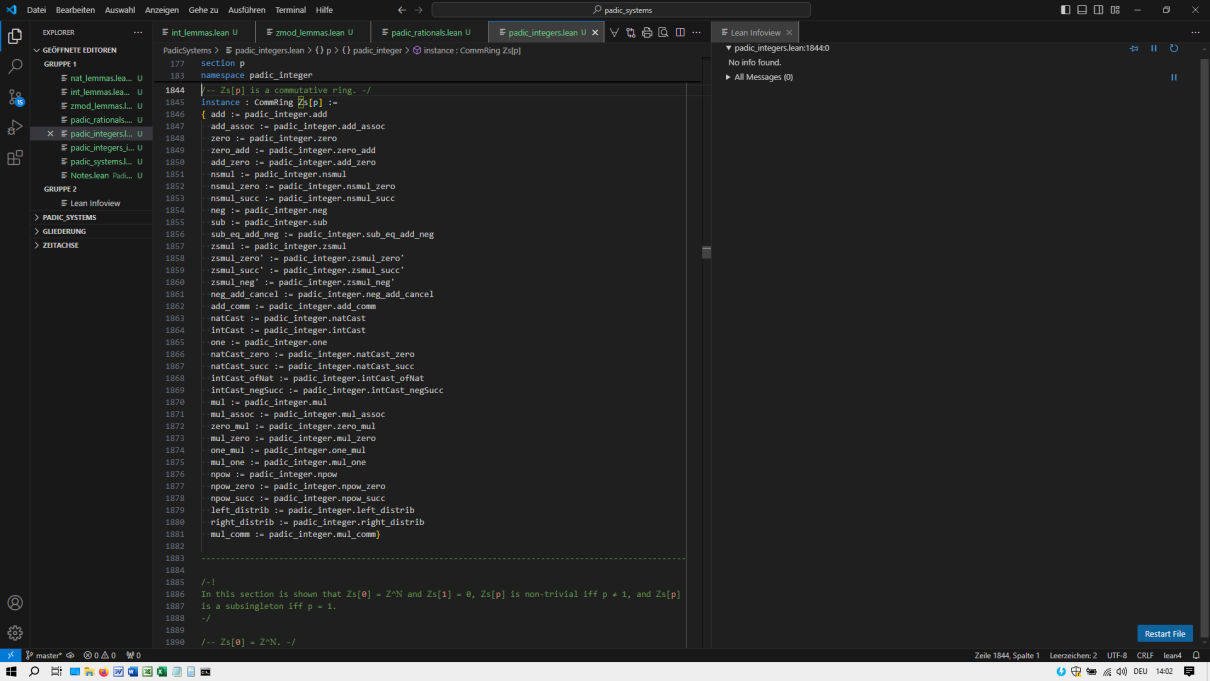

```
--0 + m = m needs to be proven by induction
lemma zero_add (n : MyNat) : add zero n = n := by
  induction n with
  | zero =>
    rfl
  | succ n' ih =>
    rw [add]   --rewrite add
    dsimp only --definitionally simplify
    rw [ih]    ----rewrite using identity ih

--Multiplication of natural numbers, defined by recursion on the second factor
def mul (m n : MyNat) : MyNat := by
  cases n with
  | zero => exact zero
  | succ n' => exact add (mul m n') m

--m * 1 = m
lemma mul_one (m : MyNat) : mul m (succ zero) = m := by
  rw [mul]
  dsimp only
  rw [mul]
  dsimp only
  rw [zero_add]

--1 * m = m
lemma one_mul (n : MyNat) : mul (succ zero) n = n := by
  induction n with
  | zero =>
    rfl
  | succ n' ih =>
    rw [mul]
    dsimp only
    rw [ih]
    rw [add]
    dsimp only
    rw [add_zero] --rewrite using a previously proven lemma

-----------------------------------------------------------------------------------------

--Using natural numbers from mathlib with pretty-print versions of add
theorem add_assoc : ∀ (m n k : ℕ), (m + n) + k = m + (n + k) := by
  intro m n k
  induction k with
  | zero =>
    rw [Nat.add_zero, Nat.add_zero]
  | succ k ih =>
    rw [Nat.add_succ, ih, ←Nat.add_succ, ←Nat.add_succ, Nat.succ_eq_add_one]

```

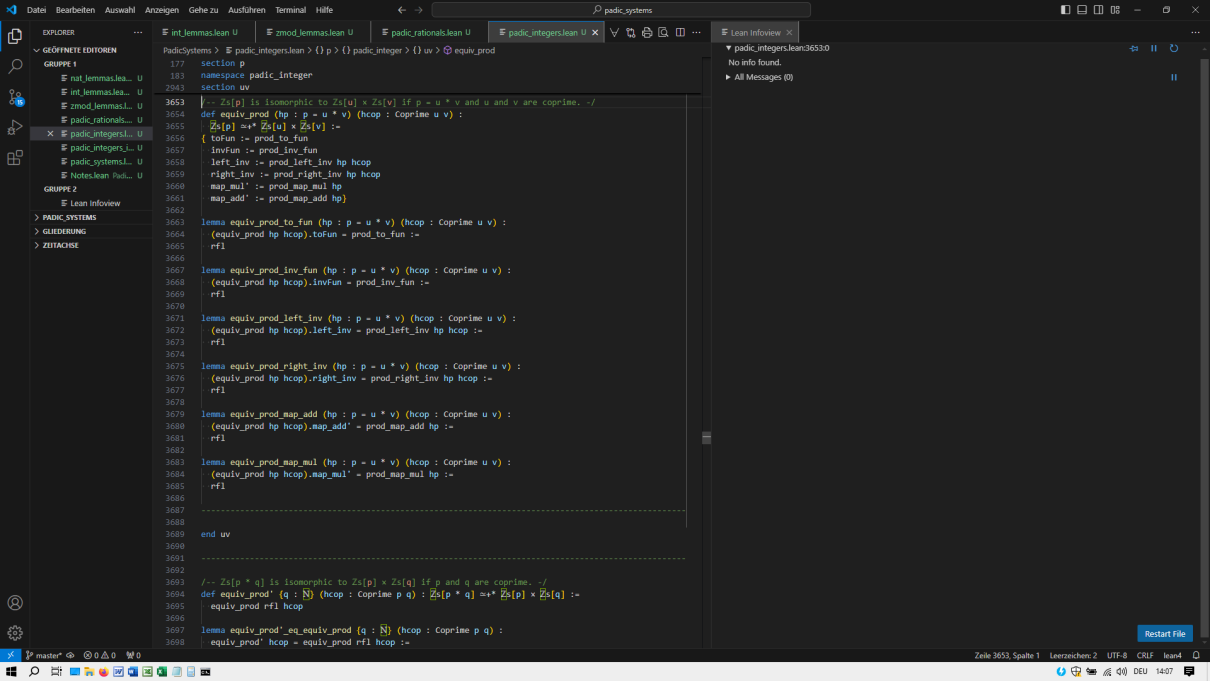

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > ...
padic_integers.lean:1:0
No info found.
All Messages (0)
1 /-
2 Copyright (c) 2023 Mario Weitzer. All rights reserved.
3 Released under Apache 2.0 license as described in the file LICENSE.
4 Authors: Mario Weitzer
5 -/
6
7 import PadicSystems.nat_lemmas
8 import PadicSystems.int_lemmas
9 import PadicSystems.zmod_lemmas
10 import PadicSystems.padic_rationals
11
12 set_option autoImplicit false
13
14 /-!
15 ## Overview
16
17 This file implements the p-adic integers `Zs[p]`, where p : ℕ (not necessarily prime and not
18 necessarily greater than 1), as a sequence d : ℕ → zmod p of digits. The `s` in Zs[p] stands
19 for both "sequence" and "sub" and distinguishes the p-adic integers as defined here from the p-adic
20 integers Z_[p] in mathlib. In contrast to Z_[p], the entire theory established in this file
21 around Zs[p] is computable.
22
23 The main theorems proved in this file are
24
25 * Zs[p] is inhabited.
26
27 * Zs[p] forms a commutative ring
28
29 * If p is prime, Zs[p] is isomorphic to Z_[p] (proved in the separate file
30 padic_integers_isomorphism to avoid any non-computable theorems in this file).
31
32 * Zs[0] ≃ ℤ^ℕ.
33
34 * Zs[1] ≃ 0.
35
36 * Zs[p] is non-trivial iff p ≠ 1.
37
38 * Zs[p] is a subsingleton iff p = 1.
39
40 * There is an embedding rats[p] →+* Zs[p] of the p-adic rationals into the p-adic integers.
41
42 * The embedding of the p-adic rationals into the p-adic integers is injective if p ≠ 1.
43
44 * If p ≠ 1, the canonical embeddings of_nat, of_int, and of_rat of ℕ, ℤ, and ℚ into Zs[p] are
45 injective.
46
47 * The characteristic of Zs[p] is 0 iff p ≠ 1.
48
49 * Zs[p] is isomorphic to Zs[u] × Zs[v] if p = u * v and u and v are coprime.
Restart File
master*
Zeile 1, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 12:55

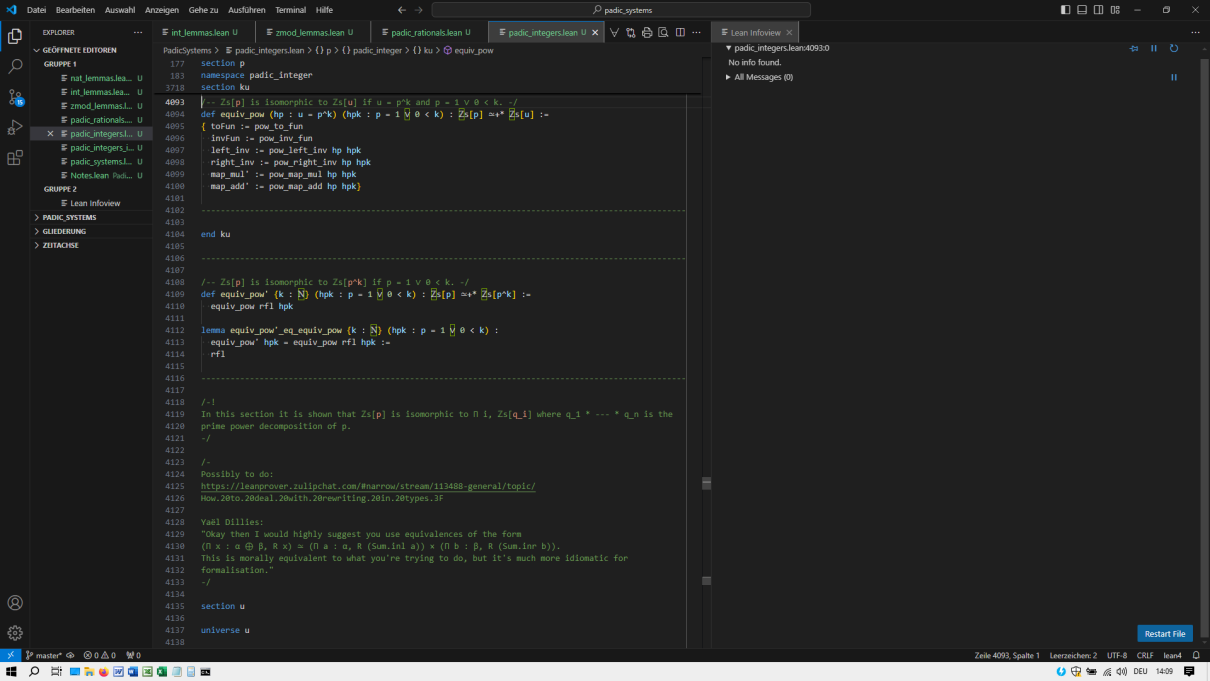
```
* Zs[p] is isomorphic to Zs[u] × Zs[v] if p = u * v and u and v are coprime.

* Zs[p] is isomorphic to Zs[u] if u = p^k and p = 1 ∨ 0 < k.

* Zs[p] is isomorphic to Π i, Zs[getD l i 1] if l = prime_pow_decomp p, the prime power
decomposition of p.

* d * d⁻¹ = 1 for a p-adic integer d iff the 0-th digit of d and p are coprime, if p ≠ 0.
  d * d⁻¹ = 1 for a p-adic integer d iff the digits of d are equal to 1 or -1, if p = 0.

* A p-adic integer d has an inverse iff the 0-th digit of d and p are coprime, if p ≠ 0.
  A p-adic integer d has an inverse iff the digits of d are equal to 1 or -1, if p = 0.

* d * e = 1 for p-adic integers d and e iff the 0-th digit of d and p are coprime and e = d⁻¹,
  if p ≠ 0.
  d * e = 1 for p-adic integers d and e iff the digits of d are equal to 1 or -1 and e = d⁻¹,
  if p = 0.
* d * e = 1 for p-adic integers d and e iff the 0-th digit of e and p are coprime and d = e⁻¹,
  if p ≠ 0.
  d * e = 1 for p-adic integers d and e iff the digits of e are equal to 1 or -1 and d = e⁻¹,
  if p = 0.

* Zs[p] is a domain iff p is a proper prime power.

* Zs[p] has no zero diviors iff p is a prime power (including the case p = 1).
mkRat_eq_divInt
* A p-adic integer d is a zero divisor iff

  * there is a q dividing p with q ≠ 1 and q and p / q coprime such that the projection of d to
  Zs[q] (where Zs[p] ≃+* Zs[q] × Zs[p / q]) is equal to 0, if 1 < p, or

  * d k = 0 for some k, if p = 0.

  * Since Zs[1] is trivial, it doesn't have any zero divisors (not even 0).

## Important definitions

* `padic_integer p`
The type of p-adic integers.

* `digit_nat n k`, `digit_int a k`, `digit_rat r k`
The k-th digit of a natural number n, an integer a, or a rational number r whose denominator is
coprime to p, as a natural number (digit_nat) or integer (digit_int and digit_rat) between 0 and
p - 1, if p ≠ 0. For p = 0 it is equal to n, a, or the numerator of r if k = 0 and equal to 0
otherwise.

* `of_nat n`, `of_int a`, `of_rat r`
The canonical embeddings of ℕ, ℤ, and ℚ (for fractions whose denominator is coprime to p) into
Zs[p], if p ≠ 0. For p = 0, the 0-th digit of the resulting p-adic integer is equal to n, a, or the
```

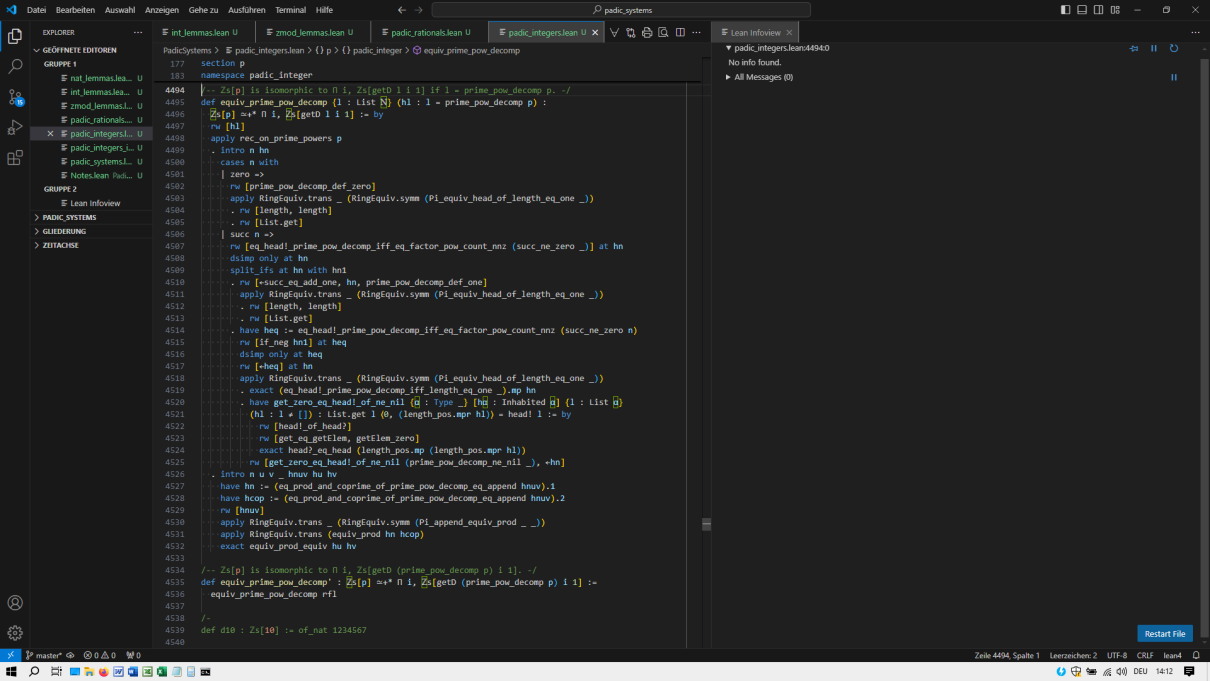
No info found.

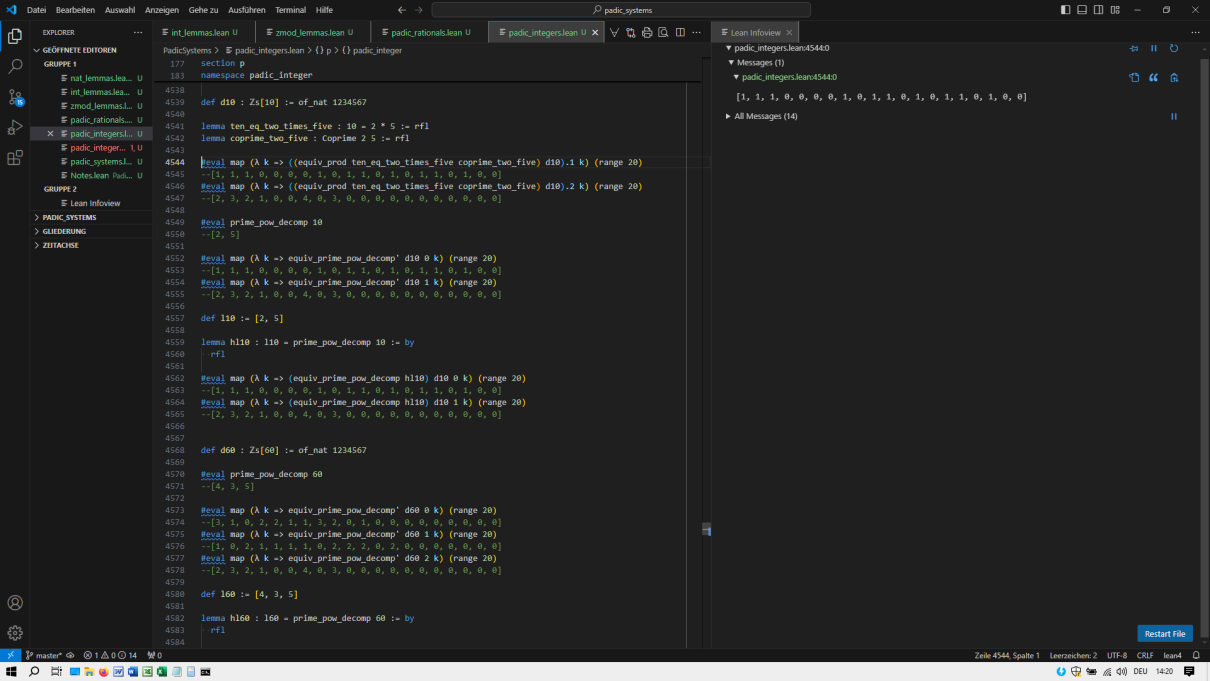

EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padic... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > ...
97 Z_[p], if p ≠ 0. For p = 0, the 0-th digit of the resulting p-adic integer is equal to n, a, or the
98 numerator of r, while all other digits are equal to 0.
99
100 * `to_nat d k`
101 The smallest natural number which is congruent to d mod p^(k+1), if p ≠ 0. For p = 0 it is the
102 absolute value of the 0-th digit of d.
103
104 * `add_nat d e k`, `mul_nat d e k`
105 The smallest natural number which is congruent to (d + e) mod p^(k+1) or (d * e) mod p^(k+1), if
106 p ≠ 0. For p = 0 it is the sum or product of the absolute values of the 0-th digits of d and e.
107
108 ## Notation
109
110 * We introduce the notation `Zs[p]` for the p-adic integers.
111
112 * Certain lemmas and theorems require some variable (usually p or n) to satisfy a technical
113 condition like p ≠ 0 or 1 < n. These conditions are indicated at the end of the name of the
114 respective lemma or theorem by a three letter combination like `pnz` (p ≠ 0) or `oln` (1 < n).
115
116 ## Implementation notes
117
118 Unlike Z_[p] in mathlib, this implementation of the p-adic integers does not assume that p is
119 prime or greater than 1.
120
121 ## References
122
123 * <https://en.wikipedia.org/wiki/P-adic_number#p-adic_integers>
124
125 ## Tags
126
127 p-adic, p adic, padic, p-adic integer
128 -/
129
130 --
131
132 open Nat
133
134 open Int (natAbs natAbs_one natAbs_ofNat natAbs_of_nonneg natAbs_neg sign sign_one sign_neg_one
135 sign_zero mul_sign sign_ofNat_of_nonzero subNatNat_of_le negSucc_ediv natCast_inj natCast_add
136 natCast_sub natCast_mul natCast_div natCast_dvd_natCast natCast_mod natCast_pow natCast_natAbs
137 ofNat_eq_natCast succ_ofNat_pos sub_emod emod_nonneg add_mul_emod_self_left mul_emod_left
138 negOfNat_eq negOfNat negSucc_emod negSucc_coe div_nonneg gcd_natCast_natCast cast_natCast
139 natCast_nonneg emod_self ediv_zero zero_ediv ediv_one emod_zero zero_emod emod_one emod_emod
140 add_emod sub_emod mul_emod mul_emod_left mul_emod_right ediv_nonneg ediv_add_emod' emod_nonneg
141 emod_lt emod_emod_of_dvd emod_eq_zero_of_dvd div_eq_ediv div_eq_ediv_of_dvd emod_eq_of_lt
142 add_mul_emod_self_left dvd_of_dvd_mul_left_of_gcd_one dvd_iff_emod_eq_zero
143 emod_eq_emod_iff_emod_sub_eq_zero ediv_eq_zero_of_lt ofNat_ediv natCast_modEq_iff gcd_a_modEq)
144
145 open Rat (
Lean Infoview
padic_integers.lean:97:0
No info found.
All Messages (0)
Restart File
master* 0 0 0
Zeile 97, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 15:40

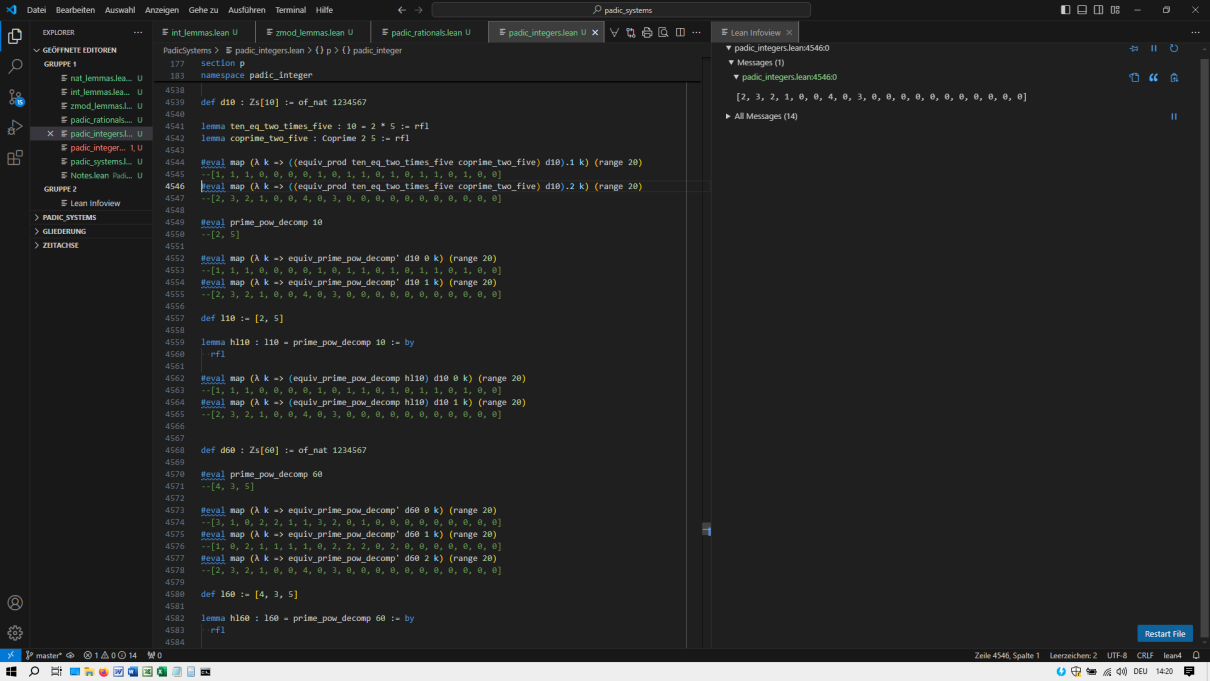

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer > instance : CommRing Zs[p]
padic_integers.lean:1844:0
No info found.
All Messages (0)
177 section p
183 namespace padic_integer
1844 -- Zs[p] is a commutative ring. -/
1845 instance : CommRing Zs[p] :=
1846 { add := padic_integer.add
1847 add_assoc := padic_integer.add_assoc
1848 zero := padic_integer.zero
1849 zero_add := padic_integer.zero_add
1850 add_zero := padic_integer.add_zero
1851 nsmul := padic_integer.nsmul
1852 nsmul_zero := padic_integer.nsmul_zero
1853 nsmul_succ := padic_integer.nsmul_succ
1854 neg := padic_integer.neg
1855 sub := padic_integer.sub
1856 sub_eq_add_neg := padic_integer.sub_eq_add_neg
1857 zsmul := padic_integer.zsmul
1858 zsmul_zero' := padic_integer.zsmul_zero'
1859 zsmul_succ' := padic_integer.zsmul_succ'
1860 zsmul_neg' := padic_integer.zsmul_neg'
1861 neg_add_cancel := padic_integer.neg_add_cancel
1862 add_comm := padic_integer.add_comm
1863 natCast := padic_integer.natCast
1864 intCast := padic_integer.intCast
1865 one := padic_integer.one
1866 natCast_zero := padic_integer.natCast_zero
1867 natCast_succ := padic_integer.natCast_succ
1868 intCast_ofNat := padic_integer.intCast_ofNat
1869 intCast_negSucc := padic_integer.intCast_negSucc
1870 mul := padic_integer.mul
1871 mul_assoc := padic_integer.mul_assoc
1872 zero_mul := padic_integer.zero_mul
1873 mul_zero := padic_integer.mul_zero
1874 one_mul := padic_integer.one_mul
1875 mul_one := padic_integer.mul_one
1876 npow := padic_integer.npow
1877 npow_zero := padic_integer.npow_zero
1878 npow_succ := padic_integer.npow_succ
1879 left_distrib := padic_integer.left_distrib
1880 right_distrib := padic_integer.right_distrib
1881 mul_comm := padic_integer.mul_comm}
1883
1884
1885 /-!
1886 In this section is shown that Zs[0] = Z^N and Zs[1] = 0, Zs[p] is non-trivial iff p ≠ 1, and Zs[p]
1887 is a subsingleton iff p = 1.
1888 -/
1889
1890 /-- Zs[0] = Z^N. -/
Restart File
master*
Zeile 1844, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:02


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer > { } uv > equiv_prod
177 section p
183 namespace padic_integer
2943 section uv
3653 -- Zs[p] is isomorphic to Zs[u] x Zs[v] if p = u * v and u and v are coprime. -/
3654 def equiv_prod (hp : p = u * v) (hcop : Coprime u v) :
3655 Zs[p] ≃+* Zs[u] × Zs[v] :=
3656 { toFun := prod_to_fun
3657 invFun := prod_inv_fun
3658 left_inv := prod_left_inv hp hcop
3659 right_inv := prod_right_inv hp hcop
3660 map_mul' := prod_map_mul hp
3661 map_add' := prod_map_add hp}
3662
3663 lemma equiv_prod_to_fun (hp : p = u * v) (hcop : Coprime u v) :
3664 (equiv_prod hp hcop).toFun = prod_to_fun :=
3665 rfl
3666
3667 lemma equiv_prod_inv_fun (hp : p = u * v) (hcop : Coprime u v) :
3668 (equiv_prod hp hcop).invFun = prod_inv_fun :=
3669 rfl
3670
3671 lemma equiv_prod_left_inv (hp : p = u * v) (hcop : Coprime u v) :
3672 (equiv_prod hp hcop).left_inv = prod_left_inv hp hcop :=
3673 rfl
3674
3675 lemma equiv_prod_right_inv (hp : p = u * v) (hcop : Coprime u v) :
3676 (equiv_prod hp hcop).right_inv = prod_right_inv hp hcop :=
3677 rfl
3678
3679 lemma equiv_prod_map_add (hp : p = u * v) (hcop : Coprime u v) :
3680 (equiv_prod hp hcop).map_add' = prod_map_add hp :=
3681 rfl
3682
3683 lemma equiv_prod_map_mul (hp : p = u * v) (hcop : Coprime u v) :
3684 (equiv_prod hp hcop).map_mul' = prod_map_mul hp :=
3685 rfl
3686
3687 ----------
3688
3689 end uv
3690
3691 ----------
3692
3693 /-- Zs[p * q] is isomorphic to Zs[p] x Zs[q] if p and q are coprime. -/
3694 def equiv_prod' {q : ℕ} (hcop : Coprime p q) : Zs[p * q] ≃+* Zs[p] × Zs[q] :=
3695 equiv_prod rfl hcop
3696
3697 lemma equiv_prod'_eq_equiv_prod {q : ℕ} (hcop : Coprime p q) :
3698 equiv_prod' hcop = equiv_prod rfl hcop :=
padic_integers.lean:3653:0
No info found.
All Messages (0)
Restart File
master*
0 0
0
Zeile 3653, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:07

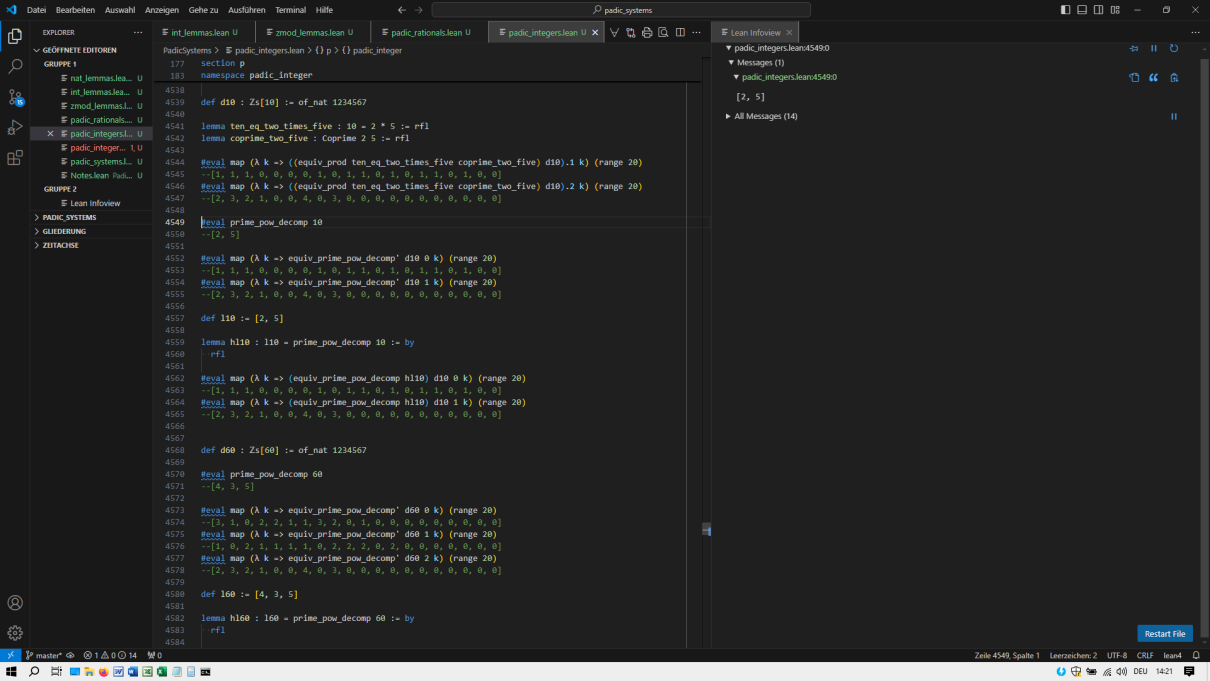

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer > { } ku > equiv_pow
177 section p
183 namespace padic_integer
3718 section ku
4093 /-- Zs[p] is isomorphic to Zs[u] if u = p^k and p = 1 ∨ 0 < k. -/
4094 def equiv_pow (hp : u = p^k) (hpk : p = 1 ∨ 0 < k) : Zs[p] ≃+* Zs[u] :=
4095 { toFun := pow_to_fun
4096 invFun := pow_inv_fun
4097 left_inv := pow_left_inv hp hpk
4098 right_inv := pow_right_inv hp hpk
4099 map_mul' := pow_map_mul hp hpk
4100 map_add' := pow_map_add hp hpk}
4101
4102 ----------
4103
4104 end ku
4105
4106 ----------
4107
4108 /-- Zs[p] is isomorphic to Zs[p^k] if p = 1 ∨ 0 < k. -/
4109 def equiv_pow' (k : ℕ) (hpk : p = 1 ∨ 0 < k) : Zs[p] ≃+* Zs[p^k] :=
4110 equiv_pow rfl hpk
4111
4112 lemma equiv_pow'_eq_equiv_pow (k : ℕ) (hpk : p = 1 ∨ 0 < k) :
4113 equiv_pow' hpk = equiv_pow rfl hpk :=
4114 rfl
4115
4116 ----------
4117
4118 /-!
4119 In this section it is shown that Zs[p] is isomorphic to Π i, Zs[q_i] where q_1 * --- * q_n is the
4120 prime power decomposition of p.
4121 -/
4122
4123 /-
4124 Possibly to do:
4125 https://leanprover.zulipchat.com/#narrow/stream/113488-general/topic/
4126 How.20to.20deal.20with.20rewriting.20in.20types.3F
4127
4128 Yaël Dillies:
4129 "Okay then I would highly suggest you use equivalences of the form
4130 (Π x : α ⊕ β, R x) ≃ (Π a : α, R (Sum.inl a)) × (Π b : β, R (Sum.inr b)).
4131 This is morally equivalent to what you're trying to do, but it's much more idiomatic for
4132 formalisation."
4133 -/
4134
4135 section u
4136
4137 universe u
4138
padic_integers.lean:4093:0
No info found.
All Messages (0)
Restart File
master*
0 0 0
Zeile 4093, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 14:09

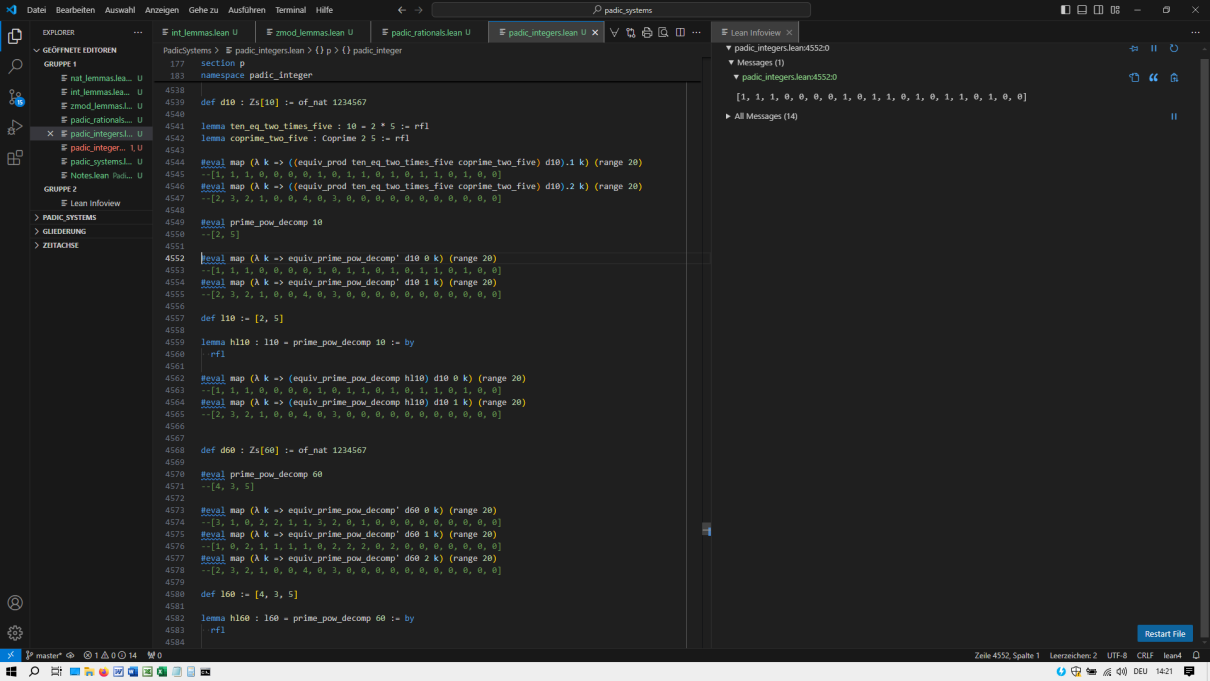

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padic... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
padic_integers.lean:4494:0
No info found.
All Messages (0)
PadicSystems > padic_integers.lean > {} p > {} padic_integer > equiv_prime_pow_decomp
177 section p
183 namespace padic_integer
4494 -- Zs[p] is isomorphic to Π i, Zs[getD l i 1] if l = prime_pow_decomp p. -/
4495 def equiv_prime_pow_decomp {l : List ℕ} (hl : l = prime_pow_decomp p) :
4496 ℤ_[p] ≃+* Π i, ℤ_[getD l i 1] := by
4497 rw [hl]
4498 apply rec_on_prime_powers p
4499 . intro n hn
4500 cases n with
4501 | zero =>
4502 rw [prime_pow_decomp_def_zero]
4503 apply RingEquiv.trans _ (RingEquiv.symm (Pi_equiv_head_of_length_eq_one _))
4504 . rw [length, length]
4505 . rw [List.get]
4506 | succ n =>
4507 rw [eq_head!_prime_pow_decomp_iff_eq_factor_pow_count_nnz (succ_ne_zero _)] at hn
4508 dsimp only at hn
4509 split_ifs at hn with hn1
4510 . rw [←succ_eq_add_one, hn, prime_pow_decomp_def_one]
4511 apply RingEquiv.trans _ (RingEquiv.symm (Pi_equiv_head_of_length_eq_one _))
4512 . rw [length, length]
4513 . rw [List.get]
4514 . have heq := eq_head!_prime_pow_decomp_iff_eq_factor_pow_count_nnz (succ_ne_zero n)
4515 rw [if_neg hn1] at heq
4516 dsimp only at heq
4517 rw [←heq] at hn
4518 apply RingEquiv.trans _ (RingEquiv.symm (Pi_equiv_head_of_length_eq_one _))
4519 . exact (eq_head!_prime_pow_decomp_iff_length_eq_one _).mp hn
4520 . have get_zero_eq_head!_of_ne_nil (α : Type _) [hα : Inhabited α] (l : List α)
4521 (hl : l ≠ []) : List.get l ⟨0, (length_pos.mpr hl)⟩ = head! l := by
4522 rw [head!_of_head?]
4523 rw [get_eq_getElem, getElem_zero]
4524 exact head?_eq_head (length_pos.mpr hl))
4525 rw [get_zero_eq_head!_of_ne_nil (prime_pow_decomp_ne_nil _), ←hn]
4526 . intro n u v _ hnuv hu hv
4527 have hn := (eq_prod_and_coprime_of_prime_pow_decomp_eq_append hnuv).1
4528 have hcop := (eq_prod_and_coprime_of_prime_pow_decomp_eq_append hnuv).2
4529 rw [hnuv]
4530 apply RingEquiv.trans _ (RingEquiv.symm (Pi_append_equiv_prod _ _))
4531 apply RingEquiv.trans (equiv_prod hn hcop)
4532 exact equiv_prod_equiv hu hv
4533
4534 /-- Zs[p] is isomorphic to Π i, Zs[getD (prime_pow_decomp p) i 1]. -/
4535 def equiv_prime_pow_decomp' : ℤ_[p] ≃+* Π i, ℤ_[getD (prime_pow_decomp p) i 1] :=
4536 equiv_prime_pow_decomp rfl
4537
4538 /-
4539 def d10 : Zs[10] := of_nat 1234567
4540
Restart File
master*
0 0
0
Zeile 4494, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:12

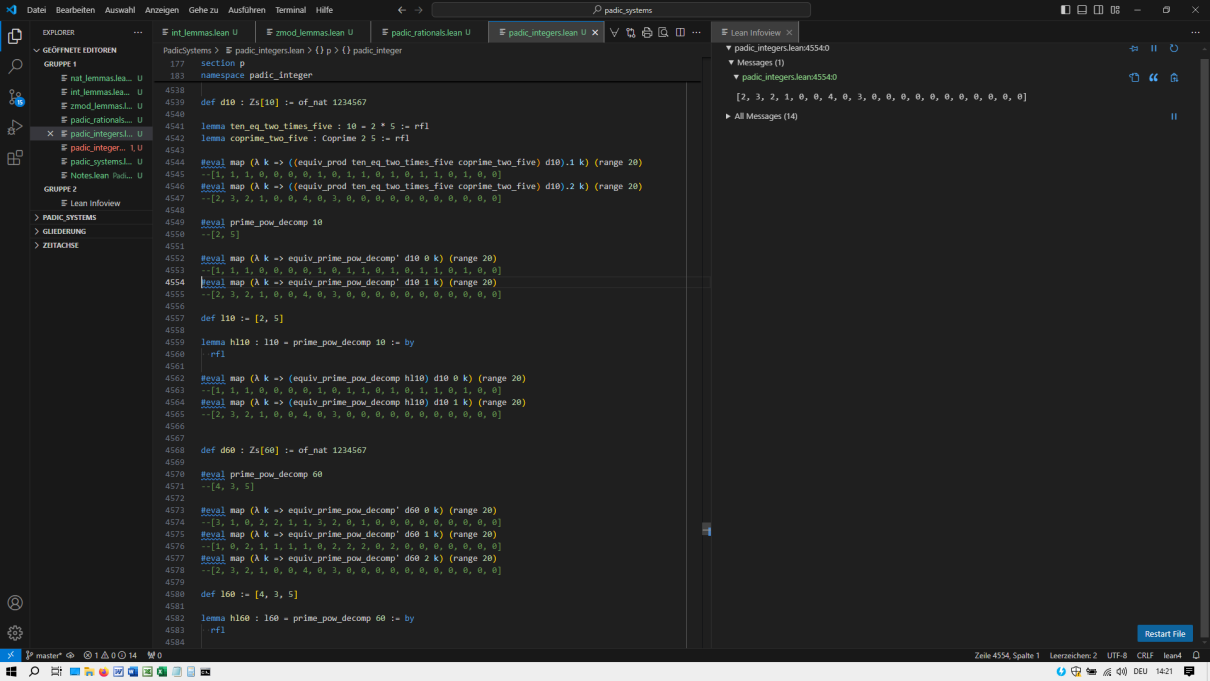

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padic... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > { } padic_integer
177 section p
183 namespace padic_integer
4538
4539 def d10 : Zs[10] := of_nat 1234567
4540
4541 lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
4542 lemma coprime_two_five : Coprime 2 5 := rfl
4543
4544 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
4545 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4546 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
4547 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4548
4549 #eval prime_pow_decomp 10
4550 --[2, 5]
4551
4552 #eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
4553 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4554 #eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
4555 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4556
4557 def l10 := [2, 5]
4558
4559 lemma hl10 : l10 = prime_pow_decomp 10 := by
4560 rfl
4561
4562 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
4563 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4564 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
4565 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4566
4567
4568 def d60 : Zs[60] := of_nat 1234567
4569
4570 #eval prime_pow_decomp 60
4571 --[4, 3, 5]
4572
4573 #eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
4574 --[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4575 #eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
4576 --[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
4577 #eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
4578 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4579
4580 def l60 := [4, 3, 5]
4581
4582 lemma hl60 : l60 = prime_pow_decomp 60 := by
4583 rfl
4584
padic_integers.lean:4544:0
Messages (1)
padic_integers.lean:4544:0
[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4544, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:20

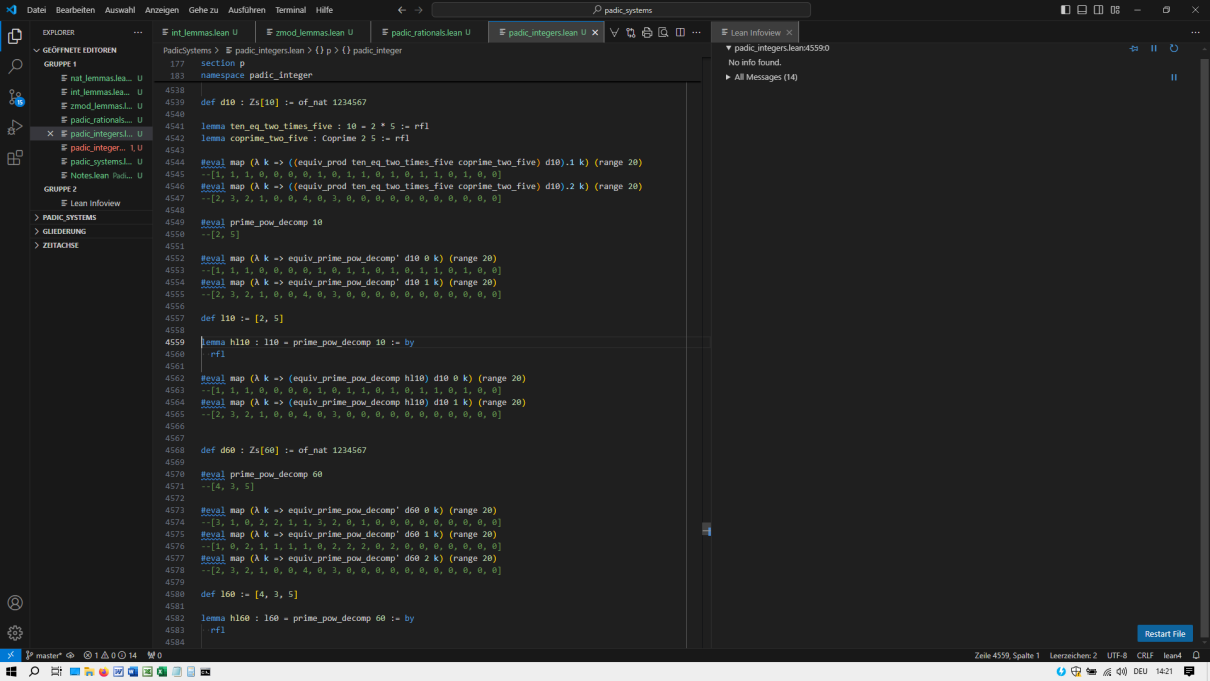


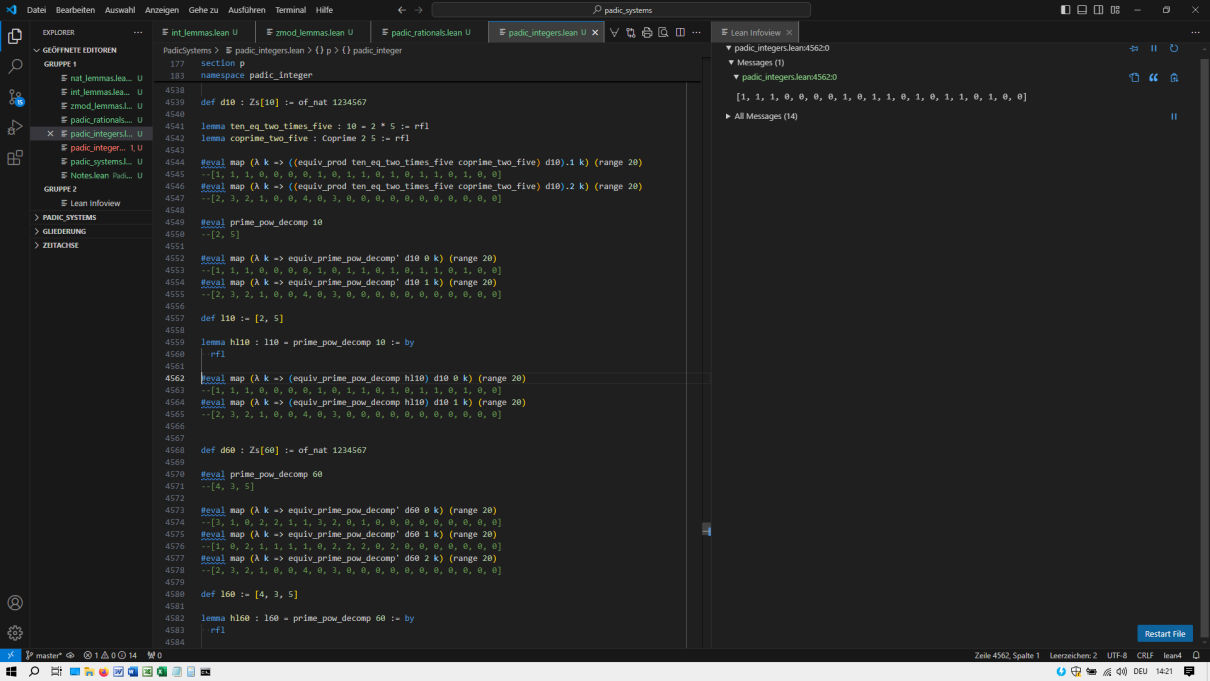

```
177  section p
183  namespace padic_integer
4538
4539 def d10 : Zs[10] := of_nat 1234567
4540
4541 lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
4542 lemma coprime_two_five : Coprime 2 5 := rfl
4543
4544 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
4545 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4546 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
4547 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4548
4549 #eval prime_pow_decomp 10
4550 --[2, 5]
4551
4552 #eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
4553 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4554 #eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
4555 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4556
4557 def l10 := [2, 5]
4558
4559 lemma hl10 : l10 = prime_pow_decomp 10 := by
4560   rfl
4561
4562 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
4563 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4564 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
4565 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4566
4567
4568 def d60 : Zs[60] := of_nat 1234567
4569
4570 #eval prime_pow_decomp 60
4571 --[4, 3, 5]
4572
4573 #eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
4574 --[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4575 #eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
4576 --[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
4577 #eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
4578 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4579
4580 def l60 := [4, 3, 5]
4581
4582 lemma hl60 : l60 = prime_pow_decomp 60 := by
4583   rfl
4584
```

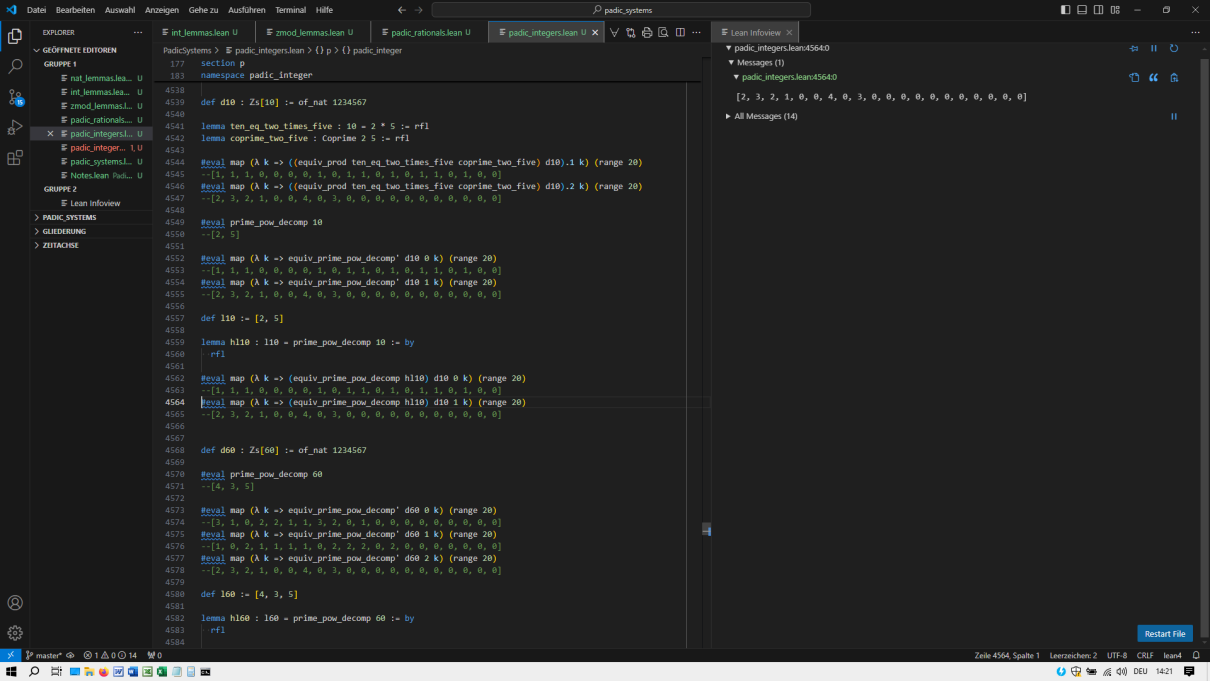

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
padic_integers.lean:4549:0
Messages (1)
padic_integers.lean:4549:0
[2, 5]
All Messages (14)
Restart File
master*
Zeile 4549, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

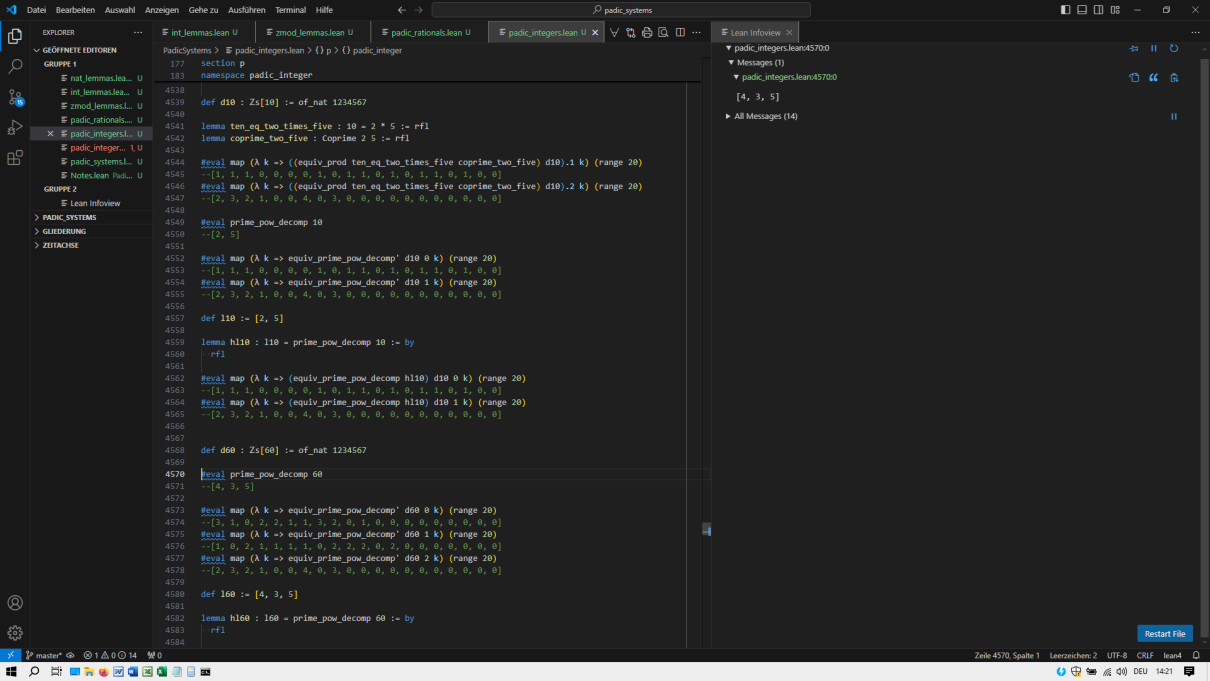

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > { } padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
Lean Infoview
padic_integers.lean:4552:0
Messages (1)
padic_integers.lean:4552:0
[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4552, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

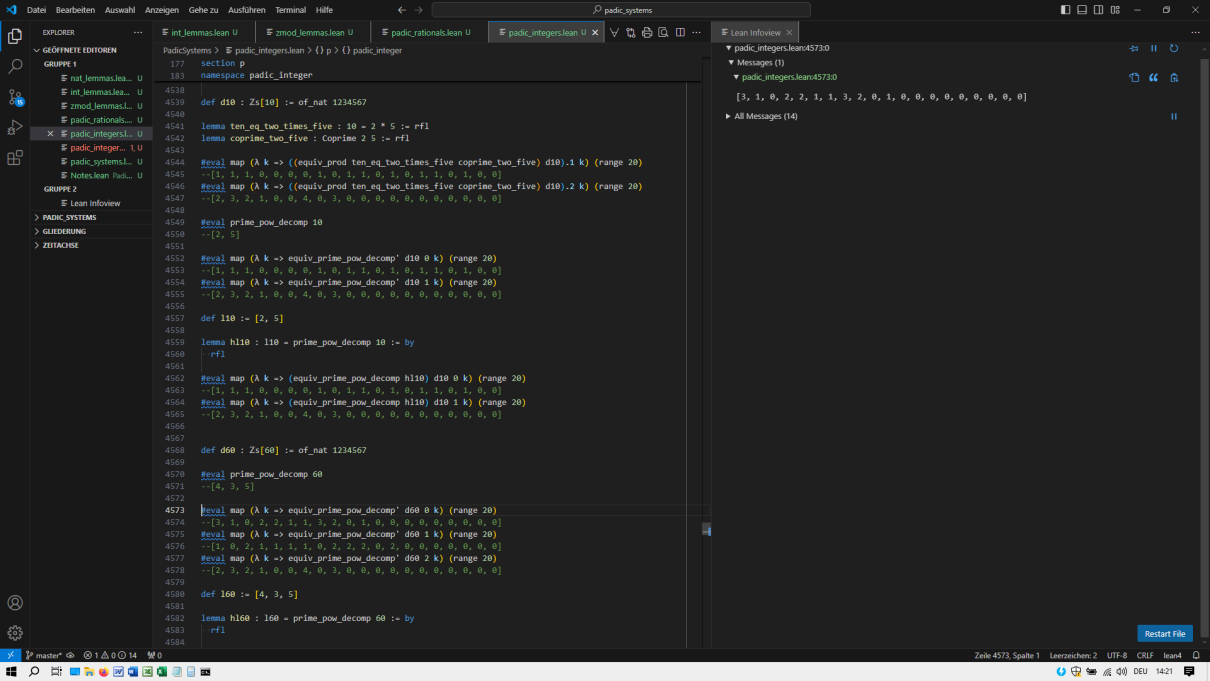

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > {} padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
Lean Infoview
padic_integers.lean:4554:0
Messages (1)
padic_integers.lean:4554:0
[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4554, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

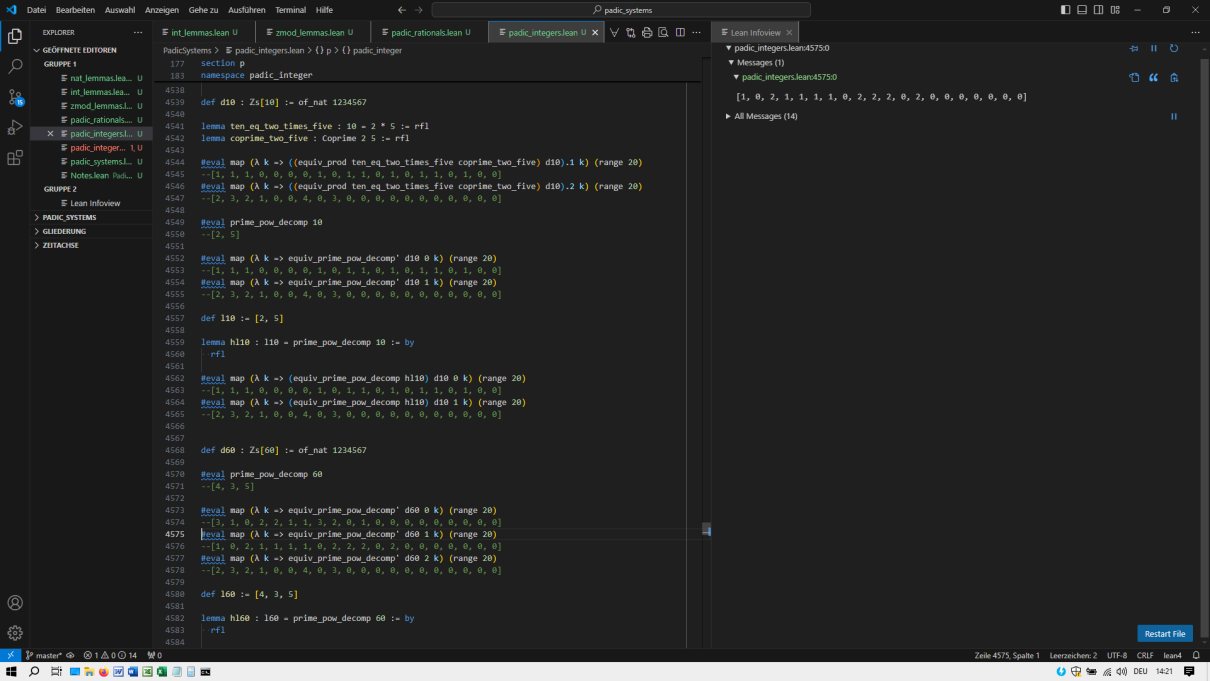

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer
177 section p
183 namespace padic_integer
4538
4539 def d10 : Zs[10] := of_nat 1234567
4540
4541 lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
4542 lemma coprime_two_five : Coprime 2 5 := rfl
4543
4544 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
4545 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4546 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
4547 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4548
4549 #eval prime_pow_decomp 10
4550 --[2, 5]
4551
4552 #eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
4553 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4554 #eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
4555 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4556
4557 def l10 := [2, 5]
4558
4559 lemma hl10 : l10 = prime_pow_decomp 10 := by
4560 rfl
4561
4562 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
4563 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4564 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
4565 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4566
4567
4568 def d60 : Zs[60] := of_nat 1234567
4569
4570 #eval prime_pow_decomp 60
4571 --[4, 3, 5]
4572
4573 #eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
4574 --[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4575 #eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
4576 --[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
4577 #eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
4578 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4579
4580 def l60 := [4, 3, 5]
4581
4582 lemma hl60 : l60 = prime_pow_decomp 60 := by
4583 rfl
4584
padic_integers.lean:4559:0
No info found.
All Messages (14)
Restart File
master*
Zeile 4559, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 14:21

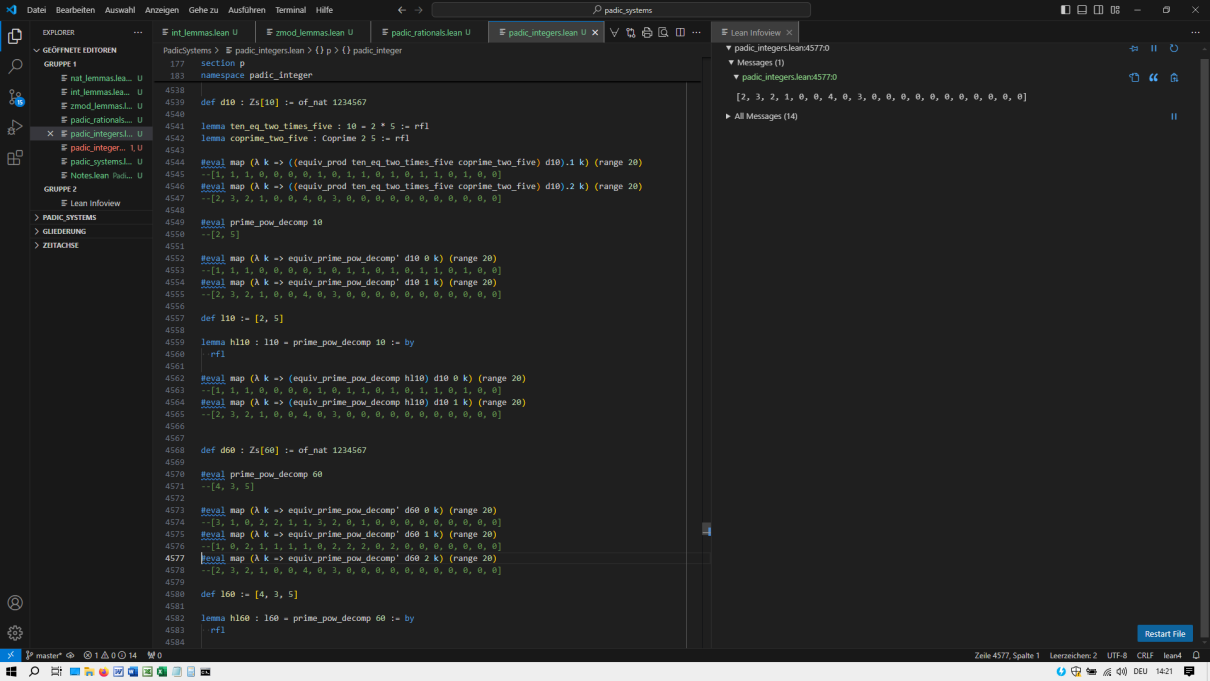

EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > {} p > { } padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
Lean Infoview
padic_integers.lean:4562:0
Messages (1)
padic_integers.lean:4562:0
[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
All Messages (14)
Restart File

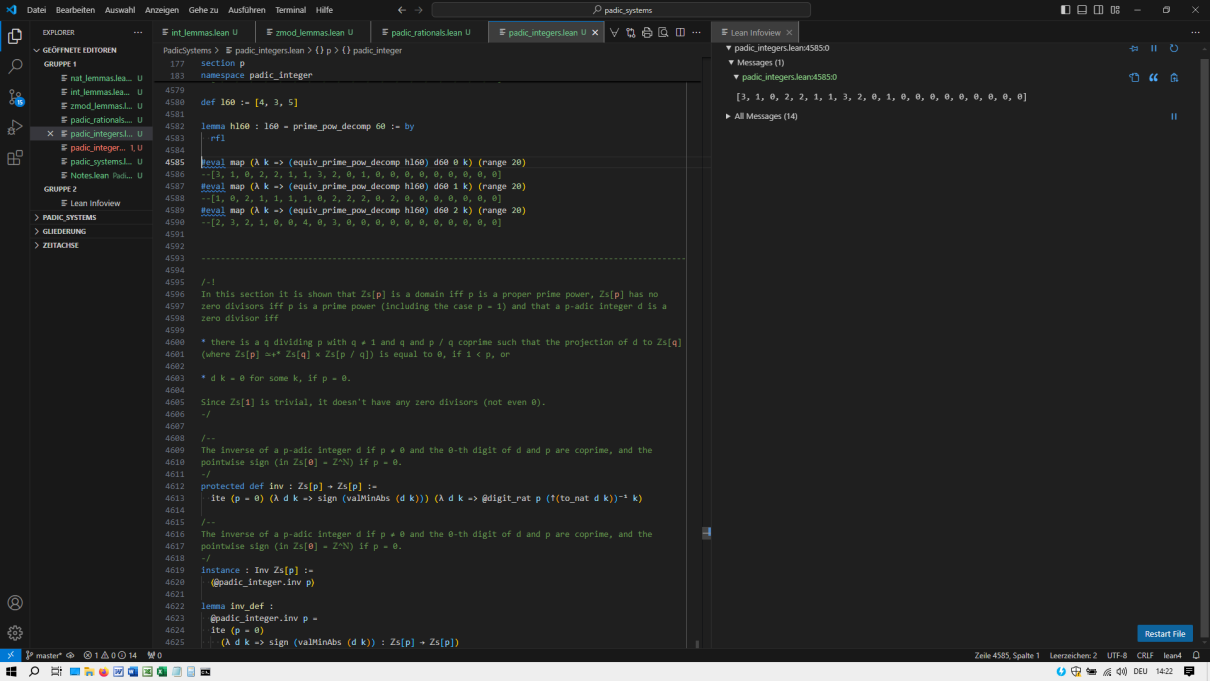

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > { } padic_integer
177 section p
183 namespace padic_integer
4538
4539 def d10 : Zs[10] := of_nat 1234567
4540
4541 lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
4542 lemma coprime_two_five : Coprime 2 5 := rfl
4543
4544 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
4545 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4546 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
4547 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4548
4549 #eval prime_pow_decomp 10
4550 --[2, 5]
4551
4552 #eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
4553 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4554 #eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
4555 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4556
4557 def l10 := [2, 5]
4558
4559 lemma hl10 : l10 = prime_pow_decomp 10 := by
4560 rfl
4561
4562 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
4563 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4564 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
4565 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4566
4567
4568 def d60 : Zs[60] := of_nat 1234567
4569
4570 #eval prime_pow_decomp 60
4571 --[4, 3, 5]
4572
4573 #eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
4574 --[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4575 #eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
4576 --[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
4577 #eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
4578 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4579
4580 def l60 := [4, 3, 5]
4581
4582 lemma hl60 : l60 = prime_pow_decomp 60 := by
4583 rfl
4584
padic_integers.lean:4564:0
Messages (1)
padic_integers.lean:4564:0
[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4564, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

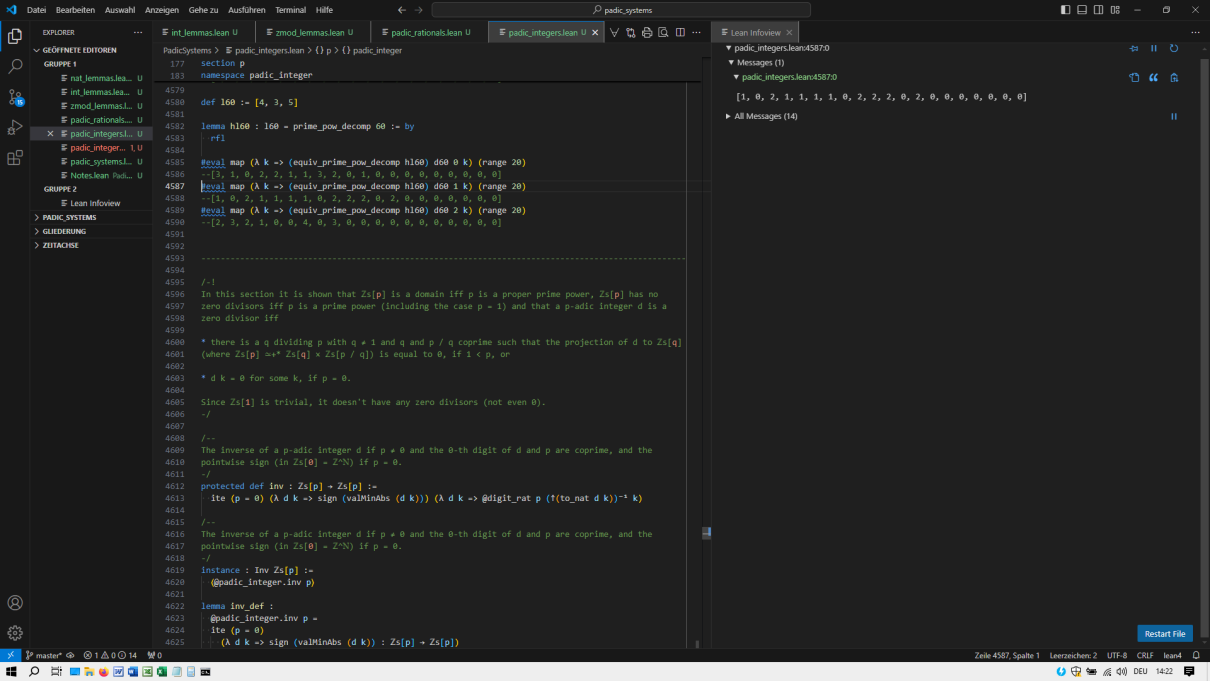

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lean U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > {} padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
padic_integers.lean:4570:0
Messages (1)
padic_integers.lean:4570:0
[4, 3, 5]
All Messages (14)
Restart File
master*
Zeile 4570, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

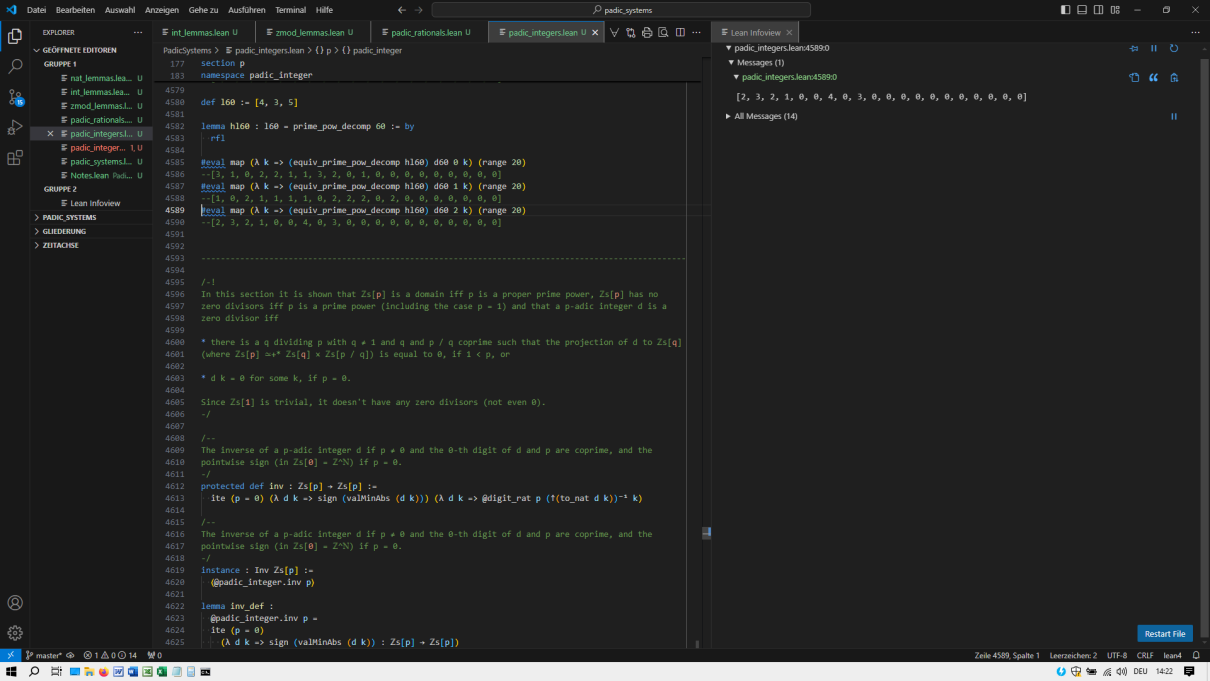

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
Lean Infoview
padic_integers.lean:4573:0
Messages (1)
padic_integers.lean:4573:0
[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4573, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

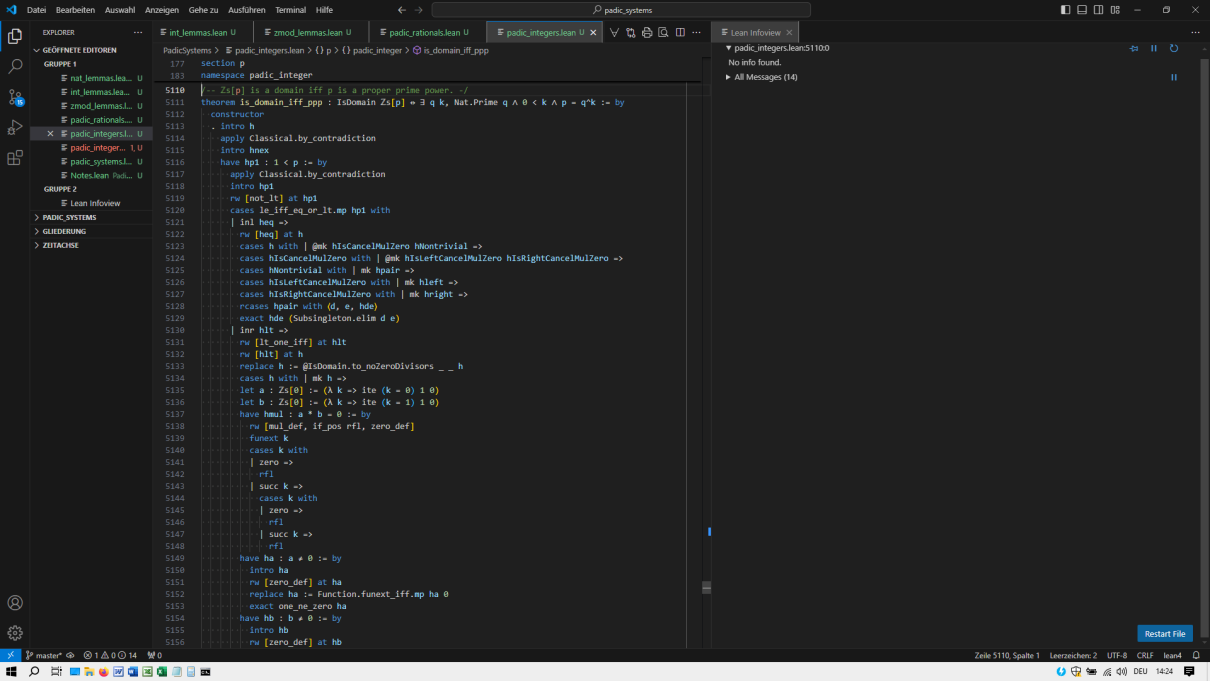

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > { } padic_integer
177 section p
183 namespace padic_integer
4538
4539 def d10 : Zs[10] := of_nat 1234567
4540
4541 lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
4542 lemma coprime_two_five : Coprime 2 5 := rfl
4543
4544 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
4545 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4546 #eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
4547 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4548
4549 #eval prime_pow_decomp 10
4550 --[2, 5]
4551
4552 #eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
4553 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4554 #eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
4555 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4556
4557 def l10 := [2, 5]
4558
4559 lemma hl10 : l10 = prime_pow_decomp 10 := by
4560 rfl
4561
4562 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
4563 --[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
4564 #eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
4565 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4566
4567
4568 def d60 : Zs[60] := of_nat 1234567
4569
4570 #eval prime_pow_decomp 60
4571 --[4, 3, 5]
4572
4573 #eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
4574 --[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4575 #eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
4576 --[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
4577 #eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
4578 --[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4579
4580 def l60 := [4, 3, 5]
4581
4582 lemma hl60 : l60 = prime_pow_decomp 60 := by
4583 rfl
4584
padic_integers.lean:4575:0
Messages (1)
padic_integers.lean:4575:0
[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4575, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

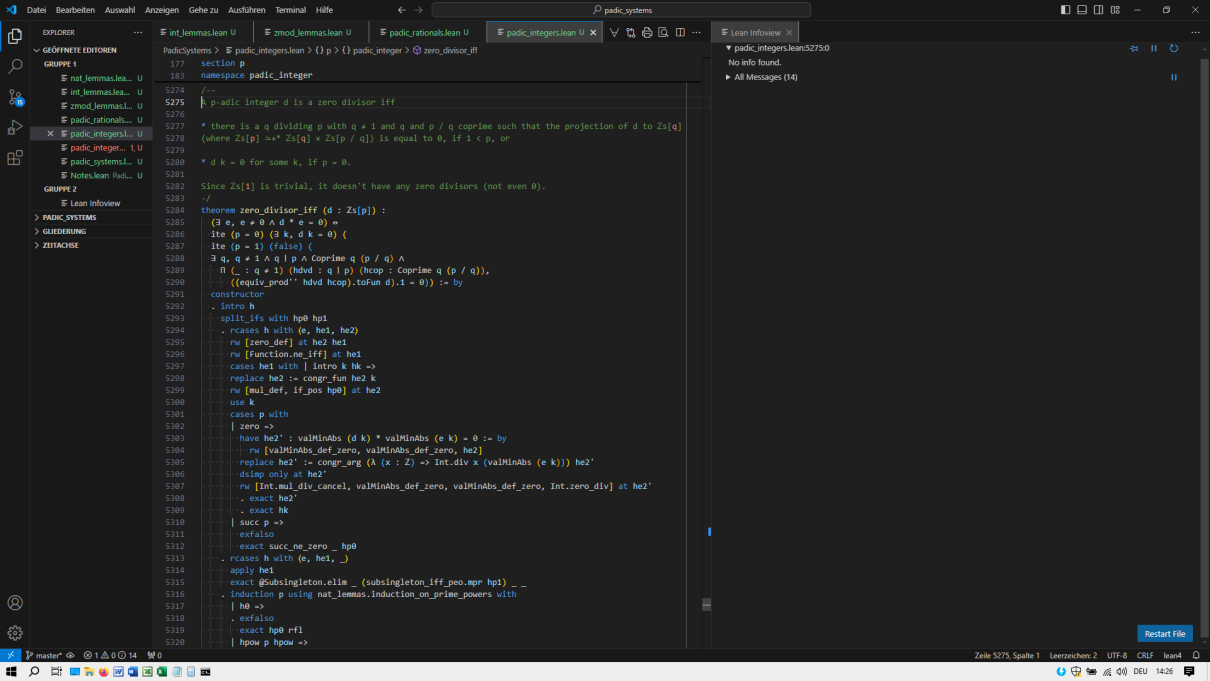

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def d10 : Zs[10] := of_nat 1234567
lemma ten_eq_two_times_five : 10 = 2 * 5 := rfl
lemma coprime_two_five : Coprime 2 5 := rfl
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).1 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => ((equiv_prod ten_eq_two_times_five coprime_two_five) d10).2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval prime_pow_decomp 10
--[2, 5]
#eval map (λ k => equiv_prime_pow_decomp' d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l10 := [2, 5]
lemma hl10 : l10 = prime_pow_decomp 10 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 0 k) (range 20)
--[1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl10) d10 1 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def d60 : Zs[60] := of_nat 1234567
#eval prime_pow_decomp 60
--[4, 3, 5]
#eval map (λ k => equiv_prime_pow_decomp' d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => equiv_prime_pow_decomp' d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
Lean Infoview
padic_integers.lean:4577:0
Messages (1)
padic_integers.lean:4577:0
[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4577, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:21

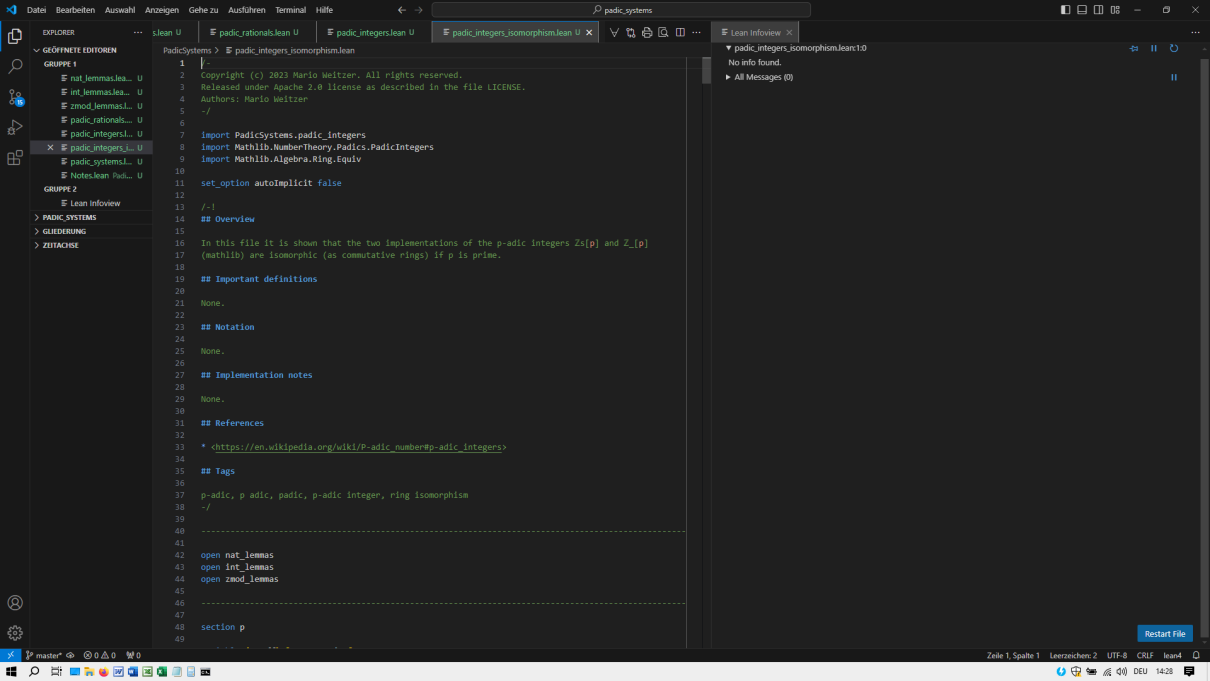

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
/-!
In this section it is shown that Zs[p] is a domain iff p is a proper prime power, Zs[p] has no
zero divisors iff p is a prime integer (including the case p = 1) and that a p-adic integer d is a
zero divisor iff
* there is a q dividing p with q ≠ 1 and q and p / q coprime such that the projection of d to Zs[q]
(where Zs[p] →+* Zs[q] × Zs[p / q]) is equal to 0, if 1 < p, or
* d k = 0 for some k, if p = 0.
Since Zs[1] is trivial, it doesn't have any zero divisors (not even 0).
-/
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
protected def inv : Zs[p] → Zs[p] :=
ite (p = 0) (λ d k => sign (valMinAbs (d k))) (λ d k => @digit_nat p (↑(to_nat d k))⁻¹ k)
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
instance : Inv Zs[p] :=
⟨@padic_integer.inv p⟩
lemma inv_def :
@padic_integer.inv p =
ite (p = 0)
(λ d k => sign (valMinAbs (d k))) : Zs[p] → Zs[p])
padic_integers.lean:4585:0
Messages (1)
padic_integers.lean:4585:0
[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4585, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:22

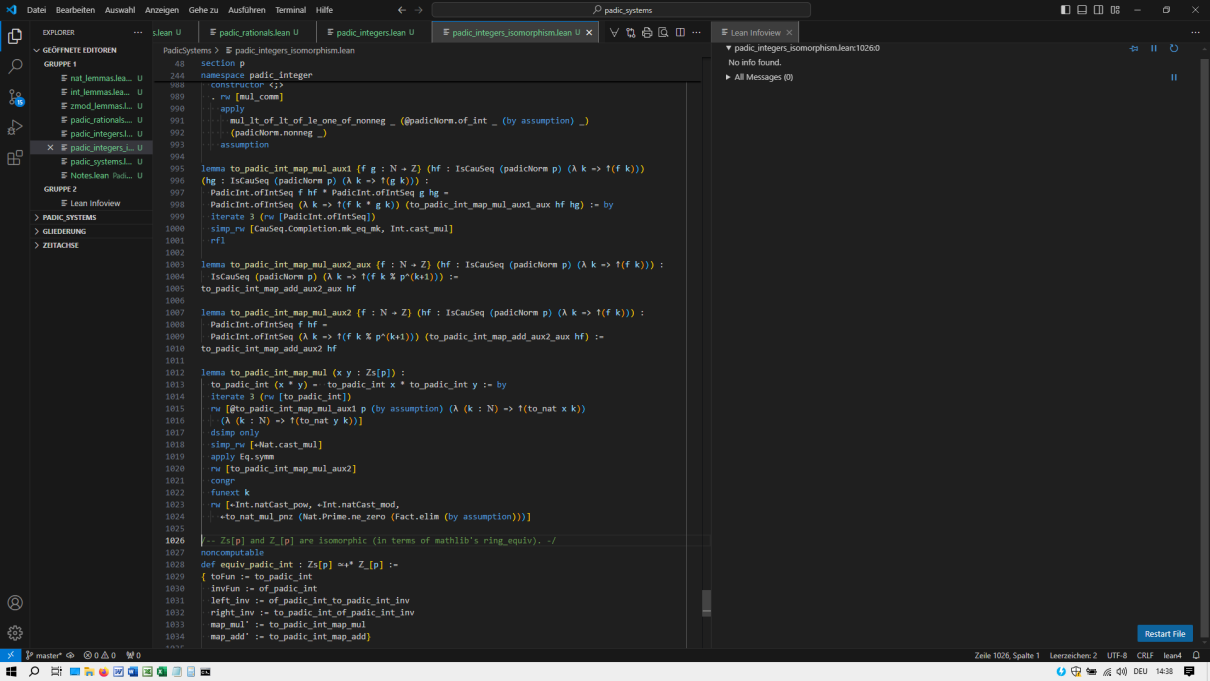

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
/-!
In this section it is shown that Zs[p] is a domain iff p is a proper prime power, Zs[p] has no
zero divisors iff p is a prime power (including the case p = 1) and that a p-adic integer d is a
zero divisor iff
* there is a q dividing p with q ≠ 1 and q and p / q coprime such that the projection of d to Zs[q]
(where Zs[p] →+* Zs[q] × Zs[p / q]) is equal to 0, if 1 < p, or
* d k = 0 for some k, if p = 0.
Since Zs[1] is trivial, it doesn't have any zero divisors (not even 0).
-/
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
protected def inv : Zs[p] → Zs[p] :=
ite (p = 0) (λ d k => sign (valMinAbs (d k))) (λ d k => @digit_nat p (↑(to_nat d k))⁻¹ k)
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
instance : Inv Zs[p] :=
⟨@padic_integer.inv p⟩
lemma inv_def :
@padic_integer.inv p =
ite (p = 0)
(λ d k => sign (valMinAbs (d k))) : Zs[p] → Zs[p])
Lean Infoview
padic_integers.lean:4587:0
Messages (1)
padic_integers.lean:4587:0
[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4587, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:22


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
PadicSystems > padic_integers.lean > { } p > { } padic_integer
section p
namespace padic_integer
def l60 := [4, 3, 5]
lemma hl60 : l60 = prime_pow_decomp 60 := by
rfl
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 0 k) (range 20)
--[3, 1, 0, 2, 2, 1, 1, 3, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 1 k) (range 20)
--[1, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#eval map (λ k => (equiv_prime_pow_decomp hl60) d60 2 k) (range 20)
--[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
/-!
In this section it is shown that Zs[p] is a domain iff p is a proper prime power, Zs[p] has no
zero divisors iff p is a prime power (including the case p = 1) and that a p-adic integer d is a
zero divisor iff
* there is a q dividing p with q ≠ 1 and q and p / q coprime such that the projection of d to Zs[q]
(where Zs[p] →+* Zs[q] × Zs[p / q]) is equal to 0, if 1 < p, or
* d k = 0 for some k, if p = 0.
Since Zs[1] is trivial, it doesn't have any zero divisors (not even 0).
-/
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
protected def inv : Zs[p] → Zs[p] :=
ite (p = 0) (λ d k => sign (valMinAbs (d k))) (λ d k => @digit_nat p (↑(to_nat d k))⁻¹ k)
/--
The inverse of a p-adic integer d if p ≠ 0 and the 0-th digit of d and p are coprime, and the
pointwise sign (in Zs[0] = ℤ^ℕ) if p = 0.
-/
instance : Inv Zs[p] :=
⟨@padic_integer.inv p⟩
lemma inv_def :
@padic_integer.inv p =
ite (p = 0)
(λ d k => sign (valMinAbs (d k))) : Zs[p] → Zs[p])
Lean Infoview
padic_integers.lean:4589:0
Messages (1)
padic_integers.lean:4589:0
[2, 3, 2, 1, 0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
All Messages (14)
Restart File
master*
Zeile 4589, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:22

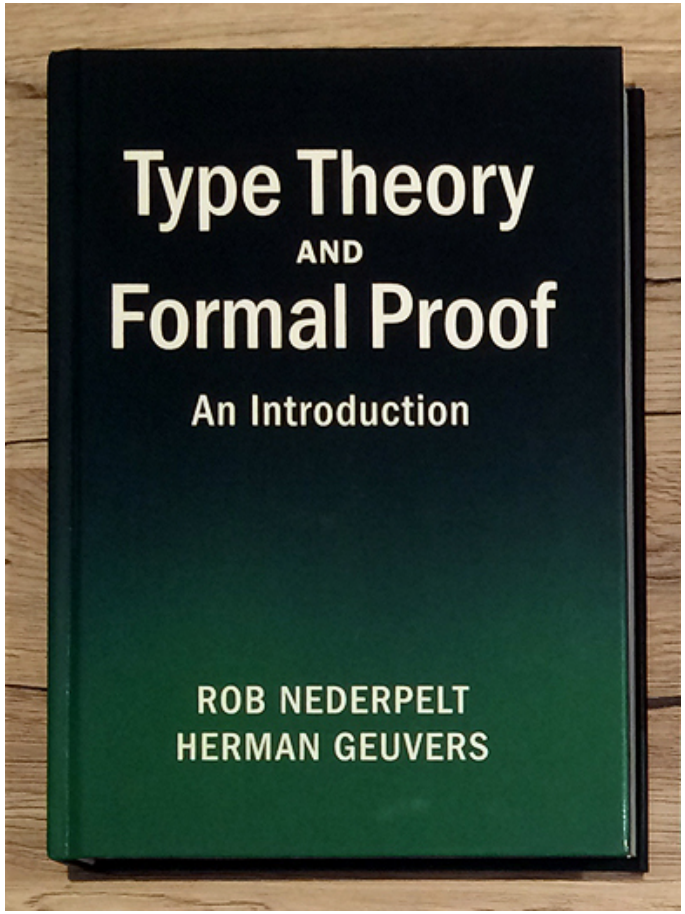

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > ( ) p > { } padic_integer > is_domain_iff_ppp
section p
namespace padic_integer
-- Zs[p] is a domain iff p is a proper prime power. -/
theorem is_domain_iff_ppp : IsDomain Zs[p] ↔ ∃ q k, Nat.Prime q ∧ 0 < k ∧ p = q^k := by
constructor
· intro h
apply Classical.by_contradiction
intro hnex
have hp1 : 1 < p := by
apply Classical.by_contradiction
intro hp1
rw [not_lt] at hp1
cases le_iff_eq_or_lt.mp hp1 with
| inl heq =>
rw [heq] at h
cases h with | @mk hIsCancelMulZero hNontrivial =>
cases hIsCancelMulZero with | @mk hIsLeftCancelMulZero hIsRightCancelMulZero =>
cases hNontrivial with | mk hpair =>
cases hIsLeftCancelMulZero with | mk hleft =>
cases hIsRightCancelMulZero with | mk hright =>
rcases hpair with (d, e, hde)
exact hde (Subsingleton.elim d e)
| inr hlt =>
rw [lt_one_iff] at hlt
rw [hlt] at h
replace h := @IsDomain.to_noZeroDivisors _ _ h
cases h with | mk h =>
let a : Zs[0] := (λ k => ite (k = 0) 1 0)
let b : Zs[0] := (λ k => ite (k = 1) 1 0)
have hmul : a * b = 0 := by
rw [mul_def, if_pos rfl, zero_def]
funext k
cases k with
| zero =>
rfl
| succ k =>
cases k with
| zero =>
rfl
| succ k =>
rfl
have ha : a ≠ 0 := by
intro ha
rw [zero_def] at ha
replace ha := Function.funext_iff.mp ha 0
exact one_ne_zero ha
have hb : b ≠ 0 := by
intro hb
rw [zero_def] at hb
padic_integers.lean:5110:0
No info found.
All Messages (14)
Restart File
master*
Zeile 5110, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:24


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integer... 1, U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
int_lemmas.lean U
zmod_lemmas.lean U
padic_rationals.lean U
padic_integers.lean U
Lean Infoview
PadicSystems > padic_integers.lean > {} p > {} padic_integer > zero_divisor_iff
padic_integers.lean:5275:0
No info found.
All Messages (14)
177 section p
183 namespace padic_integer
5274 /--
5275 A p-adic integer d is a zero divisor iff
5276
5277 * there is a q dividing p with q ≠ 1 and q and p / q coprime such that the projection of d to Zs[q]
5278 (where Zs[p] =:* Zs[q] × Zs[p / q]) is equal to 0, if 1 < p, or
5279
5280 * d k = 0 for some k, if p = 0.
5281
5282 Since Zs[1] is trivial, it doesn't have any zero divisors (not even 0).
5283 -/
5284 theorem zero_divisor_iff (d : Zs[p]) :
5285 (∃ e, e ≠ 0 ∧ d * e = 0) ↔
5286 ite (p = 0) (∃ k, d k = 0) (
5287 ite (p = 1) (false) (
5288 ∃ q, q ≠ 1 ∧ q ∣ p ∧ Coprime q (p / q) ∧
5289 Π (_ : q ≠ 1) (hdvd : q ∣ p) (hcop : Coprime q (p / q)),
5290 ((equiv_prod'' hdvd hcop).toFun d).1 = 0)) := by
5291 constructor
5292 · intro h
5293 split_ifs with hp0 hp1
5294 · rcases h with ⟨e, he1, he2⟩
5295 rw [zero_def] at he2 he1
5296 rw [Function.ne_iff] at he1
5297 cases he1 with ⟨ | intro k hk ⟩ =>
5298 replace he2 := congr_fun he2 k
5299 rw [mul_def, if_pos hp0] at he2
5300 use k
5301 cases p with
5302 | zero =>
5303 have he2' : valMinAbs (d k) * valMinAbs (e k) = 0 := by
5304 rw [valMinAbs_def_zero, valMinAbs_def_zero, he2]
5305 replace he2' := congr_arg (λ (x : ℤ) => Int.div x (valMinAbs (e k))) he2'
5306 dsimp only at he2'
5307 rw [Int.mul_div_cancel, valMinAbs_def_zero, valMinAbs_def_zero, Int.zero_div] at he2'
5308 · exact he2'
5309 · exact hk
5310 | succ p =>
5311 exfalso
5312 exact succ_ne_zero _ hp0
5313 · rcases h with ⟨e, he1, _⟩
5314 apply he1
5315 exact @Subsingleton.elim _ (subsingleton_iff_peo.mpr hp1) _ _
5316 · induction p using nat_lemmas.induction_on_prime_powers with
5317 | h0 =>
5318 · exfalso
5319 exact hp0 rfl
5320 | hpow p hpow =>
Restart File
master*
Zeile 5275, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 14:26


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padi... U
GRUPPE 2
Lean infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
.lean U
padic_rationals.lean U
padic_integers.lean U
padic_integers_isomorphism.lean U
Lean Infoview
PadicSystems > padic_integers_isomorphism.lean
padic_integers_isomorphism.lean:1:0
No info found.
All Messages (0)
/-
Copyright (c) 2023 Mario Weitzer. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Mario Weitzer
-/
import PadicSystems.padic_integers
import Mathlib.NumberTheory.Padics.PadicIntegers
import Mathlib.Algebra.Ring.Equiv
set_option autoImplicit false
/-!
Overview
In this file it is shown that the two implementations of the p-adic integers Zs[p] and Z_[p]
(mathlib) are isomorphic (as commutative rings) if p is prime.
Important definitions
None.
Notation
None.
Implementation notes
None.
References
* <https://en.wikipedia.org/wiki/P-adic_number#p-adic_integers>
Tags
p-adic, p adic, padic, p-adic integer, ring isomorphism
-/
open nat_lemmas
open int_lemmas
open zmod_lemmas
section p
Restart File
master*
Zeile 1, Spalte 1 Leerzeichen: 2 UTF-8 CRLF lean4
DEU 14:28


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
padic_systems
EXPLORER
GEÖFFNETE EDITOREN
GRUPPE 1
nat_lemmas.lea... U
int_lemmas.lea... U
zmod_lemmas.l... U
padic_rationals.... U
padic_integers.l... U
padic_integers_i... U
padic_systems.l... U
Notes.lean Padic... U
GRUPPE 2
Lean Infoview
PADIC_SYSTEMS
GLIEDERUNG
ZEITACHSE
Lean U
padic_rationals.lean U
padic_integers.lean U
padic_integers_isomorphism.lean U
PadicSystems > padic_integers_isomorphism.lean
section p
namespace padic_integer
constructor <;>
. rw [mul_comm]
apply
mul_lt_of_lt_of_le_one_of_nonneg _ (@padicNorm.of_int _ (by assumption) _)
(padicNorm.nonneg _)
assumption
lemma to_padic_int_map_mul_aux1 {f g : N → Z} (hf : IsCauSeq (padicNorm p) (λ k => ↑(f k))) (hg : IsCauSeq (padicNorm p) (λ k => ↑(g k))) :
PadicInt.ofIntSeq f hf * PadicInt.ofIntSeq g hg =
PadicInt.ofIntSeq (λ k => ↑(f k * g k)) (to_padic_int_map_mul_aux1_aux hf hg) := by
iterate 3 (rw [PadicInt.ofIntSeq])
simp_rw [CauSeq.Completion.mk_eq_mk, Int.cast_mul]
rfl
lemma to_padic_int_map_mul_aux2_aux {f : N → Z} (hf : IsCauSeq (padicNorm p) (λ k => ↑(f k))) :
IsCauSeq (padicNorm p) (λ k => ↑(f k % p^(k+1))) :=
to_padic_int_map_add_aux2_aux hf
lemma to_padic_int_map_mul_aux2 {f : N → Z} (hf : IsCauSeq (padicNorm p) (λ k => ↑(f k))) :
PadicInt.ofIntSeq f hf =
PadicInt.ofIntSeq (λ k => ↑(f k % p^(k+1))) (to_padic_int_map_add_aux2_aux hf) :=
to_padic_int_map_add_aux2 hf
lemma to_padic_int_map_mul (x y : Zs[p]) :
to_padic_int (x * y) = to_padic_int x * to_padic_int y := by
iterate 3 (rw [to_padic_int])
rw [@to_padic_int_map_mul_aux1 p (by assumption) (λ (k : N) => ↑(to_nat x k))
(λ (k : N) => ↑(to_nat y k))]
dsimp only
simp_rw [←Nat.cast_mul]
apply Eq.symm
rw [to_padic_int_map_mul_aux2]
congr
funext k
rw [←Int.natCast_pow, ←Int.natCast_mod,
←to_nat_mul_pnz (Nat.Prime.ne_zero (fact.elim (by assumption)))]
-- Zs[p] and Z_[p] are isomorphic (in terms of mathlib's ring_equiv). -/
noncomputable
def equiv_padic_int : Zs[p] ≃+* Z_[p] :=
{ toFun := to_padic_int
invFun := of_padic_int
left_inv := of_padic_int_to_padic_int_inv
right_inv := to_padic_int_of_padic_int_inv
map_mul' := to_padic_int_map_mul
map_add' := to_padic_int_map_add}
Lean Infoview
padic_integers_isomorphism.lean:1026:0
No info found.
All Messages (0)
Restart File
master*
0 0
0
Zeile 1026, Spalte 1
Leerzeichen: 2
UTF-8
CRLF
lean4
DEU
14:38

1) Optional but highly recommended: read



(do all the exercises!)

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

Overlaps with the book but fills in some gaps:

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

Overlaps with the book but fills in some gaps:

- Proof of the Church-Rosser theorem (confluence of the λ-calculus)

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

Overlaps with the book but fills in some gaps:

- Proof of the Church-Rosser theorem (confluence of the λ-calculus)
- Proof of Turing completeness of the λ-calculus

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

Overlaps with the book but fills in some gaps:

- Proof of the Church-Rosser theorem (confluence of the λ-calculus)
- Proof of Turing completeness of the λ-calculus
- Solutions to selected examples from the book

2) Optional: read my lecture notes

`https://marioweitzer.com/sites/default/files/Type%20Theory.pdf`

(ignore first page, only makes sense together with other material)

Overlaps with the book but fills in some gaps:

- Proof of the Church-Rosser theorem (confluence of the λ-calculus)
- Proof of Turing completeness of the λ-calculus
- Solutions to selected examples from the book

3) Install Lean (version 4) on Windows, Linux or MacOS

`https://leanprover-community.github.io/get_started.html`

and create your first project depending on Mathlib

`https://leanprover-community.github.io/install/project.html`

4) Read

`https://leanprover.github.io/theorem_proving_in_lean4`

(do all the exercises!)

4) Read

`https://leanprover.github.io/theorem_proving_in_lean4`

(do all the exercises!)

Find more material at

`https://leanprover-community.github.io/papers.html`

4) Read

`https://leanprover.github.io/theorem_proving_in_lean4`

(do all the exercises!)

Find more material at

`https://leanprover-community.github.io/papers.html`

Ask questions at

`https://leanprover.zulipchat.com`

(very active and helpful community of Lean developers and enthusiasts)

Thank you!