On k-colored Lambda Terms and their Skeletons

Paul Tarau

University of North Texas

PADL'2018

Research supported by NSF grant **1423324**

Overview

- this talk is about an application of logic programming to the modeling of combinatorial properties of lambda terms
- lambda terms in de Bruijn notation are Motzkin trees (also called binary-unary trees) with indices at their leaves counting up on the path to the root the steps to their lambda binder
- as a generalization of *affine lambda terms*, we introduce *k-colored lambda terms* obtained by labeling their lambda nodes with counts of the variables they bind
- we study properties of the *skeletons of k-colored lambda terms*, the Motzkin trees obtained by *erasing the de Bruijn indices labeling their leaves*
- we focus on the (difficult) case of *simply-typed closed k-colored* lambda terms for which a new combinatorial generation algorithm is given and some interesting relations between maximal coloring, size of type expressions and typability are explored

Outline

1. Lambda terms and their tree skeletons
2. Functional equations, data types, and all-term generators : all in one, with DCGs
3. Making lambda terms (somewhat) more colorful
4. Motzkin skeletons for closed, affine and linear terms
5. K-colored closed lambda terms
6. K-colored lambda terms and type inference
7. A two-stage generation algorithm for typable and untypable skeletons
8. Questions?

the paper is organized as a literate Prolog program - our code is available at:

`http://www.cse.unt.edu/~tarau/research/2017/padl18.pro`

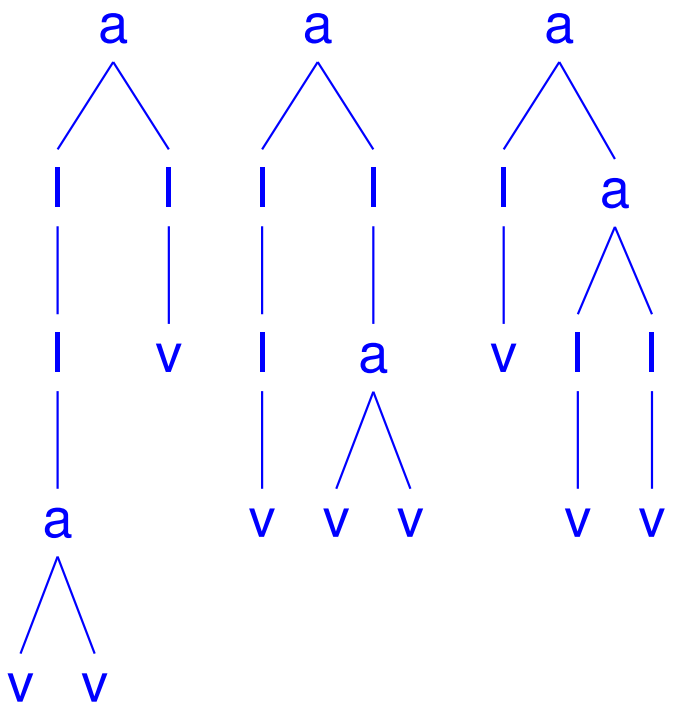
Lambda terms and their tree skeletons

Closed lambda terms and their Motzkin-trees skeletons

- a *Motzkin tree* (also called binary-unary tree) is a rooted ordered tree built from binary nodes, unary nodes and leaf nodes
- the set of Motzkin trees can be seen as the free algebra generated by the constructors `v/0`, `l/1` and `a/2`
- lambda terms in de Bruijn form: the free algebra generated by the constructors `l/1`, and `a/2` and leaves labeled with natural numbers wrapped with the constructor `v/1`
- lambda term in de Bruijn form is *closed* if for each of its de Bruijn indices it exists a lambda binder to which it points, on the path to the root of the tree representing the term
- they are counted by sequence **A135501** in OEIS
- skeleton of a lambda term: the Motzkin tree obtained by erasing the labels at its leaves
- we call a Motzkin tree *closable* if it is the skeleton of at least one closed lambda term

Some Closable and Unclosable Motzkin Skeletons

a) 3 closable skeletons



b) 3 unclosable skeletons

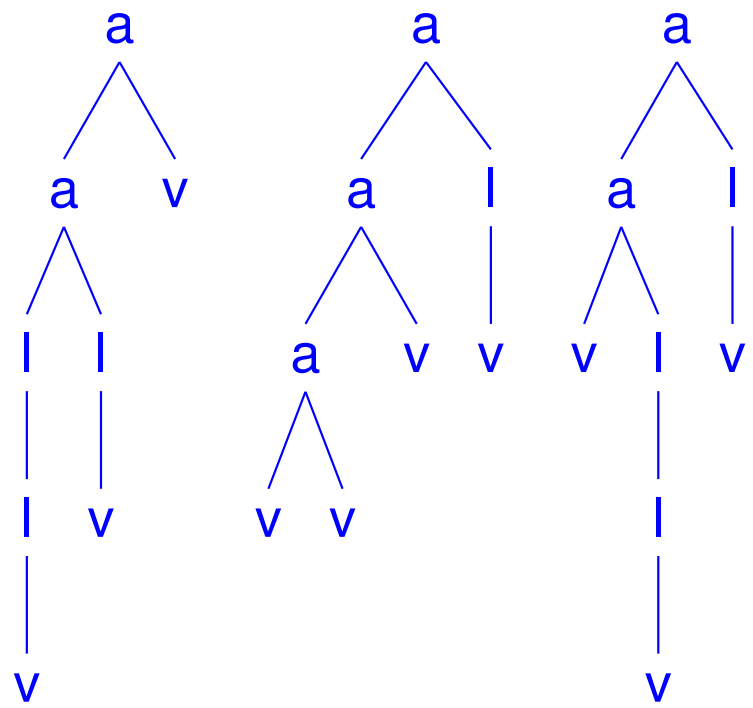


Figure: Closable vs. unclosable skeletons of size 7

Functional equations, data types, and all-term generators : all in one, with DCGs

Functional equations, data types, and all-term generators : all in one, with DCGs

- the relationship between algebraic functional equations and unambiguous CF-grammars is well known (see e.g., Flajolet's book)
- Prolog preprocesses a CF-grammar notation from a Definite Clause Grammars notation to actual code
- $\Rightarrow$ we can design a direct translation to them from algebraic functional equations
- BTW, unambiguous CF-grammars also describe data data types (e.g., Haskell's data declarations)!

The set of Motzkin trees $M(z)$ follows the algebraic functional equation $M(z) = z + zM(z) + zM^2(z)$.
Translating to Prolog's Definite Clause Grammar notation (with "," standing for conjunction and ";" for disjunction) we obtain:

```
m --> z ; z,m ; z,m,m. % we can think here that z consumes 1 size unit
```

A size definition: each constructor costs as much as its arity

Proposition

*The set of terms of size n for the size definition {*application=2, lambda=1, variable=0*} is equal to the set of terms of size* $n+1$ *for the size definition {*application=1, lambda=1, variable=1*}.*

- true, given that the number of leaves in a Motzkin tree is the number of binary nodes + *1*
- the term $l(a(v(0), v(0)))$ will have size $3 = 1+2$ with our definition, which corresponds to size $4 = 1+1+1+1$ using the size definition for sequence **A135501**
- our size definition is implemented as

```
l(SX,X):-succ(X,SX). % for l/1 constructors
a-->l,l.             % for a/2 constructors
```

with Prolog's DCG notation controlling the consumption of size units from `N` to `0`

Closable and unclosable Motzkin skeletons

- closable skeleton: a Motzkin tree that is the skeleton of at least one closed lambda term
- can such a skeleton predetermine if it can be decorated to a closed lambda term?
- YES, and there are slightly more unclosable Motzkin trees than closable ones as size grows:
- *closable:*
  0,1,1,2,5,11,26,65,163,417,1086,2858,7599,20391,55127,150028,410719, ...
- *unclosable:*
  1,0,1,2,4,10,25,62,160,418,1102,2940,7912,21444,58507,160544,442748, ...
- What happens to them asymptotically? $\frac{1}{\sqrt{5}}$ are closable (see LOPSTR'17 paper with Olivier Bodini)
- how we do it: all we need is a DCG grammar describing them!

A CF grammar for closable skeletons

Proposition

A Motzkin tree is a skeleton of a closed lambda term if and only if it exists at least one lambda binder on each path from the leaf to the root.

- at least one lambda (`l/1` constructor) on each path
- $\Rightarrow$ Motzkin trees below the `l/1` constructor contain only a/2 branches and leaves
- assuming motSkel/3 generates Motzkin trees i.e., by decorating `m/2`

```
closable(N,X):-closable(X,N,0).

closable(l(Z))-->l,motSkel(Z). % a/2 and leaves only
closable(a(X,Y))-->a,closable(X),closable(Y).
```

Making lambda terms (somewhat) more colorful

Making lambda terms (somewhat) more colorful

- *our lambda terms in de Bruijn form*: as the free algebra generated by the constructors `l/1`, `r/1` and `a/2` with leaves labeled with natural numbers (and seen as wrapped with the constructor `v/1` when convenient)
- we split lambda constructors into 2 classes:
  - *binding lambdas*, `l/1` that are reached by at least one de Bruijn index
  - *free lambdas*, `r/1`, that are not reached by any de Bruijn index
- ex: $\lambda x.\lambda y.\lambda z.(y\ (x\ x))$ is written as $l(l(r(a(v(0), a(v(1), v(1))))))$
- free lambda terms will have no say on terms being closed, but they will impact on which closed terms are simply typed
- *2-colored Motzkin trees*: the free algebra generated by the constructors `v/0`, `l/1`, `r/1` and `a/2`
- *2-colored Motzkin skeleton of a lambda term*: the tree obtained by erasing the de Bruijn indices labeling their leaves

A bijection between 2-colored Motzkin skeletons and non-empty binary trees

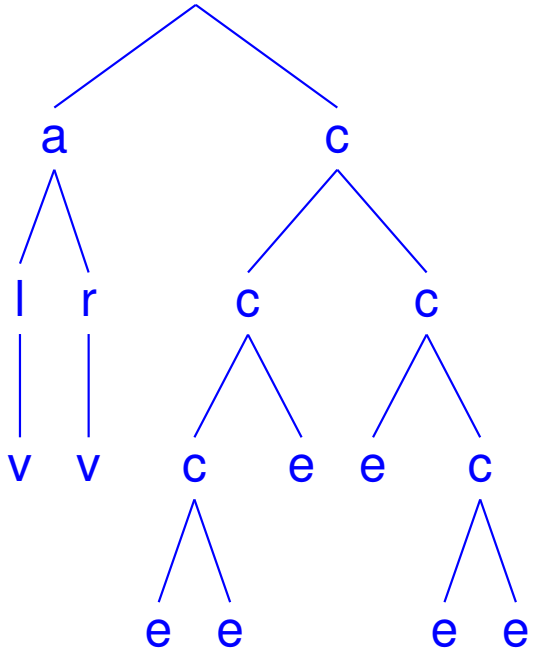
- *binary trees*: the free algebra generated by `e/0` and `c/2`
- *bijections between* the Catalan family of combinatorial objects and 2-colored Motzkin trees : they exist but they involve artificial constructs (e.g., depth first search on rose-trees, indirect definition of the mapping)
- a reason to find a new one ! In Prolog we need *one* relation for f and f^{-1}:

```
cat_mot(c(e,e),v).
cat_mot(c(X,e),l(A)):-X=c(_,_),cat_mot(X,A).
cat_mot(c(e,Y),r(B)):-Y=c(_,_),cat_mot(Y,B).
cat_mot(c(X,Y),a(A,B)):-X=c(_,_),Y=c(_,_),
  cat_mot(X,A),
  cat_mot(Y,B).
```

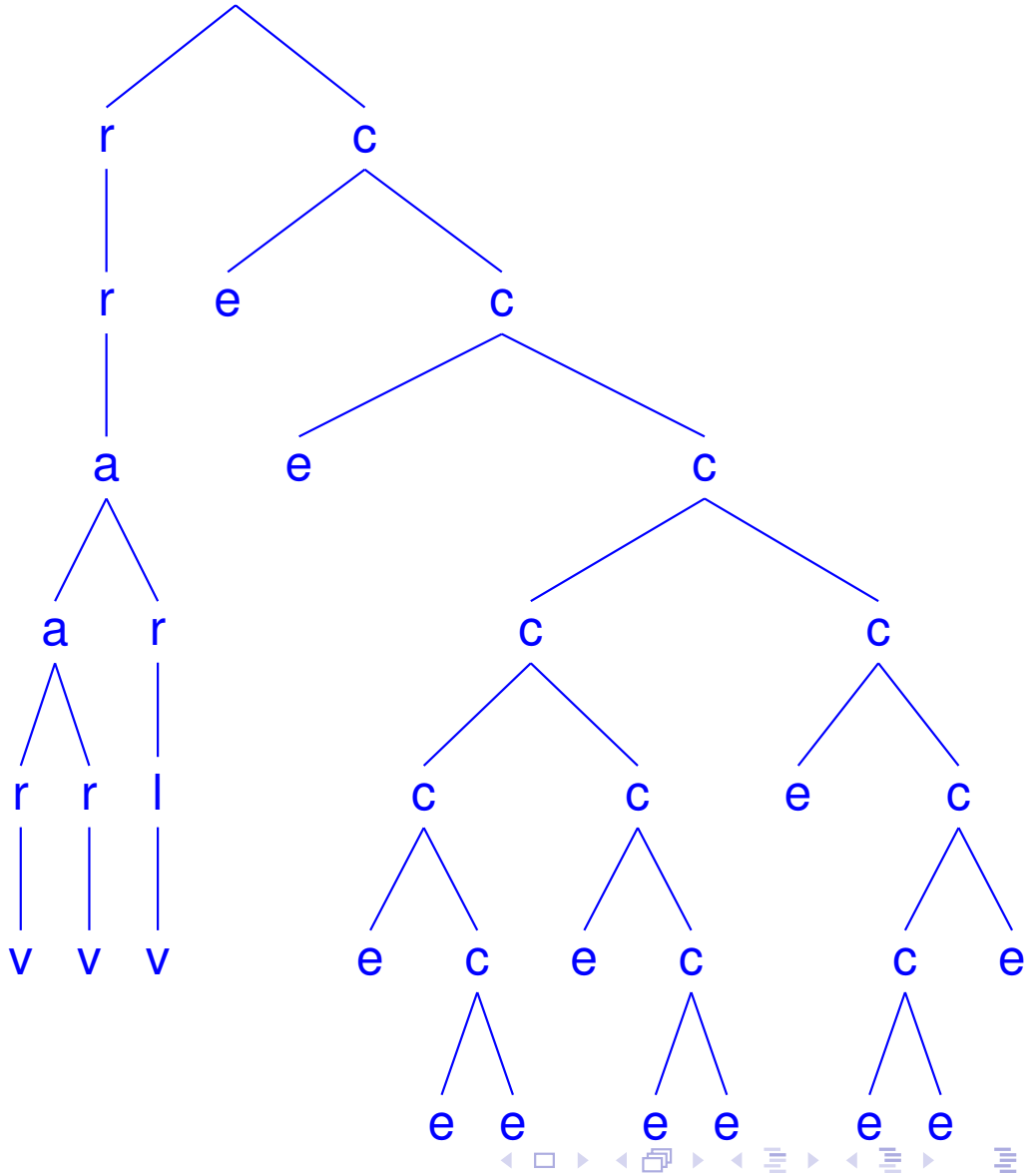
- one can include the empty tree `e` by an arithmetization mechanism that defines successor and predecessor (e.g., our PPDP'15 paper)

The 2-Motzkin to non-empty binary trees bijection at work

two “twinned” trees of size 4



two “twinned” trees size 10



Motzkin skeletons for closed, affine and linear terms

Generating 2-colored Motzkin skeletons

- Motzkin skeletons of lambda terms: we erase the de Bruijn indices at leaves
- size of constructors is their arity a=2, l=1, r=1, v=0
- in fact, this size definition matches the size of their heap representation
- generate all the splits of the size N into 2*A+L+R where A=size of a/2 etc.
- define predicates to consume size units as needed:

```
lDec(c(SL,R,A),c(L,R,A)):-succ(L,SL).
rDec(c(L,SR,A),c(L,R,A)):-succ(R,SR).
aDec(c(L,R,SA),c(L,R,A)):-succ(A,SA).
```

Proposition

If a Motzkin tree is a skeleton of a closed lambda term then it exists at least one lambda binder on each path from the leaf to the root.

Thus $L \leq A+1$. The number of leaves is $A+1$ given A application nodes.

Generating closed affine terms

```
afLam(N,T):-sum_to(N,Hi,Lo),has_enough_lambdas(Hi),afLinLam(T,[],Hi,Lo).

has_enough_lambdas(c(L,_,A)):-succ(A,L). % L=A+1

afLinLam(v(X),[X])-->[].
afLinLam(l(X,A),Vs)-->lDec,afLinLam(A,[X|Vs]).
afLinLam(r(A),Vs)-->rDec,afLinLam(A,Vs).
afLinLam(a(A,B),Vs)-->aDec,{subset_and_complement_of(Vs,As,Bs)},
  afLinLam(A,As), % a lambda cannot go
  afLinLam(B,Bs). % to both branches !!!

subset_and_complement_of([],[],[]).
subset_and_complement_of([X|Xs],NewYs,NewZs):-
  subset_and_complement_of(Xs,Ys,Zs),
  place_element(X,Ys,Zs,NewYs,NewZs).

place_element(X,Ys,Zs,[X|Ys],Zs).
place_element(X,Ys,Zs,Ys,[X|Zs]).
```

Linear terms, skeletons of affine and linear terms

```
linLam(N,T):-N mod 3=:=1,
  sum_to(N,Hi,Lo),has_no_unused(Hi),
  afLinLam(T,[],Hi,Lo).

has_no_unused(c(L,0,A)):-succ(A,L). % 0 r/1 constructors !
```

Proposition

If a Motzkin tree with n binary nodes is a skeleton of a linear lambda term, then it has exactly $n+1$ unary nodes, with one on each path from the root to its $n+1$ leaves.

- affine: 0,1,2,3,9,30,81,242,838,2799,9365,33616,122937,449698
- linear: 0,1,0,0,5,0,0,60,0,0,1105,0,0,27120,0,0,828250

K-colored closed lambda terms

K-colored closed lambda terms: affine terms ++

- a natural generalization of affine terms: each lambda can bind at most k variable occurrences
- k-colored lambda terms: same as BCK(p) terms - with $k = p + 1$
- well, for p>1 they are should actually be called BCKW(p)) as the W combinator is needed when a lambda binds at least two variable occurrences ...
- Bodini and Gittenberger: closed formula with about 100+ math symbols for BCK(2)

Proposition

General lambda terms are computationally equivalent to 3-colored lambda terms (i.e., BCK(2) terms).

Proof. Convert a lambda term to S,K combinators. S and K use at most two variables for each lambda: $S = \lambda f.\lambda g.\lambda x.(f\ x)(g\ x)$, $K = \lambda x.\lambda y.x$. □

Generating k-colored closed lambda terms

```
kColoredClosed(N,X):-kColoredClosed(X,[],N,0).

kColoredClosed(v(I),Vs)-->{nth0(I,Vs,V),inc_var(V)}.
kColoredClosed(l(K,A),Vs)-->l, % <= the K-colored lambda binder
  kColoredClosed(A,[V|Vs]),
  {close_var(V,K)}.
kColoredClosed(a(A,B),Vs)-->a,
  kColoredClosed(A,Vs),
  kColoredClosed(B,Vs).

inc_var(X):-var(X),!,X=s(_). % count new variable under the binder
inc_var(s(X)):-inc_var(X).

close_var(X,K):-var(X),!,K=0. % close and convert to ints
close_var(s(X),SK):-close_var(X,K),succ(K,SK).
```

Examples of k-colored closed lambda terms

3-colored lambda terms of size 3, exhibiting colors 0,1,2.

```
?- kColoredClosed(3,X).
X = l(0, l(0, l(1, v(0)))) ;
X = l(0, l(1, l(0, v(1)))) ;
X = l(1, l(0, l(0, v(2)))) ;
X = l(2, a(v(0), v(0))) .
```

- in a closed term with n application nodes, the counts of k-colored lambdas must sum up to $n+1$
- $\Rightarrow$ we can generate a binary tree and then decorate it with lambdas satisfying this constraint
- the constraint holds for closed subterms, recursively
- can this mechanism reduce the amount of backtracking and accelerate term generation?

K-colored lambda terms and type inference

Simply-typed lambda terms: the challenges I

The study of the combinatorial properties of simply-typed lambda terms is notoriously hard. The two most striking facts that one might notice when inferring types are:

- *non-monotonicity*, as crossing a lambda increases the size of the type, while crossing an application node trims it down
- *agreement via unification (with occurs check)* between the types of each variable under a lambda

Interestingly, to our best knowledge, no SAT or ASP algorithms exist in the literature that attack the combined type inference and combinatorial generation problem for lambda terms, most likely because of the complexity of emulating unification-with-occurs-check steps in propositional logic.

Simply-typed lambda terms: the challenges II

- the asymptotically vanishing density of simply-typed lambda terms in the set of closed terms
- even more dramatic in the case of their normal forms
- this makes their all-term and random-term generation increasingly difficult with size
- as for simply typed terms, no analytic method is known for estimating the probabilities describing Boltzmann sampling, one needs to filter through the set of closed lambda terms or normal forms until a typable term is found
- it is increasingly computationally expensive as their density decreases with size
- basic combinatorial properties like counts for terms of a given size have been obtained so far only by generating all terms

Type inference for k-colored terms

```
simplyTypedColored(N,X,T):-simplyTypedColored(X,T,[],N,0).

simplyTypedColored(v(X),T,Vss)-->{
    member(Vs:T0,Vss),
    unify_with_occurs_check(T,T0),
    addToBinder(Vs,X)
  }.
simplyTypedColored(l(Vs,A),S->T,Vss)-->l,
  simplyTypedColored(A,T,[Vs:S|Vss]),
  {closeBinder(Vs)}.
simplyTypedColored(a(A,B),T,Vss)-->a,
  simplyTypedColored(A,(S->T),Vss),
  simplyTypedColored(B,S,Vss).

addToBinder(Ps,P):-var(Ps),!,Ps=[P|_].
addToBinder([_|Ps],P):-addToBinder(Ps,P).

closeBinder(Xs):-append(Xs,[],_),!.
```

How many simply typed terms are 2-colored (i.e., affine)?

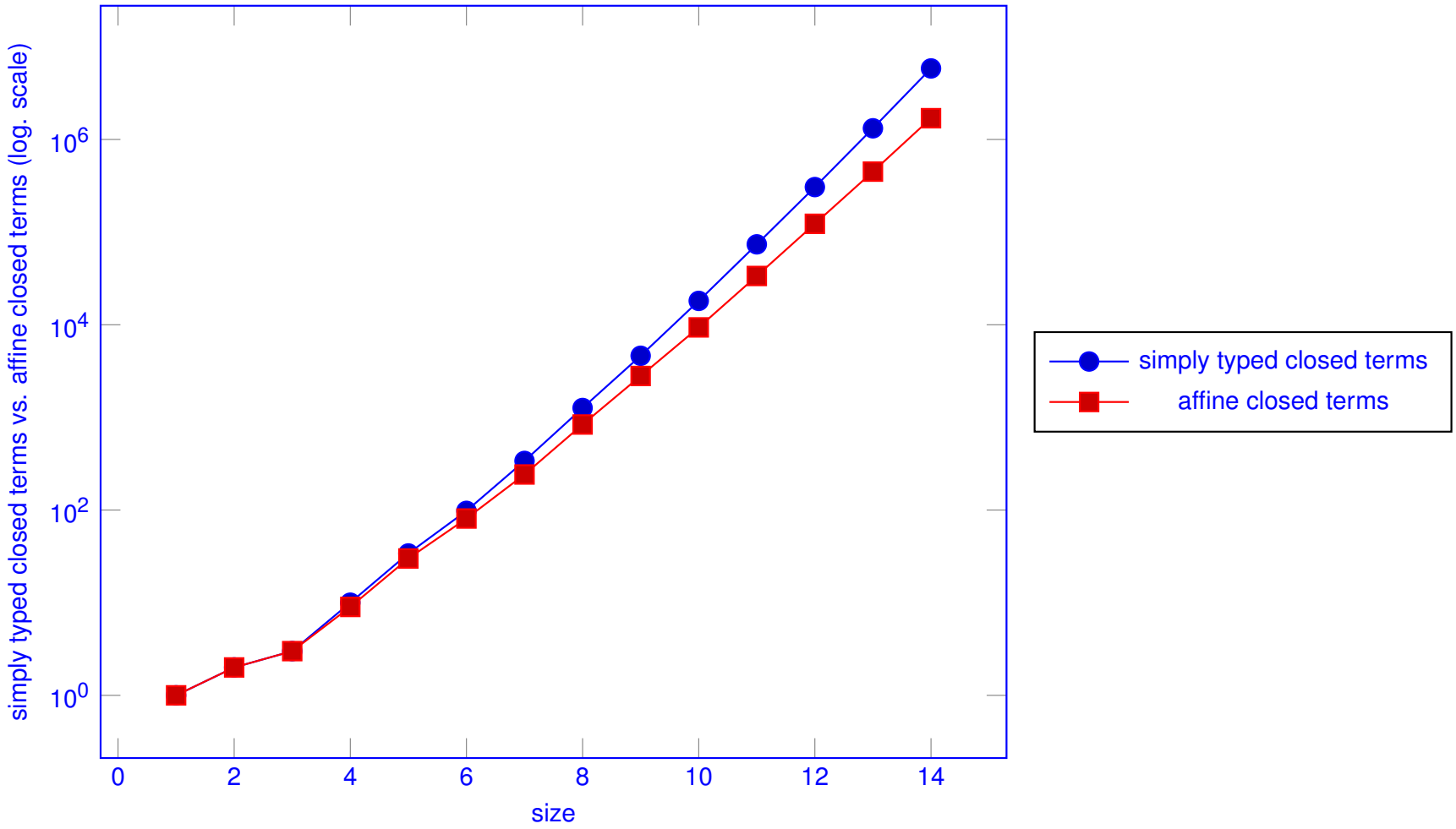


Figure: Simply typed closed terms and affine closed terms by increasing sizes

Are colors growing proportional to the log of their type-sizes?

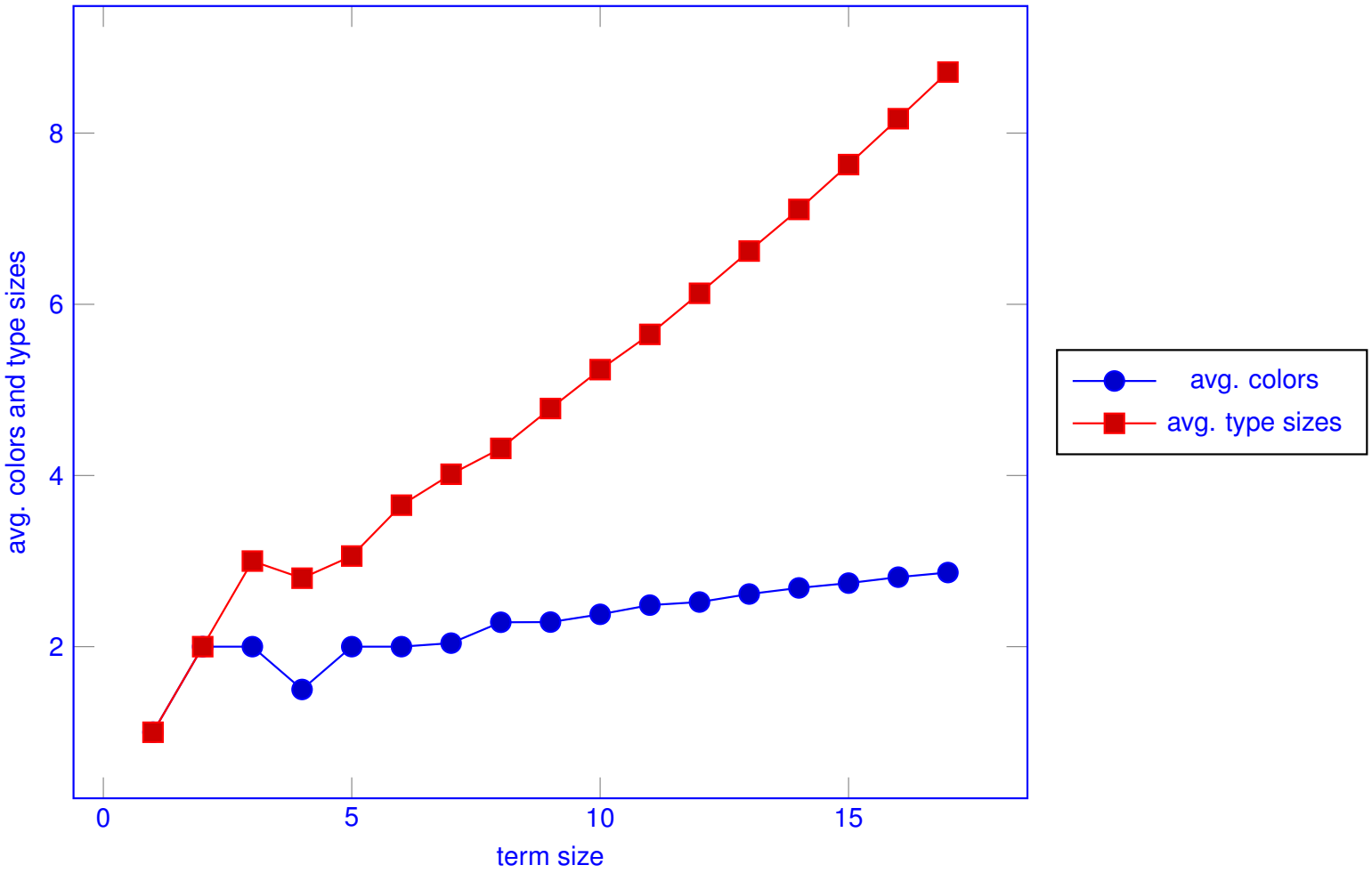


Figure: Growth of colors and type sizes

a most colorful term: reaches the maximum number of colors

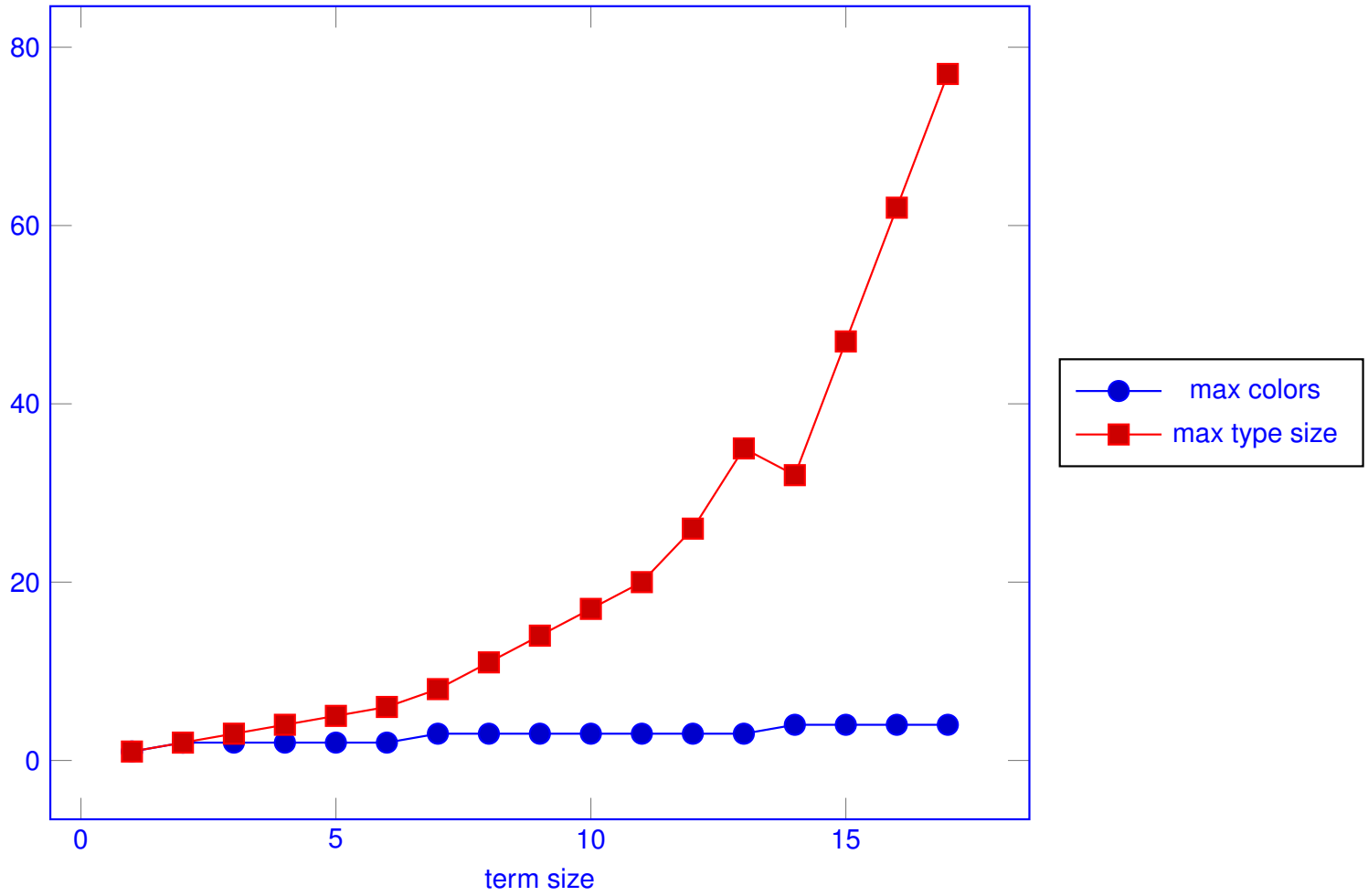


Figure: Colors of a most colorful term vs. its maximum type size

Does a most colorful term reach max type size?

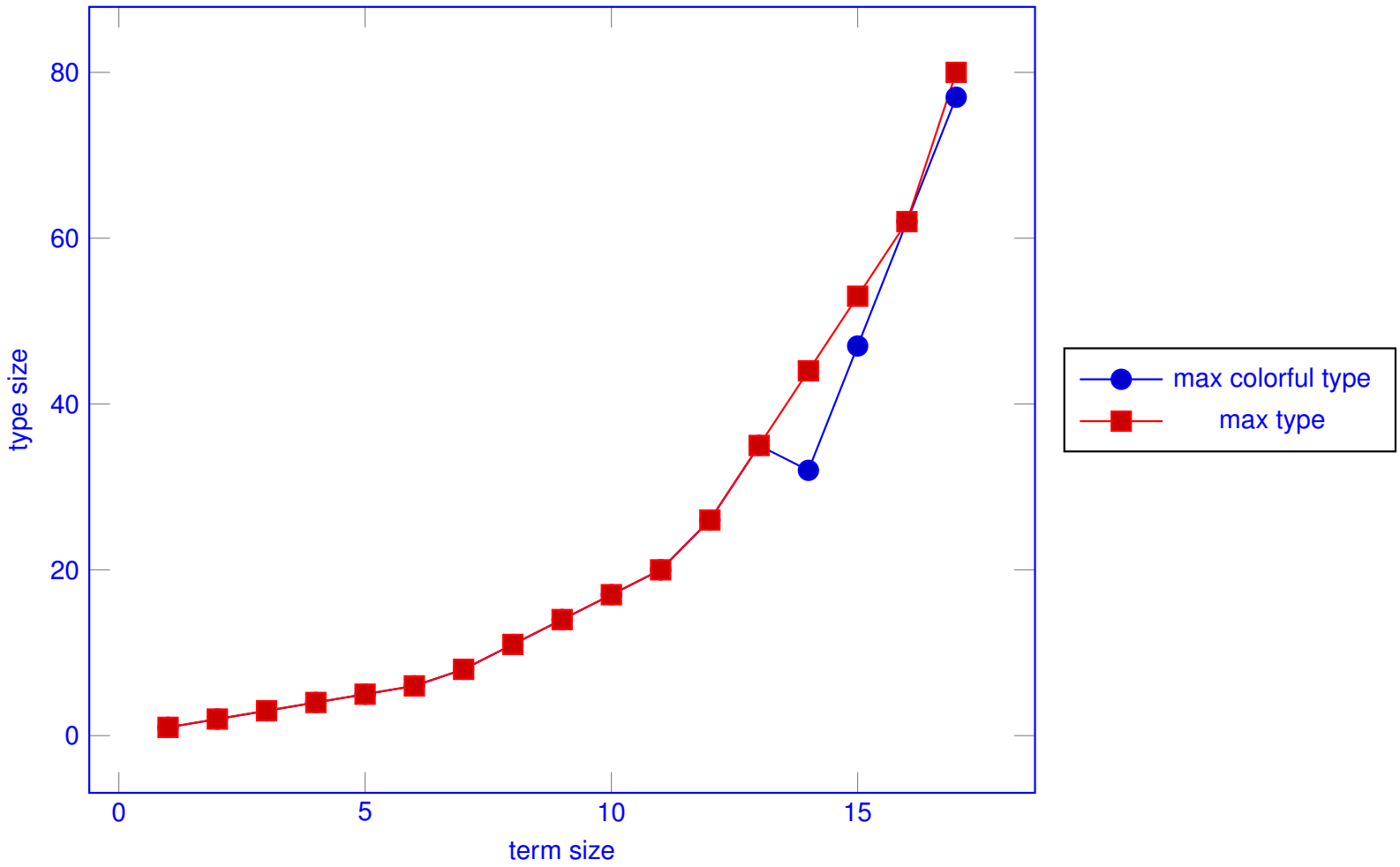


Figure: Largest type size of a most colorful term vs. largest type size

A two-stage generation algorithm for typable and untypable skeletons

Typable and untypable Motzkin skeletons

- a *typable Motzkin tree* is one for which it exists a simply-typed closed term having it as its skeleton
- a *untypable Motzkin tree* is one for which no simply-typed closed term exists having it as its skeleton
- to efficiently generate these skeletons we will split the generation of lambda terms in two stages
  - *the first stage* will generate the unification equations that need to be solved for type inference as well as the ready to be filled out lambda trees
  - it is convenient to actually generate "on the fly" the code to be executed in the second stage
  - *the second stage* will just use Prolog's metacall to activate this code

Generating the executable equation set

we generate a ready to run conjunction of unification constraints

```
genEqs(N,X,T,Eqs):-genEqs(X,T,[],Eqs,true,N,0).

genEqs(v(I),V,[V0|Vs],Es1,Es2)-->{add_eq(Vs,V0,V,I,Es1,Es2)}.
genEqs(l(A),(S->T),Vs,Es1,Es2)-->l,genEqs(A,T,[S|Vs],Es1,Es2).
genEqs(a(A,B),T,Vs,Es1,Es3)-->a,
  genEqs(A,(S->T),Vs,Es1,Es2),
  genEqs(B,S,Vs,Es2,Es3).

% solve this equation as it can either succeed once, or fail
add_eq([],V0,V,0,Es,Es):-unify_with_occurs_check(V0,V). % <==
add_eq([V1|Vs],V0,V,I,(el([V0,V1|Vs],V,0,I),Es),Es).

el(I,Vs,V):-el(Vs,V,0,I).

el([V0|_],V,N,N):-unify_with_occurs_check(V0,V).
el([_|Vs],V,N1,N3):-succ(N1,N2),el(Vs,V,N2,N3).
```

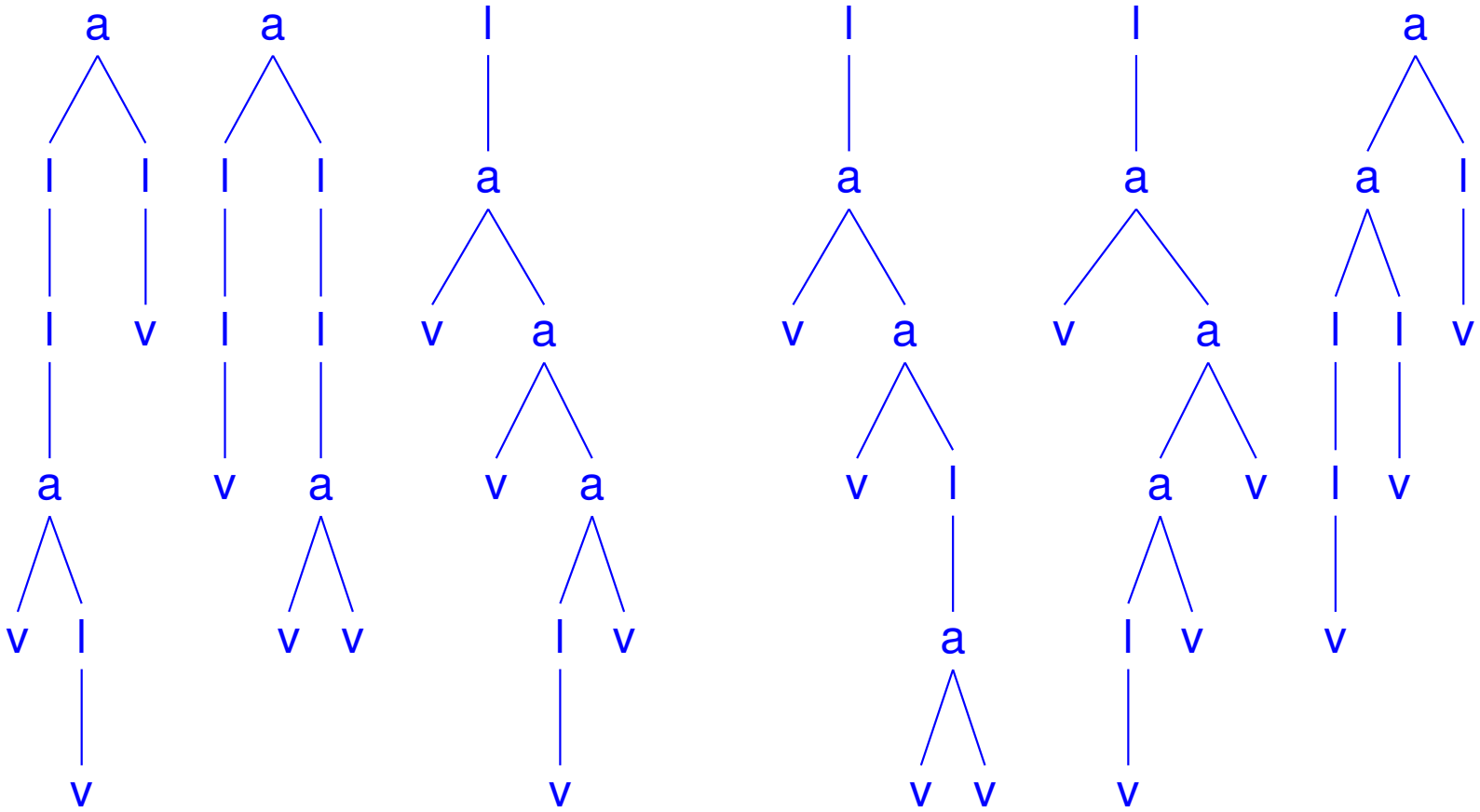
Efficient generation of typable and untypable skeletons

- we solve the equations for 2-colored terms to avoid backtracking
- Motzkin trees: `1,1,2,4,9,21,51,127,323,835,2188`
- Type equation trees: `0,1,1,1,5, 9,17, 55,122,289, 828`
- $\Rightarrow$ we can generate efficiently typeable and untypable skeletons
  - untypable: `Eqs` have no solution (their negation succeeds)
  - typable: `Eqs` have at least one solution (no need to compute all)

```
untypableSkel(N,Skel):-genEqs(N,X,_,Eqs),not(Eqs),toMotSkel(X,Skel).

typableSkel(N,Skel):-genEqs(N,X,_,Eqs),once(Eqs),toMotSkel(X,Skel).
```

3 typable and 3 untypable Motzkin trees



Growth of the number of skeletons is similar on a log-scale

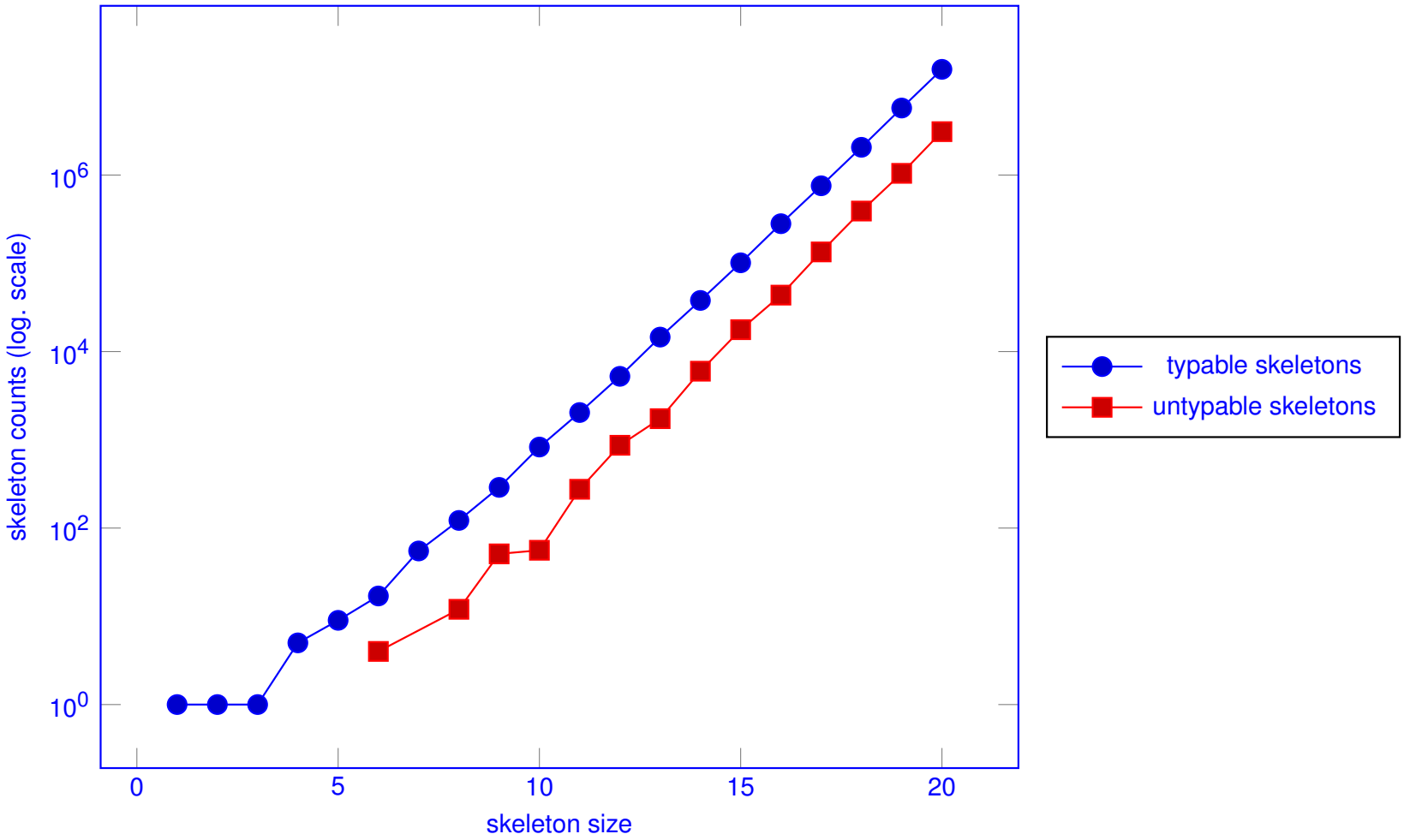


Figure: typable vs. untypable skeletons up to size 20

Counts of typable and untypable Motzkin skeletons

term size	typable skeletons	untypable skeletons
0	0	0
1	1	0
2	1	0
3	1	0
4	5	0
5	9	0
6	17	4
7	55	0
8	122	12
9	289	51
10	828	56
11	2037	275
12	5239	867
13	14578	1736
14	37942	5988
15	101307	17697
16	281041	43583
17	755726	134546
18	2062288	390872
19	5745200	1045248
20	15768207	3102275

Figure: Number of typable and untypable skeletons

Growth rates of typable and untypable skeletons

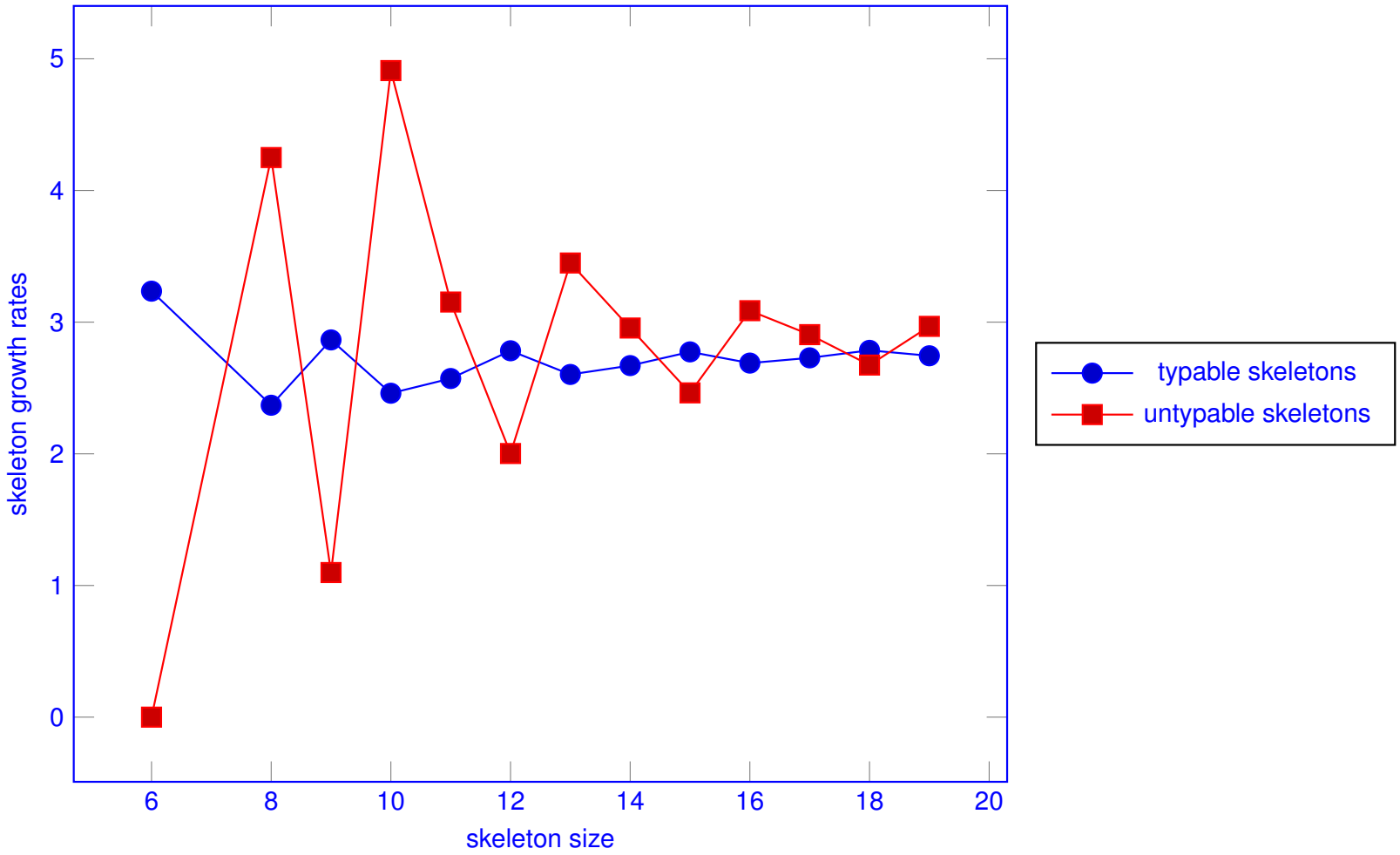


Figure: The similar growth rates of typable and untypable skeletons

Motzkin trees vs. their subset of typable skeletons

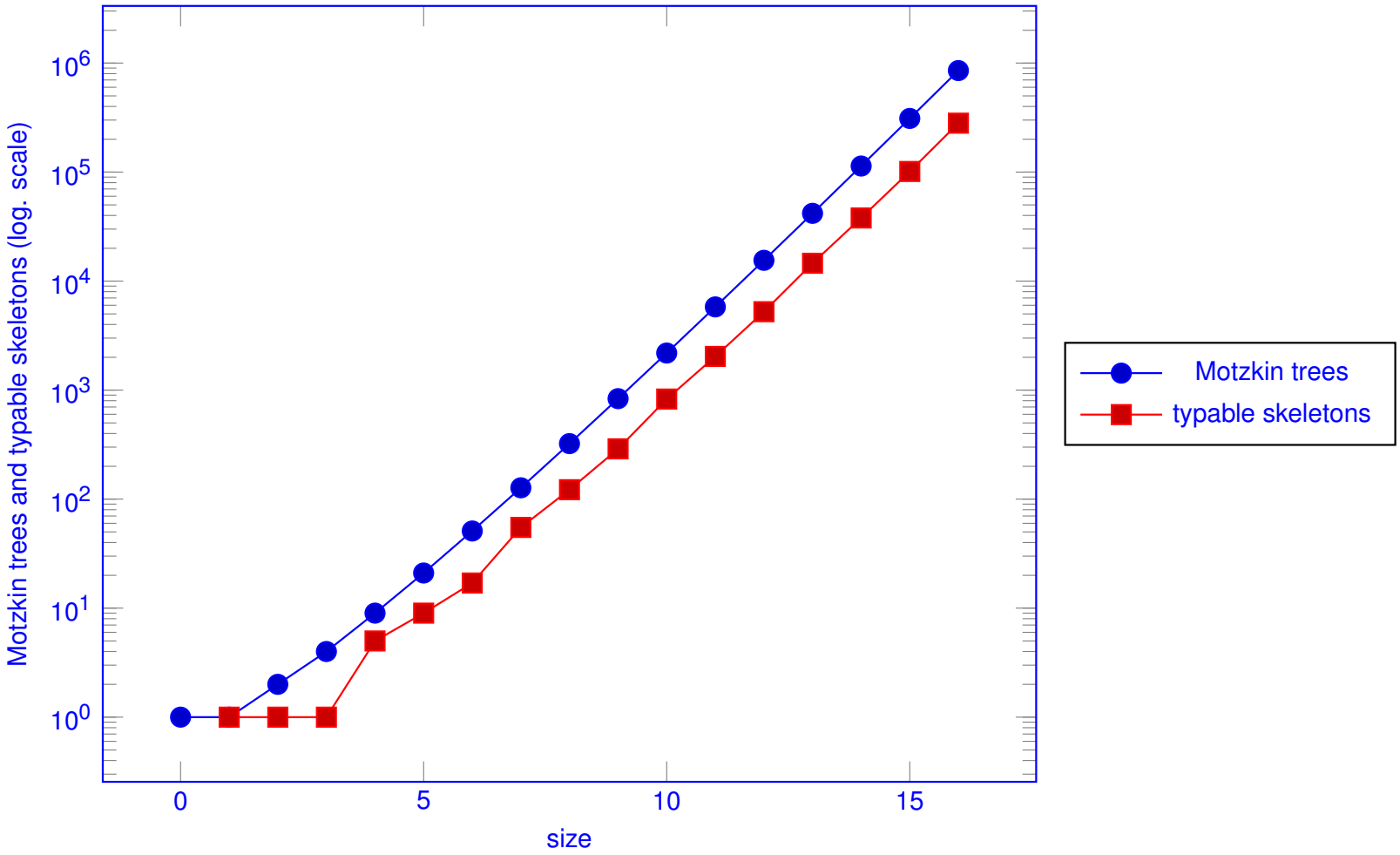


Figure: Counts for Motzkin trees and typable skeletons for increasing term sizes

Counts of simply typed closed terms vs. their skeletons

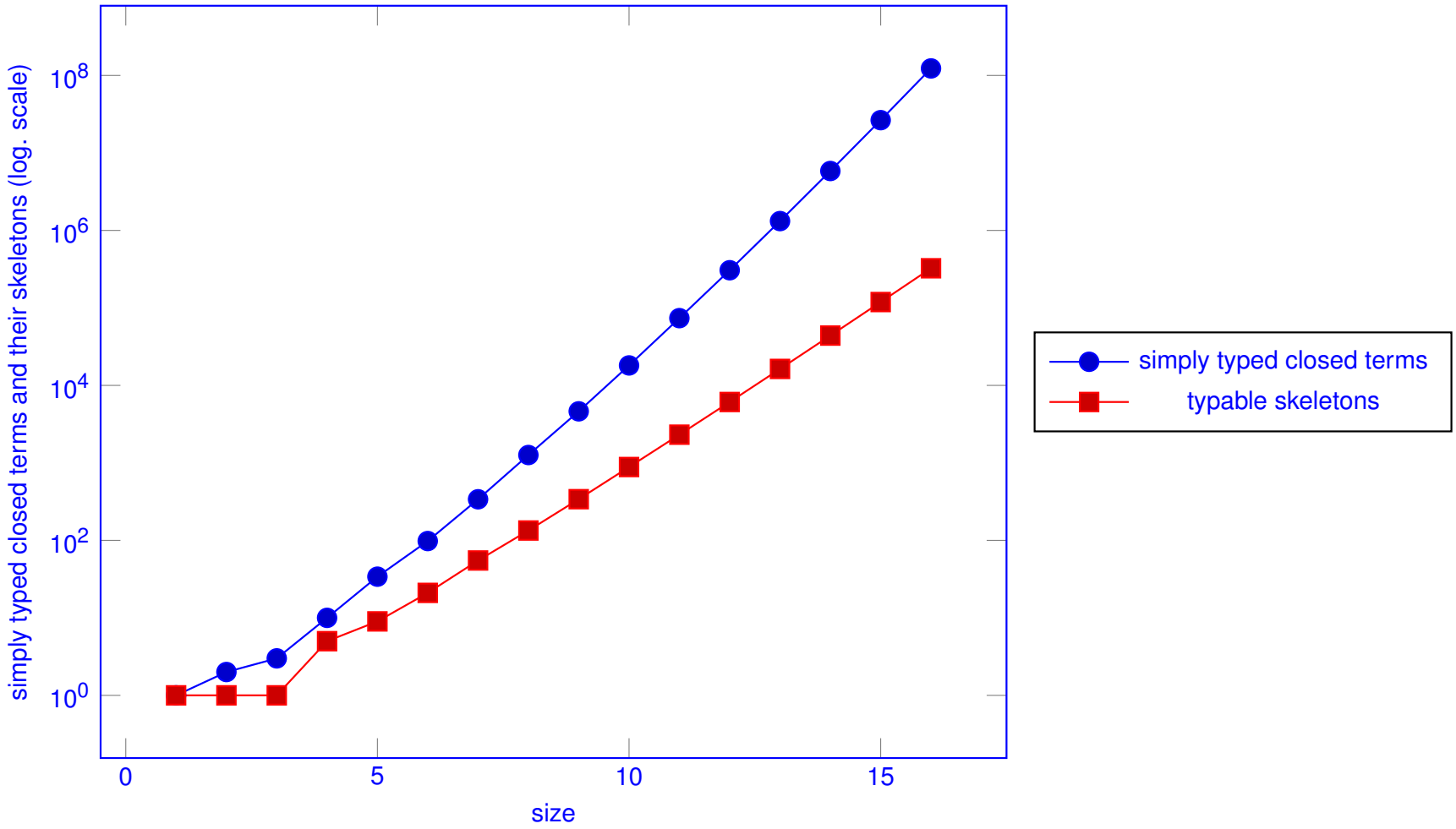


Figure: Counts of simply typed closed terms and their skeletons by increasing sizes

Some research questions for future work I

- As colors abstract numbers of occurrences of a given binder, while the length of the lambda chains in de Bruijn notation abstracts away the number of parameters of a given function, how do the distributions of these two complementary aspects of a lambda terms interact?
- What interesting empirical relations exists between colors of lambda terms and typability?
- What relations exist between colors and the complexity of type expressions?
- What are the best theoretical models that relate properties of k-colored simply-typed terms and their skeletons to properties of their type expressions?
- Can an empirical study of relations between trees representing simply typed lambda terms, their skeletons and their types reveal interesting properties of the multiple underlying formalisms ranging from computations to proofs in the implicational fragment of intuitionistic logic?

Some research questions for future work II

- What are the typical properties (size, colors, etc.) of the lambda expressions generated during compilation of actual functional programs?
- Can we, by controlling the relative weight of the colors fine-tune all-term and random term generators to reflect the distributions observed for lambda terms used as intermediate code in compilers for functional languages?
- Do bijections between simpler families of combinatorial objects, lambda terms in de Bruijn notation, their normal forms and their skeletons provide means for generating very large random lambda terms while allowing to selectively fine-tune the distribution of their constructors?
- Can such bijections help pushing further the size of random simply typed terms and normal forms generated via new sequential and parallel algorithms?

Conclusions

- we have devised abstraction mechanisms that “forget” properties of the difficult class of simply-typed closed lambda terms to reveal equivalence classes that are likely to be easier to grasp with analytic tools
- some of our findings:
  - bijection between 2-colored terms and binary trees
  - simple algorithm for generating affine and linear terms
  - efficient generation of typable and untypable skeletons, with potential uses as lemmas in dynamic programming algorithms
- k-colored terms are likely to be usable to fine-tune random generators to more closely match “color-distributions” of lambda terms representing real programs
- the tools used: a language as simple as (mostly) Horn Clause Prolog can handle elegantly combinatorial generation problems when the synergy between sound unification, backtracking and DCGs is put at work!

Questions?