[a c d] a c d
[-c e f] a d e f
[-d g h] a c f g h
[-e i j]
[-f k l]
[-g m n]
[-h o p] a i j k l m n o p
[-i r q] a r j k l m n o p q
[-q b s] a b j k l m n o p r s
[-r t u] a b j k l m n o p s t u
[-s v w] a b j k l m n o p t u v w
[-w u t no v -s r -q or w in -v clause
-v p n
-u k m
-t n m
-p o n
-o l j
-n m l
-m k l
-l j k
-k a b or -k a j or [-k b j]
[-j a b]

[-e j r q] [-i r b s]
[-e i a b]

[a b v] [a b -v] [b j -l]
[a b -k] [a b k -v]
[a b -j] [-v k j b]
[-o j b] [-v j k o]
[-o j l] [b j -l]
[b j -k] [j k -l]

In order to stop a traversal, a clause [-c   ] cannot
share terms w/ the clauses w/
[c x y]
[-x z w]
[-y u v]
Otherwize [-c - -] could pair w/ [c x y] to make [- - x y]
and if - - is z, w, u, v x or y it could combine again w/o
going over 4 terms
If x ∈ [c - -], [- y z w] ~~$\times$~~ ✓
If z ∈ [c - -] → [z y w -]
w ∈ → [z y w -]
If y ∈ [c - -] → [x - u v]
u → [x - u v]
v → [x - u v]
If y or x ∈ [c - -] → [x - y] → [- y z u] and ~~or~~ [- x u v]

To block traversal from [a c d],
~~[-c u v] cannot have a or d and~~
~~[-d w x] a or c and~~
~~[v - -]~~

Consider traversal from [a c d] on the left page
[a d e f] and [a c g h]
Can continue if a, d, or e ∈ [-f ...]
a, e, or f ∈ [-d ..]
a, d, or f ∈ [-e.]
a, c, or g ∈ [-h ...]
a, c or h ∈ [-g ..]
a, g or h ∈ [-c.]

abei -von

-kab ·swon

ab-fl

ab-um

ab-ml

ab-ul

ab-ol

ab-lh

ab-l

ab-fk

ab-oj

ab-nm

ab-mk

abm-t

ab-tk

ab-tj

ab-t

abu-w

ab-ru

-irbs

-svut

-wrkm

-wnkm

-vpml

If we traverse the graph starting from a random clause, what will stop the traversal?

The cluster has no common terms w/ its neighbors

Which means...

The cluster we're working with is unlike the original large clause in which that clause popped a term.

I think what we're trying ~~to~~ to do here is break up/reverse parts of the original clause to never deal w/ a clause larger than 3 terms.

How can we prove such a transformation is always possible?

Whenever you pop a term, you are left w/ a 2-, 3-, or 4-t clause.

In this clause, there are 1-4 terms which are also popped

Looking at the clauses containing the negatives of those 1-4 terms, we may add that clause to the cluster if at least one term overlaps.

So what? There may be no overlap.

We know the (new) ab output must happen with

[a b X]

[a b -X] where X is some OFT

Now we have two targets to hit. How can we guarantee we won't exceed 4-t?

How could the inputs to [a b X] look?

[a b Y] [-Y X a]

[a X Y] [-Y b a]

[a X Y] [-Y b X]

[a b X Y] [a b X -Y]

[a b X Y] [-Y X a]

[a b X Y] [-Y a b]   The -Y that pops always carries (X, a, or b)

Always carries a term yet to be popped

[ab]
[ab-v] [abv]
[ab-k] [abk-v]
•[bj-k] •[ab-j] •[ab-j] [bjk-v]
[bj-o] [jko-v]
•[jl-o] [bj-l] [jk-n] [no-v]
•[bj-k] [jk-l] [jk-l] [kl-n] •[np-v] •[no-p]
•[kl-m] •[lm-n]

[abv]
[ab-r] [abrv]
[ab-k] [abk-r]
•[bj-k] [ab-j] •[ab-j] [bjk-r]
[bjk-m] [jkm-r]
•[jk-l] [bjl-m] [jkm-t] [jmt-r]
•[kl-m] [bj-k] [mn-t] [jk-n] •[km-v] [tv-r]
•[jk-l] [kl-n]
•[kl-m] [lm-n]

[abrv]
[ab-sv] [abrs]
•[vw-s] [ab-w]
[ab-k] [abk-w]
•[bj-k] •[ab-j] •[ab-j] [bjk-w]
[bjk-m] [jkm-w]
•[jk-l] [bjl-m] [jkm-t] [jkt-w]
•[kl-m] [bj-k] [mn-t] [jk-n] •[jk-l] [klt-w]
•[jk-l] [kl-n]
[kl-m] [lm-n]
•[kl-m] [kmt-w]
•[km-v] [tv-w]

Observations:

- For each clause, there were two inputs composed of an OFT + the output's terms
- Each clause's inputs have two rather separate branches
- Branches terminate when they reach a given 3-t clause
- ~~None of the following terms:~~

Oops, missed the abrs branch

[a b r s]

[b r s -i] [a b i]

•[b s -q] [q r -i] •[a b -j] [a b i j]

[a b j -n] [a b i n]

[a b l -n] [b j -l]   [a b n -h] [a b h i]

•[a b -j] [b j l -n]   [b j -k] [j k -l]   •[a b -j] [b j n -h]

•[b j -k] [k l -n]   [j l n -h] [b j -l]

•[k l -m] [l m -n]   •[j l -o] [n o -h] •[b j -k] •[j k -l]

•[n o -p] •[o p -h]

[a b h i]

[a b c h] [a b i -c]

[a c g h] [a b -g]

•[g h -d] [a c d]   [a b -l] [a b l -g]

•[a b -j] [b j -l]   [a b l -n] [a b n -g]

•[b j -k] [j k -l]   •[a b -j] •[b j l -n]   •[m n -g] [a b -m]

•[b j -k] [k l -n]   •[a b -j] [b j -m]

•[k l -m] •[l m -n]   •[b j -k] [j k -m]

•[k l -m] •[j k -l]

[a b i -c]

•[a b -j] [b i j -c]

[i j k -c] •[b j -k]

•[i j -e] [e j k -c]

•[e f -c] [j k -f]

•[j k -l] [k l -f]

Observations:

- Every clause w/ ab has [a b -j] somewhere in its grand*-children
- ~~Consider an implication like [A B C], ([A B X][C A -X]), it seems that along the X branch, you will never see a -X and along the -X branch, you will never see a X~~ incorrect
- Every partner of [a b -j] has a or b
- How can it be shown that [ab] can be derived from the given clauses without processing a clause whose length exceeds 4 terms?
- Similarly to how the shrinking clauses contain the popped term + two terms yet to be popped, moving up a branch, clauses contain the popped term + two other terms in the cluster.
- Every derived clause represents the current cluster
- Sometimes you must derive multiple clusters before continuing traversal

---

Starting at [-j a b] WTS we can get [-x a b] ∀ x in the shrinking clauses

1. [-k _ _] is either
   1) [-k a b] → have [-k a b] ✓
   2) [-k a j] or [-k b j] → imply w/ [-j a b] → [-k a b]
2. [-l _ _] contains 2 terms from {a, b, j, k}
   and we know [-k a b] and [-j a b] exist
   if j or k ∈ [-l X Y], we imply [-l a b Y] where Y is in {a b j k} and is not X
   now we either have [-l a b] or can imply it in one step
3. This is true for all shrinking clauses (TODO: proper proof w/ PMI)

Next, we come across either a clause like (1) [-x y z] where y and z are the initial placements of the positive form of the popped terms or (2) [-x a z]
(3) [-x a b] (4) [x a b]

In case (1), we can derive [-x a b] since [-y a b] and [-z a b] exist
(2), we got [-x a b] since [-z a b] exists
(3) we have [-x a b]
(4) we know [x] must be popped to find [-x _ _] and it will fall into one of the 3 categories

Last part: We know the initial large clause is seeded w/ some positive terms and a, b, or both.
Therefore there must be a clause that pops these terms.
ie, there is a clause like [-x y z] where x is one of the positive terms in the seed and y, z are other terms (could be a or b)
On the last page, it is shown we can acquire a clause like [-x a b] for every term, x, that's popped
So we can pop the seeded terms and derive [a b] w/o exceeding a clause of length 4.