Rensselaer
why not change the world?®

Parallel Verification of Natural Deduction Proof Graphs

James Oswald and Brandon Rozek {oswalj,rozekb}@rpi.edu
July 2nd 2023

Problem Statement

1. How do we efficiently verify large natural deduction proofs?
2. To this end, is there a way that we can make use of parallelism?

Related Work

[Färber, 2022] breaks up a command for proof checking inside the lambda-Pi calculus modulo rewriting into four tasks: parsing, sharing, type inference, type checking.

Similarly in this work, we'll at splitting verification of a natural deduction step into two tasks: **syntax verification** *and* **assumption checks***.*

Natural Deduction

A logic calculus independently proposed by [Gentzen, 1935, Jaśkowski, 1934] in an effort to emulate human-level reasoning through assumptions and chains of inference.

$$\frac{}{\{\phi\} \vdash \phi}\ \mathsf{A} \qquad \frac{\Delta \vdash \psi \lor \phi \quad \Gamma \cup \{\psi\} \vdash \chi \quad \Sigma \cup \{\phi\} \vdash \chi}{\Delta \cup \Gamma \cup \Sigma \vdash \chi}\ \lor E$$

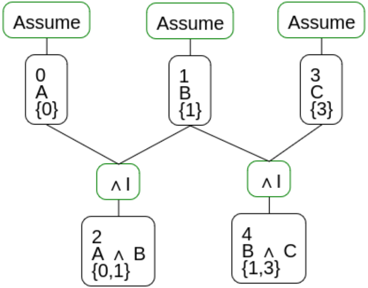
$$\frac{\Gamma \cup \{\phi\} \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} \rightarrow I \qquad \frac{\Gamma \vdash \phi \quad \Sigma \vdash \phi \rightarrow \psi}{\Gamma \cup \Sigma \vdash \psi} \rightarrow E \qquad \frac{\Gamma \cup \{\phi\} \vdash \psi \quad \Sigma \vdash \neg\psi}{\Gamma \cup \Sigma \vdash \neg\phi}\ \neg I$$

Figure: Partial Collection of Inference Schemas for Natural Deduction

Compact Representation

Within a hypergraphical representation of a natural deduction proof:

- Formula and assumptions are stored on the node.
- Justification is stored on its incoming hyperedge.

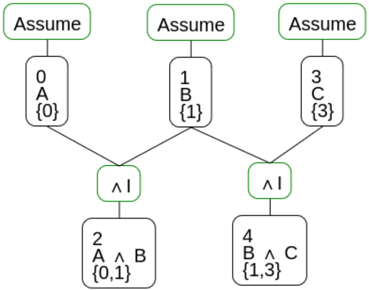


This representation allows us to compactly represent **multiple proofs** and easily **reuse subproofs**.

Graphical Interactive Theorem Provers

When building an interactive theorem prover, we want to alleviate as much burden as possible.

Both [Bringsjord et al., 2022, Oswald and Rozek, 2022] have the user not specify the assumptions used within each step.



Multiprocessing

This work uses the *shared memory model* for multiprocessing. We'll instantiate a fixed number of threads and have them operate over the same memory space.

We need a way to systematically assign nodes in our hypergraphical representation to each thread to verify.

Approach

Layering

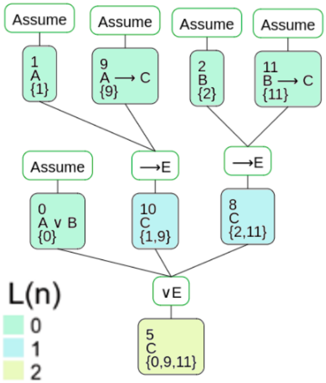
- To make use of parallelism, we want to find groups of nodes that we can verify at the same time.
- A node's assumptions are dependent on its ancestors.

We define a node n to be on layer

$$L(n) = \begin{cases} 0 & \text{if } n \text{ is an assumption} \\ 1 + \max_{m \in P(n)}(L(m)), & \text{otherwise} \end{cases} \tag{1}$$

where $P(n)$ maps a node to its parents.

Layering Example



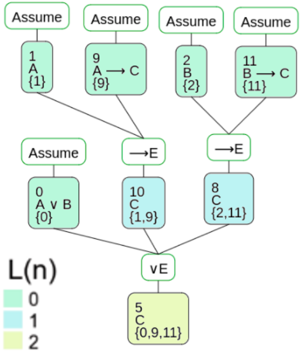
Layer	Nodes
0	0 1 9 2 11
1	10 8
2	5

Serial Example

On Layer 0, we have the following nodes:

$$\{A, A \to C, B, B \to C, A \vee B\}$$

- These trivially verify as they're assumptions
- Each node's assumption multiset will consist of its own formula.

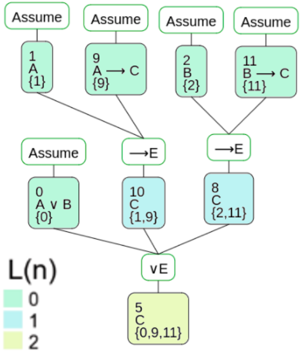


Serial Example

On Layer 1, we have the following nodes:

$$\{C_{10}, C_8\}$$

- They're both justified by conditional elimination so there's no special assumption constraint.
- Each node's assumption multiset is the union of their two parent's assumptions.

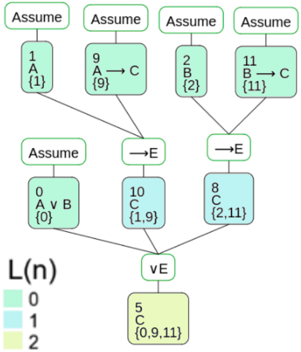


Serial Example

On Layer 2, we have the following node:

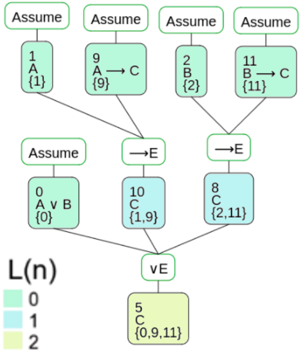
$$\{C_5\}$$

- It's justified by disjunction elimination, so it requires that each disjunct is used as an assumption.
- The resulting assumption on C is the union of the three parent multisets minus the disjunct assumptions.



Serial Example

Since we got through the entire proof without any verification failures, the entire proof graph has been verified.



Serial Version of the Algorithm

```
procedure VERIFY(ProofGraph p)
    Create set of nodes on each layer using Equation 1 and store in layerMap.
    for layerNodes in layerMap do
        for n in layerNodes do
            ruleInfo = (m, assumptions(m)) ∀ m ∈ parents(n)
            if not is_valid(n, justification, ruleInfo) then
                return false
            Update assumptions(n) using the justification and ruleInfo.
    return true
```

Parallel Algorithms

Parallel Algorithms

We consider a base parallel algorithm implementation and two optimizations:

- Static Load Balancing
- Syntax First

Simple Parallel

Simple Parallel

We iterate over each layer as with serial but verify each node in a given layer in *parallel*.

- We'll also include some additional data structures to address issues with *thread safety*.

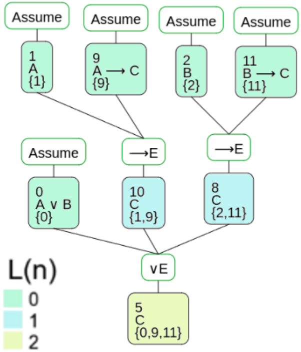
Simple Parallel Example

Recall on Layer 0, we have the following nodes:

$$\{A, A \rightarrow C, B, B \rightarrow C, A \vee B\}$$

Assuming we have three threads:

- Thread 0 will be assigned $A, A \rightarrow C$
- Thread 1 will be assigned $B, B \rightarrow C$
- Thread 2 will be assigned $A \vee B$

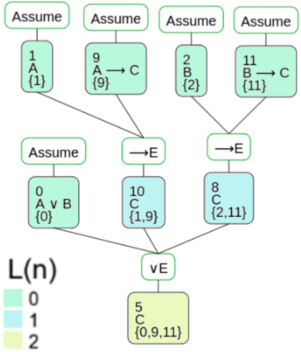


Simple Parallel Example

From the perspective of thread 0, we need to verify

$$\{A, A \rightarrow C\}$$

- After verification, calculate the assumptions for each of the nodes and update its appropriate entry in the shared memory vector.
- Repeat for the remaining nodes
- Wait for all other threads to finish
- Move to the next layer



Optimization: Static Load Balancing

Static Load Balancing

Recall the assignment from the last algorithm:

- Thread 0 will be assigned $A, A \to C$
- Thread 1 will be assigned $B, B \to C$
- Thread 2 will be assigned $A \vee B$

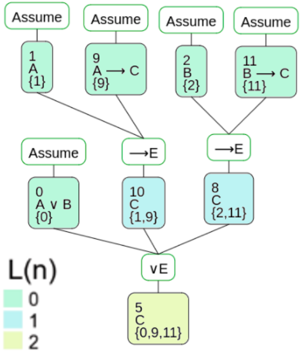
Notice how thread 2 has one less item to verify. This version of the algorithm attempts to balance this out by *syntax verifying* a node on the next layer.

Static Load Balancing

The new assignment looks like:

- Thread 0 will be assigned $\neg A, \neg B$
- Thread 1 will be assigned $A, A \vee B$
- Thread 2 will be assigned $B, (C_{10})_s$

where $_s$ denotes a syntax only check.



Syntax First Approach

Syntax First Approach

We mentioned that syntax checks can happen outside the layering structure, so why don't we perform syntax verification over all nodes in parallel first?

Algorithm Sketch:

- For every node in parallel, check syntax.
- If none fail, follow the simple parallel algorithm except instead of a full verification, we're only checking the assumption constraint.

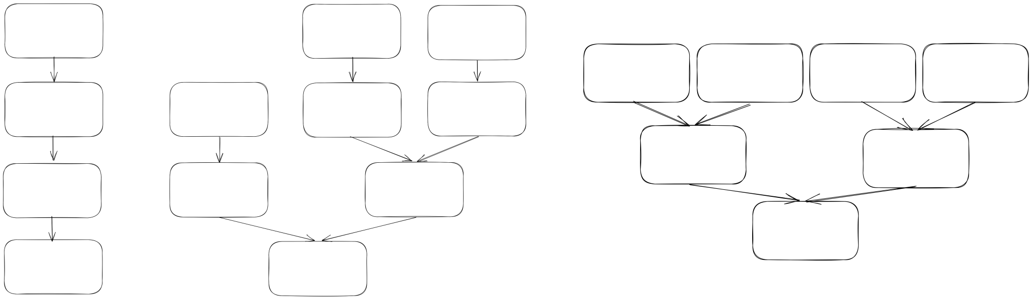
Analysis

Dataset

We're not aware of any large repository of hypergraphical natural deduction proofs to evaluate our approaches over.

Therefore, we designed a synthetic dataset. Taking inspiration from network design, we call this dataset *Directed Acyclic Network Topologies* or (DANTs).

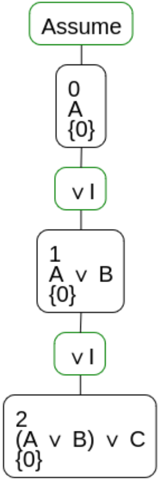
Three Case Studies



Straight, Branch, and Tree Topologies

Straight Topology (n)

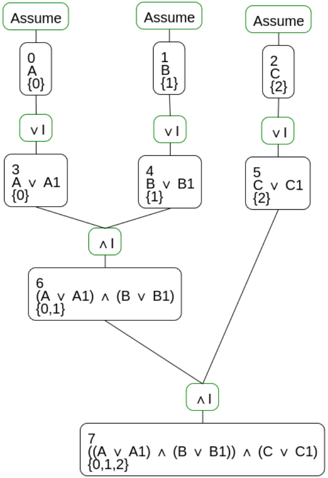
For the straight topology, we apply n disjunction introductions.



Parallel Branches (b, n)

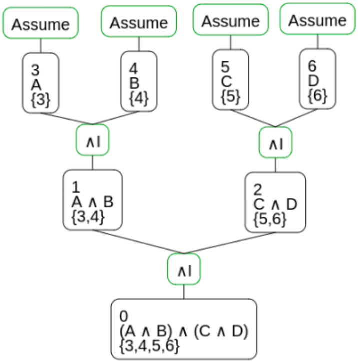
This topology emulates multiple lines of independent reasoning before combining towards the end.

- It starts off with b separate assumptions
- performs a disjunctive introduction on each assumption n times
- then iteratively applying conjunctive introduction to each branch until there's one remaining.



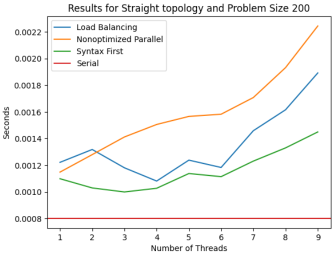
Tree (h)

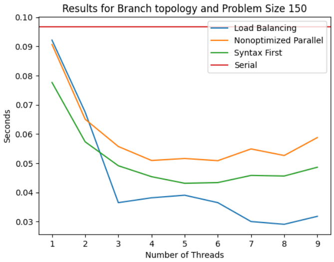
- In this topology we generate 2^h assumptions and iteratively apply conjunction introduction h times until we reach a single node.
- This creates a balanced binary tree.

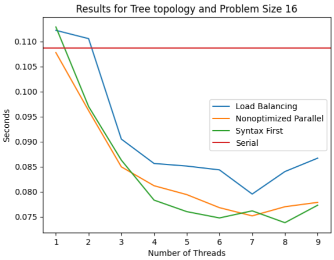


Strong Scaling Study

Performance analysis varying the number of threads over a constant problem size.

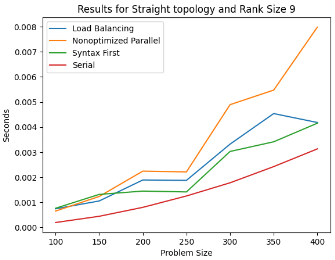


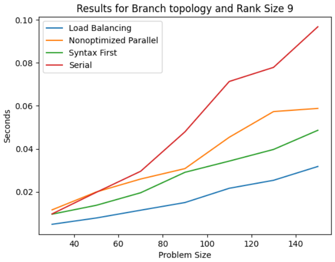


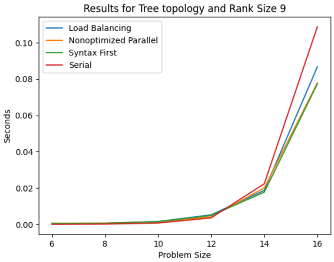


Weak Scaling Study

Performance analysis varying problem size over a constant number of threads.







Future Work

1. Perform an Amdahl's Law analysis to calculate the overall speedup factor compared to the serial version.
2. Test on randomized proof topologies.
3. Extend the logics support to first-order and modal.
4. Scale beyond a single computer with message-based parallelism.

Thanks for attending! Any Questions?

Bibliogaphy I

Bringsjord, S., Govindarajulu, N. S., Taylor, J., and Bringsjord, A. (2022).
Logic: A Modern Approach.

Färber, M. (2022).
Safe, fast, concurrent proof checking for the lambda-pi calculus modulo rewriting.
Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs.

Gentzen, G. (1935).
Untersuchungen über das logische schließen. i.
Mathematische Zeitschrift, 39:176–210.

Jaśkowski, S. (1934).
On the rules of suppositions in formal logic.

Bibliogaphy II

Oswald, J. and Rozek, B. (2022).
Lazyslate.