Navigating AND–OR Graph Modifications to Debug Failing Proof Search

Justin Lubin, Marlena Preigh, Max Willsey, Sarah E. Chasins

PLDI 2026

Proof search is ubiquitous and powerful

Proof search is ubiquitous and powerful

Databases
(Datalog)

Proof search is ubiquitous and powerful

Databases (Datalog)

Program synthesis

Proof search is ubiquitous and powerful

Databases (Datalog)

Program synthesis

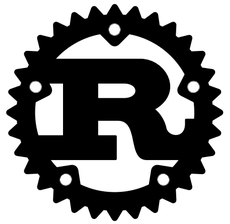
Type systems

Proof search is ubiquitous and powerful

Databases (Datalog)

Program synthesis

Type systems



Traits

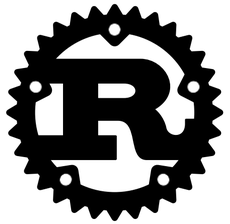
Proof search is ubiquitous and powerful

Databases (Datalog)

Program synthesis

Type systems

Interactive theorem provers



Traits

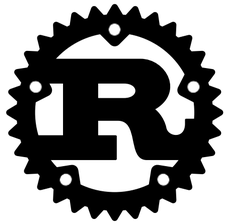
Proof search is ubiquitous and powerful

Databases (Datalog)

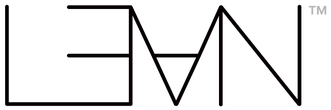
Program synthesis

Type systems

Interactive theorem provers



Traits



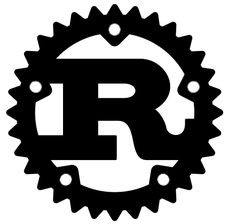
Aesop

What happens when proof search fails?

Databases (Datalog)

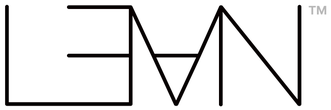
Program synthesis

Type systems



Traits

Interactive theorem provers



Aesop

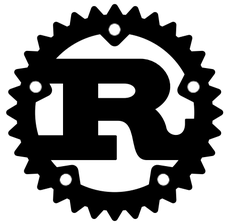
What happens when proof search fails?

Databases (Datalog)

```
Check failed:
(edge 1 3)
```

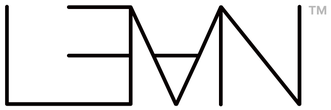
Program synthesis

Type systems



Traits

Interactive theorem provers



Aesop

What happens when proof search fails?

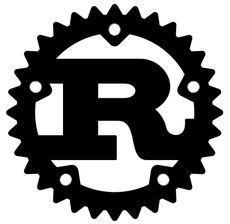
Databases (Datalog)

```
Check failed:
(edge 1 3)
```

Program synthesis

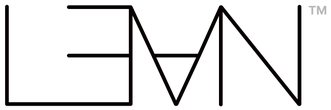
```
Not possible!
```

Type systems



Traits

Interactive theorem provers



Aesop

What happens when proof search fails?

Databases (Datalog)

Program synthesis

Type systems

Interactive theorem provers

```
Check failed:
(edge 1 3)
```

```
Not possible!
```

```
error[E0599]: the method `clone` exists for enum `top_down::Sketch<F>`, but its trait bounds were not satisfied
   --> src/top_down.rs:205:40
    |
 47 | pub enum Sketch<F> {
    | ------------------ method `clone` not found for this enum because it doesn't satisfy `top_down::Sketch<F>: Clone`
...
205 |                     *parent = Some(res.clone());
    |                                        ^^^^^ method cannot be called on `top_down::Sketch<F>` due to unsatisfied trait bounds
    |
note: trait bound `F: Clone` was not satisfied
   --> src/top_down.rs:46:17
    |
 46 | #[derive(Debug, Clone, PartialEq, Eq, Deserialize, Serial...
    |                 ^^^^^ unsatisfied trait bound introduced in this `derive` macro
note: the method `clone` exists on the type `usize`
   --> /Users/jlubin/.rustup/toolchains/stable-aarch64-apple-darwin/lib/rustlib/src/rust/library/core/src/clone.rs:236:5
    |
236 |     fn clone(&self) -> Self;
    |     ^^^^^^^^^^^^^^^^^^^^^^^^
help: consider restricting the type parameter to satisfy the trait bound
    |
 47 | pub enum Sketch<F> where F: Clone {
    |                    ++++++++++++++

error[E0308]: mismatched types
   --> src/top_down.rs:232:37
    |
232 | ...) => Some(IndexMap::from([(*h, ground.clone())])),
    |         ---- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ expected `IndexMap<usize, Sketch<F>>`, found `IndexMap<usize, &Sketch<F>>`
    |         |
    |         arguments to this enum variant are incorrect
    |
    = note: expected struct `IndexMap<_, top_down::Sketch<_>>`
               found struct `IndexMap<_, &top_down::Sketch<_>>`
help: the type constructed contains `IndexMap<usize, &top_down::Sketch<F>>` due to the type of the argument passed
   --> src/top_down.rs:232:32
    |
232 | ...=> Some(IndexMap::from([(*h, ground.clone())])),
    |       ^^^^^______________________________________^
    |            |
    |            this argument influences the type of `Some`
note: tuple variant defined here
   --> /Users/jlubin/.rustup/toolchains/stable-aarch64-apple-darwin/lib/rustlib/src/rust/library/core/src/option.rs:608:5
    |
608 |     Some(#[stable(feature = "rust1", since = "1.0.0")] T),
    |     ^^^^

error[E0369]: binary operation `==` cannot be applied to type `&F`
   --> src/top_down.rs:236:26
    |
236 |                     if f == g {
    |                        - ^^ - &F
    |                        |
    |                        &F
    |
help: consider restricting type parameter `F` with trait `PartialEq`
    |
123 | impl<F: std::cmp::PartialEq> Sketch<F> {
    |       +++++++++++++++++++++

Some errors have detailed explanations: E0308, E0369, E0599.
For more information about an error, try `rustc --explain E0308`.
error: could not compile `honeybee` (lib) due to 11 previous errors
```

One missing trait impl.

LEAN™

Aesop

What happens when proof search fails?

Databases (Datalog)

Program synthesis

Type systems

Interactive theorem provers

`Check failed: (edge 1 3)`

`Not possible!`

3/11 errors shown here

One missing trait impl.

LEAN

Aesop

What happens when proof search fails?

Databases (Datalog)

Program synthesis

Type systems

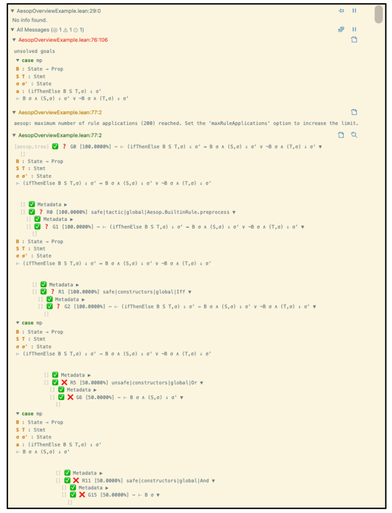
Interactive theorem provers

```
Check failed:
(edge 1 3)
```

```
Not possible!
```

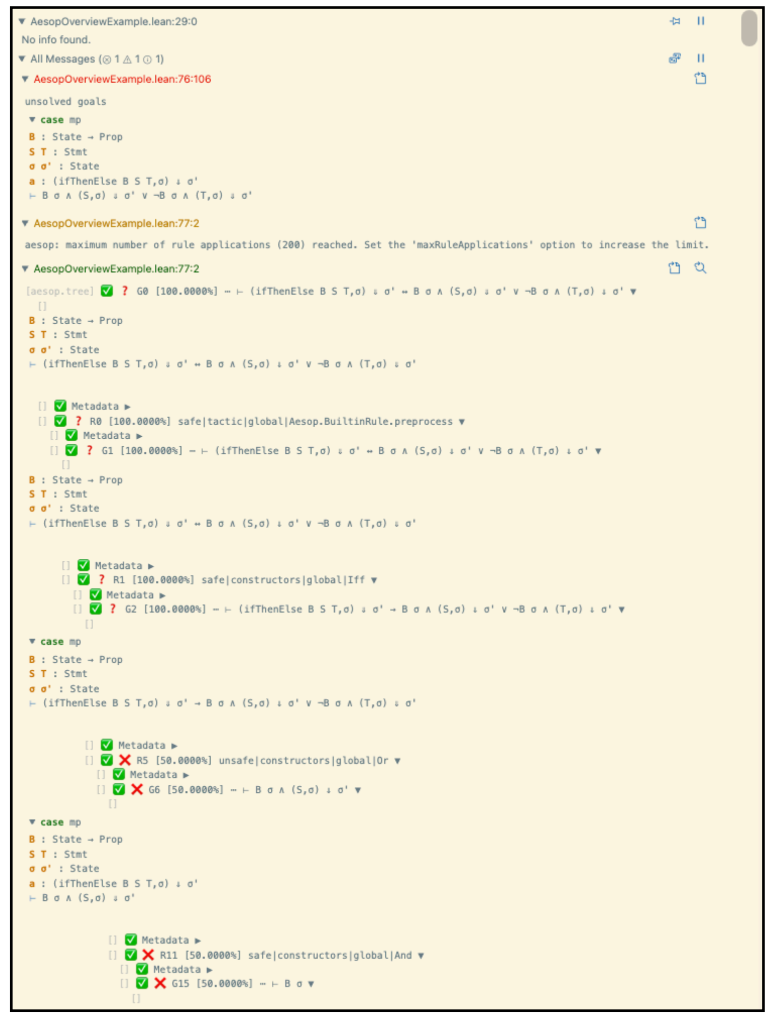
3/11 errors shown here

One missing trait impl.



One missing lemma

What happens when proof search fails?

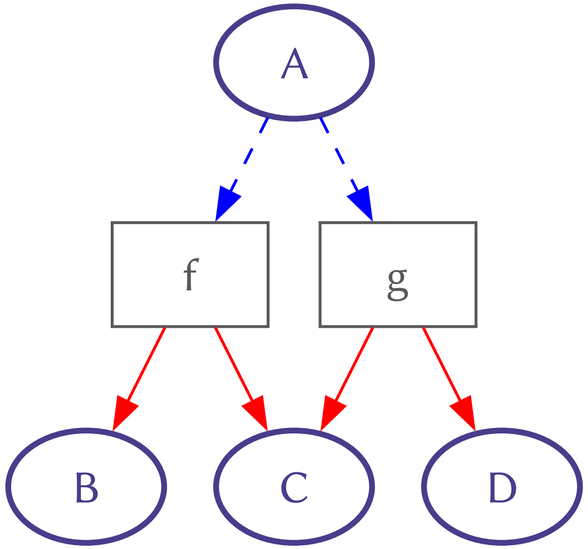
Databases (Datalog)	Program synthesis	Type systems	Interactive theorem provers
`Check failed: (edge 1 3)`	`Not possible!`		
		One missing trait impl.	One missing lemma

3/11 errors shown here

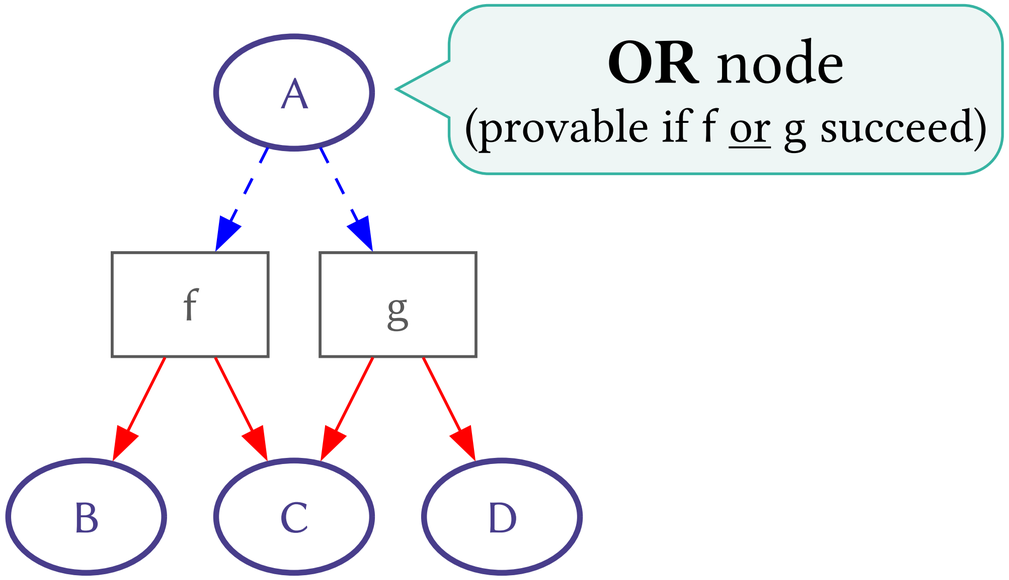
tiny % of trace

We can represent the proof search trace using an AND–OR graph

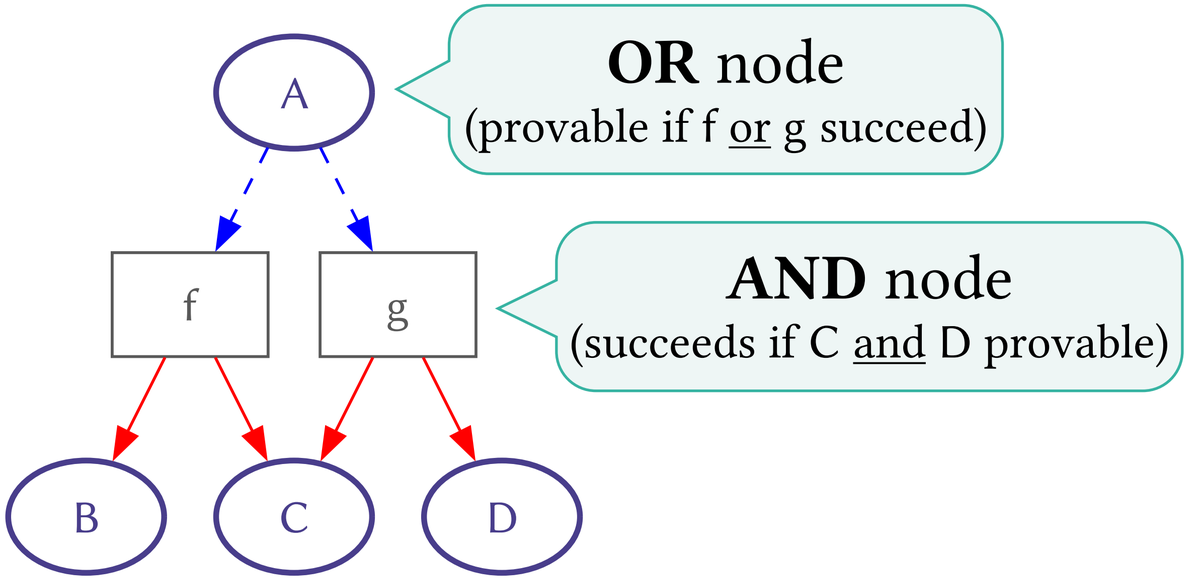
We can represent the proof search trace using an AND–OR graph



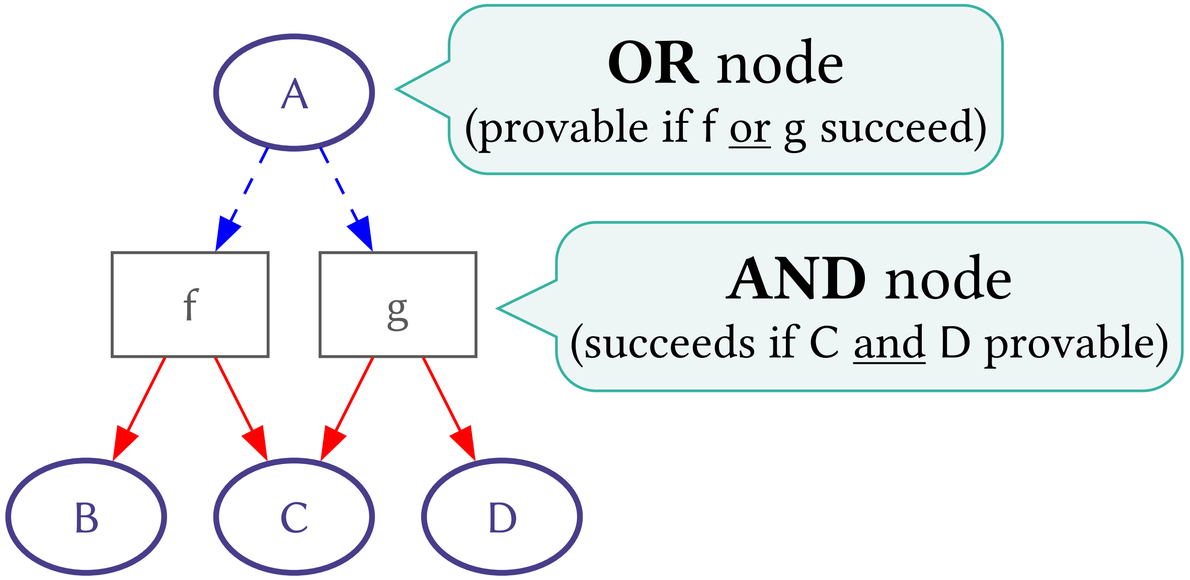
We can represent the proof search trace using an AND–OR graph

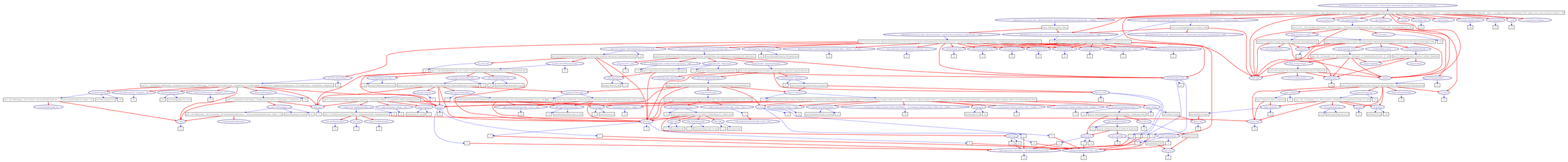


We can represent the proof search trace using an AND–OR graph



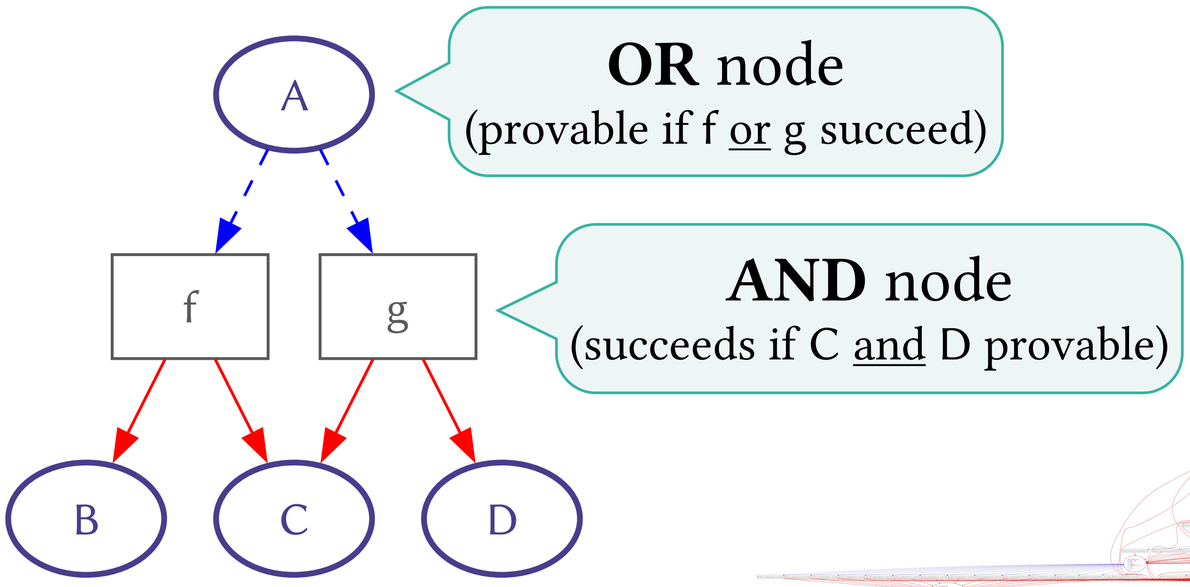
We can represent the proof search trace using an AND–OR graph

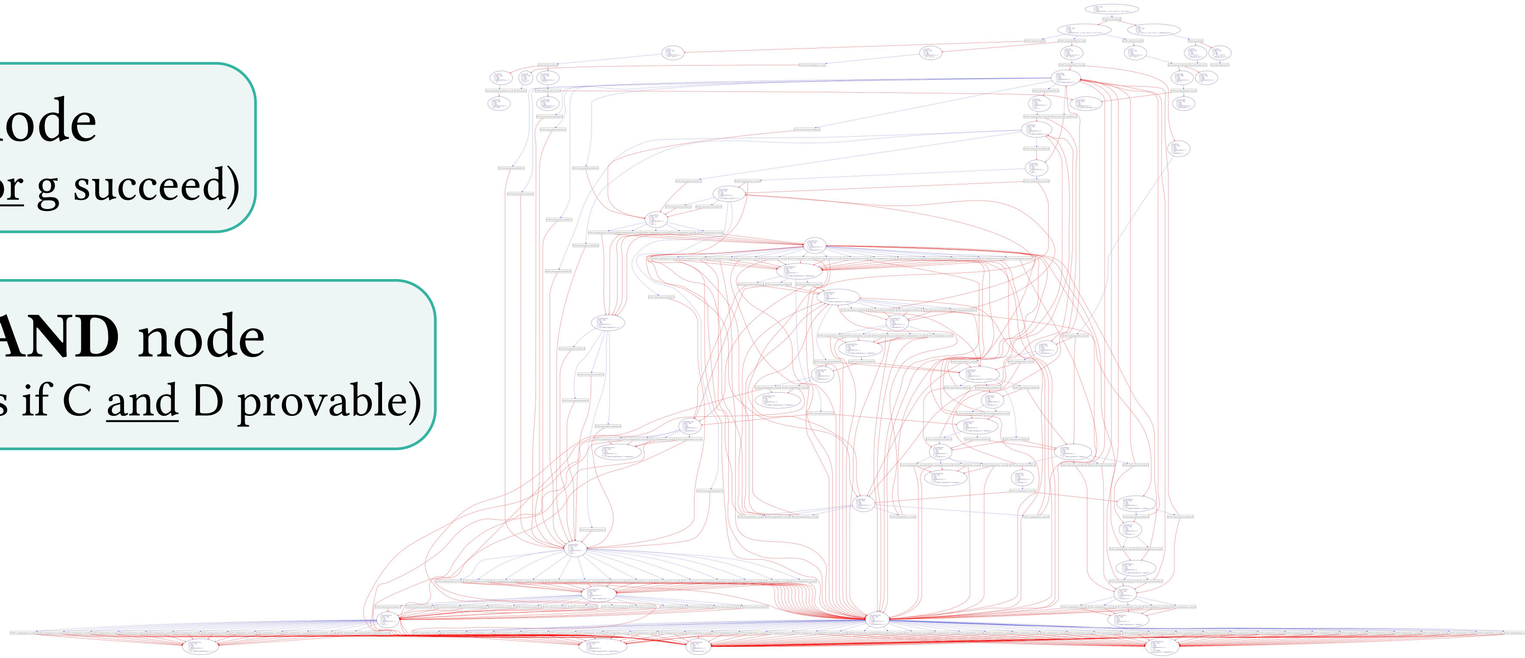


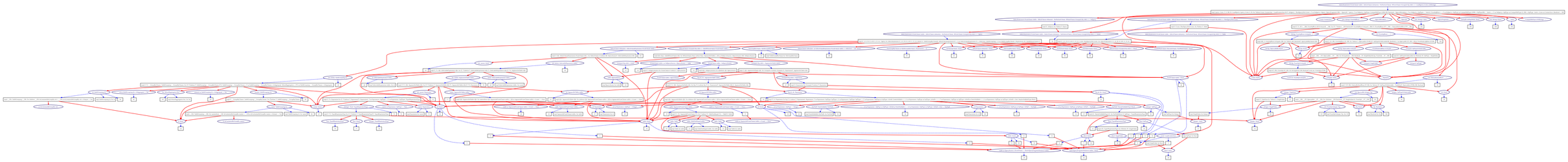


Gray, Crichton, and Krishnamurthi. An Interactive Debugger for Rust Trait Errors *(PLDI 2025)*.

We can represent the proof search trace using an AND–OR graph



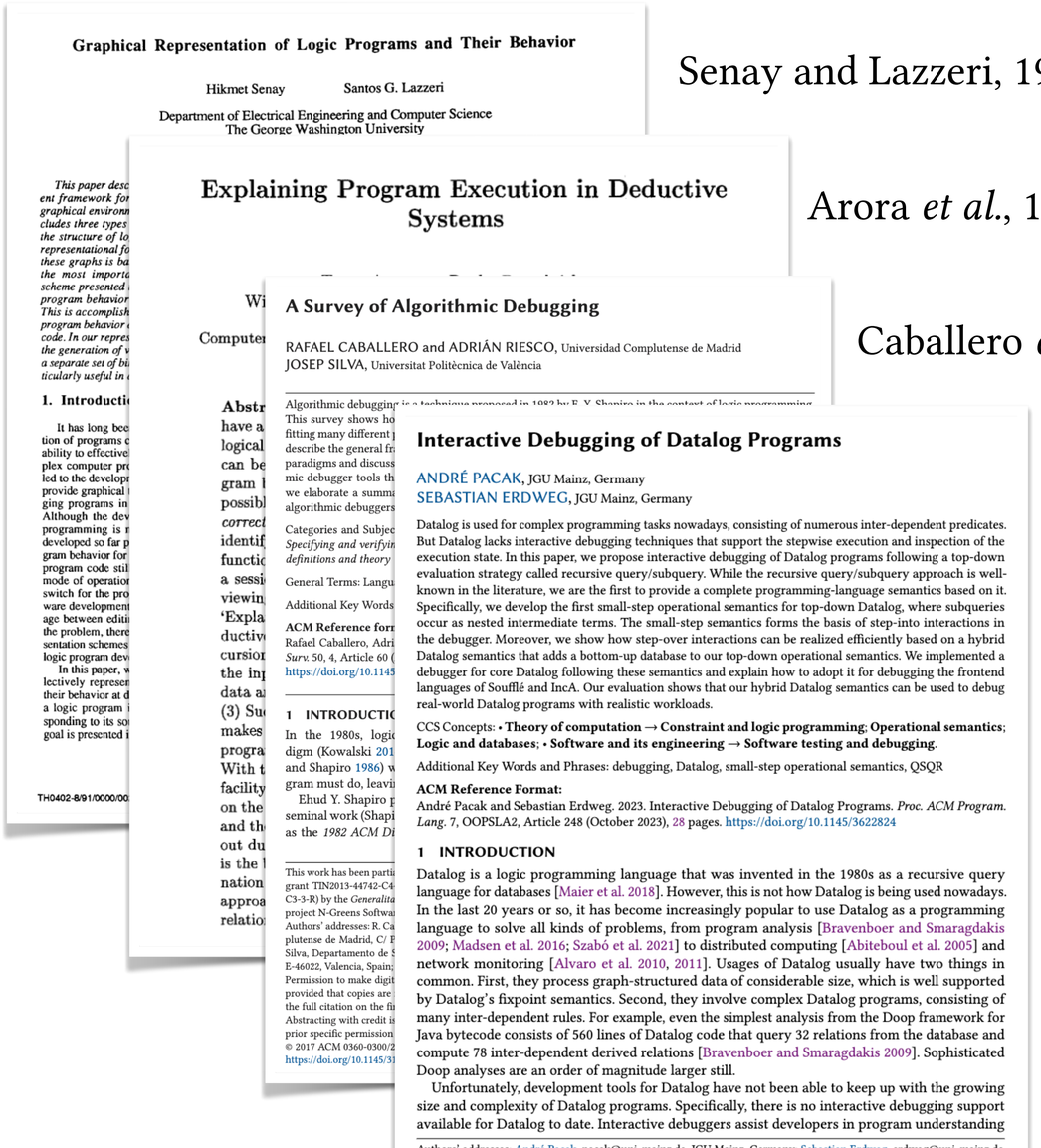




Gray, Crichton, and Krishnamurthi. An Interactive Debugger for Rust Trait Errors *(PLDI 2025).*

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Hard and harder: AND graphs vs. AND–OR graphs



Senay and Lazzeri, 1991

Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

Hard and harder: AND graphs vs. AND–OR graphs

Senay and Lazzeri, 1991

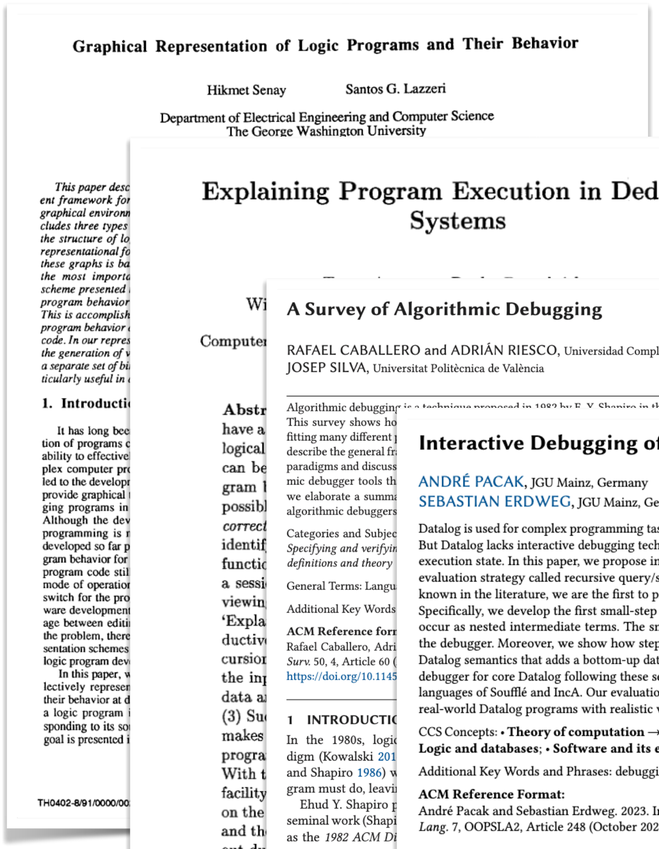
Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

> "Even though a proof tree gives a brief account of the solution steps performed during the computation, it only does so for the goals that have been solved in the process. However, it presents neither the goals that have been tried but unsolved nor the alternative clauses that could have been used to find an alternative solution."

Hard and harder: AND graphs vs. AND–OR graphs



Senay and Lazzeri, 1991

Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

> “Even though a proof tree gives a brief account of the solution steps performed during the computation, it only does so for the goals that have been solved in the process. However, it presents neither the goals that have been tried but unsolved nor the alternative clauses that could have been used to find an alternative solution.”

AND graph
(buggy program/proof)

≈

Hard and harder: AND graphs vs. AND–OR graphs


Graphical Representation of Logic Programs and Their Behavior

Hikmet Senay Santos G. Lazzeri

Department of Electrical Engineering and Computer Science
The George Washington University

Explaining Program Execution in Deductive Systems

A Survey of Algorithmic Debugging

RAFAEL CABALLERO and ADRIÁN RIESCO, Universidad Complutense de Madrid
JOSEP SILVA, Universitat Politècnica de València

Interactive Debugging of Datalog Programs

ANDRÉ PACAK, JGU Mainz, Germany
SEBASTIAN ERDWEG, JGU Mainz, Germany

Datalog is used for complex programming tasks nowadays, consisting of numerous inter-dependent predicates. But Datalog lacks interactive debugging techniques that support the stepwise execution and inspection of the execution state. In this paper, we propose interactive debugging of Datalog programs following a top-down evaluation strategy called recursive query/subquery. While the recursive query/subquery approach is well-known in the literature, we are the first to provide a complete programming-language semantics based on it. Specifically, we develop the first small-step operational semantics for top-down Datalog, where subqueries occur as nested intermediate terms. The small-step semantics forms the basis of step-into interactions in the debugger. Moreover, we show how step-over interactions can be realized efficiently based on a hybrid Datalog semantics that adds a bottom-up database to our top-down operational semantics. We implemented a debugger for core Datalog following these semantics and explain how to adopt it for debugging the frontend languages of Soufflé and IncA. Our evaluation shows that our hybrid Datalog semantics can be used to debug real-world Datalog programs with realistic workloads.

CCS Concepts: • **Theory of computation → Constraint and logic programming**; **Operational semantics**; **Logic and databases**; • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: debugging, Datalog, small-step operational semantics, QSQR

ACM Reference Format:
André Pacak and Sebastian Erdweg. 2023. Interactive Debugging of Datalog Programs. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 248 (October 2023), 28 pages. https://doi.org/10.1145/3622824

1 INTRODUCTION

Datalog is a logic programming language that was invented in the 1980s as a recursive query language for databases [Maier et al. 2018]. However, this is not how Datalog is being used nowadays. In the last 20 years or so, it has become increasingly popular to use Datalog as a programming language to solve all kinds of problems, from program analysis [Bravenboer and Smaragdakis 2009; Madsen et al. 2016; Szabó et al. 2021] to distributed computing [Abiteboul et al. 2005] and network monitoring [Alvaro et al. 2010, 2011]. Usages of Datalog usually have two things in common. First, they process graph-structured data of considerable size, which is well supported by Datalog's fixpoint semantics. Second, they involve complex Datalog programs, consisting of many inter-dependent rules. For example, even the simplest analysis from the Doop framework for Java bytecode consists of 560 lines of Datalog code that query 32 relations from the database and compute 78 inter-dependent derived relations [Bravenboer and Smaragdakis 2009]. Sophisticated Doop analyses are an order of magnitude larger still.

Unfortunately, development tools for Datalog have not been able to keep up with the growing size and complexity of Datalog programs. Specifically, there is no interactive debugging support available for Datalog to date. Interactive debuggers assist developers in program understanding

Senay and Lazzeri, 1991

> "Even though a proof tree gives a brief account of the solution steps performed during the computation, it only does so for the goals that have been solved in the process. However, it presents neither the goals that have been tried but unsolved nor the alternative clauses that could have been used to find an alternative solution."

Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

AND graph
(buggy program/proof)

≈

Tools to help: type systems, debuggers (step-based, algorithmic), *etc.*

Hard and harder: AND graphs vs. AND–OR graphs

Senay and Lazzeri, 1991

Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

> "Even though a proof tree gives a brief account of the solution steps performed during the computation, it only does so for the goals that have been solved in the process. However, it presents neither the goals that have been tried but unsolved nor the alternative clauses that could have been used to find an alternative solution."

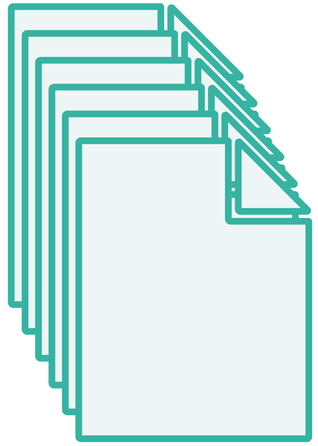
AND graph
(buggy program/proof)

≈

Tools to help: type systems, debuggers (step-based, algorithmic), *etc.*

AND–OR graph
(all possible failing programs/proofs)

≈



Hard and harder: AND graphs vs. AND–OR graphs

Senay and Lazzeri, 1991

> "Even though a proof tree gives a brief account of the solution steps performed during the computation, it only does so for the goals that have been solved in the process. However, it presents neither the goals that have been tried but unsolved nor the alternative clauses that could have been used to find an alternative solution."

Arora *et al.*, 1993

Caballero *et al.*, 2017

Pacak and Erdweg, 2023

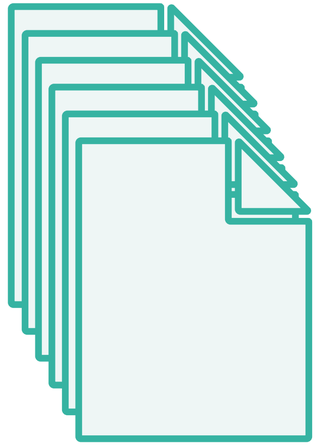
AND graph
(buggy program/proof)

≈

Tools to help: type systems, debuggers (step-based, algorithmic), *etc.*

AND–OR graph
(all possible failing programs/proofs)

≈



Tools to help: Graph visualization, graph traversal

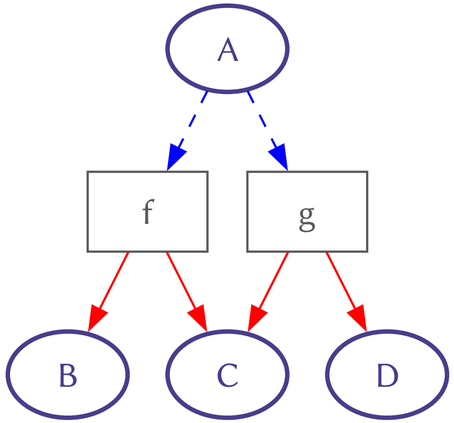
AND–OR graphs are not just graphs

AND–OR graphs are not just graphs

Observation: Hidden inside is a proof system that can be poked and prodded

AND–OR graphs are not just graphs

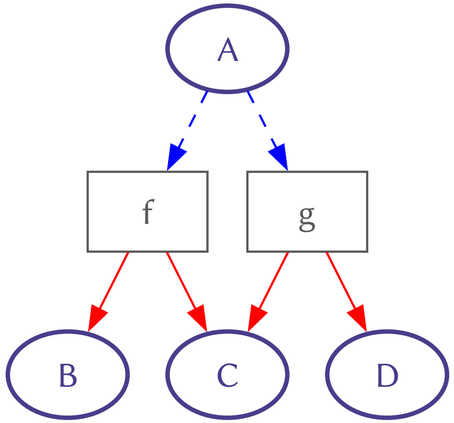
Observation: Hidden inside is a proof system that can be poked and prodded



AND–OR graphs are not just graphs

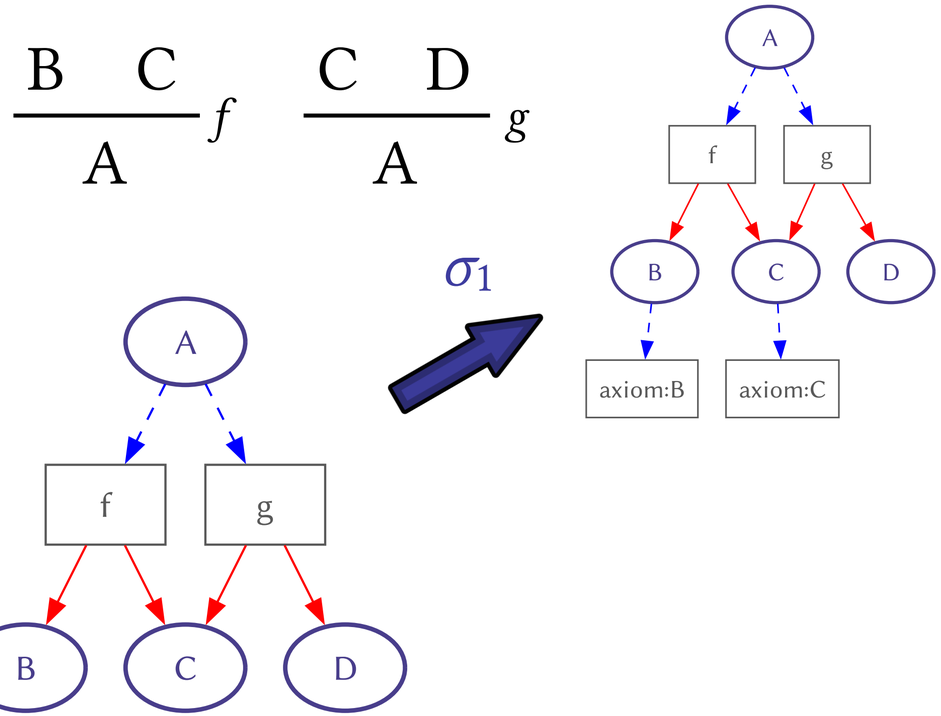
Observation: Hidden inside is a proof system that can be poked and prodded

$$\frac{B \quad C}{A}\,f \qquad \frac{C \quad D}{A}\,g$$



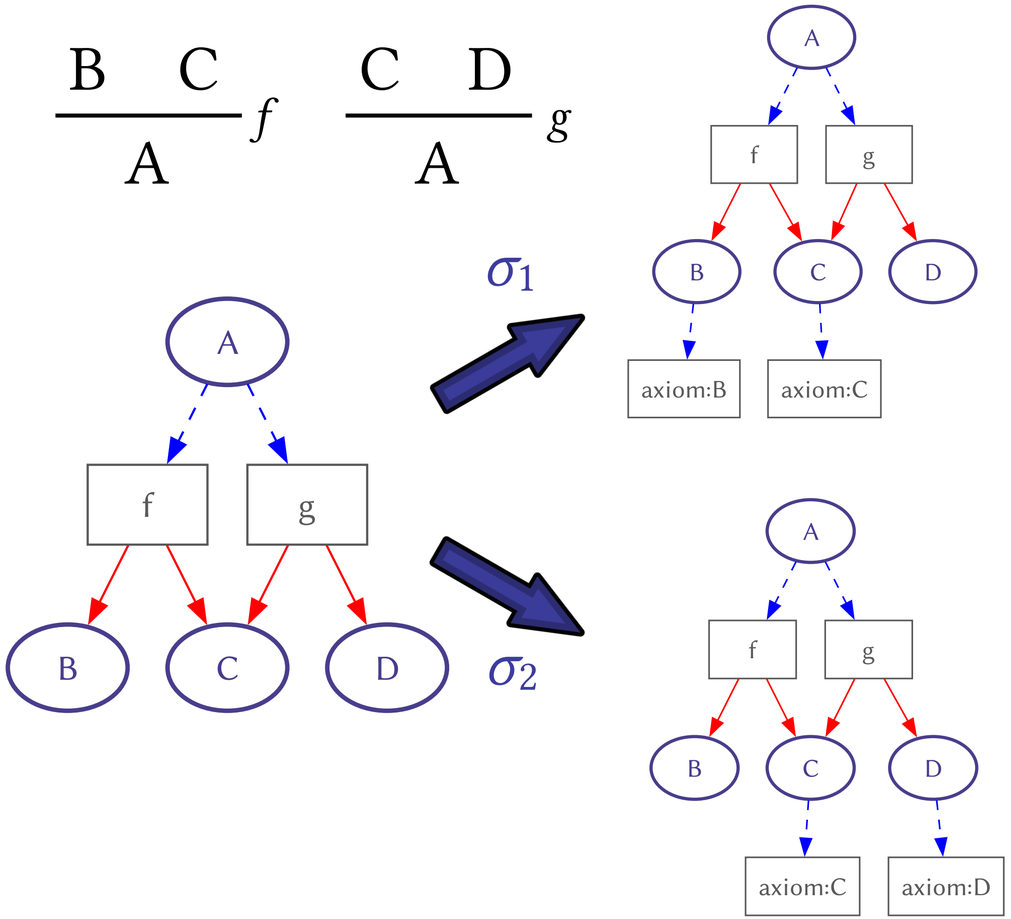
AND–OR graphs are not just graphs

Observation: Hidden inside is a proof system that can be poked and prodded



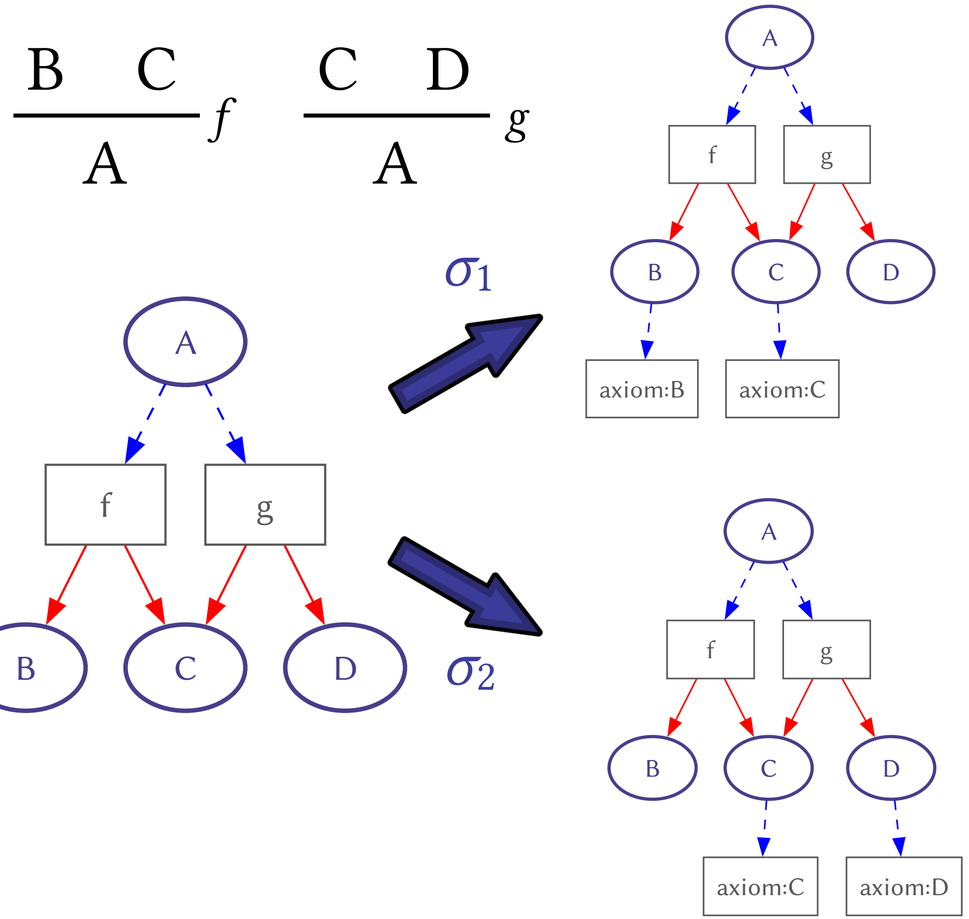
AND–OR graphs are not just graphs

Observation: Hidden inside is a proof system that can be poked and prodded



AND–OR graphs are not just graphs

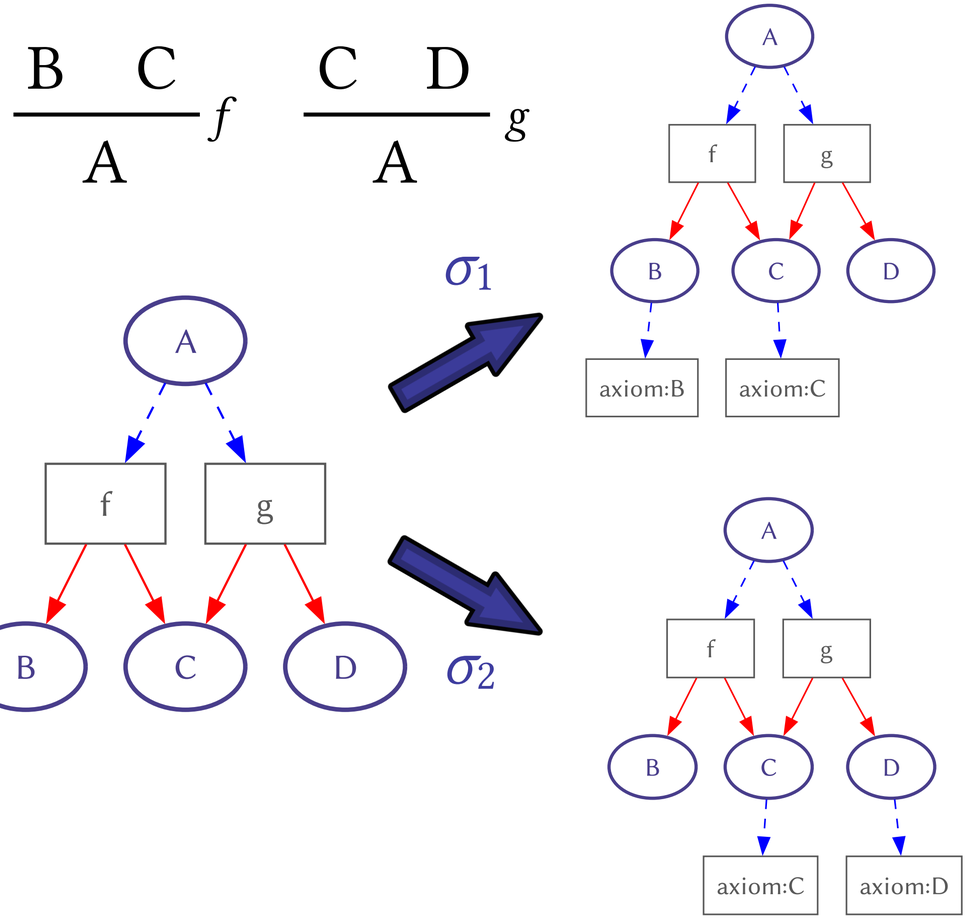
Observation: Hidden inside is a proof system that can be poked and prodded



Key insight:

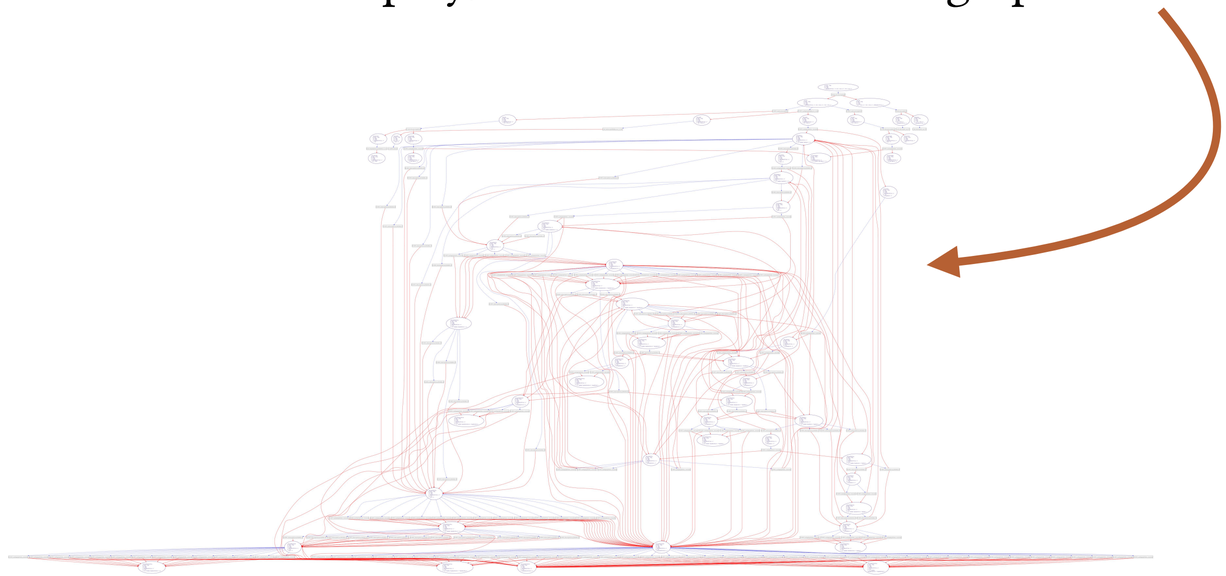
AND–OR graphs are not just graphs

Observation: Hidden inside is a proof system that can be poked and prodded



Key insight:

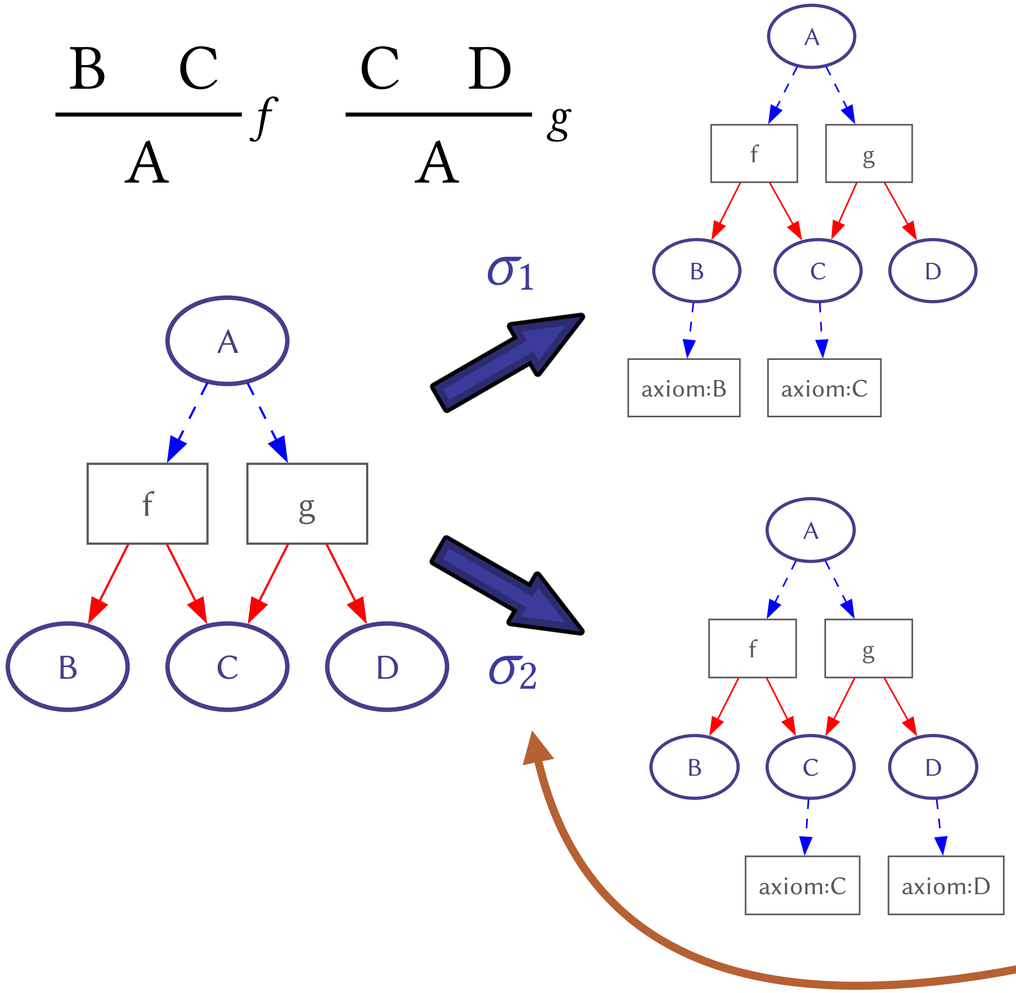
We don't want to display/traverse the AND–OR graph…



AND–OR graphs are not just graphs

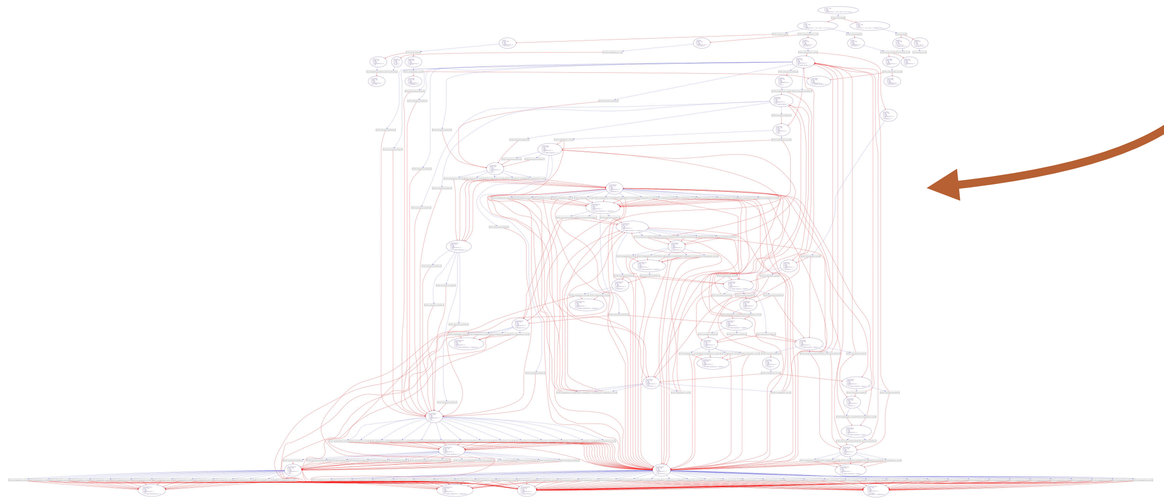
Observation: Hidden inside is a proof system that can be poked and prodded

$$\frac{B \quad C}{A}\, f \qquad \frac{C \quad D}{A}\, g$$



Key insight:

We don't want to display/traverse the AND–OR graph…



…we want to navigate the **space of modifications** to the graph

Concrete example: Debugging Aesop for Lean

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

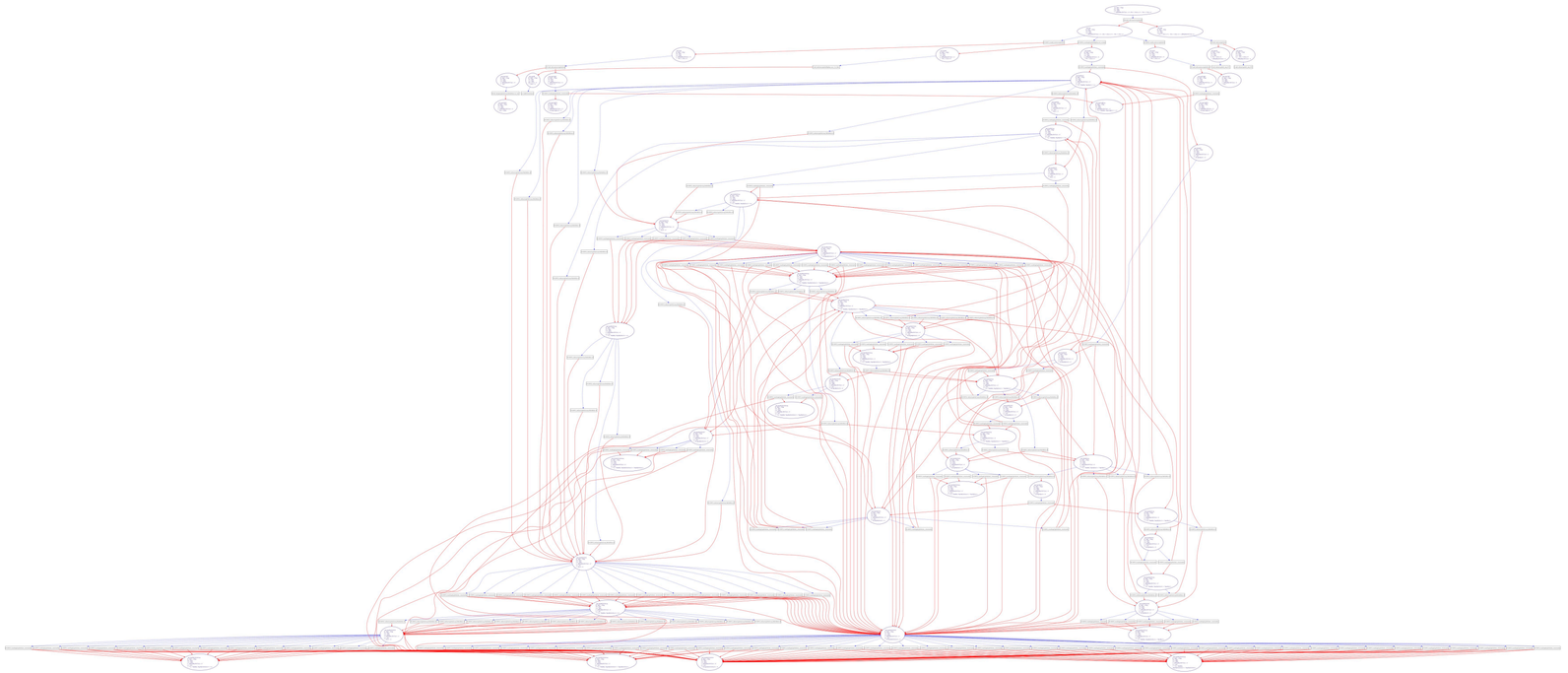
Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

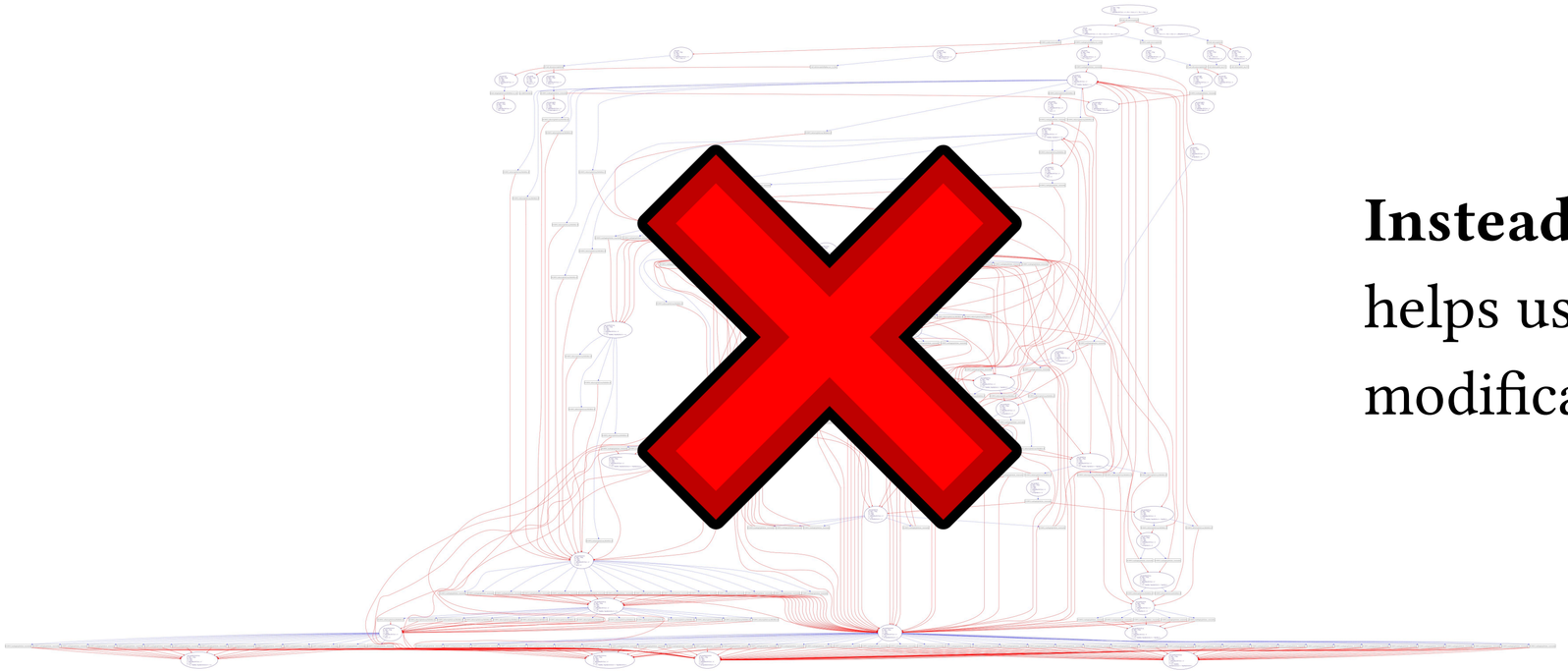


Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$



Instead: System helps user navigate modifications

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects:

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

- "Willing to prove manually"
- "Not willing to prove manually, but should be true"
- *above* + "must be used"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"
above + "must be used"	*above* + "must be used"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	
above + "must be used"	*above* + "must be used"	"I don't know"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	"Should be false"
above + "must be used"	*above* + "must be used"	"I don't know"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	"S… be …"
above + "must be used"	*above* + "must be used"	"I don't know"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

“Willing to prove manually”	“Not willing to prove manually, but should be true”	“S… be …”
above + “must be used”	*above* + “must be used”	“I don’t know”

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	"S… be …"
above + "must be used"	*above* + "must be used"	"I don't know"

System selects:

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

“Willing to prove manually”	“Not willing to prove manually, but should be true”	“S… be …”
above + “must be used”	*above* + “must be used”	“I don’t know”

System selects: $\cdots, (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \vdash B(\sigma) \rightarrow (S, \sigma) \Downarrow \sigma'$

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	"S… be …"
above + "must be used"	*above* + "must be used"	"I don't know"

System selects: $\cdots, (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \vdash B(\sigma) \rightarrow (S, \sigma) \Downarrow \sigma'$

User can respond:

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023)*.

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

“Willing to prove manually”	“Not willing to prove manually, but should be true”	“S… be … 
above + “must be used”	*above* + “must be used”	“I don’t know”

System selects: $\cdots, (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \vdash B(\sigma) \rightarrow (S, \sigma) \Downarrow \sigma'$

User can respond: *above options* + “Should be false”

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Concrete example: Debugging Aesop for Lean

Modifications allowed: labeling/assuming possible lemmas

Goal: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \leftrightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

System selects: $\cdots \vdash (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \rightarrow B(\sigma) \wedge (S, \sigma) \Downarrow \sigma' \vee \neg B(\sigma) \wedge (T, \sigma) \Downarrow \sigma'$

User can respond:

"Willing to prove manually"	"Not willing to prove manually, but should be true"	"S… be …"
above + "must be used"	*above* + "must be used"	"I don't know"

System selects: $\cdots, (\texttt{ifThenElse}\ B\ S\ T, \sigma) \Downarrow \sigma' \vdash B(\sigma) \rightarrow (S, \sigma) \Downarrow \sigma'$

User can respond: *above options* + "Should be false"

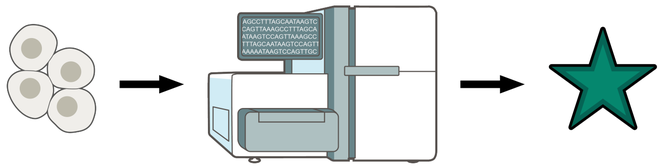
"Willing to prove manually"

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*

Programming by Navigation used for...

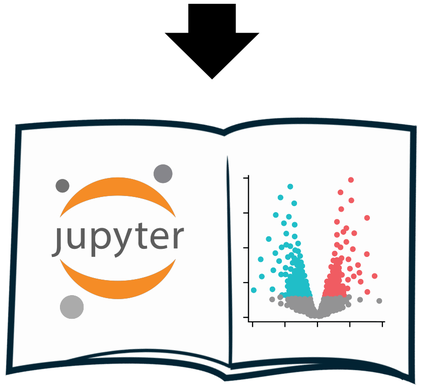
Programming by Navigation used for...

Synthesizing bioinformatics analyses from wet lab experimental workflows



Day 2: RNA-seq

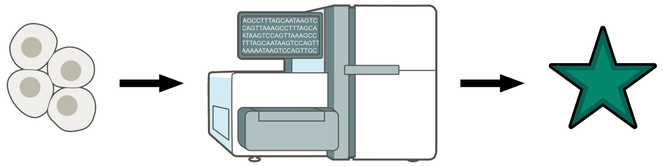
Goal: Differential gene expression



Data analysis program

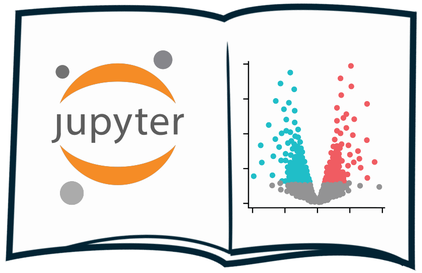
Programming by Navigation used for...

Synthesizing bioinformatics analyses from wet lab experimental workflows



Day 2: RNA-seq

Goal: Differential gene expression



Data analysis program

Creating user schedules for a sparse tensor algebra compiler

Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\forall_{ikj} A_{ij} = B_{ik} C_{kj} \boxed{?}$$ Vector width

$$\forall_{\boxed{?}} E_{ij} = A_{ik} D_{kj}\}$$

Dataflow ordering

? Memory layout

```
{ "dataflow-ordering": "i0 i1 i2 i4 i8 i5 i9 i6
i7 i3", "sparse-annotations": { "t0":
{ "format": "CSR", "indices": { "i": true, "j":
false } }, "t1": { "format": "CSR", "indices":
{ "i": true, "j": false } }, "t2": { "format":
"CSR", "indices": { "i": true, "j": false } },
"t3": { "format": "CSR", "indices": { "i":
true, "j": false } }, "t4": { "format":
"custom", "indices": { "i": false, "j":  · · ·
```

Programming by Navigation used for...

Synthesizing bioinformatics analyses from wet lab experimental workflows

Day 2: RNA-seq

Goal: Differential gene expression

Data analysis program

Creating user schedules for a sparse tensor algebra compiler

Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

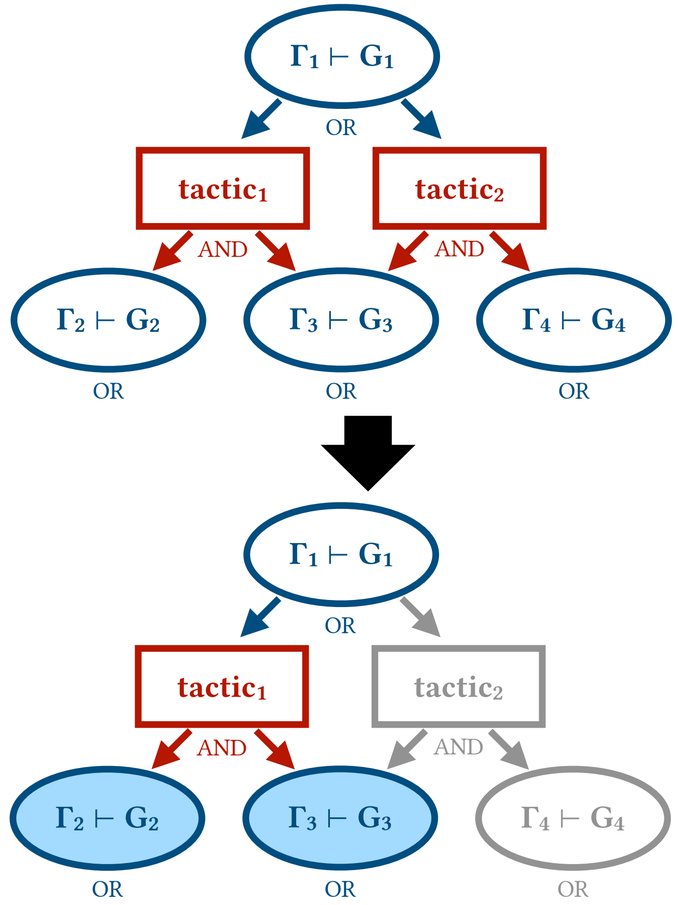
$$\texttt{Fuse}\{\forall_{ikj} A_{ij} = B_{ik} C_{kj} \;?\; \text{Vector width}$$

$$\forall_{?} E_{ij} = A_{ik} D_{kj}\}$$

Dataflow ordering ? Memory layout

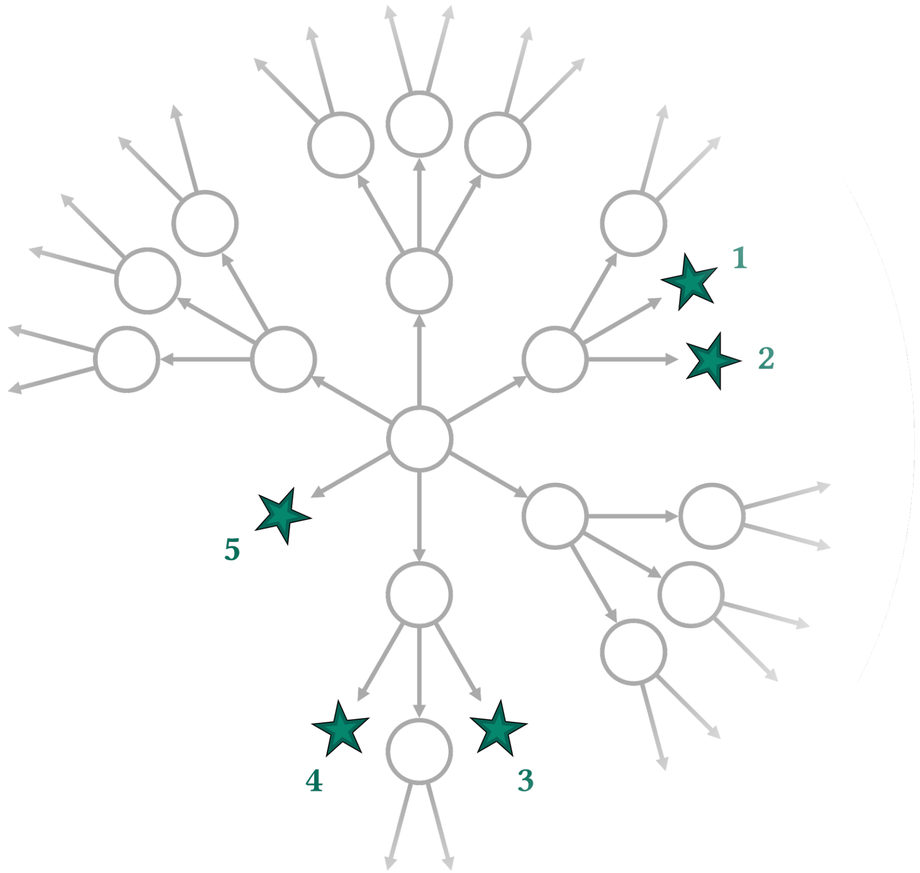
```
{ "dataflow-ordering": "i0 i1 i2 i4 i8 i5 i9 i6
i7 i3", "sparse-annotations": { "t0":
{ "format": "CSR", "indices": { "i": true, "j":
false } }, "t1": { "format": "CSR", "indices":
{ "i": true, "j": false } }, "t2": { "format":
"CSR", "indices": { "i": true, "j": false } },
"t3": { "format": "CSR", "indices": { "i":
true, "j": false } }, "t4": { "format":
"custom", "indices": { "i": false, "j":  · · ·
```

<u>This talk:</u> Debugging failing proof search
(*e.g.*, Aesop for Lean, Rust traits, Datalog, ...)

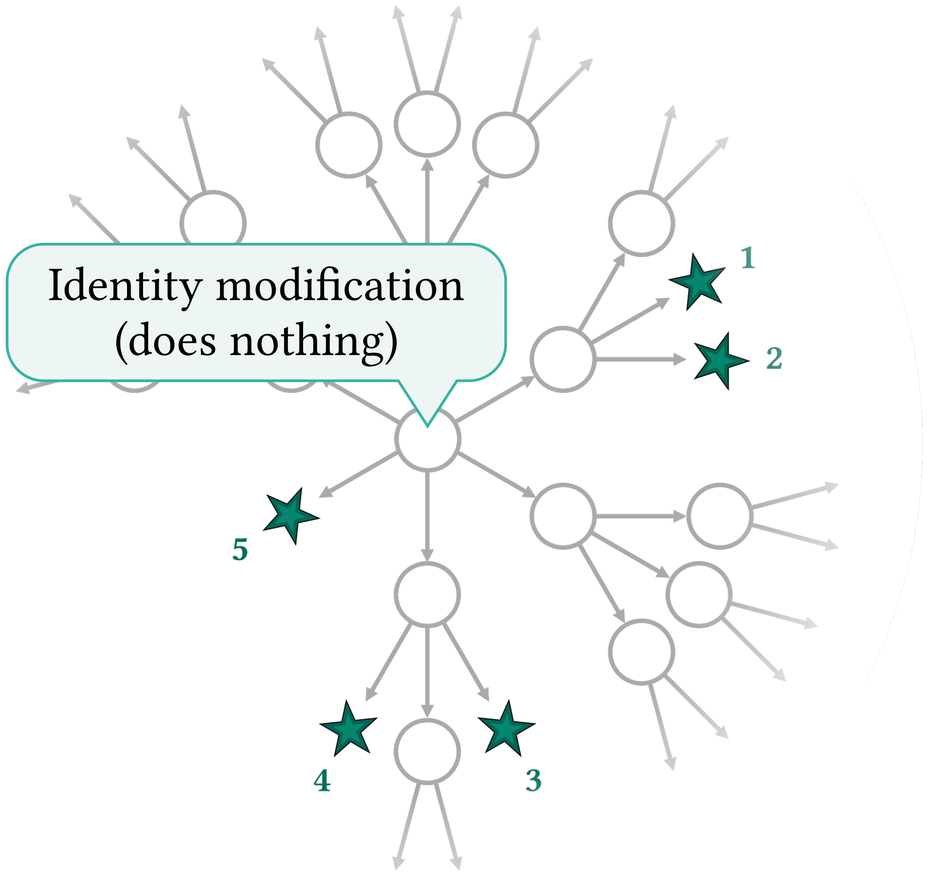


Programming by Navigation

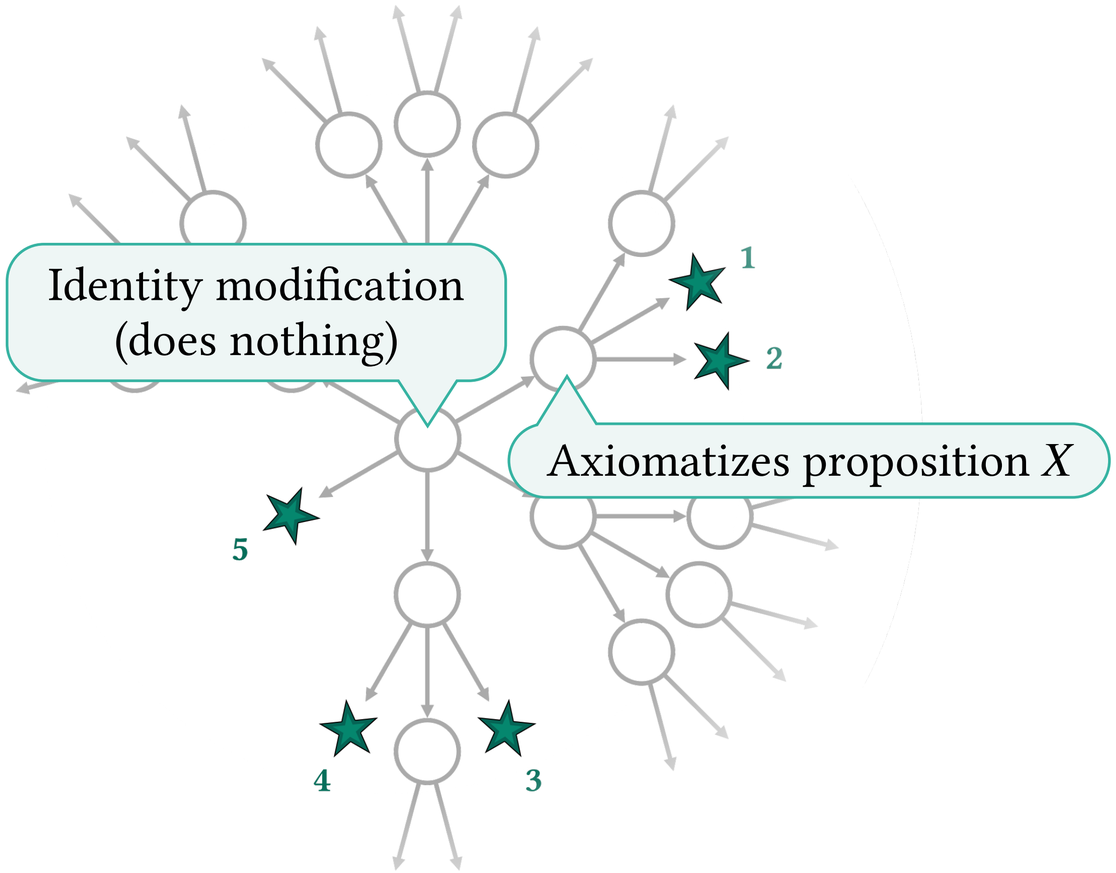
Programming by Navigation



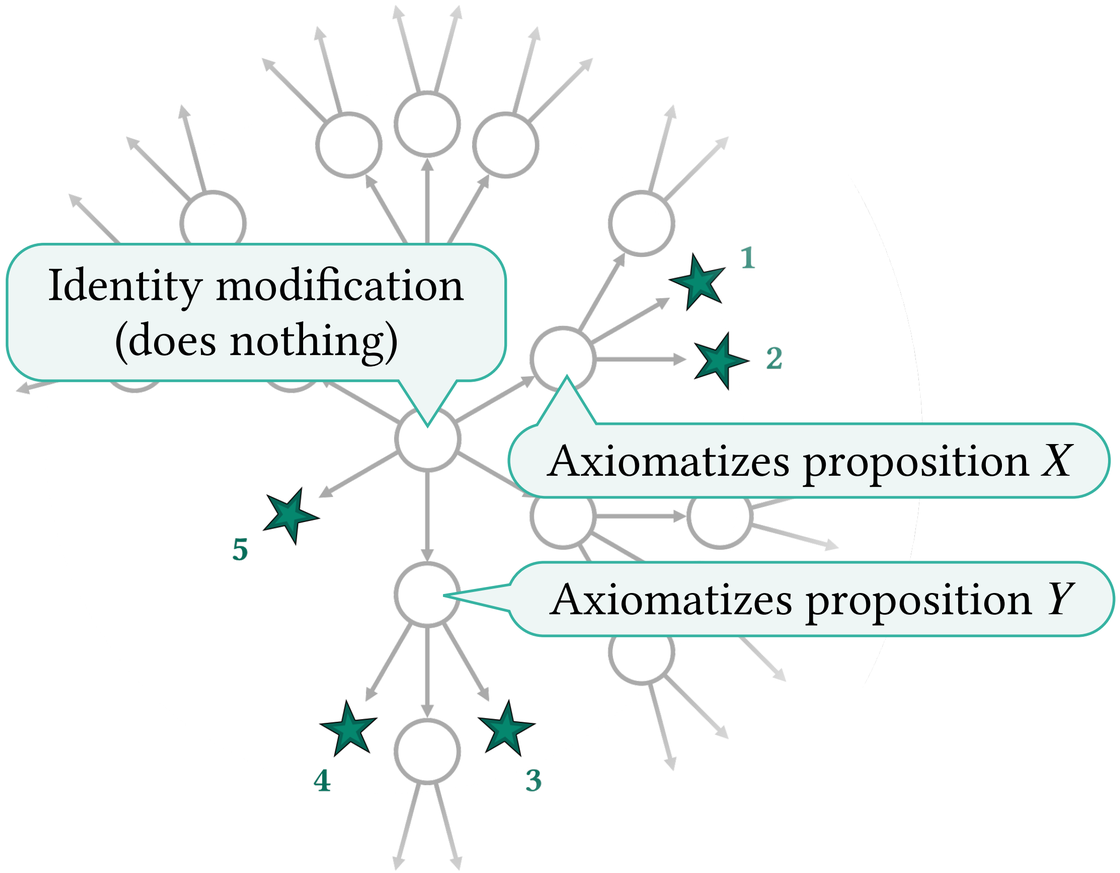
Programming by Navigation



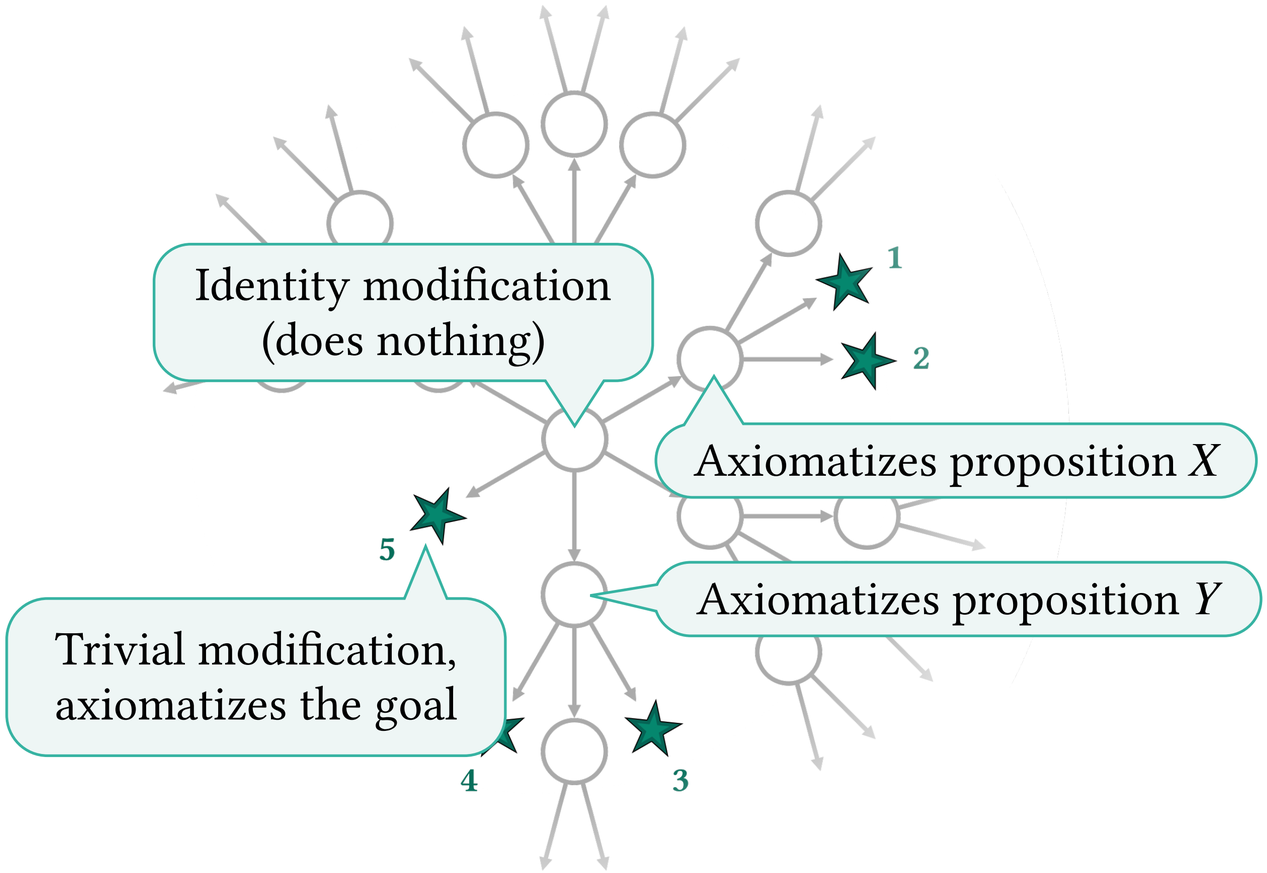
Programming by Navigation



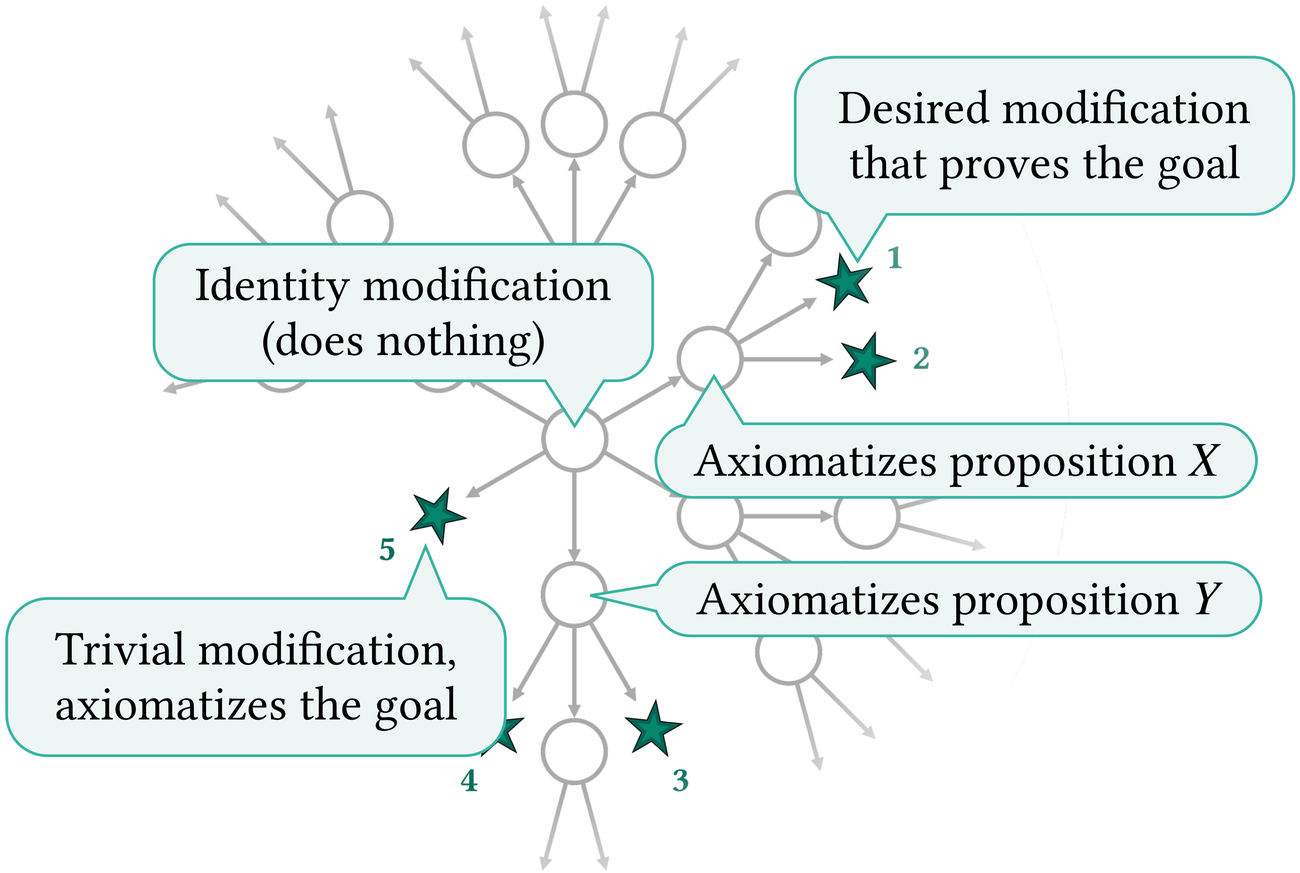
Programming by Navigation



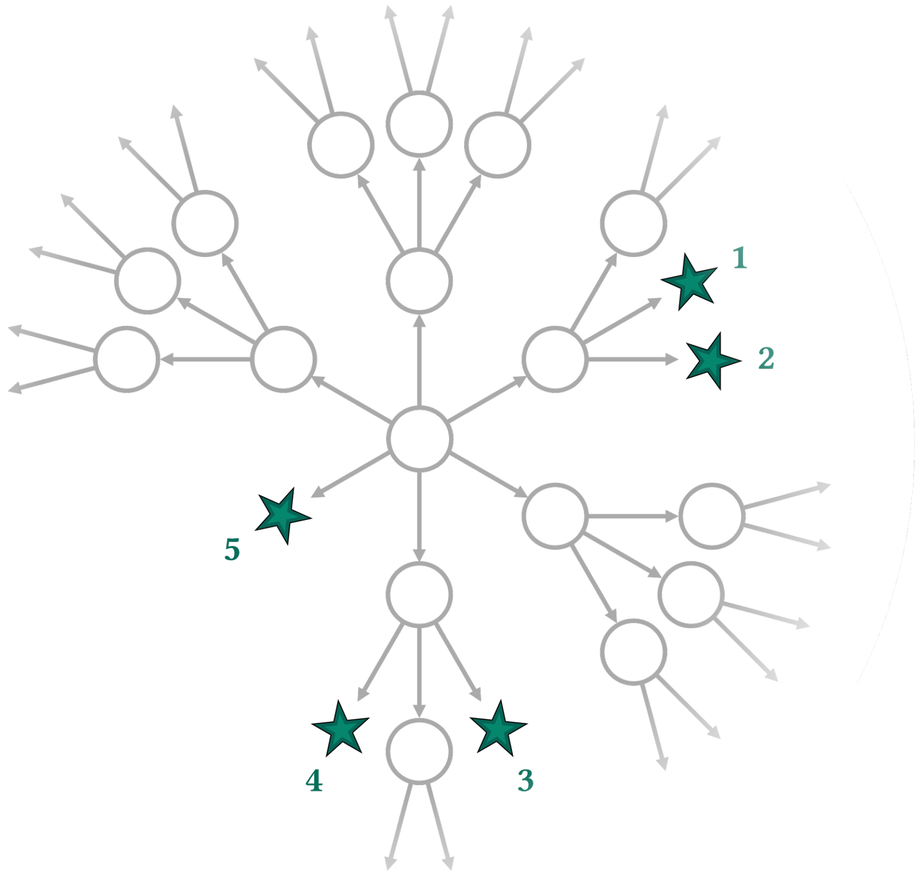
Programming by Navigation



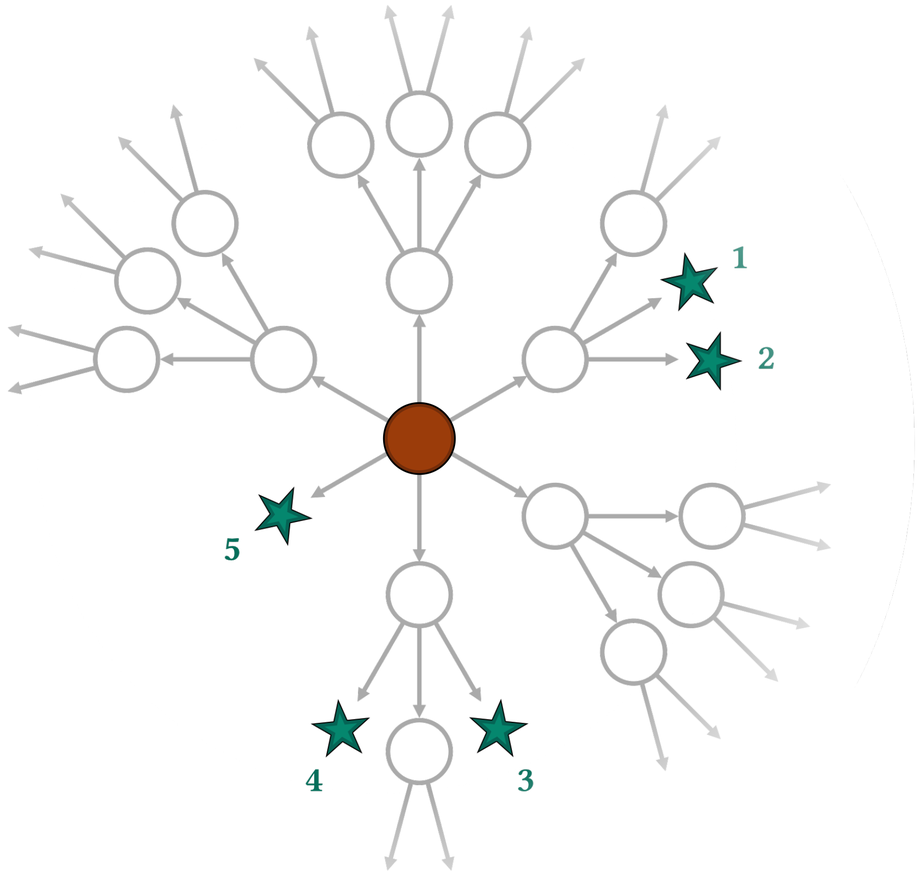
Programming by Navigation



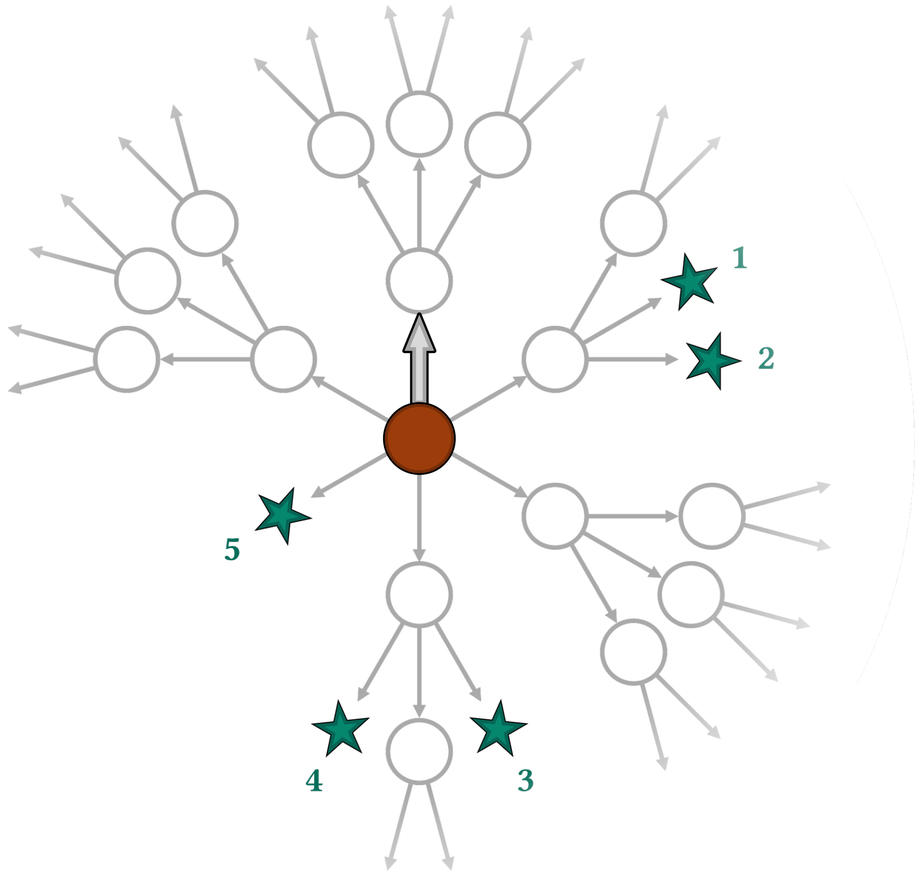
Programming by Navigation



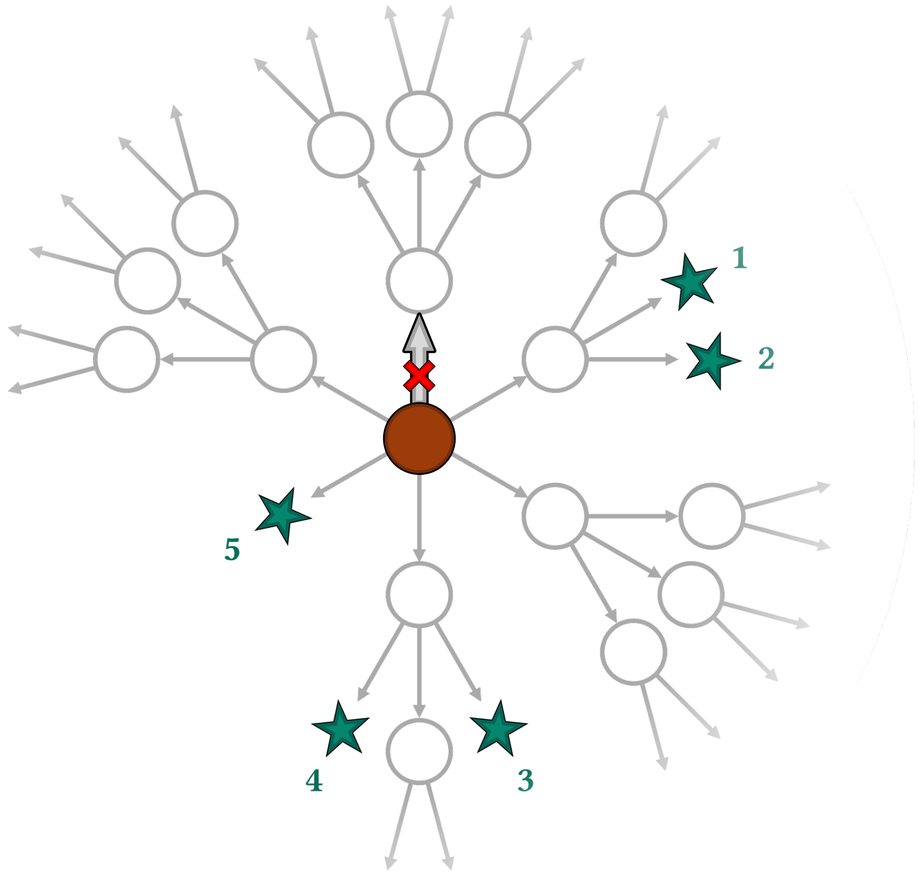
Programming by Navigation



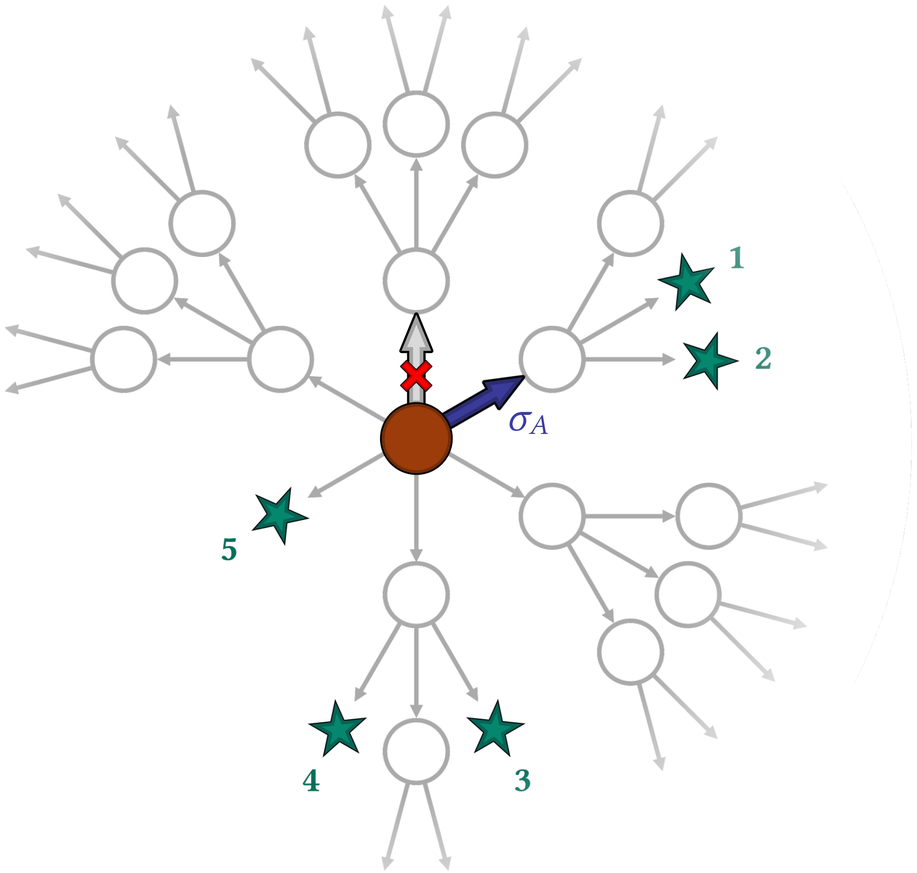
Programming by Navigation



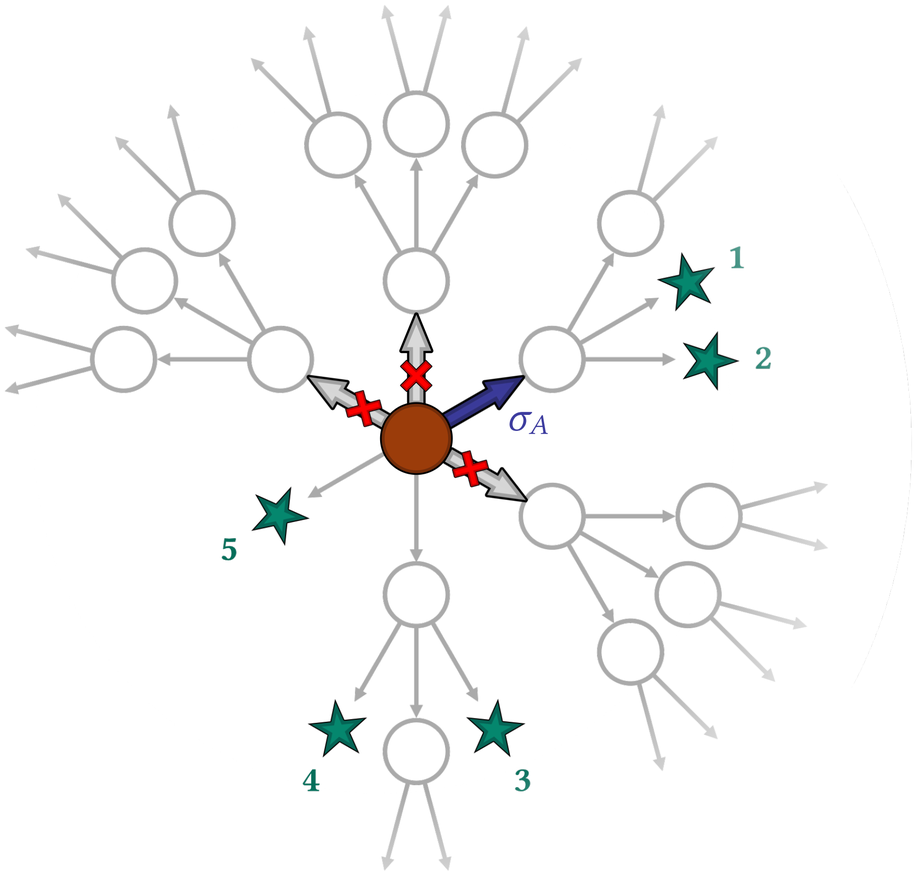
Programming by Navigation



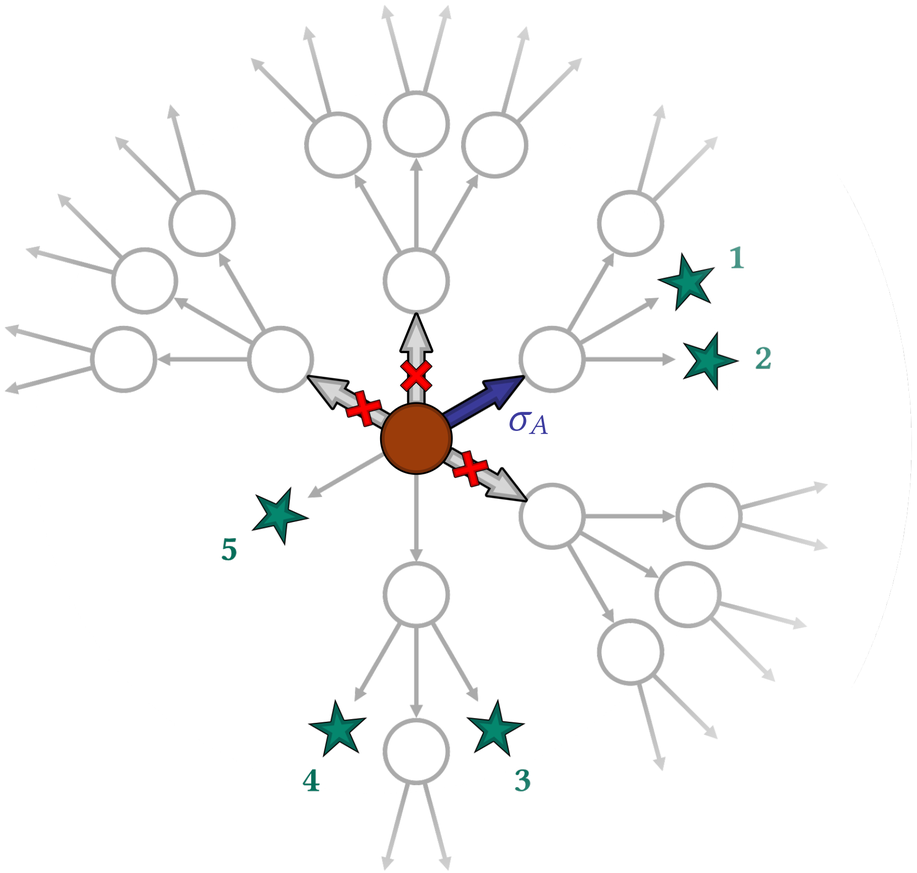
Programming by Navigation



Programming by Navigation

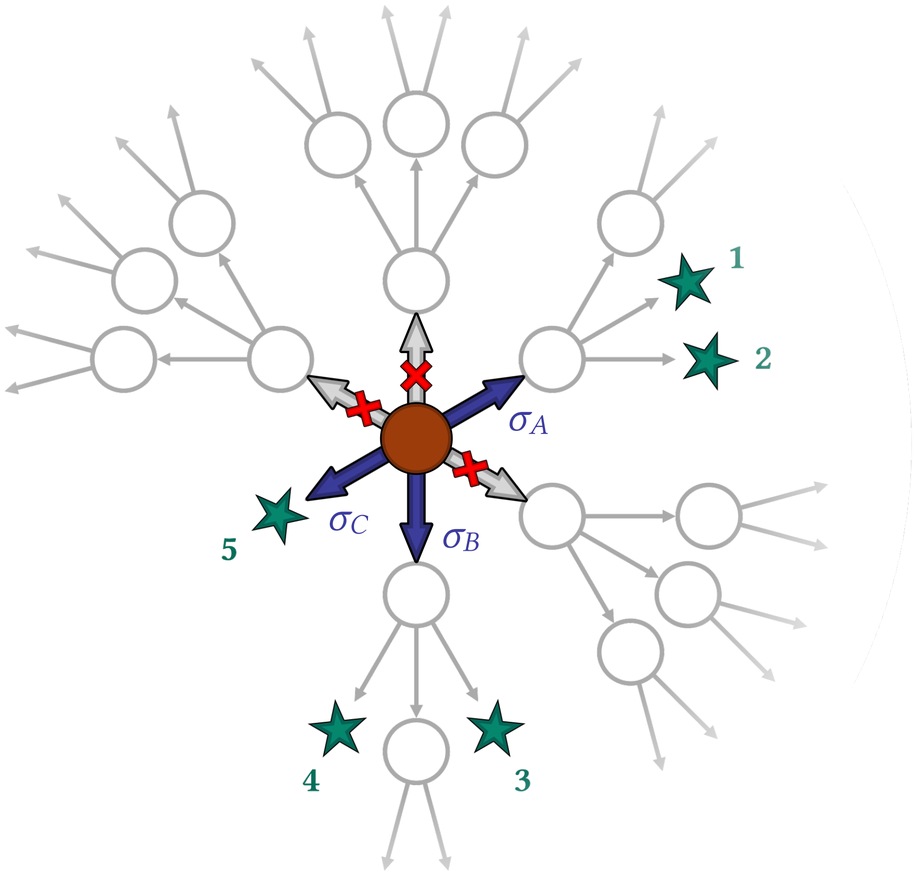


Programming by Navigation



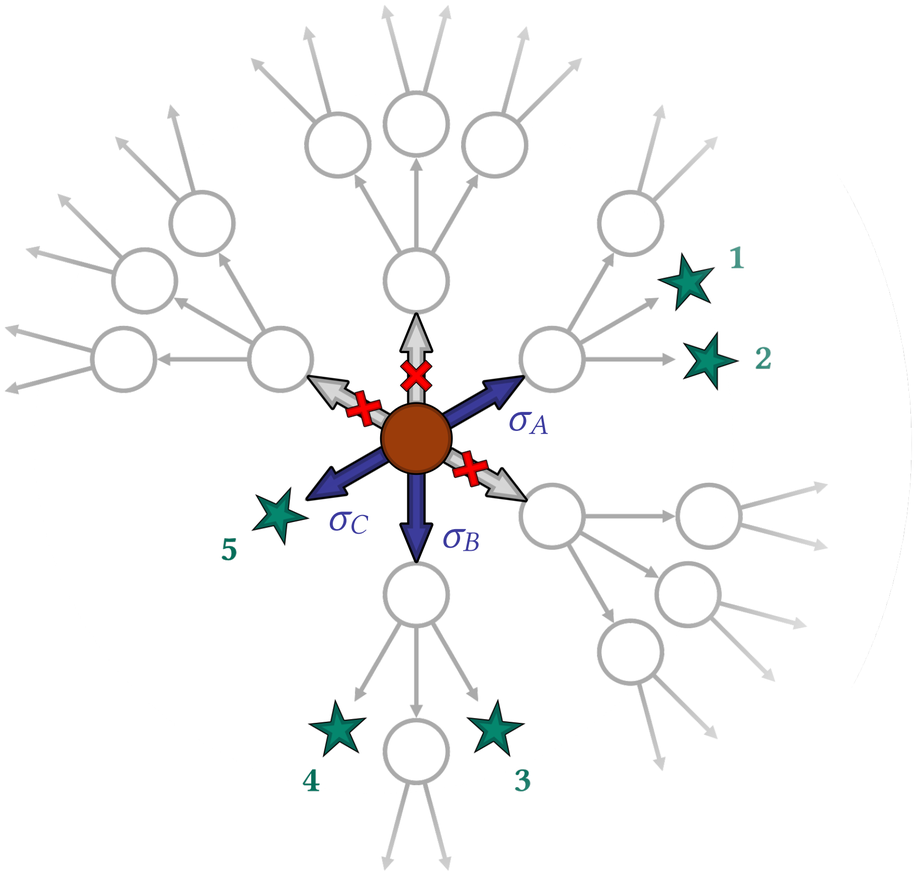
- **STRONG SOUNDNESS:**
  Only valid steps are shown

Programming by Navigation



- **STRONG SOUNDNESS:**
  Only valid steps are shown

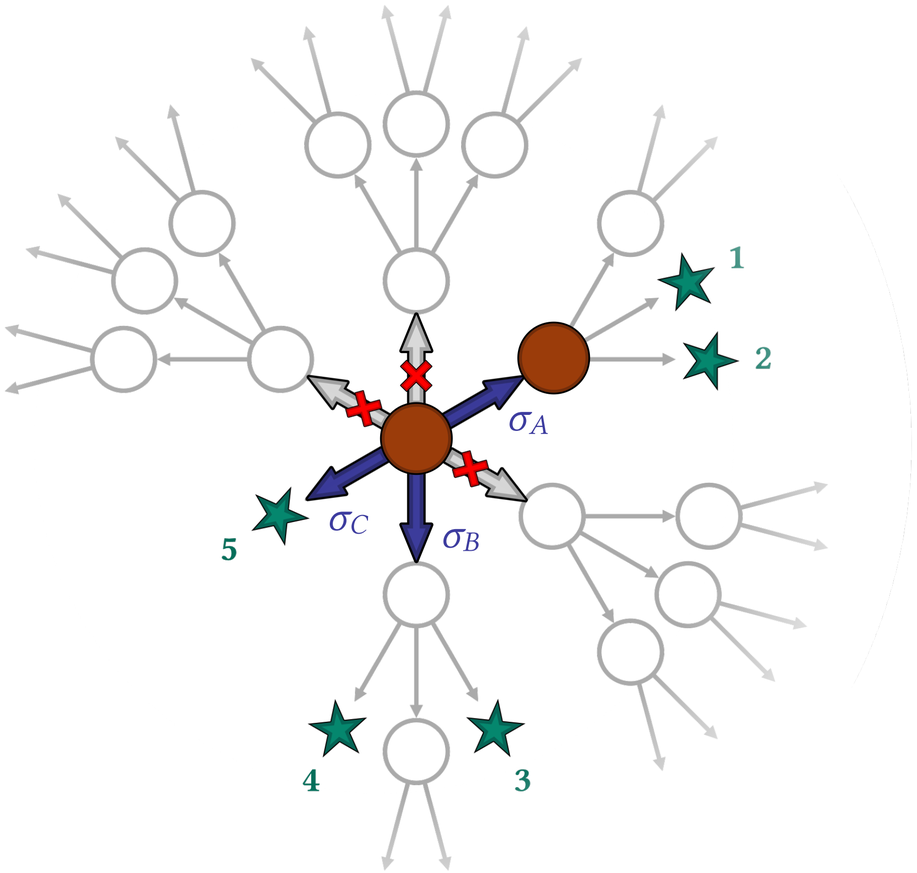
Programming by Navigation



- **Strong Soundness:**
  Only valid steps are shown

- **Strong Completeness:**
  Shown steps can reach *all* valid solutions

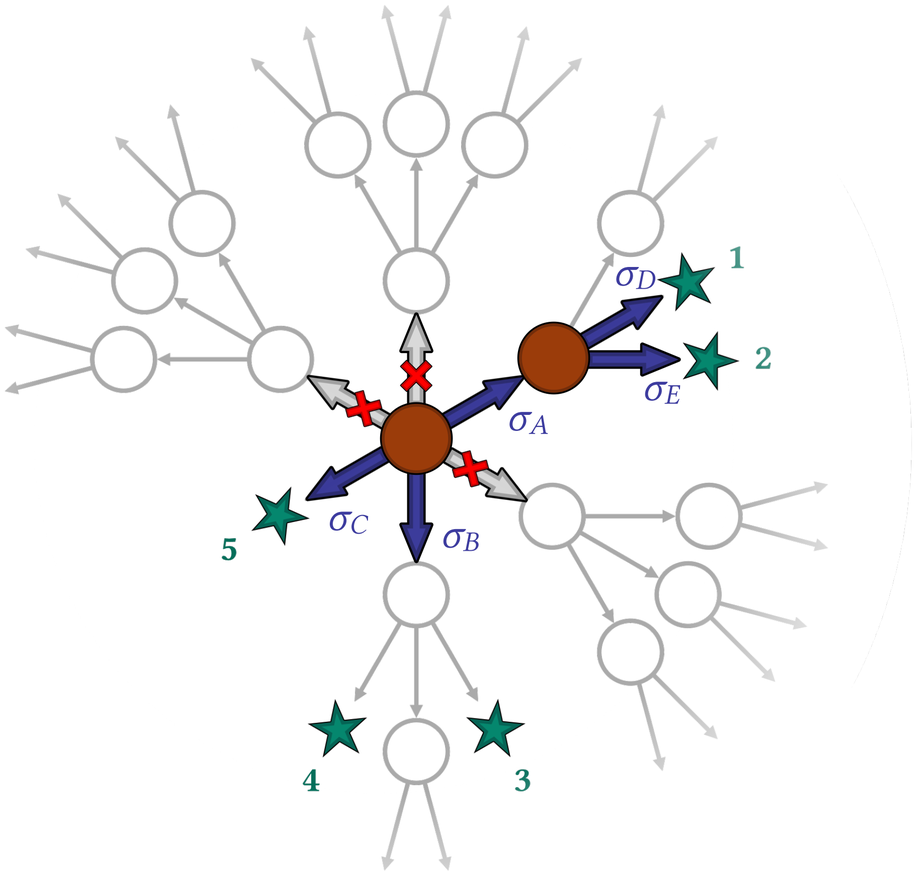
Programming by Navigation



- **Strong Soundness:**
  Only valid steps are shown

- **Strong Completeness:**
  Shown steps can reach *all* valid solutions

Programming by Navigation



- **Strong Soundness:**
  Only valid steps are shown

- **Strong Completeness:**
  Shown steps can reach *all* valid solutions

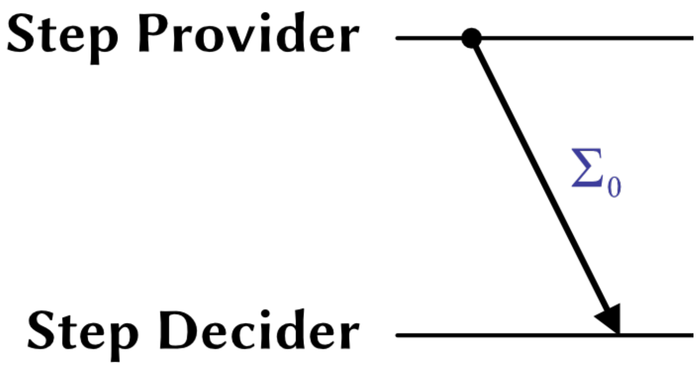
Users select between steps in Programming by Navigation

Users select between steps in Programming by Navigation

Step Provider

Step Decider

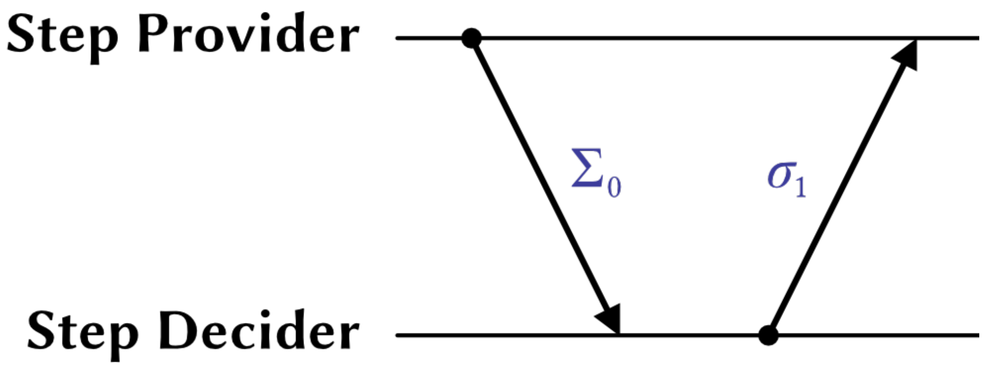
Users select between steps in Programming by Navigation



e.g. Σ_0 =

"Willing to prove manually"	"Not willing to prove manually, but should be true"	
above + "must be used"	*above* + "must be used"	"I don't know"

Users select between steps in Programming by Navigation



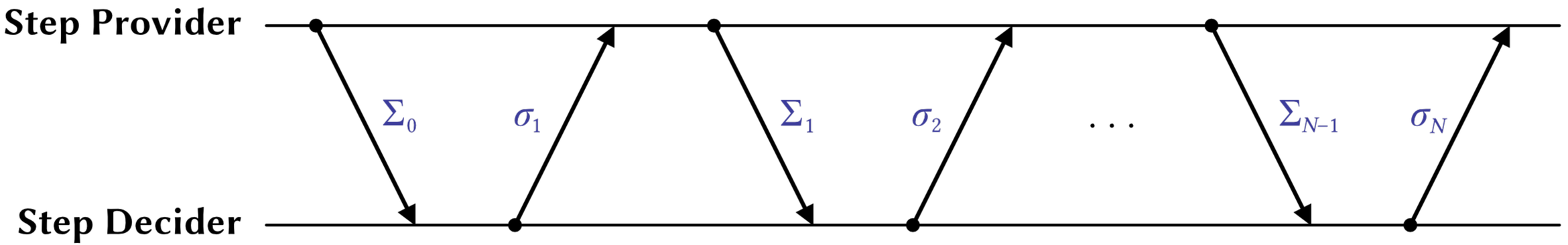
e.g. Σ_0 =

"Willing to prove manually"	"Not willing to prove manually, but should be true"	
above + "must be used"	*above* + "must be used"	"I don't know"

e.g. σ_1 =

"Not willing to prove manually, but should be true"

Users select between steps in Programming by Navigation



e.g. Σ_0 =

"Willing to prove manually"	"Not willing to prove manually, but should be true"	
above + "must be used"	*above* + "must be used"	"I don't know"

e.g. σ_1 =

"Not willing to prove manually, but should be true"

Programming by Navigation Ingredients

Programming by Navigation Ingredients

Expressions

Programming by Navigation Ingredients

Expressions

Validity

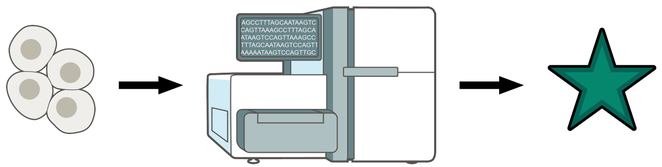
Programming by Navigation Ingredients

Expressions

Validity

Step provider

Programming by Navigation Ingredients



Day 2: RNA-seq

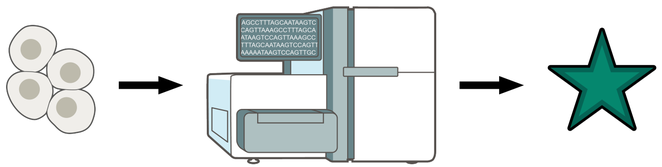
Goal: Differential gene expression

Expressions Bioinformatics pipelines

Validity Meets user-defined validity conditions

Step provider Top-down classical–constructive synthesis

Programming by Navigation Ingredients



Day 2: RNA-seq

Goal: Differential gene expression

Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\ \forall_{ikj}\ A_{ij} = B_{ik}C_{kj}\ \boxed{?}\ \text{Vector width}$$

$$\forall_{\boxed{?}}\ E_{ij} = A_{ik}D_{kj}\ \}$$

Dataflow ordering

$\boxed{?}$ Memory layout

Expressions	Bioinformatics pipelines	FuseFlow compiler schedules
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions
Step provider	Top-down classical–constructive synthesis	Big-step synthesis

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

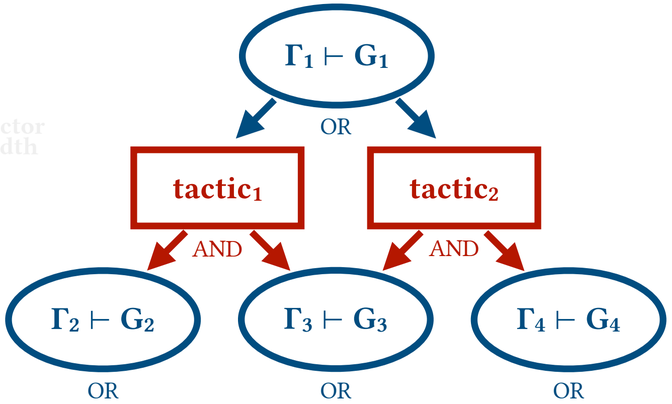
Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\ \forall_{ikj} A_{ij} = B_{ik} C_{kj}\ \boxed{?}\ \text{Vector width}$$

$$\forall_{\boxed{?}} E_{ij} = A_{ik} D_{kj}\ \}$$

Dataflow ordering ? Memory layout



Expressions	Bioinformatics pipelines	FuseFlow compiler schedules
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions
Step provider	Top-down classical–constructive synthesis	Big-step synthesis

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

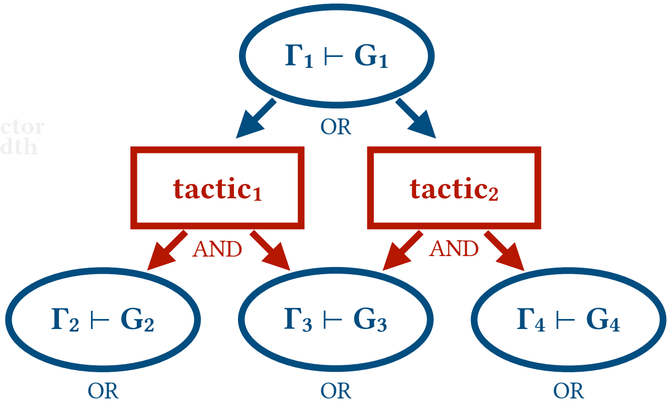
Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\ \forall_{ikj} A_{ij} = B_{ik} C_{kj}\ \boxed{?}\ \text{Vector width}$$

$$\forall_{\boxed{?}} E_{ij} = A_{ik} D_{kj}\ \}$$

Dataflow ordering ? Memory layout



Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

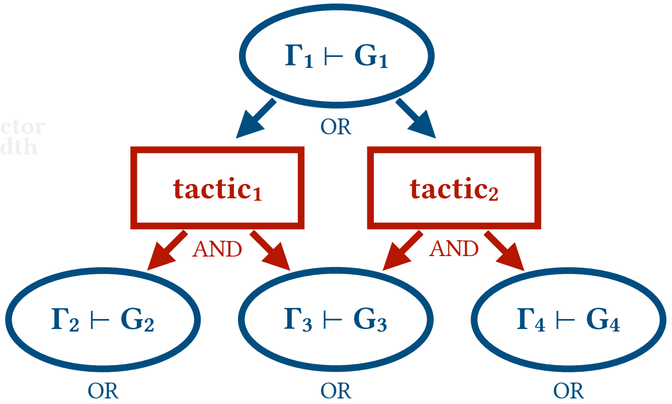
Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\ \forall_{ikj} A_{ij} = B_{ik} C_{kj}$$? Vector width

$$\forall_{?} E_{ij} = A_{ik} D_{kj}\ \}$$

Dataflow ordering ? Memory layout



Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

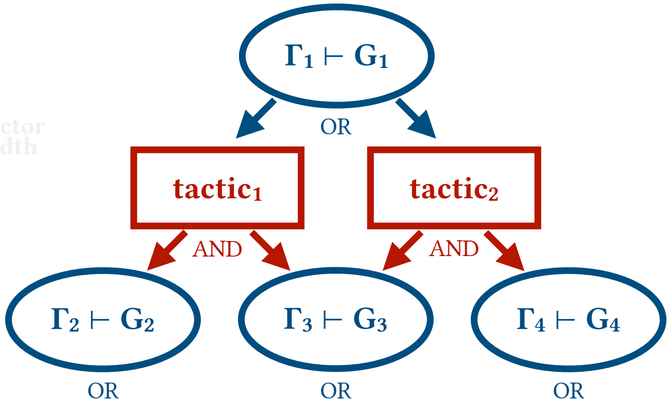
Lacouture *et al.* FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow *(ASPLOS 2026).*

Parallelization factor ? ? Memory layout

$$\texttt{Fuse}\{\forall_{ikj} A_{ij} = B_{ik} C_{kj}$$? Vector width

$$\forall_{?} E_{ij} = A_{ik} D_{kj}\}$$

Dataflow ordering ? Memory layout



Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	?

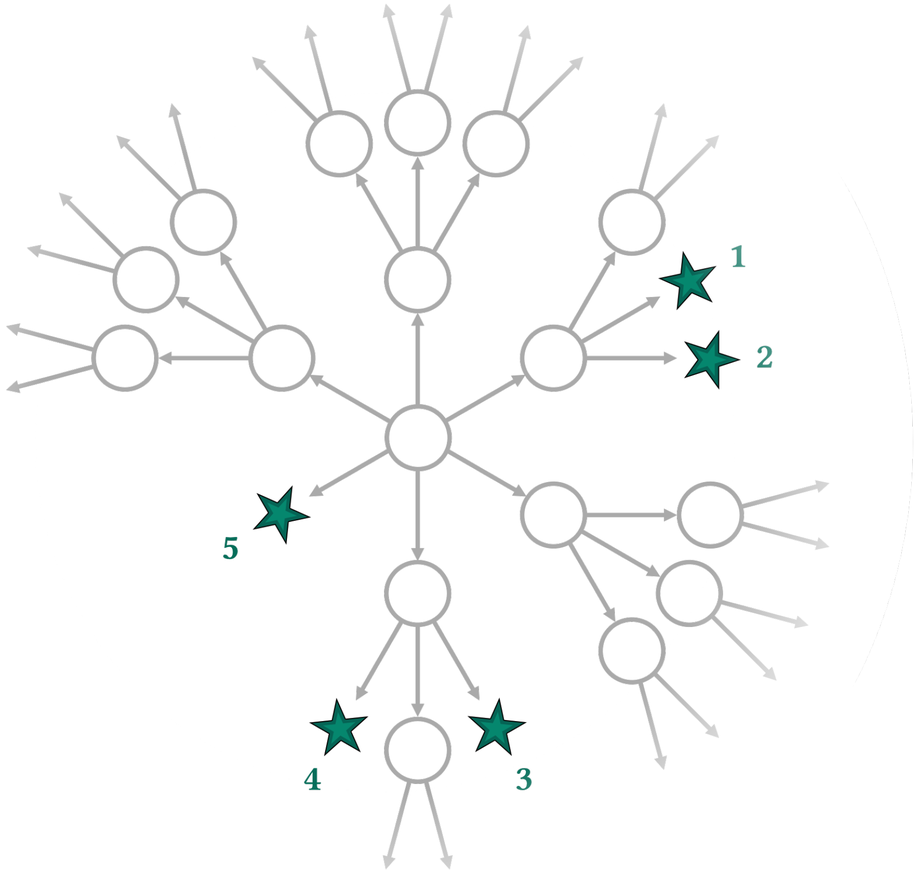
Helpful building block: Nonempty-completion oracle

Helpful building block: Nonempty-completion oracle

$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$

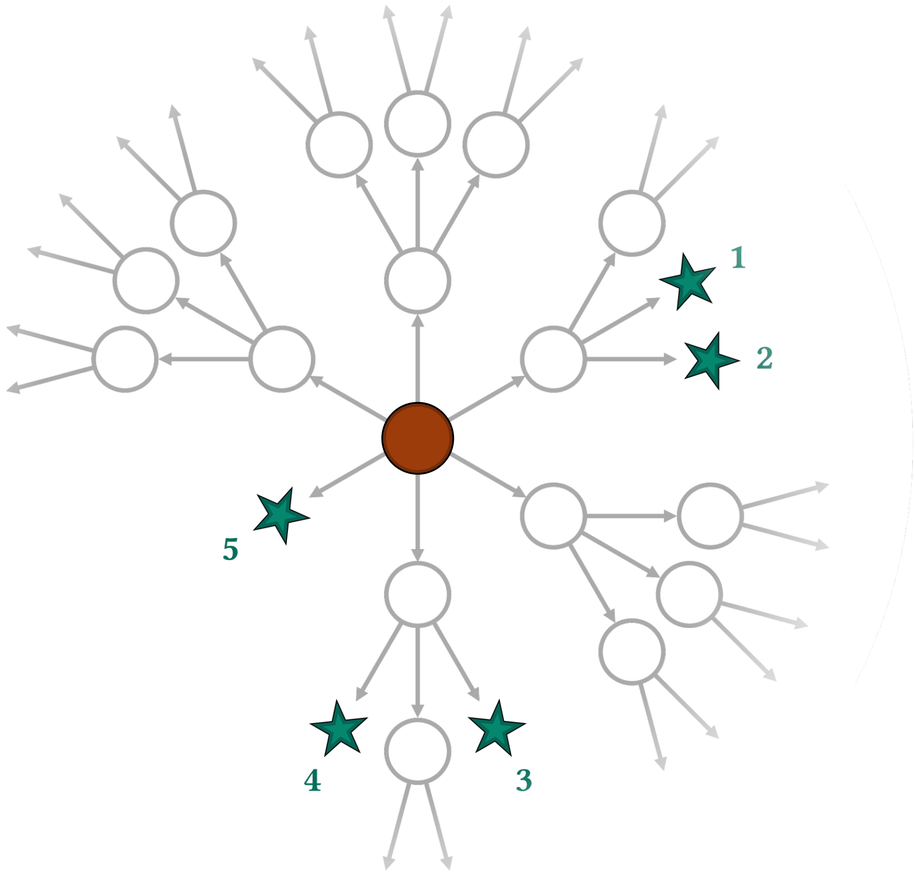
Helpful building block: Nonempty-completion oracle

$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



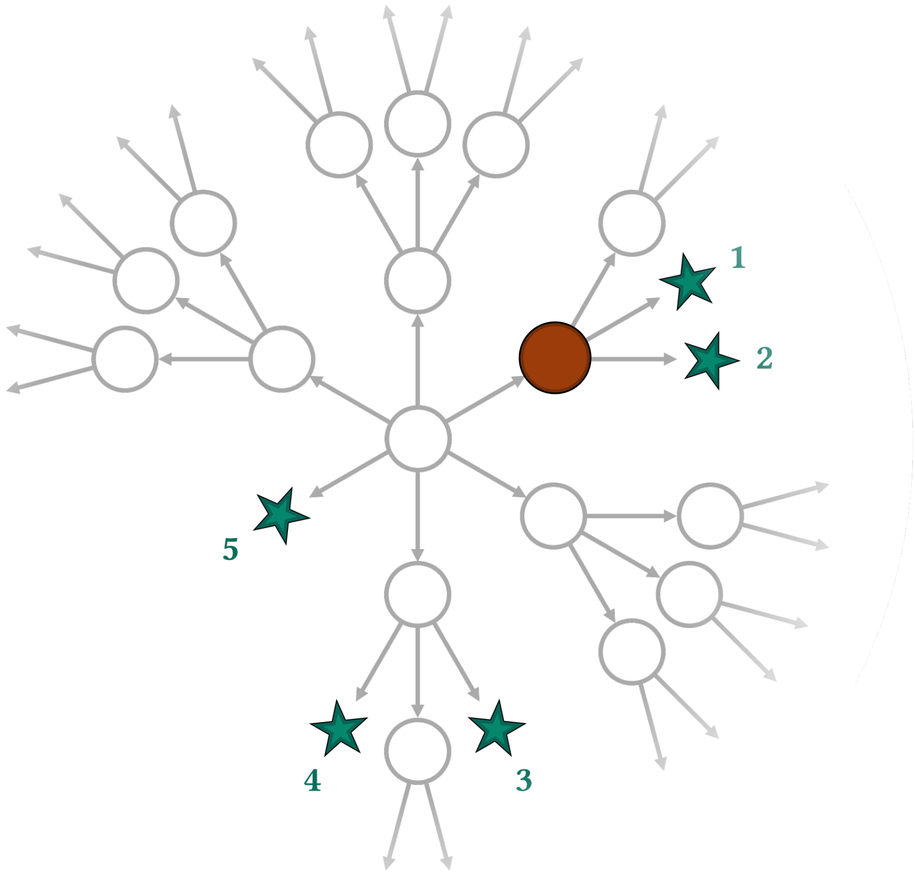
Helpful building block: Nonempty-completion oracle

$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



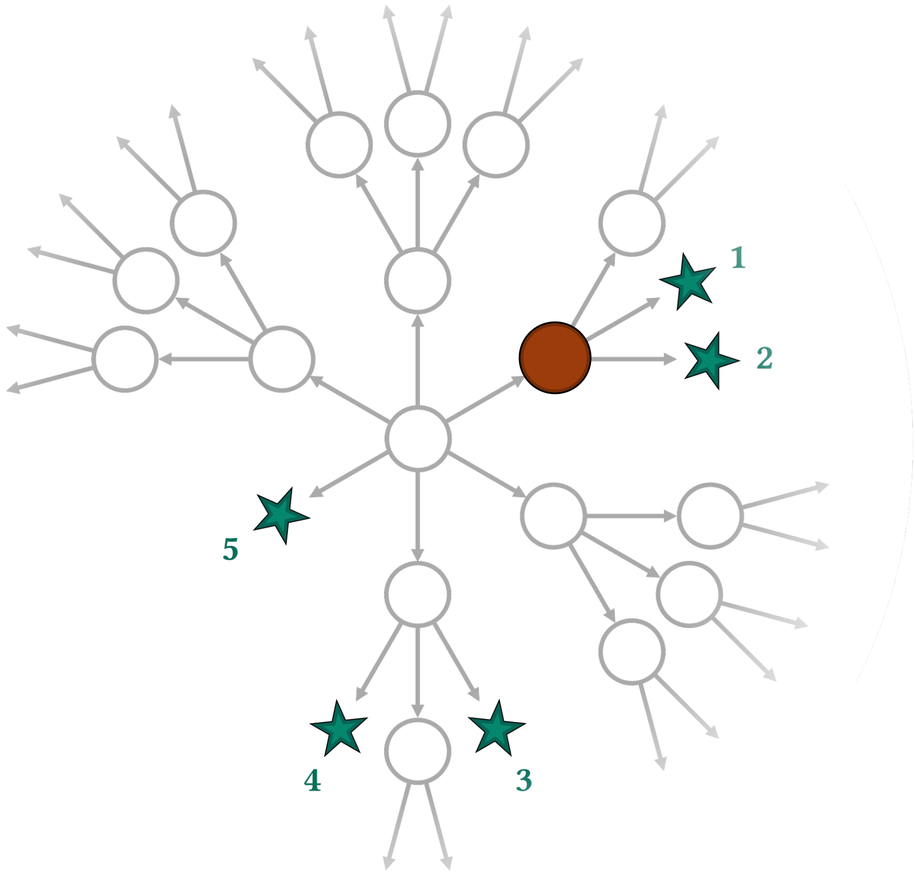
Helpful building block: Nonempty-completion oracle

$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



Helpful building block: Nonempty-completion oracle

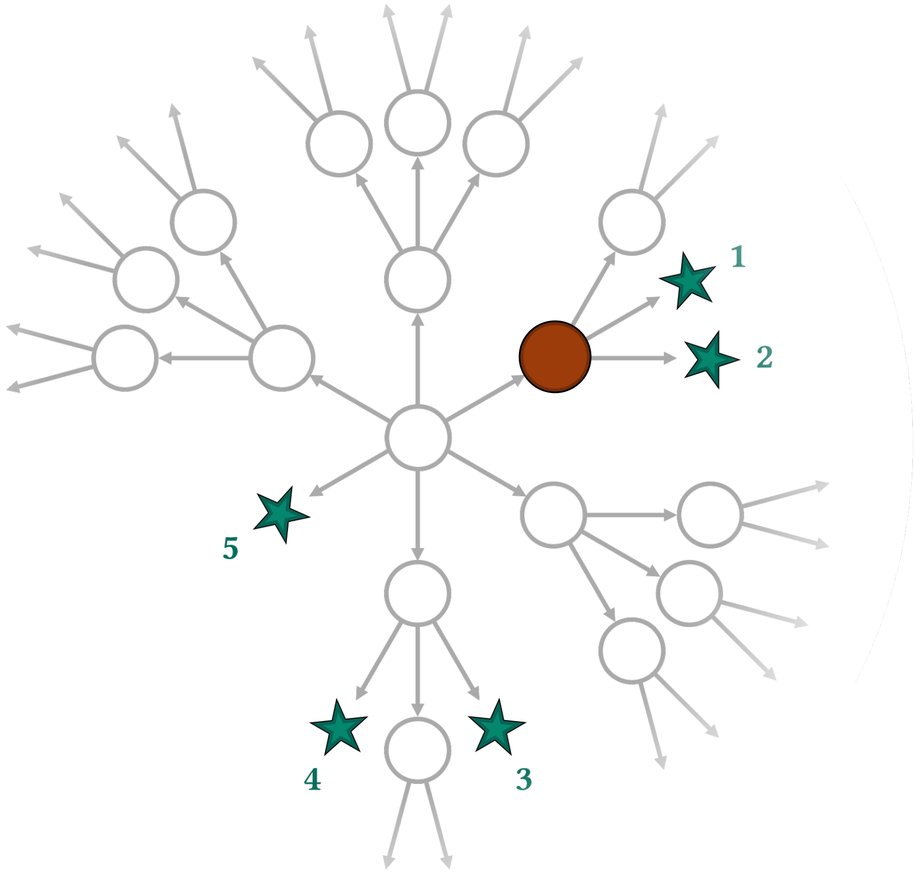
$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



Define via compilation φ to SAT:

Helpful building block: Nonempty-completion oracle

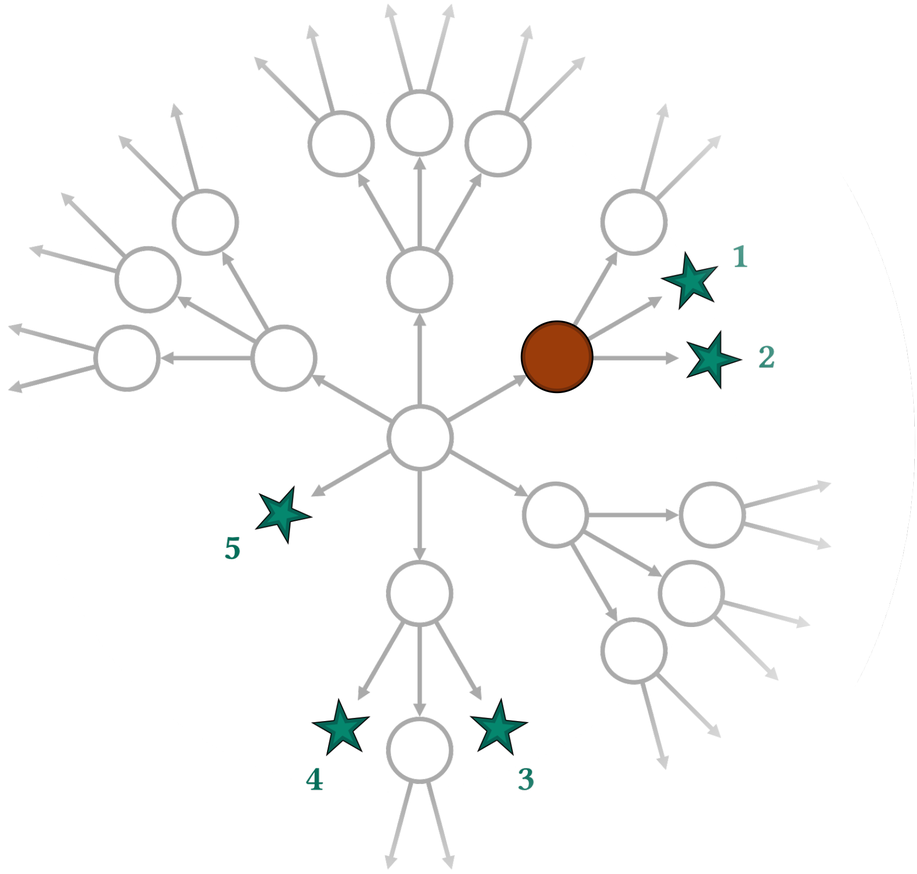
$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



Define via compilation φ to SAT: SAT($\varphi(e)$) iff $C(e) \neq \varnothing$

Helpful building block: Nonempty-completion oracle

$$C(e) = \{e' \mid e \preceq e' \wedge e' \text{ valid}\}$$



Define via compilation φ to SAT: SAT($\varphi(e)$) iff $C(e) \neq \emptyset$

B SAT-Based Oracle to Check for Nonempty Completions

Semantic Formulas for Propositions. Propositions can either be assumed, in which case they are automatically true, or else they are subject to the standard interpretation of Horn clauses in a proof system: a proposition is true iff there exists a rule that fires with that proposition as a conclusion. This is captured by the following two formulas for each $Q \in \mathcal{A}$:

$$\text{assume}_Q \implies \text{true}_Q \quad \text{and} \quad \neg\text{assume}_Q \implies \left(\text{true}_Q \iff \bigvee_{R \text{ with conclusion } Q} \text{true}_R\right)$$

Activity Formulas for Propositions. We start with two necessary conditions for a proposition $Q \in \mathcal{A}$ to be active. First, Q can be active only if it is actually true ($\text{active}_Q \implies \text{true}_Q$). Moreover, if Q is not the goal node, then it must have an active node that uses it:

$$\text{active}_Q \implies \bigvee_{R \text{ with premise } Q} \text{active}_R \qquad \text{(only for non-goal nodes)}$$

Only a single rule that provides a proposition can be active at a given time.

$$\sum_{R \text{ with conclusion } Q} \text{active}_R \leq 1$$

This constraint uses a *cardinality constraint* to limit the number of literals set to true in a given set, which is often directly supported by off-the-shelf SAT solvers; alternatively, since the upper bound of the constraint is 1, it can easily be desugared into standard boolean SAT.

Finally, if Q is to be assumed, then none of the rules that provide Q as a conclusion can be active. Otherwise, Q will be active iff there exists such a rule that is active:

$$\text{assume}_Q \implies \bigwedge_{R \text{ with conclusion } Q} \neg\text{active}_R$$

$$\neg\text{assume}_Q \implies \left(\text{active}_Q \iff \bigvee_{R \text{ with conclusion } Q} \text{active}_R\right)$$

Consistency Formulas for Propositions. For each proposition $Q \in \mathcal{A}$, we add a simple consistency formula that enforces the desired meanings of the label partitions:

$$\begin{cases} \top & \text{if } e(Q) = \bot, \\ \neg\text{assume}_Q & \text{if } e(Q) = \textbf{?}, \\ \neg\text{true}_Q \wedge \neg\text{assume}_Q \wedge \neg\text{active}_Q & \text{if } e(Q) = \textbf{F}, \\ \text{true}_Q & \text{if } e(Q) = \textbf{T/A}, \\ \text{true}_Q \wedge \text{active}_Q & \text{if } e(Q) = \textbf{T/A!}, \\ \text{true}_Q \wedge \neg\text{assume}_Q & \text{if } e(Q) = \textbf{T}, \\ \text{true}_Q \wedge \neg\text{assume}_Q \wedge \text{active}_Q & \text{if } e(Q) = \textbf{T!}, \\ \text{true}_Q \wedge \text{assume}_Q & \text{if } e(Q) = \textbf{A}, \\ \text{true}_Q \wedge \text{assume}_Q \wedge \text{active}_Q & \text{if } e(Q) = \textbf{A!}. \end{cases}$$

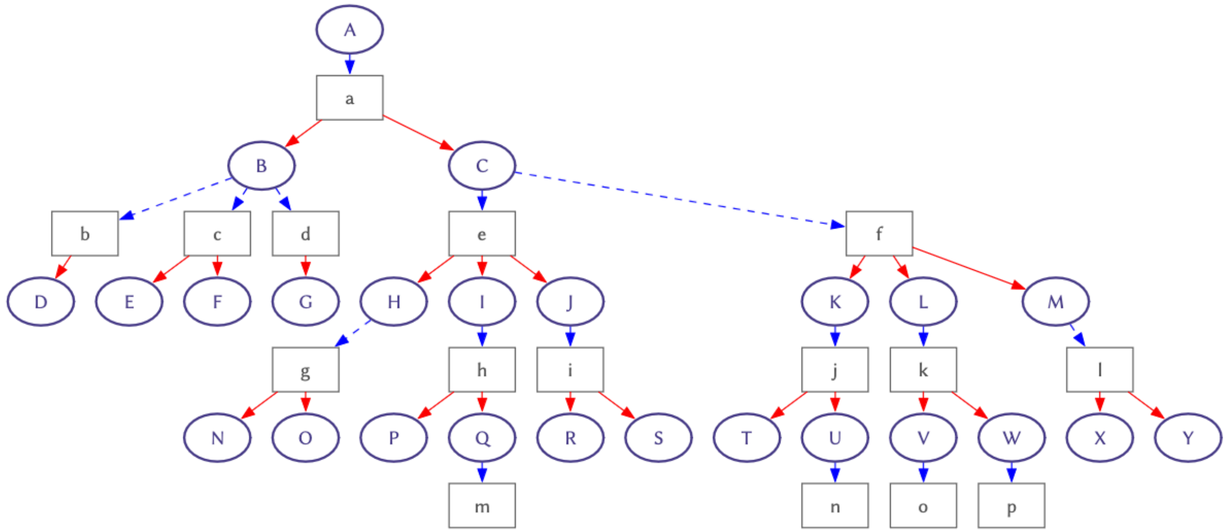
Semantic and Activity Formulas for Rules. Because rules do not have partition labels or assume-semantics, the formulas for them are much simpler:

$$\text{true}_R \iff \bigwedge_{Q \text{ is premise of } R} \text{true}_Q \quad \text{and} \quad \text{active}_R \implies \text{true}_R \quad \text{and} \quad \text{active}_R \implies \bigwedge_{Q \text{ is premise of } R} \text{active}_Q$$

The first formula is standard AND semantics, the second captures that a rule can only be active if it is true, and the third captures that a rule can only be active if all its premises are active.

Warm-up: TOP-DOWN step provider

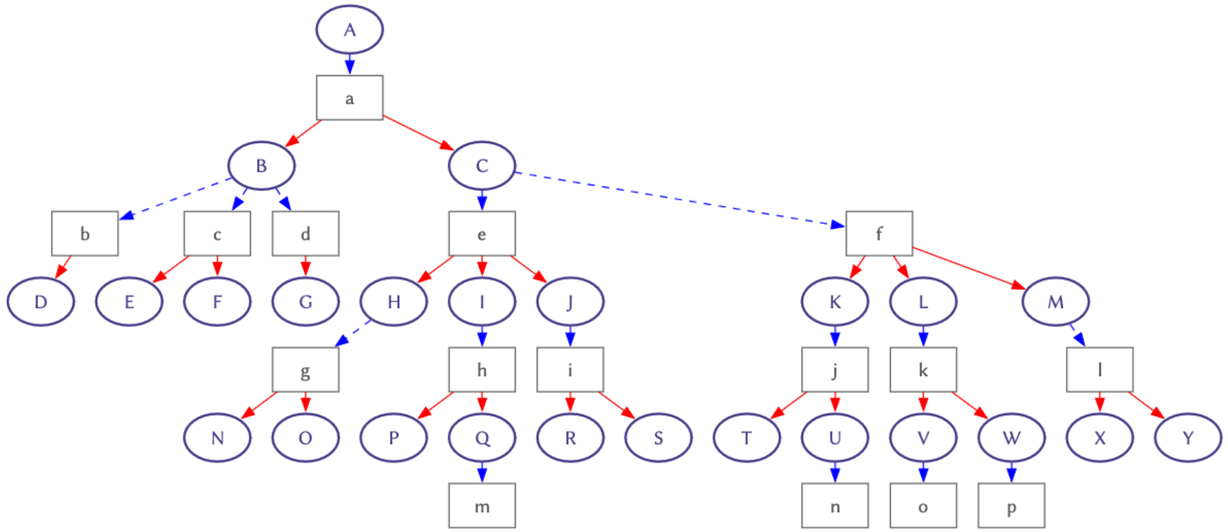
1: **procedure** TOPDOWNINVERSION(e)
2: **yield all from** COMMIT(e)
3: **for** $Q \in e^{-1}[\{\text{T/A!}, \text{T!}\}]$ **do**
4: **for** $R \in \mathcal{R}$ with conclusion Q **do**
5: $P \leftarrow$ premises of R with label $\perp$
6: **if** P empty **then continue**
7: $\sigma \leftarrow P_1 := \text{T/A!}\,; \cdots ; P_{|P|} := \text{T/A!}$
8: **if** $e(Q) = \text{T/A!}$ **then** $\sigma \leftarrow \sigma\,; Q := \text{T!}$
9: **if** SAT($\varphi(\sigma e)$) **then yield** σ



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

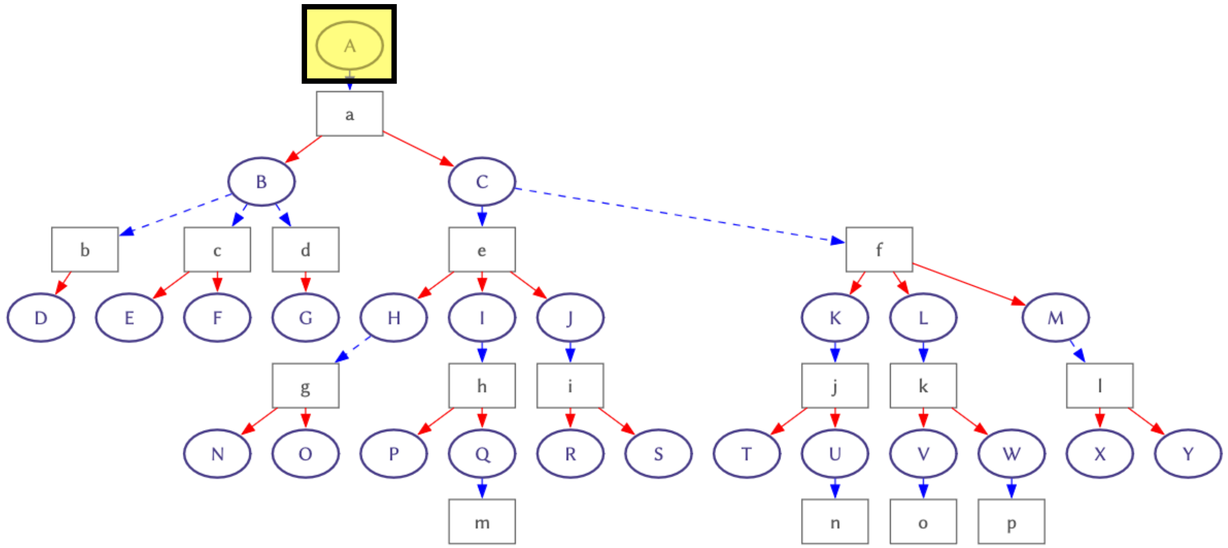
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

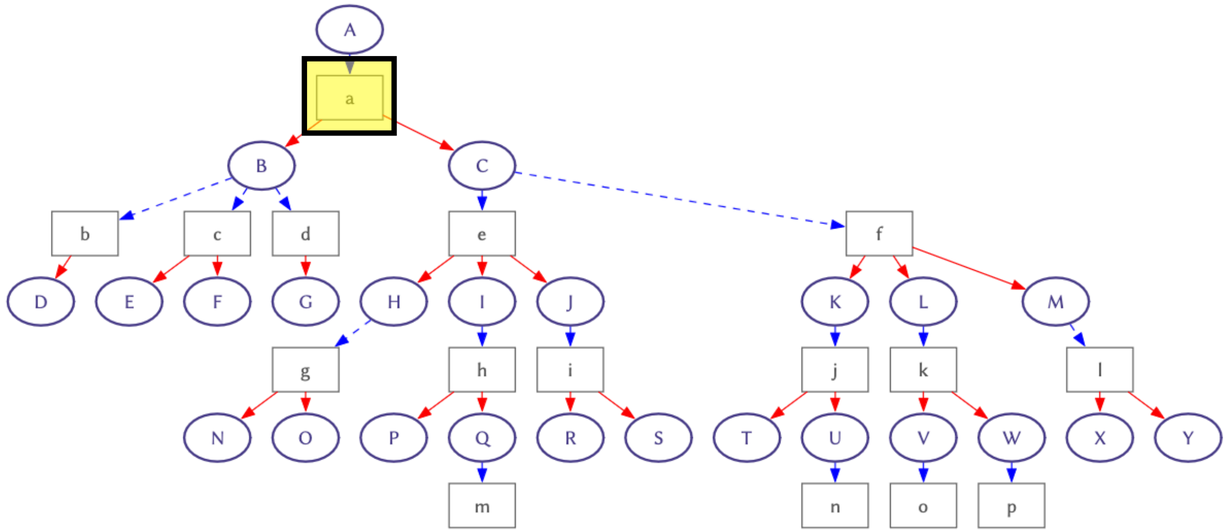
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

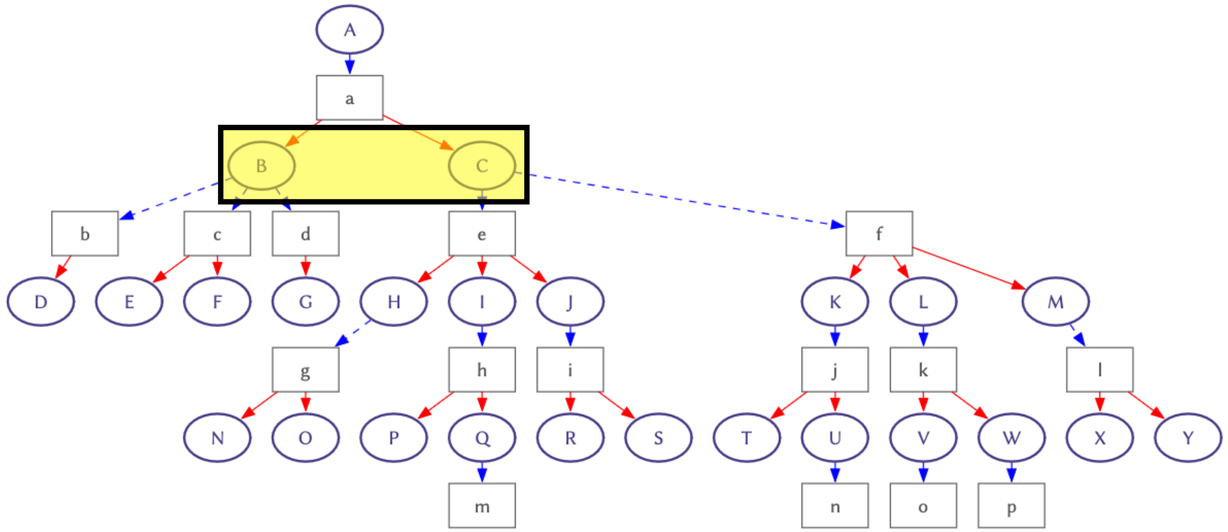
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

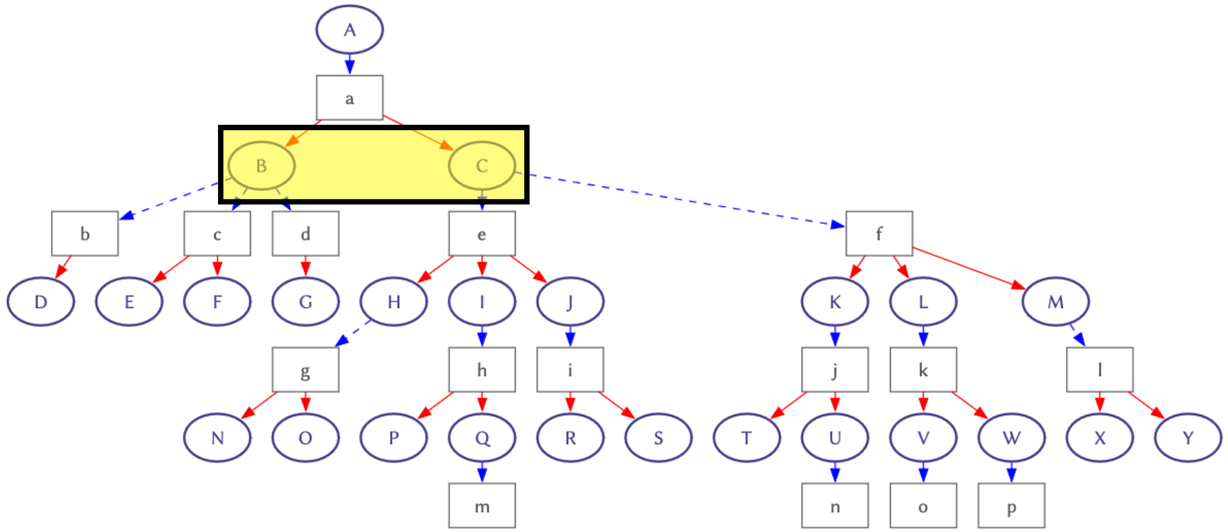
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```

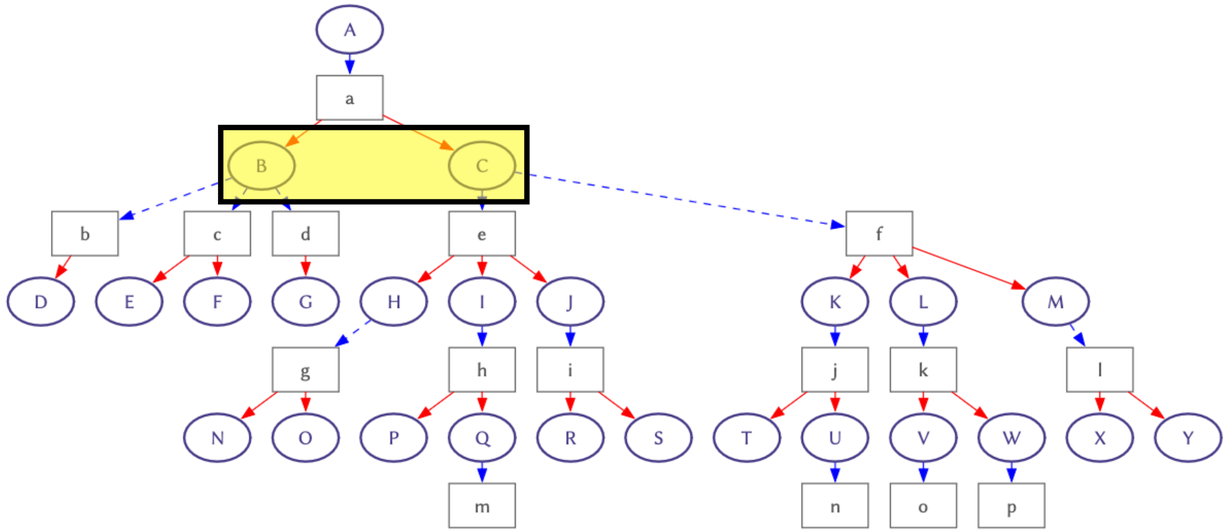


(User selection)

Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

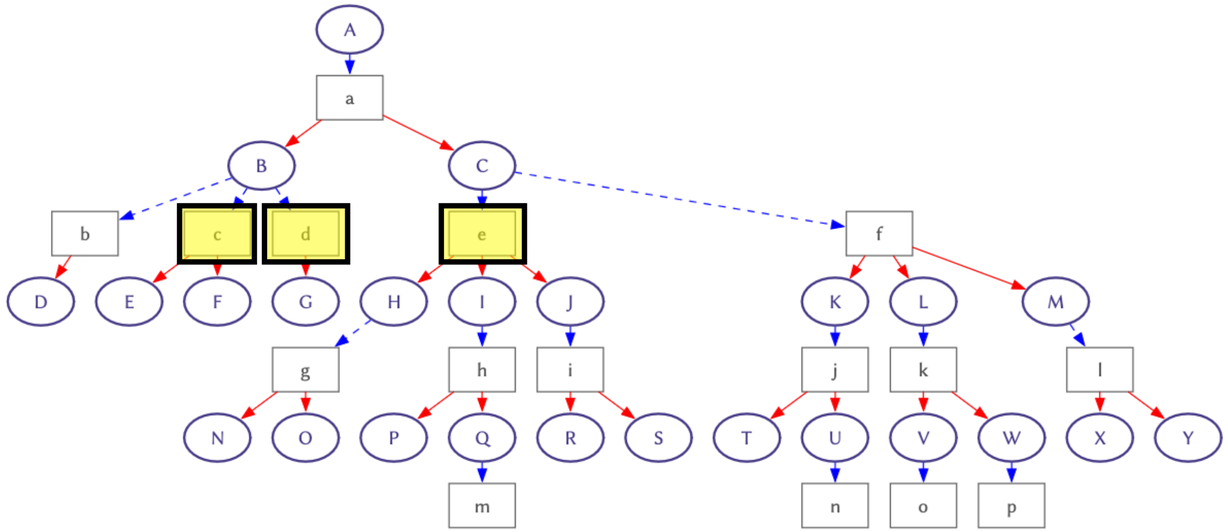
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

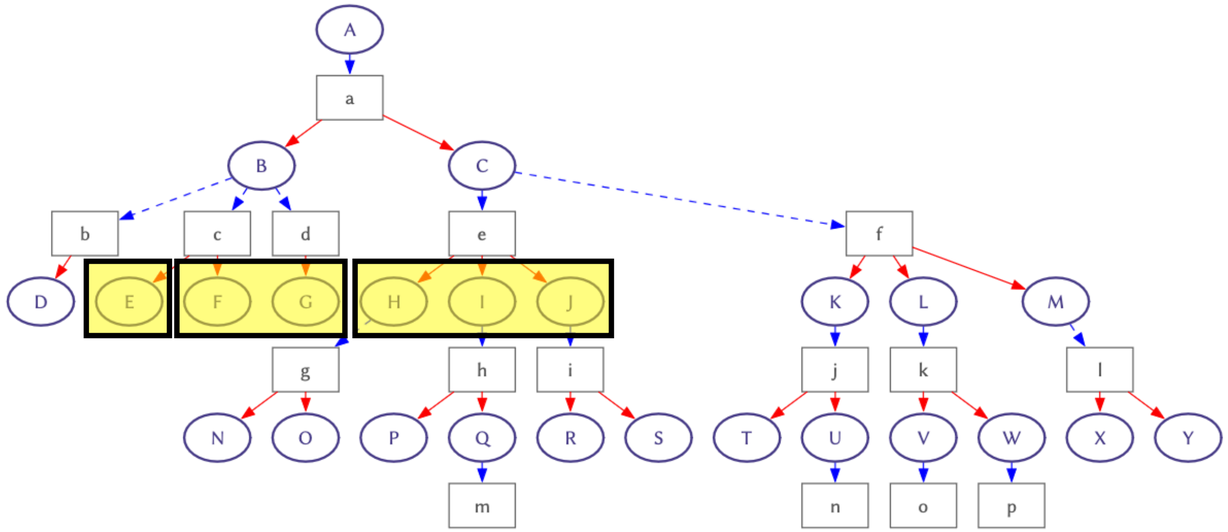
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

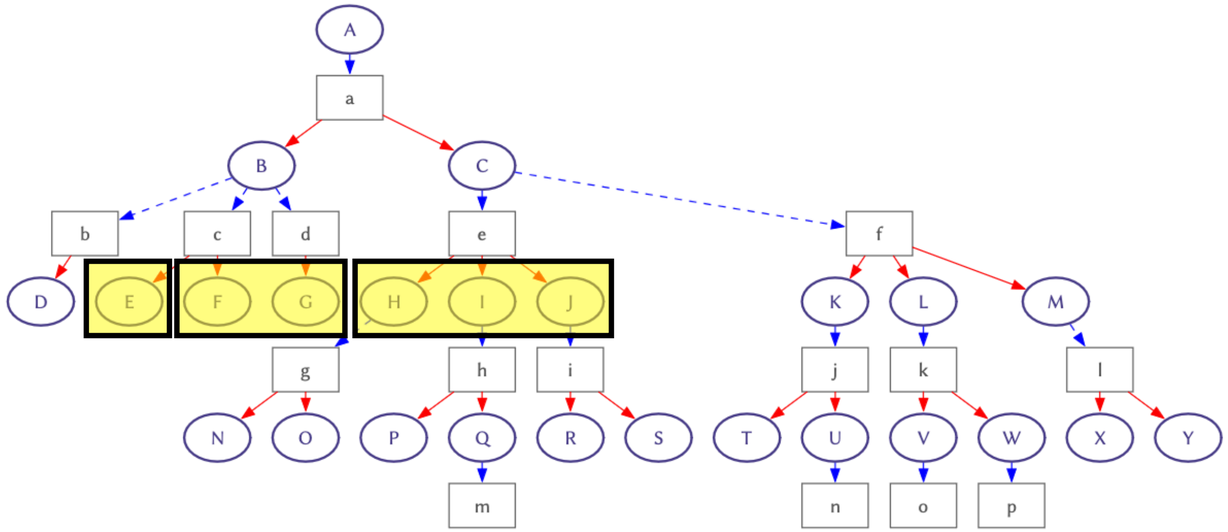
```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



Warm-up: TOP-DOWN step provider

(*i.e.*, we don't lose anything with Programming by Navigation)

```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```

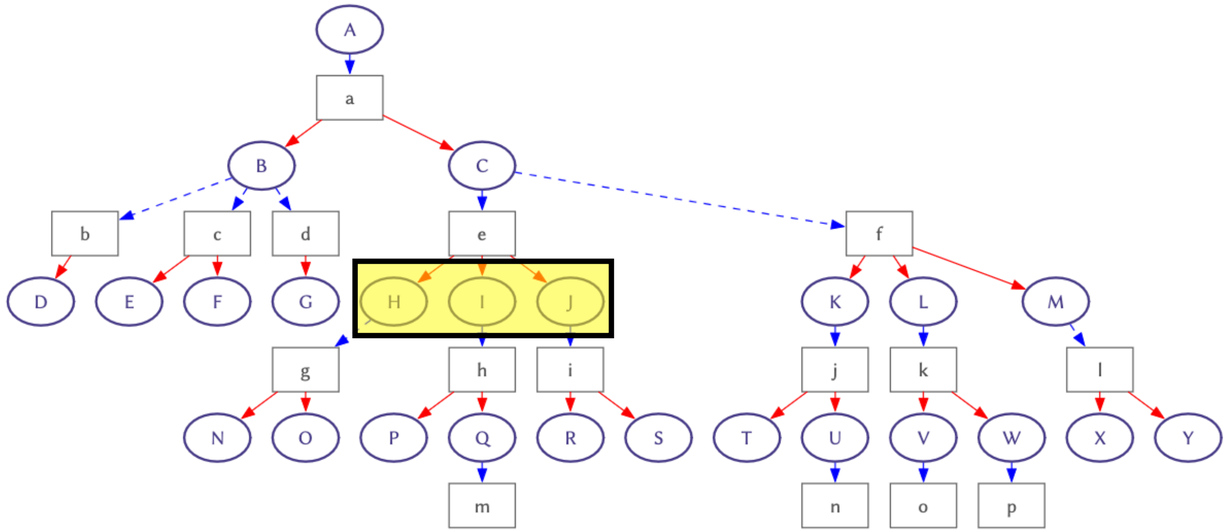


(User selection)

Warm-up: TOP-DOWN step provider

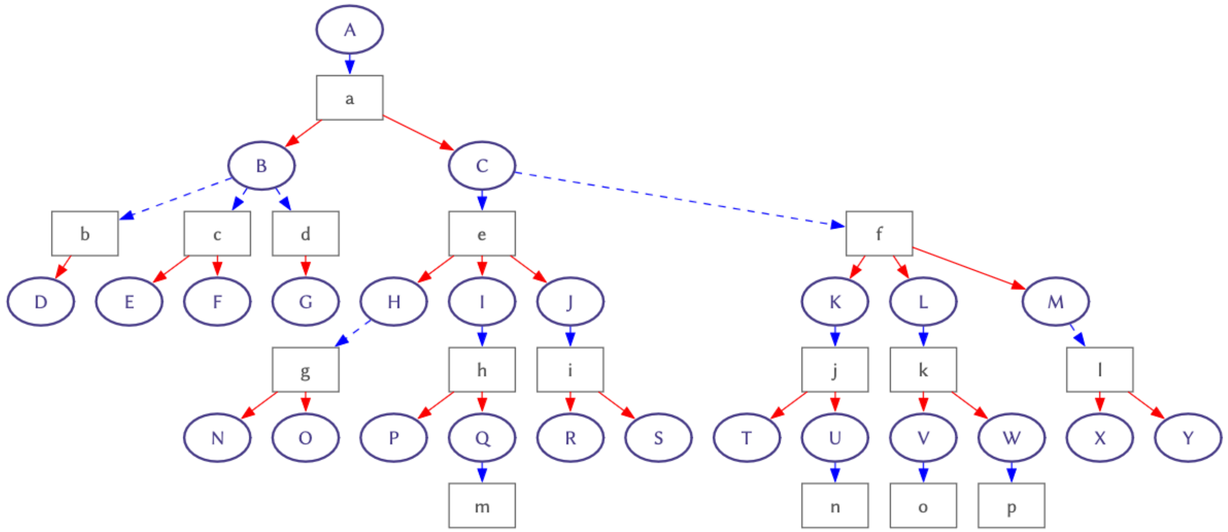
(*i.e.*, we don't lose anything with Programming by Navigation)

```
1: procedure TopDownInversion(e)
2:     yield all from Commit(e)
3:     for Q ∈ e^{-1}[{T/A!, T!}] do
4:         for R ∈ ℛ with conclusion Q do
5:             P ← premises of R with label ⊥
6:             if P empty then continue
7:             σ ← P_1 := T/A! ; ··· ; P_|P| := T/A!
8:             if e(Q) = T/A! then σ ← σ ; Q := T!
9:             if SAT(φ(σe)) then yield σ
```



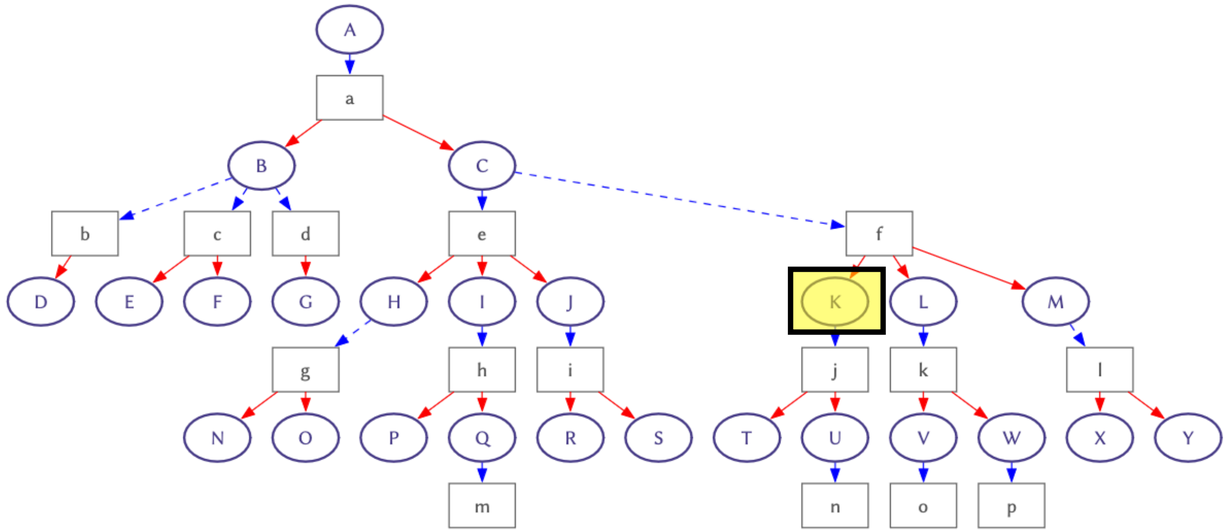
Non-topological: RANDOM step provider

1: **procedure** RANDOM(e)
2: $Q \leftarrow$ random element of $e^{-1}[\{\ell \in L : \ell \text{ non-maximal}\}]$
3: **for** $\ell > e(Q)$ **do**
4: $\sigma \leftarrow Q := \ell$
5: **if** SAT($\varphi(\sigma e)$) **then yield** σ



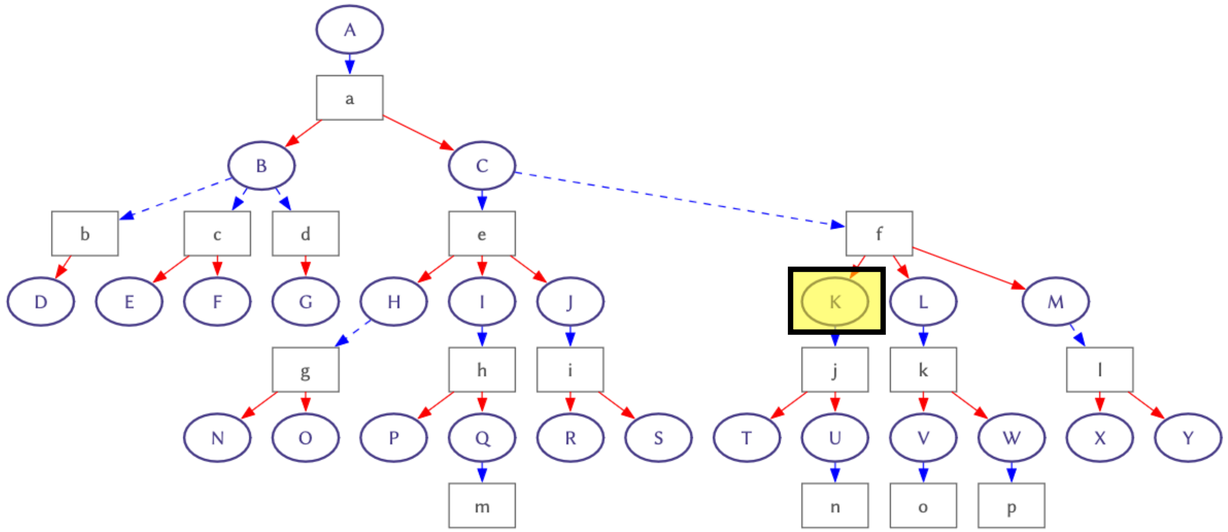
Non-topological: RANDOM step provider

```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```



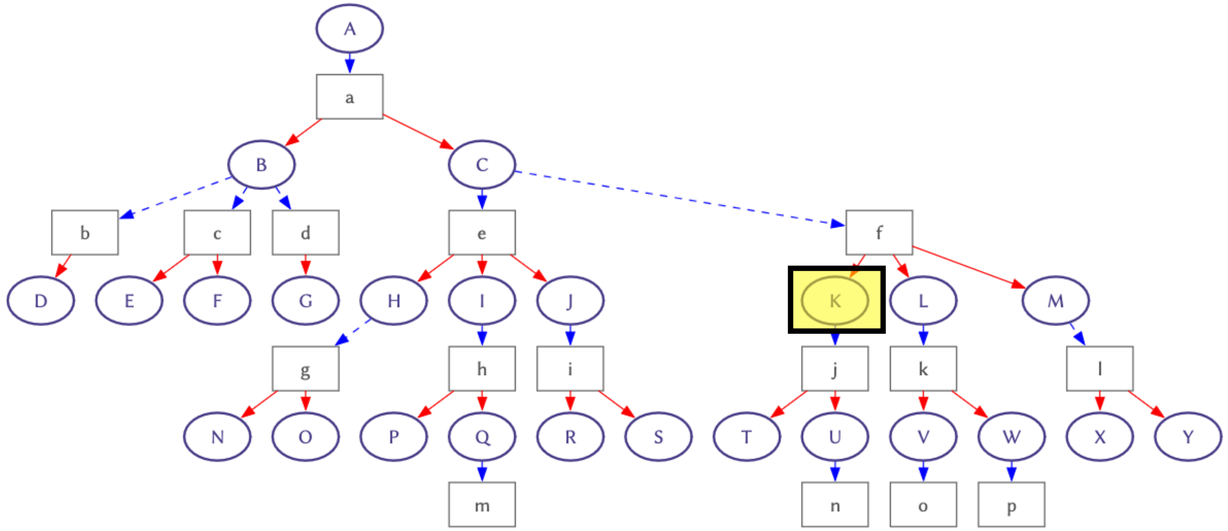
Non-topological: RANDOM step provider

```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```



Non-topological: RANDOM step provider

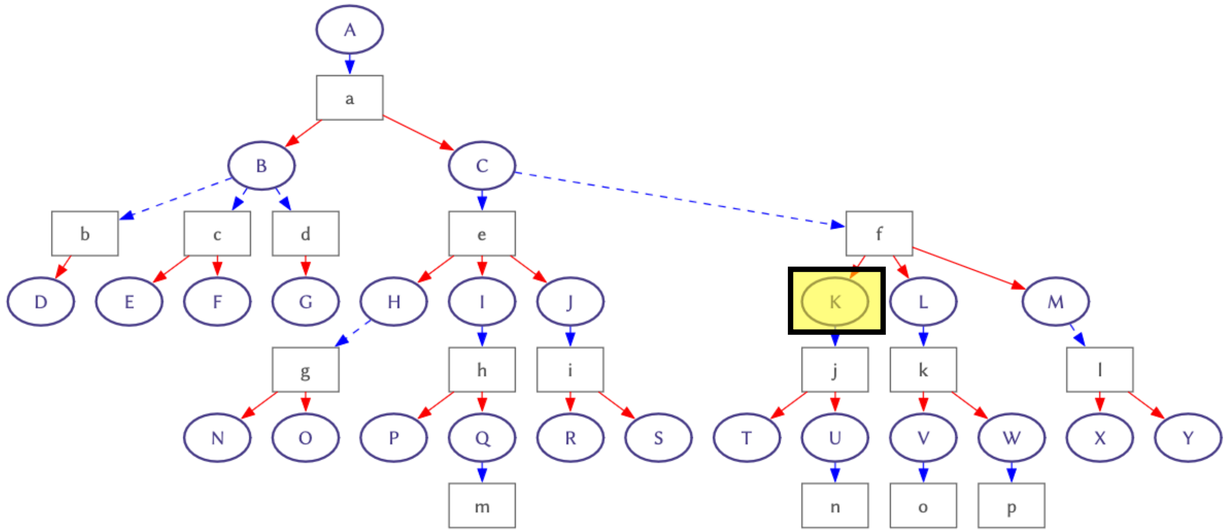
1: **procedure** RANDOM(e)
2: $Q \leftarrow$ random element of $e^{-1}[\{\ell \in L : \ell \text{ non-maximal}\}]$
3: **for** $\ell > e(Q)$ **do**
4: $\sigma \leftarrow Q := \ell$
5: **if** SAT($\varphi(\sigma e)$) **then yield** σ



Even though the **propositions** are randomly selected,
the **labels** are still filtered to satisfy **Strong Soundness**

Non-topological: RANDOM step provider

1: **procedure** RANDOM(e)
2: $Q \leftarrow$ random element of $e^{-1}[\{\ell \in L : \ell \text{ non-maximal}\}]$
3: **for** $\ell > e(Q)$ **do**
4: $\sigma \leftarrow Q := \ell$
5: **if** SAT($\varphi(\sigma e)$) **then yield** σ

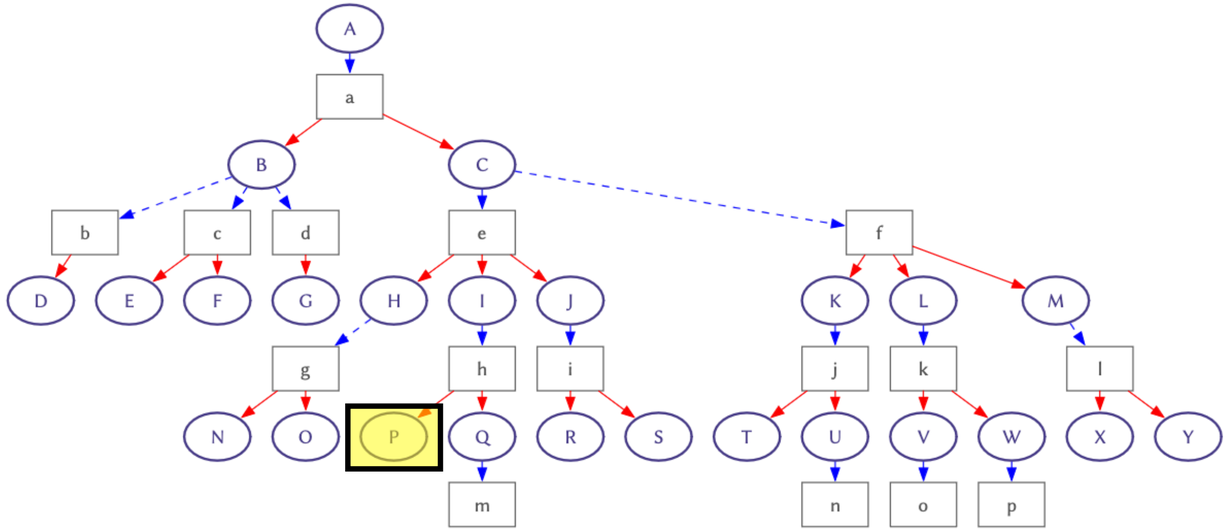


Even though the **propositions** are randomly selected, the **labels** are still filtered to satisfy **Strong Soundness**

(User selection)

Non-topological: RANDOM step provider

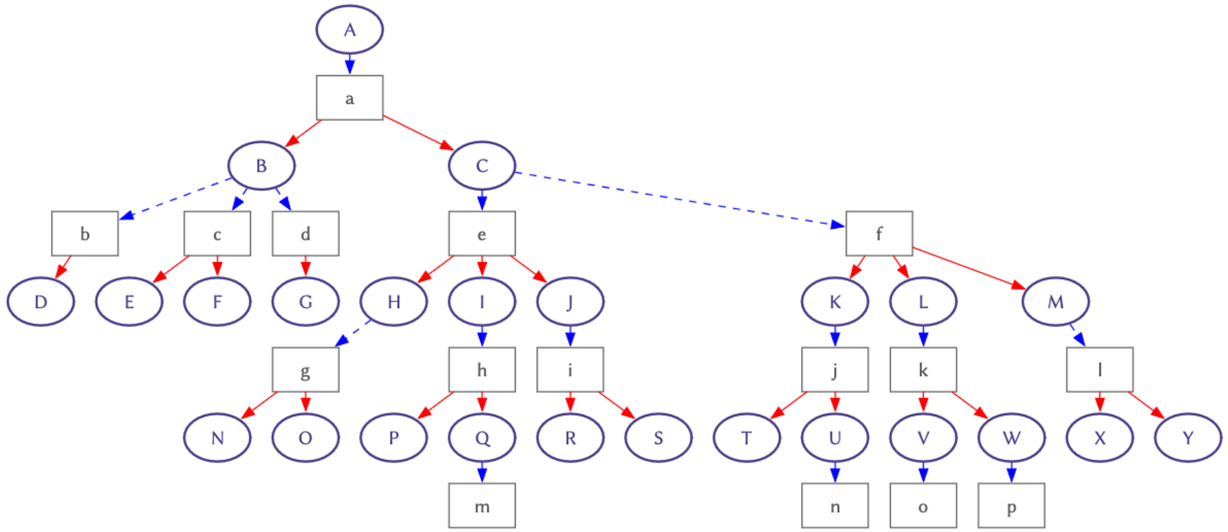
```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```



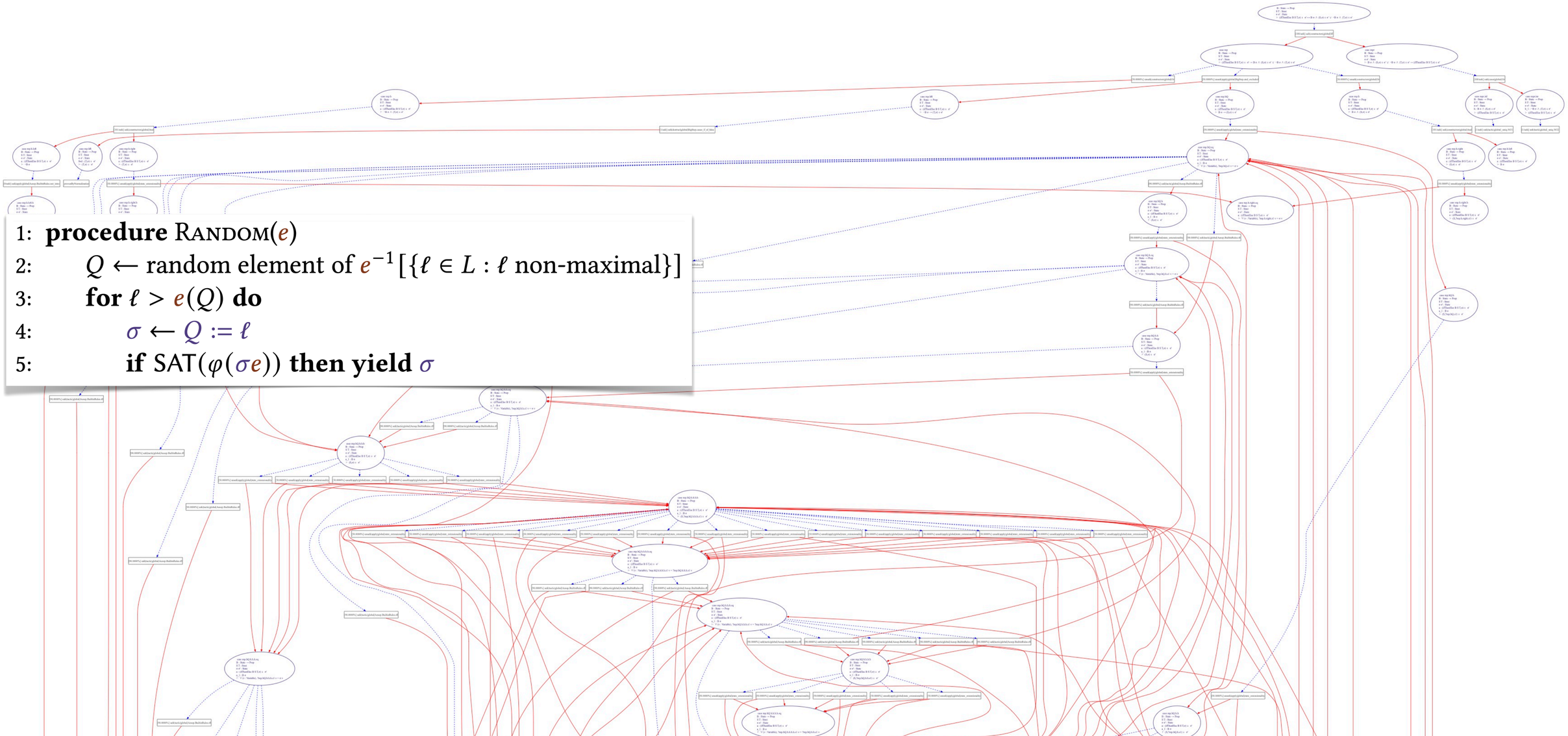
Even though the **propositions** are randomly selected,
the **labels** are still filtered to satisfy **Strong Soundness**

Non-topological: CUT step provider

1: **procedure** RANDOM(e)
2: $Q \leftarrow$ random element of $e^{-1}[\{\ell \in L : \ell \text{ non-maximal}\}]$
3: **for** $\ell > e(Q)$ **do**
4: $\sigma \leftarrow Q := \ell$
5: **if** SAT($\varphi(\sigma e)$) **then yield** σ

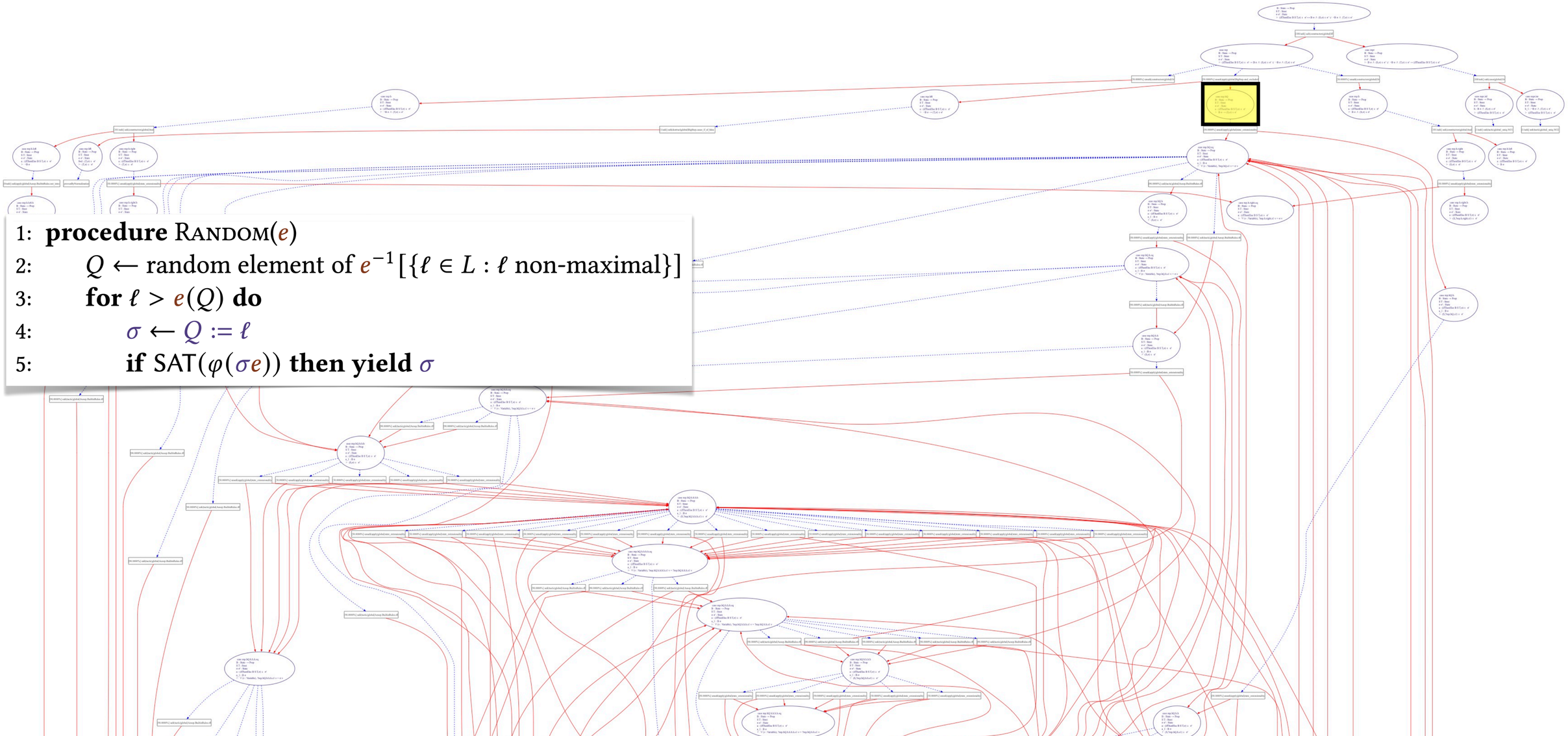


Non-topological: CUT step provider



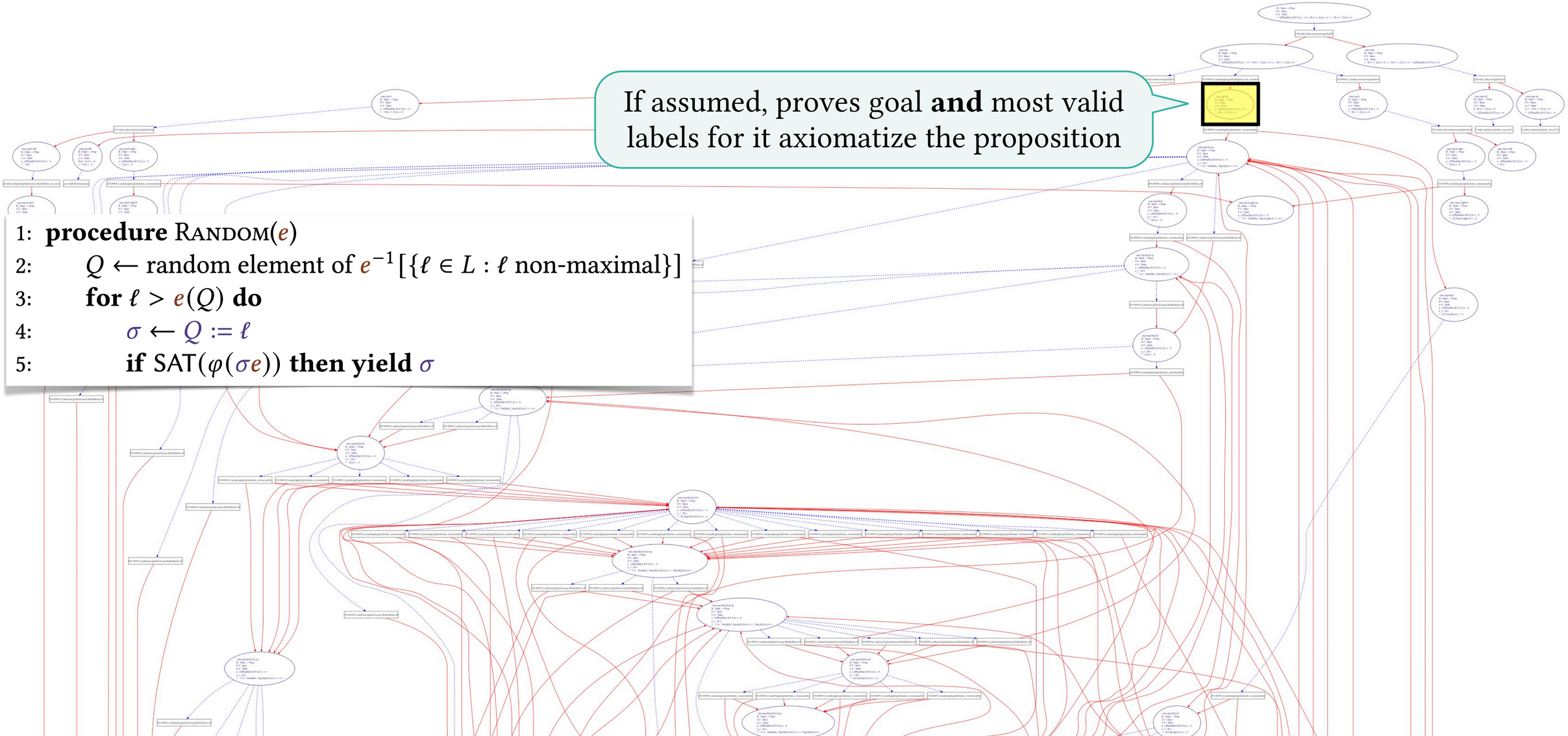
```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```

Non-topological: CUT step provider



```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```

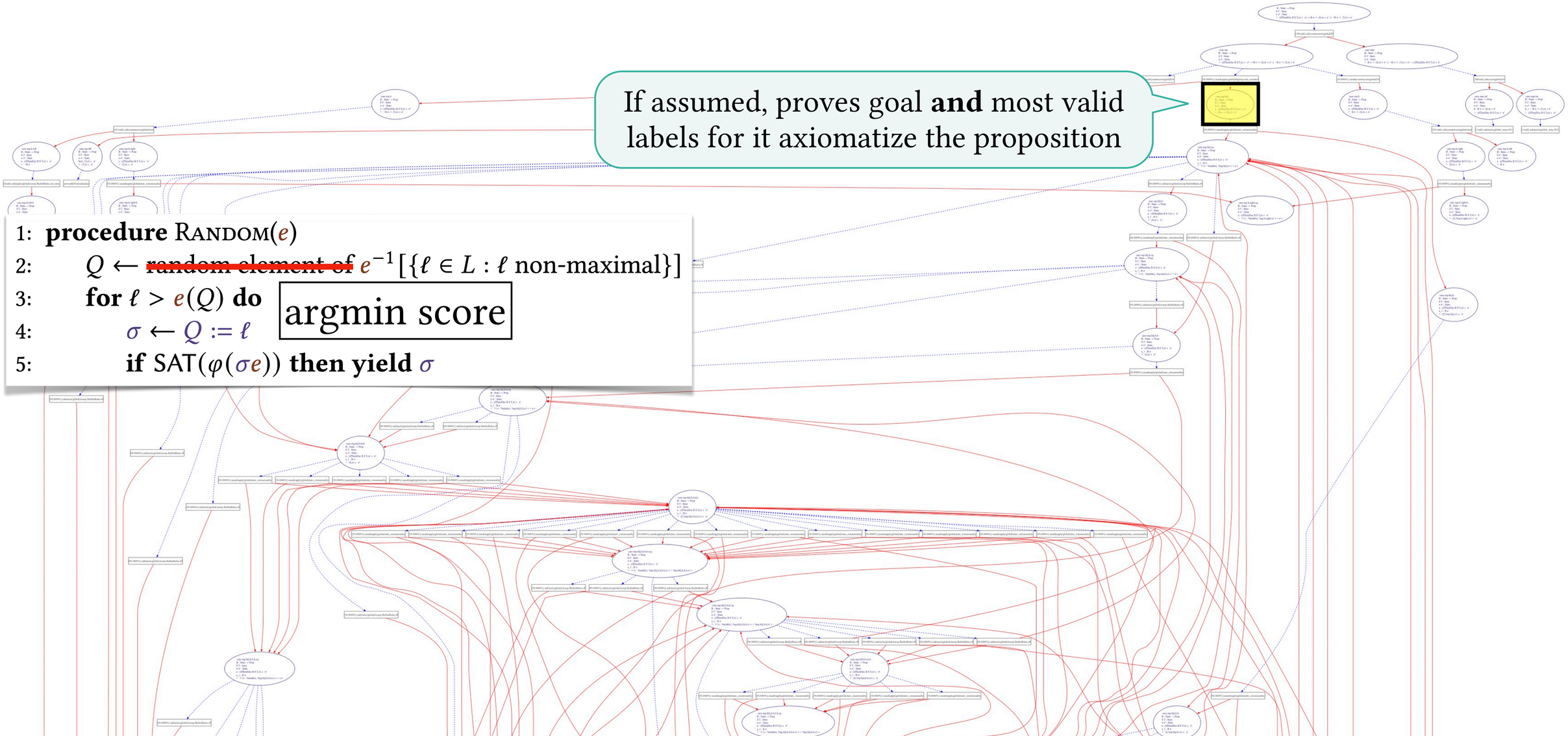
Non-topological: CUT step provider



If assumed, proves goal **and** most valid labels for it axiomatize the proposition

```
1: procedure RANDOM(e)
2:     Q ← random element of e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```

Non-topological: CUT step provider

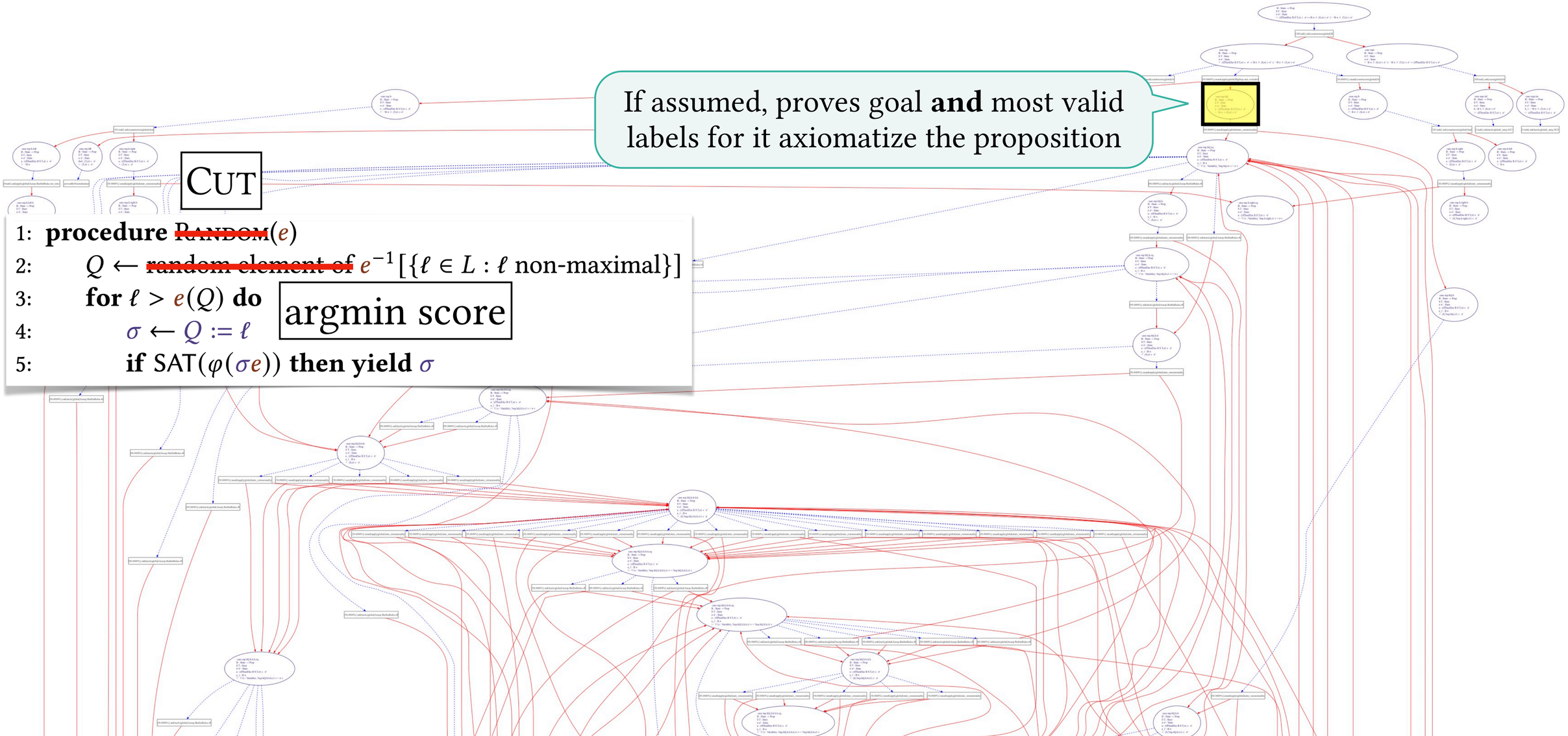


If assumed, proves goal **and** most valid labels for it axiomatize the proposition

procedure RANDOM(e)
  $Q \leftarrow$ ~~**random element of**~~ $e^{-1}[\{\ell \in L : \ell \text{ non-maximal}\}]$
  for $\ell > e(Q)$ **do**
    $\sigma \leftarrow Q := \ell$
    if SAT($\varphi(\sigma e)$) **then yield** σ

argmin score

Non-topological: CUT step provider



Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

$$\texttt{Fuse}\{\forall_{ikj} A_{ij} = B_{ik} C_{kj}$$

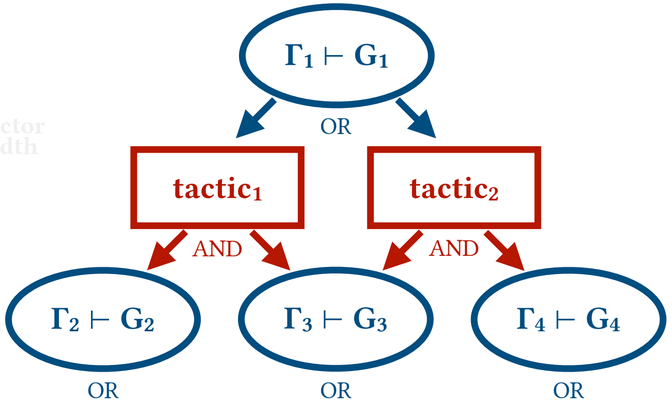
Parallelization factor ? ? Memory layout

? Vector width

$$\forall_{?} E_{ij} = A_{ik} D_{kj}\}$$

Dataflow ordering

? Memory layout



Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression

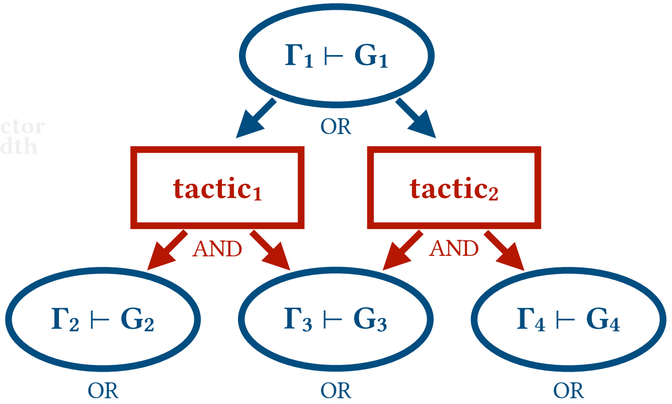
$$\texttt{Fuse}\{\forall_{ikj} A_{ij} = B_{ik} C_{kj}\quad \forall_{?} E_{ij} = A_{ik} D_{kj}\}$$

Parallelization factor ? ? Memory layout

? Vector width

Dataflow ordering

? Memory layout

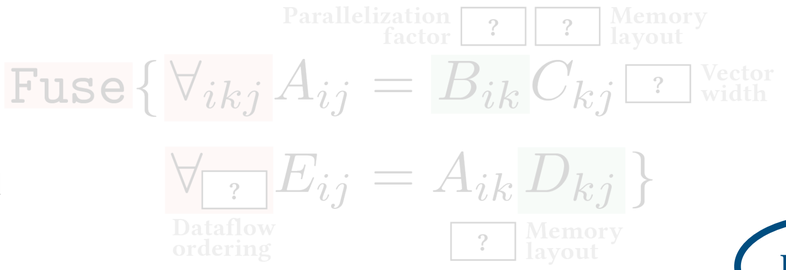


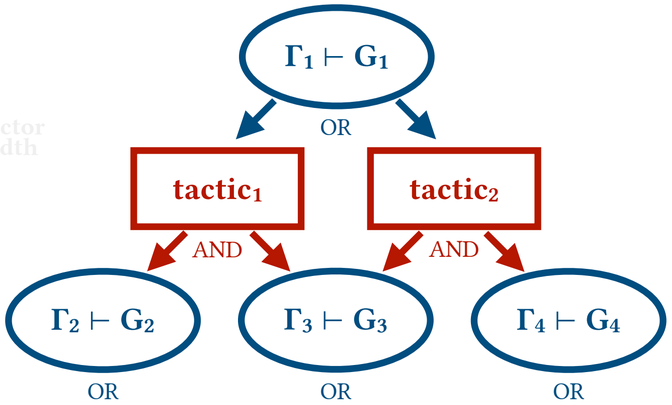
Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	Maximum Information Gain, Cut, Random, Top-Down, Bottom-Up, Minimal Leaf Heuristic, …

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression



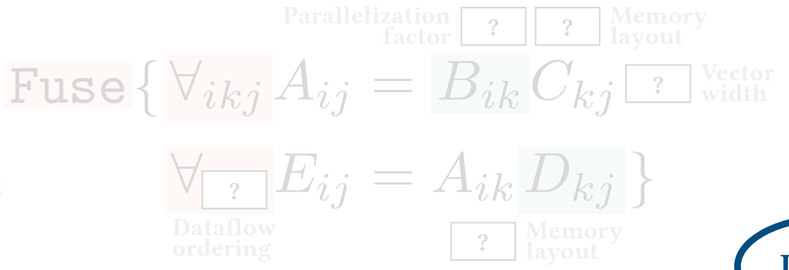


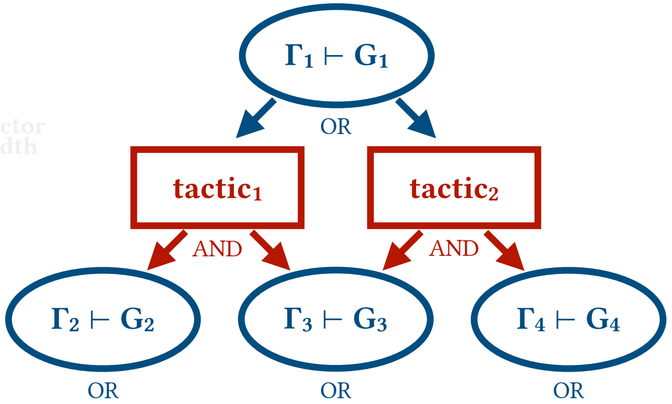
Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	Maximum Information Gain, Cut, Random, Top-Down, Bottom-Up, Minimal Leaf Heuristic, …
Nonempty-completion oracle	Datalog	FuseFlow compiler + Datalog	

Programming by Navigation Ingredients

Day 2: RNA-seq

Goal: Differential gene expression





Expressions	Bioinformatics pipelines	FuseFlow compiler schedules	Proof system modifications
Validity	Meets user-defined validity conditions	Meets FuseFlow-defined validity conditions	Modification proves final goal node
Step provider	Top-down classical–constructive synthesis	Big-step synthesis	Maximum Information Gain, Cut, Random, Top-Down, Bottom-Up, Minimal Leaf Heuristic, …
Nonempty-completion oracle	Datalog	FuseFlow compiler + Datalog	SAT

Navigating AND–OR Graph Modifications to Debug Failing Proof Search


Marlena Preigh


Max Willsey


Sarah E. Chasins

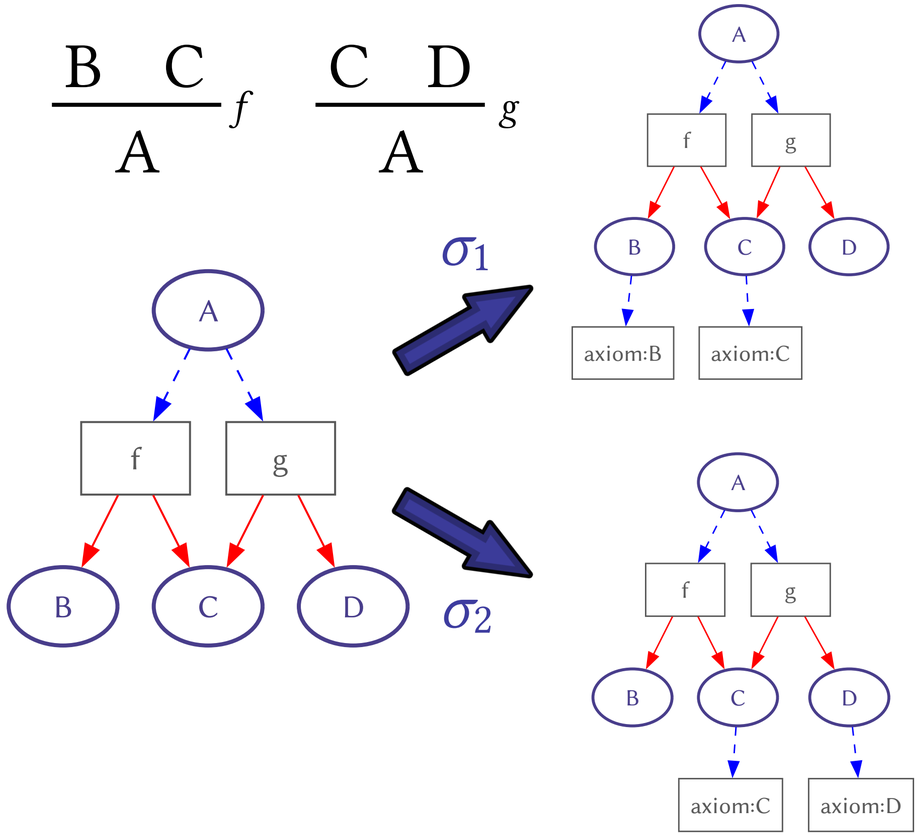
justinlubin@berkeley.edu

Expressions — Proof system modifications

Validity — Modification proves final goal node

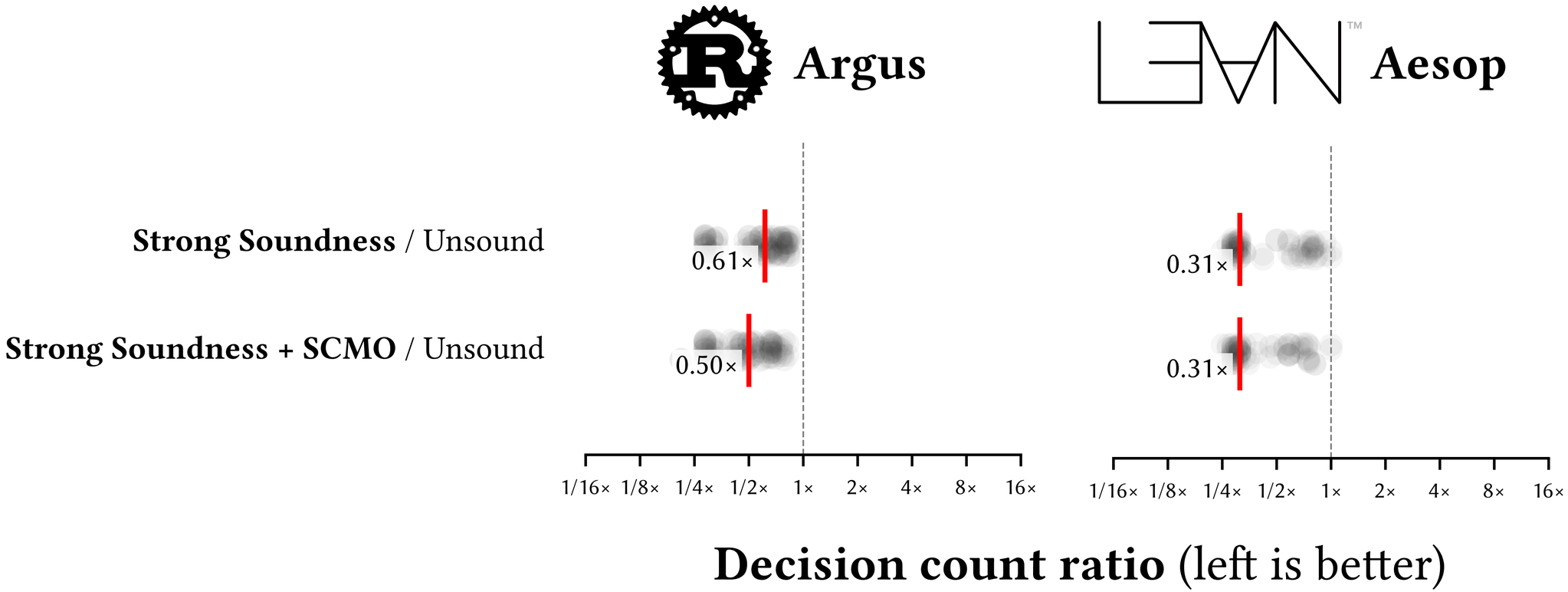
Step provider — Maximum Information Gain, Cut, Random, Top-Down, Bottom-Up, Minimal Leaf Heuristic, …

Nonempty-completion oracle — SAT



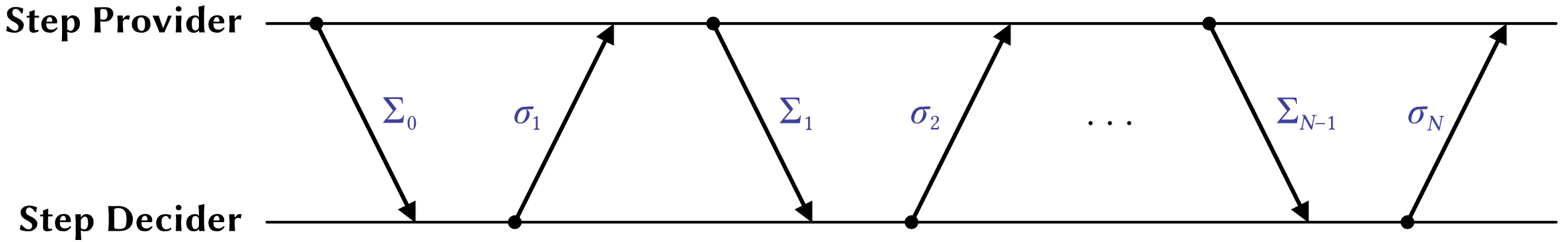
Extra Slides

Strong Completeness Modulo Observability (SCMO) reduces decision counts



Gray, Crichton, and Krishnamurthi. An Interactive Debugger for Rust Trait Errors *(PLDI 2025).*

Limperg and From. Aesop: White-Box Best-First Proof Search for Lean *(CPP 2023).*



CUT

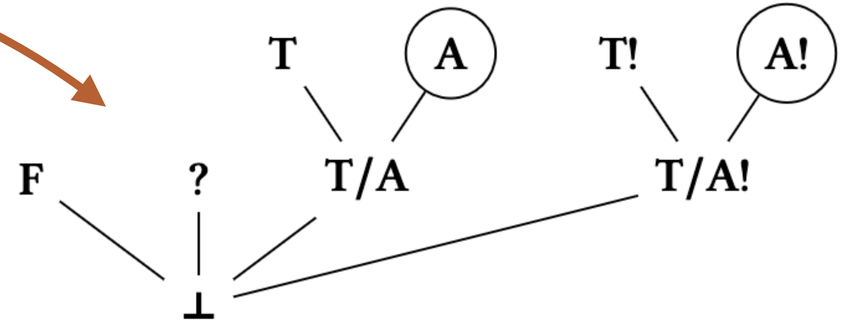
```
1: procedure ~~RANDOM~~(e)
2:     Q ← ~~random element of~~ e^{-1}[{ℓ ∈ L : ℓ non-maximal}]
3:     for ℓ > e(Q) do
4:         σ ← Q := ℓ
5:         if SAT(φ(σe)) then yield σ
```

argmin score

Syntax, semantics, and steps for assume-partitions

- Expression e: Partition of propositions
  - *Notation:* $e(Q)$ is label of proposition Q in e
  - Label of goal is **T!** (must be true and used, but not assumed)
- Semantics $[\![e]\!]$: Axiomatizes nodes labeled with **A** or **A!**
- e valid iff:
  1. $e(Q)$ maximal for all Q
  2. $[\![e]\!]\ \mathbf{P} \vdash Q$ for $e(Q) = \mathbf{T}$ or $e(Q) = \mathbf{T!}$
  3. $[\![e]\!]\ \mathbf{P} \nvdash Q$ for $e(Q) = \mathbf{F}$
  4. There exists a derivation of goal containing all propositions Q with $e(Q) = \mathbf{T!}$ or $e(Q) = \mathbf{A!}$
- Step $\sigma ::= Q := l \mid \sigma_1 \,;\, \sigma_2$

with these labels



$$\frac{e(Q) < \ell}{e \xrightarrow{Q:=\ell} [Q := \ell]e}\ \textsc{Step/Set}$$

$$\frac{e_1 \xrightarrow{\sigma_1} e_2 \qquad e_2 \xrightarrow{\sigma_2} e_3}{e_1 \xrightarrow{\sigma_1 \,;\, \sigma_2} e_3}\ \textsc{Step/Seq}$$

with these semantics

Model proof search debugging as interactive program synthesis (specification refinement)

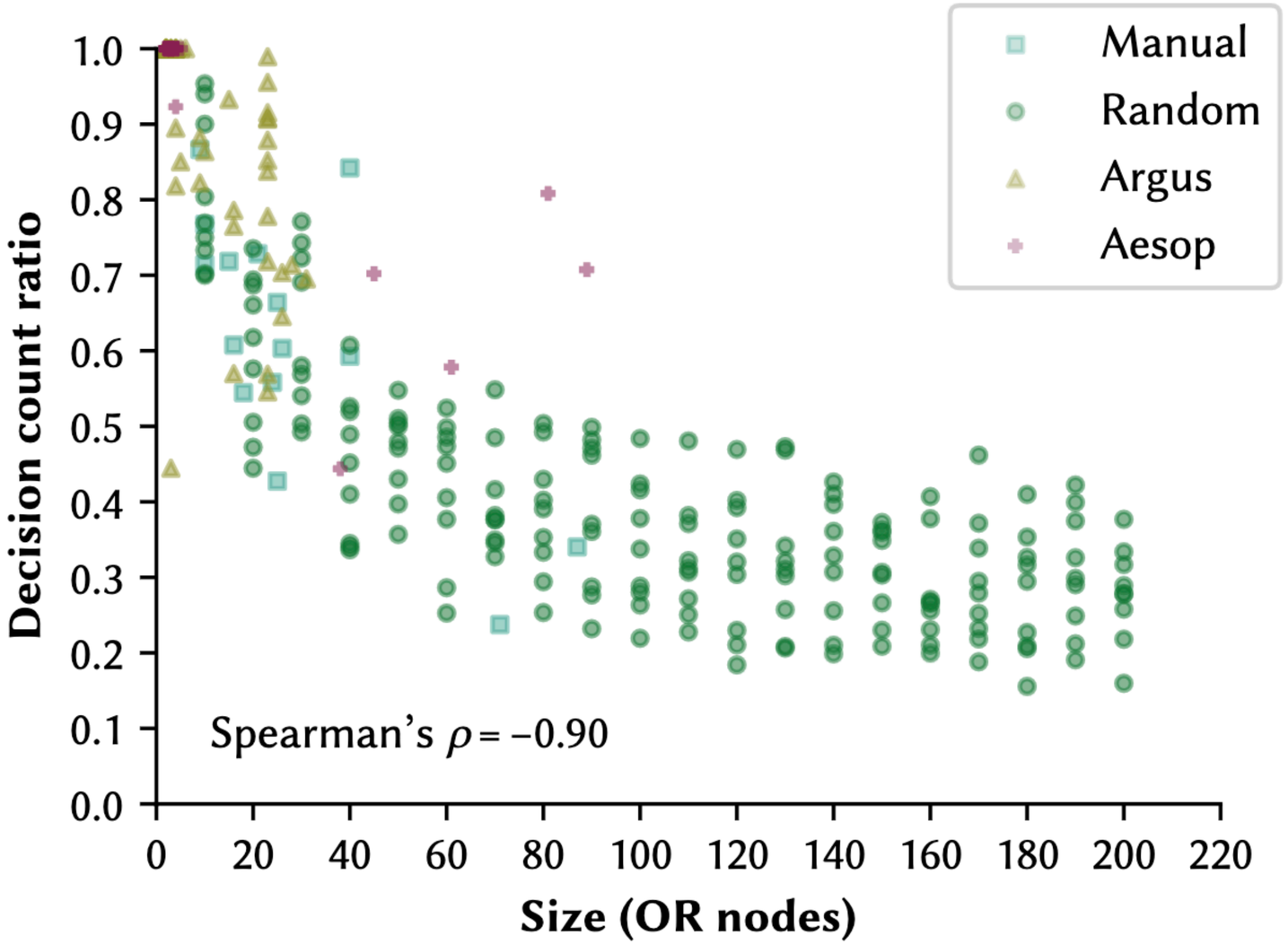
Goal: Synthesize modification to proof system that proves the goal… but not just any modification!

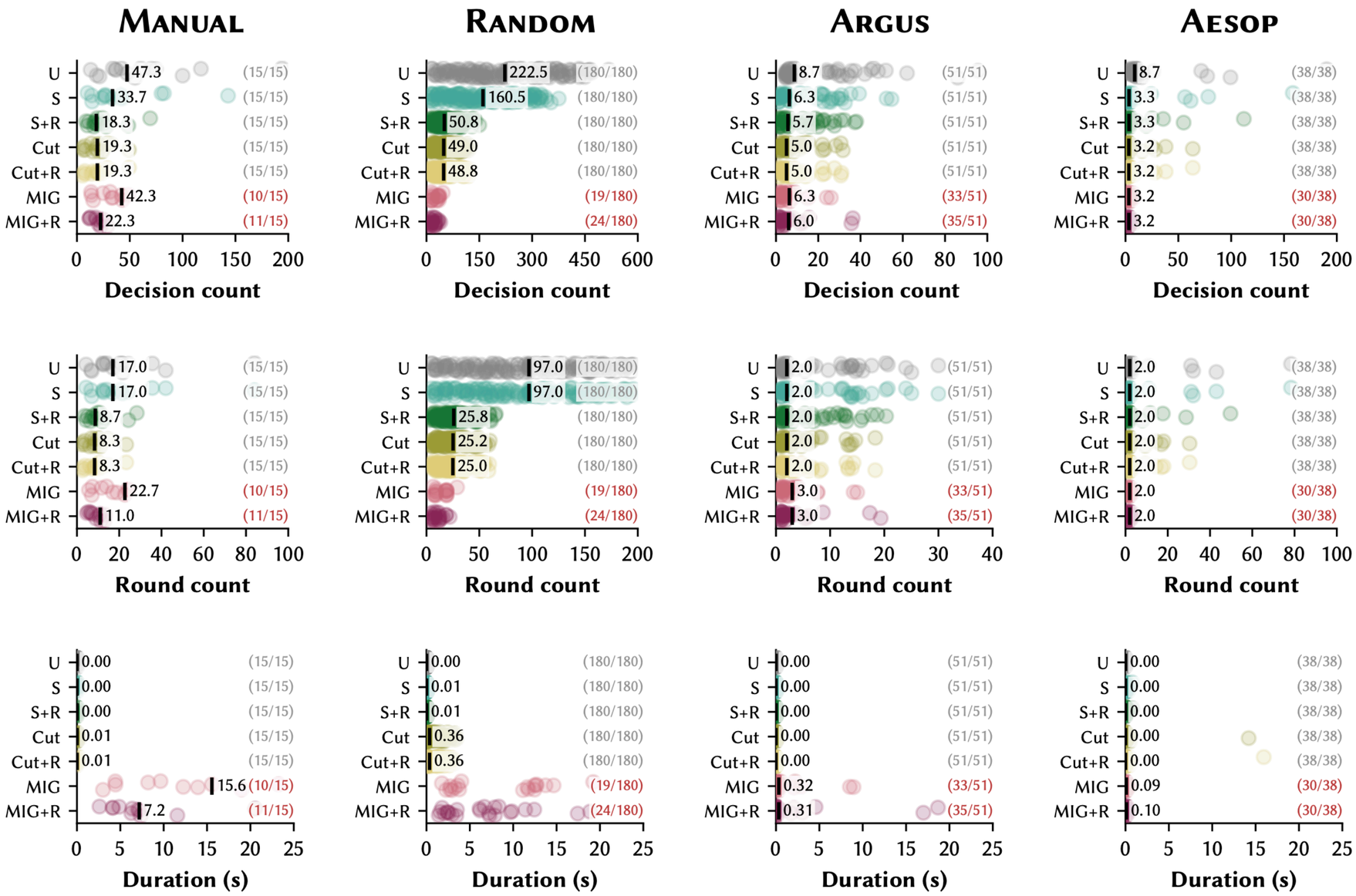
Ingredients

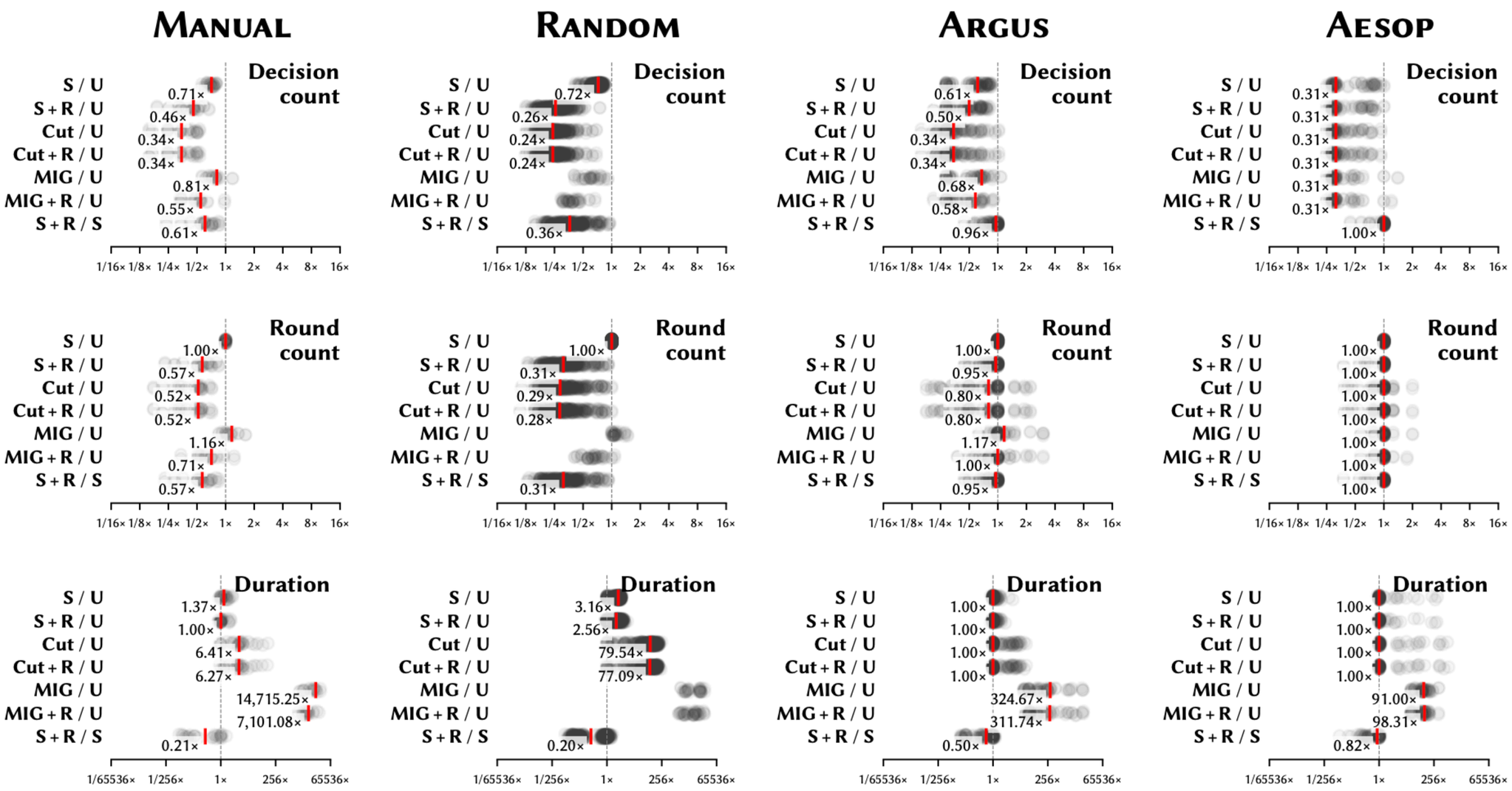
- Expressions e
- Semantics $[\![e]\!]$, maps proof systems to proof systems
  - *E.g.*, axiomatize a proposition, introduce new rules, remove rules, *etc.*
- For a proof system $\mathbf{P}$ and goal proposition G, a notion of validity (written e valid) such that e valid implies $[\![e]\!]\ \mathbf{P} \vdash G$.
  - Specifies "successful" modifications

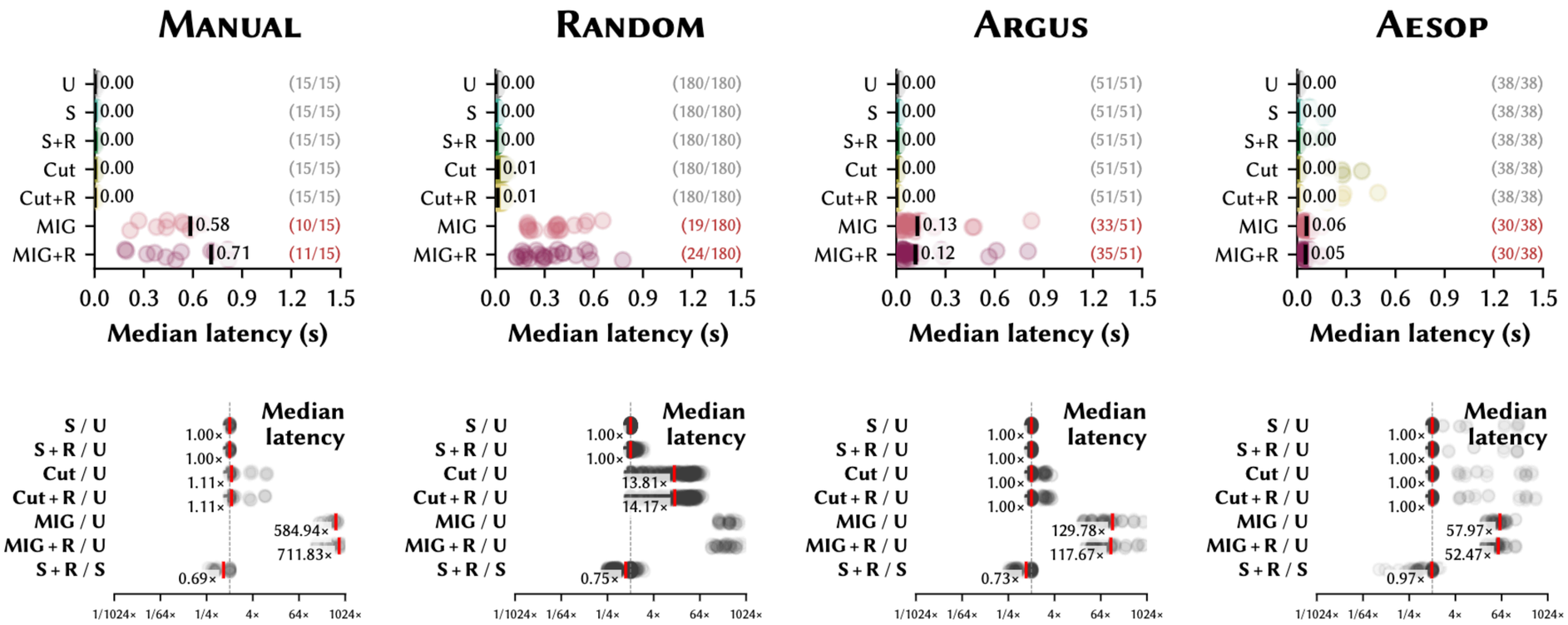
Goal restated: Synthesize e such that e valid

… but this is an **underspecification**; need to interactively **refine** specification with user


Manual
Random
Argus
Aesop
Decision count ratio
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Spearman's ρ = −0.90
0
20
40
60
80
100
120
140
160
180
200
220
Size (OR nodes)


MANUAL
RANDOM
ARGUS
AESOP
U
S
S+R
Cut
Cut+R
MIG
MIG+R
47.3 (15/15)
33.7 (15/15)
18.3 (15/15)
19.3 (15/15)
19.3 (15/15)
42.3 (10/15)
22.3 (11/15)
222.5 (180/180)
160.5 (180/180)
50.8 (180/180)
49.0 (180/180)
48.8 (180/180)
(19/180)
(24/180)
8.7 (51/51)
6.3 (51/51)
5.7 (51/51)
5.0 (51/51)
5.0 (51/51)
6.3 (33/51)
6.0 (35/51)
8.7 (38/38)
3.3 (38/38)
3.3 (38/38)
3.2 (38/38)
3.2 (38/38)
3.2 (30/38)
3.2 (30/38)
0 50 100 150 200
Decision count
0 150 300 450 600
Decision count
0 20 40 60 80 100
Decision count
0 50 100 150 200
Decision count
17.0 (15/15)
17.0 (15/15)
8.7 (15/15)
8.3 (15/15)
8.3 (15/15)
22.7 (10/15)
11.0 (11/15)
97.0 (180/180)
97.0 (180/180)
25.8 (180/180)
25.2 (180/180)
25.0 (180/180)
(19/180)
(24/180)
2.0 (51/51)
2.0 (51/51)
2.0 (51/51)
2.0 (51/51)
2.0 (51/51)
3.0 (33/51)
3.0 (35/51)
2.0 (38/38)
2.0 (38/38)
2.0 (38/38)
2.0 (38/38)
2.0 (38/38)
2.0 (30/38)
2.0 (30/38)
0 20 40 60 80 100
Round count
0 50 100 150 200
Round count
0 10 20 30 40
Round count
0 20 40 60 80 100
Round count
0.00 (15/15)
0.00 (15/15)
0.00 (15/15)
0.01 (15/15)
0.01 (15/15)
15.6 (10/15)
7.2 (11/15)
0.00 (180/180)
0.01 (180/180)
0.01 (180/180)
0.36 (180/180)
0.36 (180/180)
(19/180)
(24/180)
0.00 (51/51)
0.00 (51/51)
0.00 (51/51)
0.00 (51/51)
0.00 (51/51)
0.32 (33/51)
0.31 (35/51)
0.00 (38/38)
0.00 (38/38)
0.00 (38/38)
0.00 (38/38)
0.00 (38/38)
0.09 (30/38)
0.10 (30/38)
0 5 10 15 20 25
Duration (s)
0 5 10 15 20 25
Duration (s)
0 5 10 15 20 25
Duration (s)
0 5 10 15 20 25
Duration (s)


Manual
Random
Argus
Aesop
Decision count
Round count
Duration
S / U
S + R / U
Cut / U
Cut + R / U
MIG / U
MIG + R / U
S + R / S
0.71×
0.46×
0.34×
0.34×
0.81×
0.55×
0.61×
0.72×
0.26×
0.24×
0.24×
0.36×
0.61×
0.50×
0.34×
0.34×
0.68×
0.58×
0.96×
0.31×
0.31×
0.31×
0.31×
0.31×
0.31×
1.00×
1.00×
0.57×
0.52×
0.52×
1.16×
0.71×
0.57×
1.00×
0.31×
0.29×
0.28×
0.31×
1.00×
0.95×
0.80×
0.80×
1.17×
1.00×
0.95×
1.00×
1.00×
1.00×
1.00×
1.00×
1.00×
1.00×
1.37×
1.00×
6.41×
6.27×
14,715.25×
7,101.08×
0.21×
3.16×
2.56×
79.54×
77.09×
0.20×
1.00×
1.00×
1.00×
1.00×
324.67×
311.74×
0.50×
1.00×
1.00×
1.00×
1.00×
91.00×
98.31×
0.82×
1/16× 1/8× 1/4× 1/2× 1× 2× 4× 8× 16×
1/65536× 1/256× 1× 256× 65536×


Manual
Random
Argus
Aesop
U
S
S+R
Cut
Cut+R
MIG
MIG+R
0.00
0.58
0.71
0.01
0.13
0.12
0.06
0.05
(15/15)
(10/15)
(11/15)
(180/180)
(19/180)
(24/180)
(51/51)
(33/51)
(35/51)
(38/38)
(30/38)
0.0 0.3 0.6 0.9 1.2 1.5
Median latency (s)
Median latency
S / U
S + R / U
Cut / U
Cut + R / U
MIG / U
MIG + R / U
S + R / S
1.00×
1.11×
584.94×
711.83×
0.69×
13.81×
14.17×
0.75×
129.78×
117.67×
0.73×
57.97×
52.47×
0.97×
1/1024× 1/64× 1/4× 4× 64× 1024×

Incremental SAT Solving

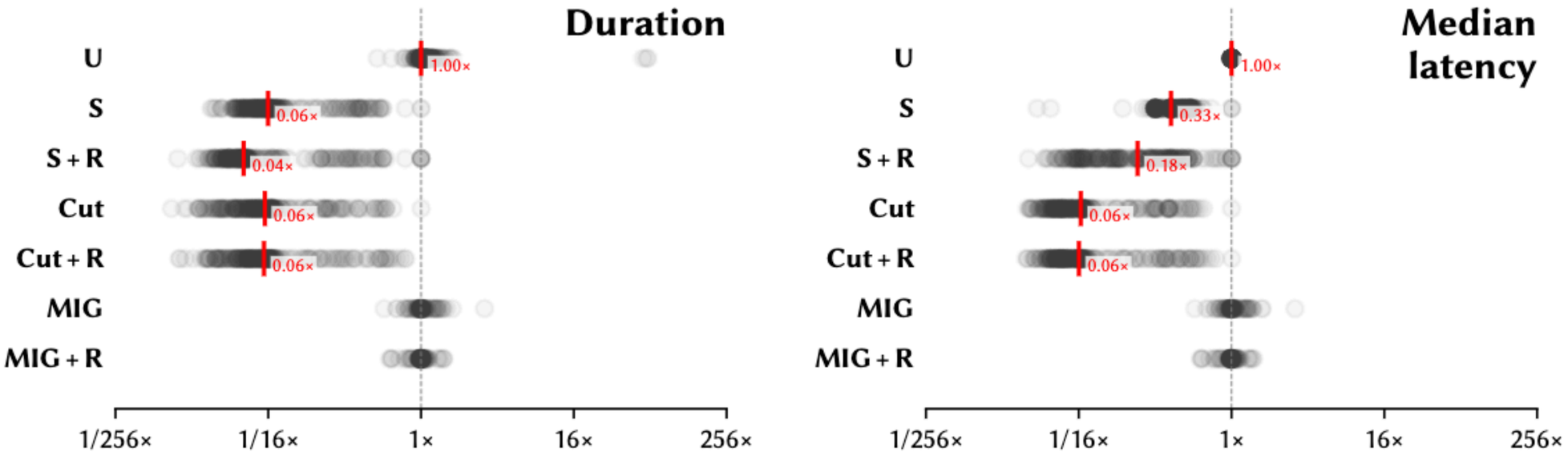


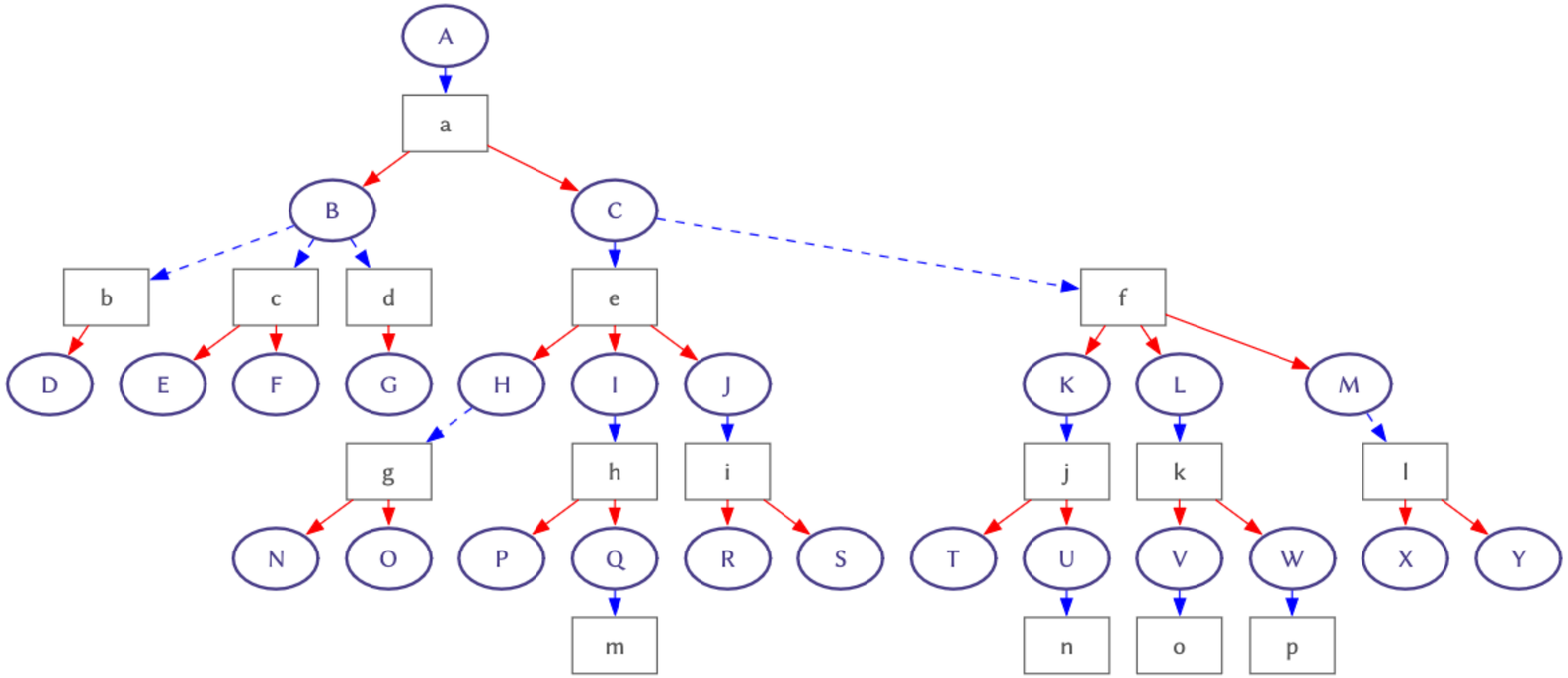
Table 1. **Benchmark summary statistics.** Each entry describes the median and IQR. **Depth** is the maximum graph distance from the goal to any other node in the same connected component. **ORs** and **ANDs** are the OR and AND node count. **Consumers** is the median number of parents for OR nodes, **Providers** is the median number of children for OR nodes, and **Premises** is the median number of children for AND nodes.

Suite	Depth	ORs	ANDs	Consumers	Providers	Premises
Manual (15 entries)	8 (6–12)	24 (16–40)	12 (9–32)	1 (1–1)	0 (0–1)	2 (1–2)
Random (180 entries)	8 (6–12)	105 (60–150)	30 (16–48)	1 (1–1)	0 (0–0)	3 (2–4)
Argus (51 entries)	4 (2–4)	5 (3–23)	3 (2–18)	1 (0.5–1)	0 (0–0.5)	1 (1–4)
Aesop (38 entries)	2 (2–4)	3 (3–4)	1 (1–5)	1 (1–1)	0.5 (0–1)	1 (1–2)

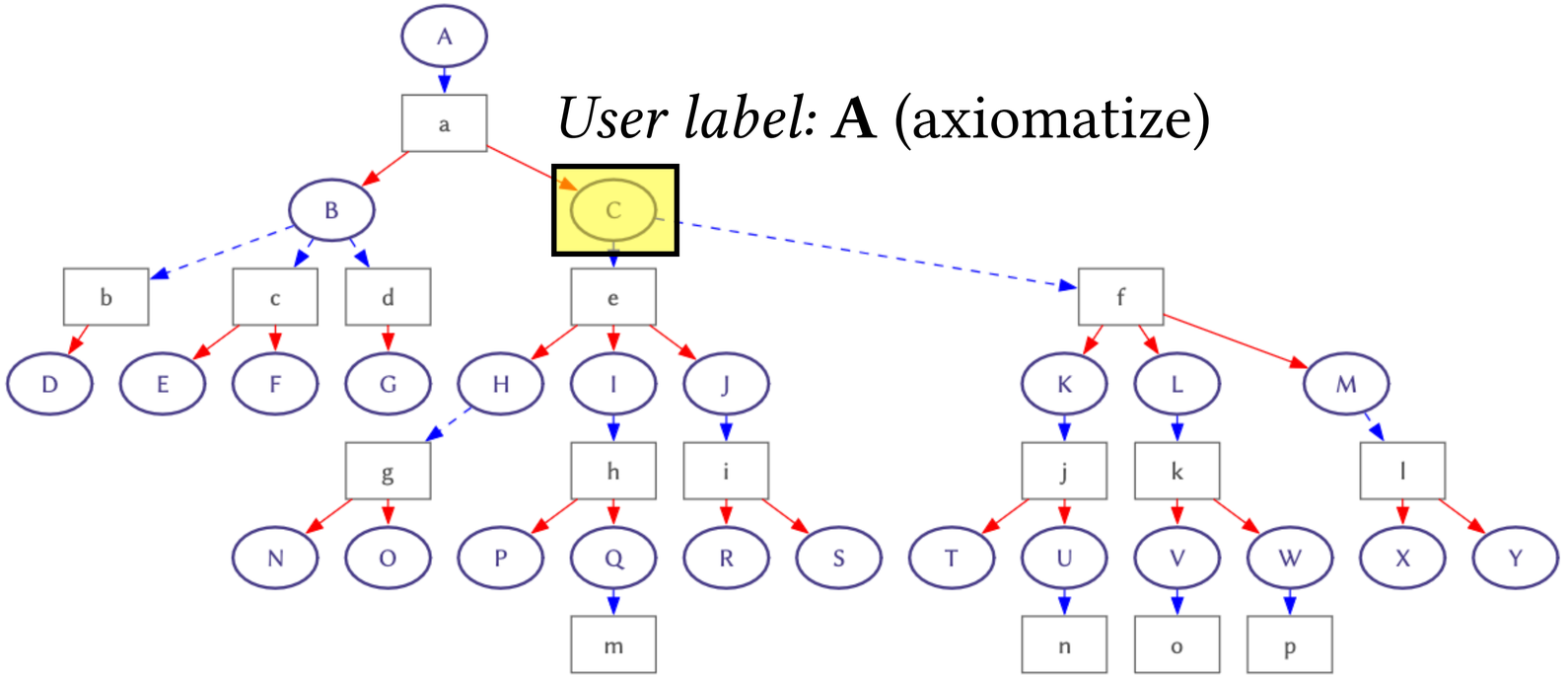
Table 2. **Maximum latencies.** Each entry is the maximum latency of the step provider on the given suite.

	Manual	Random	Argus	Aesop
U	<1 ms	<1 ms	<1 ms	<1 ms
S	<1 ms	1 ms	4 ms	224 ms
S+R	<1 ms	6 ms	<1 ms	201 ms
Cut	16 ms	92 ms	4 ms	907 ms
Cut+R	15 ms	83 ms	4 ms	**1084 ms**

One last technique: We don't need to reach <u>every</u> partition

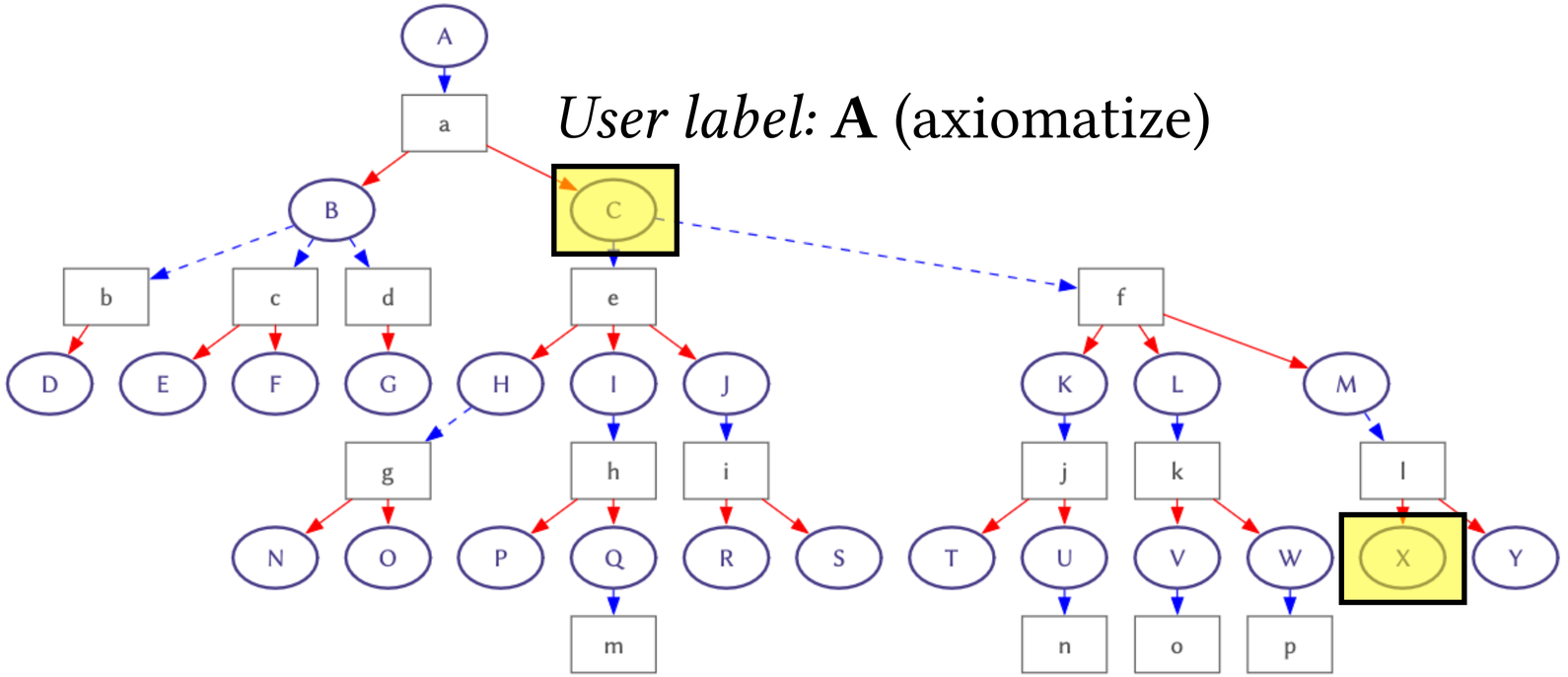


One last technique: We don't need to reach <u>every</u> partition



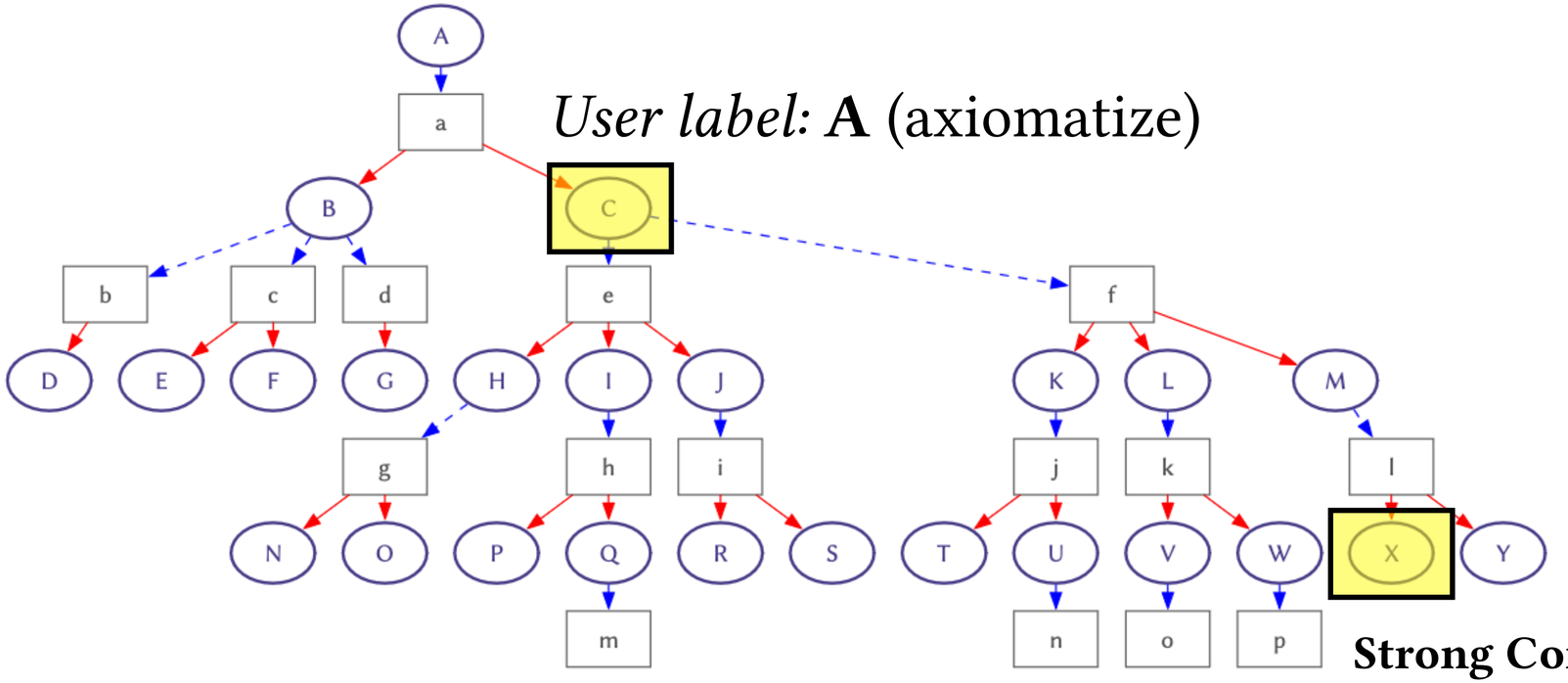
User label: **A** (axiomatize)

One last technique: We don't need to reach <u>every</u> partition



User label: **A** (axiomatize)

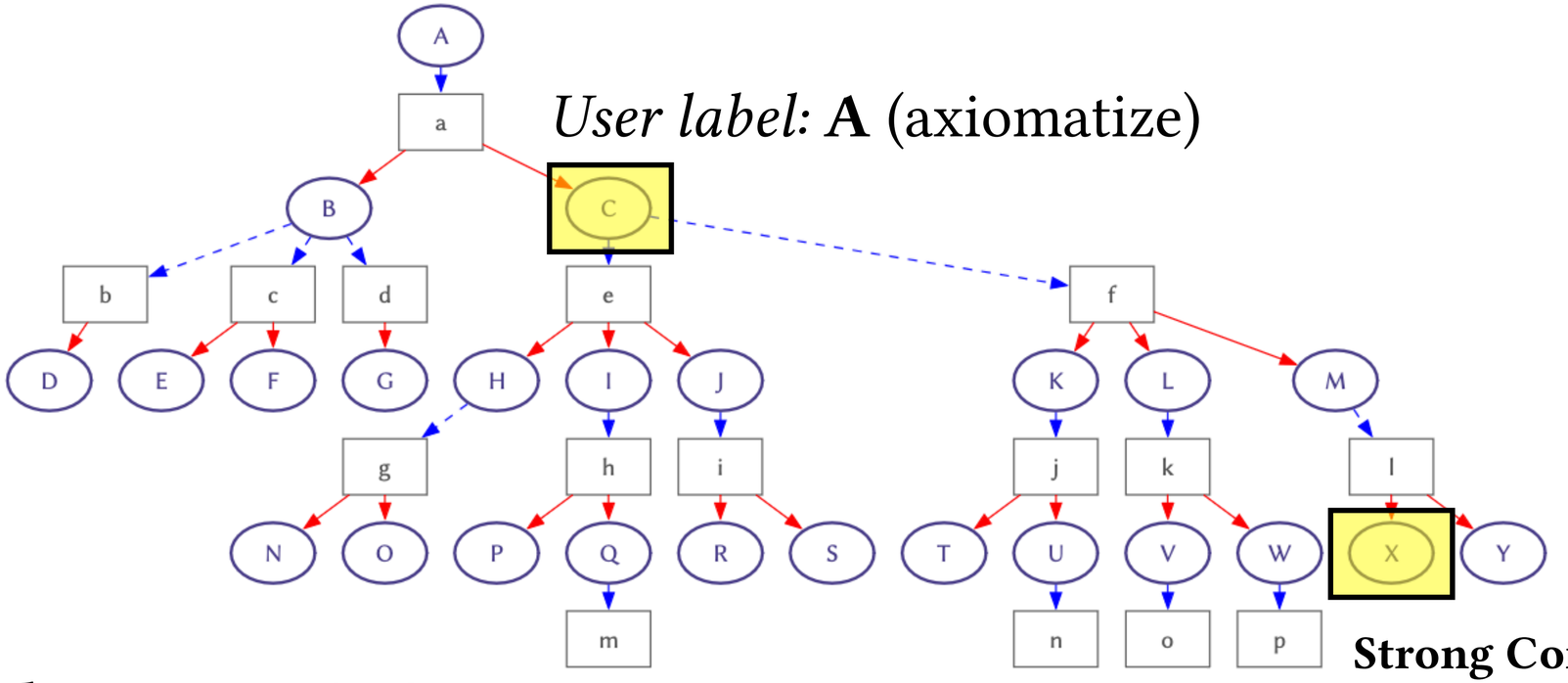
One last technique: We don't need to reach every partition



User label: **A** (axiomatize)

Strong Completeness requires us to still be able to label this as, *e.g.*, **A**

One last technique: We don't need to reach <u>every</u> partition



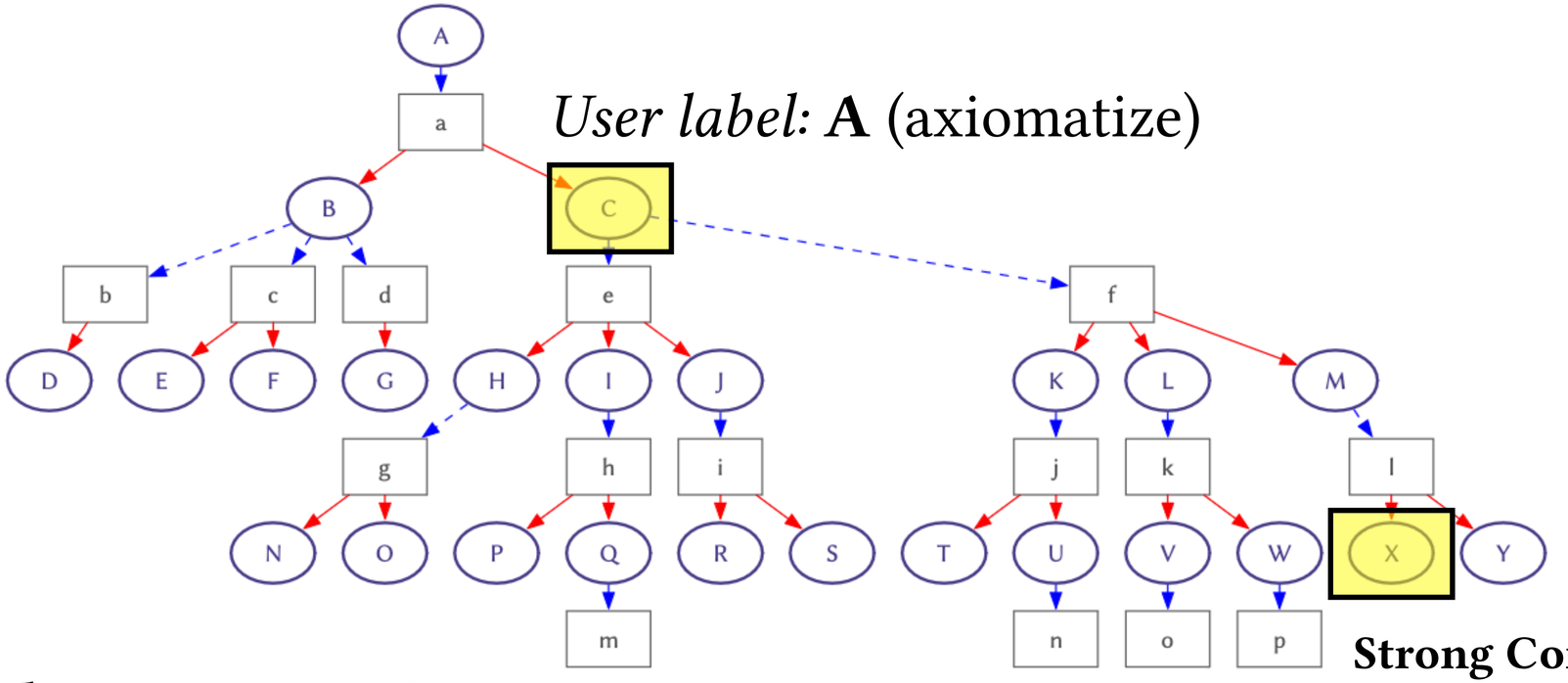
User label: **A** (axiomatize)

Strong Completeness Modulo Observability

$$\bigcup_{\sigma \in \Sigma} [C(\sigma e)]_{\approx} \supseteq [C(e) \setminus \{e\}]_{\approx}$$

Strong Completeness requires us to still be able to label this as, *e.g.*, **A**

One last technique: We don't need to reach every partition



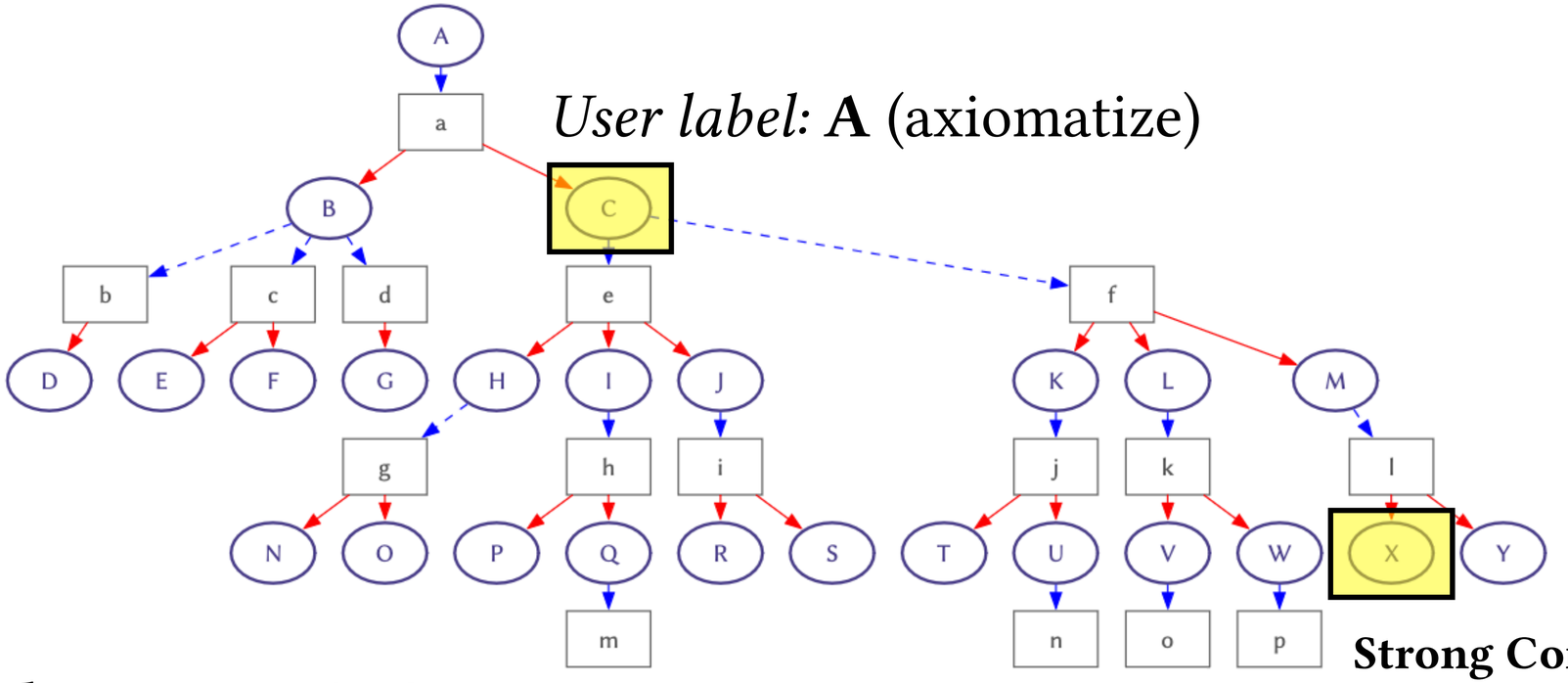
User label: **A** (axiomatize)

Strong Completeness Modulo Observability

$$\bigcup_{\sigma \in \Sigma} [C(\sigma e)]_{\approx} \supseteq [C(e) \setminus \{e\}]_{\approx}$$

Strong Completeness requires us to still be able to label this as, *e.g.*, **A**

One last technique: We don't need to reach <u>every</u> partition



User label: **A** (axiomatize)

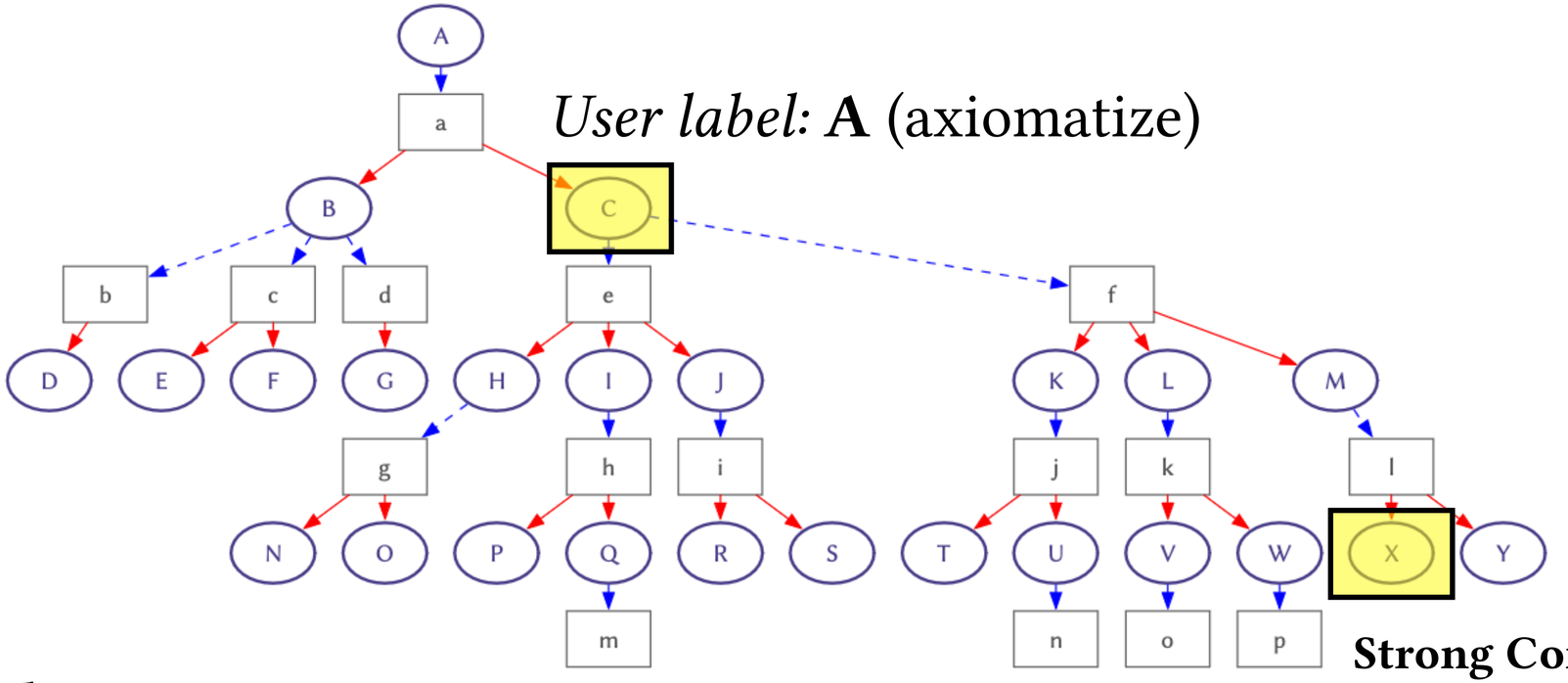
Strong Completeness Modulo Observability

$$\bigcup_{\sigma \in \Sigma} [C(\sigma e)]_{\approx} \supseteq [C(e) \setminus \{e\}]_{\approx}$$

Strong Completeness requires us to still be able to label this as, *e.g.*, **A**

Standard observational equivalence works ($e_1 \approx e_2$ iff $[\![e_1]\!] = [\![e_2]\!]$), but we can go coarser:

One last technique: We don't need to reach every partition



User label: **A** (axiomatize)

Strong Completeness Modulo Observability

$$\bigcup_{\sigma \in \Sigma} [C(\sigma e)]_{\approx} \supseteq [C(e) \setminus \{e\}]_{\approx}$$

Strong Completeness requires us to still be able to label this as, *e.g.*, **A**

Standard observational equivalence works ($e_1 \approx e_2$ iff $[\![e_1]\!] = [\![e_2]\!]$), but we can go coarser:

$$e_1 \approx e_2 \quad \text{iff} \quad \mathbf{Assumes}(e_1 ; \mathcal{P} \vdash G) = \mathbf{Assumes}(e_2 ; \mathcal{P} \vdash G),$$

where

$$\mathbf{Assumes}(e ; \mathcal{P} \vdash G) = \{\{Q \in e^{-1}[L_{\text{assume}}] : Q \text{ appears in } P\} : P \text{ a proof of } G \text{ in } [\![e]\!]\,\mathcal{P} \text{ respecting } e\}$$

Special type of "why-provenance"