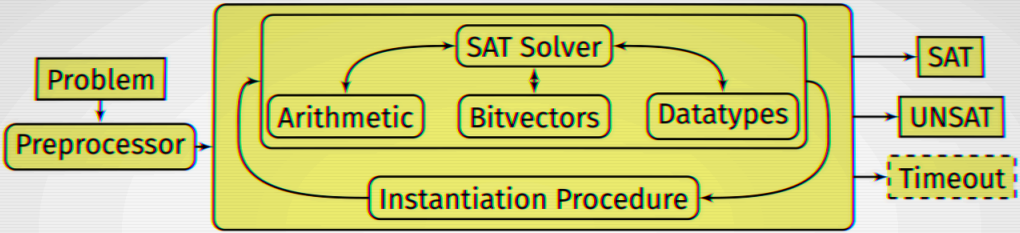
A Fast Proof Checker for the Eunoia Logical Framework

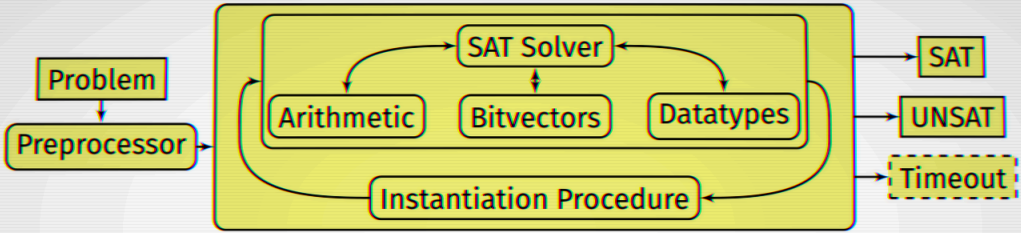
Andrew Reynolds, **Hans-Jörg Schurr**, Mallku Soldevila,
Haniel Barbosa, Cesare Tinelli, Clark Barrett

IJCAR — Lisbon, Portugal
July 29, 2026

Proof Certificates For This?



Proof Certificates For This?



Two Ideas

LFSC A framework to model proof rules.
Problem: Hard to use.

Alethe A proof format based on the familiar SMT-LIB language.
Problem: English language specification.

Eunoia combines the good ideas of both!

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

theory.eo

```
(declare-const Int Type)
(declare-consts <numeral> Int)

(declare-const not (-> Bool Bool))
(declare-const and (-> Bool Bool Bool) :right-assoc-nil true)

(declare-parameterized-const = ((A Type :implicit)) (-> A A Bool))

(declare-const BitVec (-> Int Type))
(declare-parameterized-const concat
  ((m Int :implicit) (n Int :implicit))
  (-> (BitVec n) (BitVec m) (BitVec (eo::add m n)))
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

rules.eo

```
(include "theory.eo")

(declare-rule contra ((F Bool))
  :premises (F (not F))
  :conclusion false
)

(program pick ((F Bool) (Fs Bool :list) (n Int))
  :signature (Bool Int) Bool
  (((pick  F    0)    F)
   ((pick (and F Fs) n) (pick Fs (eo::add n -1))))
)
(declare-rule andE ((Fs Bool) (i Int))
  :premises (Fs)
  :args (i)
  :conclusion (pick Fs i)
)
```

Introducing Eunoia!

```
(reference "problem.smt2")

(include "rules.eo")

(declare-const F Bool)
(assume a1 (and F (not F)))

(step s1 (F)     :rule andE   :premises (a1) :args (0))
(step s2 (not F) :rule andE   :premises (a1) :args (1))
(step s3 (false) :rule contra :premises (s1 s2))
```

Introducing Eunoia!

```
(reference "problem.smt2")

(include "rules.eo")

(declare-const F Bool)
(assume a1 (and F (not F)))

(step s1 (F)     :rule andE   :premises (a1) :args (0))
(step s2 (not F) :rule andE   :premises (a1) :args (1))
(step s3 (false) :rule contra :premises (s1 s2))
```

Introducing Eunoia!

```
(reference "problem.smt2")

(include "rules.eo")

(declare-const F Bool)
(assume a1 (and F (not F)))

(step s1 (F)     :rule andE   :premises (a1) :args (0))
(step s2 (not F) :rule andE   :premises (a1) :args (1))
(step s3 (false) :rule contra :premises (s1 s2))
```

Introducing Eunoia!

```
(reference "problem.smt2")

(include "rules.eo")

(declare-const F Bool)
(assume a1 (and F (not F)))

(step s1 (F)     :rule andE   :premises (a1) :args (0))
(step s2 (not F) :rule andE   :premises (a1) :args (1))
(step s3 (false) :rule contra :premises (s1 s2))
```

Eunoia Proofs for cvc5!
You saw it at CAV 2026 on Monday!
Paper: “The Cooperating Proof Calculus: Comprehensive Proofs for an SMT Solver”

Eunoia Proofs for cvc5!
You saw it at CAV 2026 on Monday!
Paper: “The Cooperating Proof Calculus: Comprehensive Proofs for an SMT Solver”

Eunoia Proofs in `lambdapi` !
You were there! Just over 30 minutes ago!

Eunoia Proofs for cvc5!
You saw it at CAV 2026 on Monday!
Paper: "The Cooperating Proof Calculus: Comprehensive Proofs for an SMT Solver"

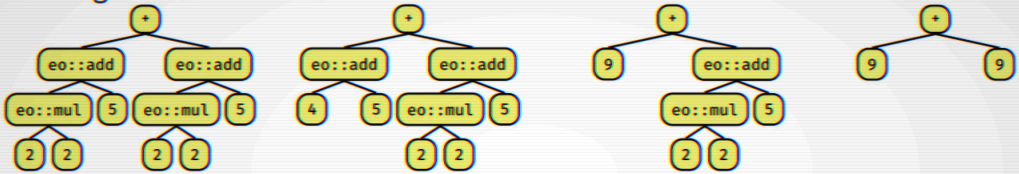
Eunoia Proofs in `lambdapi` !
You were there! Just over 30 minutes ago!

Now! Checking Eunoia Proofs Fast!

- **Ethos** is your bread and butter proof checker for Eunoia.
- Want to get your proofs checked fast? **Ethos!**
- Want to debug your Eunoia signatures? **Ethos!**
- Want to skip writing an Eunoia parser? **Ethos!**
- TCP from > 400k lines of cvc5 C++ to 12k + 10k lines of Eunoia and Ethos C++

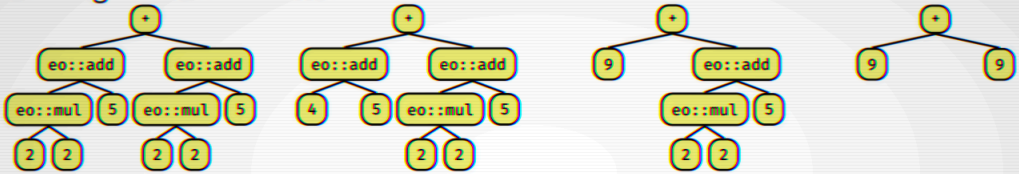
Optimizations!

Caching of Evaluation Results

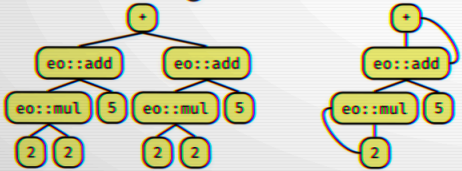


Optimizations!

Caching of Evaluation Results

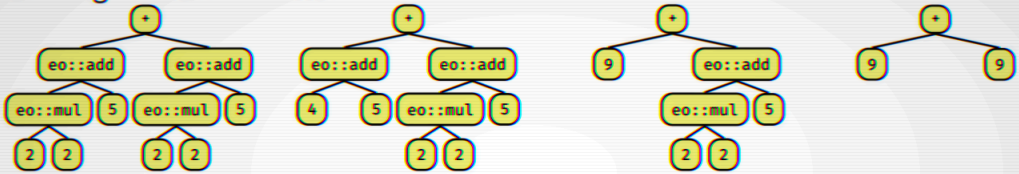


Perfect Sharing

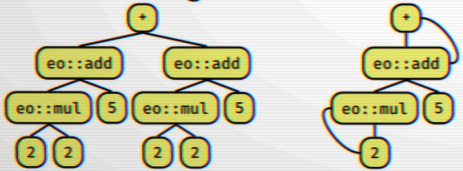


Optimizations!

Caching of Evaluation Results



Perfect Sharing



Keep Tails

`eo::list_erase_all … 4`

How Fast Is Ethos?

Evaluation

- Problem: do not measure solver behavior!

How Fast Is Ethos?

Evaluation

- Problem: do not measure solver behavior!
- We evaluate on proofs generated by **cvc5**.
- Runtime cost: internal elaboration, calculus difference, checker speed.

How Fast Is Ethos?

Evaluation

- Problem: do not measure solver behavior!
- We evaluate on proofs generated by **cvc5**.
- Runtime cost: internal elaboration, calculus difference, checker speed.
- Formats: Alethe (with **Carcara**), **LFSC**, and CPC (with **Ethos**).

How Fast Is Ethos?

	Verified	Generation	Checking	Failures
CPC	20 813	85 868s	93 157s	571
Alethe	19 502	90 908s	13 869s	
Holes	867	7 172s	1 192s	1 211
LFSC	454	36s	42s	
Holes	19 275	81 459s	223 989s	127

UFLIRA
20 940 benchmarks, not known **sat**, solvable by cvc5 in 60s.
600s timeout.

How Fast Is Ethos?

	Verified	Generation	Checking	Failures
CPC	20 813	85 868s	93 157s	571
Alethe	19 502	90 908s	13 869s	
Holes	867	7 172s	1 192s	1 211
LFSC	454	36s	42s	
Holes	19 275	81 459s	223 989s	127

UFLIRA
20 940 benchmarks, not known **sat**, solvable by cvc5 in 60s.
600s timeout.

How Fast Is Ethos?

	Verified	Generation	Checking	Failures
CPC	20 813	85 868s	93 157s	571
Alethe	19 502	90 908s	13 869s	
Holes	867	7 172s	1 192s	1 211
LFSC	454	36s	42s	
Holes	19 275	81 459s	223 989s	127

UFLIRA
20 940 benchmarks, not known **sat**, solvable by cvc5 in 60s.
600s timeout.

How Fast Is Ethos?

	Verified	Generation	Checking	Failures
CPC	20 813	85 868s	93 157s	571
Alethe	19 502	90 908s	13 869s	
Holes	867	7 172s	1 192s	1 211
LFSC	454	36s	42s	
Holes	19 275	81 459s	223 989s	127

UFLIRA
20 940 benchmarks, not known **sat**, solvable by cvc5 in 60s.
600s timeout.

How Fast Is Ethos?

	Verified	Generation	Checking	Failures
CPC	138 912	657 911s	219 267s	1 337
LFSC	705	36s	34s	
Holes	136 482	229 300s	221 911s	3 061

The LFSC numbers are distorted due to the large number of holes!

140 248 benchmarks, not known **sat**, solvable by cvc5 in 60s.
600s timeout.

Thank You!