Trimming Pseudo-Boolean Proofs

Arthur GONTIER et al

July 19, 2026

Authors

- Berhan Oumer Adame[1]
- Bart Bogaerts[2]
- Benjamin Bogø[3]
- Simon Dold[4]
- **Arthur Gontier**[5]
- Wietze Koops[6]
- Ciaran McCreesh[5]
- Matthew J. McIlree[5]
- Jakob Nordström[3]
- Andy Oertel[6]
- Adrian Rebola-Pardo[7]
- Mark Turnbull[5]

[1]Vrije Universiteit Brussel, Belgium; Arba Minch University, Ethiopia [2]KU Leuven, Belgium; Vrije Universiteit Brussel, Belgium
[3]University of Copenhagen, Denmark; Lund University, Sweden [4]University of Basel, Switzerland
[5]University of Glasgow, United Kingdom [6]Lund University, Sweden; University of Copenhagen, Denmark

[7]Vienna University of Technology, Austria; Johannes Kepler University Linz, Austria

Trimming Pseudo-Boolean Proofs — paper under submission.

Table of Contents

Proof logs

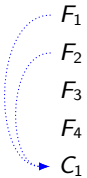
F_1

F_2

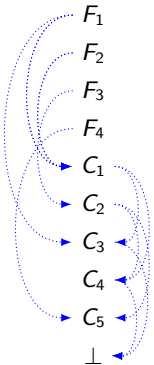
F_3

F_4

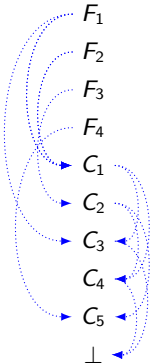
Proof logs

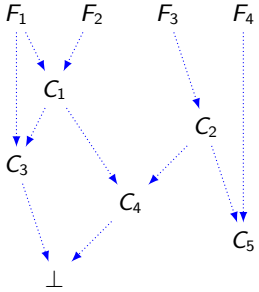


Proof logs

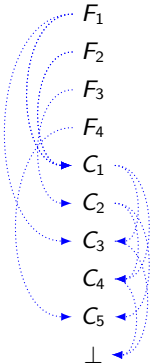


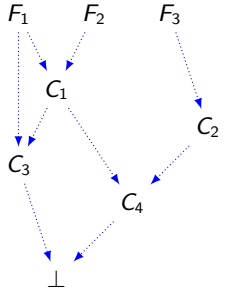
Proof logs





Proof logs





Looking for antecedents

pol F_1 3 * C_1 + s

Looking for antecedents

pol F_1 3 * C_1 + s

Antecedents: F_1, C_1

Looking for antecedents

pol F_1 3 * C_1 + s rup $ax + by + cz \geq r$

Antecedents: F_1, C_1

Looking for antecedents

pol F_1 3 * C_1 + s

Antecedents: F_1, C_1

rup $ax + by + cz \geq r$

Antecedents: All ctrs used in rup check

Looking for antecedents

pol F_1 3 * C_1 + s

Antecedents: F_1, C_1

rup $ax + by + cz \geq r$

Antecedents: All ctrs used in rup check

red $ax + by \geq r$: x 0 : subproof
  proofgoal C_1
    pol C_2 F_3 + s
    rup $0 \geq 1$
  qed : -1
  proofgoal C_2
    pol C_1 F_2 + s
    rup $0 \geq 1$
  qed : -1
qed red

Looking for antecedents

pol F_1 3 * C_1 + s

Antecedents: F_1, C_1

rup $ax + by + cz \geq r$

Antecedents: All ctrs used in rup check

red $ax + by \geq r$: x 0 : subproof
  proofgoal C_1
    pol C_2 F_3 + s
    rup $0 \geq 1$
  qed : -1
  proofgoal C_2
    pol C_1 F_2 + s
    rup $0 \geq 1$
  qed : -1
qed red

Antecedents: If proofgoal is marked then subproof antecedants are.

Cone-first Rup

Algorithm 1: rup

```
1 while no contradiction do
2     propagate()
```

Cone-first Rup

Algorithm 2: rup

```
1 while no contradiction do
2     propagate()
```

Algorithm 3: Cone-first rup

```
1 only_marked ← true
2 while no contradiction do
3     if only_marked then
4         only_marked ← try_propagate_marked()
5     else
6         propagate_one()
7         only_marked ← true
```

Why is PB trimming harder than SAT trimming?

- Constraints are not always listed explicitly:
  cutting planes (`pol`) derivations specify *how* a constraint is derived, not *what* it looks like.

Why is PB trimming harder than SAT trimming?

- Constraints are not always listed explicitly:
  cutting planes (`pol`) derivations specify *how* a constraint is derived, not *what* it looks like.
- Proof line deletion can be restricted by the proof format.

Why is PB trimming harder than SAT trimming?

- Constraints are not always listed explicitly:
  cutting planes (`pol`) derivations specify *how* a constraint is derived, not *what* it looks like.
- Proof line deletion can be restricted by the proof format.
- The redundance-based strengthening rule (`red`) and dominance rule induces *global* proof dependencies that can propagate during the backward pass.

Why is PB trimming harder than SAT trimming?

- Constraints are not always listed explicitly:
  cutting planes (`pol`) derivations specify *how* a constraint is derived, not *what* it looks like.
- Proof line deletion can be restricted by the proof format.
- The redundance-based strengthening rule (`red`) and dominance rule induces *global* proof dependencies that can propagate during the backward pass.

Our answer: conditional marking. Rather than deciding immediately which constraints to keep, we collect deferred conditions of the form

"if C is ultimately needed, then D must be kept too"

and resolve them lazily at the end of the backward pass.

Table of Contents

Trimmer for faster proof checks

1) Decoration :

- Forward pass on the proof
- Collect what's needed to replay it backwards (ids, deletions, order/objective state)
- No propagation checks $\Rightarrow$ cheap
- Pol lines kept lazily, not evaluated yet

Trimmer for faster proof checks

1) Decoration :
   - Forward pass on the proof
   - Collect what's needed to replay it backwards (ids, deletions, order/objective state)
   - No propagation checks ⇒ cheap
   - Pol lines kept lazily, not evaluated yet

2) Backward trimming :
   - Backward pass from first contradiction, marking constraints
   - Rup: hints if given, else **cone-first rup** + conflict analysis
   - Pol: evaluate the needed lazy pol on the fly
   - Deletions: rescheduled to last use
   - Red/dom: **conditional marking** of proof obligations

Trimmer for faster proof checks

1) Decoration :
   - Forward pass on the proof
   - Collect what's needed to replay it backwards (ids, deletions, order/objective state)
   - No propagation checks $\Rightarrow$ cheap
   - Pol lines kept lazily, not evaluated yet

2) Backward trimming :
   - Backward pass from first contradiction, marking constraints
   - Rup: hints if given, else **cone-first rup** + conflict analysis
   - Pol: evaluate the needed lazy pol on the fly
   - Deletions: rescheduled to last use
   - Red/dom: **conditional marking** of proof obligations

3) Compaction :
   - Forward pass writing the trimmed proof
   - remap ids
   - Resolve conditional markings

Trimmer for faster proof checks

1) Decoration :
   - Forward pass on the proof
   - Collect what's needed to replay it backwards (ids, deletions, order/objective state)
   - No propagation checks $\Rightarrow$ cheap
   - Pol lines kept lazily, not evaluated yet

2) Backward trimming :
   - Backward pass from first contradiction, marking constraints
   - Rup: hints if given, else **cone-first rup** + conflict analysis
   - Pol: evaluate the needed lazy pol on the fly
   - Deletions: rescheduled to last use
   - Red/dom: **conditional marking** of proof obligations

3) Compaction :
   - Forward pass writing the trimmed proof
   - remap ids
   - Resolve conditional markings

PS: *Agnostic Trimming* mode skips decoration entirely for proofs with only pol/hinted-rup steps

Experimental setup

- Six solvers across five paradigms, on their own published benchmark sets:
  - Glasgow Subgraph Solver (subgraph isomorphism, LV instances) + clique instances
  - Glasgow Constraint Solver
  - Pacose (MaxSAT, MSE 2023)
  - RoundingSat (pseudo-Boolean optimisation, PB comp. 2024)
  - kissat + Satsuma (SAT with symmetry breaking, SAT comp. 2024)

Experimental setup

- Six solvers across five paradigms, on their own published benchmark sets:
  - Glasgow Subgraph Solver (subgraph isomorphism, LV instances) + clique instances
  - Glasgow Constraint Solver
  - Pacose (MaxSAT, MSE 2023)
  - RoundingSat (pseudo-Boolean optimisation, PB comp. 2024)
  - kissat + Satsuma (SAT with symmetry breaking, SAT comp. 2024)
- **2656** instance-proof pairs, **2.7 TB** of proof files

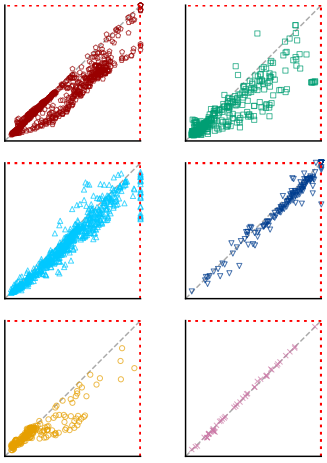
Experimental setup

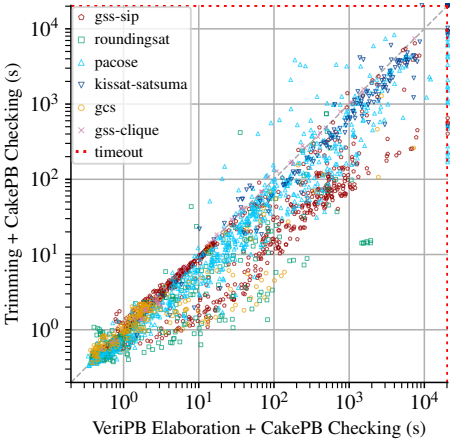
- Six solvers across five paradigms, on their own published benchmark sets:
  - Glasgow Subgraph Solver (subgraph isomorphism, LV instances) + clique instances
  - Glasgow Constraint Solver
  - Pacose (MaxSAT, MSE 2023)
  - RoundingSat (pseudo-Boolean optimisation, PB comp. 2024)
  - kissat + Satsuma (SAT with symmetry breaking, SAT comp. 2024)
- **2656** instance-proof pairs, **2.7 TB** of proof files
- Compared: elaborate + CakePB check vs trim + CakePB check

Experimental setup

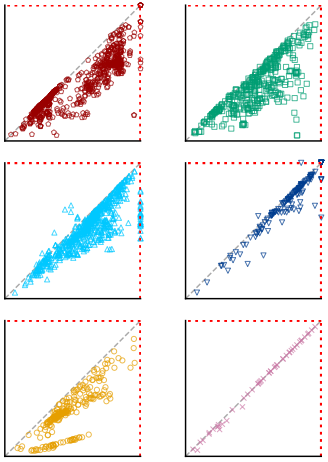
- Six solvers across five paradigms, on their own published benchmark sets:
  - Glasgow Subgraph Solver (subgraph isomorphism, LV instances) + clique instances
  - Glasgow Constraint Solver
  - Pacose (MaxSAT, MSE 2023)
  - RoundingSat (pseudo-Boolean optimisation, PB comp. 2024)
  - kissat + Satsuma (SAT with symmetry breaking, SAT comp. 2024)
- **2656** instance-proof pairs, **2.7 TB** of proof files
- Compared: elaborate + CakePB check vs trim + CakePB check
- Excluded from ratios (kept in the plots): 54 elab. timeouts, 182 SAT/NONE conclusions

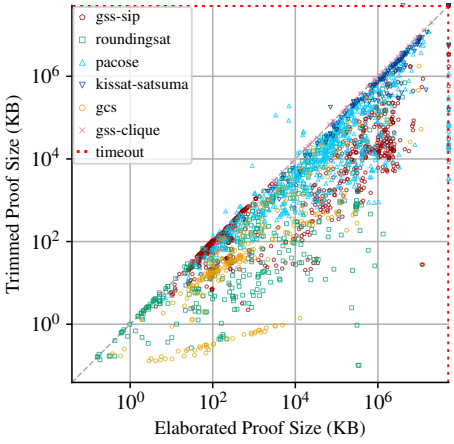
Timing: elaboration+CakePB vs. trimming+CakePB





Proof size: elaborated vs. trimmed





Results

Solver	elab./trim time	CakePB time	total time	size
gss-sip	1.62	3.36	2.29	6.85
roundingsat	1.39	2.70	2.02	11.12
pacose	0.95	2.73	1.59	4.22
kissat-satsuma	1.32	1.70	1.45	1.74
gcs	1.54	2.07	1.68	17.13
gss-clique	1.00	1.01	1.01	1.11
all	**1.27**	**2.57**	**1.81**	**6.37**

Table: 1-shifted geometric means of ratios (elaborated / trimmed).

6.37× smaller proofs, 1.81× faster total checking.

Table of Contents

Trimnalyser: smaller proof for analysis

Use cases:

- Using proof as certificate
- Try understanding your problem

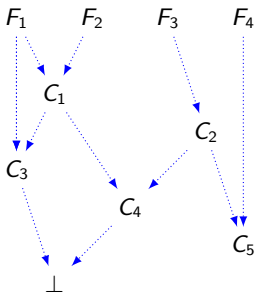
Trimnalyser: smaller proof for analysis

Use cases:

- Using proof as certificate
- Try understanding your problem

Philosophy:

- Take more time and resources to trim more
- Use a subset of proof rules (rup, pol).
- Has optimisations dedicated to proof reduction (can be x10 slower)



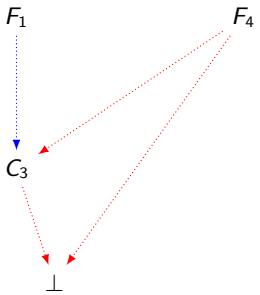
Trimnalyser: smaller proof for analysis

Use cases:

- Using proof as certificate
- Try understanding your problem

Philosophy:

- Take more time and resources to trim more
- Use a subset of proof rules (rup, pol).
- Has optimisations dedicated to proof reduction (can be x10 slower)



Do not trust deletions

- A derived constraint may be useful later.

$C_{1.1}$
$C_{1.2}$
C_1
del id $C_{1.1}$ $C_{1.2}$
$C_{2.1}$
$C_{2.2}$
C_2
del id $C_{2.1}$ $C_{2.2}$
$C_{3.1}$
$C_{3.2}$
C_3
del id $C_{3.1}$ $C_{3.2}$
C_4
$\bot$

Do not trust deletions

- A derived constraint may be useful later.

$C_{1.1}$
$C_{1.2}$
C_1
del id $C_{1.1}$ $C_{1.2}$
$C_{2.1}$
$C_{2.2}$
C_2
del id $C_{2.1}$ $C_{2.2}$
$C_{3.1}$
$C_{3.2}$
C_3
del id $C_{3.1}$ $C_{3.2}$
C_4
$\bot$

$C_{1.1}$
$C_{1.2}$
C_1
C_2
C_3
del id $C_{1.1}$ $C_{1.2}$
C_4
$\bot$

Do not trust deletions

- A derived constraint may be useful later.
- Be careful with redundance; they may need prior deletions to work.

```
red ¬y ≥ 1 :  y 0
del id -1
red y ≥ 1 :  y 1
```

$C_{1.1}$
$C_{1.2}$
C_1
del id $C_{1.1}$ $C_{1.2}$
$C_{2.1}$
$C_{2.2}$
C_2
del id $C_{2.1}$ $C_{2.2}$
$C_{3.1}$
$C_{3.2}$
C_3
del id $C_{3.1}$ $C_{3.2}$
C_4
$\bot$

$C_{1.1}$
$C_{1.2}$
C_1
C_2
C_3
del id $C_{1.1}$ $C_{1.2}$
C_4
$\bot$

Do not trust deletions

- A derived constraint may be useful later.
- Be careful with redundancy; they may need prior deletions to work.

```
red ¬y ≥ 1 :  y 0
del id -1
red y ≥ 1 :  y 1
```

- Fine if you don't reuse the names

$C_{1.1}$
$C_{1.2}$
C_1
del id $C_{1.1}$ $C_{1.2}$
$C_{2.1}$
$C_{2.2}$
C_2
del id $C_{2.1}$ $C_{2.2}$
$C_{3.1}$
$C_{3.2}$
C_3
del id $C_{3.1}$ $C_{3.2}$
C_4
$\bot$

$C_{1.1}$
$C_{1.2}$
C_1
C_2
C_3
del id $C_{1.1}$ $C_{1.2}$
C_4
$\bot$

Better rup for trimming ?

- Cone-first

Better rup for trimming ?

- Cone-first
- Heuristic prioritizing the highest ctrs

Better rup for trimming ?

- Cone-first
- Heuristic prioritizing the highest ctrs
- Conflict analysis to get smaller antecedent set

Better rup for trimming ?

- Cone-first
- Heuristic prioritizing the highest ctrs
- Conflict analysis to get smaller antecedent set

Algorithm 4: rup

```
1 only_m ← true
2 while no contradiction do
3     if only_m then
4         only_m ← try_prop_m_ordered()
5     else
6         prop_one_ordered()
7         only_m ← true
8     update_implG(lit,lvl,ctrID)
9 MarkAntecedantsFromConflictAnalysis()
```

Better rup for trimming ?

- Cone-first
- Heuristic prioritizing the highest ctrs
- Conflict analysis to get smaller antecedent set

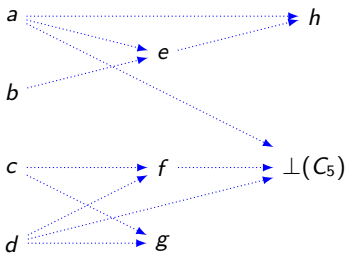
Algorithm 5: rup

```
1 only_m ← true
2 while no contradiction do
3     if only_m then
4         only_m ← try_prop_m_ordered()
5     else
6         prop_one_ordered()
7         only_m ← true
8     update_implG(lit,lvl,ctrID)
9 MarkAntecedantsFromConflictAnalysis()
```

Algorithm 6: MAFCA

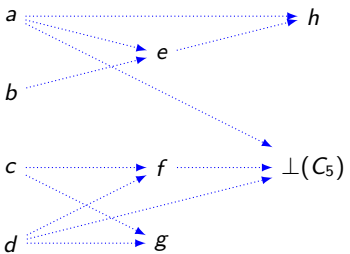
```
1  enqueue(PQ,⊥,1)
2  while not_empty(PQ) do
3      lit ← dequeue(PQ)
4      id ← implG[lit]
5      mark(id)
6      lits ← contributing_lits(id)
7      sort(lits,by:order)
8      for l ∈ lits do
9          if propagation then
10             break
11         else
12             set_lit(l,false)
13             enqueue(PQ,l,lvl)
```

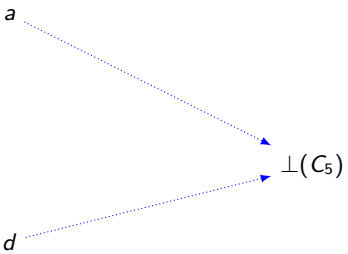
Conflict analysis example



$$\neg a + \neg d + \neg f \geq 2 \qquad (C_5)$$

Conflict analysis example





$$\neg a + \neg d + \neg f \geq 2 \qquad (C_5)$$

Width trimming

- Idea: weaken unused literals in equations

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

$$a + b + c + d + e + 10x \geq 1$$

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

$$a + b + c + d + e + 10x \geq 1$$

- How to propagate this conelits through pol lines ?

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

$$a+b+c+d+e+10x \geq 1$$

- How to propagate this conelits through pol lines ?

$$a+\neg b+\neg c+d+e+7x \geq 5 \quad (1)$$

$$2b+2x \geq 1 \quad (2)$$

$$c+x \geq 1 \quad (3)$$

`pol 1 2 3 + +` (4)

`e` $a+b+d+e+10x \geq 5$ (5)

Width trimming

- Idea: weaken unused literals in equations

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+10x+y+z \geq 21$$

$$a+b+c+d+e+10x \geq 1$$

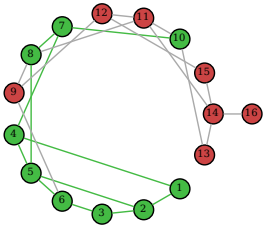
- How to propagate this conelits through pol lines ?

$$a+\neg b+\neg c+d+e+7x \geq 5 \quad (1)$$
$$2b+2x \geq 1 \quad (2)$$
$$c+x \geq 1 \quad (3)$$
`pol 1 2 3 + +` (4)
`e` $a+b+d+e+10x \geq 5$ (5)

$$conelits \cup (poslits \cap neglits)$$

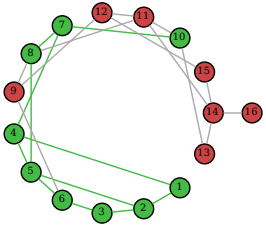
Visualisations: Core pattern graphs for SIP

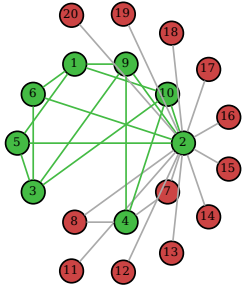
green = kept in the core pattern graph red = pruned



Visualisations: Core pattern graphs for SIP

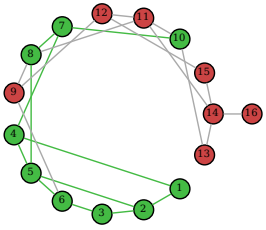
green = kept in the core pattern graph red = pruned

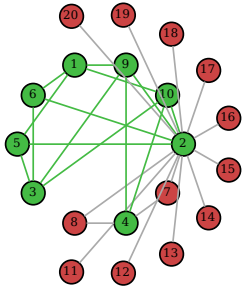


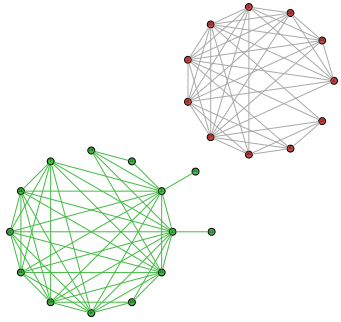


Visualisations: Core pattern graphs for SIP

green = kept in the core pattern graph red = pruned





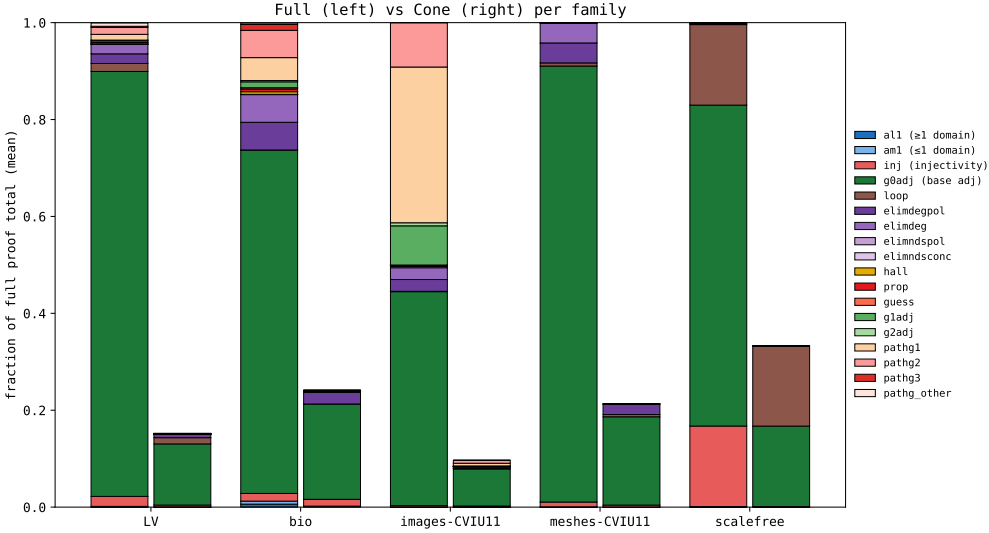


Resolv shrinkage

Family	n resolved	n unreduced	mean pattern shrinkage	max pattern shrinkage
LV	2540	3251	61.2%	99.9%
bio	4293	5443	40.0%	92.7%
images-CVIU11	1205	1641	33.4%	99.2%
meshes-CVIU11	1644	2902	59.2%	99.4%
scalefree	16	16	99.6%	99.6%

Table: Shrinkage $= (|V_{\text{init}}| - |V_{\text{trimmed}}|)/|V_{\text{init}}|$ of the pattern graph. n unreduced = instances where resolv did not shrink the pattern graph.

Cone vs. full: how much of the proof survives trimming?



The certificate needs only ≈10–33% of what the solver actually did

Which proof steps survive?

Label	LV	bio	img	mesh	sf
al1 ($\geq$ 1 domain)	33.3%	29.4%	23.3%	22.4%	93.0%
am1 ($\leq$ 1 domain)	17.5%	3.4%	8.7%	4.1%	93.0%
inj (injectivity)	18.8%	88.3%	77.2%	41.2%	0.0%
g0adj (base adjacency)	14.4%	27.8%	17.4%	20.3%	25.1%
loop (all-different)	78.1%	—	87.3%	73.8%	99.6%
elimdegpol (deg. elim. pol)	34.7%	42.3%	16.1%	52.4%	0.0%
elimdeg (degree elim.)	0.0%	0.1%	0.5%	0.0%	0.0%
elimndspol (nds elim. pol)	37.5%	4.7%	1.0%	—	—
elimndsconc (nds elim.)	0.0%	0.0%	0.0%	—	—

Label	LV	bio	img	mesh	sf
hall (Hall set)	12.4%	5.8%	5.4%	23.9%	—
prop (propagation)	5.2%	8.1%	12.2%	6.8%	—
guess (branching)	61.1%	33.5%	41.2%	39.3%	—
g1adj	0.7%	1.6%	1.6%	—	—
g2adj	1.6%	1.5%	0.2%	—	—
pathg1	0.8%	1.4%	1.7%	—	—
pathg2	6.8%	2.0%	6.8%	—	—
pathg3	4.9%	4.2%	—	—	—
pathg_other	7.4%	4.0%	—	—	—

img = images-CVIU11, mesh = meshes-CVIU11, sf = scalefree

Conclusion

Obrigado pela atenção!

Perguntas?

Future idea: justification

Log a small skeleton proof, trim it, then *justify* it back into a full proof on demand.

1 Log proof skeleton

F_1
F_2
assert C_1
assert C_2
assert C_3
assert C_4
C_5
C_6
$\bot$

Future idea: justification

Log a small skeleton proof, trim it, then *justify* it back into a full proof on demand.

1. Log proof skeleton
2. Trim proof skeleton

F_1	F_1
F_2	
assert C_1	
assert C_2	assert C_2
assert C_3	
assert C_4	assert C_4
C_5	C_5
C_6	
$\bot$	$\bot$

Future idea: justification

Log a small skeleton proof, trim it, then *justify* it back into a full proof on demand.

1. Log proof skeleton
2. Trim proof skeleton
3. Justify to get a full proof
4. Normal trimming and/or checking

F_1	F_1	F_1
F_2		$C_{2.1}$
assert C_1		$C_{2.2}$
assert C_2	assert C_2	$C_{2.3}$
assert C_3		$C_{2.4}$
assert C_4	assert C_4	$C_{2.5}$
C_5	C_5	$C_{2.6}$
C_6		C_2
$\bot$	$\bot$	$C_{4.1}$
		$C_{4.2}$
		$C_{4.3}$
		C_4
		C_5
		$\bot$