Skeptik

A Proof Compression System

Bruno Woltzenlogel Paleo
Joseph Boudou
Andreas Fellner

Why do we want to compress proofs?

- the fastest techniques to find proofs do not necessarily find the best proofs
- automatically generated proofs can be redundant
- Can we automatically improve proofs?
- Hilbert's 24th Problem:
  when is a proof better than another?
- Smaller proofs, faster proof checking, smaller unsat cores, better interpolants, ...

100% User Satisfaction

(but only one external user so far)

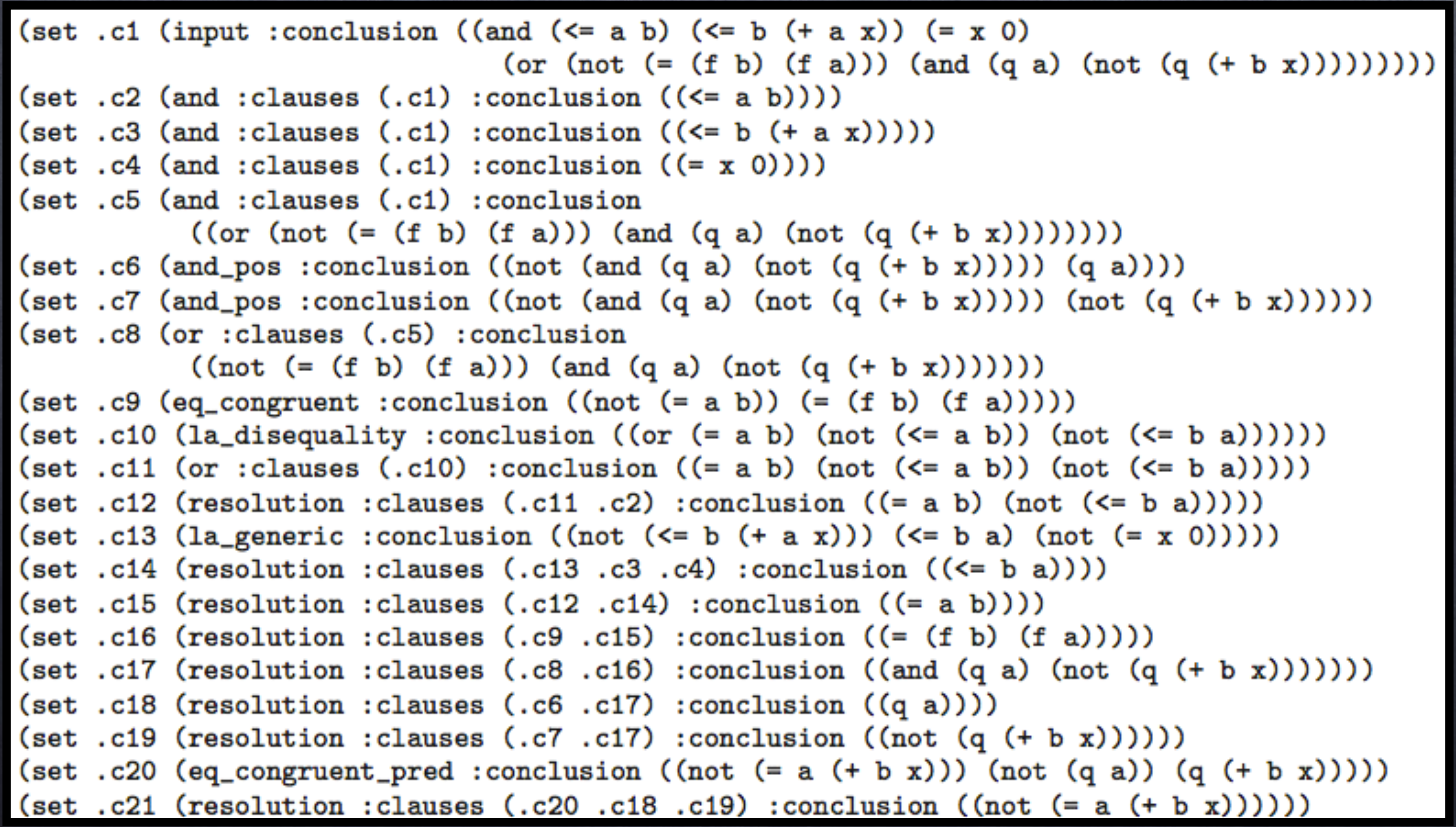
[Interpolation-Based Synthesis of Hardware Controllers]

"I tried, as explained in an earlier email, to obtain a new (purely propositional) proof from veriT, based on the stronger (colorable) theory lemmas that my splitting procedure generated.
This veriT proof has 1,870,407 nodes.
Skeptik reduced it (using RPI) to 868,760 nodes
(63,6% reduction).
[...]
So, aside from the long parsing times and
the rather high memory consumption,
Skeptik performs very well on my proofs and is definitely helpful. "
- Georg Hofferek

(more modest average compression ratio observed on problems of the SAT and SMT competitions)

Which kinds of proofs can Skeptik currently compress?

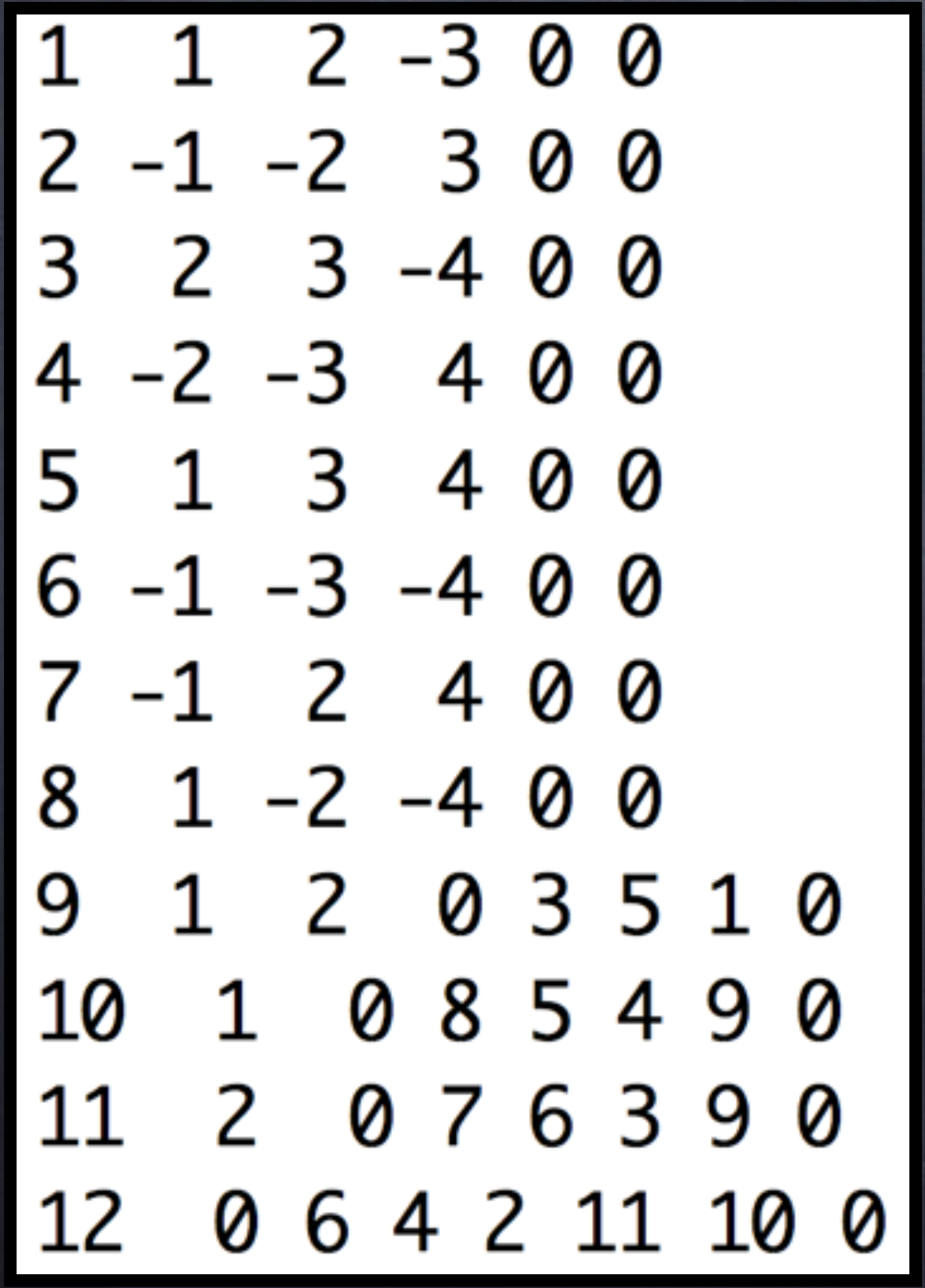
- SMT proofs produced by The veriT solver



```
(set .c1 (input :conclusion ((and (<= a b) (<= b (+ a x)) (= x 0)
                              (or (not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))))
(set .c2 (and :clauses (.c1) :conclusion ((<= a b))))
(set .c3 (and :clauses (.c1) :conclusion ((<= b (+ a x)))))
(set .c4 (and :clauses (.c1) :conclusion ((= x 0))))
(set .c5 (and :clauses (.c1) :conclusion
           ((or (not (= (f b) (f a))) (and (q a) (not (q (+ b x))))))))
(set .c6 (and_pos :conclusion ((not (and (q a) (not (q (+ b x))))) (q a))))
(set .c7 (and_pos :conclusion ((not (and (q a) (not (q (+ b x))))) (not (q (+ b x))))))
(set .c8 (or :clauses (.c5) :conclusion
           ((not (= (f b) (f a))) (and (q a) (not (q (+ b x)))))))
(set .c9 (eq_congruent :conclusion ((not (= a b)) (= (f b) (f a)))))
(set .c10 (la_disequality :conclusion ((or (= a b) (not (<= a b)) (not (<= b a))))))
(set .c11 (or :clauses (.c10) :conclusion ((= a b) (not (<= a b)) (not (<= b a)))))
(set .c12 (resolution :clauses (.c11 .c2) :conclusion ((= a b) (not (<= b a)))))
(set .c13 (la_generic :conclusion ((not (<= b (+ a x))) (<= b a) (not (= x 0)))))
(set .c14 (resolution :clauses (.c13 .c3 .c4) :conclusion ((<= b a))))
(set .c15 (resolution :clauses (.c12 .c14) :conclusion ((= a b))))
(set .c16 (resolution :clauses (.c9 .c15) :conclusion ((= (f b) (f a)))))
(set .c17 (resolution :clauses (.c8 .c16) :conclusion ((and (q a) (not (q (+ b x)))))))
(set .c18 (resolution :clauses (.c6 .c17) :conclusion ((q a))))
(set .c19 (resolution :clauses (.c7 .c17) :conclusion ((not (q (+ b x))))))
(set .c20 (eq_congruent_pred :conclusion ((not (= a (+ b x))) (not (q a)) (q (+ b x)))))
(set .c21 (resolution :clauses (.c20 .c18 .c19) :conclusion ((not (= a (+ b x))))))
```

Which kinds of proofs can Skeptik currently compress?

- SAT proofs in the TraceCheck Format



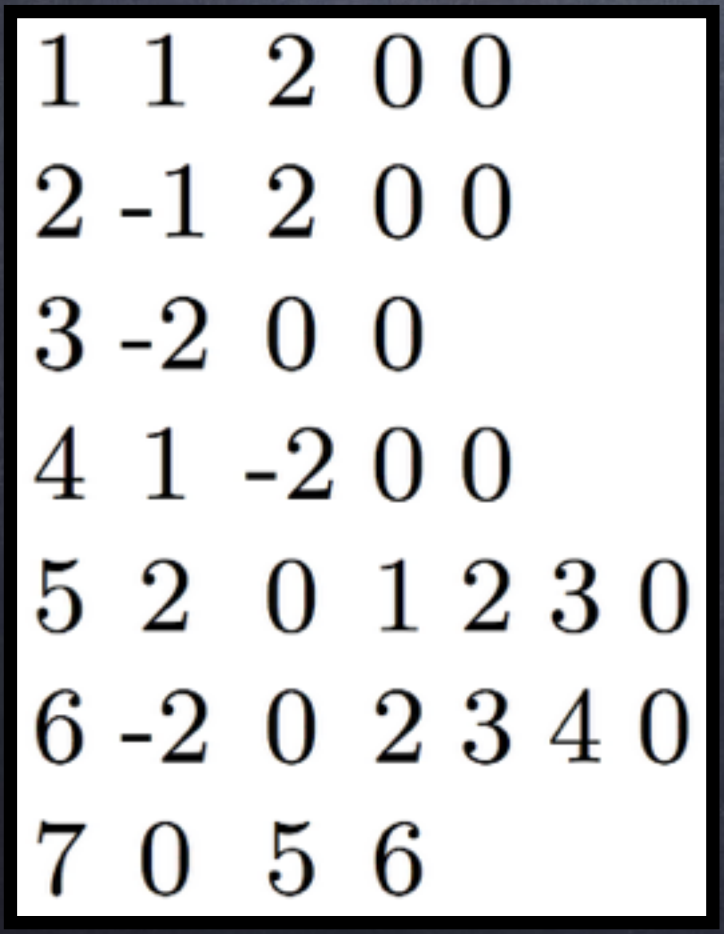
```
1  1  2 -3 0 0
2 -1 -2  3 0 0
3  2  3 -4 0 0
4 -2 -3  4 0 0
5  1  3  4 0 0
6 -1 -3 -4 0 0
7 -1  2  4 0 0
8  1 -2 -4 0 0
9  1  2  0 3 5 1 0
10  1  0 8 5 4 9 0
11  2  0 7 6 3 9 0
12  0 6 4 2 11 10 0
```

Conversion from DRUP to TraceCheck format possible with Marijn Heule's DRUP-Trim tool

Which kinds of proofs can Skeptik currently compress?

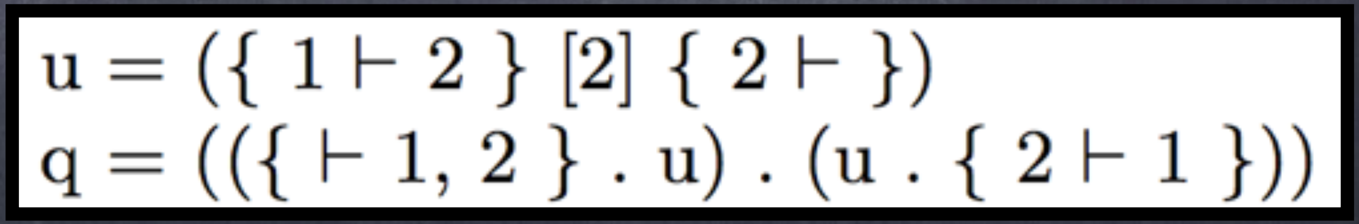
- Proofs in Skeptik's own proof format

TraceCheck format



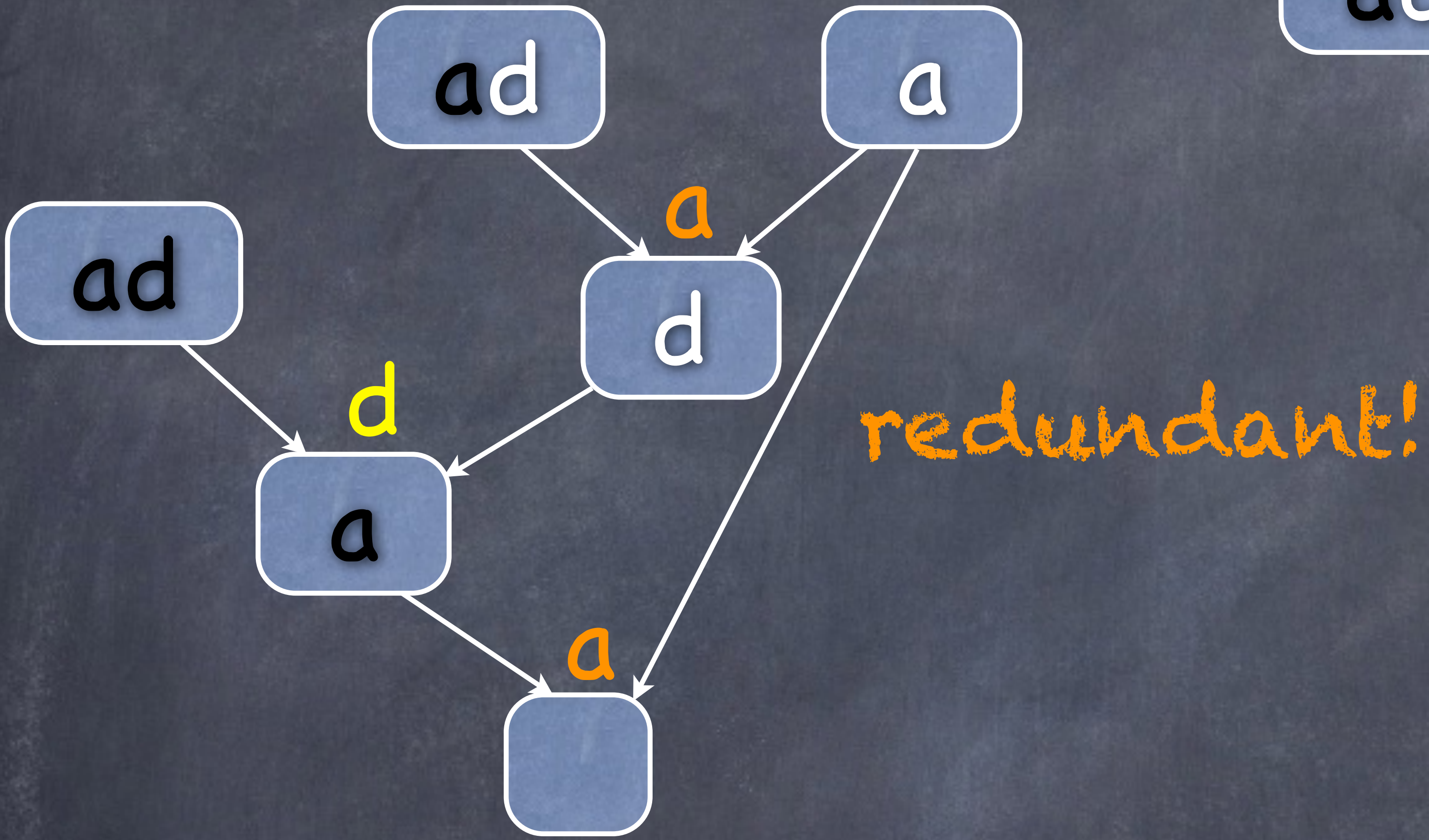
```
1  1  2 0 0
2 -1  2 0 0
3 -2  0 0
4  1 -2 0 0
5  2  0 1 2 3 0
6 -2  0 2 3 4 0
7  0  5 6
```

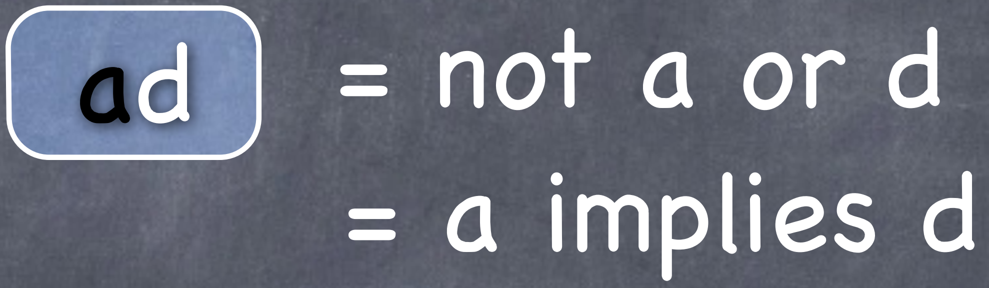
Skeptik's own format



$$u = (\{\, 1 \vdash 2 \,\}\ [2]\ \{\, 2 \vdash \,\})$$
$$q = (((\{\, \vdash 1, 2 \,\}\ .\ u)\ .\ (u\ .\ \{\, 2 \vdash 1 \,\}))$$

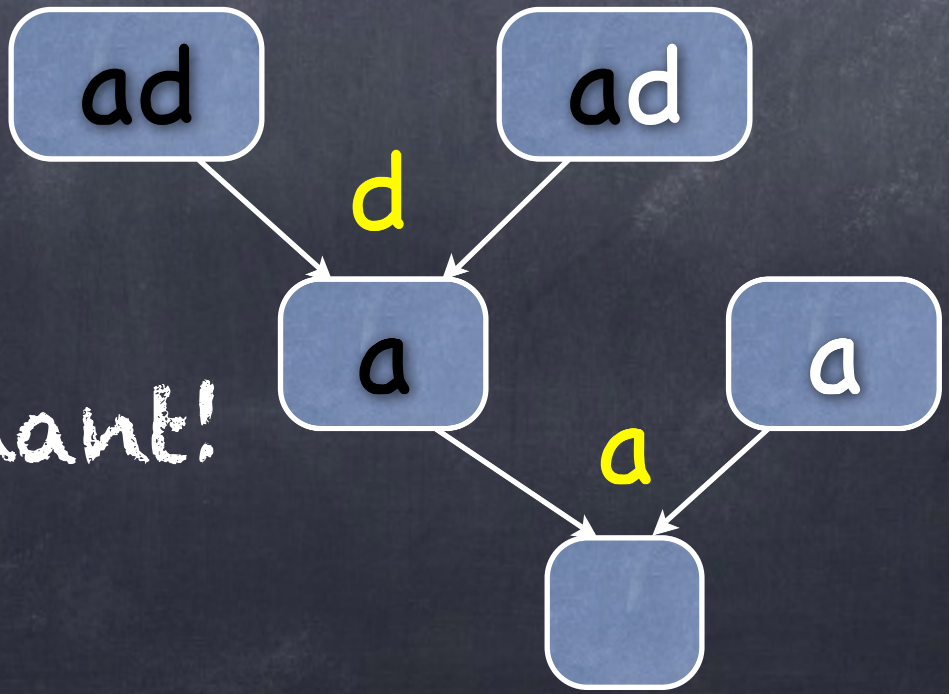
How does Skeptik compress proofs?





= not a or d

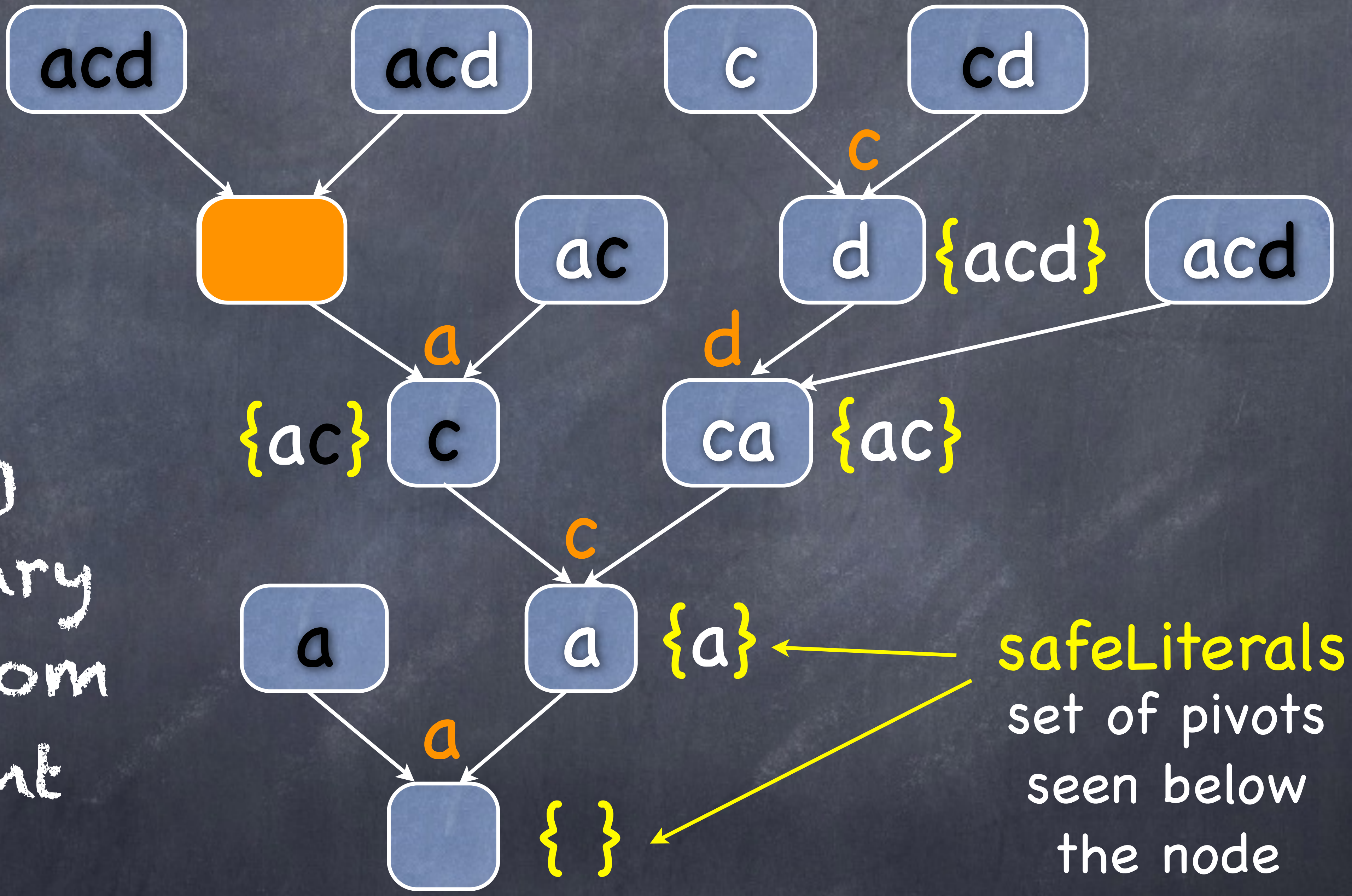
= a implies d



not redundant!

RecyclePivots

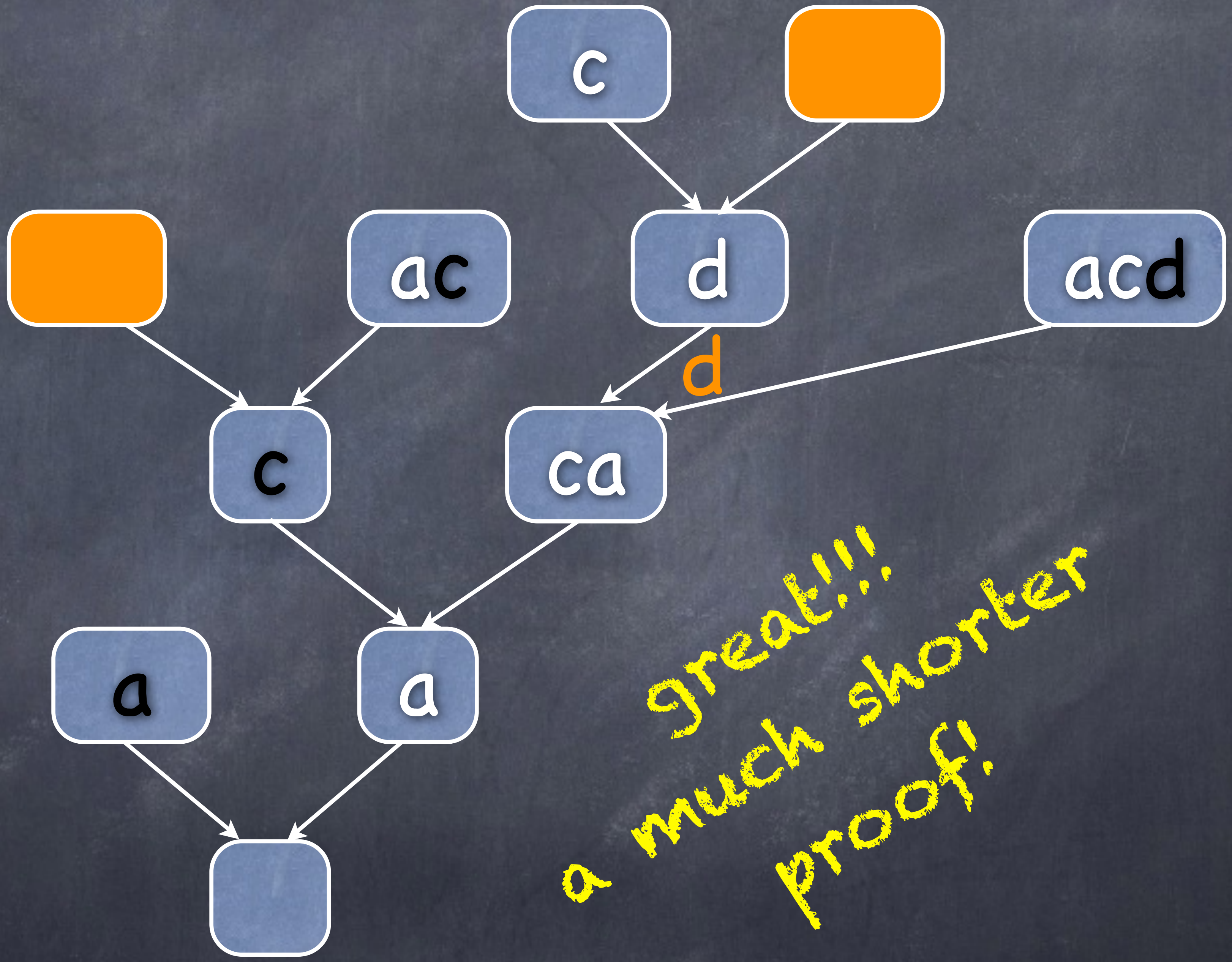
deleting unnecessary parents from redundant nodes



Bottom-up traversal

RecyclePivots

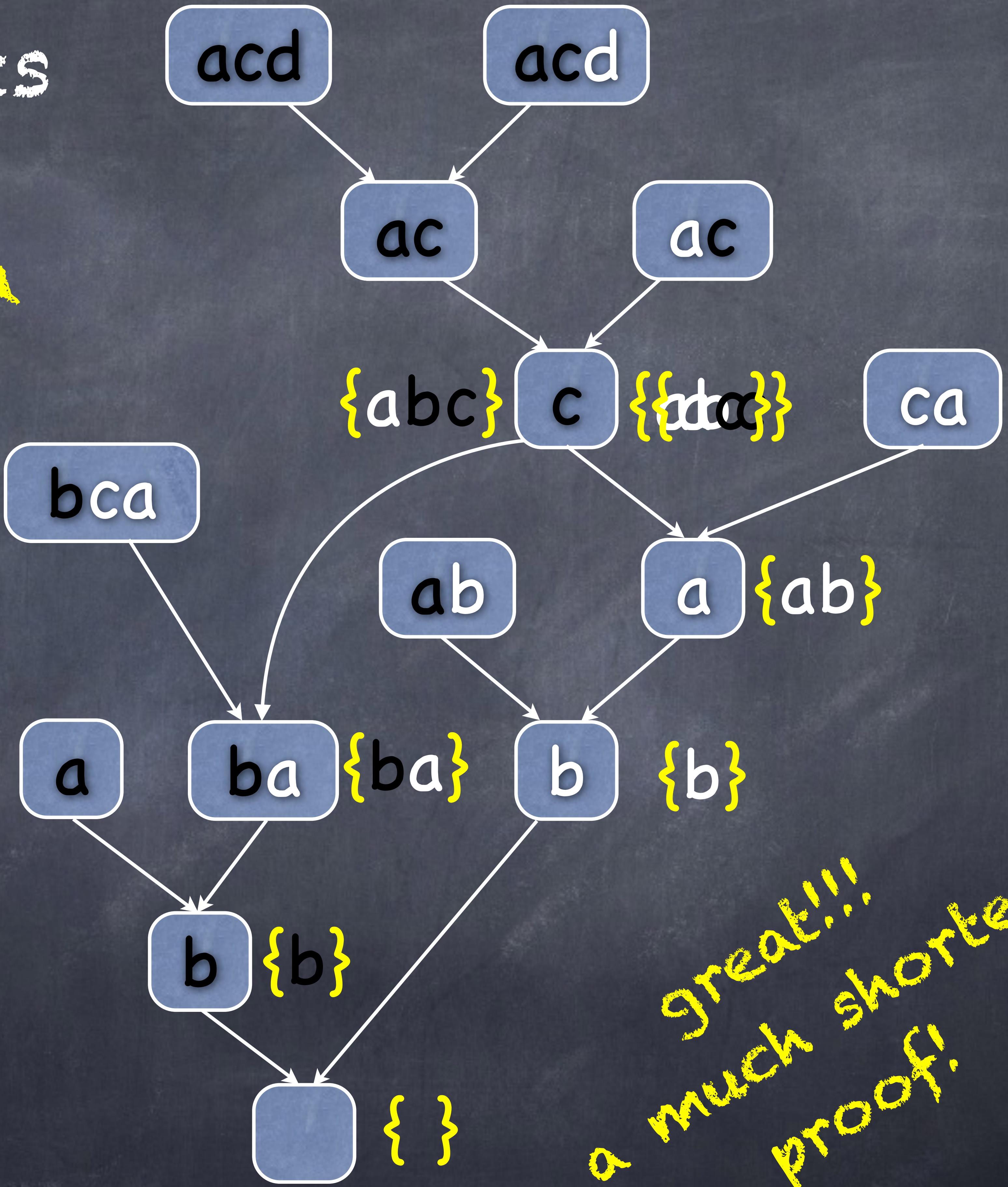
fixing the proof



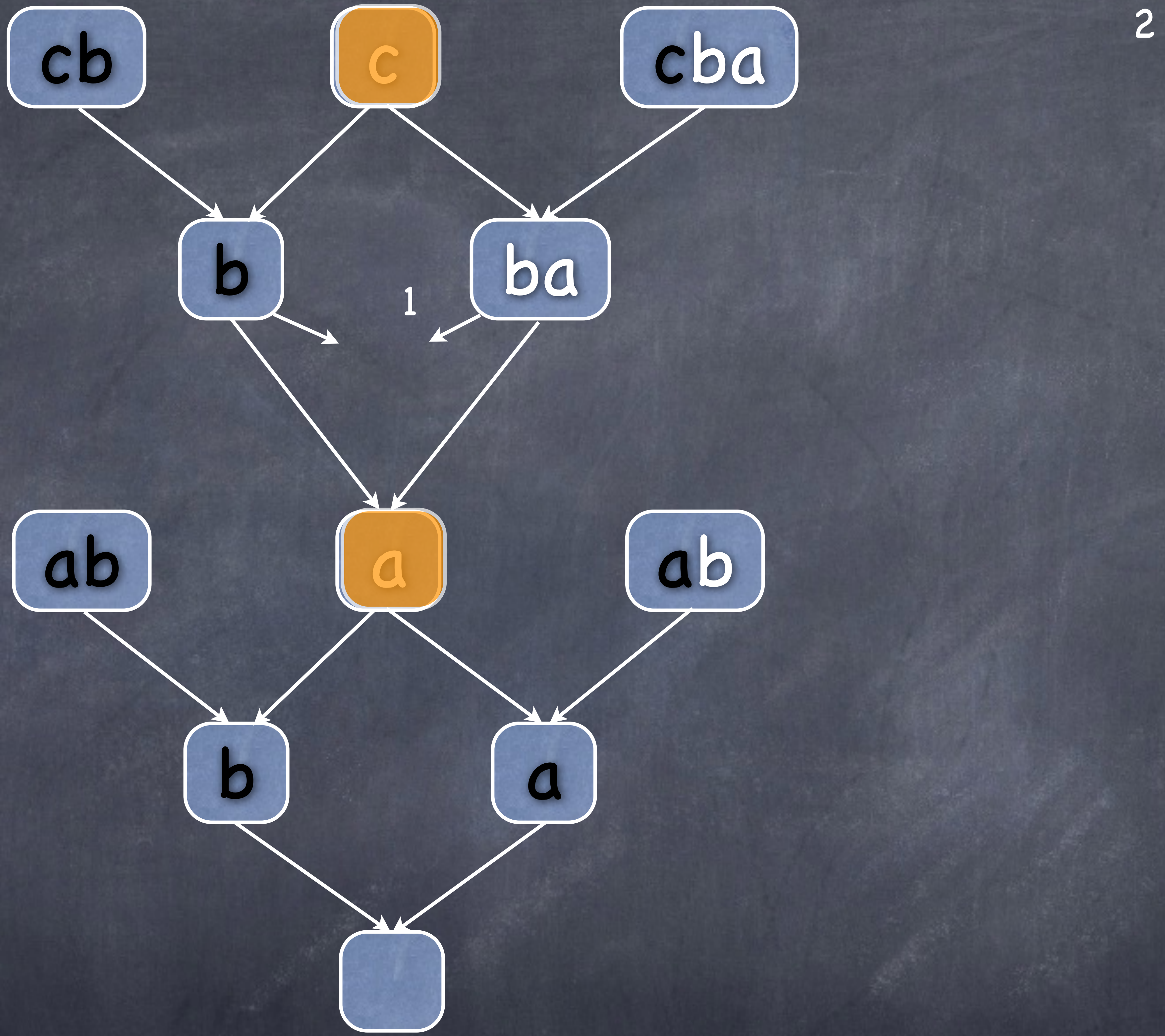
great!!!
a much shorter proof!

Top-down traversal

RecyclePivots with Intersection

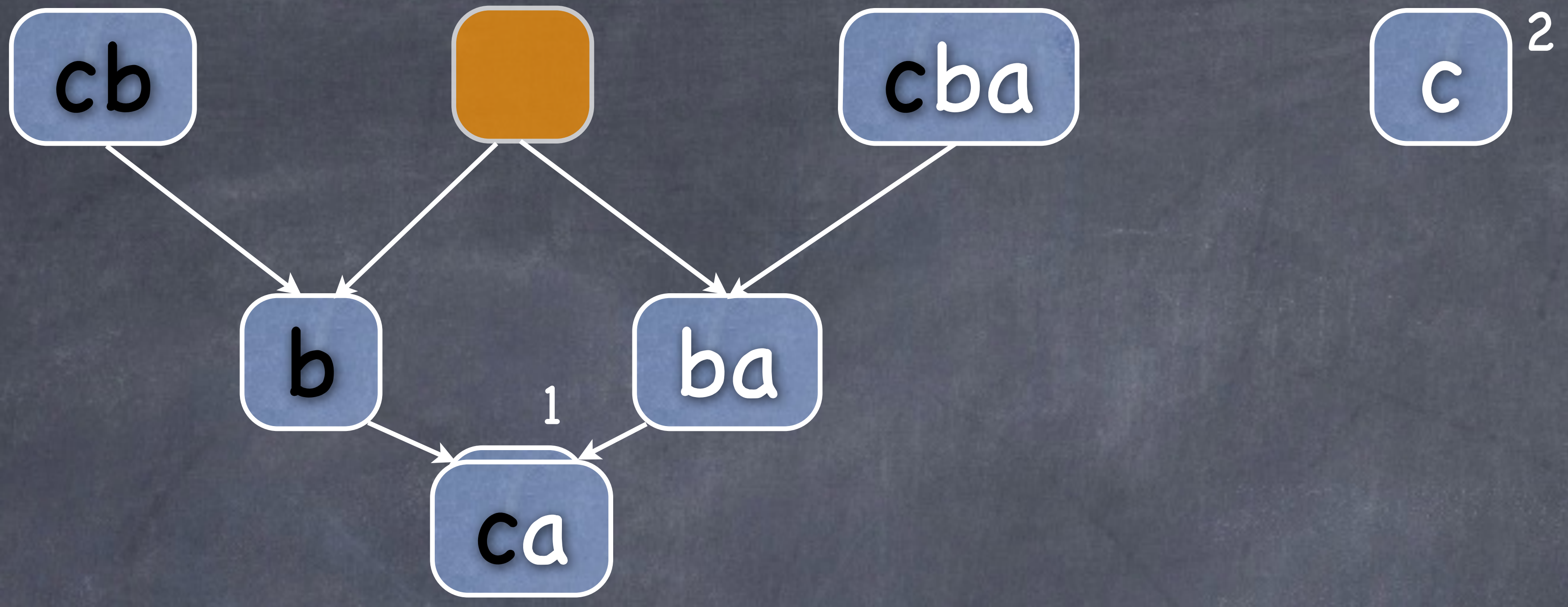


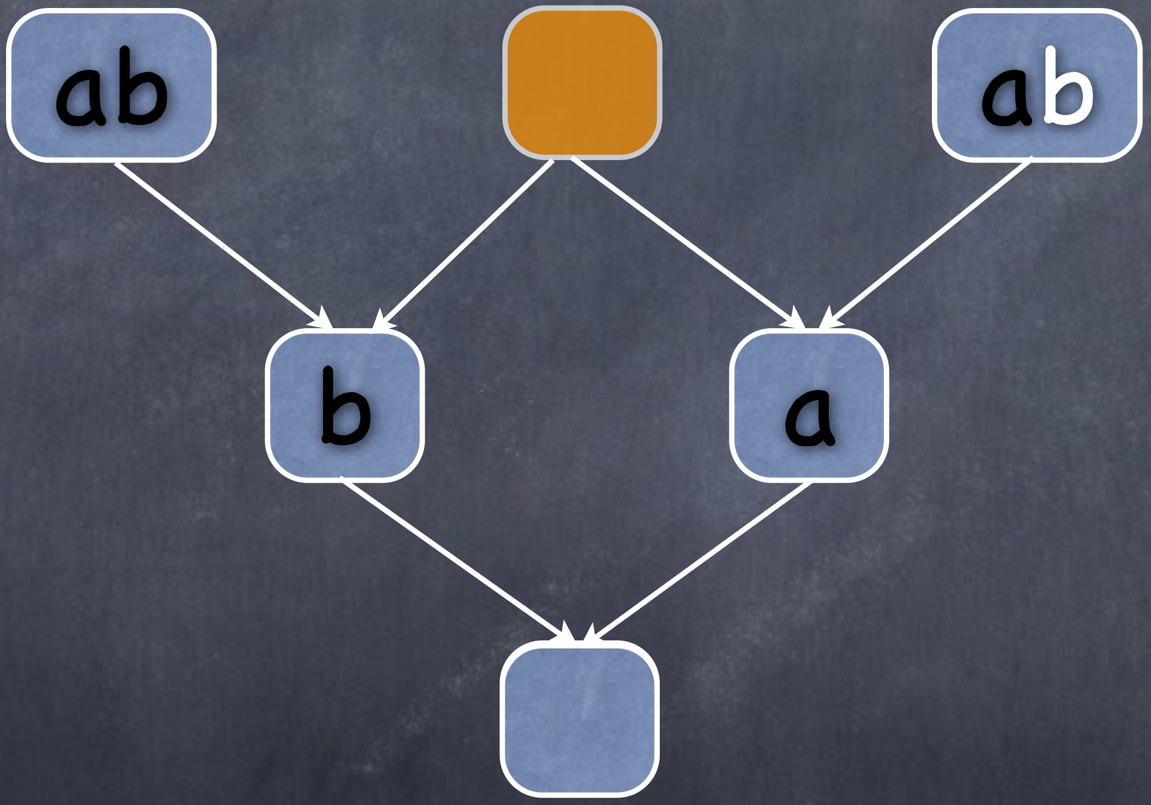
great!!! a much shorter proof!



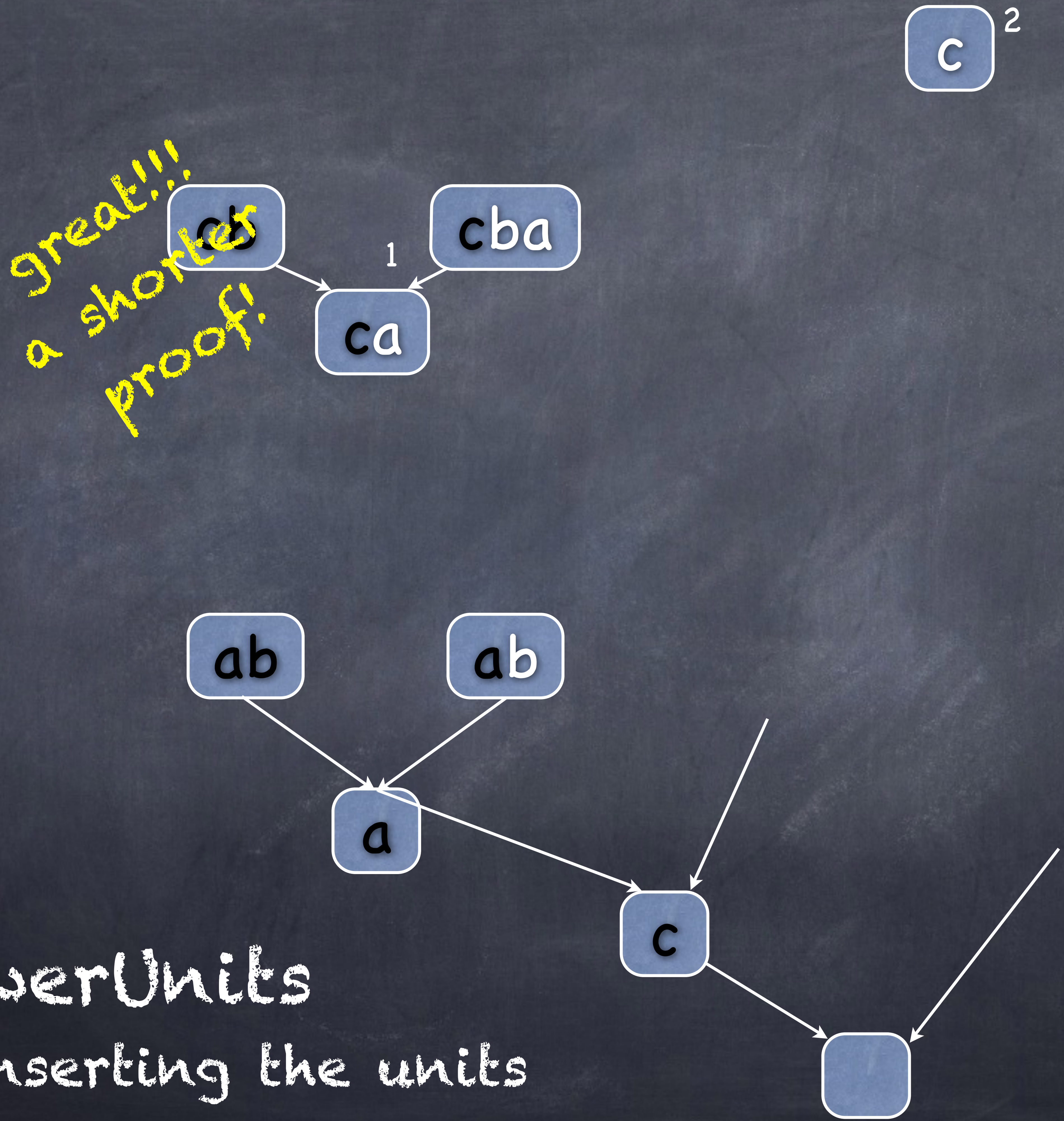
LowerUnits

collecting the units in a queue





LowerUnits
fixing the proof



LowerUnits

reinserting the units

What are Skeptik's algorithms?

- Algorithms focused on compressing proof length:
  - RecyclePivots, RecyclePivotsWithIntersection, LowerUnits, LowerUnivalents, RecycleUnits, Split, Reduce&Reconstruct, DAGify, EliminateTautologies, Combinations of RPI and LUV,...
- Pebbling algorithms for compressing proof space
- Algorithms for compressing the size of congruence closure lemmas

How is Skeptik implemented?

Scala Hybrid Functional Object-Oriented Language for the Java Virtual Machine

Immutability: no change on proofs after construction

Traits: reuse of code among compression algorithms

Case classes: emulation of inductive datatypes

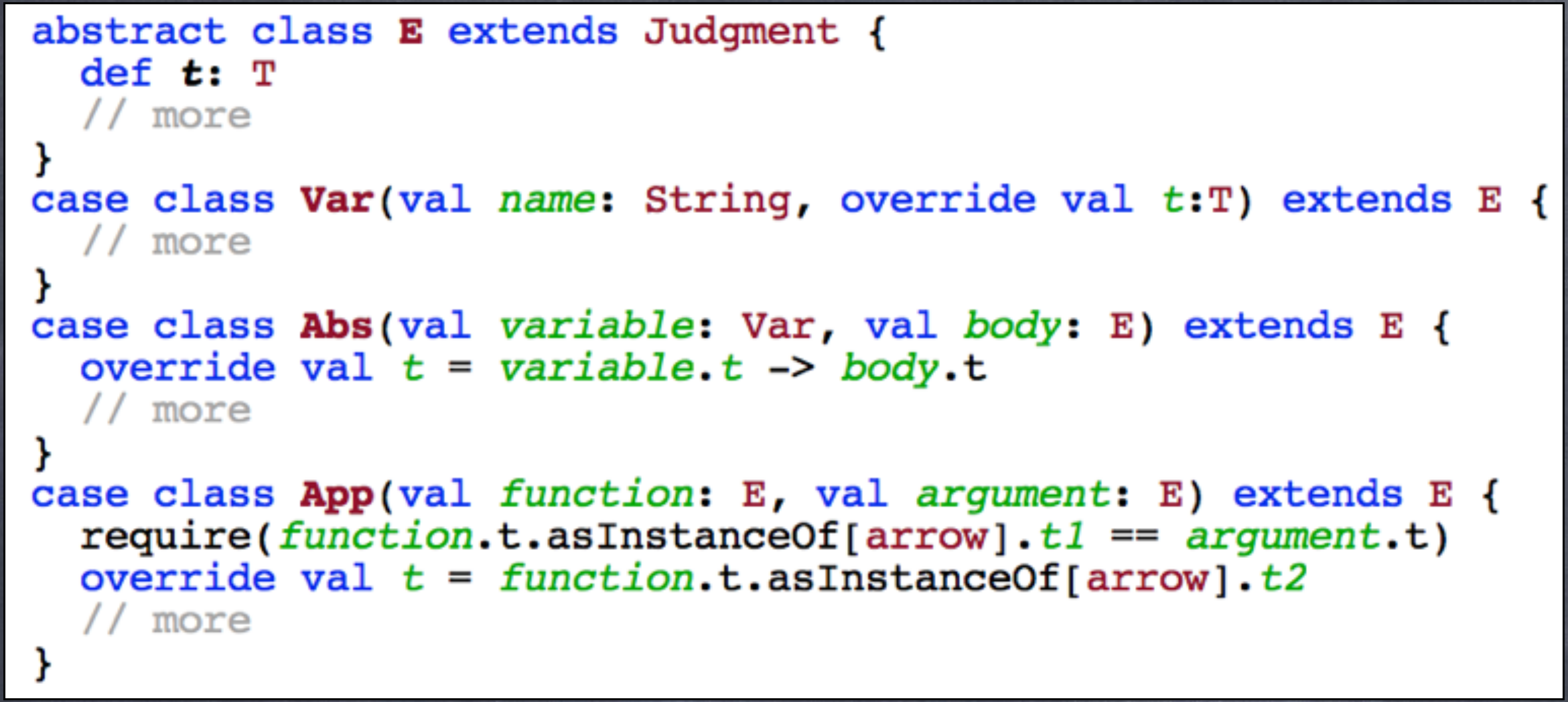
Combinator parsing: easy support for different proof formats

Type inference: less boilerplate than Java

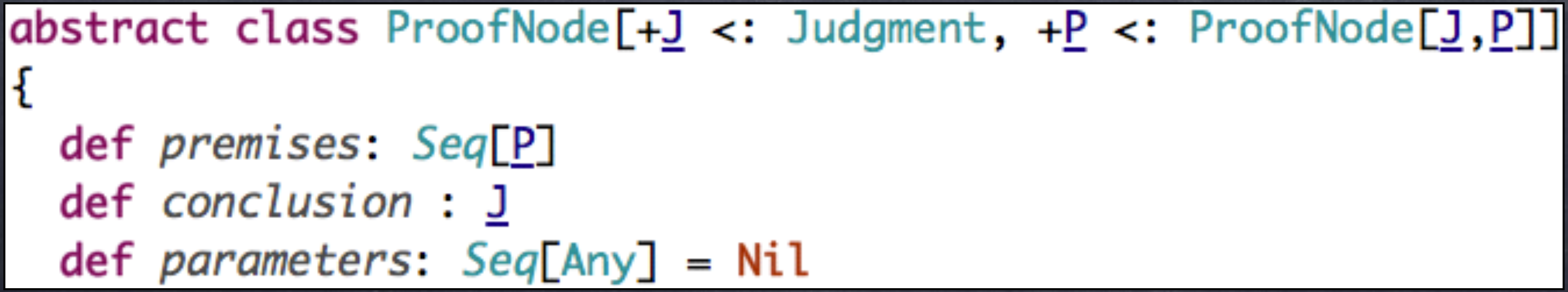
extractors, implicit arguments, implicit conversions, DSLs ...

(Functional + OO) = too many ways of doing the same thing

How is Skeptik implemented?

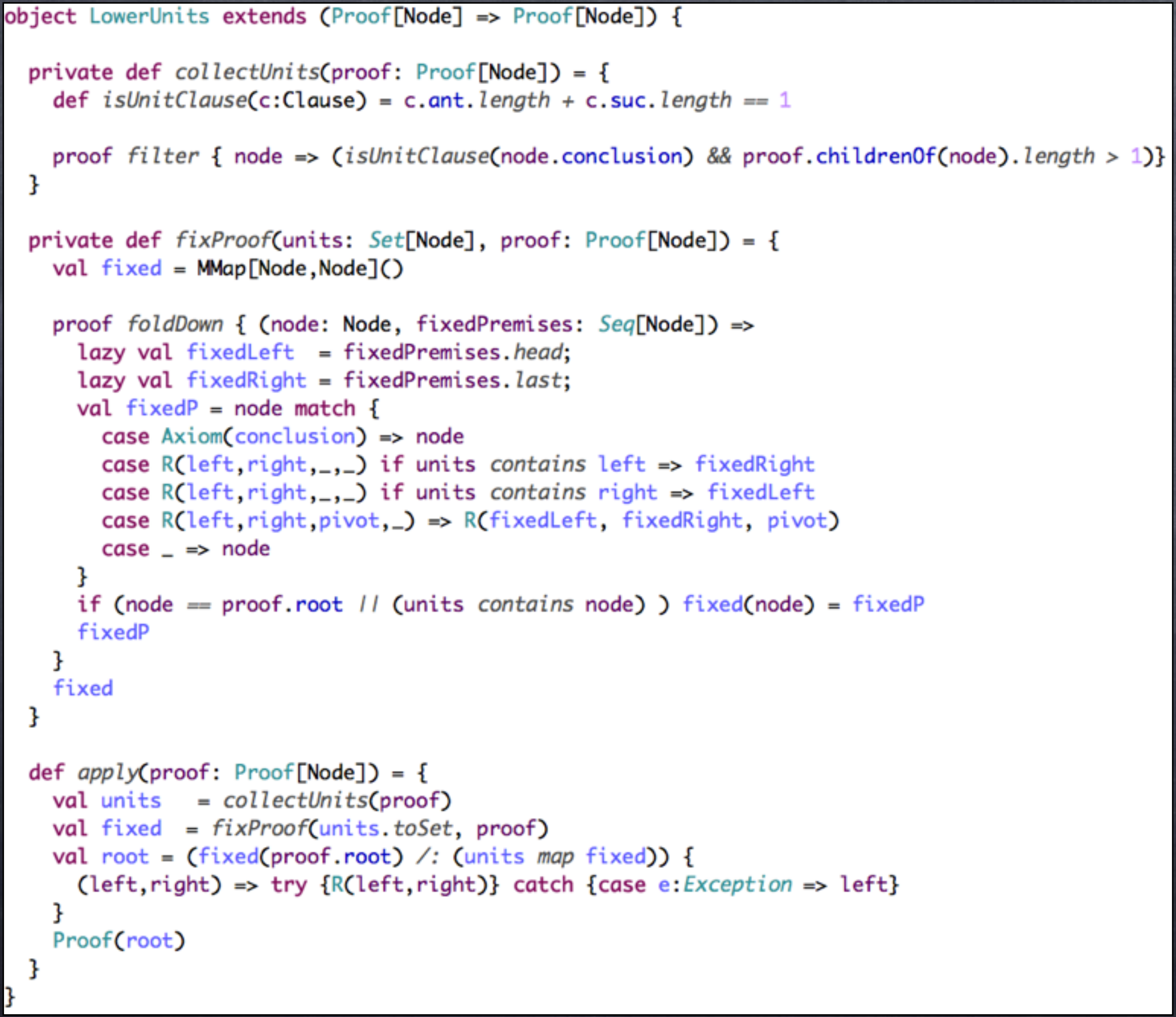


```
abstract class E extends Judgment {
  def t: T
  // more
}
case class Var(val name: String, override val t:T) extends E {
  // more
}
case class Abs(val variable: Var, val body: E) extends E {
  override val t = variable.t -> body.t
  // more
}
case class App(val function: E, val argument: E) extends E {
  require(function.t.asInstanceOf[arrow].t1 == argument.t)
  override val t = function.t.asInstanceOf[arrow].t2
  // more
}
```



```
abstract class ProofNode[+J <: Judgment, +P <: ProofNode[J,P]]
{
  def premises: Seq[P]
  def conclusion : J
  def parameters: Seq[Any] = Nil
```

How is Skeptik implemented?



```scala
object LowerUnits extends (Proof[Node] => Proof[Node]) {

  private def collectUnits(proof: Proof[Node]) = {
    def isUnitClause(c:Clause) = c.ant.length + c.suc.length == 1

    proof filter { node => (isUnitClause(node.conclusion) && proof.childrenOf(node).length > 1)}
  }

  private def fixProof(units: Set[Node], proof: Proof[Node]) = {
    val fixed = MMap[Node,Node]()

    proof foldDown { (node: Node, fixedPremises: Seq[Node]) =>
      lazy val fixedLeft  = fixedPremises.head;
      lazy val fixedRight = fixedPremises.last;
      val fixedP = node match {
        case Axiom(conclusion) => node
        case R(left,right,_,_) if units contains left => fixedRight
        case R(left,right,_,_) if units contains right => fixedLeft
        case R(left,right,pivot,_) => R(fixedLeft, fixedRight, pivot)
        case _ => node
      }
      if (node == proof.root || (units contains node) ) fixed(node) = fixedP
      fixedP
    }
    fixed
  }

  def apply(proof: Proof[Node]) = {
    val units   = collectUnits(proof)
    val fixed  = fixProof(units.toSet, proof)
    val root = (fixed(proof.root) /: (units map fixed)) {
      (left,right) => try {R(left,right)} catch {case e:Exception => left}
    }
    Proof(root)
  }
}
```

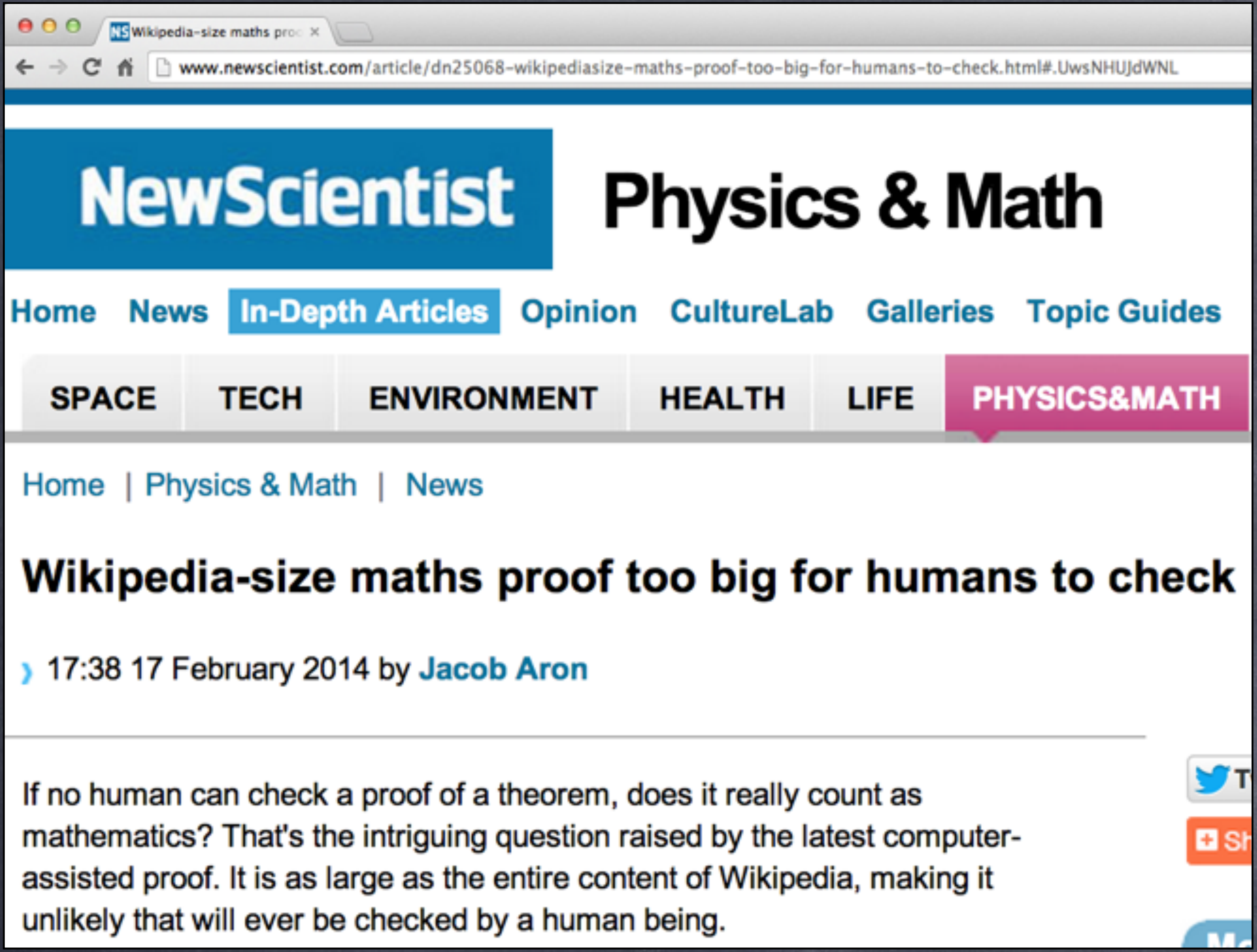
Conclusion

- Skeptik is ready! Please use it!
- https://github.com/Paradoxika/Skeptik

Current Work

- 2014 GSoC student extending RPI and LU to first-order

Thanks !!!



Wikipedia-size maths proo ×

www.newscientist.com/article/dn25068-wikipediasize-maths-proof-too-big-for-humans-to-check.html#.UwsNHUJdWNL

NewScientist Physics & Math

Home News In-Depth Articles Opinion CultureLab Galleries Topic Guides

SPACE TECH ENVIRONMENT HEALTH LIFE PHYSICS&MATH

Home | Physics & Math | News

Wikipedia-size maths proof too big for humans to check

17:38 17 February 2014 by Jacob Aron

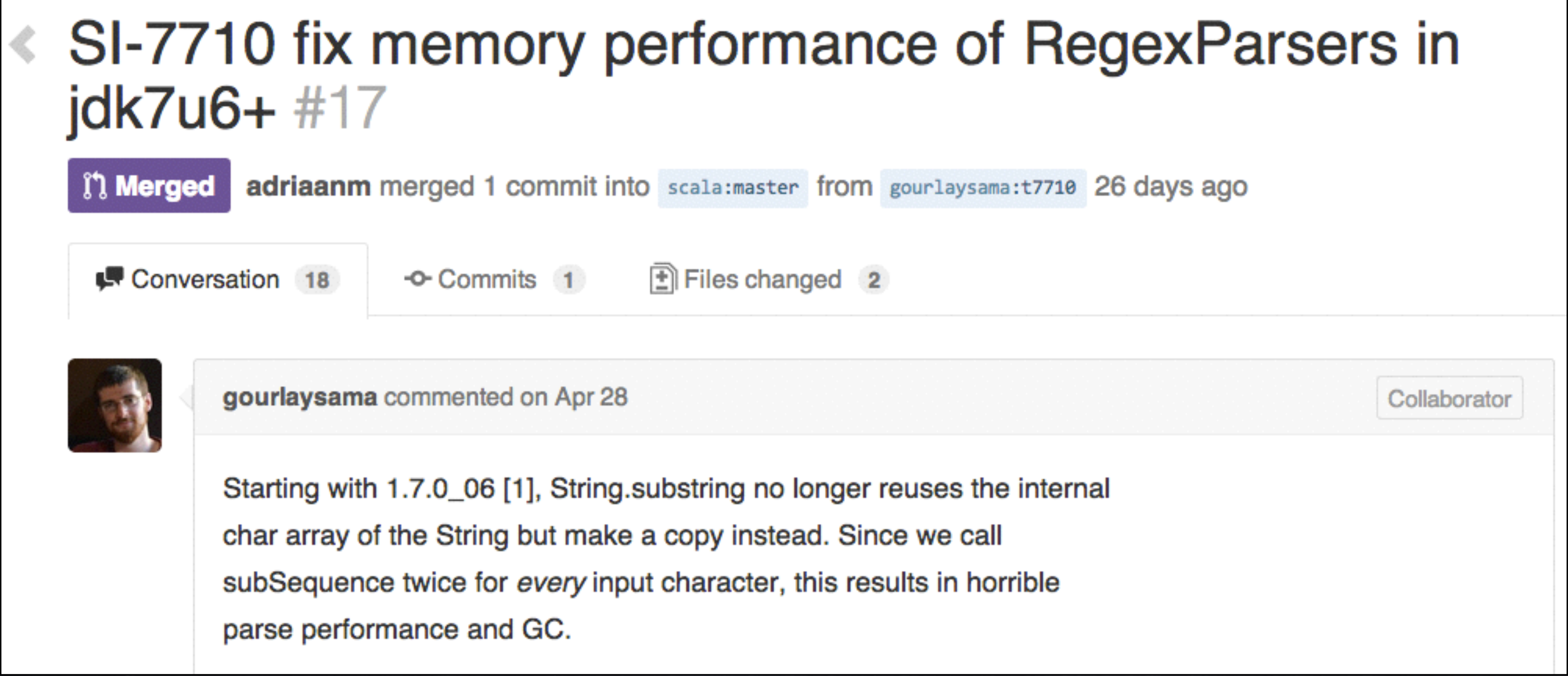
If no human can check a proof of a theorem, does it really count as mathematics? That's the intriguing question raised by the latest computer-assisted proof. It is as large as the entire content of Wikipedia, making it unlikely that will ever be checked by a human being.

13GB proof file in DRUP format

6 hours to generate

6 hours to convert to TraceCheck format

Skeptik unable to parse it within 3 days!



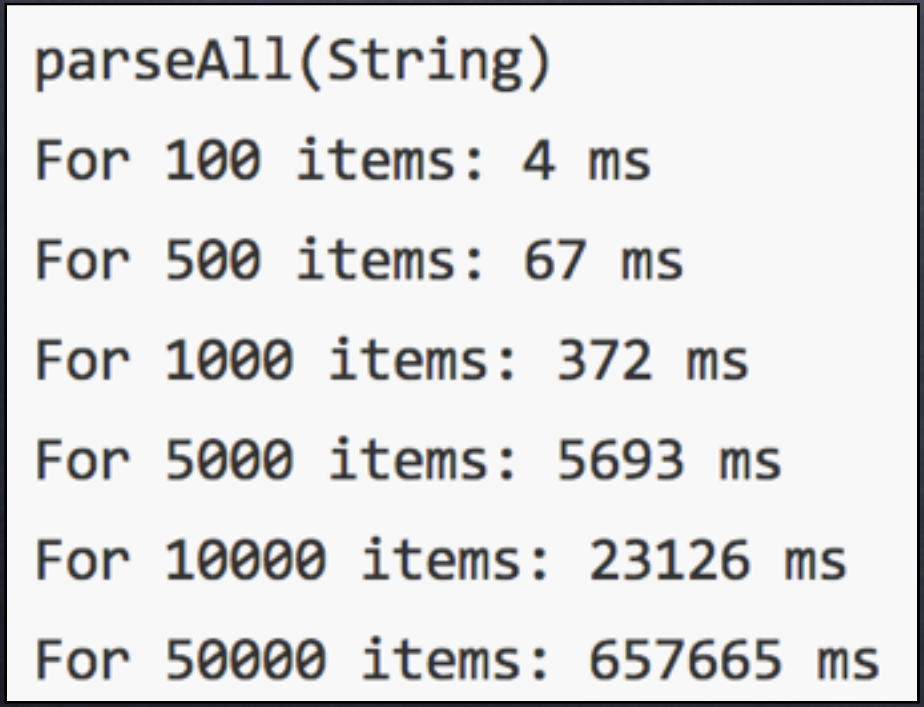
SI-7710 fix memory performance of RegexParsers in jdk7u6+ #17

Merged adriaanm merged 1 commit into `scala:master` from `gourlaysama:t7710` 26 days ago

gourlaysama commented on Apr 28 Collaborator

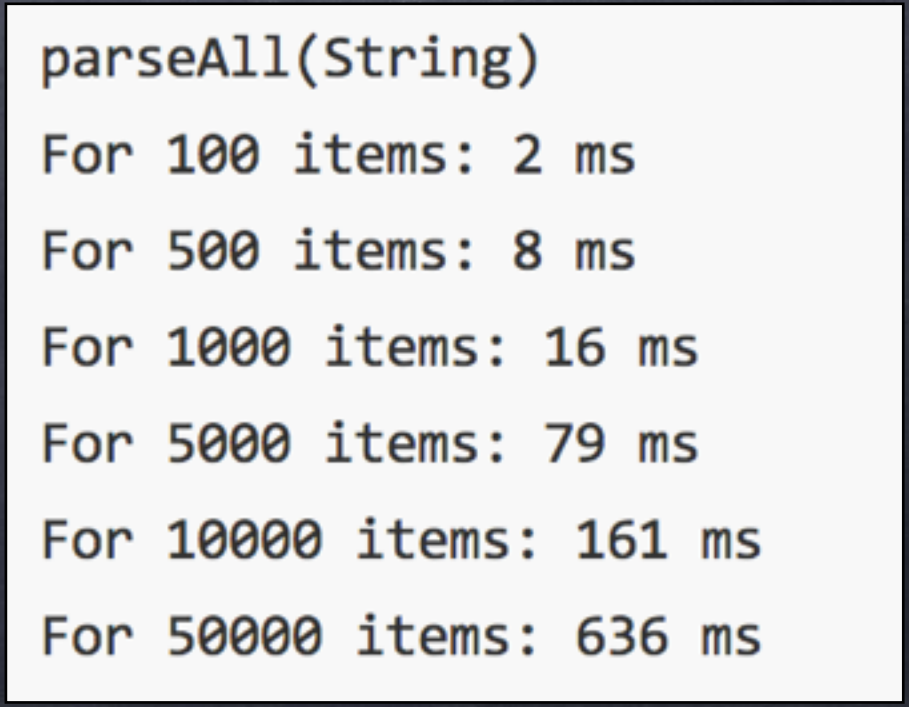
Starting with 1.7.0_06 [1], String.substring no longer reuses the internal char array of the String but make a copy instead. Since we call subSequence twice for *every* input character, this results in horrible parse performance and GC.

before the fix



```
parseAll(String)
For 100 items: 4 ms
For 500 items: 67 ms
For 1000 items: 372 ms
For 5000 items: 5693 ms
For 10000 items: 23126 ms
For 50000 items: 657665 ms
```

after the fix



```
parseAll(String)
For 100 items: 2 ms
For 500 items: 8 ms
For 1000 items: 16 ms
For 5000 items: 79 ms
For 10000 items: 161 ms
For 50000 items: 636 ms
```