Proof Schema and the Refutational Complexity of their Cut Structure

David M. Cerna

July 19^{th}, 2018

Motivating example

- A colleague often mentions the following problem as a canonical "difficult" problem for inductive theorem provers.

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$
$$x + (x + x) = (x + x) + x$$

- Why is it hard? Easy, it requires "hard to find" lemmata.
- But what does this tell us about theorem provers which
  a) find the required lemmata? (Like Viper [Eberhard & Hetzl , 2015], [Ebner & Hetzl , 2015])
  b) do not find the required lemmata? (Everything else?)
- What we want to know is what to expect from a given prover.
- Essentially, *complexity measures* for inductive theorem prover.
- Unexpectedly we start from the analysis of recursively defined formal proofs.

Proof Schema: a.k.a Yet Another Formalism of Induction

- A *schema of proofs* was used to analyze Fürstenberg's proof of the infinitude of primes [Baaz et al. 2008].
- Proof Schema are a formal description of this concept.
- More precisely, they are recursively defined infinite sequence of finite proofs indexed by a vector of free numeric parameters, which when grounded and normalized produce a *first-order proof*.
- Links between proofs define the recursive construction.
- Recent work [Cerna & Lolic , 2018], has shown equivalence to Peano Arithmetic.

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$\frac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash P(\beta), \Delta, \Gamma}$$

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$\frac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash \xcancel{P(\beta)}, \Delta, \Gamma}$$

$$P(s(\alpha))$$

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$
\dfrac{\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'}}{\vdots} \qquad \dfrac{\dfrac{\varphi(0)}{\Sigma \vdash P(0), \Delta}}{\vdots}}{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''}
$$

$$
\dfrac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash \cancel{P(\beta)}, \Delta, \Gamma} \qquad \Rightarrow
$$

$$P(s(\alpha))$$

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$\dfrac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash \underset{P(s(\alpha))}{\cancel{P(\beta)}}, \Delta, \Gamma} \quad \Rightarrow \quad \begin{array}{c}
\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'} \qquad \dfrac{\dfrac{\varphi(0)}{\Sigma \vdash P(0), \Delta}}{}}{\begin{array}{cc} \vdots & \vdots \end{array}} \\
\overline{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''} \\
\Downarrow \\
\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'} \qquad \dfrac{\dfrac{\varphi(1)}{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''}}{}}{\begin{array}{cc} \vdots & \vdots \end{array}} \\
\overline{\Pi''', \Sigma \vdash P(2), \Delta, \Gamma'''}
\end{array}$$

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$
\dfrac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash \cancel{P(\beta)}, \Delta, \Gamma}
$$

$$P(s(\alpha))$$

$$\Rightarrow$$

$$
\dfrac{\begin{array}{cc} \dfrac{\Psi}{\Pi' \vdash \Gamma'} & \dfrac{\overset{\varphi(0)}{\cdots\cdots\cdots}}{\Sigma \vdash P(0), \Delta} \\ \vdots & \vdots \end{array}}{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''}
$$

$$\Downarrow$$

$$
\dfrac{\begin{array}{cc} \dfrac{\Psi}{\Pi' \vdash \Gamma'} & \dfrac{\overset{\varphi(1)}{\cdots\cdots\cdots}}{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''} \\ \vdots & \vdots \end{array}}{\Pi''', \Sigma \vdash P(2), \Delta, \Gamma'''}
$$

$$\Downarrow \quad \alpha - 2 \ \textit{times}$$

$$
\dfrac{\begin{array}{cc} \dfrac{\Psi}{\Pi' \vdash \Gamma'} & \dfrac{\overset{\varphi(\alpha)}{\cdots\cdots\cdots}}{\Pi^{(\alpha+1)}, \Sigma \vdash P(\alpha), \Delta, \Gamma^{(\alpha+1)}} \\ \vdots & \vdots \end{array}}{\Pi, \Sigma \vdash P(\alpha + 1), \Delta, \Gamma}
$$

Mechanism behind Schematic Proofs

- Proof Schemata interpret arithmetic induction as a primitive recursive proof definition.

$$\dfrac{\Sigma \vdash P(0), \Delta \qquad \Pi, P(\alpha) \vdash P(s(\alpha)), \Gamma}{\Pi, \Sigma \vdash \cancel{P(\beta)}, \Delta, \Gamma}$$
$$P(s(\alpha))$$

$$\Rightarrow$$

$$\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'} \qquad \dfrac{\overset{\varphi(0)}{\cdots\cdots}}{\Sigma \vdash P(0), \Delta}}{\begin{array}{cc} \vdots & \vdots \end{array}}$$
$$\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'} \qquad \dfrac{\overset{\varphi(1)}{\cdots\cdots}}{\Pi'', \Sigma \vdash P(1), \Delta, \Gamma''}}{\begin{array}{cc} \vdots & \vdots \end{array}}$$
$$\Pi''', \Sigma \vdash P(2), \Delta, \Gamma'''$$

$$\Downarrow \quad \alpha - 2 \ \textit{times}$$

$$\dfrac{\dfrac{\Psi}{\Pi' \vdash \Gamma'} \qquad \dfrac{\overset{\varphi(\alpha)}{\cdots\cdots}}{\Pi^{(\alpha+1)}, \Sigma \vdash P(\alpha), \Delta, \Gamma^{(\alpha+1)}}}{\begin{array}{cc} \vdots & \vdots \end{array}}$$
$$\Pi, \Sigma \vdash P(\alpha + 1), \Delta, \Gamma$$

- The proof is indexed by α.
- Instantiating α results in an **LK**-proof .
- Formally a proof pair $\langle \varphi(0), \varphi(n+1) \rangle$.

Proof Schema: by comparison

- Unlike formal systems using so called ω-rules, the recursive construction is an explicit part of the object language.
- In contrast to cyclic proof formalisms, proofs are not by infinite descent, i.e. they do not unroll into regular infinite proof trees.
- Informally, one can think of proof schemata as a sequence of proofs converging to a regular infinite proof tree.
- The formalism allows easy tracking of formula occurrences.
- Occurrence tracking is essential for schematic cut-elimination.

Global versus *Local* cut-elimination

Local cut-elimination reduces a cut formula's complexity or its distance from the leaves.

- Introduced by Gentzen as a method of proving consistency, the concept has been expanded well beyond the intended scope.

Global versus *Local* cut-elimination

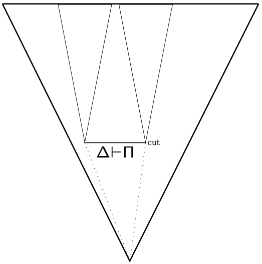
Local cut-elimination reduces a cut formula's complexity or its distance from the leaves.

- Introduced by Gentzen as a method of proving consistency, the concept has been expanded well beyond the intended scope.

Global cut-elimination produces an intermediate representation of a formal proofs cut-structure.

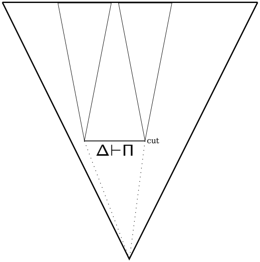
- From this intermediate representation a new proof with a **trivial cut-structure** is produced.

CERES: The Characteristic Clause Set representation

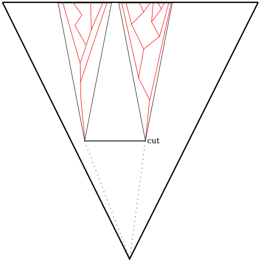


LK-Proof with cuts

CERES: The Characteristic Clause Set representation

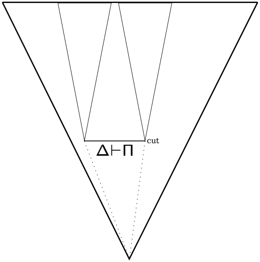


LK-Proof with cuts

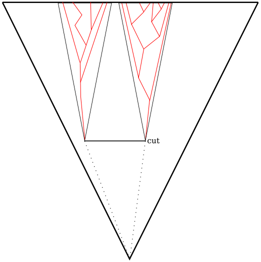


Paths to cut ancestors

CERES: The Characteristic Clause Set representation



LK-Proof with cuts

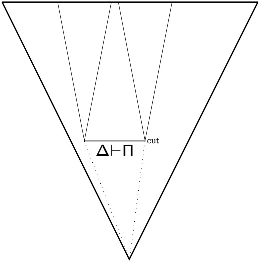


Paths to cut ancestors

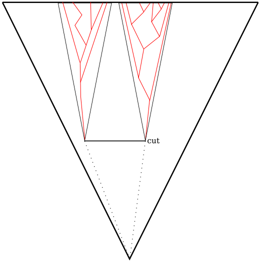
$CL(A \vdash A) \equiv \{\vdash A\}$
$CL(A \vdash A) \equiv \{A \vdash\}$
$CL(A \vdash A) \equiv \{A \vdash A\}$

- Construct a clause set from the cut ancestors relation.
- Such a clause set is always unsatisfiable.

CERES: The Characteristic Clause Set representation



LK-Proof with cuts



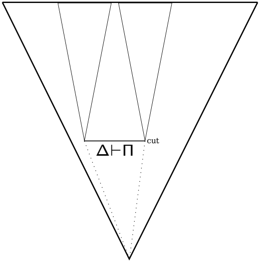
Paths to cut ancestors

$$CL(A \vdash A) \equiv \{\vdash A\}$$
$$CL(A \vdash A) \equiv \{A \vdash\}$$
$$CL(A \vdash A) \equiv \{A \vdash A\}$$

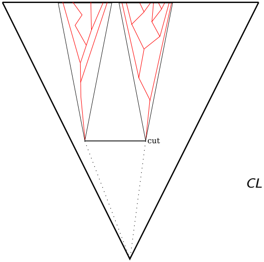
$$CL\left(\frac{\Delta \vdash \Pi}{\Delta' \vdash \Pi'} \; \rho \right) \equiv CL(\Delta \vdash \Pi)$$

- Construct a clause set from the cut ancestors relation.
- Such a clause set is always unsatisfiable.

CERES: The Characteristic Clause Set representation



LK-Proof with cuts



Paths to cut ancestors

$$CL(A \vdash A) \equiv \{\vdash A\}$$
$$CL(A \vdash A) \equiv \{A \vdash\}$$
$$CL(A \vdash A) \equiv \{A \vdash A\}$$

$$CL\left(\frac{\Delta \vdash \Pi}{\Delta' \vdash \Pi'}\ \rho\right) \equiv CL(\Delta \vdash \Pi)$$

$$CL\left(\frac{\Delta \vdash \Pi \qquad \Delta' \vdash \Pi'}{\Delta'' \vdash \Pi''}\ \rho\right) \equiv$$

$$\begin{cases} CL(\Delta \vdash \Pi) \cup CL(\Delta' \vdash \Pi') \\ CL(\Delta \vdash \Pi) \times CL(\Delta' \vdash \Pi') \end{cases}$$

- Construct a clause set from the cut ancestors relation.
- Such a clause set is always unsatisfiable.

Local Cut-elimination and Recursion

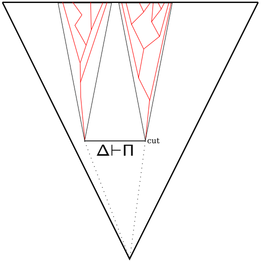
- Essentially cut reduction fails once it reaches a link (recursive call).

$$\dfrac{\dfrac{(\varphi_I, t, \bar{x})}{C, \Delta \vdash \Gamma} \qquad \dfrac{(\varphi_j, t', \bar{x})}{\Delta' \vdash \Gamma', C}}{\Delta, \Delta' \vdash \Gamma, \Gamma'}\ \mathrm{cut}$$

- For related formalisms cuts are eliminated from an infinite proof tree.
- One can define a relation between proof schema extending local cut-elimination and providing a sort of "cut-elimination" through clausal subsumption [Cerna & Lettmann 2017].

Clausal analysis: Reductive C.E. and Global Cut Structure

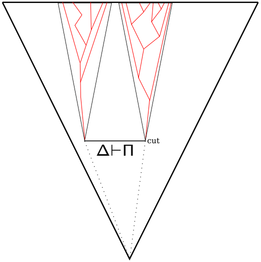
- Baaz and Leitsch, 2006 show how locally reducing cuts impacts the global cut structure.
- Every proof can be transformed into a proof with a minimally complex cut structure.
- The extracted clause set, is subsumed by the clause sets of the more complex cut structure.



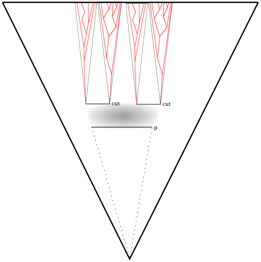
Reduction can result in

Clausal analysis: Reductive C.E. and Global Cut Structure

- Baaz and Leitsch, 2006 show how locally reducing cuts impacts the global cut structure.
- Every proof can be transformed into a proof with a minimally complex cut structure.
- The extracted clause set, is subsumed by the clause sets of the more complex cut structure.



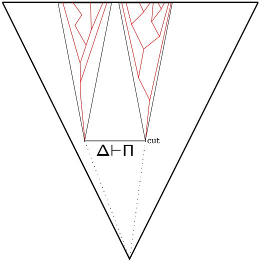
Reduction can result in



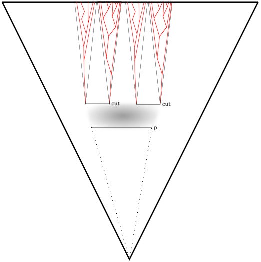
the following proof.

Clausal analysis: Reductive C.E. and Global Cut Structure

- Baaz and Leitsch, 2006 show how locally reducing cuts impacts the global cut structure.
- Every proof can be transformed into a proof with a minimally complex cut structure.
- The extracted clause set, is subsumed by the clause sets of the more complex cut structure.



Reduction can result in

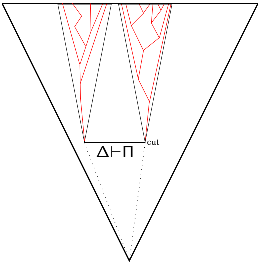


the following proof.

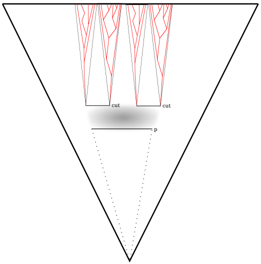
- Local elimination can result in a multiplication of the cuts
- Essentially, the cut-structure gets more redundant.

Clausal analysis: Reductive C.E. and Global Cut Structure

- Baaz and Leitsch, 2006 show how locally reducing cuts impacts the global cut structure.
- Every proof can be transformed into a proof with a minimally complex cut structure.
- The extracted clause set, is subsumed by the clause sets of the more complex cut structure.



Reduction can result in



the following proof.

- Local elimination can result in a multiplication of the cuts
- Essentially, the cut-structure gets more redundant.
- Redundancy $\equiv$ **structural simplicity**.

The Structurally Simplest Clause set

- What does this structural simplicity get you in the end?
- Consider the following:

The Structurally Simplest Clause set

- What does this structural simplicity get you in the end?
- Consider the following:

Clause Set

$$\begin{array}{r}
\vdash P(1) \\
\vdash P(2) \\
P(2) \vdash P(3) \\
P(2) \vdash P(4) \\
P(1), Q(2), Q(1) \vdash \\
P(1), R(2), R(1) \vdash \\
P(4), Q(3), Q(1) \vdash \\
P(4), R(3), R(1) \vdash \\
P(3), Q(3), Q(2) \vdash \\
P(3), R(3), R(2) \vdash \\
\vdash Q(1), R(1) \\
\vdash Q(2), R(2) \\
\vdash Q(3), R(3)
\end{array}$$

Top Clause Set

$$\begin{array}{r}
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3), P(4) \vdash \\
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3) \vdash P(4) \\
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(4) \vdash P(3) \\
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3) \vdash P(3), P(4) \\
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3), P(4) \vdash R(3) \\
Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3) \vdash R(3), P(4) \\
\vdots \\
R(3), P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3) \\
R(3) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3), P(4) \\
P(3), P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3) \\
P(3) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(4) \\
P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3) \\
\vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3), P(4)
\end{array}$$

The Structurally Simplest Clause set

- What does this structural simplicity get you in the end?
- Consider the following:

Clause Set

$\vdash P(1)$
$\vdash P(2)$
$P(2) \vdash P(3)$
$P(2) \vdash P(4)$
$P(1), Q(2), Q(1) \vdash$
$P(1), R(2), R(1) \vdash$
$P(4), Q(3), Q(1) \vdash$
$P(4), R(3), R(1) \vdash$
$P(3), Q(3), Q(2) \vdash$
$P(3), R(3), R(2) \vdash$
$\vdash Q(1), R(1)$
$\vdash Q(2), R(2)$
$\vdash Q(3), R(3)$

Top Clause Set

HAS 1024 CLAUSES!!!

$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3), P(4) \vdash$
$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3) \vdash P(4)$
$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(4) \vdash P(3)$
$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3) \vdash P(3), P(4)$
$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3), P(4) \vdash R(3)$
$Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3) \vdash R(3), P(4)$
$\vdots$
$R(3), P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3)$
$R(3) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), P(3), P(4)$
$P(3), P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3)$
$P(3) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(4)$
$P(4) \vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3)$
$\vdash Q(1), R(1), P(1), Q(2), R(2), P(2), Q(3), R(3), P(3), P(4)$

A Recursive Refutation

- Huge but easy to refute.
- Quantifier instantiation is still hard, i.e. avoided the "hard problem".

$$
\dfrac{\dfrac{\vdash \Delta, P(3), P(4) \vdash \qquad \vdash \Delta, P(3) \vdash P(4)}{\Delta, P(3) \vdash} \qquad \dfrac{\vdash \Delta, R(3), P(4) \vdash P(3) \qquad \vdash \Delta \vdash P(3), P(4)}{\Delta, R(3) \vdash P(3)}}{\Delta, R(3) \vdash}
$$

$$\vdots$$

- As one might imagine to refute $\Delta, R(3) \vdash$ we need a derivation using

$$\Delta \vdash R(3), P(3), P(4)$$
$$\Delta \vdash R(3), P(3)$$

- Similar to the construction of a semantic tree.

Global Cut-elimination and Proof Schema

- Recursive clausal analysis provides insight into the structure of proof schema.
- But it's too close for comfort to the infinite constructions of other formalisms
- Also, formula occurrence tracking is loss.
- In [Leitsch et al., 2017] a solution is provided preserving the occurrence tracking mechanism.
- Why not transform the Cut Structure into a Inductive Definition of an unsatisfiable NNF formula definition?
- Such formula are well studied for first-order theorem proving.

A Normal Form

$$
\dfrac{\dfrac{\color{red}{\phi_\alpha}}{\Gamma \vdash \Delta, \color{blue}{F_\alpha}} \qquad \dfrac{\dfrac{\dfrac{\color{red}{\phi_2}}{\Gamma \vdash \Delta, \color{blue}{F_2}} \qquad \dfrac{\dfrac{\color{red}{\phi_1}}{\Gamma \vdash \Delta, \color{blue}{F_1}} \qquad \dfrac{\dfrac{\Phi}{\color{blue}{F_1, \ldots, F_\alpha} \vdash}}{F_1, \ldots, F_\alpha, \Gamma \vdash \Delta}\,(w:l)}{\Gamma, F_2, \ldots, F_\alpha \vdash \Delta}\,(cut + c^*)}{\Gamma, F_3, \ldots, F_\alpha \vdash \Delta}\,(cut + c^*)}{\vdots}}{\Gamma \vdash \Delta}\,(cut + c^*)
$$

- The cut structure is turned into a recursively defined formula based on subformula occurance (**BLUE**).
- The schema itself is transformed into a schema with the cut structure as a formula in the consequent (**RED**).
- The formula is Σ_1 and unsatisfiable. The sequence $F_1, \ldots, F_\alpha$ contain the term tuples of a Schematic Herbrand Sequent.
  - Quantifier instantiations.

How to deal with F

- F is an inductive definition of an unsatisfiable Σ_1 formula indexed by a single free parameter, say n.
- We can instantiate n by an arbitrary natural number and get an instance, a first-order formula.
- Pick your favorite theorem prover and you can (possibly) get a refutation in no time.
- This is what was done in [Cerna & Leitsch, 2016] for the following theorem:

Eventually Constant Assertion (ECA)

Theorem

Let $n \in \mathbb{N}$ and $f_n : \mathbb{N} \to \{0, \cdots, n\}$ be a total monotonically decreasing function. Then there exists an $x \in \mathbb{N}$ such that for all $y \in \mathbb{N}$, where $x \leq y$, it is the case that $f(x) = f(y)$.

Proof.

Trivial, but we can make it ~~hard~~ weird by using the following cut formula

$$\exists x \forall y \left(((x \leq y) \to n + 1 = f(y)) \vee f(y) < n + 1\right)$$

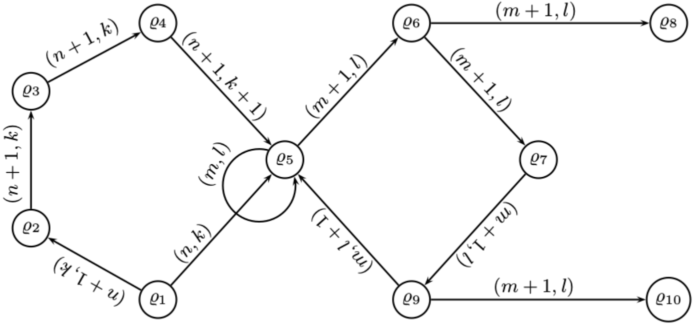
$\square$

ECA: Cut Structure Inductive Definition

$$
\begin{aligned}
\phi(0) \Longrightarrow & \forall x(x \leq x) \wedge \forall x(x \leq g(x)) \wedge \\
& \forall x(0 \neq f(x) \vee 0 \neq f(g(x))) \vee f(x) = f(g(x))) \wedge \\
& \forall x(f(x) \neq f(g(x))) \wedge \forall x(f(x) \not< 0) \wedge \forall x(f(h(x)) \not< 0) \\
\phi(s(n)) \Longrightarrow & \forall x(x \leq x) \wedge \forall x(x \leq g(x)) \wedge \forall x(f(x) \neq f(g(x))) \wedge \phi(n) \wedge \\
& \forall x(s(n) \neq f(x) \vee s(n) \neq f(g(x))) \vee f(x) = f(g(x))) \wedge \\
& \forall x \forall y(x \not\leq y \vee f(x) \not< s(n) \vee n = f(y) \vee f(y) < n) \\
\Omega(0) \Longrightarrow & \forall x(0 = f(x)) \wedge \forall x(f(x) \neq f(g(x))) \wedge \\
& \forall x(f(x) = f(g(x)) \vee 0 = f(x)) \\
\Omega(s(n)) \Longrightarrow & \forall x(f(x) < s(n) \vee s(n) = f(x)) \wedge \phi(s(n))
\end{aligned}
$$

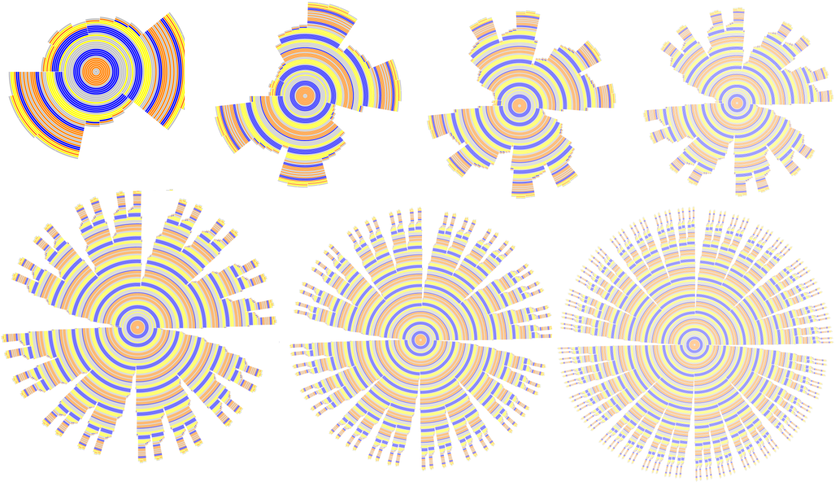
- Using Vampire and SPASS to produce the first 5 instances we found a refutation of the above inductive definition.

ECA: The Refutation



ECA: Recursion

ECA: Exponential Growth



Viper And The ECA

- Given Viper's efficacy concerning $(x+x)+x = x+(x+x)$ one would expect ECA to be much easier.
- Five days later... Viper was still working.
- ECA is easy from a first order point of view, Vampire and SPASS can handle instances as high as 10 in roughly a 1 second and higher given more time.
- What is hard about ECA? It is a simplification of the following statement

Theorem (Non-Injectivity Assertion (NIA))

Let $n \in \mathbb{N}$ and $f_n : \mathbb{N} \rightarrow \{0, \cdots, n\}$ be a total function. Then there exists an $x, y \in \mathbb{N}$, where $x < y$, s.t. $f(x) = f(y)$.

Proof.

Variant of the infinitary Pigeonhole Principle $\square$

NIA: Inductive Definition of Cut Structure

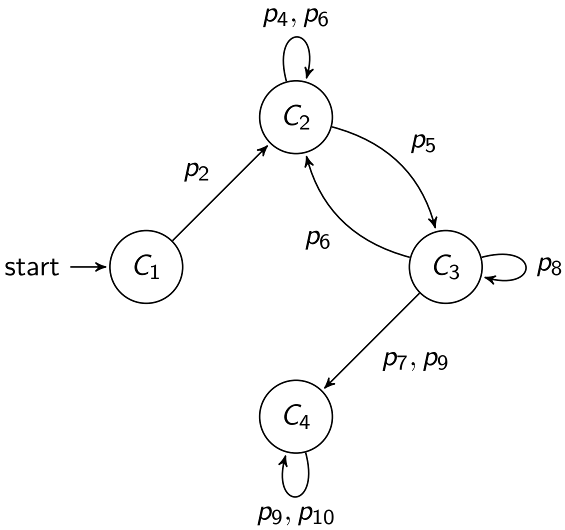
$$
\begin{aligned}
R(s(n)) &\Longrightarrow \forall x(\ x \leq x\ )\ \wedge\ T(s(n))\ \wedge\ \forall x(\ Q(s(n), x)\ ) \\
R(0) &\Longrightarrow \begin{array}{l} \forall x, y(\neg(s(y) \leq x\ \wedge\ f(y) = 0\ \wedge \\ f(x) = 0))\ \wedge\ \forall x(\ x \leq x\ ) \wedge \\ \forall x(f(x) = 0) \end{array}
\end{aligned}
$$

$$
\begin{aligned}
T(s(n)) &\Longrightarrow \begin{array}{l} \forall x, y, z(\ m(x, y) \leq z\ \Longrightarrow\ x \leq z\ )\ \wedge \\ \forall x, y, z(\ m(x, y) \leq z\ \Longrightarrow\ y \leq z\ )\ \wedge \\ \forall x, y(s(y) \not\leq x\ \vee\ f(y) \neq s(n) \vee f(x) \neq s(n)))\ \wedge \\ T(n) \end{array} \\
T(0) &\Longrightarrow \forall x, y(\neg(s(y) \leq x\ \wedge\ f(y) = 0\ \wedge\ f(x) = 0)) \\
Q(s(n), a) &\Longrightarrow f(a) = s(n) \vee Q(n, a) \\
Q(0, a) &\Longrightarrow f(a) = 0
\end{aligned}
$$

NIA: Refutation

- Constructing a formal refutation of $\forall x R(n, x) \vdash$ is quite difficult.
- Remember, the Infinitrary pigeonhole principle is $I\Sigma_2$.
- In recent work [D. M. Cerna, 2018], currently in review, introduced a formal system, which seems to be complete with respect to certain restrictions of schematic proof analysis (still an open question), which can formalize $\forall x R(n, x) \vdash$.
- The following diagram represents the linking dependencies of the formal proof.

Refutation Link Dependences



Other than Viper...

- The work [Leitsch *et al.* , 2017] is dependent on a superposition schematic prover introduced by [Aravantinos *et al.* , 2013].
- Viper takes a novel path towards invariant discovery by constructing tree grammars.
- Aravantinos *et al.* took a more traditional path (though in a novel way) of looking for loop invariants.
- So far our inductive definitions are hard for both, furthermore, the NIA schema is hard for first-order provers too.
- Maybe we simplified the NIA schema the wrong way.

A Way Out, 1-Strict Monotone Assertion (1-SMA)

Definition

a total function $f : A \to B$ is *k-strict monotone decreasing* if there exists a set of $A' \subset A$, whose cardinality is k, s.t. if $x \in A'$ then $f(x) = f(x+1)$, and if $x \in A \setminus A'$ then $f(x) < f(x+1)$.

- If $A = \mathbb{N}$ and $B = \{0, \cdots, n\}$, then we can pose the following theorem

Theorem

If $f : \mathbb{N} \to \{0, \cdots, n\}$ is a total monotone decreasing function, then f is at least 1-strict monotone decreasing.

Proof.

Can be proven using a sequence of Δ_2 cuts. $\square$

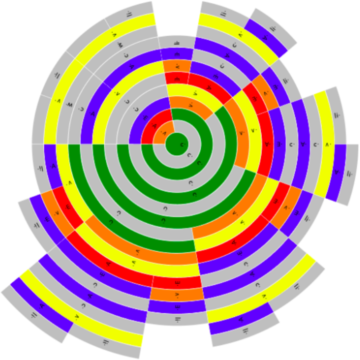
1-SMA: Inductive Definition of the Cut Structure

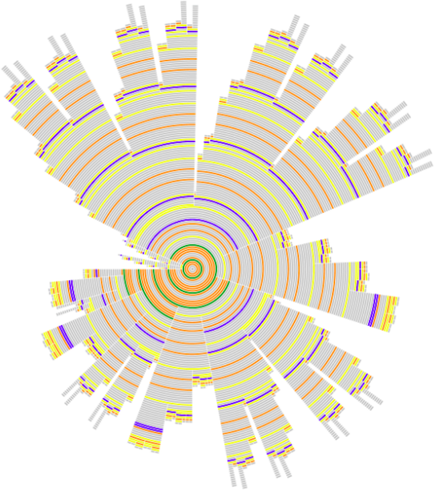
$$
\begin{aligned}
Top(0) &= Next(0) \wedge (0 = f(\mathbf{0}) \vee 0 = f(\mathbf{S}(\mathbf{0}))) \\
Top(n+1) &= \forall x((n+1) = f(\mathbf{S}(x)) \vee f(x) < (n+1)) \wedge \\
&\quad \forall x((n+1) = f(x) \vee f(x) < (n+1)) \wedge Next(n+1) \\
Next(0) &= (\neg f(\mathbf{0}) < 0) \wedge \forall x((\neg 0 = f(x)) \vee (\neg 0 = f(\mathbf{S}(x)))) \\
Next(n+1) &= \forall x((\neg(n+1) = f(x)) \vee (\neg(n+1) = f(\mathbf{S}(x)))) \wedge \\
&\quad \forall x((\neg f(x) < (n+1)) \vee n = f(x) \vee f(x) < n) \wedge \\
&\quad \forall x((\neg f(\mathbf{S}(x)) < (n+1)) \vee n = f(\mathbf{S}(x)) \vee f(x) < n) \\
&\quad \wedge Next(n)
\end{aligned}
$$

1-SMA: Important Properties

- Unlike the previous examples $\forall n Top(n) \vdash$ has a single proof (modulo structural changes)!
- This was shown in [D. M. Cerna, 2018]
- Viper can prove this statement in roughly 5 hours
- The superposition prover [Aravantinos *et al.* , 2013] cannot on theoretic grounds.
- Every quantifier of $Top(n)$ needs to be instantiated by a number of terms dependent on n, a simple diagonalization argument shows that we leave LOOP_1 programs, i.e. their loop discovery mechanism.

1-SMA: Proof & Refutation





1-SMA: Hardness

- To refute ECA's inductive definition one needs the following instances of $\forall x(f(x) < s(n) \vee s(n) = f(x))$

$$f(0) < s(n) \vee s(n) = f(0) \quad f(g(0)) < s(n) \vee s(n) = f(g(0))$$

- To refute 1-SMA's inductive definition for instance five one needs six instances of $\forall x(f(x) < s^5(0) \vee s^5(0) = f(x))$

$$\begin{array}{rr} f(0) < s^5(0) \vee s^5(0) = f(0) & f(\mathbf{S}(0)) < s^5(0) \vee s^5(0) = f(\mathbf{S}(0)) \\ f(\mathbf{S}^2(0)) < s^5(0) \vee s^5(0) = f(\mathbf{S}^2(0)) & f(\mathbf{S}^3(0)) < s^5(0) \vee s^5(0) = f(\mathbf{S}^3(0)) \\ f(\mathbf{S}^4(0)) < s^5(0) \vee s^5(0) = f(\mathbf{S}^4(0)) & f(\mathbf{S}^5(0)) < s^5(0) \vee s^5(0) = f(\mathbf{S}^5(0)) \end{array}$$

- Like the Infinitary Pigeonhole Principle, 1-SMA captures some fundamental combinatorial complexity.
- We can exploit this and define a complexity based on it. But before doing so, we can ask if 1-SMA is alone?

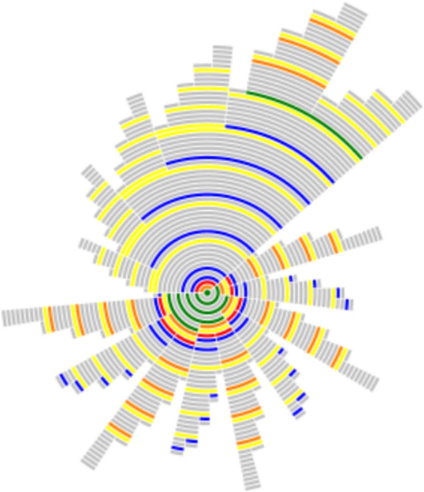
Scalar 1-SMA & Matrix 1-SMA

- By manipulating the proof in such a way that the clause $\forall x((\neg(n+1) = f(x)) \vee (\neg(n+1) = f(\mathbf{S}(x))))$ changes we can get stronger or weaker statements.
- Scalar 1-SMA contains the clause $\forall x((\neg(n+1) = f(x)))$ instead. Every quantifier is instantiated by a single term.
- Matrix 1-SMA contains the clause

$$\forall x((\neg(n+1) = f(x)) \vee \cdots \vee (\neg(n+1) = f(\mathbf{S}^k(x))))$$

  where k is a second free parameter. Every quantifier is instantiated by $n \cdot k$ terms.
- There is no need to stop here, we can make an 1-SMA statement such that every quantifier is instantiated by $\prod_{i=0}^{m} n_i$ terms where n_i are the free parameters.
- We conjecture (perhaps unsurprisingly) that the limit of the 1-SMA hierarchy is precisely the NIA schema which requires Every quantifier to be instantiated by at least n^n terms.

Matrix 1-SMA: proof



Complexity Measure

- To define the measure we need a way to relate clauses derived from different instances of an NNF formula's inductive definition.
- We want to define complexity in terms of the *number of instantiations* of related clauses necessary for refuting the instance of the inductive definition.
- We say an inductive definition of a NNF formula F over parameters $n_0, \cdots, n_m$ is $O(f(n_0, \cdots, n_m))$-unsat if all clauses and there relatives require at most $O(f(t_0, \cdots, t_m))$ quantifier instantiation when refuting $C\{n_0 \leftarrow t_0, \cdots, n_m \leftarrow t_m\}$
- scalar 1-SMA is $O(1)$-unsat, 1-SMA is $O(n)$-unsat, matrix 1-SMA $O(n \cdot k)$-unsat, ECA is $O(2^n)$-unsat and NIA is $O(n^n)$-unsat.
- But, ECA is some what special.

Recursively Unsatisfiable

- If we only focus on the clauses of ECA's inductive definition indexed by $n, \cdots, n-k$ for some constant k, notice that all these clauses are instantiated a constant number of times.
- In some sense ECA is recursive O(1)-unsat, i.e. it is surprisingly easy when you find out that an exponential function is needed for term construction.
- This observation points towards a secondary hierarchy dependent on recursive constructions in some sense simpler than the non-recursive hierarchy.
- This secondary hierarchy points towards problems which might be susceptible to methods of automated theorem proving.

Conclusions & future work

- Essentially loop discovery methods can easily get stuck below $O(n)$-unsat because they would need to discover a loop in a loop!
- This complexity measure discussed above seems to capture Vipers abilities quite well.
- The method is limited to tree grammars with an exponential language and Viper suffered while attempting to prove ECA.
- Interesting question is how would a tree grammar based prover handle classes $O(n^k)$-unsat for a constant k?
- Future work is for the most part development and further formalization of the concepts.

Thank you for your time.