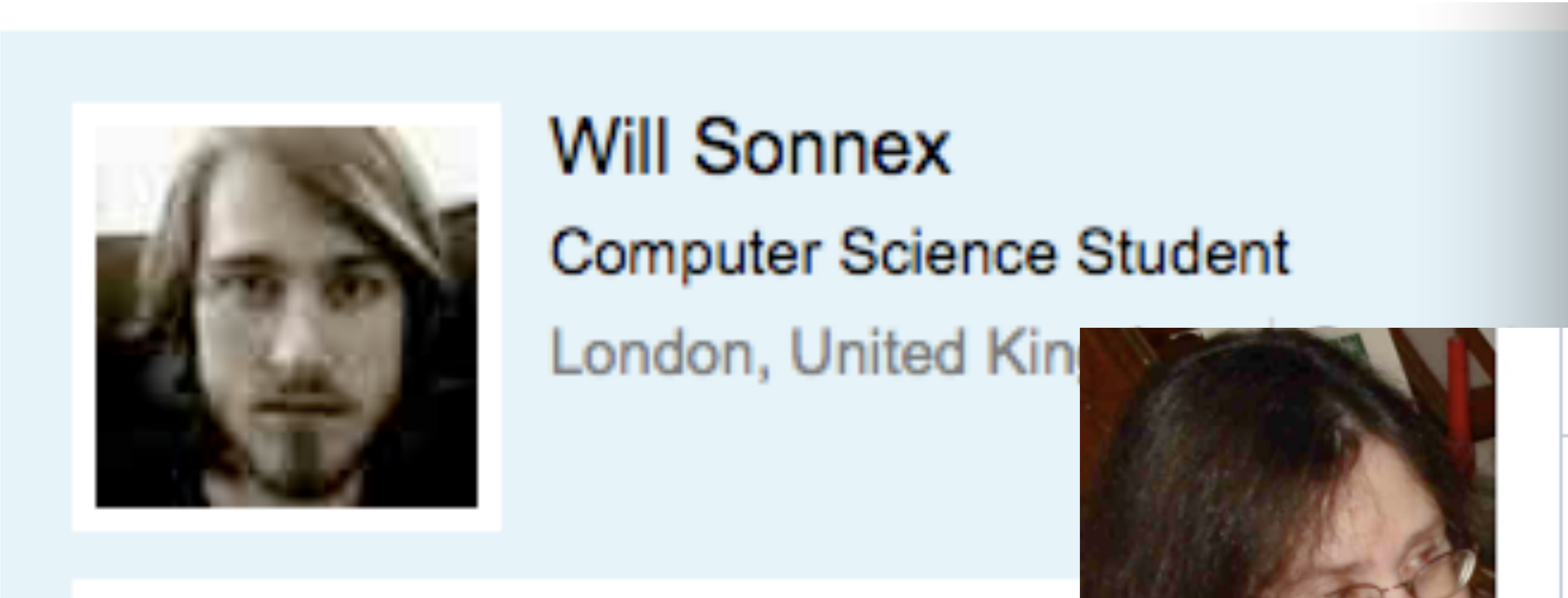
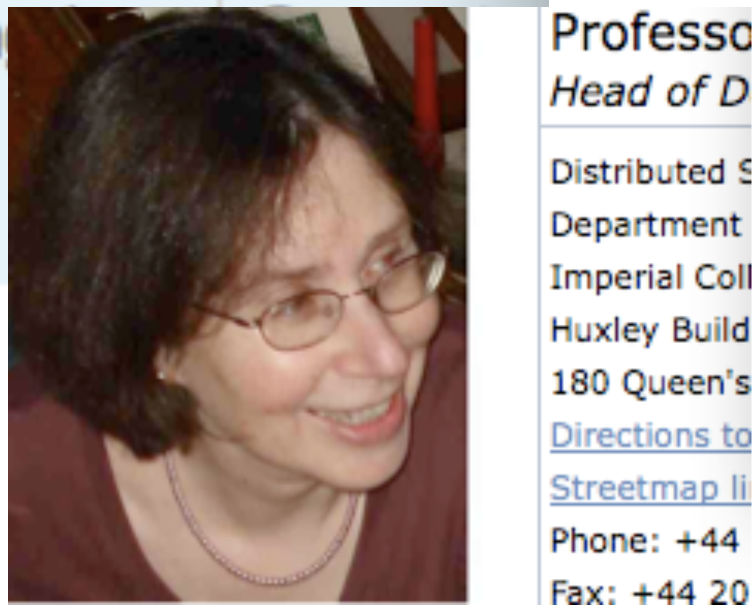
Zeno

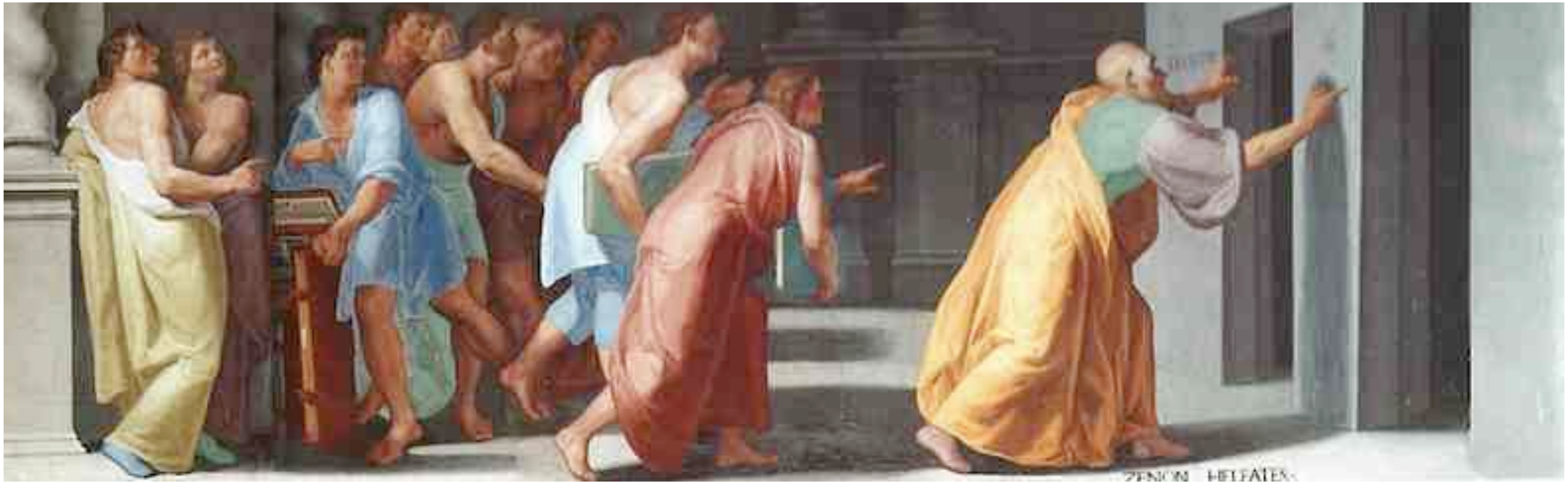
an automated theorem prover for functional programs





Will Sonnex,
Sophia Drossopoulou and Susan Eisenbach

Imperial College London



- [Zeno of Elea](#) (c.490–c.430 BC), philosopher, follower of Parmenides, famed for his *paradoxes*.
- [Zeno of Citium](#) (333 BC - 264 BC), founder of the Stoic school of philosophy
- [Zeno of Tarsus](#) (200s BC), Stoic philosopher
- [Zeno of Sidon](#) (1st century BC), Epicurean philosopher
- Zeno at http://www.haskell.org/haskellwiki/Zeno

Zeno

- Proves equality over Haskell-like expressions of the form
  $\mathtt{E_1=E_2, \ldots, E_{2n+1}=E_{2n+2} \quad ==> \ E = E'}$
  where `E` may mention recursively defined functions
- Variables are implicitly universally quantified; no support for existentials
- Booleans are encoded through the `Bool` data type.
- Zeno can prove properties like
  - `rev (rev xs) = xs`
  - `order (order xs) = order xs`
  - `mult x (succ 0) = x`
- From a benchmark suite suggested by Isaplanner, Zeno can prove more properties than Isaplanner and ACL2s
- Zeno can discover necessary auxiliary lemmas, but cannot use theories.

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

Example - Haskell

```
data Nat = Zero | Succ Nat

(<=) :: Nat -> Nat -> Bool
Zero <= _ = True
Succ x <= Zero = False
Succ x <= Succ y = x <= y

srtd :: [Nat] -> Bool
srtd [] = True
srtd [x] = True
srtd (x:y:zs) = (x <= y) && srtd (y:zs)

ordr :: [Nat] -> [Nat]
ordr [] = []
ordr (x:xs) = ins x (ordr xs)

ins :: Nat -> [Nat] -> [Nat]
ins n [] = [n]
ins n (x:xs) | n<=x = n:x:xs | otherwise x:(ins n xs)
```

Example - Haskell and HC

```
data Nat = Zero | Succ Nat

(<=) :: Nat -> Nat -> Bool
Zero <= _ = True
Succ x <= Zero = False
Succ X <= Succ y = x <= y
```

```
data Nat = Zero | Succ Nat

letrec (<=) = λ x. λ y. case x of
     { Zero -> True;
       Succ x′ -> case y of
             { Zero -> False;
               Succ y′ -> x′ <= y′ }
```

Example - Haskell and HC

```
srtd :: [Nat] -> Bool
srtd [] = True
srtd [x] = True
srtd (x:y:ys) = (x <= y) && srtd (y:ys)

letrec srtd = λ ns. case ns of
     { [] -> True;
       x:xs -> case xs of
            { [] -> True;
              y:ys -> case x<=y of
                   { True -> srtd (y:ys);
                     False -> False } }  }
```

Example Haskell and HC

```
ordr :: [Nat] -> [Nat]
ordr [] = []
ordr (x:xs) = ins x (ordr xs)
```

```
letrec ordr = λ ns. case ns of
     { [] -> [];
       x:xs -> ins x (ordr xs)} }  }
```

Example Haskell and HC

```
ins :: Nat -> [Nat] -> [Nat]
ins n [] = [n]
ins n (x:xs) | n<=x = n:x:xs | otherwise x:(ins n xs)
```

```
letrec ins = λ n. λ ns. case ns of
     { [] -> n:[];
       x:xs -> case n<=x of
                 { True -> n:x:xs;
                   False -> x:(ins n xs)} }  }
```

Example in HC

…

```
letrec srtd = λ ns. case ns of
     { [] -> True;
       x:xs -> case xs of
            { [] -> True;
              y:ys -> case x<=y of
                   { True -> srtd (y:ys);
                     False -> False } }  }

letrec ordr = λ ns. case ns of
     { [] -> [];
       x:xs -> ins n (ordr xs)} }  }

letrec ins = λ n. λ ns. case ns of
     { [] -> n:[];
       x:xs -> case n<=x of
                   { True -> n:x:xs;
                     False -> x:(ins n xs)} }  }
```

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

Zeno supports sequent-style proof rules.
It applies these rules backwards, possibly trying several.
These rules are:

- CON - contradiction
- EQL – substitute equals for equals
- IND - induction
- EXP – expansion
- GEN – generalization
- CASE – case analysis
- Modus Ponens

So, we want to prove

```
srtd (ordr is)
```

We will first outline part of the proof, and then we will show the rules for the individual steps.

So, we want to prove

`srtd (ordr is)`

We will first outline part of the proof, and then we will show the rules for the individual steps.

Proving `srtd (ordr is)`

$$\frac{\texttt{????}}{\texttt{srtd (ordr is)}}\;\texttt{???}$$

Proving `srtd (ordr is)`

$$\dfrac{\dfrac{???}{\texttt{srtd (ord [])}}\;??? \qquad \dfrac{????}{\texttt{srtd (ord js) => srtd (ord j:js)}}\;????}{\texttt{srtd (ordr is)}}\;\texttt{IND}$$

Proving `srtd (ordr is)`

```
                                                    ????
                          ---------------------------------------------------------------- ???
                              srtd (ord js) => srtd (ins j (ord js))
         ???
 ------------------------ ???  ------------------------------------------------------------ EXP
 srtd (ord [])                 srtd (ord js) => srtd (ord j:js)
 ----------------------------------------------------------------------------------------- IND
                                srtd (ordr is)
```

Proving `srtd (ordr is)`

```
   ------------------------------------------------------------------ ???
                       srtd (ks) => srtd (ins i ks)
                  ---------------------------------------------------- GEN
                  srtd (ord js) => srtd (ins j (ord js))
.    ???
--------------- ???   -------------------------------------------- EXP
srtd (ord [])         srtd (ord js) => srtd (ord j:js)
 .
 ------------------------------------------------------------------ IND
                       srtd (ordr is)
```

Proving `srtd (ordr is)`

Note:
Zeno discovered the auxiliary lemma

`srtd ks => srtd (ins j ks)`

```
 ___________________________________________________________________________???
                           srtd (ks) => srtd (ins j ks)
                     ______________________________________________________GEN
                         srtd (ord js) => srtd (ins j (ord js))
.     ???
___________________???       ______________________________________________EXP
srtd (ord [])                srtd (ord js) => srtd (ord j:js)
 .
___________________________________________________________________________IND
                              srtd (ordr is)
```

Proving `srtd (ordr is)`

```
                                              ???
                      ---------------------------------------------------------- ???
        ????          srtd (ms)=> srtd (ins i (ord ms)
-------------------- ???                        =>
srtd ([])              srtd (m:ms)=> srtd (ins i (ord m:ms)
  => srtd (ins i [])
------------------------------------------------------------------------------ IND
                     srtd (ks) => srtd (ins i ks)
               ---------------------------------------------------------- GEN
                  srtd (ord js) => srtd (ins j (ord js))
.    ???         ------------------------------------------------------- EXP
------------- ???  srtd (ord js) => srtd (ord j:js)
srtd (ord [])
.
------------------------------------------------------------------------------ IND
                          srtd (ordr is)
```

So, we want to prove

```
srtd (ordr is)
```

We will first outline part of the proof, and then we will show the rules for the individual steps.

Proving `srtd (ordr is)` – the IND step

$$\dfrac{\texttt{srtd (ord [])} \qquad \texttt{srtd (ord js) => srtd (ord j:js)}}{\texttt{srtd (ordr is)}}\ \texttt{IND}$$

Proving `srtd (ordr is)` – the IND step

$$\frac{\begin{array}{l} x \text{ has type } \mathtt{T} \\ \text{for each } \mathtt{K} \in \text{Constrs}(\mathtt{T}) . \\ \quad \vdash \phi[x{:=}z_1], \ldots \phi[x{:=}z_m] \Rightarrow \phi[x{:=}\mathtt{K}\ y_1\ldots y_n] \\ \quad \text{where } \mathtt{K}\ y_1\ldots y_n \text{ has type } \mathtt{T}, \\ \quad y_1\ldots y_n \text{ are fresh variables,} \\ \quad z_1\ldots z_m \text{ are those variables from } y_1\ldots y_n \text{ with type } \mathtt{T} \end{array}}{\vdash \phi}\ \mathtt{IND}$$

$$\frac{\mathtt{srtd\ (ord\ [])} \qquad \mathtt{srtd\ (ord\ js)\ \Rightarrow\ srtd\ (ord\ j{:}js)}}{\mathtt{srtd\ (ordr\ is)}}\ \mathtt{IND}$$

Proving `srtd (ordr is)` – the EXP step

$$\dfrac{\dfrac{???}{\texttt{srtd (ord [])}}\,{???} \qquad \dfrac{\texttt{srtd (ord js) => srtd (ins j (ord js))}}{\texttt{srtd (ord js) => srtd (ord j:js)}}\,\texttt{EXP}}{\texttt{srtd (ordr is)}}\,\texttt{IND}$$

Proving `srtd (ordr is)` – the EXP step

`E` evaluates to `E′`

$$\dfrac{\vdash\ \phi\,[\texttt{E:=E'}]}{\vdash\ \phi}\ \texttt{EXP}$$

$$\dfrac{\cdot\qquad \dfrac{\cdot\qquad ???}{\texttt{srtd (ord [])}}\ ??? \qquad\qquad \dfrac{\texttt{srtd (ord js) => srtd (ins j (ord js))}}{\texttt{srtd (ord js) => srtd (ord j:js)}}\ \texttt{EXP}}{\texttt{srtd (ordr is)}}\ \texttt{IND}$$

Proving `srtd (ordr is)` – the GEN step

$$\dfrac{\begin{array}{l}\texttt{x}\ \text{is fresh, and has type}\ \texttt{T}\\ \texttt{E}\ \text{has type}\ \texttt{T}\\ \vdash\ \phi\,\texttt{[E:=x]}\end{array}}{\vdash\ \phi}\ \texttt{GEN}$$

$$\dfrac{\dfrac{\cdot\quad ???}{\texttt{srtd (ord [])}}\ ???\qquad \dfrac{\dfrac{\texttt{srtd (ks) => srtd (ins i ks)}}{\texttt{srtd (ord js) => srtd (ins j (ord js))}}\ \texttt{GEN}}{\texttt{srtd (ord js) => srtd (ord j:js)}}\ \texttt{EXP}}{\texttt{srtd (ordr is)}}\ \texttt{IND}$$

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

Zeno is sound

- Zeno’s proof rules correspond to sequent calculus
- Zeno emits Isabelle proofs, which it checks through Isabelle

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

… using the Isaplanner test suite

Theorem prover	Percentage proven	Identifiers of unproven properties
Dafny (Z3 and IND)	53.5%	45-85
Isaplanner	55%	47-85
ACL2s – coded types	87%	47, 50, 54, 56, 72, 73, 74, 81, 83, 84, 85
Zeno	96%	72, 74, 85

What gives Zeno its performance?

- Trimming the search space, ie heuristics which reduce applicability of the rules.
  These heuristics can be understood as further conditions on the rules.

This Talk

- Example Zeno code
- The proof steps – by example
- Is Zeno sound?
- Zeno performance
- Trimming the search space

Zeno’s trimming heuristics

- Prioritize CON and EQL steps.
- Search for counterexample.
- Critical expressions.
- Critical paths.
- ….

Zeno's trimming heuristics

- Prioritize CON and EQL steps.
  - CON and EQL "close" proof braches;

$$\frac{\texttt{K},\ \texttt{K}' \text{ are constructors} \qquad \texttt{K} =/= \texttt{K}'}{\vdash (\texttt{K}\ \ \texttt{E}_1 \ldots \texttt{E}_n) = (\texttt{K}'\ \ \texttt{E}'_1 \ldots \texttt{E}'_n) \Rightarrow \phi}\ \texttt{CON}$$

  therefore it pays to apply them ASAP

- Search for counterexample.
- Critical expressions.
- Critical paths.
- ….

Zeno's trimming heuristics

- Prioritize CON and EQL steps.
- Search for counterexample.
  - After generation of new proof goal (eg through GEN), create examples (using critical expressions/paths) to quickly test the new proof goal, and discard the branch if counterexample found.
- Critical expressions.
- Critical paths.
- ….

Zeno's trimming heuristics

- Prioritize CON and EQL steps.
- Search for counterexample after GEN steps.
- Critical expressions.
  - Aim to steer the proof search so that EXP steps become applicable (ie function definitions may be applied).
- Critical paths.
- ….

Zeno’s trimming heuristics

- Prioritize CON and EQL steps.
- Search for counterexample after GEN steps.
- Critical expressions.
  - Aim to steer the proof search so that EXP steps become applicable (ie function definitions may be applied).
    This is in contrast with rippling (Isaplanner), which, instead, tries to make the inductive hypothesis applicable.
- Critical paths.
- ….

Critical expressions - example

???

srtd (ordr is) ???

Critical expressions - example

At this point, many steps could be considered:

- `IND` on `is`
- `CASE` on `ord is`
- `CASE` on `srtd(ord is)`
- `IND` on `ord is`
- `CASE` on `first (is)`
- …

```
---------------------------------------------------------------???
                    srtd (ordr is)
```

Critical expressions - example

Similarly, at this point, the following steps could be considered:

- `IND` on `js`
- `IND` on `j`
- `CASE` on `js`
- `CASE` on `j`
- `CASE` on `ord js`
- `CASE` on `ord j:js`
- …

$$\dfrac{\dots \qquad \dfrac{\texttt{????}}{\texttt{srtd (ord js) => srtd (ord j:js)}}\;\texttt{????}}{\texttt{srtd (ordr is)}}\;\texttt{IND}$$

Critical expressions - definition

We want to consider only those expressions which are critical for the execution of the term, ie those expressions where execution of a term will get stuck.

$$
\mathrm{Crits}(\mathtt{E}) = \begin{cases} \mathtt{E} & \text{if } \mathtt{E} \text{ is normal} \\ \mathrm{Crits}(\mathtt{E'}) & \text{if } \mathtt{E} \texttt{->*} \ \textbf{case}\ \mathtt{E'}\ \textbf{of}\ \ldots,\ \mathtt{E'} \in \mathtt{E} \\ \mathtt{E'} & \text{if } \mathtt{E} \texttt{->*} \ \textbf{case}\ \mathtt{E'}\ \textbf{of}\ \ldots,\ \mathtt{E'} \notin \mathtt{E} \end{cases}
$$

$\mathtt{E}$ is *normal* if it cannot be further re-written

Critical expressions - examples

$$\text{Crits}(\mathtt{E}) = \begin{cases} \mathtt{E} & \text{if } \mathtt{E} \text{ is normal} \\ \text{Crits}(\mathtt{E'}) & \text{if } \mathtt{E} \text{->* } \mathbf{case}\ \mathtt{E'}\ \mathbf{of}\ \ldots,\ \mathtt{E'} \in \mathtt{E} \\ \mathtt{E'} & \text{if } \mathtt{E} \text{->* } \mathbf{case}\ \mathtt{E'}\ \mathbf{of}\ \ldots,\ \mathtt{E'} \notin \mathtt{E} \end{cases}$$

```
ord(is) ->* case is of { [] -> …; x:xs -> … }
srtd(ord(is)) ->* case ord(is) of
                      { True -> …; False -> … }
```

Crits(`ord(is) ) = is`

Crits(`srtd(ord (is)) ) = is`

Using Critical Expressions - IND

Without Crits, following steps possible

- `IND` on `is`
- `CASE` on `ord is`
- `CASE` on `srtd(ord is)`
- `IND` on `ord is`
- `CASE` on `first(is)`
- …

….

`srtd (ordr is)`

Using Critical Expressions - IND

reduces the proof search space

$$\frac{\begin{array}{l} \texttt{x} \text{ has type } \texttt{T}, \ \texttt{x} \in \text{Crits}(\phi) \\ \text{for each } \texttt{K} \in \text{Constrs}(\texttt{T}) .\ \vdash \ \phi[\texttt{x:=z}_1], \ldots \phi[\texttt{x:=z}_m] \Rightarrow \phi[\texttt{x:=K y}_1 \ldots \texttt{y}_n] \\ \qquad \text{where} \ldots \end{array}}{\vdash \ \phi} \texttt{IND}$$

$$\frac{\ldots .}{\texttt{srtd (ordr is)}}$$

Using Critical Expressions - IND

reduces the proof search space

Crits(`srtd(ordr is)`) = { `is` }

With Crits, several steps *not* applicable

- `IND` on `is`
- ~~`CASE` on `ord is`~~
- ~~`CASE` on `srtd(ord is)`~~
- ~~`IND` on `ord is`~~
- ~~`CASE` on `first (is)`~~
- …

….

`srtd (ordr is)`

Using Critical Expressions - IND

reduces the proof search space

$$\frac{\texttt{x}\ \text{has type}\ \texttt{T},\quad \texttt{x} \in \text{Crits}(\phi) \qquad \text{for each}\ \texttt{K} \in \text{Constrs}(\texttt{T})\ .\ \vdash\ \phi[\texttt{x}:=\texttt{z}_1], \ldots \phi[\texttt{x}:=\texttt{z}_m] \Rightarrow \phi[\texttt{x}:=\texttt{K}\ \texttt{y}_1 \ldots \texttt{y}_n]\ \ \text{where} \ldots}{\vdash\ \phi}\ \texttt{IND}$$

Crits(`srtd (ordr is)`) = { `is` }

With Crits, several steps not applicable

- `IND` on `is`
- ~~`CASE` on `ord is`~~
- ~~`CASE` on `srtd(ord is)`~~
- ~~`IND` on `ord is`~~
- ~~`CASE` on `first (is)`~~

$$\frac{\texttt{srtd (ord [])} \qquad \texttt{srtd (ord js) => srtd (ord j:js)}}{\texttt{srtd (ordr is)}}\ \texttt{IND}$$

Critical expressions need not be subterms

$$\text{Crits}(\mathtt{E}) = \begin{cases} \mathtt{E} & \text{if } \mathtt{E} \text{ is normal} \\ \text{Crits}(\mathtt{E'}) & \text{if } \mathtt{E} \text{->* } \textbf{case } \mathtt{E'} \textbf{ of } \ldots, \ \mathtt{E'} \in \mathtt{E} \\ \mathtt{E'} & \text{if } \mathtt{E} \text{->* } \textbf{case } \mathtt{E'} \textbf{ of } \ldots, \ \mathtt{E'} \notin \mathtt{E} \end{cases}$$

```
ins i (j:js) ->* case i <= j of
                   { True -> …; False -> …}
srtd(ins i (j:js)) ->*
                 case (ins i (j:js) ) of
                   { [] -> …; y:ys -> …}
```

Crits(`ins i (j:js) )  = i<=j`

Crits(`srtd(ins i (j:js))) = i<=j`

Use of critical Expressions which are not subterms

$$\frac{}{\texttt{srtd (j:js)} \quad \Rightarrow \quad \texttt{srtd(ins i (j:js))}} \; ?????$$

Use of critical Expressions which are not subterms

Critical expressions which are not subterms are used for case analysis

```
Crits (srtd(ins i (j:js)) ) =    i<=j
```

```
                                                              CASE
--------------------------------------------------------------
        srtd (j:js)    =>    srtd( ins i (j:js) )
```

Use of critical Expressions which are not subterms

Critical expressions which are not subterms are used for case analysis

Crits(`srtd(ins i (j:js))`) = `i<=j`

```
------------------------------------??      ------------------------------------??
 i<=j = True =>                              i<=j = False =>
    srtd (j:js)    =>                           srtd (j:js)    =>
    srtd( ins i (j:js) )                        srtd( ins i (j:js) )
   ----------------------------------------------------------------------------CASE
             srtd (j:js)    =>    srtd( ins i (j:js) )
```

However, consider …

$$\dfrac{\dfrac{\ldots}{\texttt{srtd (ord [])}} \qquad \dfrac{???}{\texttt{srtd (ord js) => srtd (ord j:js)}}\,{\color{green}\texttt{???}}}{\texttt{srtd (ordr is)}}\,{\color{green}\texttt{IND}}$$

However, consider …

Crits(`srtd(ordr js)`) = Crits(`srtd(ordr js)`) = `js`

$$\dfrac{\dfrac{\ldots}{\texttt{srtd (ord [])}} \qquad \dfrac{\texttt{???}}{\texttt{srtd (ord js) => srtd (ord j:js)}}\;\texttt{???}}{\texttt{srtd (ordr is)}}\;\texttt{IND}$$

However, consider …

Crits(`srtd(ordr js)`) = Crits(`srtd(ordr js)`) = `js`

Should we apply induction on `js`?

```
       …                                          ???
-----------------       --------------------------------------------------???
srtd (ord [])           srtd (ord js) => srtd (ord j:js)
 .--------------------------------------------------------------------------IND
                          srtd (ordr is)
```

However, consider …

Crits (`srtd(ordr js)`) = Crits (`srtd(ordr js)`) = `i<=j`

Should we apply induction on `js`? Again induction? ☹

```
        ...                              ???
_______________      __________________________________________ ???
srtd (ord [])        srtd (ord js)  =>  srtd (ord j:js)
 .___________________________________________________________ IND
                      srtd (ordr is)
```

Zeno's trimming heuristics

- Prioritize CON and EQL steps.
- Search for counterexample after GEN steps.
- Critical expressions.
- Critical paths.
- ....

Critical Pairs

We enhance our approach so that

P1 Case statements are labeled.

Critical Pairs

We enhance our approach so that

P1 Case statements are labeled.

P2 Critical expressions are decorated with paths of labels; these describe the “intention” of the expression, ie the case statements that this expression would represent.

Critical Pairs

We enhance our approach so that

- P1 Case statements are labeled.
- P2 Critical expressions are decorated with paths of labels; these describe the “intention” of the expression, ie the case statements that this expression would represent.
- P3 Variables are decorated with paths of labels; these describe the “history” of these variables, ie case statements that these variables have represented.

Critical Pairs

We enhance our approach so that

P1 Case statements are labeled.

P2 Critical expressions are decorated with paths of labels; these describe the "intention" of the expression, ie the case statements that this expression would represent.

P3 Variables are decorated with paths of labels; these describe the "history" of these variables, ie case statements that these variables have represented.

P4 Induction avoids revisiting (parts of) an already visited path. Therefore, induction not applicable when history of critical expression "covers" its intention. Similar for case analysis, generalization, etc.

P1: Labelling Case Statments

…

```
letrec srtd = λ ns. case^s1 ns of
      { [] -> True;
        x:xs -> case^s2 xs of
            { [] -> True;
              y:ys -> case^s3 x<=y of
                 { True -> srtd (y:ys);
                   False -> False } }  }

letrec ordr = λ ns. case^o1 ns of
      { [] -> [];
        x:xs -> ins n (ordr xs)} }  }

letrec ins = λ n. λ ns. case^i1 ns of
      { [] -> n:[];
        x:xs -> case^i2 n<=x of
                 { True -> n:x:xs;
                   False -> x:(ins n xs)} }  }
```

P2: Decorating critical expressions

Critical expressions record their intention: which cases they will consider, if chosen

$$\text{Crits}(\mathtt{E}) = \begin{cases} \mathtt{E}, \mathtt{[]} & \text{if } \mathtt{E} \text{ is normal} \\ \mathtt{E''}, \mathtt{l:p} & \text{if } \mathtt{E}\text{->*}\ \mathbf{case}^{\mathtt{l}}\ \mathtt{E'}\ \mathbf{of}\ \ldots,\ \ \mathtt{E'} \in \mathtt{E} \\ & \text{Crits}(\mathtt{E'}) = \mathtt{E''}, \mathtt{p} \\ \mathtt{E'}, \mathtt{l} & \text{if } \mathtt{E}\text{->*}\ \mathbf{case}^{\mathtt{l}}\ \mathtt{E'}\ \mathbf{of}\ \ldots,\ \ \mathtt{E'} \notin \mathtt{E} \end{cases}$$

P2: Decorating critical expressions - examples

$$
\text{Crits}(\mathtt{E}) = \begin{cases} \mathtt{E}, \mathtt{[]} & \text{if } \mathtt{E} \text{ is normal} \\ \mathtt{E''}, \mathtt{l.p} & \text{if } \mathtt{E} \to^* \mathtt{case}^{\mathtt{l}}\ \mathtt{E'}\ \mathtt{of}\ \ldots,\ \mathtt{E'} \in \mathtt{E} \\ & \text{Crits}(\mathtt{E'}) = \mathtt{E''}, \mathtt{p} \\ \mathtt{E'}, \mathtt{l} & \text{if } \mathtt{E} \to^* \mathtt{case}^{\mathtt{l}}\ \mathtt{E'}\ \mathtt{of}\ \ldots,\ \mathtt{E'} \notin \mathtt{E} \end{cases}
$$

```
ord(is[]) ->* caseo1 is of { [] -> …; x:xs -> … }
srtd(ord(is[])) ->* cases1 ord(is) of
                        { True -> …; False -> … }
```

Crits(ord(is[])) = is[],o1.[]

Crits(srtd(ord(is[]))) = is[],s1.o1.[]

P2: Decorating critical expressions - examples

$$
\text{Crits}(E) = \begin{cases} E, [\,] & \text{if } E \text{ is normal} \\ E'', l.p & \text{if } E \to^* \textbf{case}^{l}\ E'\ \textbf{of}\ \ldots,\ E' \in E \\ & \text{Crits}(E') = E'', p \\ E', l & \text{if } E \to^* \textbf{case}^{l}\ E'\ \textbf{of}\ \ldots,\ E' \notin E \end{cases}
$$

```
ord(is[]) ->* case^o1 is of { [] -> …; x:xs -> … }
srtd(ord(is[])) ->* case^s1 ord(is) of
                        { True -> …; False -> … }
```

Crits(`ord(is[])`) = `is[],o1.[]`

Crits(`srtd(ord(is[]))`) = `is[],s1.o1.[]`

When `is[]` is taken for `srtd(ord(is[]))`, it intends to cover the cases **`s1.o1`**

P3: Decorating variables

•

$$\frac{}{\texttt{srtd (ordr is}^{\texttt{[]}}\texttt{)}}\ \texttt{IND}$$

P4: Induction – only when intention is not "covered" by history

$$\frac{\begin{array}{l} x\ \text{has type}\ \mathtt{T},\ \ \mathtt{x}^{p},\ p' \in \text{Crits}(\phi) \\ \ldots\mathtt{x}^{p''}\ldots,\ p''' \in \text{Crits}(\phi)\ \ \text{implies}\ \ p'\ \text{not a subpath of}\ p'' \\ \text{for each}\ \mathtt{K} \in \text{Constrs}(\mathtt{T}).\ \vdash\ \phi[\mathtt{x}:=\mathtt{z}_1], \ldots \phi[\mathtt{x}:=\mathtt{z}_m] \Rightarrow \phi[\mathtt{x}:=\mathtt{K}\ \mathtt{y}_1 \ldots \mathtt{y}_n] \\ \qquad \text{where} \ldots \end{array}}{\vdash\ \phi}\ \mathtt{IND}$$

P4: Induction – only when intention is not "covered" by history

Crits (`srtd(ordr is`$^{[]}$`)`) = `is`$^{[]}$**`,p1`**

where

`p1 = s1.o1.[]`

Therefore, IND applicable now. ☺

$$\dfrac{\begin{array}{l} x \text{ has type } \mathtt{T},\ \ \mathtt{x}^{\mathbf{p}},\ p' \in \text{Crits}(\phi) \\ \ldots \mathtt{x}^{p''}\ldots,\ p''' \in \text{Crits}(\phi) \ \text{ implies } \ p' \text{ not a subpath of } p'' \\ \text{for each } \mathtt{K} \in \text{Constrs}(\mathtt{T}).\ \vdash \ \phi[\mathtt{x}:=\mathtt{z}_1], \ldots \phi[\mathtt{x}:=\mathtt{z}_m] \Rightarrow \phi[\mathtt{x}:=\mathtt{K}\ \mathtt{y}_1\ldots\mathtt{y}_n] \\ \quad\quad \text{where } \ldots \end{array}}{\vdash \phi}\ \mathtt{IND}$$

$$\dfrac{\cdot}{\mathtt{srtd\ (ordr\ is}^{[]}\mathtt{)}}\ \mathtt{IND}$$

Second step in proof

Remember, here we wanted to avoid application of induction.

$$\dfrac{\dfrac{\dots}{\texttt{srtd (ord [])}} \qquad \dfrac{\texttt{???}}{\texttt{srtd (ord js}^{\texttt{p1}}\texttt{) => srtd (ord j}^{\texttt{p1}}\texttt{:js}^{\texttt{p1}}\texttt{)}}\,\texttt{???}}{\texttt{srtd (ordr is}^{\texttt{[]}}\texttt{)}}\,\texttt{IND}$$

P4: Induction only applicable when intention not covered by history

Crits($\texttt{srtd(ordr js}^{\mathbf{p1}}\texttt{)}$) = $\texttt{js}^{\mathbf{p1}}$,**p1**

Crits($\texttt{srtd(ordr j}^{\mathbf{p1}}\texttt{:js}^{\mathbf{p1}}\texttt{)}$) = $\texttt{js}^{\mathbf{p1}}$,**p1**

where

p1 = s1.o1.[]

$$\dfrac{\dfrac{\dots}{\texttt{srtd (ord [])}} \qquad \dfrac{???}{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ord j}^{p1}\texttt{:js}^{p1}\texttt{)}}\,???}{\texttt{srtd (ordr is}^{[]}\texttt{)}}\,\texttt{IND}$$

P4: Induction only applicable when intention not covered by history

Crits(`srtd(ordr js^p)`) = js^{p1}, **p1**

Crits(`srtd(ordr j^p:js^p)`) = js^{p1}, **p1**

Therefore, IND not applicable now. ☺

$$\frac{\begin{array}{l} x \text{ has type } T,\ x^{p}, p' \in \text{Crits}(\phi) \\ \ldots x^{p''}\ldots,\ p''' \in \text{Crits}(\phi) \text{ implies } p' \text{ not a subpath of } p'' \\ \text{for each } K \in \text{Constrs}(T).\ \vdash\ \phi[x{:=}z_1], \ldots \phi[x{:=}z_m] \Rightarrow \phi[x{:=}K\ y_1\ldots y_n] \\ \quad \text{where} \ldots \end{array}}{\vdash\ \phi}\ \text{IND}$$

$$\frac{\dfrac{\ldots}{\texttt{srtd (ord [])}} \qquad \dfrac{???}{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ord j}^{p1}\texttt{:js}^{p1}\texttt{)}}\ ???}{\texttt{srtd (ordr is)}}\ \text{IND}$$

So far, …

- Induction applicable in the first step. ☺
- Induction not applicable in the second step. ☺

What about the later steps?

We shall look at the fourth proof step

At the **fourth** step

Remember, we wanted to
be allowed to apply IND here.

$$\dfrac{\dfrac{\dfrac{}{\texttt{srtd (ks}^{p1}\texttt{) => srtd (ins i}^{p1}\texttt{ ks}^{p1}\texttt{)}}\;???}{\dfrac{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ins j}^{p1}\texttt{ (ord js}^{p1}\texttt{)}}{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ord j}^{p1}\texttt{:js}^{p1}\texttt{)}}\;\mathrm{EXP}}\;\mathrm{GEN}}{\texttt{srtd (ordr is}^{p1}\texttt{)}}\;\mathrm{IND}$$

At the **fourth** step

Crits(srtd(ks^{p1})) = ks^{p1},**p2**

Crits(srtd(ins i^{p1} ks^{p1})) = ks^{p1},**p3**

where

p1 = s1.o1.[]

p2 = s1.[]

p3 = s1.i1.[]

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\texttt{srtd (ks}^{p1}\texttt{) => srtd (ins i}^{p1}\texttt{ ks}^{p1}\texttt{)}}\;\texttt{???}}{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ins j}^{p1}\texttt{ (ord js}^{p1}\texttt{)}}\;\texttt{GEN}}{\texttt{srtd (ord js}^{p1}\texttt{) => srtd (ord j}^{p1}\texttt{:js}^{p1}\texttt{)}}\;\texttt{EXP}\qquad .}{\texttt{srtd (ordr is}^{p1}\texttt{)}}\;\texttt{IND}$$

At the **fourth** step

Crits (srtd(ks^{p1})) = ks^{p1}**,p2**
Crits (srtd(ins i^{p1} ks^{p1})) = ks^{p1}**,p3**

where
p1 = s1.o1.[]
p2 = s1.[]
p3 = s1.i1.[]

p1 covers **p2**
p1 does not cover **p3**

???
srtd (ks^{p1}) => srtd (ins i^{p1} ks^{p1})
GEN
srtd (ord js^{p1}) => srtd (ins j^{p1} (ord js^{p1})
EXP
srtd (ord js^{p1}) => srtd (ord j^{p1}:js^{p1})
.
IND
srtd (ordr is^{p1})

At the **fourth** step

Crits(srtd(ks^{p1})) = ks^{p1},**p2**

Crits(srtd(ins i^{p1} ks^{p1})) = ks^{p1},**p3**

where

p1 = s1.o1.[]

p2 = s1.[]

p3 = s1.i1.[]

p1 covers **p2**

p1 does not cover **p3**

Therefore, IND is applicable. ☺

???

srtd (ks^{p1}) => srtd (ins i^{p1} ks^{p1})

GEN

srtd (ord js^{p1}) => srtd (ins j^{p1} (ord js^{p1})

EXP

srtd (ord js^{p1}) => srtd (ord j^{p1}:js^{p1})

.

IND

srtd (ordr is^{p1})

Summary

- Zeno proves equality over Haskell-like terms.
- Variables implicitly universally quantified; no support for existentials. Booleans are encoded through the `Bool` data type.
- From Isaplanner benchmark suite, Zeno can prove more properties than Isaplanner and ACL2s
- Zeno sometimes discovers useful further lemmas.
- Zeno's heuristics
  - Counteraxamples
  - Prioritize EQL and CON
  - Critical expressions restrict antecedents to "relevant ones" - they move the proof search towards making it possible to expand function bodies – as opposed to rippling
  - Paths keep track of the proof cases visited so far and avoid revisiting these cases; some "forbidden" steps my become allowed later in the poof.
  - …

Zeno

Contents [hide]

1 Introduction

Zeno is an automated proof system for Haskell program properties; developed at Imperial College London by William Sonnex, Sophia Drossopoulou and Susan Eisenbach. It aims to solve the general problem of equality between two Haskell terms, for any input value.

Many program verification tools available today are of the model checking variety; able to traverse a very large but finite search space very quickly. These are well suited to problems with a large description, but no recursive datatypes. Zeno on the other hand is designed to inductively prove properties over an infinite search space, but only those with a small and simple specification.