Zeno
an automated theorem prover for functional programs

Will Sonnex,
Computer Science Student
London, United Kingdom

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
  \[ E_1 = E_2, \ldots, E_{2n+1} = E_{2n+2} \implies E = E' \]
  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
  - \( \text{rev} (\text{rev} \ xs) = xs \)
  - \( \text{order} (\text{order} \ xs) = \text{order} \ xs \)
  - \( \text{mult} \ x \ (\text{succ} \ 0) = x \)

• 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

```haskell
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 (<=) = \x. \y. case x of
    { Zero -> True;
    Succ x' -> case y of
                { Zero -> False;
                Succ y' -> x' <= y' }
```
Example - Haskell and HC

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

letrec srt$ = λ ns. case ns of
  { [] -> True;
    x:xs -> case xs of
      { [] -> True;
        y:ys -> case x <= y of
          { True -> srt$ (y:ys);
            False -> False } } } }
Example Haskell and HC

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

letrec ordr = \ ns. case ns of
    { [] -> []; x:xs -> ins x (ordr xs) }
```
Example Haskell and HC

\[\text{ins} :: \text{Nat} \to [\text{Nat}] \to [\text{Nat}]\]
\[\text{ins} \ n \ [] = [n]\]
\[\text{ins} \ n \ (x:xs) \mid n \leq x = n:x:xs \mid \text{otherwise} \ x:(\text{ins} \ n \ xs)\]

\text{letrec} \ \text{ins} = \lambda \ n \ . \ \lambda \ ns \ . \ \text{case} \ ns \ of \\
\{ \ [] \ \to \ n:[]; \\
\ x:xs \ \to \ \text{case} \ n \leq x \ of \\
\{ \ True \ \to \ n:x:xs; \\
\ False \ \to \ x:(\text{ins} \ n \ xs)\} \} \} \}
Example in HC

\[
\begin{align*}
\text{letrec } & \text{ srtd } = \lambda \text{ ns. case ns of} \\
& \{ \text{ [] } \rightarrow \text{ True;} \\
& \text{ x:xs } \rightarrow \text{ case xs of} \\
& \{ \text{ [] } \rightarrow \text{ True;} \\
& \text{ y:ys } \rightarrow \text{ case x} \leq \text{ y of} \\
& \{ \text{ True } \rightarrow \text{ srtd (y:ys);} \\
& \text{ False } \rightarrow \text{ False } \} \} \} \\
\text{letrec } & \text{ ordr } = \lambda \text{ ns. case ns of} \\
& \{ \text{ [] } \rightarrow \text{ [];} \\
& \text{ x:xs } \rightarrow \text{ ins n (ordr xs)} \} \} \} \\
\text{letrec } & \text{ ins } = \lambda \text{ n. } \lambda \text{ ns. case ns of} \\
& \{ \text{ [] } \rightarrow \text{ n:[]} ; \\
& \text{ x:xs } \rightarrow \text{ case n} \leq \text{ x of} \\
& \{ \text{ True } \rightarrow \text{ n:x:xs;} \\
& \text{ False } \rightarrow \text{ x:(ins n xs)} \} \} \} 
\end{align*}
\]
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

\[ \text{srtd (ordr is)} \]

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) \n
We will first outline part of the proof, and then we will show the rules for the individual steps.
Proving $\text{srtd } (\text{ordr is})$
Proving $\text{srtd (ordr is)}$

\[
\begin{align*}
???? & \quad ??? \\
\text{srtd (ord [])} & \quad \text{srtd (ord js)} \Rightarrow \text{srtd (ord j:js)} \\
\text{srtd (or dr is)} & \quad \text{IND}
\end{align*}
\]
Proving \texttt{srdt (ordr is)}

\[
\begin{align*}
\text{srdt (ordr is)} & \Rightarrow \text{srdt (ins j (ordr js))} \\
\text{srdt (ordr [])} & \Rightarrow \text{srdt (ordr js)} \\
\text{srdt (ordr js)} & \Rightarrow \text{srdt (ordr j:js)} \\
\text{srdt (ordr is)} & \text{EXP} \\
\text{IND}
\end{align*}
\]
Proving \( \text{srtd} \ (\text{ordr is}) \)

\[
\frac{\text{srtd} \ (\text{ks}) \Rightarrow \text{srtd} \ (\text{ins i ks})}{\text{GEN}}
\]

\[
\frac{\text{srtd} \ (\text{ord js}) \Rightarrow \text{srtd} \ (\text{ins j (ord js)})}{\text{EXP}}
\]

\[
\frac{\text{srtd} \ (\text{ord js}) \Rightarrow \text{srtd} \ (\text{ord j:js})}{\text{IND}}
\]

\[
\frac{}{\text{srtd} \ (\text{ordr is})}
\]
Proving \( \text{srt} \text{d (ordr is)} \)

**Note:**
Zeno discovered the auxiliary lemma

\[
\text{srt} \text{d ks} \implies \text{srt} \text{d (ins j ks)}
\]

\[
\begin{align*}
\text{srt} \text{d (ks)} & \implies \text{srt} \text{d (ins j ks)} \\
\text{srt} \text{d (ord js)} & \implies \text{srt} \text{d (ins j (ord js))} \\
\text{srt} \text{d (ord [])} & \implies \text{srt} \text{d (ord j:js)} \\
\text{srt} \text{d (ordr is)} & \implies \text{srt} \text{d (ordr is)}
\end{align*}
\]
Proving \( \text{srtd} \ (\text{ordr is}) \)

\[
\begin{align*}
\text{srtd} \ ([]) & \Rightarrow \text{srtd} \ (\text{ins } i \ []) \\
\text{srtd} \ (\text{ks}) & \Rightarrow \text{srtd} \ (\text{ins } i \ \text{ks}) \\
\text{srtd} \ (\text{ord } []) & \Rightarrow \text{srtd} \ (\text{ord } \text{js}) \\
\text{srtd} \ (\text{ordr is}) & \Rightarrow \text{srtd} \ (\text{ordr is})
\end{align*}
\]
So, we want to prove

\[ \text{srtd (ordr is)} \]

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

\[
srtd \ (ord \ []) \quad srtd \ (ord \ js) \Rightarrow srtd \ (ord \ j:js)
\]

\[
\underline{srtd \ (ordr \ is)} \quad \text{IND}
\]
Proving \text{srtd (ordr is)} – the IND step

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

\text{srtd (ord [])} \quad \text{srtd (ord js)} \Rightarrow \text{srtd (ord j::js)}

\hline
\text{srtd (ordr is)} \quad \text{IND}
Proving \( \text{srtd (ordr \ is)} \) – the EXP step

\[
\begin{align*}
\text{srtd (ordr \ is)} & \Rightarrow \text{srtd (ord \ [])} \\
\text{srtd (ord \ is)} & \Rightarrow \text{srtd (ins \ j \ (ord \ js))} \\
\text{srtd (ord \ js)} & \Rightarrow \text{srtd (ord \ j:js)} \\
\text{srtd (ordr \ is)} & \end{align*}
\]
Proving \text{srtd (ordr is) – the EXP step}

\begin{align*}
\text{E evaluates to } E' \\
\vdash \phi [E:=E'] \\
\vdash \phi
\end{align*}

\begin{align*}
\text{srtd (ord [])} \\
\vdash \text{srtd (ord js) \Rightarrow srtd (ord j:js)} \\
\vdash \text{srtd (ord js) \Rightarrow srtd (ins j (ord js))} \\
\vdash \text{srtd (ordr is)}
\end{align*}
Proving \texttt{srtd (ordr is)} – the GEN step

\[
\begin{align*}
\text{x is fresh, and has type } T \\
\text{E has type } T \\
\vdash \phi [E := x] \quad \text{GEN} \\
\vdash \phi
\end{align*}
\]

\[
\begin{align*}
\text{srtd (ks) } \Rightarrow \text{srtd (ins i ks)} \\
\quad \text{GEN} \\
\text{srtd (ord js) } \Rightarrow \text{srtd (ins j (ord js))} \\
\quad \text{EXP} \\
\text{srtd (ord js) } \Rightarrow \text{srtd (ord j:js)} \\
\quad \text{IND} \\
\text{srtd (ordr is)}
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>Theorem prover</th>
<th>Percentage proven</th>
<th>Identifiers of unproven properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dafny (Z3 and IND)</td>
<td>53.5%</td>
<td>45-85</td>
</tr>
<tr>
<td>Isaplanner</td>
<td>55%</td>
<td>47-85</td>
</tr>
<tr>
<td>ACL2s – coded types</td>
<td>87%</td>
<td>47, 50, 54, 56, 72, 73, 74, 81, 83, 84, 85</td>
</tr>
<tr>
<td>Zeno</td>
<td>96%</td>
<td>72, 74, 85</td>
</tr>
</tbody>
</table>
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 branches;
    
    $K, K' \text{ are constructors}$
    
    $K \neq K'$
    
    \[
    \vdash (K \ E_1 \ldots E_n) = (K' \ E'_1 \ldots E'_n) \Rightarrow \emptyset
    \]
    
    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 (i.e., 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
Critical expressions - example

At this point, many steps could be considered:

- \textbf{IND on is}
- \textbf{CASE on ord is}
- \textbf{CASE on srt\textit{d}}(ord is)
- \textbf{IND on ord is}
- \textbf{CASE on first (is)}
- ...

\textbf{srt\textit{d}} (or\textit{dr} is)
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
- ...

\[
\begin{align*}
... & \quad ???? \\
\text{srtd (ord js)} & \Rightarrow \text{srtd (ord j:js)} \\
\end{align*}
\]

\[
\text{srtd (ordr is)}
\]
Critical expressions - definition

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

\[
\text{Crits}(E) = \begin{cases} 
E & \text{if } E \text{ is normal} \\
\text{Crits}(E') & \text{if } E \rightarrow^* \text{ case } E' \text{ of } ..., E' \in E \\
E' & \text{if } E \rightarrow^* \text{ case } E' \text{ of } ..., E' \notin E
\end{cases}
\]

\(E\) is normal if it cannot be further re-written.
Critical expressions - examples

\[
\begin{aligned}
\text{Crits}(E) &= E \quad \text{if } E \text{ is normal} \\
\text{Crits}(E') &= \text{if } E \rightarrow^* \text{case } E' \text{ of } \ldots, \ E' \subseteq E \\
E' &= \text{if } E \rightarrow^* \text{case } E' \text{ of } \ldots, \ E' \not\subseteq E
\end{aligned}
\]

\[
\begin{aligned}
\text{ord}(is) &\rightarrow^* \text{case } is \text{ of } \{ \left[ \right] \rightarrow \ldots; \ x:xs \rightarrow \ldots \} \\
\text{srted}(\text{ord}(is)) &\rightarrow^* \text{case } \text{ord}(is) \text{ of } \\
&\quad \{ \text{True } \rightarrow \ldots; \ \text{False } \rightarrow \ldots \}
\end{aligned}
\]

\[
\begin{aligned}
\text{Crits}(\text{ord}(is)) &= is \\
\text{Crits}(\text{srted}(\text{ord } (is))) &= is
\end{aligned}
\]
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)
- ...

...
Using Critical Expressions - IND

reduces the proof search space

\[
x \text{ has type } T, \quad x \in \text{Crits } (\phi) \quad \text{for each } K \in \text{Constrs } (T) \Rightarrow \phi [x := z_1], \ldots, \phi [x := z_m] \Rightarrow \phi [x := K \ y_1 \ldots y_n]
\]

where ... \hspace{10cm} \text{IND}
\[
\vdash \phi
\]

\[
\ldots.
\]

\[
srtd \ (\text{ordr is})
\]
Using Critical Expressions - IND

reduces the proof search space

\[ \text{Crits}(\text{srtd(ordr is)}) = \{ \text{is} \} \]

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)
- ...

\[ \text{srtd (ordr is)} \]
Using Critical Expressions - IND

reduces the proof search space

\[ x \text{ has type } T, \quad x \in \text{Crits}(\phi) \]
\[ \text{for each } K \in \text{Constrs}(T) . \quad \vdash \phi[x:=z_1], \ldots \phi[x:=z_m] \Rightarrow \phi[x:=K \ y_1 \ldots y_n] \]
\[ \text{where } \ldots \]

\[ \text{IND} \]
\[ \vdash \phi \]

\[ \text{Crits}\ (\text{srtd}\ (\text{ordr\ is})) = \{\ is\ \} \]

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)

\[ \text{srtd}\ (\text{ord } []) \quad \text{srtd}\ (\text{ord js}) \Rightarrow \text{srtd}\ (\text{ord j:js}) \]
\[ \text{IND} \]

\[ \text{srtd}\ (\text{ordr\ is}) \]
Critical expressions need not be subterms

\[
\text{Crits}(E) = \begin{cases} 
E & \text{if } E \text{ is normal} \\
\text{Crits}(E') & \text{if } E \rightarrow^* \text{ case } E' \text{ of } \ldots, E' \in E \\
E' & \text{if } E \rightarrow^* \text{ case } E' \text{ of } \ldots, E' \notin E
\end{cases}
\]

\[
\text{ins } i \ (j:js) \rightarrow^* \text{ case } i \leq j \ of \\
\{ \text{ True } \rightarrow \ldots; \text{ False } \rightarrow \ldots \}
\]

\[
\text{srtd} (\text{ins } i \ (j:js)) \rightarrow^* \\
\text{case} \ (\text{ins } i \ (j:js)) \ of \\
\{ \ [ ] \rightarrow \ldots; y:ys \rightarrow \ldots \}
\]

\[
\text{Crits} (\ \text{ins } i \ (j:js)) = i \leq j \\
\text{Crits} (\ \text{srtd} (\text{ins } i \ (j:js))) = i \leq j
\]
Use of critical Expressions which are not subterms

\[
\text{srtd} (j:js) \Rightarrow \text{srtd}(\text{ins} i (j:js))
\]
Use of critical Expressions which are not subterms

Critical expressions which are not subterms are used for case analysis

\[ \text{Crits} \left( \text{srtd}(\text{ins } i \ (j:js)) \right) = i \leq j \]
Use of critical Expressions which are not subterms

Critical expressions which are not subterms are used for case analysis

\[
\text{Crits( srt\text{d}(\text{ins } i \ (j:j\text{s})))} = \begin{cases} 
\text{i} \leq \text{j} 
\end{cases}
\]

\[
\begin{align*}
\text{i} \leq \text{j} = \text{True} & \Rightarrow \\
\text{srt\text{d}} \ (j:j\text{s}) & \Rightarrow \\
\text{srt\text{d( ins } i \ (j:j\text{s}))} & \\
\text{srt\text{d}} \ (j:j\text{s}) & \Rightarrow \\
\text{srt\text{d( ins } i \ (j:j\text{s}))}
\end{align*}
\]
However, consider ...

... srtd (ord []) srtd (ord js) => srtd (ord j:js) ???

. srtd (ordr is)
However, consider ...

\[
\text{Crits}\left( \text{srtd}(\text{ordr } j s) \right) = \text{Crits}\left( \text{srtd}(\text{ordr } j s) \right) = js
\]
However, consider ...

$$\text{Crits}(\text{srt}(\text{ord r j})) = \text{Crits}(\text{srt}(\text{ord l j})) = \text{j}$$

Should we apply induction on \text{j}\text{s}? 

\[ \ldots \text{srt}(\text{ord} []), \quad \text{srt}(\text{ord} \text{j}) \Rightarrow \text{srt}(\text{ord} \text{j:j}) \quad \text{IND} \]

\[ . \text{srt}(\text{ord} \text{i}) \]
However, consider ...

\[
\text{Crits}(\text{srtd(ordr js)}) = \text{Crits}(\text{srtd(ordr js)}) = i \leq j
\]

Should we apply induction on \( js \)? Again induction?

... 
\[
\text{srtd(ord \ [\])} \\
\text{srtd(ord js) => srtd(ord j:js)}
\]

. 
\[
\text{srtd(ordr is)}
\]

IND
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.
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 Statements

...
P2: Decorating critical expressions

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

\[
\text{Crits}(E) = \begin{cases} 
E, [] & \text{if } E \text{ is normal} \\
E'', 1:p & \text{if } E \rightarrow^* \text{ case}^1 E' \text{ of } ..., \ E' \in E \\
E', 1 & \text{if } E \rightarrow^* \text{ case}^1 E' \text{ of } ..., \ E' \notin E \\
\text{Crits}(E') = E'', p \end{cases}
\]
P2: Decorating critical expressions - examples

\[
\text{Crits}(E) = \begin{cases} 
E, [] & \text{if } E \text{ is normal} \\
E'', \text{l.p} & \text{if } E \rightarrow^* \text{case}^1 E' \text{ of } \ldots, \ E' \in E \\
\text{Crits}(E') = E'', \text{p} \\
E', \text{l} & \text{if } E \rightarrow^* \text{case}^1 E' \text{ of } \ldots, \ E' \notin E 
\end{cases}
\]

\[
\text{ord(is}[^1] \rightarrow^* \text{ case}^0 \text{ is of } \{ [ ] -> \ldots; \ x:xs -> \ldots \} \\
\text{srted(ord(is}[^1]) \rightarrow^* \text{ case}^1 \text{ ord(is) of} \\
\{ \text{True -> } \ldots; \ \text{False -> } \ldots \} 
\]

\[
\text{Crits( ord(is}[^1]) ) = is[^1], \text{o1.}[] \\
\text{Crits( srted(ord(is}[^1])) ) = is[^1], \text{s1.o1.}[] 
\]
P2: Decorating critical expressions - examples

Crits(E) =

\[
\begin{align*}
\text{E,[]} & \quad \text{if E is normal} \\
\text{E'',l.p} & \quad \text{if E->* case}^1 \text{ E' of } ..., \text{ E' } \in \text{E} \\
\text{Crits(E')} & = \text{E'',p} \\
\text{E',l} & \quad \text{if E->* case}^1 \text{ E' of } ..., \text{ E' } \notin \text{E}
\end{align*}
\]

ord(is^[1]) ->* case^[01] is of \{ [] -> ...; x:xs -> ... \}
srtd(ord(is^[1])) ->* case^[s1] ord(is) of
\{ True -> ...; False -> ... \}

Crits( ord(is^[1]) ) = is^[1],o1.[] 
Crits( srtd(ord(is^[1])) ) = is^[1],s1.o1.[] 

When is^[1] is taken for srtd(ord(is^[1])), it intends to cover the cases s1.o1
P3: Decorating variables

srt\(d\) (ordr is\(^1\))
P4: Induction – only when intention is not “covered” by history

\( x \) has type \( T \), \( x^p, p' \in \text{Crits}(\phi) \)

\( \ldots x^{p''} \ldots, p''' \in \text{Crits}(\phi) \) implies \( p' \) not a subpath of \( p'' \)

for each \( K \in \text{Constrs}(T) \) \( \vdash \phi[x:=z_1], \ldots, \phi[x:=z_m] \Rightarrow \phi[x:=K_y_1 \ldots y_n] \)

where \( \ldots \)

\( \vdash \phi \)
P4: Induction – only when intention is not “covered” by history

\[
\text{Crits} ( \text{srtd} (\text{ordr is}^{[1]})) = \text{is}^{[1]}, p1
\]

where
\[
p1 = s1.o1.[]
\]

Therefore, IND applicable now. 😊

\[
x \text{ has type } T, \quad \forall p, p' \in \text{Crits} (\phi) \\
\ldots x^{p^\prime}\ldots, p'' \in \text{Crits} (\phi) \implies p' \text{ not a subpath of } p''
\]

for each \( K \in \text{Constrs} (T) \).

\[
\vdash \phi [x:=z_1], \ldots, \phi [x:=z_m] \Rightarrow \phi [x:=K \ y_1 \ldots y_n]
\]

where ...

\[
\vdash \phi
\]

\[
\text{srtd} (\text{ordr is}^{[1]})
\]

\[
\text{IND}
\]
Second step in proof

Remember, here we wanted to avoid application of induction.

\[
\begin{align*}
\text{...} & \quad \text{srtd (ord \hspace{1em} [])} \\
& \quad \text{srtd (ord js^{p1})} \implies \text{srtd (ord j^{p1}:js^{p1})} \\
& \quad \text{IND} \\
\hline
& \quad \text{srtd (ordr is^{[1]})}
\end{align*}
\]
P4: Induction only applicable when intention not covered by history

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

where
p1 = s1.o1.[]

... srtd(ordr []) srtd(ordr js^{p1}) \Rightarrow srtd(ordr j^{p1}:js^{p1})
\text{IND}
**P4: Induction only applicable when intention not covered by history**

\[
\text{Crits( srtd(ordr js^p)) = js^{p_1},p_1} \\
\text{Crits( srtd(ordr j^p:js^p)) = js^{p_1},p_1}
\]

Therefore, IND not applicable now. 😊

\[
x \text{ has type } T, \quad x^p, p' \in \text{Crits( } \phi \text{ )} \\
...x^{p''...}, p''' \in \text{Crits( } \phi \text{ )} \implies p' \text{ not a subpath of } p'''
\]

for each \( K \in \text{Constrs} (T) \). \[ \vdash \phi [x:=z_1], ... \phi [x:=z_m] \Rightarrow \phi [x:=K \ y_1...y_n] \]

\[ \text{where } ... \]

\[ \vdash \phi \]

\[ \vdash \phi \]

\[ \text{srtd (ord [ ])} \quad \text{srtd (ord js^{p_1})} \Rightarrow \text{srtd (ord j^{p_1}:js^{p_1})} \]

\[ \text{IND} \]
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.

\[
\begin{align*}
\text{srtd (ks}^{p_1}) & \Rightarrow \text{srtd (ins }i^{p_1} \text{ ks}^{p_1}) \tag{??}
\text{srtd (ord }j\text{s}^{p_1}) & \Rightarrow \text{srtd (ins }j^{p_1} \text{ (ord }j\text{s}^{p_1}) \\
\text{srtd (ord }j\text{s}^{p_1}) & \Rightarrow \text{srtd (ord }j^{p_1}:j\text{s}^{p_1}) \tag{EXP}
\text{srtd (ordr }i\text{s}^{p_1}) & \tag{IND}
\end{align*}
\]
At the fourth step

\[
\text{Crits( } \text{srtd}(ksp^1) \text{) } = \text{ } ksp^1, p2
\]
\[
\text{Crits( } \text{srtd}(\text{ins } i^{p1} \text{ } ksp^1) \text{) } = \text{ } ksp^1, p3
\]

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

\[
\text{srtd} (ksp^1) \Rightarrow \text{srtd} (\text{ins } i^{p1} \text{ } ksp^1)
\]

\[
\text{srtd} (\text{ord } jsp^1) \Rightarrow \text{srtd} (\text{ins } j^{p1} \text{ (ord jsp^1)}
\]

\[
\text{srtd} (\text{ord } jsp^1) \Rightarrow \text{srtd} (\text{ord } j^{p1}:jsp^{p1})
\]

\[
\text{srtd} (\text{ordr is}^{p1})
\]
At the fourth step

\[
\text{Crits}(\text{srtd}(k^{p_1})) = k^{p_1},p_2 \\
\text{Crits}(\text{srtd}(\text{ins } i^{p_1} \ k^{p_1})) = k^{p_1},p_3
\]

where

\[
\begin{align*}
p_1 &= s1.o1.[] \\
p_2 &= s1.[] \\
p_3 &= s1.i1.[]
\end{align*}
\]

p1 covers p2
p1 does not cover p3

\[
\begin{align*}
\text{srtd} (k^{p_1}) &\Rightarrow \text{srtd} (\text{ins } i^{p_1} \ k^{p_1}) \\
\text{srtd} (\text{ord } j^{p_1}) &\Rightarrow \text{srtd} (\text{ins } j^{p_1} \ (\text{ord } j^{s^{p_1}})) \\
\text{srtd} (\text{ord } j^{p_1}) &\Rightarrow \text{srtd} (\text{ord } j^{p_1}:j^{s^{p_1}}) \\
\text{srtd} (\text{ordr } i^{s^{p_1}})
\end{align*}
\]
At the **fourth** step

Crits( srtd(ks$^{p1}$)) = ks$^{p1}$,p$^2$
Crits( srtd(ins i$^{p1}$ ks$^{p1}$)) = ks$^{p1}$,p$^3$

where

p$^1$ = s1.o1.[[]]  p$^1$ covers p$^2$
p$^2$ = s1.[[]]  p$^1$ does not cover p$^3$
p$^3$ = s1.i1.[[]]  Therefore, IND is applicable. 😊

\[
\begin{align*}
\text{srt}d (\text{ks}^{p1}) & \Rightarrow \text{srt}d (\text{ins i}^{p1} \text{ ks}^{p1}) \\
\text{GEN} & \\
\text{srt}d (\text{ord js}^{p1}) & \Rightarrow \text{srt}d (\text{ins j}^{p1} (\text{ord js}^{p1})) \\
\text{EXP} & \\
\text{srt}d (\text{ord js}^{p1}) & \Rightarrow \text{srt}d (\text{ord j}^{p1} : \text{js}^{p1}) \\
\text{IND} & \\
\text{srt}d (\text{ordr is}^{p1}) & 
\end{align*}
\]
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 Isaplaner benchmark suite, Zeno can prove more properties than Isaplaner and ACL2s
• Zeno sometimes discovers useful further lemmas.
• Zeno’s heuristics
  – Counterexamples
  – 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 may become allowed later in the proof.
  – ...

...
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.