Data Layout from a Type-Theoretic Perspective

Frank Pfenning
Joint work with Henry DeYoung

Computer Science Department
Carnegie Mellon University

MFPS 2022
July 11, 2022
Invited Talk

Logic and Computation

- One Important thread:
  1. Propositions as types
  2. Proofs as programs
  3. Reduction as computation
- Co-design programming language and reasoning principles
- Provides some extensibility and robustness
- (1) depends on logic and its vocabulary
- (2,3) depend on details of presentation
- (2,3) yield preservation and progress
- This talk:
  - Fix the logic (= intuitionistic propositional logic)
  - Vary the judgmental principles of proof

Judgmental Principles of Proof (Examples)

Intuitionistic Logic	**Functional Programming**
Axioms + MP [Hilbert'27]	Combinators [Curry'35]
Natural Deduction [Gentzen'35]	λ-Calculus [Howard'69]
Sequents with Stoop (LJT)	Explicit Substitutions [Herbelin'94]
Semi-Axiomatic Seq Calc (SAX)	Futures [DeYoung,Pf,Pruiksma'20]
SAX with Snip (SNAX)	Data Layout [this talk]

Outline

- From Natural Deduction (ND) to Semi-Axiomatic Sequent Calculus (SAX)
- Programming in SAX
- Cut Elimination and Snips in SAX
- Data Layout and SAX with Snips (SNAX)
- Examples Revisited

Example: Disjunction in Natural Deduction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee I_1 \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee I_2 \quad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee E$$

$$\frac{\Gamma \vdash M_1 : A}{\Gamma \vdash \mathtt{l}(M_1) : A \vee B}\ \vee I_1 \quad \frac{\Gamma \vdash M_2 : B}{\Gamma \vdash \mathtt{r}(M_2) : A \vee B}\ \vee I_2$$

$$\frac{\Gamma \vdash M : A \vee B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, y : B \vdash N_2 : C}{\Gamma \vdash \textbf{case}\ M\ (\mathtt{l}(x) \Rightarrow N_1 \mid \mathtt{r}(y) \Rightarrow N_2) : C}\ \vee E$$

$$\textbf{case}\ \mathtt{l}(M_1)\ (\mathtt{l}(x) \Rightarrow N_1 \mid \ldots) \longrightarrow [M_1/x]N_1$$
$$\textbf{case}\ \mathtt{r}(M_2)\ (\ldots \mid \mathtt{r}(y) \Rightarrow N_2) \longrightarrow [M_2/y]N_2$$

- Fundamental operation underlying reduction
  - Substitution of proof for hypothesis (logic)
  - Substitution of term for variable (computation)
- Derives from meaning of hypothetical judgments

Futures

- Define futures with

$$x \leftarrow P \,;\, Q$$

  - Allocate a fresh future for x
  - Compute P with destination x
  - In parallel, compute Q which may read from x
  - Q blocks if it tries to read from x before P has written to x
- Scheduling
  - Futures: compute P and Q in parallel
  - Call-by-value: complete P before starting Q
  - Call-by-need: postpone P until Q needs x

Futures, Logically

- Logically, futures are a cut from the sequent calculus

$$\dfrac{\overset{P}{\Gamma \vdash A} \quad \overset{Q}{\Gamma, A \vdash C}}{\Gamma \vdash C}\ \text{cut} \qquad \dfrac{\Gamma \vdash P :: (x : A) \quad \Gamma, x : A \vdash Q :: (w : C)}{\Gamma \vdash x \leftarrow P\ ;\ Q :: (w : C)}\ \text{cut}$$

- Interpret sequents with addresses a_i and w

$$\underbrace{a_1 : A_1, \ldots, a_n : A_n}_{\text{read from}} \vdash P :: (\ \underbrace{w : C}_{\text{write to}}\ )$$

- Types A_i and C represent types of value at address a_i and w

Not Quite the Sequent Calculus

- Writer terminates after writing to destination
- Example: Disjunction / Sums

$$\frac{}{a : A \vdash \textbf{write}\ c\ \mathtt{l}(a) :: (c : A \vee B)}\ \vee X_1$$

$$\frac{}{b : B \vdash \textbf{write}\ c\ \mathtt{r}(b) :: (c : A \vee B)}\ \vee X_2$$

- a, b, and c are addresses: values are small ($\mathtt{l}(a)$ and $\mathtt{r}(b)$)
- Reader continues based on value read

$$\frac{\Gamma, x : A \vdash Q :: (w : C) \quad \Gamma, y : B \vdash R :: (w : C)}{\Gamma, c : A \vee B \vdash \textbf{read}\ c\ (\mathtt{l}(x) \Rightarrow Q \mid \mathtt{r}(y) \Rightarrow R) :: (w : C)}\ \vee L$$

Specifying Dynamics as Multiset Rewriting

- State of computation represented as a multiset of objects
- Any subset can be rewritten by a transition rule
- For shared memory dynamics
  - Ephemeral objects **thread** P represent running threads
  - Ephemeral objects **cell** c $\square$ were allocated but not yet written
  - Persistent objects !**cell** c V were allocated and written
- Sample rules (disjunction/sums)

$$\text{thread}\ (\textbf{write}\ c\ \mathtt{l}(a)), \text{cell}\ c\ \square \longrightarrow\ !\text{cell}\ c\ \mathtt{l}(a)$$
$$\text{thread}\ (\textbf{write}\ c\ \mathtt{r}(b)), \text{cell}\ c\ \square \longrightarrow\ !\text{cell}\ c\ \mathtt{r}(b)$$

$$!\text{cell}\ c\ \mathtt{l}(a), \text{thread}\ (\textbf{read}\ c\ (\mathtt{l}(x) \Rightarrow Q \mid \ldots)) \longrightarrow \text{thread}\ [a/x]Q$$
$$!\text{cell}\ c\ \mathtt{r}(b), \text{thread}\ (\textbf{read}\ c\ (\ldots \mid \mathtt{r}(y) \Rightarrow R)) \longrightarrow \text{thread}\ [b/y]R$$

Semi-Axiomiatic Sequent Calculus (SAX)

- Disjunction, purely logically

$$\frac{}{A \vdash A \vee B}\ \vee X_1 \qquad \frac{}{B \vdash A \vee B}\ \vee X_2 \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L$$

- General rules

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\ \mathsf{cut} \qquad\qquad \frac{}{A \vdash A}\ \mathsf{id}$$

- Leave weakening and contraction implicit, for conciseness
- All positive right and negative left rules become axioms
- [DeYoung, Pf, Pruiksma; FSCD 2020]

SAX, Positive Connectives

- Positive connectives ($\vee$, $\otimes$, 1)
- Right rules are noninvertible*, become axioms $\vee X_i$, $\otimes X$, $1X$
- Left rules are invertible, remain $\vee L$, $\otimes L$, $1L$

$$\frac{}{A \vdash A \vee B}\ \vee X_1 \qquad \frac{}{B \vdash A \vee B}\ \vee X_2 \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L$$

$$\frac{}{A, B \vdash A \otimes B}\ \otimes X \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ \otimes L$$

$$\frac{}{\cdot \vdash 1}\ 1X \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C}\ 1L$$

SAX, Negative Connectives

- Negative connectives ($\supset$, $\wedge$)
- Right rules are invertible, remain $\supset R$, $\wedge R$
- Left rules are noninvertible, become axioms $\supset X$, $\wedge X_i$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \supset R \qquad \frac{}{A, A \supset B \vdash B} \supset X$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge R \qquad \frac{}{A \wedge B \vdash A} \wedge X_1 \quad \frac{}{A \wedge B \vdash B} \wedge X_2$$

Back to Futures

- Generalize $A \vee B$ to $\Sigma_{\ell \in L}(\ell : A_\ell)$, where $A \vee B = (\mathtt{l} : A) + (\mathtt{r} : B)$
- Generalize $A \wedge B$ to $\&_{\ell \in L}(\ell : A_\ell)$, where $A \wedge B = (\mathtt{l} : A) \mathbin{\&} (\mathtt{r} : B)$

Labels/Tags	ℓ, k				
Addresses	$a, b, c, d, w\ ;\ x, y, z$				
Small values	V	$::=$	$k(a)$	$(\Sigma_\ell(\ell : A_\ell))$	$(\&_\ell(\ell : A_\ell))$
		$\mid$	$\langle a, b\rangle$	$(A \otimes B)$	$(A \supset B)$
		$\mid$	$\langle\,\rangle$	(1)	
Continuations	K	$::=$	$(\ell(x) \Rightarrow P_\ell)_\ell$	$(\Sigma_\ell(\ell : A_\ell))$	$(\&_\ell(\ell : A_\ell))$
		$\mid$	$\langle x, y\rangle \Rightarrow P$	$(A \otimes B)$	$(A \supset B)$
		$\mid$	$\langle\,\rangle \Rightarrow P$	(1)	
Processes	P	$::=$	**write** $a\ V$	$(\Sigma, \otimes, 1)$	
		$\mid$	**read** $a\ K$	$(\Sigma, \otimes, 1)$	
		$\mid$	**write** $a\ K$		$(\&, \supset)$
		$\mid$	**read** $a\ V$		$(\&, \supset)$
		$\mid$	$x \leftarrow P\ ;\ Q$	(allocate x)	
		$\mid$	**copy** $a\ b$	(copy to a from b)	

Cut/Allocate and Identity/Copy

- Cut/Allocate: $x \leftarrow P \,;\, Q$ (only form of allocation)
- P writes to x, Q may read from x

$$\frac{\Gamma \vdash P :: (x : A) \quad \Gamma, x : A \vdash Q :: (w : C)}{\Gamma \vdash (x \leftarrow P \,;\, Q) :: (w : C)}\ \text{cut}$$

$$\text{thread}\ (x \leftarrow P \,;\, Q) \longrightarrow \text{thread}\ [a/x]P, \text{cell}\ a\ \square, \text{thread}\ [a/x]Q \qquad (a\ \text{fresh})$$

- Id/Copy: **copy** $a\ b$
- Copy to a from b

$$\frac{}{\Gamma, b : A \vdash \mathbf{copy}\ a\ b :: (a : A)}\ \text{Id}$$

$$!\text{cell}\ b\ V, \text{thread}\ (\mathbf{copy}\ a\ b), \text{cell}\ a\ \square \longrightarrow\ !\text{cell}\ a\ V$$

Write and Read

- Generic over values and continuations

$$\begin{array}{lcl} \text{thread}\ (\textbf{write}\ a\ V), \text{cell}\ a\ \Box & \longrightarrow & !\text{cell}\ a\ V \\ \text{thread}\ (\textbf{read}\ a\ K), \text{cell}\ a\ V & \longrightarrow & \text{thread}\ (V \triangleright K) \end{array}$$

$$\begin{array}{lcl} \text{thread}\ (\textbf{write}\ a\ K), \text{cell}\ a\ \Box & \longrightarrow & !\text{cell}\ a\ K \\ \text{thread}\ (\textbf{read}\ a\ V), \text{cell}\ a\ K & \longrightarrow & \text{thread}\ (V \triangleright K) \end{array}$$

- Passing values to continuations

$$\begin{array}{lcl} k(a) \triangleright (\ell(x) \Rightarrow P_\ell)_{\ell \in L} & = & [a/x]P_k \\ \langle a, b \rangle \triangleright (\langle x, y \rangle \Rightarrow P) & = & [a/x, b/y]P \\ \langle\,\rangle \triangleright (\langle\,\rangle \Rightarrow P) & = & P \end{array}$$

Translation from Natural Deduction

- $[\![e]\!]\ d = P$ where P computes the value of e with destination d

$$
\begin{array}{lcl}
[\![x]\!]\ d & = & \mathbf{copy}\ d\ x \\
\hline
[\![\langle e_1, e_2\rangle]\!]\ d & = & x_1 \leftarrow [\![e_1]\!]\ x_1\ \mathbf{;} \\
 & & x_2 \leftarrow [\![e_2]\!]\ x_2\ \mathbf{;} \\
 & & \mathbf{write}\ d\ \langle x_1, x_2\rangle \\
[\![\mathbf{case}\ e\ (\langle x, y\rangle \Rightarrow e')]\!]\ d & = & z \leftarrow [\![e]\!]\ z\ \mathbf{;} \\
 & & \mathbf{read}\ z\ (\langle x, y\rangle \Rightarrow [\![e']\!]\ d) \\
\hline
[\![e_1\ e_2]\!]\ d & = & x_1 \leftarrow [\![e_1]\!]\ x_1\ \mathbf{;} \\
 & & x_2 \leftarrow [\![e_2]\!]\ x_2\ \mathbf{;} \\
 & & \mathbf{read}\ x_1\ \langle x_2, d\rangle \\
[\![\lambda x.\, e]\!]\ d & = & \mathbf{write}\ d\ (\langle x, z\rangle \Rightarrow [\![e]\!]\ z)
\end{array}
$$

Outline

- From Natural Deduction (ND) to Semi-Axiomatic Sequent Calculus (SAX)
- Programming in SAX
- Cut Elimination and Snips in SAX
- Data Layout and SAX with Snips (SNAX)
- Examples Revisited

Let's Program: Operations on Bits

- Type definitions $t = A$
- Cell definitions $c : A = P$ (A positive)
- Process definitions $f\ w\ x_1\ \ldots\ x_n = P$
  - Requires $x_1 : A_1, \ldots, x_n : A_n \vdash P :: (w : C)$
  - First "argument" is always the destination

$\text{Unit} : 1 = \textbf{write}\ \text{Unit}\ \langle\,\rangle$

$\text{bool} = (\texttt{false} : 1) + (\texttt{true} : 1)$
$\text{False} : \text{bool} = \textbf{write}\ \text{False}\ \texttt{false}(\text{Unit})$
$\text{True} : \text{bool} = \textbf{write}\ \text{True}\ \texttt{true}(\text{Unit})$

$b : \text{bool} \vdash \text{neg} :: (c : \text{bool})$
$\text{neg}\ c\ b =$
$\quad \textbf{read}\ b\ (\,\texttt{false}(u) \Rightarrow \textbf{write}\ c\ \texttt{true}(u)$
$\qquad\qquad \mid \texttt{true}(u) \Rightarrow \textbf{write}\ c\ \texttt{false}(u)\,)$

$b : \text{bool}, c : \text{bool} \vdash \text{or} :: (d : \text{bool})$
$\text{or}\ d\ b\ c =$
$\quad \textbf{read}\ b\ (\,\texttt{false}(u) \Rightarrow \textbf{copy}\ d\ c$
$\qquad\qquad \mid \texttt{true}(u) \Rightarrow \textbf{write}\ d\ \texttt{true}(u)\,)$

Let's Program: Binary Successor

- Type and process definitions may be recursive (beyond logic)

$\mathrm{bin} = (\mathtt{b0} : \mathrm{bin}) + (\mathtt{b1} : \mathrm{bin}) + (\mathtt{e} : 1)$

$$
\begin{array}{lll}
\mathrm{Six} : \mathrm{bin} = & x_0 \leftarrow \textbf{write}\ x_0\ \mathtt{e}(\mathrm{Unit})\ \textbf{;} & \%\ !\mathrm{cell}\ c_0\ \mathtt{e}(\mathrm{Unit}) \\
& x_1 \leftarrow \textbf{write}\ x_1\ \mathtt{b1}(x_0)\ \textbf{;} & \%\ !\mathrm{cell}\ c_1\ \mathtt{b1}(c_0) \\
& x_2 \leftarrow \textbf{write}\ x_2\ \mathtt{b1}(x_1)\ \textbf{;} & \%\ !\mathrm{cell}\ c_2\ \mathtt{b1}(c_1) \\
& \textbf{write}\ \mathrm{Six}\ \mathtt{b0}(x_2) & \%\ !\mathrm{cell}\ \mathrm{Six}\ \mathtt{b0}(c_2)
\end{array}
$$

$x : \mathrm{bin} \vdash \mathrm{succ} :: (y : \mathrm{bin})$

$\mathrm{succ}\ y\ x =$

$$
\begin{array}{l}
\quad \textbf{read}\ x\ (\ \mathtt{b0}(x') \Rightarrow \textbf{write}\ y\ \mathtt{b1}(x') \\
\qquad\qquad \mid \mathtt{b1}(x') \Rightarrow y' \leftarrow \mathrm{succ}\ y'\ x'\ \textbf{;} \qquad \%\ \textit{allocate destination } y' \textit{ for } \mathrm{succ} \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ y\ \mathtt{b0}(y') \\
\qquad\qquad \mid \mathtt{e}(u) \Rightarrow \quad \textbf{write}\ y\ \mathtt{e}(u)\ )
\end{array}
$$

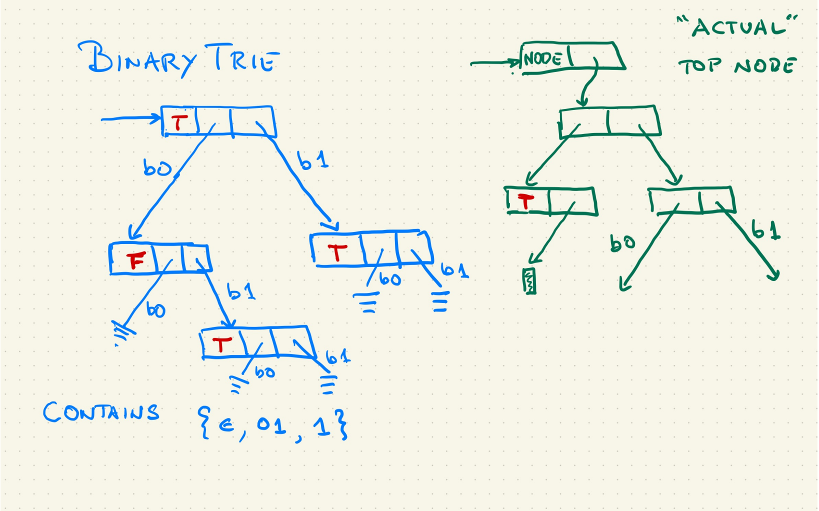
$x : \mathrm{bin} \vdash \mathrm{plus2} :: (z : \mathrm{bin})$ % *a trivial pipeline*

$\mathrm{plus2}\ z\ x =$

$\quad y \leftarrow \mathrm{succ}\ y\ x\ \textbf{;}$

$\quad \mathrm{succ}\ z\ y$

Example: Binary Tries



Let's Program: Binary Tries

$$\text{trie} = (\texttt{leaf} : 1) + (\texttt{node} : \text{bool} \otimes (\text{trie} \otimes \text{trie}))$$

$$x : \text{bin} \vdash \text{singleton} :: (t : \text{trie})$$

$$
\begin{array}{l}
\text{singleton}\ t\ x = \\
\quad \textbf{read}\ x\ (\ \texttt{b0}(x') \Rightarrow t_0 \leftarrow \text{singleton}\ t_0\ x'\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad p \leftarrow \textbf{write}\ p\ \langle t_0, \text{Leaf} \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad n \leftarrow \textbf{write}\ n\ \langle \text{False}, p \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ t\ \texttt{node}(n) \\
\qquad\qquad\quad \mid \texttt{b1}(x') \Rightarrow t_1 \leftarrow \text{singleton}\ t_1\ x'\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad p \leftarrow \textbf{write}\ p\ \langle \text{Leaf}, t_1 \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad n \leftarrow \textbf{write}\ n\ \langle \text{False}, p \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ t\ \texttt{node}(n) \\
\qquad\qquad\quad \mid \texttt{e}(u) \Rightarrow \quad p \leftarrow \textbf{write}\ p\ \langle \text{Leaf}, \text{Leaf} \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad n \leftarrow \textbf{write}\ n\ \langle \text{True}, p \rangle\ \textbf{;} \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ t\ \texttt{node}(n)\ )
\end{array}
$$

Let's Program: Union of Binary Tries

$s : \text{trie}, t : \text{trie} \vdash \text{union} :: (u : \text{trie})$
union $u\ s\ t =$
  read s (leaf(_) $\Rightarrow$ **copy** $u\ t$
      | node(m) $\Rightarrow$ **read** m ($\langle b, p \rangle \Rightarrow$ **read** p ($\langle s_0, s_1 \rangle \Rightarrow$
  read t (leaf(_) $\Rightarrow$ **copy** $u\ s$
      | node(n) $\Rightarrow$ **read** n ($\langle c, q \rangle \Rightarrow$ **read** q ($\langle t_0, t_1 \rangle \Rightarrow$
  $d \leftarrow$ or $d\ b\ c$ **;**
  $u_0 \leftarrow$ union $u_0\ s_0\ t_0$ **;**
  $u_1 \leftarrow$ union $u_1\ s_1\ t_1$ **;**
  $r \leftarrow$ **write** $r\ \langle u_0, u_1 \rangle$ **;**
  $o \leftarrow$ **write** $o\ \langle d, r \rangle$ **;**
  write u node(o))))))

$x : \text{bin}, t : \text{trie} \vdash \text{insert} :: (u : \text{trie})$
insert $u\ x\ t =$
  $s \leftarrow$ singleton $s\ x$ **;**
  union $u\ s\ t$ *% pipeling possible here*

Outline

- From Natural Deduction (ND) to Semi-Axiomatic Sequent Calculus (SAX)
- Programming in SAX
- Cut Elimination and Snips in SAX
- Data Layout and SAX with Snips (SNAX)
- Examples Revisited

What is the Layout of a Trie?

- The standard "abstract" version requires many cells

  $\text{trie} = (\texttt{leaf} : 1) + (\texttt{node} : \text{bool} \otimes (\text{trie} \otimes \text{trie}))$

  $!\text{cell Unit } \langle\,\rangle$

  $!\text{cell } t\ \texttt{leaf}(\text{Unit})$
  or
  $!\text{cell } t\ \texttt{node}(n)$
  $!\text{cell } n\ \langle b, p\rangle$
  $!\text{cell } b\ \texttt{false}(\text{Unit})$ or $!\text{cell } b\ \texttt{true}(\text{Unit})$
  $!\text{cell } p\ \langle t_0, t_1\rangle$

- Alternative "flat" layout (cell size as subscript)

  $!\text{cell}_4\ t\ [\texttt{leaf}]$
  or $!\text{cell}_4\ t\ [\texttt{node} \cdot \texttt{false} \cdot t_0 \cdot t_1]$
  or $!\text{cell}_4\ t\ [\texttt{node} \cdot \texttt{true} \cdot t_0 \cdot t_1]$

- How do we get there?

Return to Logic for Inspiration

- In SAX, standard cut elimination fails
- Example

$$\dfrac{\dfrac{}{B, C \vdash B \otimes C}\ \otimes X \qquad \dfrac{}{A, B \otimes C \vdash A \otimes (B \otimes C)}\ \otimes X}{A, B, C \vdash A \otimes (B \otimes C)}\ \mathsf{cut}_{B \otimes C}$$

- In code

$a : A, b : B, c : C \vdash f :: (d : A \otimes (B \otimes C))$
$f\ d\ a\ b\ c =$
$\quad bc \leftarrow \textbf{write}\ bc\ \langle b, c \rangle$
$\quad \textbf{write}\ d\ \langle a, bc \rangle$

A New Cut-Free Form

- Notice: $B \otimes C$ is a subformula of $A \otimes (B \otimes C)$
- Therefore, $B \otimes C$ is eligible for a cut that preserves the subformula property
  - Special case of an analytic cut
  - Eligible formulas are underlined
  - A cut with an eligible formula is a snip
- Example revisited

$$\dfrac{\dfrac{}{\underline{B}, \underline{C} \vdash B \otimes C}\ \otimes X \qquad \dfrac{}{\underline{A}, \underline{B \otimes C} \vdash A \otimes (B \otimes C)}\ \otimes X}{\underline{A}, \underline{B}, \underline{C} \vdash A \otimes (B \otimes C)}\ \text{snip}_{B \otimes C}$$

- **Theorem** [DeYoung, Pf, Pruiksma'20] Cut-free SAX proofs (possibly with snips) satisfy the subformula property
- **Theorem** [DeYoung, Pf, Pruiksma'20] If $\Gamma \vdash A$ in SAX, then there is a cut-free proof of $\Gamma \vdash A$ (possibly with snips).

Eligibility in SAX, Positive Connectives

- We can (implicitly) ignore that a formula is eligible
- We leave weakening and contraction implicit for conciseness
- Positive connectives: right rules become axioms

$$\frac{}{\underline{A} \vdash A \vee B}\ \vee X_1 \qquad \frac{}{\underline{B} \vdash A \vee B}\ \vee X_2 \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L$$

$$\frac{}{\underline{A}, \underline{B} \vdash A \otimes B}\ \otimes X \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ \otimes L$$

$$\frac{}{\cdot \vdash 1}\ 1X \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C}\ 1L$$

$$\frac{\Gamma \vdash A \quad \Gamma, \underline{A} \vdash C}{\Gamma \vdash C}\ \mathsf{snip}^+$$

Eligibility in SAX, Negative Connectives

- Negative connectives (left rules become axioms)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \supset R \qquad \frac{}{\underline{A}, A \supset B \vdash \underline{B}} \supset X$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge R \qquad \frac{}{A \wedge B \vdash \underline{A}} \wedge X_1 \quad \frac{}{A \wedge B \vdash \underline{B}} \wedge X_2$$

$$\frac{\Gamma \vdash \underline{A} \quad \Gamma, A \vdash C}{\Gamma \vdash C} \text{snip}^-$$

- General rules

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C} \text{cut} \qquad \frac{}{A \vdash A} \text{id}$$

Outline

- From Natural Deduction (ND) to Semi-Axiomatic Sequent Calculus (SAX)
- Programming in SAX
- Cut Elimination and Snips in SAX
- Data Layout and SAX with Snips (SNAX)
- Examples Revisited

SNAX and Layout

- Consider
$$\frac{\Gamma \vdash A \quad \Gamma, \underline{A} \vdash C}{\Gamma \vdash C}\ \mathsf{snip}^{+}$$
- Because $\underline{A}$ is a subformula of C, it denotes an address at an offset from the address of C
- Snips do not allocate, but provide the location of $\underline{A}$
- Example, with $|A| = |B| = 1$ (where $|A| =$ size of A)

$$\dfrac{\dfrac{}{(\underline{\alpha{+}1 : B}), (\underline{\alpha{+}2 : C}) \vdash (\alpha{+}1 : B \otimes C)}\ \otimes X \quad \dfrac{}{(\underline{\alpha : A}), (\underline{\alpha{+}1 : B \otimes C}) \vdash (\alpha : A \otimes (B \otimes C))}\ \otimes X}{(\underline{\alpha : A}), (\underline{\alpha{+}1 : B}), (\underline{\alpha{+}2 : C}) \vdash (\alpha : A \otimes (B \otimes C))}\ \mathsf{snip}$$

- Drop eligibility from SAX rules for negative types because we do not model layout of continuations

Pointer Values

- Recall

  trie $= (\texttt{leaf} : 1) + (\texttt{node} : \text{bool} \otimes (\text{trie} \otimes \text{trie}))$
- Both sums and pairs are positive—size would be unbounded
- Introduce positive $\downarrow A$ representing pointers
  - Logically, $\downarrow A \equiv A$
  - Computationally, $\downarrow a$ is small value of type $\downarrow A$
  - A is not eligible since $a : A$ not at a fixed offset from $b : \downarrow A$.

$$\frac{}{A \vdash \downarrow A}\ \downarrow X \qquad \frac{\Gamma, A \vdash C}{\Gamma, \downarrow A \vdash C}\ \downarrow L$$

- We **write** and **read** small values $\downarrow a$

$$\frac{}{a : A \vdash \textbf{write}\ b\ \downarrow a :: (b : \downarrow A)}\ \downarrow X \qquad \frac{\Gamma, x : A \vdash P :: (w : C)}{\Gamma, b : \downarrow A \vdash \textbf{read}\ b\ (\downarrow x \Rightarrow P) :: (w : C)}\ \downarrow L$$

- Recursion in type definitions must be guarded by a shift $\downarrow$.
- $\downarrow$ also includes negative types $(\supset, \wedge)$ in positive ones

Examples Revisited

- Pointer tag $\downarrow$ takes no space at runtime
- cell_n means cell of size n
- Nested pairs
  - Mapping from SAX layout

    $\downarrow A \otimes \downarrow(\downarrow B \otimes \downarrow C)$ $\quad$ $!\mathsf{cell}_2\ c\ [\downarrow c_A \cdot \downarrow d]$

    $!\mathsf{cell}_2\ d\ [\downarrow c_B \cdot \downarrow c_C]$
  - Identical flat layouts in SNAX (where $n = |A| + |B| + |C|$)

    $A \otimes (B \otimes C)$ $\quad$ $!\mathsf{cell}_n\ c\ [V_A \cdot V_B \cdot V_C]$

    $(A \otimes B) \otimes C$ $\quad$ $!\mathsf{cell}_n\ c\ [V_A \cdot V_B \cdot V_C]$

- Unit and Booleans

  $!\mathsf{cell}_0$ Unit [ ]

  bool $= (\texttt{false} : 1) + (\texttt{true} : 1)$

  $!\mathsf{cell}_1$ False [`false`]

  $!\mathsf{cell}_1$ True [`true`]
- Note $|1| = 0$

Examples Revisited: Tries

- Consider two different layout choices; others are possible
- Pointers to subtries

$\mathsf{trie} = (\texttt{leaf} : 1) + (\texttt{node} : \mathsf{bool} \otimes ({\downarrow}\mathsf{trie} \otimes {\downarrow}\mathsf{trie}))$

$$
\begin{array}{ll}
 & !\mathsf{cell}_4\ t\ [\texttt{leaf} \cdot \square \cdot \square \cdot \square] \\
\text{or} & !\mathsf{cell}_4\ t\ [\texttt{node} \cdot \texttt{false} \cdot {\downarrow}t_0 \cdot {\downarrow}t_1] \\
\text{or} & !\mathsf{cell}_4\ t\ [\texttt{node} \cdot \texttt{true} \cdot {\downarrow}t_0 \cdot {\downarrow}t_1]
\end{array}
$$

- Pointers to nodes

$\mathsf{trie} = (\texttt{leaf} : 1) + (\texttt{node} : {\downarrow}\mathsf{node})$
$\mathsf{node} = \mathsf{bool} \otimes (\mathsf{trie} \otimes \mathsf{trie})$

$$
\begin{array}{ll}
 & !\mathsf{cell}_2\ t\ [\texttt{leaf} \cdot \square] \\
\text{or} & !\mathsf{cell}_2\ t\ [\texttt{node} \cdot {\downarrow}n]
\end{array}
$$

$$
\begin{array}{ll}
 & !\mathsf{cell}_5\ n\ [\texttt{false} \cdot \texttt{leaf} \cdot \square \cdot \texttt{leaf} \cdot \square] \\
\text{or} & !\mathsf{cell}_5\ n\ [\texttt{true} \cdot \texttt{node} \cdot {\downarrow}n_0 \cdot \texttt{node} \cdot {\downarrow}n_1] \\
\text{or} & \ldots
\end{array}
$$

- Some space optimizations on sums may apply

Layout Rules without Process Terms (Positive Connectives)

$$\frac{(k \in L)}{(\underline{a{+}1 : A_k}) \vdash a : \Sigma_{\ell \in L}(\ell : A_\ell)}\ {+}X \qquad \frac{\Gamma, (a{+}1 : A_\ell) \vdash w : C \quad (\text{for all } \ell \in L)}{\Gamma, (a : \Sigma_{\ell \in L}(\ell : A_\ell)) \vdash w : C}\ {+}L$$

$$\frac{}{(\underline{a : A}), (\underline{a{+}|A| : B}) \vdash a : A \otimes B}\ {\otimes}X \qquad \frac{\Gamma, (a : A), (a{+}|A| : B) \vdash w : C}{\Gamma, (a : A \otimes B) \vdash w : C}\ {\otimes}L$$

$$\frac{}{\cdot \vdash a : 1}\ 1X \qquad \frac{\Gamma \vdash w : C}{\Gamma, a : 1 \vdash w : C}\ 1L$$

$$\frac{}{\Gamma, b : A \vdash a : {\downarrow}A}\ {\downarrow}X \qquad \frac{\Gamma, y : A \vdash w : C}{\Gamma, a : {\downarrow}A \vdash w : C}\ {\downarrow}L$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, (\underline{a : A}) \vdash w : C}{\Gamma \vdash w : C}\ \text{snip} \qquad \frac{\Gamma \vdash \alpha : A \quad \Gamma, \alpha : A \vdash w : C \quad \alpha \text{ fresh}}{\Gamma \vdash w : C}\ \text{cut}$$

Cut, Snip, and Identity

- Cut/allocate requires size or type

$$\frac{\Gamma \vdash P :: (\alpha : A) \quad \Gamma, \alpha : A \vdash Q :: (c : C) \quad (\alpha \text{ fresh})}{\Gamma \vdash \alpha_{|A|} \leftarrow P \, \mathbf{;} \, Q :: (c : C)} \text{ cut}$$

- Snip no longer allocates memory

$$\frac{\Gamma \vdash P :: (a : A) \quad \Gamma, (\underline{a : A}) \vdash Q :: (c : C)}{\Gamma \vdash P \, \mathbf{;} \, Q :: (c : C)} \text{ snip}$$

- Identity/copy requires size or type

$$\frac{}{b : A \vdash \mathbf{copy}_{|A|} \; a \; b :: (a : A)} \text{ id}$$

- Cut and snip can still be parallel, call-by-value, or call-by-need

Disappearing Act

- Eligibility is sharpened
  - We can no longer silently drop or ignore it
  - Instead, we use snip with identity

$$\dfrac{\dfrac{}{b : A \vdash \mathbf{copy}_{|A|}\ a\ b :: (a : A)}\ \text{id} \qquad \Gamma, \underline{a : A} \vdash P :: (w : C)}{\Gamma, b : A \vdash \mathbf{copy}_{|A|}\ a\ b\ \mathbf{;}\ P :: (w : C)}\ \text{snip}$$

- $\otimes X$, $\otimes L$, $1X$, and $1L$ just calculate addresses and have no operational significance!
- $+X$, $+L$, $\downarrow X$, $\downarrow R$ remain

Outline

- From Natural Deduction (ND) to Semi-Axiomatic Sequent Calculus (SAX)
- Programming in SAX
- Cut Elimination and Snips in SAX
- Data Layout and SAX with Snips (SNAX)
- Examples Revisited
- Conclusion

Example Revisited: Booleans

$\mathsf{bool} = (\texttt{false} : 1) + (\texttt{true} : 1)$

$b : \mathsf{bool} \vdash \mathsf{neg} :: (c : \mathsf{bool})$
$\mathsf{neg}\ c\ b =$
$\quad \textbf{read}\ b\ (\ \texttt{false} \Rightarrow \textbf{write}\ c\ \texttt{true}$
$\qquad\qquad \mid \texttt{true} \Rightarrow \textbf{write}\ c\ \texttt{false}\ )$

$b : \mathsf{bool}, c : \mathsf{bool} \vdash \mathsf{or} :: (d : \mathsf{bool})$
$\mathsf{or}\ d\ c\ b =$
$\quad \textbf{read}\ b\ (\ \texttt{false} \Rightarrow \textbf{copy}_{|\mathsf{bool}|}\ d\ c \qquad \%\ |\mathsf{bool}| = 1$
$\qquad\qquad \mid \texttt{true} \Rightarrow \textbf{write}\ d\ \texttt{true}\ )$

Size Calculation and Copying

- For simplicity, tags and pointers all have size 1

$$
\begin{array}{lcl}
|1| & = & 0 \\
|A \otimes B| & = & |A| + |B| \\
|\Sigma_{\ell \in L}(\ell : A_\ell)| & = & 1 + \max_\ell(A_\ell) \\
|{\downarrow}A| & = & 1
\end{array}
$$

- Type-directed definition of copying is shallow η-expansion

  $s : A \vdash \mathsf{copy}_A :: (d : A)$

  $\mathsf{copy}_1\ d\ s = (noop)$
  $\mathsf{copy}_{A \otimes B}\ d\ s = \mathsf{copy}_A\ d\ s\ \mathbf{;}\ \mathsf{copy}_B\ (d + |A|)\ (s + |A|)$
  $\mathsf{copy}_{\sum_{\ell \in L}(\ell : A_\ell)}\ d\ s = \mathbf{read}\ s\ (\ell \Rightarrow \mathbf{write}\ d\ \ell\ \mathbf{;}\ \mathsf{copy}_{A_\ell}\ (d+1)\ (s+1))_{\ell \in L}$
  $\mathsf{copy}_{\downarrow A}\ d\ s = \mathbf{read}\ s\ ({\downarrow}x \Rightarrow \mathbf{write}\ d\ {\downarrow}x)$

- May be implemented more efficiently

Example Revisited: Binary Numbers

$$\mathsf{bin} = (\mathtt{b0} : {\downarrow}\mathsf{bin}) + (\mathtt{b1} : {\downarrow}\mathsf{bin}) + (\mathtt{e} : 1)$$

$$\begin{array}{l}
\mathsf{Six} : \mathsf{bin} = \\
\quad x_0 \leftarrow \textbf{write}\ x_0\ \mathtt{e}\ \textbf{;} \\
\quad x_1 \leftarrow (\ \textbf{write}\ x_1\ \mathtt{b1}\ \textbf{;}\ \textbf{write}\ (x_1{+}1)\ {\downarrow}x_0\ )\ \textbf{;} \\
\quad x_2 \leftarrow (\ \textbf{write}\ x_2\ \mathtt{b1}\ \textbf{;}\ \textbf{write}\ (x_2{+}1)\ {\downarrow}x_1\ )\ \textbf{;} \\
\quad \textbf{write}\ \mathsf{Six}\ \mathtt{b0}\ \textbf{;}\ \textbf{write}\ (\mathsf{Six}{+}1)\ {\downarrow}x_2
\end{array}$$

$$\begin{array}{l}
x : \mathsf{bin} \vdash \mathsf{succ} :: (y : \mathsf{bin}) \\
\mathsf{succ}\ y\ x = \\
\quad \textbf{read}\ x\ (\ \mathtt{b0} \Rightarrow \textbf{write}\ y\ \mathtt{b1}\ \textbf{;} \\
\qquad\qquad\qquad \textbf{copy}_{|{\downarrow}\mathsf{bin}|}\ (y{+}1)\ (x{+}1) \qquad \%\ |{\downarrow}\mathsf{bin}| = 1 \\
\qquad\quad \mid \mathtt{b1} \Rightarrow \textbf{read}\ (x{+}1)\ (\ {\downarrow}x' \Rightarrow \\
\qquad\qquad\qquad y' \leftarrow \mathsf{succ}\ y'\ x'\ \textbf{;} \\
\qquad\qquad\qquad \textbf{write}\ y\ \mathtt{b0}\ \textbf{;} \\
\qquad\qquad\qquad \textbf{write}\ (y{+}1)\ {\downarrow}y'\ ) \\
\qquad\quad \mid \mathtt{e} \Rightarrow \textbf{write}\ y\ \mathtt{e}\ )
\end{array}$$

Example Revisited: Tries

$$
\begin{array}{l}
\text{trie} = (\texttt{leaf} : 1) + (\texttt{node} : \text{bool} \otimes (\downarrow\text{trie} \otimes \downarrow\text{trie})) \\
x : \downarrow\text{bin} \vdash \text{singleton} :: (t : \downarrow\text{trie}) \\
\text{singleton}\ t\ x = \\
\quad \textbf{read}\ x\ (\downarrow n \Rightarrow \\
\quad \textbf{read}\ n\ (\ \texttt{b0} \Rightarrow m \leftarrow (\ \textbf{write}\ m\ \texttt{node}\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}1)\ \texttt{false}\ ; \\
\qquad\qquad\qquad\qquad\quad \text{singleton}\ (m{+}2)\ (n{+}1)\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}3)\ \text{Leaf}\ )\ ; \\
\qquad\qquad\qquad \textbf{write}\ t\ \downarrow m \\
\qquad\quad \mid \texttt{b1} \Rightarrow m \leftarrow (\ \textbf{write}\ m\ \texttt{node}\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}1)\ \texttt{false}\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}2)\ \text{Leaf}\ ; \\
\qquad\qquad\qquad\qquad\quad \text{singleton}\ (m{+}3)\ (n{+}1)\ )\ ; \\
\qquad\qquad\qquad \textbf{write}\ t\ \downarrow m \\
\qquad\quad \mid \texttt{e} \Rightarrow \ m \leftarrow (\ \textbf{write}\ m\ \texttt{node}\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}1)\ \texttt{true}\ ; \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}2)\ \text{Leaf} \\
\qquad\qquad\qquad\qquad\quad \textbf{write}\ (m{+}3)\ \text{Leaf}\ )\ ; \\
\qquad\qquad\qquad \textbf{write}\ t\ \downarrow m\ )\ )
\end{array}
$$

Example Revisited: Union of Binary Tries

```
s : ↓trie, t : ↓true ⊢ union :: (u : ↓trie)
union u s t =
  read s (↓s′ ⇒ read s′ ( leaf ⇒ copy_|↓trie| u t            % |↓_| = 1
                        | node ⇒
  read t (↓t′ ⇒ read t′ ( leaf ⇒ copy_|↓trie| u s
                        | node ⇒
  d ← or d (s′+1) (t′+1) ;
  u0 ← union u0 (s′+2) (t′+2) ;
  u1 ← union u1 (s′+3) (t′+3) ;
  u′ ← ( write u′ node ;
         write (u′+1) d ;
         write (u′+2) u0 ;
         write (u′+3) u1 ) ;
  write u ↓u′ ) ) ) )

x : ↓bin, t : ↓trie ⊢ insert :: (u : ↓trie)
insert u x t =
  s ← singleton s x ;
  union u s t            % pipelining possible here!
```

Dynamics Revisited

- We can still give a parallel semantics
  - Synchronize only on sums and shifts (pairs and unit are silent)
  - Must be able to recognize an unwritten tag or pointer
  - Faithful to SAX since the straightforward translation inserts a shift before every sub-type
- Similarly for call-by-need schedule
- Call-by-value schedule guarantees a cell is written before any attempt to read from it

Layout Identities $A \equiv B$

- In a functional language: $A \leq B$ if $v : A$ implies $v : B$
- Example:

  $\mathsf{nat} = (\mathtt{z} : 1) + (\mathtt{s} : \mathsf{nat})$
  $\mathsf{even} = (\mathtt{z} : 1) + (\mathtt{s} : \mathsf{odd})$
  $\mathsf{odd} = (\mathtt{s} : \mathsf{even})$

  $\mathsf{even} \leq \mathsf{nat}$ and $\mathsf{odd} \leq \mathsf{nat}$
- In SNAX, a final configuration $\mathcal{F}$ contains only !cell a V and !cell b K.
- In SNAX: $A \leq B$ if $\mathcal{F} :: (c : A)$ implies $\mathcal{F} :: (c : B)$.
- Additional examples for $A \equiv B$ (that is, $A \leq B$ and $B \leq A$)
  - $A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C$
  - $A \otimes 1 \equiv A \equiv 1 \otimes A$
  - $((\mathtt{false} : 1) + (\mathtt{true} : 1)) \otimes A \equiv (\mathtt{false} : A) + (\mathtt{true} : A)$
  - $(A + B) \otimes C \equiv (A \otimes C) + (B \otimes C) \quad (\text{if } |A| = |B|)$

Internal Pointers and Polymorphism

- Internal pointers

  $a : A \vdash \mathsf{self} :: (p : A \otimes {\downarrow}A)$
  $\mathsf{self}\ p\ a =$
  $\quad \mathbf{copy}_{|A|}\ p\ a\ ;$
  $\quad \mathbf{write}\ (p + |A|)\ p$

- Boxed vs. unboxed representations of polymorphism

  $\mathsf{list}_1\ A = (\mathtt{nil} : 1) + (\mathtt{cons} : {\downarrow}A \otimes {\downarrow}\mathsf{list}_1\ A) \qquad \%\ |\mathit{list}_1\ A| = 3$
  $\mathsf{list}_2\ A = (\mathtt{nil} : 1) + (\mathtt{cons} : A \otimes {\downarrow}\mathsf{list}_2\ A) \qquad \%\ |\mathit{list}_2\ A| = |A| + 2$
  $\mathsf{list}_3\ A = (\mathtt{nil} : 1) + (\mathtt{cons} : {\downarrow}\mathsf{node}_3\ A) \qquad \%\ |\mathit{list}_3\ A| = 2$
  $\mathsf{node}_3\ A = {\downarrow}A \otimes \mathsf{list}_3\ A \qquad \%\ |\mathsf{node}_3\ A| = 3$

Summary

- Fix propositional intuitionistic logic
- Natural deduction (ND)
  - Introduction and elimination rules
  - Computation by substitution
  - "Large" values v
- Semi-axiomatic sequent calculus (SAX)
  - Axioms for non-invertible rules ($\otimes R, 1R, \vee R, \supset L, \wedge L$)
  - Cut elimination and subformula property via snips
  - Computation via futures (write-once shared memory with concurrent threads)
  - Call-by-value and call-by-need as particular schedules
- Semi-axiomatic sequent calculus with snips (SNAX)
  - Layout for positive types ($\otimes, 1, \Sigma$) is flat
  - Type $\downarrow A$, logically equivalent to A
  - Computationally, value of type $\downarrow A$ is an address
  - Pairs ($\otimes$) and unit (1) become computationally irrelevant
  - Admits futures, call-by-value, call-by-need

Further Related Work

- [Morrisett; PhD 1995] Compiling with Types
  - Data layout is significant for performance
  - Not explicit in the type
- [Tarditi, Morrisett, Cheng, Stone, Harper, Lee; PLDI 1996] TIL: A Type-Directed Optimizing Compiler for ML
  - Importance of typed intermediate languages
- [Morrisett, Walker, Crary, Glew; TOPLAS 1999] From System F to Typed Assembly Language
  - Richer type system (e.g., polymorphism, closures)
  - Lower-level code (RISC-like instruction set)
  - Connection to high-level proof systems only via translation
  - Continuation-passing vs. destination-passing
  - No parallelism, layout not a point of emphasis

Further Related Work

- [Petersen, Harper, Crary, Pf; POPL 2003] A Type Theory for Memory Allocation and Data Layout
  - Approach based on ordered logic (no weakening, contraction, exchange)
  - Worked well as far as it went
  - Limitation: adjacency is not an intrinsic property of ordered logic
- Many works on data description languages

Outlook

- SNAX satisfies the usual preservation and progress (in progress)
- Logical/type-theoretic foundation enable generalizations
- Indexed types to allow flat layouts ("array")

  $\mathsf{seq}\ A\ n = (\texttt{nil} : (n = 0) \otimes 1) + (\texttt{cons} : (n > 0) \otimes A \otimes \mathsf{seq}\ A\ (n-1))$
  $\mathsf{sequence}\ A = (n : \mathit{nat}) \otimes \mathsf{seq}\ A\ n$
- Polymorphism
- Concrete representation of values of negative type ("closures")

Thanks

- SNAX Joint work with Henry DeYoung
- SAX joint work with Henry DeYoung and Klaas Pruiksma
- Thanks to Stephanie Balzer, Luís Caires, Ankush Das, Farzaneh Derakshan, Siva Somayyajula, Bernardo Toninho