ADVENTURES IN TIME & SPACE

Jim Royer
Syracuse University

Joint work with
Norman Danner
Wesleyan University

GeoCal’06

THE PROBLEM

Solve for $X$ in

$$(\text{PCF} - \text{fix}) + X = \begin{cases} \text{a serviceable programming language} \\
\text{for higher-type polynomial time} \end{cases}$$

where

- $X$’s constraints are via typing (implicit complexity)
- serviceable $\approx$ lots of algorithms are directly expressible
- $X$ includes something “close to” fix

PCF =

- simply-typed $\lambda$-calc. + basic string ops + fixed pt. comb. (call-by-value)

higher-type poly-time =

- Mehlhorn’s and Cook-Urquhart’s basic feasible functionals (more later)
WHY THIS PROBLEM?

It is at a crossroads of many interesting paths.

► Efficiency as a safety property
  E.g., for proof-carrying code

► Street-level view of higher types
  E.g., Bird’s chapter on efficiency:
  • Cunning, sound program transforms,
  • These can be used achieve efficiency
  • But how to do so? . . . only heuristic advice and (wonderful) examples.

► Applications in Cryptography and Learning
  E.g., Transformations on pseudo-random generators

► . . .

► Insight into higher-type complexity
TWO APPROACHES TO HIGHER TYPE POLY-TIME

Higher types over ordinary polynomial time

Focus: Bringing higher types & other modern accoutrements into the programming type-1 poly-time functions.

Hard Part: Gracefully handling the inevitable restrictions.

Examples: Aehlig, Bellantoni, Hofmann, Niggl, Schwichtenberg, . . .

Polynomial time over higher type objects

Focus: Bringing complexity theoretic concerns and questions into the realm of higher types. (Why bother?)

Hard Part: Re-thinking the world.

E.g., What is the “computational complexity” of \((f, g) \mapsto f \circ g\)?

Focus of on type-level 2 (Type-levels 3, 4, 5, . . . ? another day)

Examples: Kapron-Cook, Seth, . . . , this talk
**TYPE-2 POLY-TIME I: SIZES**

**Conventions**

- $\mathbb{N} = \{0, 1\}^* \equiv \text{values}$, to be computed over
- $\omega = \{0\}^* \equiv \text{tallies}$, results of size measurements

**Measuring sizes**

- $|x| = \text{def } 0^\ell$, where $\ell = \text{length of } x \in \mathbb{N}$. E.g., $|101| = 000$.
- For $f: \mathbb{N} \rightarrow \mathbb{N}$, we have $|f|: \omega \rightarrow \omega \ni$ for each $n \in \omega$,
  $$|f|(n) = \text{def } \max(\{|f(x)| : |x| \leq n\}).$$ (Kapron-Cook)
  E.g., $|\lambda x \in \mathbb{N}. (x \oplus x)| = \lambda n \in \omega \cdot 2n$.
- and so on for type-2 functions, types, type-contexts, \ldots

$| \cdot | : \text{the realm of values} \xrightarrow{\text{"functorially"}} \text{the realm of sizes}$
Example  For \( C = \lambda f, g . \lambda x . f(g(x)) \):

\[
|C(f, g)(x)| \leq \frac{|f|(|g||x|)|}{\text{poly over } |f|, |g|, |x|}.
\]

The second-order polynomials: Syntax (Kapron-Cook)

\( \text{the simply-typed } \lambda \text{-calculus over base type } T \) \( \mid \text{type levels } \leq 2 \)

+ tally constants and type-1 binary ops \( +, \ast, \text{ and } \lor \)

The second-order polynomials: Semantics \( (\mathcal{L}[\cdot]) \)

\[
\begin{align*}
\mathcal{L}[T] &= \text{def } \omega, \\
\mathcal{L}[T \rightarrow T] &= \text{def } \omega \Rightarrow \omega, \\
\mathcal{L}[x \lor y] &= \text{def } \max(x, y), \\
\mathcal{L}[x + y] &= \text{def } x + y.
\end{align*}
\]
THE KAPRON-COOK THEOREM

\( F: (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N} \) is a basic feasible functional iff

- there is a machine \( M \) and
- there is second-order poly \( q \in \exists \)

\[ \forall f, x \quad M \text{ on input } (f, x) \]
- (a) outputs \( F(f, x) \), and
- (b) runs within time \( q(|f|, |x|) \).

Machines & costs: anything “sensible” works

We use a CEK-abstract machine for PCF with a particular cost measure.

Now back to our problem
INGREDIENTS OF X: TIERED BASE TYPES

(\text{PCF} - \text{fix}) + X = \text{a nice PL for type-level 2 poly-time}

Labels:

\[ \varepsilon < \Diamond < \Diamond\Diamond < \Diamond\Diamond\Diamond < \Diamond\Diamond\Diamond\Diamond < \ldots \]

Base Types:

\[ N\varepsilon \leq: N\Diamond \leq: N\Diamond\Diamond \leq: N\Diamond\Diamond\Diamond \leq: \ldots \]

Intuitive Interpretation: The labels describe size bounds

\[ x: N\varepsilon \approx |x| \leq |\text{some input string}| \]
\[ x: N\Diamond \approx |x| \leq \text{poly}(|\text{some input string}|) \]
\[ x: N\Diamond\Diamond \approx |x| \leq |f|(\text{poly}(|\text{some input string}|)) \]
\[ x: N\Diamond\Diamond\Diamond \approx |x| \leq \text{poly}(|f|(\text{poly}(|\text{some input string}|))) \]
\[ x: N\Diamond\Diamond\Diamond\Diamond \approx |x| \leq |f|(\text{poly}(|f|(\text{poly}(|\text{some input string}|)))) \]

\[ \ldots \]

(Bellantoni & Cook connection: \( N\varepsilon \approx \text{normal} \) \( N\Diamond \approx \text{safe} \))
(PCF − fix) + X = a nice PL for type-level 2 poly-time

Arrow types = the simple types over the base types
E.g., \( f : N^\Diamond \rightarrow N^{\Diamond\Diamond} \)

But how to type \( f(f(x)) \)?

Subsumption + covariant shifting of arrow types
E.g., \( f : N^\Diamond \rightarrow N^{\Diamond\Diamond} \) \(\Rightarrow\) \( f : N^{\Diamond\Diamond} \rightarrow N^{\Diamond\Diamond\Diamond\Diamond} \)

But, we need to control more than size . . .
(PCF − fix) + X = a nice PL for type-level 2 poly-time

**Clocked Recursion:** \text{crec} \ a \ (\lambda_r.f \ . \ E)

For constructive runtime bounds on programs, something equivalent to clocking is necessary.

**One use recursion:** \( f \) as above is affinely restricted

This allows us to handle “linear,” poly-depth recursions.

**Provides:** \( \text{Time}(m, \vec{n}) \leq \text{Time}(m - 1, \vec{n}) + \text{poly}(\vec{n}) \)

**Built on Plotkin and Barber’s DILL** \( (\text{E.g., } \Gamma; f: \sigma \vdash E: \sigma \to \sigma) \)

**Tail recursion:**

Mainly for simplicity.

Nearly everyone else uses primitive recursions.

Putting things together \ldots
ATR: AFFINE TAIL-RECURSION

Grammar of Raw Expressions

\[ E ::= K \mid (c_a E) \mid (d E) \mid (t_a E) \mid (\text{down } E E) \mid V \mid (E E) \mid (\lambda V . E) \mid (\text{if } E \text{ then } E \text{ else } E) \mid (\text{crec } K (\lambda_r V . E)) \]

\[ K ::= \{ 0, 1 \}^* \]

Rewrite Rules: \( (\oplus = \text{string concatenation}) \)

\[ (c_a x) \leadsto a \oplus x. \quad (d (a \oplus x)) \leadsto x. \quad (d \epsilon) \leadsto \epsilon. \]

\[ (t_a x) \leadsto \begin{cases} 0, & \text{if } x \text{ begins with } a; \\ \epsilon, & \text{otherwise}. \end{cases} \]

\[ (\text{down } x y) \leadsto \begin{cases} x, & \text{if } |x| \leq |y|; \\ \epsilon, & \text{otherwise}. \end{cases} \]

\[ (\text{if } x \text{ then } y \text{ else } z) \leadsto \begin{cases} y, & \text{if } x \neq \epsilon; \\ z, & \text{if } x = \epsilon. \end{cases} \]

more..
ATR: MORE REWRITE RULES

Call-By-Value $\beta$-Reduction

As usual

A Standard Call-By-Value Rewrite Rule for fix  
(Not part of ATR!)

$$\text{fix } (\lambda f . E) \leadsto E[f := (\text{fix } (\lambda f . E))].$$

The Rewrite Rule for crec

$$\text{crec } a (\lambda_r f . E) \leadsto \lambda \vec{v}. \text{ (if } |a| \leq |v_1| \text{ then } (E' \vec{v}) \text{ else } \epsilon)$$
with 
$$E' = E[f := \left(\text{crec } (0 \oplus a) (\lambda_r f . E)\right)].$$

$\blacktriangleright$ $a \approx$ the internal clock — that counts up.

$\blacktriangleright$ $0 \oplus a \approx$ a tick of the clock

$\blacktriangleright$ Typing constraints will make sure $|v_1|$ is bounded.
(Zero-I) \[ \Gamma; \Delta \vdash \epsilon : N_\varepsilon \]

(Const-I) \[ \Gamma; \Delta \vdash K : N_\diamond \]

(Int-Id-I) \[ \Gamma, v : \sigma; \Delta \vdash v : \sigma \]

(Aff-Id-I) \[ \Gamma; v : \gamma \vdash v : \gamma \]

(Subsumption) \[ \Gamma; \Delta \vdash E : \sigma \quad (\sigma \leq : \tau) \]

(op-I) \[ \Gamma; \Delta \vdash E : N_{\diamond d} \]

(Shift) \[ \Gamma; \Delta \vdash E : \sigma \quad (\sigma \propto \tau) \]

(op ranges over \(c_0, c_1, d, t_0, \) and \(t_1\))

(down-I) \[ \Gamma; \Delta \vdash E : N_L \quad \Gamma; \Delta' \vdash E' : N_{L'} \]

\[ \Gamma; \Delta, \Delta' \vdash (\text{down } E \ E') : N_{L'} \]

(if-I) \[ \Gamma; _\vdash E_0 : N_L \quad \Gamma; \Delta_1 \vdash E_1 : N_{L'} \quad \Gamma; \Delta_2 \vdash E_2 : N_{L'} \]

\[ \Gamma; \Delta_1 \cup \Delta_2 \vdash (\text{if } E_0 \ \text{then } E_1 \ \text{else } E_2) : N_{L'} \]
(→-I) \[ \frac{\Gamma, \nu: \sigma; \Delta \vdash E: \tau}{\Gamma; \Delta \vdash (\lambda \nu . E): \sigma \rightarrow \tau} \]

(→-E) \[ \frac{\Gamma; \Delta \vdash E_0: \sigma \rightarrow \tau \quad \Gamma; \_ \vdash E_1: \sigma}{\Gamma; \Delta \vdash (E_0 \ E_1): \tau} \]

(crec-I) \[ \frac{\_ \vdash K: N_\diamondsuit \quad \Gamma; f: \gamma \vdash E: \gamma}{\Gamma; \_ \vdash (\text{crec} \ K \ (\lambda r. f . E)): \gamma} \quad \left(\text{TailPos}(f, E) \text{ and } \gamma \in \mathcal{R}\right) \]

where:

\[ \text{TailPos}(f, E) \stackrel{\text{def}}{=} \left[ \text{Each occurrence of } f \text{ in } E \text{ is as the head of a tail call} \right] . \]

\[ \mathcal{R} \stackrel{\text{def}}{=} \{ (b_1, b_2, \ldots, b_k) \rightarrow b : b_1 \text{ and each } b_i \leq: b_1 \text{ is oracular} \} . \]

The oracular base types = \( N_\varepsilon, N_{\Box\Diamond}, N_{\Box\Diamond\Box}, N_{\Box\Diamond\Box\Box}, \ldots \)
EXAMPLE: REVERSE

\[ \text{reverse} : \mathbb{N}_\varepsilon \rightarrow \mathbb{N}_\Diamond = \]
\[ \lambda w . \ \text{letrec } f : \mathbb{N}_\varepsilon \rightarrow \mathbb{N}_\Diamond \rightarrow \mathbb{N}_\Diamond \rightarrow \mathbb{N}_\Diamond = \]
\[ \lambda b, x, r . \ \text{if } (t_0 x) \ \text{then } f b (d x) (c_0 r) \]
\[ \text{else if } (t_1 x) \ \text{then } f b (d x) (c_1 r) \]
\[ \text{else } r \]

\[ \text{in } f w w \varepsilon \]

\[ (\text{letrec } f = D \ \text{in } E) \equiv E[f := (\text{crec } \varepsilon (\lambda r.f . D))] \]

Recall:

\[ (t_a x) \equiv [x \ \text{starts with } a?] \]
\[ (c_a x) = a \oplus x. \]
\[ (d (a \oplus x)) = x. \]

\[ b - \text{programmer’s bound on the number of recursions} \]
EXAMPLE: PRIM. REC. ON NOTATION

\[ \text{prn: } (N_\Diamond \to N_\Diamond \to N_\Diamond) \to N_\varepsilon \to N_\Diamond = \]
\[ \lambda g, y \cdot \text{letrec } f : N_\varepsilon \to N_\Diamond \to N_\Diamond \to N_\Diamond \to N_\Diamond \to N_\Diamond = \]
\[ \lambda b, x, z, r . \]
\[ \text{if } (t_0 x) \text{ then } f b (d x) (c_0 z) (g (c_0 z) r) \]
\[ \text{else if } (t_1 x) \text{ then } f b (d x) (c_1 z) (g (c_1 z) r) \]
\[ \text{else } \quad r \]
\[ \text{in } f y \ (\text{reverse } y) \in (g \in \varepsilon) \]

where

\[ \left\{ \begin{array}{c}
\text{prn } g \in \varepsilon \leadsto g \in \varepsilon . \\
\text{prn } g (a \oplus y) \leadsto g (a \oplus y) \ (\text{prn } g y). \end{array} \right\} \quad (\ast) \]

\[ \text{As before} \]
\[ (\text{letrec } f = D \text{ in } E) \equiv E[f := (\text{crec } \varepsilon (\lambda_r f . D))] \quad \text{and} \]
\[ b - \text{programmer’s bound on the number of recursions} \]

\textbf{N.B.} The \textit{prn} functional as defined by \((\ast)\), \textbf{is not} a BFF.
\textbf{The side-conditions that tame} \((\ast)\) \textbf{are part of ATR’s semantics.}
kfun: (\(N \diamond \rightarrow N \Box \diamond\)) \(\rightarrow N_\varepsilon \rightarrow N_\varepsilon\) = // Computes K given below
\(\lambda f, x . \) letrec \(h: N \Box \diamond \rightarrow N_\varepsilon \rightarrow N_\varepsilon =\)
\(\lambda m, k . \) // Invariant: \(k \leq \text{len}(m)\) and \(|m| \leq |f|(|x|)\)
if \((k == x)\) or \((k == (\text{len } m))\)
then \(k\)
else \(h (\text{max } (f (k + 1)) m) (\text{down } (k + 1) x)\)
in \(h (f \varepsilon) \varepsilon\)

(Fixed from the POPL proceedings.)

where

- \(\text{len}(z)\) = the dyadic representation of the length of \(z\).
- \(K(f, x) = \begin{cases} (\mu k < x) \left[ k = \max_{i \leq k} \text{len}(f(i)) \right], & \text{if such a } k \text{ exists;} \\ x, & \text{otherwise;} \end{cases}\)
- various secondary functions are given (correctly typed) definitions

**N.B.** \(K\) is a BFF and the key example that lead to the Kapron-Cook Thm.
THEOREM

If $\Gamma; \Delta \vdash E : \sigma$,
then there is a $|\Gamma; \Delta| \vdash q_E : |\sigma| \ni$

$$|\mathcal{V}_{wt}[E] \rho| \leq |\mathcal{L}_{wt}[q_E] \rho|, \text{ for all } \rho \in V_{wt}[\Gamma; \Delta].$$

- 2nd-order polys are also typed. E.g., $|f| : T^\Diamond \to T^{\Box\Box}$
- $\mathcal{V}_{wt}[\cdot] = \text{the naive value semantics, pruned.}$
  where the naive semantics $\mathcal{V}[\cdot] = \text{a standard Scott semantics for PCF.}$
- $\mathcal{L}_{wt}[\cdot] = \text{the naive length semantics, pruned.}$
  where the naive semantics $\mathcal{L}[\cdot] = \text{as before.}$
- This pruning the subtlest part of the paper.

a few details . . .
Definitions

- \( \text{tail}(N_L) = N_L \).
- \( \text{tail}(\sigma \rightarrow \tau) = \text{tail}(\tau) \).

- \( \gamma \) is \textbf{predicative} when \( \gamma \) is a base type or else \( \gamma = \sigma \rightarrow \tau \) where \( \tau \) is predicative and \( \text{tail}(\sigma) \leq: \text{tail}(\tau) \).

- \( \sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow N_L \) is \textbf{flat} when \( (\exists i) \text{tail}(\sigma_i) = N_L \).

- \textbf{impredicative} \( \equiv \neg \text{predicative} \) \hspace{1cm} \textbf{strict} \( \equiv \neg \text{flat} \)

Examples

<table>
<thead>
<tr>
<th></th>
<th>predicative</th>
<th>impredicative</th>
</tr>
</thead>
<tbody>
<tr>
<td>strict</td>
<td>( N_\varepsilon \rightarrow N_\Diamond )</td>
<td>( N_\Diamond \rightarrow N_\varepsilon )</td>
</tr>
<tr>
<td>flat</td>
<td>( N_\Diamond \rightarrow N_\Diamond )</td>
<td>( N_\Diamond \rightarrow N_{\square \Diamond} \rightarrow N_\Diamond )</td>
</tr>
</tbody>
</table>

...and the point is?

The semantic interpretations of impredicative and flat types must be restricted.

Why?
TWO COUNTER-EXAMPLES

\[ E_1 : N_\varepsilon \rightarrow N_\diamond = \quad // \text{Assume } g_1 : N_\diamond \rightarrow N_\varepsilon. \]

\[ \lambda w . \text{let } h_1 : N_\diamond \rightarrow N_\diamond \rightarrow N_\diamond = \]

\[ \lambda x, y . \text{if } x \neq \epsilon \text{ then } (\text{dup } (g_1 y) (g_1 y)) \text{ else } w \]

\[ \text{in } \text{prn } h_1 w \]

\[ E_2 : N_\varepsilon \rightarrow N_\diamond = \quad // \text{Assume } g_2 : N_\diamond \rightarrow N_\diamond. \]

\[ \lambda w . \text{let } h_2 : N_\diamond \rightarrow N_\diamond \rightarrow N_\diamond = \]

\[ \lambda x, y . \text{if } x \neq \epsilon \text{ then } (g_2 y) \text{ else } w \]

\[ \text{in } \text{prn } h_2 w \]

where

\[ (\text{let } x = D \text{ in } E) \equiv E[x := D] \]

\[ (\text{prn } f (a \oplus y)) \leadsto (f (a \oplus y) (\text{prn } f y)) \]

\[ |(\text{dup } x y)| = |x| \cdot |y| \]

\[ (\text{prn } f \epsilon) \leadsto (f \epsilon \epsilon) \]

NOTE:

\[ \text{If } \rho_1(g_1) = \lambda z \in N. z, \]

then \[ |\mathcal{V}[E_1] \rho_1| = \lambda n \in \omega . n^{2^n}. \]

A normality violation.

\[ \text{If } \rho_2(g_2) = \lambda z \in N. z \oplus z, \]

then \[ |\mathcal{V}[E_2] \rho_2| = \lambda n \in \omega . n \cdot 2^n. \]

A safety violation.
PRUNING THE NAIVE SEMANTICS

For the impredicative types:

▶ Note:

bullet \(|(\text{down } x \ y)| \leq |y|\)
bullet \(|(\text{if } x \text{ then } y \text{ else } z)| \leq |y| \lor |z|\)

▶ So prune the semantics to make sure that all values have “strictly predicative upper bounds.”

For flat types:

▶ Recall Bellantoni & Cook’s poly-max bounds

\[|f(\vec{x}; \vec{y})| \leq p(|\vec{x}|) + \bigvee |\vec{y}|\]

▶ Flat types have to be bounded by higher-type analogues of the poly-max bounds.

▶ This is hard work and not for today.
|\mathcal{V}_{\text{wt}}[E] \rho| is an extensional property of $E$.

Whereas the time needed to evaluate $E$ is certainly not.

∴ We introduce $\mathcal{T}[E] = \text{time complexity of } E$.

With size, the $| \cdot |$ “functor” dictates the shape of things.

What happens with time? Some examples:
TIME COMPLEXITIES I: CURRYING AND TIME

Case 1: \( E : N_\varepsilon \)
\[ t \in \omega \quad \exists \quad t = \text{the time required to compute } E \text{'s value} \]

Think of this as time passed

Case 2: \( E : N_\varepsilon \to N_\Diamond \)
\[ t : \omega \to \omega \quad \exists \quad t(n) = \text{the time to evaluate } E \text{ on an size-} n \text{ input.} \]

Think of this as time in possible futures.

Case 3: \((E_0 E_1) : N_\varepsilon \to N_\Diamond\), where \( E_0 : N_\varepsilon \to N_\varepsilon \to N_\Diamond \) and \( E_1 : N_\varepsilon \)

\((E_0 E_1)\) has “time complexities” in the senses of both Cases 1 and 2.

So \((E_0 E_1)\) has a past and futures of interest.

Case 4: \( E_0 \) as above.

\( E_0 \) has “time complexities” in the senses of Cases 1, 2, & 3.

Case 5: Consider how type-2 terms complicate things.

Is it all clear now?
A time complexity for $E$ has two components, a cost and a potential.

The cost (a positive elm. of $\omega$) $\approx$ the cost to evaluate $E$.

The form of the potential depends on $E$'s type.

- For $E$ of base type, the potential $= \max(1, |\text{the value of } E|)$. (Why?)

- For $E : N_{\varepsilon} \rightarrow N_{\Diamond}$ we take the potential to be $p_E : \omega^2 \rightarrow \omega^2 \ni \begin{align*} p_E(c_{\text{arg}}, p_{\text{arg}}) &= (c_{\text{res}}, p_{\text{res}}) \text{ where} \\ * (c_{\text{arg}}, p_{\text{arg}}) &= \text{the t.c. of an argument, } A, \text{ to } E \\ * (c_{\text{res}}, p_{\text{res}}) &= \text{the t.c. of } (E A)'s \text{ result} \end{align*}$

- For $E$ of type $\sigma \rightarrow \tau$ our motto is:
  
  a potential for a thing of type $\sigma \rightarrow \tau$ is a map
from time complexities for things of type $\sigma$
to time complexities for things of type $\tau$.

Others have used the cost/pot. decomposition: Sands, Shultis, van Stone
A type translation For ATR types $N_L$, $\sigma$, $\tau$:

\[
\|\sigma\| = \text{def } T \times \langle\sigma\rangle \quad \text{the type of a t.c. of a type } \sigma \text{ thing}
\]
\[
\langle N_L \rangle = \text{def } T_L \quad \text{the type of a pot. of a type } N_L \text{ thing}
\]
\[
\langle \sigma \to \tau \rangle = \text{def } \|\sigma\| \to \|\tau\| \quad \text{the type of a pot. of a type } \sigma \to \tau \text{ thing}
\]

**DEFINITION** (The $T[\cdot]$-interpretation of ATR types)

\[
T[\sigma] = \text{def } L_{\text{wt}}[\|\sigma\|].
\]

Do interesting, sensible things live in these spaces?
1. We use our machine cost-model and the $\mathcal{T}[\sigma]$ spaces to build a model for the simply typed $\lambda$-calculus.

E.g., $\mathcal{T}$-application. Suppose:

\[(c_{\text{opr}}, p_{\text{opr}}) \in \mathcal{T}[\sigma \rightarrow \tau]\]
\[(c_{\text{arg}}, p_{\text{arg}}) \in \mathcal{T}[\sigma]\]
\[(c_{\text{res}}, p_{\text{res}}) \in \mathcal{T}[\tau], \text{ where } (c_{\text{res}}, p_{\text{res}}) = p_{\text{opr}}\left((c_{\text{arg}}, p_{\text{arg}})\right)\]

Then \((c_{\text{opr}}, p_{\text{opr}}) \star (c_{\text{arg}}, p_{\text{arg}}) =_{\text{def}} (c_{\text{opr}} + c_{\text{arg}} + c_{\text{res}} + 3, p_{\text{res}})\).

Similarly, for currying, environments, . . .

Note: Things are quite intensional here. E.g., The $\eta$-law fails.

2. Based on this model we define $\mathcal{T}[E]$ for $\text{ATR}^{-} (= \text{ATR sans crec})$ and show soundness & polytime boundedness. (Details shortly)

3. Then the affine types earn their keep . . .
**AFFINE DECOMPOSITION**

**DEFINITION**

\[(c_0, p_0) \uplus (c_1, p_1) = \text{def} \ (c_0 + c_1, p_0 \lor p_1)\]

**THEOREM**

Suppose \(\Gamma; f: \gamma \vdash E: b\). Then \(\forall \varrho \in \mathcal{T}[\Gamma; f: \gamma]:\)

\[
\mathcal{T}[E] \varrho \leq \mathcal{T}[E[f := \lambda \vec{x}. \epsilon]] \varrho \uplus (\mathcal{T}[f] \ast \vec{t}) \varrho
\]

where the \(\vec{t}\)'s are max's over appropriate subterms of \(E\).

This decomposition sets us up for solving the recurrences in . . .
THEOREM (Soundness and Polynomial Boundedness ATR)

(a) The $\mathcal{T}[\cdot]$ is sound for ATR wrt the costs for our machine model.

(b) Given $\Gamma; \Delta \vdash_{\text{ATR}} E : \gamma$, $\exists$ effectively $q_E$, a poly. time bound for $E$.

▶ polynomial time-boundedness $\approx \mathcal{T}[E]$ is not too big

▶ soundness $\approx \mathcal{T}[E]$ is not too small
$\approx \mathcal{T}[E]$ gives an upper bound on the cost of $E$ evaluation

▶ The poly-size boundedness results play a central role here.

Also

THEOREM (Computational Completeness)
All of the type-2 BFFs are ATR-computable.
▶ **ATR** is far from perfect: [audience inserts long list here]

▶ **ATR** is a demonstration of concept

- **ATR** and its complexity properties are treated via standard PL tools
- $\mathcal{V}_{wt}[\cdot]$, a denotational semantics for a ramified type system
- $\mathcal{T}[\cdot]$, a (machine dependent) semantics of time complexity.
- These semantics reveal complexity theoretic details.
- The type system, while ad hoc, has a **sound** intuitive story behind it. 
  ⊳ There is some hope programmers could think in it.
POSSIBLE EXTENSIONS

- Higher-type parameters in recursions (continuations)
- other recursion patterns (prim. rec., tree rec., etc.)
- Call-by-name/call-by-need
- Other machine+cost choices
- lists, trees, streams, . . .
- type-level 3 and beyond
- making the type system less ad hoc
- . . .

Next goal:
“1st order” call-by-(name/need) + streams of strings

Near term goal:
Go through Bird, Paulson, . . . and by annotating types and minor repro-
gramming show many/most programs are poly-time.
A Lisp programmer knows the value of everything, but the cost of nothing. — Alan Perlis

Let us work on removing every excuse a programmer might have for this.