# A Word Sampler for Well-Typed Functions

Breandan Considine

## Syntactic Terms

Consider a simply-typed, first-order functional programming language with function calls, conditionals, and binary operators:

```
FUN ::= fun f0 ( PRM ) : 𝕋 = EXP          INV ::= FID ( ARG )
PRM ::= PID : 𝕋 | PRM , PID : 𝕋           ARG ::= EXP | ARG , EXP
EXP ::= ⌜𝕋⌟ | PID | INV | IFE | OPX        OPR ::= + | * | < | ==
OPX ::= ( EXP OPR EXP )                   PID ::= p1 | ... | pk
IFE ::= if EXP { EXP } else { EXP }       FID ::= f0 | ... | fn
```

**Type universe.** We assume a finite set $\mathbb{T}$ (size $d$) with at least $\mathbb{B}, \mathbb{N}$, and an ambient global context $\Gamma$ of named functions $\texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau$.

## Static Semantics

Typing judgements are standard; we highlight just a few of them below:

$$\frac{\Gamma \vdash e_c : \quad \Gamma \vdash e_\top : \tau \quad \Gamma \vdash e_\bot : \tau}{\Gamma \vdash \texttt{if}\ e_c\ \texttt{\{}\ e_\top\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ e_\bot\ \texttt{\}} : \tau}\,\texttt{IFE}$$

$$\frac{\Gamma \vdash \texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau \quad \Gamma \vdash e_i : \tau_i\ \forall i \in [1, m]}{\Gamma \vdash \texttt{f\_}\ \texttt{(}\ e_1\ \texttt{,}\ \ldots\ \texttt{,}\ e_m\ \texttt{)} : \tau}\,\texttt{INV}$$

$$\frac{\delta_{\texttt{OPR}}(\odot, \tau, \tau') = \hat{\tau} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash \texttt{(}\ e_1\ \odot\ e_2\ \texttt{)} : \hat{\tau}}\,\texttt{OPX}$$

where the operator typing function $\delta_{\texttt{OPR}}$ is:

$$\delta_{\texttt{OPR}}(\odot, \tau, \tau') = \begin{cases} \mathbb{B} & \odot = \texttt{<},\ \tau = \tau' = \mathbb{B} \\ \mathbb{N} & \odot \in \{\texttt{+}, \texttt{*}\},\ \tau = \tau' = \mathbb{N} \\ \mathbb{B} & \odot = \texttt{==},\ \tau = \tau' \end{cases}$$

## Embedding the Type Checker

Typing derivations are compiled by decorating nonterminals with a pair, $\texttt{EXP}[\cdot, \cdot]$, carrying the type annotation, $e : \tau$, and type signature $\texttt{f0} : \vec{\tau} \to \dot{\tau}$.

$$\frac{\langle \vec{\tau}, \dot{\tau} \rangle \in \mathbb{T}^{0..k} \times \mathbb{T} \qquad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(S_\Gamma \to \texttt{fun}\ \texttt{f0}\ \texttt{(} \mathop{\Large ,}_{i=1}^{|\vec{\tau}|} (p_i\ \texttt{:}\ \vec{\tau}_i)\ \texttt{)}\ \texttt{:}\ \dot{\tau}\ \texttt{=}\ \texttt{EXP}[\dot{\tau}, \vec{\tau} \to \dot{\tau}]\right) \in P_\Gamma}\,\texttt{FUN}_\varphi$$

$$\frac{\texttt{EXP}[\tau, \pi] \in V_\Gamma \qquad \Gamma \vdash \texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau}{\left(\texttt{EXP}[\tau, \pi]\ \to\ \texttt{f\_}\ \texttt{(} \mathop{\Large ,}_{i=1}^{m} \texttt{EXP}[\tau_i, \pi]\ \texttt{)}\right) \in P_\Gamma}\,\texttt{INV}_\varphi$$

$$\frac{\texttt{EXP}[\tau, \vec{\tau} \to \dot{\tau}] \in V_\Gamma \qquad \tau = \dot{\tau} \qquad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(\texttt{EXP}[\tau, \vec{\tau} \to \dot{\tau}]\ \to\ \texttt{f0}\ \texttt{(} \mathop{\Large ,}_{i=1}^{|\vec{\tau}|} \texttt{EXP}[\vec{\tau}_i, \vec{\tau} \to \dot{\tau}]\ \texttt{)}\right) \in P_\Gamma}\,\texttt{REC}_\varphi$$

$$\frac{\texttt{EXP}[\tau, \pi] \in V_\Gamma \qquad \tau = \tau' \qquad \tau, \tau' \in \mathbb{T}}{\left(\begin{array}{ll} \texttt{EXP}[\tau, \pi]\ \to & \texttt{if}\ \texttt{EXP}[\mathbb{B}, \pi]\ \texttt{\{}\ \texttt{EXP}[\tau, \pi]\ \texttt{\}} \\ & \texttt{else}\ \texttt{\{}\ \texttt{EXP}[\tau', \pi]\ \texttt{\}} \end{array}\right) \in P_\Gamma}\,\texttt{IFE}_\varphi$$

$$\frac{\texttt{EXP}[\hat{\tau}, \pi] \in V_\Gamma \qquad \delta_{\texttt{OPR}}(\odot, \tau, \tau') = \hat{\tau} \qquad \odot \in \{\texttt{==}, \texttt{<}, \texttt{+}, \texttt{*}\}}{\left(\texttt{EXP}[\hat{\tau}, \pi]\ \to\ \texttt{(}\ \texttt{EXP}[\tau, \pi]\ \odot\ \texttt{EXP}[\tau', \pi]\ \texttt{)}\right) \in P_\Gamma}\,\texttt{OPX}_\varphi$$

$$\frac{\texttt{EXP}[\tau, \vec{\tau} \to \dot{\tau}] \in V_\Gamma \quad \exists \vec{\tau}_i = \tau}{\left(\texttt{EXP}[\tau, \vec{\tau} \to \dot{\tau}] \to \texttt{pi}\right) \in P_\Gamma}\,\texttt{PID}_\varphi \qquad \frac{\texttt{EXP}[\tau, \pi] \in V_\Gamma \quad \texttt{\_} : \tau \in \{\mathbb{B}, \mathbb{N}\}}{\left(\texttt{EXP}[\tau, \pi] \to \texttt{\_}\right) \in P_\Gamma}\,\ulcorner\mathbb{T}\lrcorner_\varphi$$

Finally, normalize to CNF ($G'_\Gamma$), pruning unreachable/unproductive nonterminals. In this case, the blow-up is close to linear in $|G_\Gamma|$.

## Finite Language Intersection

Context-free languages are closed under intersection with regular languages. Constructively, given CNF productions $W \to XZ$ and $\alpha = \langle Q, \Sigma, \delta, q_\alpha, F \rangle$, we build synthetic nonterminals $pWr$ (for all $p, r \in Q$) and then add:

- $pWr \to pXq\ \ qZr$ for each $W \to XZ$ and $p, q, r \in Q$,
- $pWq \to a$ for each $W \to a$ with $\delta(p, a) = q$,
- start rules $S \to q_\alpha S q_\omega$ for each $q_\omega \in F$.
- start rules $S \to q_\alpha S q_\omega$ for each $q_\omega \in F$.

If $\alpha$ is acyclic, the $\mathcal{L}(\alpha_\cap)$ admits efficient enumeration and exact sampling.

## Generalized Regular Expressions

Finite slices of a CFL are finite and therefore regular. We use GREs (union, intersection, concatenation, star, complement) as a compact algebra for propagating regular constraints during parsing and decoding.

$$\begin{array}{ll} \mathcal{L}(\ \varnothing\ ) = \varnothing & \mathcal{L}(\ R^*\ ) = \{\varepsilon\} \cup \mathcal{L}(R \cdot R^*) \\ \mathcal{L}(\ \varepsilon\ ) = \{\varepsilon\} & \mathcal{L}(\ R \vee S\ ) = \mathcal{L}(R) \cup \mathcal{L}(S) \\ \mathcal{L}(\ a\ ) = \{a\} & \mathcal{L}(\ R \wedge S\ ) = \mathcal{L}(R) \cap \mathcal{L}(S) \\ \mathcal{L}(\ R \cdot S\ ) = \mathcal{L}(R) \times \mathcal{L}(S) & \mathcal{L}(\ \neg R\ ) = \Sigma^* \setminus \mathcal{L}(R) \end{array}$$

When computing the closure of a parse chart, each chart cell carries a vector of GREs, one per nonterminal, representing exactly which terminal strings can fill that span while respecting the constraints.

## Brzozowski Differentiation

Brzozowski derivatives provide a rewrite-based procedure for incremental LTR decoding:

$$\partial_a L = \{b \in \Sigma^* \mid ab \in L\}.$$

Enables sampling from intersections without materializing product automata.

$$\begin{array}{ll} \partial_a(\ \varnothing\ ) = \varnothing & \delta(\ \varnothing\ ) = \varnothing \\ \partial_a(\ \varepsilon\ ) = \varnothing & \delta(\ \varepsilon\ ) = \varepsilon \\ \partial_a(\ a\ ) = \varepsilon & \delta(\ a\ ) = \varnothing \\ \partial_a(\ b\ ) = \varnothing \text{ for each } a \neq b & \delta(\ R^*\ ) = \varepsilon \\ \partial_a(\ R^*\ ) = (\partial_x R) \cdot R^* & \delta(\ \neg R\ ) = \varepsilon \text{ if } \delta(R) = \varnothing \\ \partial_a(\ \neg R\ ) = \neg \partial_a R & \delta(\ \neg R\ ) = \varnothing \text{ if } \delta(R) = \varepsilon \\ \partial_a(\ R \cdot S\ ) = (\partial_a R) \cdot S \vee \delta(R) \cdot \partial_a S & \delta(\ R \cdot S\ ) = \delta(R) \wedge \delta(S) \\ \partial_a(\ R \vee S\ ) = \partial_a R \vee \partial_a S & \delta(\ R \vee S\ ) = \delta(R) \vee \delta(S) \\ \partial_a(\ R \wedge S\ ) = \partial_a R \wedge \partial_a S & \delta(\ R \wedge S\ ) = \delta(R) \wedge \delta(S) \end{array}$$

## Takeaways & Next Steps

- A fixed-parameter tractable reduction from syntax-directed type system to a CFG with soundness and completeness guarantees
- Intersection with an acyclic regular grammar yields a regular expression
- Brzozowski derivatives enable incremental autoregressive decoding
- TODO
- TODO
- Future: lazy CNF materialization, quotienting by syntactic symmetries (e.g., $\alpha$-equivalence / invariants), richer typing (subtyping, polymorphism, substructural constraints).