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 universe $\mathbb{T}$ with at two base types $\mathbb{B}, \mathbb{N}$, and an ambient global context Γ of named functions `f_` : $(\tau_1, \ldots, \tau_m) \to \tau$.

Static Semantics

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

$$\frac{\Gamma \vdash e_c : \mathbb{B} \qquad \Gamma \vdash e_\top : \tau \qquad \Gamma \vdash e_\bot : \tau}{\Gamma \vdash \texttt{if}\ e_c\ \{\ e_\top\ \}\ \texttt{else}\ \{\ e_\bot\ \} : \tau}\ \textsc{IFE}$$

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

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

where the infix operator typing function $\delta_{\textsc{OPR}}$ is defined in the following way:

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

Embedding the Type Checker

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

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

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

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

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

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

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

Finally, we normalize to Chomsky Normal Form (CNF), pruning unreachable and unproductive nonterminals. Expansion will be close to linear in $|G_\Gamma|$.

Finite Language Intersection

Context-free languages are closed under intersection with regular languages. Constructively, given CNF productions of the form $W \to XZ$ or $W \to a$, and $\alpha = \langle Q, \Sigma, \delta, q_\alpha, F \rangle$, we build synthetic nonterminals pWr, then add:

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

This naïve construction (Salomaa, 1973) can be significantly improved via a semiring dynamic programming algorithm that avoids useless productions. When α is acyclic, $\mathcal{L}(\alpha_\cap)$ admits efficient enumeration and exact sampling.

Autoregressive Decoding

Finite slices of a CFL are finite and therefore regular. We use star free regular expressions as a compact algebra for propagating regular constraints during parsing and decoding. Let $e : E$ be an expression defined by the grammar:

$$e \to \varnothing \mid \varepsilon \mid \Sigma \mid e \cdot e \mid e \vee e \mid e \wedge e$$

where ε is the empty symbol and Σ is a finite alphabet. We interpret these expressions as denoting regular languages, where $a \in \Sigma$:

$$\begin{aligned} \mathcal{L}(\varnothing) &= \varnothing & \mathcal{L}(\,x \cdot z\,) &= \mathcal{L}(x) \circ \mathcal{L}(z) \\ \mathcal{L}(\,\varepsilon\,) &= \{\varepsilon\} & \mathcal{L}(x \vee z) &= \mathcal{L}(x) \cup \mathcal{L}(z) \\ \mathcal{L}(\,a\,) &= \{a\} & \mathcal{L}(x \wedge z) &= \mathcal{L}(x) \cap \mathcal{L}(z) \end{aligned}$$

where $\mathcal{L}(x) \circ \mathcal{L}(z) \coloneqq \{a \cdot b \mid a \in \mathcal{L}(x) \wedge b \in \mathcal{L}(z)\}$. Brzozowski (1962) introduces an equational theory for quotients, $\partial_a(L) = \{b \mid ab \in L\}$:

$$\begin{aligned} \partial_a(\ \ \varnothing\ \ ) &= \varnothing & \delta(\ \ \varnothing\ \ ) &= \varnothing \\ \partial_a(\ \ \varepsilon\ \ ) &= \varnothing & \delta(\ \ \varepsilon\ \ ) &= \varepsilon \\ \partial_a(\ \ b\ \ ) &= \begin{cases} \varepsilon & \text{if } a = b \\ \varnothing & \text{if } a \neq b \end{cases} & \delta(\ \ a\ \ ) &= \varnothing \\ \partial_a(\,x \cdot z\,) &= (\partial_a x) \cdot z \vee \delta(x) \cdot \partial_a z & \delta(\,x \cdot z\,) &= \delta(x) \wedge \delta(z) \\ \partial_a(x \vee z) &= \partial_a x \vee \partial_a z & \delta(x \vee z) &= \delta(x) \vee \delta(z) \\ \partial_a(x \wedge z) &= \partial_a x \wedge \partial_a z & \delta(x \wedge z) &= \delta(x) \wedge \delta(z) \end{aligned}$$

Now, for any nonempty $(\varepsilon, \wedge)$-free regex, e, choose (e) witnesses $\sigma \in \mathcal{L}(e)$:

$$\texttt{follow}\,(e) : E \to 2^\Sigma = \begin{cases} \{e\} & \text{if } e \in \Sigma \\ \texttt{follow}\,(x) & \text{if } e = x \cdot z \\ \texttt{follow}\,(x) \cup \texttt{follow}\,(z) & \text{if } e = x \vee z \end{cases}$$

$$\text{choose}\,(e) : E \to \Sigma^+ = \begin{cases} e & \text{if } e \in \Sigma \\ \left(s \leftsquigarrow \texttt{follow}\,(e)\right) \cdot \text{choose}\,(\partial_s e) & \text{if } e = x \cdot z \\ \text{choose}\left(e' \leftsquigarrow \{x, z\}\right) & \text{if } e = x \vee z \end{cases}$$

This enables LTR decoding without materializing the product automaton.

Takeaways & Next Steps

- A fixed-parameter $\langle k, |\mathbb{T}| \rangle$ tractable embedding from a syntax-directed type system to a CFG with soundness and completeness guarantees.
- Intersection with an acyclic FSA yields a star-free regular expression.
- Brzozowski derivatives enable incremental autoregressive decoding.
- Future work: lazy CNF materialization, quotienting (e.g., α-equivalence), richer type theories (subtyping, polymorphism, substructural constraints).