$$
\begin{array}{llr}
\text{(Regexs) } r ::= & U & \text{variable} \\
& \mid \ \mathrm{s} \in \Sigma^* & \text{base} \\
& \mid \ r* & \text{star} \\
& \mid \ r_1 r_2 & \text{concat} \\
& \mid \ r_1 | r_2 & \text{or}
\end{array}
$$

Figure 1: Regex Syntax

CONSTANT EXAMPLE
$$\frac{s \in \Sigma^*}{\Delta \vdash s : s}$$

USERDEF EXAMPLE
$$\frac{\Delta' \vdash s : r}{\Delta' \cup \{(r, U)\} \vdash s : U}$$

CONCAT EXAMPLE
$$\frac{\Delta \vdash s_1 : r_1 \qquad \Delta \vdash s_2 : r_2}{\Delta \vdash s_1 s_2 : r_1 r_2}$$

OR EXAMPLE
$$\frac{\Delta \vdash s : r_1}{\Delta \vdash s : r_1 \mid r_2}$$

EMPTY STAR
$$\frac{}{\Delta \vdash \epsilon : r^*}$$

NONEMPTY STAR
$$\frac{\Delta \vdash s_1 : r \qquad \Delta \vdash s_2 : r^*}{\Delta \vdash s_1 s_2 : r^*}$$

Figure 2: Regex Semantics, $\mathcal{L}_\Delta(r) = \{s | \Delta \vdash s : r\}$

let Δ be the set of user defined data types. let Σ^* be the set of words over the alphabet Σ

$$
\begin{array}{llr}
\text{(Lenses) } l ::= & const(s_1 \in \Sigma^*, s_2 \in \Sigma^*) & \text{const} \\
& \mid \ identity & \text{identity} \\
& \mid \ iterate(l) & \text{iterate} \\
& \mid \ concat(l_1, l_2) & \text{concat} \\
& \mid \ swap(l_1, l_2) & \text{swap} \\
& \mid \ or(l_1, l_2) & \text{or} \\
& \mid \ l_1 \circ l_2 & \text{compose}
\end{array}
$$

Figure 3: Lens Syntax

Abstract

Categories and Subject Descriptors F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic—Proof Theory; I.2.2 [*Artificial Intelligence*]: Automatic Programming—Program Synthesis

General Terms Languages, Theory

Keywords Functional Programming, Proof Search, Program Synthesis, Type Theory

1. Introduction

We have regular expressions as normal regular expressions. We expand it with having user defined data types as well. We have lenses, defined in Figure 3, typed as in Figure 4. These lenses have underlying functions, given via Figure 5. We would like to be able to synthesize these lenses automatically, given a specification as two regular expressions, an a set of values that are mapped to each other.

2. DNF Regular Expressions and Lenses

We can push the boundaries of what is expressible through syntactic lenses through finding a language, equivalent to regular expressions, with fewer equivalences.

A language which removes some of those equivalences is the language of regular expressions in disjunctive normal form. This language removes the equivalences corresponding to distributivity, associativity, and concatenation identity. Figure 6 defines this, with semantics as in 7 The full DNF Regex is a list of Clauses. This corresponds to a chain of $+$s in the normal language of regular expressions. A Clause is a list of Atoms, with strings in between. This corresponds to a chain of composition. These strings correspond to regular expression base types, and their requirement to exist between every atom removes the ϵ equivalences. Atoms correspond to pieces that either cannot be broken up uniquely, or we feel should not be broken up. These are the star regular expressions, and the user defined regular expressions. The star regular expressions

CONSTANT LENS
$$\frac{s_1 \in \Sigma^* \qquad s_2 \in \Sigma^*}{\Delta \vdash const(s_1, s_2) : s_1 \Leftrightarrow s_2}$$

IDENTITY LENS
$$\frac{}{\Delta \vdash identity : r \Leftrightarrow r}$$

ITERATE LENS
$$\frac{\Delta \vdash l : r_1 \Leftrightarrow r_2 \qquad \mathcal{L}_\Delta(r_1)^{!*} \qquad \mathcal{L}_\Delta(r_2)^{!*}}{\Delta \vdash iterate(l) : r_1^* \Leftrightarrow r_2^*}$$

CONCAT LENS
$$\frac{\begin{array}{cc}\Delta \vdash l_1 : r_1 \Leftrightarrow s_1 & \Delta \vdash l_2 : r_2 \Leftrightarrow s_2 \\ \mathcal{L}_\Delta(r_1).!\mathcal{L}_\Delta(r_2) & \mathcal{L}_\Delta(s_1).!\mathcal{L}_\Delta(s_2)\end{array}}{\Delta \vdash concat(l_1, l_2) : r_1 r_2 \Leftrightarrow s_1 s_2}$$

SWAP LENS
$$\frac{\begin{array}{cc}\Delta \vdash l_1 : r_1 \Leftrightarrow s_1 & \Delta \vdash l_2 : r_2 \Leftrightarrow s_2 \\ \mathcal{L}_\Delta(r_1).!\mathcal{L}_\Delta(r_2) & \mathcal{L}_\Delta(s_2).!\mathcal{L}_\Delta(s_1)\end{array}}{\Delta \vdash concat(l_1, l_2) : r_1 r_2 \Leftrightarrow s_2 s_1}$$

OR LENS
$$\frac{\begin{array}{cc}\Delta \vdash l_1 : r_1 \Leftrightarrow s_1 & \Delta \vdash l_2 : r_2 \Leftrightarrow s_2 \\ \mathcal{L}_\Delta(r_1) \cap \mathcal{L}_\Delta(r_2) = \emptyset & \mathcal{L}_\Delta(s_1) \cap \mathcal{L}_\Delta(s_2) = \emptyset\end{array}}{\Delta \vdash or(l_1, l_2) : r_1 | s_1 \Leftrightarrow r_2 | s_2}$$

COMPOSE LENS
$$\frac{\Delta \vdash l_1 : r_1 \Leftrightarrow r_2 \qquad \Delta \vdash l_2 : r_2 \Leftrightarrow r_3}{\Delta \vdash l_2 \circ l_1 : r_1 \Leftrightarrow r_3}$$

RETYPE LENS
$$\frac{\Delta \vdash l : r_1 \Leftrightarrow r_2 \qquad \mathcal{L}_\Delta(r_1) = \mathcal{L}_\Delta(r_1') \qquad \mathcal{L}_\Delta(r_2) = \mathcal{L}_\Delta(r_2')}{\Delta \vdash l : r_1' \Leftrightarrow r_2'}$$

Figure 4: Lens Typing

- $const(s_1, s_2)(s_1, s_2)$
- $identity(s, s)$
- $iterate(l)(\epsilon, \epsilon)$
- $iterate(l)(s, t)$ if $l(s_1, t_1)$ and $iterate(l)(s_2, t_2)$ and $s = s_1 s_2$ and $t = t_1 t_2$
- $concat(l_1, l_2)(s, t)$ if $l_1(s_1, t_1)$ and $l_2(s_2, t_2)$ and $s = s_1 s_2$ and $t = t_1 t_2$
- $swap(l_1, l_2)(s, t)$ if $l_1(s_1, t_1)$ and $l_2(s_2, t_2)$ and $s = s_1 s_2$ and $t = t_2 t_1$
- $or(l_1, l_2)(s, t)$ if $l_1(s, t)$ or $l_2(s, t)$
- $(l_2 \circ l_1)(s, t)$ if there exists a u such that $l_1(s, u)$ and $l_2(u, t)$

Figure 5: Lens Semantics

$$
\begin{array}{llll}
\text{(Atoms)} & a & ::= & U & \text{variable} \\
 & & & \mid\ dr* & \text{iterate} \\
\text{(Clauses)} & cl & ::= & (\lambda i : [1,n].a_i, \lambda i : [0,n].s_i) & \text{clause} \\
\text{(DNF Regex)} & dr & ::= & \lambda i : [1,n].cl_i & \text{dnf regex}
\end{array}
$$

Figure 6: DNF Regex Syntax

ATOM USERDEF
$$\frac{\Delta' \vdash s : r}{\Delta' \cup \{(r, U)\} \vdash s : U}$$

ATOM EMPTY STAR
$$\frac{}{\Delta \vdash \epsilon : dr^*}$$

ATOM NONEMPTY STAR
$$\frac{\Delta \vdash s_1 : dr \qquad \Delta \vdash s_2 : dr^*}{\Delta \vdash s_1 s_2 : dr^*}$$

CLAUSE
$$\frac{\lambda i : [1,n].\Delta \vdash s_i : a_i}{\Delta \vdash t_0 s_1 \ldots s_n t_n : (\lambda i : [1,n].a_i, \lambda i : [0,n].t_i)}$$

DNF REGEX
$$\frac{\Delta \vdash s : cl_i}{\Delta \vdash s : (\lambda i : [1,n].cl_i)}$$

Figure 7: Regex Semantics $\mathcal{L}_\Delta(dr) = \{s | \Delta \vdash s : dr\}$ $\mathcal{L}_\Delta(cl) = \{s | \Delta \vdash s : cl\}$ $\mathcal{L}_\Delta(a) = \{s | \Delta \vdash s : a\}$

$$
\begin{array}{llll}
\text{(Atom Lenses)} & al & ::= & Iterate(dl) & \text{iterate} \\
 & & & \mid\ Identity & \text{identity} \\
\text{(Clause Lenses)} & cll & ::= & (\lambda i : [0,n].(s_i, t_i), & \\
 & & & \lambda i : [1,n].a_i, & \\
 & & & \sigma \in [1,n] \rightarrowtail [1,n]) & \text{clause lens} \\
\text{(DNF Lenses)} & dl & ::= & (\lambda i : [1,n].cl_i, & \\
 & & & \sigma \in [1,n] \rightarrowtail [1,n]) & \text{dnf lens}
\end{array}
$$

Figure 8: DNF Lens Syntax

Semantics of DNF Lenses:

- $(\lambda i : [1,n].cl_i, \sigma \in [1,n] \rightarrowtail [1,n])(s,t)$ if there exists an i such that $cl_i(s,t)$

Semantics of Clause Lenses:

- $(\lambda i : [0,n].(s'_i, t'_i), \lambda i : [1,n].a_i, \sigma \in [1,n] \rightarrowtail [1,n])$ (s,t) if $a_i(s_i, t_i)$ for all i and $s = s'_0 s_1 \ldots s_n s'_n$ and $t = t'_0 t_{\sigma(1)} \ldots t_{\sigma(n)} t'_n$

Semantics of Atom Lenses:

- $identity(s,s)$
- $iterate(dl)(\epsilon, \epsilon)$
- $iterate(dl)(s,t)$ if $dl(s_1,t_1)$ and $iterate(dl)(s_2,t_2)$ and $s = s_1 s_2$ and $t = t_1 t_2$

Figure 9: DNF Lens Semantics

IDENTITY ATOM LENS
$$\frac{}{\Delta \vdash identity : U \Leftrightarrow U}$$

ITERATE ATOM LENS
$$\frac{\Delta \vdash dl : dr_1 \Leftrightarrow dr_2 \qquad \mathcal{L}_\Delta(dr_1)^{!*} \qquad \mathcal{L}_\Delta(dr_2)^{!*}}{\Delta \vdash iterate(dl) : dr_1^* \Leftrightarrow dr_2^*}$$

CLAUSE LENS
$$\frac{\begin{array}{c}\lambda i : [1,n].(\Delta \vdash al_i : a_i \Leftrightarrow b_i) \\ \sigma \in [1,n] \rightarrowtail [1,n] \qquad \mathcal{L}_\Delta(s_{i-1} a_i \epsilon).!\mathcal{L}_\Delta(s_i a_{i+1} \epsilon) \\ \mathcal{L}_\Delta(t_{i-1} b_{\sigma(i)} \epsilon).!\mathcal{L}_\Delta(t_i b_{\sigma(i+1)} \epsilon)\end{array}}{\begin{array}{c}\Delta \vdash (\lambda i : [0,n].(s_i, t_i), \lambda i : [1,n].a_i, \sigma) : \\ (\lambda i : [1,n].a_i, \lambda i : [0,n].s_i) \Leftrightarrow ((\lambda i : [1,n].b_i) \circ \sigma, \lambda i : [0,n].t_i)\end{array}}$$

DNF REGEX LENS
$$\frac{\begin{array}{c}\Delta \vdash cll_i : cl_i \Leftrightarrow dl_i \qquad \sigma \in [1,n] \rightarrowtail [1,n] \\ i \neq j \Rightarrow cl_i \cap cl_j = \emptyset \qquad i \neq j \Rightarrow dl_i \cap dl_j = \emptyset\end{array}}{\Delta \vdash (\lambda i : [1,n].cll_i, \sigma) : (\lambda i : [1,n].cl_i) \Leftrightarrow (\lambda i : [1,n].dl_i \circ \sigma)}$$

Figure 10: DNF Lens Typing

can be broken up in an infinite number of ways. We feel that a user grouping together aspects of their regular expression together signifies that it will likely stay together in the transformation.

As we have dnf regular expressions, it also makes sense to have dnf lenses, whose types are dnf regular expressions. We define dnf lenses in Figure 8. These dnf lenses have the comparable underlying functions to the underlying functions of normal lenses, and are given in Figure 9 TODO: make a nice diagram of how clause lenses work. They can be typed according to the rules given in Figure 10

One thing to note about these lenses is that they are slightly more expressive than the syntactic lenses given previously. Not all permutations are possible through only using swap. Furthermore, before we did not allow for permutations on union clauses, which previously were dealt with through the type change rule of pure lenses. Now, these permutations are possible. So, for example, there are no syntactic lenses between $a + b*$ and $z * +y$. However, with allowing for permutations, the new language can synthesize these types of lenses.

Just as we used the parse trees of strings within a regular expression, to help guide the synthesis for syntactic lenses, we can do the same for dnf lenses.

3. Proofs

3.1 Completeness of DNF Regular Expressions to Regular Expressions

Is suffices to provide a mapping from DNF regular expressions to regular expressions.

Definition 1 (RepDnfRegex)**.** *We define a representative dnf regex for a regex as follows:*

$$
\begin{array}{rl}
\textit{variable} & RepDnfRegex(U) = [([U], [\epsilon, \epsilon])] \\
\textit{base} & RepDnfRegex(s) = [([], [s])] \\
\textit{star} & RepDnfRegex(r^*) = [([RepDnfRegex(r)^*], [\epsilon, \epsilon])] \\
\textit{concat} & RepDnfRegex(r_1 r_2) = Concat(RepDnfRegex(r_1), RepDnfRegex(r_2 \\
 & \textit{where Concat is defined via something horrible} \\
\textit{or} & RepDnfRegex(r_1 | r_2) = append(RepDnfRegex(r_1), RepDnfRegex(r_2
\end{array}
$$

Lemma 1 (Equivalence of $RepDnfRegex$)**.** *For all* Δ, $\mathcal{L}_\Delta(RepDnfRegex(r))$ $\mathcal{L}_\Delta(r)$

Proof. □

Theorem 1 (Completeness of DNF Regular Expressions)**.** *For any regular expression r, there exists a DNF regular expression, dr, such that for all Δ, $\mathcal{L}_\Delta(dr) = \mathcal{L}_\Delta(r)$.*

Proof. Consider $dr = RepDnfRegex(r)$. By above lemma, $\mathcal{L}_\Delta(dr) = \mathcal{L}_\Delta(r)$. □

3.2 Soundness of DNF Regular Expressions to Regular Expressions

3.3 Completeness

Lemma 2 (Identity Completeness)**.** *Identity is expressible in the language of DNF Lenses*

Proof. By induction on regular expressions

r=U a

□

3.4 Soundness

We say that dnf lenses are *sound* if, there is a dnf lens between two dnf regular expressions, then between any two regular expressions equivalent to the two dnf regular expressions, there is a lens between those regular expressions such that the lens and dnf lens have the same semantics.

Definition 2 (repregex)**.** *We define a representative regex for a dnf regex as follows:*

- $repregex([cl]) = repregex(cl)$
- $repregex([cl_1; \ldots; cl_n]) =$ $repregex(cl_1)|repregex([cl_2; \ldots; cl_n])$
- $regregex([s]) = s$
- $repregex([s_1; a_1; \ldots; a_n; s_{n+1}]) =$ $s_1(repregex(a_1)repregex([s_2; a_2; \ldots; s_{n+1}]))$
- $repregex(U) = U$
- $repregex(dr^*) = repregex(dr)^*$

Lemma 3 (Equivalence of repregex)**.** *$\mathcal{L}_{dr}(=)\mathcal{L}_{repregex(dr)}()$, $\mathcal{L}_{cl}(=)\mathcal{L}_{repregex(cl)}()$, and $\mathcal{L}_a(=)\mathcal{L}_{repregex(a)}()$*

Proof. By induction on the structure of dr, cl, and a □

Definition 3 (Adjacent Swapping Permutation)**.** *Let $\sigma_i : [0, n] \rightarrowtail [0, n]$ be the permutation, where $\sigma_i(i) = i + 1$, $\sigma_i(i + 1) = i$, $\sigma_{i,j}(k \neq i, i + 1) = k$*

Lemma 4 (Expressibility of Adjacent Swapping Permutation)**.** *Let σ_i be an adjacent element swapping permutation. The language of lenses can express $([(s_1, s_1); identity; \ldots; identity; (s_n, s_n)], \sigma_i)$.*

Proof. Consider the regular expressions $repregex([s_1; a_1; \ldots; s_i])$ $(repregex(a_i)(s_{i+1}repregex(a_{i+1})))$ $repregex([s_{i+2}; \ldots; a_n; s_{n+1}])$ and $repregex([s_{1,2}; a_{1,2}; \ldots; s_{i,1}])$ $((a_{i+1,1}s_{i+1,1})a_{i,1})$ $repregex([s_{i+2,1}; \ldots; a_n; s_{n+1}])$
Consider the lens between them
$concat(concat(identity, swap(identity, swap(identity, identity))), identity)$
By inspection, this lens is equivalent to the adjacent swapping permutation. □

Lemma 5 (Expressibility of Permutation)**.** *The language of lenses can express $([(s_1, s_1); identity; \ldots; identity; (s_n, s_n)], \sigma)$ for any permutation σ.*

Proof. Let σ be a permutation. Consider the clause lens $([(s_1, s_1); identity; \ldots; identity; (s_n, s_n)], \sigma_i)$. From algebra, we know that the group of permutations is generated by all adjacent swaps $\sigma_i = (i, i + 1)$. So there exists an adjacency swap decomposition of $\sigma = \sigma_{i_1} \ldots \sigma_{i_m}$. Consider the dnf lens $([(s_1, s_1); identity; \ldots; identity; (s_n, s_n)], \sigma_{i_j})$ for each σ_{i_j}. By the above lemma, there exists a l_j for each of these adjacency swaps. Consider the lens $l = l_{i_1} \circ l_{i_2} \circ \ldots \circ l_{i_m}$ By the semantics, they are the same. □

Theorem 2 (Soundness)**.** *Let r and s be two regular expressions, and dr and ds be two dnf regular expressions. If $\mathcal{L}_r(=)\ \mathcal{L}_{dr}(a)nd$ $\mathcal{L}_s(=)\ \mathcal{L}_{ds}(,)$ then if there exists a dnf lens $dl : dr \Leftrightarrow ds$, then there exists a lens $l : r \Leftrightarrow s$ such that $dl.putr = l.putr$.*

Proof. By induction on the typing dl.

DNF Lens Intro Let $\Delta \vdash ([cll_1; \ldots; cll_n], \sigma) : [cl_{1,1}, \ldots, cl_{n,1}] \Leftrightarrow [cl_{\sigma(1),2}, \ldots, cl_{\sigma(n),2}]$. This comes from the derivations that $\Delta \vdash cll_i : cl_{i,1} \Leftrightarrow cl_{i,2}$ for all i. Consider instead the lens $dl' = \Delta \vdash ([cll_1; \ldots; cll_n], \sigma_i d) : [cl_{1,1}, \ldots, cl_{n,1}] \Leftrightarrow [cl_{1,2}, \ldots, cl_{n,2}]$. By Lemma (TODO: this lemma), these two lenses are semantically equivalent. By Lemma (TODO: this lemma), $\Delta \vdash repregex(dl') : repregex(dr_1) \Leftrightarrow repregex(dr_2')$, with $repregex(dl)$ semantically equivalent to dl. So $repregex(dl')$ is semantically equivalent to dl', and $DNFLens'$ is semantically equivalent dl, so $repregex(dl')$ is semantically equivalent to dl. Merely adding in a retyping rule at the end of $repregex(dl')$ (as they have the same type), completes this case.

Clause Lens Intro Let $\Delta \vdash ([(s_{1,1}, s_{1,2}); al_1; \ldots; al_n; (s_{n+1,1}, s_{n+1,2})], \sigma) : [s_{1,1}; a_{1,1}; \ldots; a_{1,n}; s_{1,n+1}] \Leftrightarrow [s_{1,2}; a_{\sigma(1),2}; \ldots; a_{\sigma(n),2}; s_{n+1,2}]$.
Consider two lenses, $\Delta \vdash ([(s_{1,1}, s_{1,2}); al_1; \ldots; al_n; (s_{n+1,1}, s_{n+1,2})], \sigma_i d) : [s_{1,1}; a_{1,1}; \ldots; a_{1,n}; s_{1,n+1}] \Leftrightarrow [s_{1,2}; a_{1,2}; \ldots; a_{n,2}; s_{n+1,2}]$,
and $\Delta \vdash ([(s_{1,2}, s_{1,2}); identitylens(a_{1,2}); \ldots; identitylens(a_{n,2}); (s_{n+1,2},$
$[s_{1,2}; a_{1,2}; \ldots; a_{n,2}; s_{n+1,2}] \Leftrightarrow [s_{1,2}; a_{\sigma(1),2}; \ldots; a_{\sigma(n),2}; s_{n+1,2}]$.
By Lemma (TODO:), there exists a lens equivalent to the first one, call it $l_{transform}$. By Lemma (TODO:), there exists a lens equivalent to the second one, call it l_σ. Consider $l_\sigma \circ l_{transform}$.
Go through semantics, oh look they are equivalent.

□

4. Search Strategy

4.1 Equivalence Relation

4.2 Distance Metric

5. Implementation

5.1 Evaluation

6. Related Work

6.1 FlashFill

6.2 FlashExtract

6.3 SemFill

7. Future Work

7.1 Complex Data Structures

7.2 Richer Classes of Lenses

Acknowledgments

References