

CS 260: Assignment #4

Regular Expression

Due on Tuesday, November 24, 2015

Prof. Hardekopf

Chad Spensky

1 Abstraction Domain Lattice

Let our original lattice $L = (\mathbb{P}(S), \leq)$ and our abstract-domain lattice $L^\# = (\mathbf{regExpSet}(x), x \in \mathbb{P}(S \cup \{ |, *, +, (,) \}) \cup \{ \top, \perp \}, \sqsubseteq)$ where S is the set of all strings (including the empty string) where $\top = S$ and \perp is *undefined*. For convenience let all regular expressions, $expr$, denote the set of all possible strings that can be constructed from the expression, $\mathbf{regExpSet}(expr)$, and s denote the singleton set $\{s\}$, thus $\sqsubseteq = \subseteq$ in our lattice. Intuitively, $\mathbf{regExpSet}(s) = s$. Also, let $\mathbf{genRegExp}(\{a, b, c\})$ be the function that will generate the most concise regular expression to that contains all the elements in the set (Note that \top will never be returned because of the $|$ operator, i.e. a regular expression can always be generated).

$$\alpha(x) : L \rightarrow L^\# = \begin{cases} \perp & \text{if } x \text{ is } \{\} \\ \mathbf{genRegExp}(s) & \text{Otherwise} \end{cases}$$

Meet ($x \in L^\#$)

- $\perp \sqcap x = \perp, \forall x$
- $\top \sqcap x = x, \forall x$
- $x \sqcap y = x, \mathbf{genRegExp}(\mathbf{regExpSet}(x) \cap \mathbf{regExpSet}(y))$

Join ($x \in L^\#$)

- $\perp \sqcup x = x, \forall x$
- $\top \sqcup x = \top, \forall x$
- $x \sqcup y = \mathbf{genRegExp}(\mathbf{regExpSet}(x) \cup \mathbf{regExpSet}(y))$

The lattice is infinite, and also of infinite height because of the inclusion of the $|$ operator. The $+$ can only express strings which have common subsequences, and are not expressive enough to grow infinitely since our $\mathbf{genRegExp}()$ will always return the most concise possible regular expression. Since the $|$ operator is our pain point, our widening operator must remove the $|$'s. Let our widening operator be defined as follows:

$$x \nabla_{lim} y : L^\# \times L^\# \rightarrow L^\# = \begin{cases} x \sqcup y & \# \text{ of } | \text{'s in } x \sqcup y \leq lim \\ \top & \text{Otherwise} \end{cases}$$

This definition trivially satisfies condition 1, $x \sqsubseteq x \nabla_{lim} y, y \sqsubseteq x \nabla_{lim} y$. Similarly, because of our definition, no chains can be strictly ascending, as they will be limited by lim .

	\perp	y	\top
\perp	\perp	y	\top
x	x	$x y$	\top
\top	\top	\top	\top

(a) Concatenation (+)

	\perp	y	\top
\perp	T	T	T
x	F	$x \sqsubseteq y?$	T, F
\top	F	T, F	T, F

(b) Comparator (\leq)

Figure 1: Arithmetic Tables

1.1 Monotone Operators

Concatenation (+) For + to be monotone the following must hold: $x + y \leq x' + y' \Rightarrow \alpha(x) + \alpha(y) \sqsubseteq \alpha(x') + \alpha(y')$, where $x, y, x', y' \in \mathbb{P}(S)$ and $a \leq b \Rightarrow \text{substring}(a, b)$ for $a, b \in S$ and $x + y \Rightarrow \{x_i y_i\} \forall x_i \in x, y_i \in y$. Similarly $x \leq y \Rightarrow \text{substring}(x_i, y_i) \forall x_i \in x, y_i \in y$. In all cases $|x| = |y|$ and $|x'| = |y'|$.

- For case where $x = y = \text{" "}$, this holds trivially.
- Otherwise, we need to show that $\text{genRegExp}(x) || \text{genRegExp}(y) \sqsubseteq \text{genRegExp}(x') || \text{genRegExp}(y')$. We know that $xy \leq x'y'$, i.e. that xy is a substring of $x'y'$. Thus since $\text{genRegExp}()$ returns the most concise regular expression that represents all $a \in x$, $x \sqsubseteq \text{regExpSet}(\text{genRegExp}(x))$. Thus, $xy \sqsubseteq \text{regExpSet}(\text{genRegExp}(xy))$ and $x'y' \sqsubseteq \text{regExpSet}(\text{genRegExp}(x'y'))$. Since each regular expression necessarily contained the initial set, concatenating the expressions will still satisfy the condition of each, by taking the two representative sets and concatenating the strings of each. If there exists an element $z \in \text{regExpSet}(\text{genRegExp}(x \cup y)), z \notin \text{regExpSet}(\text{genRegExp}(x' \cup y'))$, this would imply that there are elements that are represented by $\text{genRegExp}(xy)$ that are not represented by $\text{genRegExp}(x'y')$, which we know can't happen since $xy \leq x'y'$. Therefore our assertion must hold.

□

Comparison (\leq) The same logic follows for \leq . For \leq to be monotone the following must hold: $x \leq y \Rightarrow \alpha(x) \sqsubseteq \alpha(y)$, where $x, y \in \mathbb{P}(S)$ and $x \leq y \Rightarrow x \sqsubseteq y$.

- For case where $x = y = \{\}$, this holds trivially.
- Otherwise, we must show that $\text{genRegExp}(x) \sqsubseteq \text{genRegExp}(y)$. Since $\text{genRegExp}()$ returns the most concise regular expression, we note that $x \sqsubseteq \text{regExpSet}(\text{genRegExp}(x))$. Thus $\text{regExpSet}(\text{genRegExp}(x)) \sqsubseteq \text{regExpSet}(\text{genRegExp}(y))$, which by similar logic to above, i.e. $z \in \text{regExpSet}(\text{genRegExp}(x)) \Rightarrow \text{regExpSet}(\text{genRegExp}(y))$, that is the regular expression for $\alpha(y)$ must at least express everything can be expressed by $\alpha(x)$.

□

1.2 Galois Connection

$$\gamma(\hat{x}) : L^\# \rightarrow L = \begin{cases} \{\} & \text{if } \hat{x} \text{ is } \perp \\ \mathbf{regExpSet}(x) & \text{if } \hat{x} \text{ is a regular expression} \\ S & \text{if } \hat{x} \text{ is } \top \end{cases}$$

We must show that $\alpha(x) \sqsubseteq \hat{x} \iff x \subseteq \gamma(\hat{x})$.

$\alpha(x) \sqsubseteq \hat{x} \Rightarrow x \subseteq \gamma(\hat{x})$:

- If $x = \{\}$, this holds trivially.
- Otherwise, either $\hat{x} = \top$, which holds trivially, or \hat{x} is a regular expression. Since our regular expressions are the most concise possible expressions that represent all elements the input set, we know that $x \subseteq \gamma(\hat{x})$ must be true.

$x \subseteq \gamma(\hat{x}) \Rightarrow \alpha(x) \sqsubseteq \hat{x}$:

- If $x = \{\}$, this holds trivially.
- Otherwise, every element in x can be represented by \hat{x} , thus $\alpha(x) \sqsubseteq \hat{x}$ must be true since, $\alpha(x)$ returns the most precise regular expression representing all of x . Any other conclusion would be a contradiction to our definition.

□

1.3 Soundness

Because a Galois connection exists, our approximation is sound. However, because we are using a widening operator, we lose precision. However, in the best case, i.e. we never hit our limit of $|$'s, we will actually return the exact input. Otherwise, we will only lose precision on those few cases.