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 $L^{\#} = (\textbf{regExpSet}(x), x \in \mathbb{P}(S \cup \{|, *, +, (,)\}) \cup \{\top, \bot\}, \sqsubseteq)$ where S is the set of all strings (including the empty string) where $\top = S$ and $\bot$ is *undefined*. For connivence let all regular expressions, $expr$, denote the set of all possible strings that can be constructed from the expression, $\textbf{regExpSet}(expr)$, and s denote the singleton set $\{s\}$, thus $\sqsubseteq = \subseteq$ in our lattice. Intuitively, $\textbf{regExpSet}(s) = s$. Also, let $\textbf{genRegExp}(\{a, b, c\})$ 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).

$$\alpha(x) : L \to L^{\#} = \begin{cases} \bot & \text{if } x \text{ is } \{\} \\ \textbf{genRegExp}(s) & \text{Otherwise} \end{cases}$$

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

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

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

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

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 $\textbf{genRegExp}()$ 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:

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

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

	$\bot$	y	$\top$
$\bot$	$\bot$	y	$\top$
x	x	$x \sqcup y$	$\top$
$\top$	$\top$	$\top$	$\top$

(a) Addition (+)

	$\bot$	y	$\top$
$\bot$	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 $x, y, x', y' \in \mathbb{P}(S)$ $x+y \Rightarrow x \cup y$. Similarly $x \leq y \Rightarrow x \subseteq y$.

- For case where $x = y =$ "", this holds trivially.
- Otherwise, we need to show that $\textbf{genRegExp}(x) \sqcup \textbf{genRegExp}(y) \sqsubseteq \textbf{genRegExp}(x') \sqcup \textbf{genRegExp}(y')$. We know that $x \cup y \subseteq x' \cup y'$. Thus since **genRegExp**() returns the most concise regular expression that represents all $a \in x$, $x \subseteq \textbf{regExpSet}(\textbf{genRegExp}(x))$. Thus, $x \cup y \subseteq \textbf{regExpSet}(\textbf{genRegExp}(x \cup y))$ and $x' \cup y' \subseteq \textbf{regExpSet}(\textbf{genRegExp}(x' \cup y'))$. If there exists an element $z \in \textbf{regExpSet}(\textbf{genRegExp}(x \cup y)), z \notin \textbf{regExpSet}(\textbf{genRegExp}(x' \cup y'))$, this would imply that there are elements that are represented by $\textbf{genRegExp}(x \cup y)$ that are not represented by $\textbf{genRegExp}(x' \cup y')$, which we know can't happen since $x \cup y \subseteq x' \cup y'$. 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 $x \leq y \Rightarrow x \subseteq y$.

- For case where $x = y = \{\}$, this holds trivially.
- Otherwise, we must show that $\textbf{genRegExp}(x) \sqsubseteq \textbf{genRegExp}(y)$. Since **genRegExp**() returns the most concise regular expression, we not that $x \subseteq \textbf{regExpSet}(\textbf{genRegExp}(x))$. Thus $\textbf{regExpSet}(\textbf{genRegExp}(x)) \sqsubseteq \textbf{regExpSet}(\textbf{genRegExp}(y))$, which by similar logic to above, i.e. $z \in \textbf{regExpSet}(\textbf{genRegExp}(x)) \Rightarrow \textbf{regExpSet}(\textbf{genRegExp}(y))$, that is the regular expression for $\alpha(y)$ must at least express everything can be expressed by $\alpha(x)$.

□

1.2 Galois Connection

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

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 expression 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.

$\square$

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.