1 Fixed Pattern

Definition 1.1. A fixed pattern pt is a tree structure for matching ASTs. It can be defined with a node set $N \cup \{*\}$ recursively as follows:

- pt can be a terminal n that matches AST terminals with the name n.name.
- pt can be a wildcard symbol `*` that matches any AST.
- pt can be a root nonterminal n with child fixed patterns in the form of $n(pt_1, \dots, pt_k)$, e.g., `expression(*,‘+’,*)`. An AST $I|n'$ is matched if n.name $= n'$.name and the k ordered children can be matched by $pt_1, \dots, pt_k$, respectively.

Definition 1.2. We define a partial order $\preceq$ among fixed patterns to compare their explicitness in matching ASTs. A pattern pt^a is more explicit than or equal to a pattern pt^b (denoted by $pt^a \preceq pt^b$) if all the ASTs that can be matched by pt^a can also be matched by pt^b. $pt^a \preceq pt^b$ holds if and only if any of the following holds:

- $pt^b = *$.
- pt^a and pt^b are terminals with the same name.
- $pt^a = n^a(pt^a_1, \dots, pt^a_k)$ and $pt^b = n^b(pt^b_1, \dots, pt^b_k)$ have the same root nonterminals (i.e., n^a.name $= n^b$.name) and $pt^a_i \preceq pt^b_i$ for $i = 1, \dots, k$.

Theorem 1.1. *$\preceq$ on fixed patterns is a partial order.*

Proof. $\preceq$ is reflexive:

- If $pt = *$, then $pt \preceq pt$ by definition.
- If pt is a terminal, then $pt \preceq pt$ by definition.
- Otherwise, $pt = n(pt_1, \dots, pt_k)$, then $pt \preceq pt$ holds if and only if $pt_i \preceq pt_i$ for $i = 1, \dots, k$, which is proved recursively.

$\preceq$ is antisymmetric. Supposed $pt^a \preceq pt^b$ and $pt^b \preceq pt^a$.

- If $pt^a = *$ and $pt^a \preceq pt^b$, by definition, pt^b can only be $*$, so $pt^a = pt^b$.
- If pt^a and pt^b are terminals with the same name, obviously, $pt^a = pt^b$.

- Otherwise, $pt^a = n^a(pt_1^a, \dots, pt_k^a)$, $pt^b = n^b(pt_1^b, \dots, pt_k^b)$, $n^a = n^b$, and $pt_i^a \preceq pt_i^b \wedge pt_i^b \preceq pt_i^a$ for $i = 1, \dots, k$. Therefore, as proved recursively, $pt_i^a = pt_i^b$ for $i = 1, \dots, k$, and $pt^a = pt^b$.

Finally, $\preceq$ is transitive, i.e., $pt^a \preceq pt^b \wedge pt^b \preceq pt^c \Rightarrow pt^a \preceq pt^c$.

- If $pt^c = *$, then $pt^a \preceq pt^c$ by definition.
- If pt^c is a terminal, then we must have $pt^c = pt^b = pt^a$ and $pt^a \preceq pt^c$.
- Otherwise, $pt^c = n^c(pt_1^c, \dots, pt_k^c)$, $pt^b = n^b(pt_1^b, \dots, pt_k^b)$, and $pt^a = n^a(pt_1^a, \dots, pt_k^a)$. We must have n^c.name $= n^b$.name $= n^a$.name and $pt_i^a \preceq pt_i^b \wedge pt_i^b \preceq pt_i^c$ for $i = 1, \dots, k$. Then $pt_i^a \preceq pt_i^c$ for $i = 1, \dots, k$, which is proved recursively.

$\square$

Definition 1.3. We define a join operation $\vee$ such that $pt^a \vee pt^b$ is a fixed pattern that can cover the ASTs matched by the inputted fixed patterns pt^a and pt^b. $pt^a \vee pt^b$ is defined as follows:

- $pt^a \vee pt^b = *$, if either pt^a or pt^b is $*$, or the root nodes in pt^a and pt^b have different names.
- If pt^a and pt^b are terminals with the same name, $pt^a \vee pt^b = pt^a = pt^b$.
- In the case $pt^a = n^a(pt_1^a, \dots, pt_k^a)$, $pt^b = n^b(pt_1^b, \dots, pt_k^b)$, and $n^a = n^b = n$, $pt^a \vee pt^b = n(pt_1^a \vee pt_1^b, \dots, pt_k^a \vee pt_k^b)$.

Theorem 1.2. *Fixed patterns partially ordered by $\preceq$ is a join-semilattice under $\vee$.*

Proof. We can show that $pt^a \preceq pt^a \vee pt^b$ with the previous two conditions as initial cases, and for the condition where $pt^a = n^a(pt_1^a, \dots, pt_k^a)$, $pt^b = n^b(pt_1^b, \dots, pt_k^b)$, and $n^a = n^b$, we can show $pt_i^a \preceq pt_i^a \vee pt_i^b$ for $i = 1, \dots, k$ recursively.

Similarly, we have $pt^b \preceq pt^a \vee pt^b$.

Finally, we can show that supposed the least upper bound of pt^a and pt^b is l, then $l = pt^a \vee pt^b$.

For the previous two conditions, the result is obvious.

For the condition where $pt^a = n^a(pt_1^a, \dots, pt_k^a)$, $pt^b = n^b(pt_1^b, \dots, pt_k^b)$, and $n^a = n^b = n$, l must have the form $n(l_1, \dots, l_k)$, and $l = pt^a \vee pt^b$ if and only if $l_i = pt_i^a \vee pt_i^b$ for $i = 1, \dots, k$, which can be proved recursively. $\square$

Algorithm 1 Mining fixed patterns modulo example

function MINELUBS(I)
    $\Sigma \leftarrow \{\text{PATTERN}(n) \mid n \in I.N\}$ ▷ Σ is the accumulated fixed patterns.
    $L \leftarrow \Sigma$ ▷ L is the newly mined fixed patterns.
    while $L \neq \emptyset$ **do**
        $L' \leftarrow \{\text{MERGELUB}(u, v) \mid u \in L \wedge v \in L \cup \Sigma\}$
        $L \leftarrow L' - \Sigma$
        $\Sigma \leftarrow \Sigma \cup L$
    return Σ
function MERGELUB(u, v)
    if $u = v$ **then**
        return u
    else if $u = *$ or $v = *$ or u, v do not have the same AST type **then**
        return $*$
    else
        $lu \leftarrow u$.children
        $lv \leftarrow v$.children
        return PATTERN(u.astType, MERGELUB(lu_1, lv_1), ..., MERGELUB(lu_k, lv_k))

Theorem 1.3. *Algorithm 1 is bound to terminate.*

Proof. As shown in Theorem 1.2, the MERGELUB(u, v) returns the least upper bound of u and v, and Σ is initialized by AST subtrees in I, which are also least upper bounds of themselves. Hence, Σ stores at most all the least upper bounds of the AST subsets in I, which is finite. After each iteration, L is either updated with unseen fixed patterns, or becomes empty. If L becomes empty, the algorithm terminates. If L is updated with unseen fixed patterns, the algorithm will terminate after a finite number of iterations. □

Theorem 1.4. *Algorithm 1 is sound (i.e., all the discovered patterns are LUBs of some AST subtrees in I) and complete (i.e., no LUB or equivalent class is missed).*

Proof. (Soundness) Σ is initialized with the LUBs for each AST subtree in I (LUB for a subtree is the subtree itself), and MERGELUB(u, v) returns the LUB of u and v, which is the LUB of the subsets of ASTs represented by u and v. Therefore, all the fixed patterns in Σ are LUBs of some AST subtrees in I.

(Completeness) For any non-empty subset $\{n_1, \ldots, n_k\} \subseteq I.N$, the corresponding least upper bound must be in Σ. The iteration at line 5 of Algorithm 1 ensures that $n_1 \vee n_2$, $n_1 \vee n_2 \vee n_3$, $\ldots$, $n_1 \vee \ldots \vee n_k$ are all computed. Theorem 1.2 shows that $a \vee b$ is the least upper bound of a and b. Hence, the least upper bound of $\{n_1, \ldots, n_k\}$ must be in Σ. $\square$