

Holding Analysis with Shape Graph

(Technical Report)

Yu Zhang Yxian Zhang

School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, China
 Software Security Lab., Suzhou Institute for Advanced Study, University of Science and Technology of China,
 Suzhou, Jiangsu 215123, China
 yuzhang@ustc.edu.cn

Abstract

1. Shape Graph

1.1 Definition of Shape Graph

Suppose M is a function of a program, and V denotes the set of all pointer variables associated with function M . A shape graph at some control location of M is represented as a directed graph, denoted as $G = \langle \mathcal{N}, E, \mu_K, \mu_P, \mu_S, \mu_H \rangle$, where:

- $\mathcal{N} = N \cup \{n_N, n_D\}$, \mathcal{N} represents the set of nodes, N consists of normal node (eg. a node of a linked list) and abstract node satisfying some predefined predicate of data structure (eg. a segment of a linked list which contains zero or several adjacent list nodes). n_N pointed-to by null pointers, a node n_D pointed-to by dangling pointers, They both don't have relationship with holding.
- $E = E_P \cup E_F$, E represents the set of edges, it consists of points-to edge E_P and field edge E_F . The points-to edge $E_P = \{(v, n) | v \in V, n \in \mathcal{N}\}$, while the field edge $E_F = \{(n_1, f, n_2) | n_1, n_2 \in \mathcal{N}, f \in F\}$, where F is the set of field name.
- $\mu_K : \mathcal{N} \rightarrow K$, $K = \{a, s\}$ represents the map from node to its kind, the node N consists of normal node and abstract node.
- $\mu_P : \mathcal{N} \rightarrow Nat$, 0 represents singly linked list node, and 1 presents doubly linked list node. In order to maintain more data structures, the value of μ_P is defined on Natural number Nat , for example, 2 represents binary tree node, these values can be defined in future.
- $\mu_S : \mathcal{N} \rightarrow B$, $B = \{T, F\}$ represents the map from node to its share information, T represents that the node is shared, in other words, the node can be accessed by many threads simultaneously, F represents that the node is private, in other words, the node can only be accessed by one thread.
- $\mu_H : \mathcal{N} \rightarrow B$, $B = \{T, F\}$ represents the map from node to its holding information, T represents the node has been held, F represents other situation (the node is not held or the node has no holding information), the initial value is F .

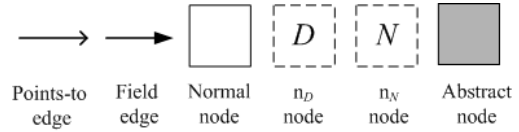


Figure 1. nodes and edges.

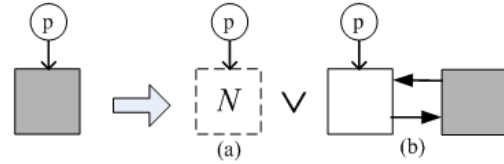


Figure 2. an example of a doubly linked list node unfolding

Figure 1 shows the nodes and edges of shape graph.

1.2 Unfolding and Folding

Unfolding rule is shown in Figure 3. $unfold(G, n)$, G is the shape graph before unfolding, $G = (\mathcal{N}, E, \mu_K, \mu_P, \mu_S, \mu_H)$, n is the abstract node will unfold. after unfolding, the shape graph is G' and $G' = (\mathcal{N}', E', \mu'_K, \mu'_P, \mu'_S, \mu'_H)$, the unfolding rule has two cases: node n is a n_N node or a normal node. When node n is a n_N node, the node n will be changed into n_N node, and its attribute will be removed. When node n is not a n_N node, node n will be changed into a new normal node n' and a abstract node n , the holding information of node n is F and other information can get from node n . the edge $\{(n, n', f)\}$ represents that when it is a singly linked list, node n points to node n' by field edge f , when it is a doubly linked list, node n and n' have total two edges from each other. Figure 2 is an example of a doubly linked list node unfolding.

Folding rule is shown in Figure 4, before folding a shape graph, the shape graph is G and $G = (\mathcal{N}, E, \mu_K, \mu_P, \mu_S, \mu_H)$, after folding nodes into abstract node, the shape graph is G' and $G' = (\mathcal{N}', E', \mu'_K, \mu'_P, \mu'_S, \mu'_H)$. Figure 5 is an example of doubly linked list node folding. in Figure 4, when there are two nodes n_1 and n_2 , they are adjoint nodes and their holding information is F , meaning that they can be folded into a abstract node n_1 . n_2 will be removed from \mathcal{N} , all the information about n_2 also will be removed from G , the edge between n_1 and n_2 will be removed from E .

$$\text{unfold}(G, n) \stackrel{\text{def}}{=} \begin{cases} G' = (N', E', \mu'_K, \mu'_P, \mu'_S, \mu'_H) & \text{case: } n == n_N \\ N' = N - \{n\} \uplus \{n_N\} \\ E' = E \\ \mu'_K = \mu_K - \{n \mapsto a\} \\ \mu'_P = \mu_P - \{n \mapsto \mu_P(n)\} \\ \mu'_S = \mu_S - \{n \mapsto \mu_S(n)\} \\ \mu'_H = \mu_H - \{n \mapsto \mu_H(n)\} \\ \\ G' = (N', E', \mu'_K, \mu'_P, \mu'_S, \mu'_H) & \text{case: } n! = n_N \\ N' = N \uplus \{n'\} \\ E' = E \uplus \{(n, n', f)\} \\ \mu'_K = \mu_K \{n \mapsto s\} \cup \{n' \mapsto a\} \\ \mu'_P = \mu_P \cup \{n' \mapsto \mu_P(n)\} \\ \mu'_S = \mu_S \cup \{n' \mapsto \mu_S(n)\} \\ \mu'_H = \mu_H \cup \{n' \mapsto F\} \end{cases}$$

Figure 3. unfolding rule

$$\text{fold}(G) \stackrel{\text{def}}{=} \begin{cases} G' = (N', E', \mu'_K, \mu'_P, \mu'_S, \mu'_H) \\ \forall n_1, n_2 \in G, \text{two}(G, n_1, n_2), \mu_H(n_1) = F, \mu_H(n_2) = F \\ N' = N - \{n_2\} \\ E' = E - \{(n_1, n_2, f)\} \\ \mu'_K = \mu_K \{n_1 \mapsto a\} - \{n_2 \mapsto \mu_K(n_2)\} \\ \mu'_P = \mu_P - \{n_2 \mapsto \mu_P(n_2)\} \\ \mu'_S = \mu_S - \{n_2 \mapsto \mu_S(n_2)\} \\ \mu'_H = \mu_H - \{n_2 \mapsto F\} \end{cases}$$

Figure 4. folding rule

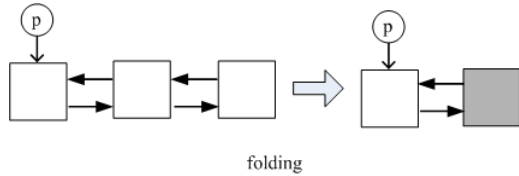


Figure 5. an example of doubly linked list node folding

1.3 Transition of Shape Graph

We introduce a function γ to get all the access paths of a statement s , the function γ is shown in Figure 6, It computes all the access paths in a statement s , and $p \rightarrow f$ is written as pf for short.

Figure 8 shows the assisted definition of shape graph which parts

$$\gamma \begin{cases} \gamma([p = q]^l) = \{p, q\} \\ \gamma([p \rightarrow f = q]^l) = \{p, pf, q\} \\ \gamma([p = q \rightarrow f]^l) = \{p, q, qf\} \\ \gamma([(p \rightarrow f)^i = (q \rightarrow f)^j]^l) = \{p, q, pf, qf, \dots, (pf)^i, (pf)^j\} \\ \gamma([(p \rightarrow f)^i = \text{malloc}(type)]^l) = \{p, pf, \dots, (pf)^i\} \\ \gamma([\text{free}(p)]^l) = \{p\} \\ \gamma([(p \rightarrow f)^i = \text{NULL}]^l) = \{p, pf, \dots, (pf)^i\} \end{cases}$$

Figure 6. the $\gamma(s)$ function

have changed. $(M\{a \rightarrow b\})(x), M(x) - \{a \rightarrow b\}$ represent updating, adding and deleting elements of map respectively.

$$\begin{aligned} G &\stackrel{\text{def}}{=} (N, E, \mu_K, \mu_S, \mu_H) \\ G|_{\mu'_H} &\stackrel{\text{def}}{=} (G.N, G.E, G.\mu_K, G.\mu_S, \mu'_H) \\ G|_{(E', \mu'_S)} &\stackrel{\text{def}}{=} (G.N, E', G.\mu_K, \mu'_S, G.\mu_H) \\ \text{dom}(M\{a \rightarrow b\}) &\stackrel{\text{def}}{=} \begin{cases} \text{dom}(M) & \text{if } a \in \text{dom}(M) \\ \text{dom}(M) \uplus \{a\} & \text{otherwise} \end{cases} \\ (M\{a \rightarrow b\})(x) &\stackrel{\text{def}}{=} \begin{cases} M(x) & \text{if } x = a \\ M(x) \uplus \{b\} & \text{otherwise} \end{cases} \\ M(x) - \{a \rightarrow b\} &\stackrel{\text{def}}{=} \begin{cases} M - \{b\} & \text{if } a \in \text{dom}(M) \\ M(x) & \text{otherwise} \end{cases} \end{aligned}$$

Figure 7. assisted definition

We can preprocess a statement s to make all the node pointed to by the access path in the statement can be found in the shape graph. The function $\gamma(s)$ will get a set R of access paths, The preprocess function is $f_{pre}(G, R)$, for a set R and a shape graph G , it will get a new shape graph G' and M_{atom} , and M_{atom} is a map from access path to node for a statement.

$R = \gamma(s), \langle G', M_{atom} \rangle = \text{unfoldR}(G, R)$
The function $\text{unfoldR}(G, R)$ will iterator all the elements of R , for each r in R it will call function $\text{getNode}(G, r)$, by this function, It will find the node pointed to by r in shape graph G , If it finds the node in shape graph G successfully, It will return the node, if it is an abstract node, it will unfold the node by unfolding rule, otherwise, It will give the error. For example, a statement s is: $p \rightarrow f = q \rightarrow f$, then the $M_{atom} = (p, n_p), (pf, n_{pf}), (q, n_q), (qf, n_{qf})$.

$$\text{getNode}(G, r) = \begin{cases} (G, n) & \text{if } r \in V \wedge (r, n) \in G.E \\ (G', n_r) & \text{if } r == r_p - > f \wedge \\ & (G', n_{r_p}) = \text{getNode}(G, r_p) \wedge \\ & (n_{r_p}, f, n_r) \in G'.E \\ \text{getNode}(G'', r) & \text{if } r = r_p - > f \wedge \\ & (G', n_{r_p}) = \text{getNode}(G, r_p) \wedge \\ & G'.\mu_K(n_{r_p}) = K_A \wedge \\ & G'' = \text{unfold}(G', n_{r_p}) \\ \text{ERROR} & \text{otherwise} \end{cases}$$

$$\text{unfoldR}(G, R) = \begin{cases} (G, \emptyset) & \text{if } R = \emptyset \\ (G'', M_1 \uplus \{r \mapsto n_r\}) & \text{if } R = r \uplus R' \wedge \\ & (G', n_r) = \text{getNode}(G, r) \wedge \\ & (G'', M_1) = \text{unfoldR}(G', R') \end{cases}$$

Figure 8. the unfoldR function

The function $f_{trans}(G, s) = G'$ represents the shape graph G will transfer into G' after a statement s . Assume $G = \langle N, E, \mu_K, \mu_P, \mu_S, \mu_H \rangle, G' = \langle N', E', \mu_K, \mu_P, \mu'_S, \mu_H \rangle$

1. $p(- > f)^i = q(- > f)^j$
 $n = M_{atom}((p \rightarrow f)^i), \mu_S(n) = T$

if $unref(n), \mu'_S = \mu_S(n \rightarrow F)(x)$

- $p(- > f)^i = q(- > f)^j, i = 0, j = 0$
 $E' = E - \{(p, M_{atom}(p))\} \cup \{(p, M_{atom}(q))\}$
- $p(- > f)^i = q(- > f)^j, i = 0, j = 1$
 $E' = E - \{(p, M_{atom}(p))\} \cup \{(p, M_{atom}(q \rightarrow f))\}$
- $p(- > f)^i = q(- > f)^j, i = 1, j = 0$
 $E' = E - \{(M_{atom}(p), f, M_{atom}(p \rightarrow f))\}$
 $\cup \{(M_{atom}(p), f, M_{atom}(q))\}$
- $p(- > f)^i = q(- > f)^j, i \geq 1, j \geq 1$
 $n_1 = M_{atom}((p \rightarrow f)^{i-1}),$
 $n_2 = M_{atom}((p \rightarrow f)^i),$
 $n_3 = M_{atom}((p \rightarrow f)^j),$
 $E' = E - \{(n_1, f, n_2)\} \cup \{(n_1, f, n_3)\}$

2. $p(- > f)^i = \mathbf{malloc}(type)$
 $n = M_{atom}((p \rightarrow f)^i), \mu_S(n) = T$
 if $unref(n), \mu'_S = \mu_S(n \rightarrow F)(x)$

- $p = \mathbf{malloc}(type), i = 0$
 $E' = E - \{(p, n) | (p, n) \in E\} \cup \{(p, n_1) | n_1 \notin \mathcal{N}\}$
- $p(- > f)^i = \mathbf{malloc}(type), i \geq 1$
 $n_0 = M_{atom}((p \rightarrow f)^{i-1})$
 $n_1 = M_{atom}((p \rightarrow f)^i)$
 $E' = E - \{(n_0, f, n_1)\} \cup \{(n_0, f, n_2)\}$

3. $\mathbf{free}(p)$
 $E' = E - \{(p, n_p)\} \cup \{(p, n_D)\}$
 $\mu'_S = \mu_S(n_p \rightarrow F)(x)$

4. $p(- > f)^i \neq NULL$
case true:
 $n_0 = M_{atom}((p \rightarrow f)^i)$
 if $n_0! = n_N, ERROR$
case false:
 $n_0 = M_{atom}((p \rightarrow f)^i)$
 if $n_0 == n_N, ERROR$

2. Holding Analysis

2.1 Global data structure

There are some global data structures will be used to save the holding analysis results and shape graph, they will be used in all holding analysis phases, they are showed as follows:

- G : the current shape graph.
- M_{acq} : holding acquire information of map from access path to node before current statement.
- M_{rel1} : holding release information of map from access path to node before current statement.
- M_{rel2} : holding release information of map from access path to node after current statement.

The initial shape graph $G = \langle \mathcal{N}, E, \mu_K, \mu_P, \mu_S, \mu_H \rangle$.

2.2 Preprocess

$f_{pre} : \mathbb{G} \times S \rightarrow \mathbb{G} \times \mathbb{M}$

$f_{pre}(G, s) = (G', M_{atom})$

Only V has not been changed, all the nodes pointed to by the access path in s can be found in the shape graph, and these access path and the nodes pointed to by them will be recorded in M_{atom} .

case $[l_1 = e]^l$:

case $[l_1 = \mathbf{malloc}(type)]^l$:

case $[\mathbf{free}(l_1)]^l$:

$R = \gamma(s), unfoldR(G, R) \mapsto (G', M_{atom})$

2.3 Holding acquire

2.3.1 Implicit holding acquire

$f_{impl} : \mathbb{G} \times S \rightarrow \mathbb{G} \times \mathbb{M}$

$f_{impl}(G, s) = (G', M_{impl})$

Only μ_H has changed, if the nodes pointed to by the right hand of statement s haven't been held, change their holding nature into T and add them in to M_{impl} , then get shape graph G' .

case $[l_1 = e]^l$:

$n_e = M_{atom}(e)$

$\mu'_H = \mu_H\{n_e \rightarrow T\}(x), M_{impl} = M_{impl}\{e \rightarrow n_e\}(x),$

where $\mu_H(n_e) = F$

$G' = G|_{\mu'_H}$

case $[l_1 = \mathbf{malloc}(type)]^l$:

there is nothing to do

case $[\mathbf{free}(l_1)]^l$:

there is nothing to do

2.3.2 Atomic holding acquire

$f_{atom} : \mathbb{G} \times \mathbb{M} \rightarrow \mathbb{G} \times \mathbb{M}$

$f_{atom}(G, M_{atom}) = (G', M'_{atom}), M'_{atom} \subset M_{atom}$

Only μ_H has changed, if the nodes pointed to by access path or variable haven't been held, change their holding nature into T .

case $[l_1 = e]^l$:

case $[l_1 = \mathbf{malloc}(type)]^l$:

case $[\mathbf{free}(l_1)]^l$:

$G' = G|_{\mu'_H}$

$\forall r \in dom(M_{atom}), n = M_{atom}(r), \mu_H(n) = T, M'_{atom} = M_{atom}\{r \rightarrow n_r\}(x)$

2.3.3 Prerecord of holding release

$f_{rel} : S \rightarrow \mathbb{M}$

$f_{rel}(s) = M_{pre}$

The shape graph hasn't changed, if the statement s is a define statement to pointer variable q , then q will be added into M_{pre} .

case $[l_1 = e]^l$:

$M_{pre}\{l_1 \rightarrow M_{atom}(l_1)\}, \mathbf{where} \mathit{decl}(l_1)$

case $[l_1 = \mathbf{malloc}(type)]^l$:

there is nothing to do

case $[\mathbf{free}(l_1)]^l$:

there is nothing to do

2.4 Shape graph transition

$f_{trans} : \mathbb{G} \times S \rightarrow \mathbb{G}$

$f_{trans}(G, s) \rightarrow G'$

Only μ_S and E have changed, by the shape graph transition rule, the shape graph G will transfer into G' .

case $[l_1 = e]^l$:

case $[l_1 = \mathbf{malloc}(type)]^l$:

case $[\mathbf{free}(l_1)]^l$:

$G' = G|_{(E', \mu'_S)}$

2.5 Holding release

2.5.1 Atomic holding release

$f'_{atom} : \mathbb{G} \times \mathbb{M} \rightarrow \mathbb{G} \times \mathbb{M}$

$f'_{atom}(G, M'_{atom}) = (G, M'_{atom})$

Only μ_H has changed, the node in M'_{atom} will be atomic released, then get shape graph G' .

case $[l_1 = e]^{l'}$:
case $[l_1 = \text{malloc}(type)]^{l'}$:
case $[\text{free}(l_1)]^{l'}$:

$G' = G|_{\mu'_H}$

$\forall r \in \text{dom}(M'_{atom}), n = M'_{atom}(r) \rightarrow$

$\mu'_H = \mu_H \{n \rightarrow F\}(x)$

2.5.2 Implicit holding release

$f'_{rel} : \mathbb{G} \times \mathbb{M} \times S \rightarrow \mathbb{G} \times \mathbb{M}$

$f'_{rel}(G, M_{pre}, s) = (G', M_{rel1})$

Only μ_S has changed, M_{rel1} begin form \emptyset , if node have the release condition(2), the node in M_{pre} will be released, if node have the condition(1),(3), then the node will be released and added into M_{rel1} , then get shape graph G' .

case $[l_1 = e]^{l'}$:
case $[l_1 = \text{malloc}(type)]^{l'}$:
case $[\text{free}(l_1)]^{l'}$:

$G' = G|_{\mu'_H}$

$\forall r \in \text{dom}(M_{pre}), n = M_{pre}(r), \mu_H(n) = T \wedge$

$\exists \text{unref}(G, n), M_{rel1} = M_{rel1} \{r \rightarrow n\}(x)$

$\mu_H(n_r) = T \wedge (\mu_S(n_r) = F || \text{!slive}(p, r))$

$M_{rel1} = M_{rel1} \{r \rightarrow n_r\}(x)$

2.6 Postprocess

compute holding acquire: $M_{acq} = M_{acq} \uplus M'_{atom} \uplus M_{impl}$,

compute holding release: $M_{rel2} = M_{rel2} \uplus M'_{atom}$

by the rule of fold(G), if the node have the fold condition, then all the nodes will be folded, can get the shape graph G' .

2.7 Data flow equation

The paper uses a forward data flow analysis, the in set of data flow of current statement is computed by the union of all it's predecessor's out set, where the out set is computed by itself in set.

Analysis direction: forward data flow analysis

Element of the set: $\langle \mathbb{G}, M_{acq}, M_{rel1}, M_{rel2} \rangle, \mathbb{G}$ represents the power set of holding analysis graph, $M_{acq}, M_{rel1}, M_{rel2}$ represents the new holding acquire or release information of map from access path to node.

$$H_{in}([s]^{l'}) = \begin{cases} \langle G_0, \emptyset, \emptyset, \emptyset \rangle & \text{if } l \in \text{init}(S) \\ \uplus H_{out}([s]^{l'}) & (l', l) \in \text{flow}(S) \end{cases}$$

$$H_{out}([s]^{l'}) = f_{trans}(\mathbb{G}, M_{acq}, M_{rel1}, M_{rel2}, s) \\ \langle \mathbb{G}, M_{acq}, M_{rel1}, M_{rel2} \rangle = \text{Min}([s]^{l'})$$

for $H_{in}([s]^{l'}) = \uplus H_{out}([s]^{l'})$:

1. Union of shape graph

• Multi-branch case

For each sub-branch graph of Multi-branch case, as they may not be the same, so at their convergence, the results are computed by the union of holding analysis sub-graph.

• Subgraph generated by iteration

merge the G_1 into the current power set \mathbb{G} , then:

$\text{merge}(\mathbb{G}, G_1) =$
 $\text{flag} := \text{false};$
for $G \in \mathbb{G}$ **do**
 if $\text{sameG}(G, G_1)$

return $\mathbb{G};$
else if $\text{subG}(G_1, G)$
 return $\mathbb{G};$
else if $\text{subG}(G_1, G)$
 return $(\mathbb{G} - \{G\} \uplus \{G_1\});$
else if (!flag)
 continue;
else
 return $(\mathbb{G} \uplus \{G_1\});$
endfor

2. Unify of holding analysis results

• Unify of holding analysis results

• Subgraph generated by iteration

$\text{unify}(R, r_1) =$
for $r \in R$ **do**
 if $r = r_1$
 return $R;$
 else
 return $(R \uplus \{r_1\});$
endfor