Vortrag zur Shape Analyse

Daniel Fritsch

24. Juni 2009

Inhaltsverzeichnis

1. Einleitung
   - Neue Elemente der While Sprache
   - Bedeutung der Elemente
1. Shape Graphen
   - Abstrakte Locations
   - Abstrakte States
   - Abstrakte Heaps
   - Sharing Information
1. Analyse
   - Monotone Framework
   - Transferfunktionen

Inhaltsverzeichnis

1. Einleitung
   - Neue Elemente der While Sprache
   - Bedeutung der Elemente

1. Shape Graphen
   - Abstrakte Locations
   - Abstrakte States
   - Abstrakte Heaps
   - Sharing Information

1. Analyse
   - Monotone Framework
   - Transferfunktionen

Inhaltsverzeichnis

1. Einleitung
   - Neue Elemente der While Sprache
   - Bedeutung der Elemente
1. Shape Graphen
   - Abstrakte Locations
   - Abstrakte States
   - Abstrakte Heaps
   - Sharing Information
1. Analyse
   - Monotone Framework
   - Transferfunktionen

Ziele der Shape Analyse

- Erkennung von deferenzieren von Null-Pointern
- Erkennung von Zugriffe auf deallocated storage
- Bestimmung der Erreichbarkeit einer Heap-Zelle (Garbage Collection..)
- etc

Neue Elemente der While Sprache

Erweiterung der While-Sprache um neue Elemente, sodass Zellen im Heap gespeichert werden können:

- Selektoren: $sel \in \mathbf{Sel}$
- Pointer: $p \in \mathbf{PExp}$, mit $p ::= x \mid x.sel$
- malloc - Anweisung

Komplette While-Sprache

$$a ::= p \mid n \mid a_1\ op_a\ a_2 \mid \texttt{nil}$$

$$b ::= \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \mid op_p\ p$$

$$S ::= [p{:=}a]^l \mid [\texttt{skip}]^l \mid S_1; S_2 \mid \texttt{if}\ [b]^l\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ [b]^l\ \texttt{do}\ S \mid [\texttt{malloc}\ p]^l$$

Strukturelle Semantik

Um die obere Sprache auch modellieren zu können, werden nun einige Begriffe eingeführt, nämlich die der Locations, States und Heaps.

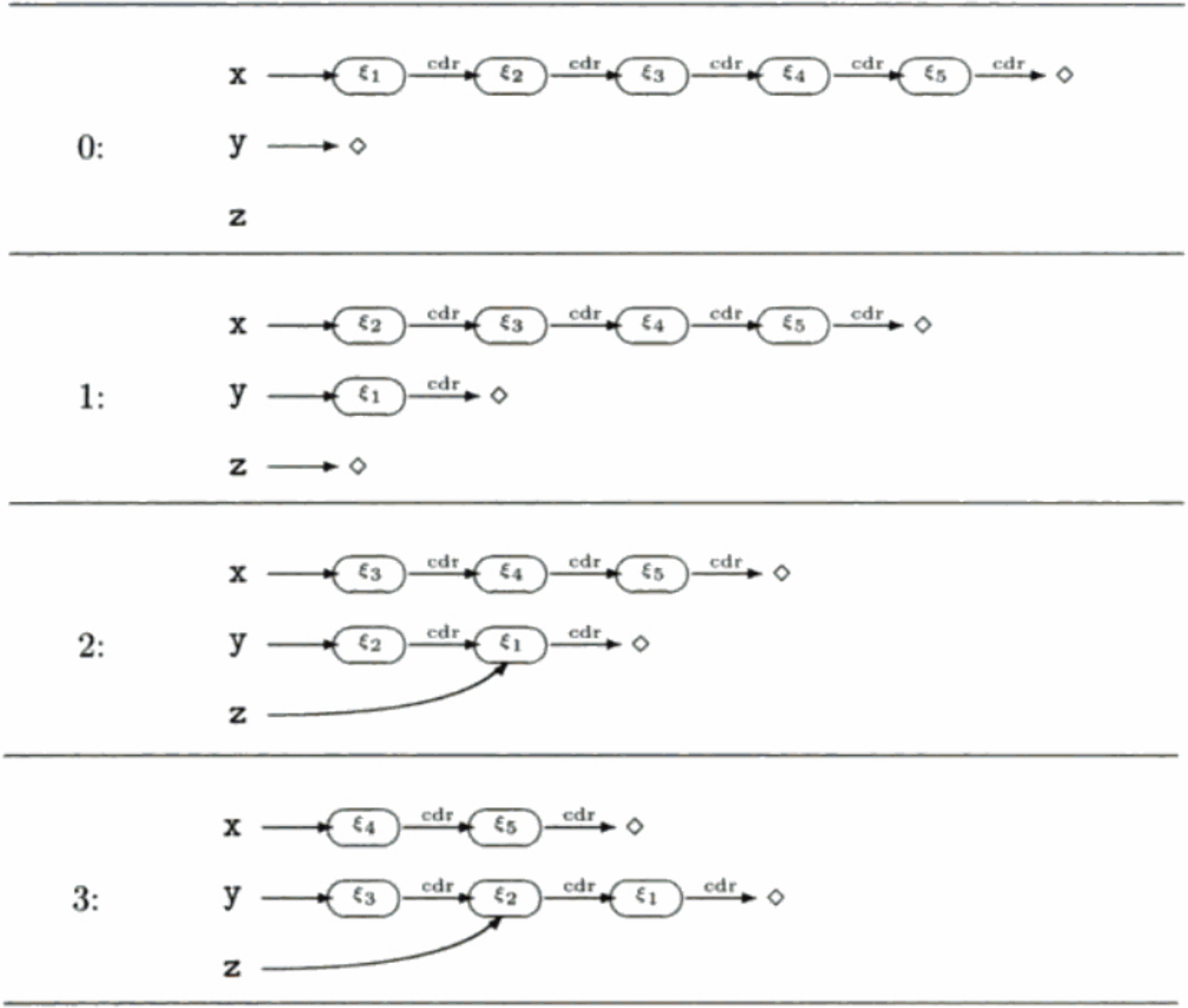
- $\xi \in \mathbf{Loc}$
- $\sigma \in \mathbf{State} = \mathbf{Var}_\star \to (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$
- $\mathcal{H} \in \mathbf{Heap} = (\mathbf{Loc} \times \mathbf{Sel}) \to_{fin} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

Beispiel

Programm zum Umdrehen (reverse) von verketteten Listen:

```
[y:= nil]1
while[not is-nil(x)]2 do
    ([z:= y]3;[y:= x]4;[x:= x.cdr]5;[y.cdr:= z]6);
[z:=nil]7
```

Ausführung des Programms



Pointer Ausdrücke

Pointer-Ausdruck p muss Element von Z + Loc + $\{\diamond\}$ zurückgeben, deshalb wird

$$\varrho : \mathbf{PExp}_\star \to (\mathbf{State} \times \mathbf{Heap}) \to_{fin} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$$

eingeführt.

Definition

$$\varrho[[x]](\sigma, \mathcal{H}) = \sigma(x)$$

$$\varrho[[x.sel]](\sigma, \mathcal{H}) = \begin{cases} \mathcal{H}(\sigma(x), sel) \\ \quad \text{wenn } \sigma(x) \in \mathbf{Loc} \text{ und} \\ \quad \mathcal{H} \text{ ist definiert auf } (\sigma(x), sel) \\ undef \\ \quad \text{wenn } \sigma(x) \notin \mathbf{Loc} \text{ oder} \\ \quad \mathcal{H} \text{ ist undefiniert auf } (\sigma(x), sel) \end{cases}$$

Malloc

Die Malloc-Anweisung ist verantwortlich für das Erstellen neuer Zellen im Heap

Definition

$\langle [\texttt{malloc}\ \ x]^l, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma[x \mapsto \xi], \mathcal{H} \rangle$

dabei kommt ξ weder in σ noch $\mathcal{H}$ vor

$\langle [\texttt{malloc}\ \ x.sel]^l, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma, \mathcal{H}[(\sigma(x), sel) \mapsto \xi] \rangle$

dabei kommt ξ weder in σ noch $\mathcal{H}$ vor und $\sigma(x) \in \mathbf{Loc}$

Weitere Änderungen

Es werden auch weitere Elemente der While-Sprache geändert (z.B. Operatoren und Zuweisungen), sodass sie mit Pointer und Selektoren umgehen können, z.B.

- beim Speichern von Pointern in Variablen
- beim (boolschen) Vergleich von zwei Operatoren
- etc

Shape Graphen

- es existieren Programme für die das Heap unbegrenzt wachsen kann
- um effektiv Aussagen treffen zu können, müssen Heaps also endlich dargestellt werden
- zu diesem Zweck werden Shape Graphen eingeführt, ein Tripel aus abstrakte Heaps, abstrakte States und die sharing Information

Abstrakte Location

Definition

ALoc $= \{n_X \mid X \subseteq \mathbf{Var}_\star\}$

- wenn $x \in X$, dann stellt n_X (unter anderen) die Location $\sigma(x)$ dar
- abstrakte Location $n_\emptyset$ stellt Locations dar, die nicht von einem State erreichbar sind

Zudem wird verlangt:

Invariante 1: Wenn 2 abstrakte Locations n_X und n_Y im gleichen Shape-Graph vorkommen, so ist entweder $X = Y$ oder $X \cap Y = \emptyset$

Abstrakte States

Definition

$S \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc})$

- abstrakte States ordnen einer Variable x eine abstrakte Location n_X zu

Es wird verlangt:

Invariante 2: Wenn die Variable x einer abstrakten Location n_X zugeordnet wird, so ist $x \in X$

Zusammen mit Invariante 1 folgt daraus, dass jede Variable höchstens in einer abstrakten Location vorkommen darf

Abstrakte Heaps

Definition

$H \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc})$

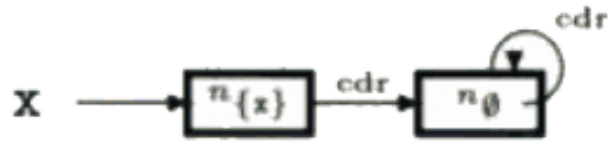
- abstrakte Heaps sind Verbindungen zwischen zwei abstrakte Locations

Es wird verlangt:

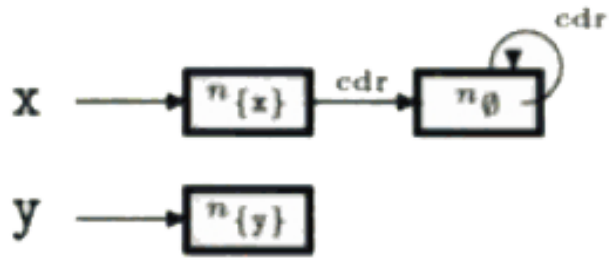
Invariante 3: Wenn (n_X, sel, n_Y) und (n_X, sel, n_Y') in einem abstrakten Heap sind, so ist entweder $X = \emptyset$, oder $Y = Y'$

Beispiel

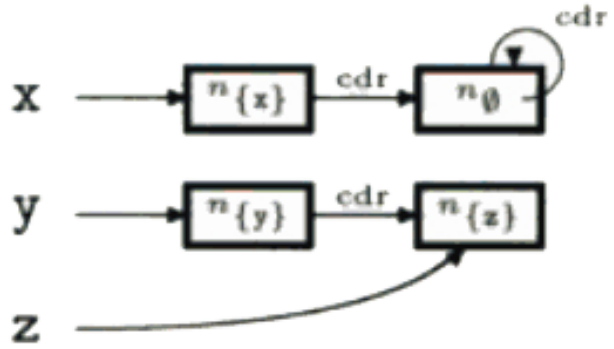
0:



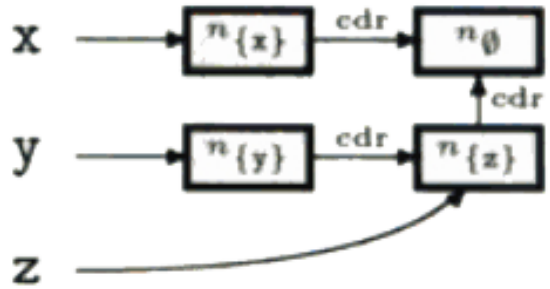
1:



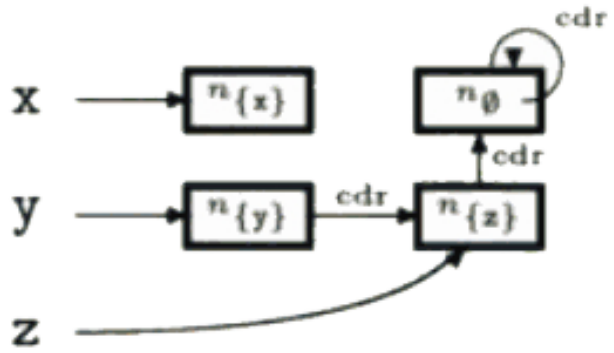
2:



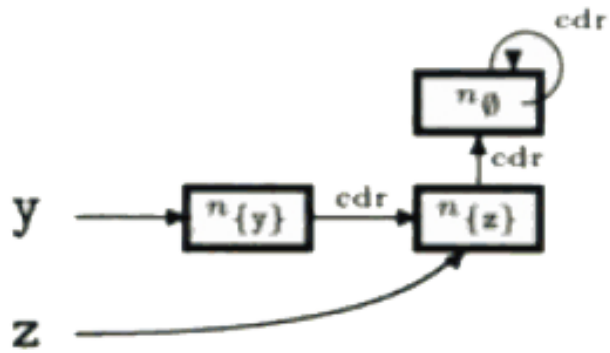
3:



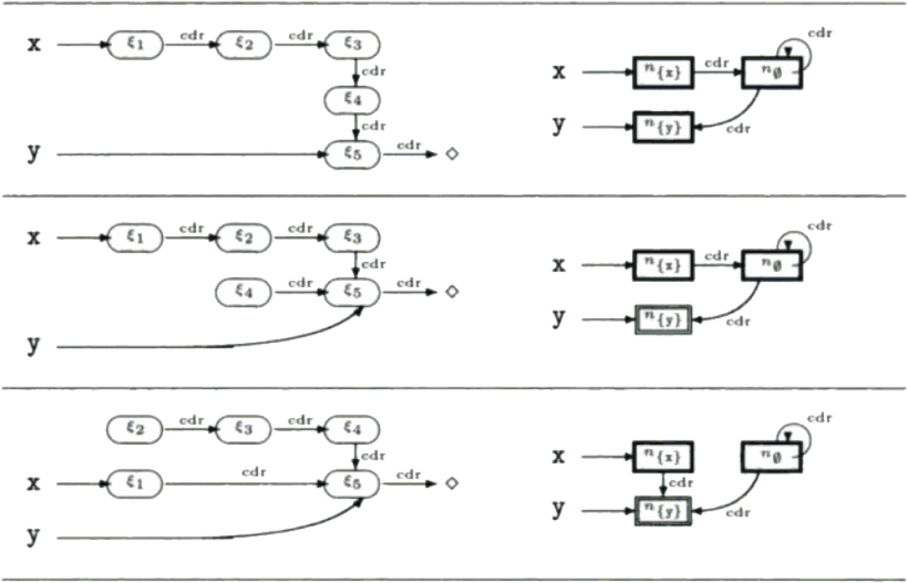
4:



5:



Problem



Sharing Information

- `is` ist eine Teilmenge von abstrakten Locations, die eine Location darstellen, die geteilt wird
- n_X ist Element von `is`, wenn es eine Location darstellt, die von mehr als ein Pointer gezielt wird

Es wird verlangt:

Invariante 4: Wenn $n_X \in$ is, dann ist entweder:
a) $(n_\emptyset, sel, n_X)$ ist im abstrakten Heap für mehr als ein sel, oder
b) es existieren zwei verschiedene Tripel (n_Y, sel_1, n_X) und $(n_{Y'}, sel_2, n_X)$ im abstrakten Heap (also ist entweder $sel_1 \neq sel_2$ oder Y $\neq$ Y')

und **Invariante 5:** Wenn zwei verschiedene Tripel (n_Y, sel_1, n_X) und $(n_{Y'}, sel_2, n_X)$ im abstrakten Heap existieren und $n_X \neq n_\emptyset$,
so ist $n_X \in$ is.

Zusammenfassung

Definition

$S \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc})$

$H \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc})$

$is \in \mathbf{IsShared} = \mathcal{P}(\mathbf{ALoc})$

ein Shape Graph (S, H, is) heißt *kompatibel* wenn es die fünf oben genannten Invarianten erfüllt Die Menge der kompatiblen Shape Graphen wird als

$\mathbf{SG} = \{ (S, H, is) \mid (S, H, is) \text{ ist kompatibel } \}$

bezeichnet.

Analyse

Die Analyse ist eine Instanz eines *monotonen Frameworks*, also eine Menge von Gleichungen der Form

Definition

$$Shape_\circ(l) = \begin{cases} \iota & \text{für } l = init(S_\star), \text{ sonst:} \\ \cup\{Shape_\bullet(l') \mid (l', l) \in flow(S_\star)\} \end{cases}$$

$$Shape_\bullet(l) = f_l^{SA}(Shape_\circ(l))$$

wobei $\iota \in \mathcal{P}(\mathbf{SG})$ der Extremwert beim Eingang in $S_\star$ ist, und f_l^{SA} Transfer-Funktionen sind, die noch zu spezifizieren sind

Zur Erinnerung

```
[y:= nil]^1
while[not is-nil(x)]^2 do
    ([z:= y]^3;[y:= x]^4;[x:= x.cdr]^5;[y.cdr:= z]^6);
[z:=nil]^7
```

Resultierende Gleichungen für $\text{Shape}_\bullet(I)$

$$\text{Shape}_\bullet(1) = \text{f}_1^{SA}(\text{Shape}_\circ(1)) = \text{f}_1^{SA}(\iota)$$
$$\text{Shape}_\bullet(2) = \text{f}_2^{SA}(\text{Shape}_\circ(2)) = \text{f}_2^{SA}(\text{Shape}_\bullet(1) \cup \text{Shape}_\bullet(6))$$
$$\text{Shape}_\bullet(3) = \text{f}_3^{SA}(\text{Shape}_\circ(3)) = \text{f}_3^{SA}(\text{Shape}_\bullet(2))$$
$$\text{Shape}_\bullet(4) = \text{f}_4^{SA}(\text{Shape}_\circ(4)) = \text{f}_4^{SA}(\text{Shape}_\bullet(3))$$
$$\text{Shape}_\bullet(5) = \text{f}_5^{SA}(\text{Shape}_\circ(5)) = \text{f}_5^{SA}(\text{Shape}_\bullet(4))$$
$$\text{Shape}_\bullet(6) = \text{f}_6^{SA}(\text{Shape}_\circ(6)) = \text{f}_6^{SA}(\text{Shape}_\bullet(5))$$
$$\text{Shape}_\bullet(7) = \text{f}_7^{SA}(\text{Shape}_\circ(7)) = \text{f}_7^{SA}(\text{Shape}_\bullet(2))$$

die Transferfunktion $f_I^{SA} : \mathcal{P}(\mathbf{SG}) \rightarrow \mathcal{P}(\mathbf{SG})$ hat folgende Form:

$$f_I^{SA}(SG) = \cup\{\phi_I^{SA}((\text{S, H, is})) \mid (\text{S, H, is}) \in SG\}$$

wobei $\phi_I^{SA} : \mathbf{SG} \rightarrow \mathcal{P}(\mathbf{SG})$ festlegt, wie ein *einzelner* Shape Graph in $\text{Shape}_\circ(I)$ zu einer *Menge* von Shape Graphen in $\text{Shape}_\bullet(I)$ wird

Transferfunktion für $[b]^{\prime}$ und $[skip]^{\prime}$

bool'sche Tests und die Anweisung `skip` verändern den Heap nicht, also:

$\phi_{I}^{SA}((S, H, is)) = \{(S, H, is)\}$

das heißt hier ist die Transferfunktion die Identität.

Transferfunktion für $[x{:=}a]^l$

- Verbindung von x entfernen
- x aus abstrakten Locations entfernen

Dies geschieht durch die Funktion $\phi_l^{SA}((\mathrm{S}, \mathrm{H}, \mathrm{is})) = \{kill_x((\mathrm{S}, \mathrm{H}, \mathrm{is}))\}$ mit $kill_x((\mathrm{S}, \mathrm{H}, \mathrm{is}))$ folgendermaßen definiert:

Definition

$\mathrm{S'} = \{(z, k_x(n_Z)) \mid (z, n_Z) \in S \wedge z \neq x)\}$

$\mathrm{H'} = \{(k_x(n_V), sel, k_x(n_W)) \mid (n_V, sel, n_W) \in H\}$

$\mathrm{is'} = \{(k_x(nx)) \mid (nx) \in \mathtt{is}\}$

dabei ist $k_x(n_Z) = n_{Z \setminus \{x\}}$

Transferfunktion für $[x{:=}y]^l$

- für x = y trivial
- Verbindungen zu x entfernen (durch $kill_x$)
- neue Verbindung zu x erstellen, d.h. abstrakte Locations die y enthalten sollen nun auch x enthalten

Also ist $\phi_l^{SA}((S, H, is)) = \{(S'', H'', is'')\}$

Definition

$$S'' = \{(z, g_x^y(n_Z)) \mid (z, n_Z) \in S'\} \cup \{(x, g_x^y(n_Y)) \mid (y', n_Y) \in S' \wedge y' = y\}$$

$$H'' = \{(g_x^y(n_V), sel, g_x^y(n_W)) \mid (n_V, sel, n_W) \in H'\}$$

$$is'' = \{(g_x^y(n_Z) \mid n_Z \in is'\}$$

mit (S', H', is') = $kill_x((S, H, is))$ und $g_x^y(n_Z) = \begin{cases} n_{Z\cup\{x\}} & \text{wenn } y \in Z \\ n_Z & \text{sonst} \end{cases}$

Transferfunktion für $[x{:=}y.sel]^l$

falls x = y kann man die Anweisung folgendermaßen umschreiben:

$[t := y.sel]^{l_1}; [x := t]^{l_2}; [t := \texttt{nil}]^{l_3}$

t ist eine neue (temporäre) Variable, und l_1, l_2, l_3 neue Labels. Die Transferfunktion f_l^{SA} ist dann:

$$f_l^{SA} = f_{l_3}^{SA} \circ f_{l_2}^{SA} \circ f_{l_1}^{SA}$$

$f_{l_2}^{SA}$ und $f_{l_3}^{SA}$ sind bekannt. Es soll nun die Transferfunktion $f_{l_1}^{SA}$ untersucht werden oder die gleichbedeutende f_l^{SA} im Falle x $\neq$ y, also muss die abstrakte Location die y.sel entspricht so umbenannt werden, dass sie auch x beinhaltet

Transferfunktion für $[x{:=}y.sel]'$

3 Möglichkeiten

1. im Shape Graph sind y oder y.sel entweder ein Integer oder `nil` oder undefiniert sind
   (es existiert keine abstrakte Location n_Y mit $(y, n_Y) \in$ S' oder es gibt eine abstrakte Location n_Y mit $(y, n_Y) \in$ S' aber kein n_Z mit $(n_Y, sel, n_Z) \in$ H')
2. im Shape Graph wird die Location die von y.sel gezielt wird auch von anderen Variablen (in U) gezielt
   (es existiert eine abstrakte Location n_Y mit $(y, n_Y) \in$ S' und es existiert eine abstrakte Location $n_U \neq n_\emptyset$ mit $(n_Y, sel, n_U) \in$ H')
3. im Shape Graph zielt keine andere Variable zur Location, auf die y.sel zielt
   (es gibt eine abstrakte Location n_Y mit $(y, n_Y) \in$ S' und $(n_Y, sel, n_\emptyset) \in$ H')

Transferfunktion für $[x{:=}y.sel]^l$

Fall 1:
entweder es existiert kein n_Y mit $(y, n_Y) \in$ S':

- es existiert keine abstrakte Location für y.sel
- keine abstrakte Locations zum umbenennen sodass sie x enthalten
- $\phi_l^{SA}((S, H, is)) = \{kill_x((S, H, is))\}$

oder es gibt eine abstrakte Location n_Y mit $(y, n_Y) \in$ S' aber keine abstrakte Location n_Z mit $(n_Y, sel, n_Z) \in$ H':

- aus den Invarianten folgt: n_Y ist eindeutig
- keine abstrakte Locations zum umbenennen sodass sie x enthalten
- $\phi_l^{SA}((S, H, is)) = \{kill_x((S, H, is))\}$

Transferfunktion für $[x{:=}y.sel]'$

Fall 2:
es existiert ein n_Y mit $(y, n_Y) \in$ S' und eine abstrakte Location $n_U \neq n_\emptyset$ mit $(n_Y, sel, n_U) \in$ H'

- n_Y und n_U sind eindeutig (Invarianten)
- n_U wird so umbenannt, dass es x enthält
  durch $h_x^U(n_Z) = \begin{cases} n_{U \cup \{x\}} & \text{wenn } Z = U \\ n_Z & \text{sonst} \end{cases}$
- $\phi_l^{SA}((S, H, is)) = \{((S'', H'', is''))\}$
  mit $(S', H', is') = kill_x((S, H, is))$ und

Definition

$$S'' = \{(z, h_x^U(n_Z)) \mid (z, n_Z) \in S'\} \cup \{(x, h_x^U(n_U))\}$$

$$H'' = \{(h_x^U(n_V), sel', h_x^U(n_W)) \mid (n_V, sel', n_W) \in H'\}$$

$$is'' = \{(h_x^U(n_Z) \mid n_Z \in is'\}$$

Transferfunktion für $[x.sel{:=}a]^l$

- falls es kein n_X mit $(x, n_X) \in$ S gibt, so ist die Transferfunktion trivial
- falls es ein n_X mit $(x, n_X) \in$ S gibt, aber kein n_U mit $(n_X, sel, n_U) \in$ H (die Zelle auf die x.sel zielt, zielt auf keine weitere Zelle) ist die Transferfunktion auch trivial
- falls es ein n_X mit $(x, n_X) \in$ S und n_U mit $(n_X, sel, n_U) \in$ H gibt, muss (n_X, sel, n_U) aus H entfernt werden, und is aktualisiert werden
  $\phi_l^{SA}((S, H, is)) = \{kill_{x.sel}((S, H, is))\}$
  mit $kill_{x.sel}((S, H, is))$ definiert als

Definition

S' = S

$$H' = \{(n_V, sel', n_W) \mid (n_V, sel', n_W) \in H \wedge \neg(X = V \wedge sel = sel')\}$$

$$is' = \begin{cases} is \backslash \{n_U\} & \text{wenn } n_U \in is \wedge \#into(n_U, H') \leq 1 \wedge \\ & \quad \neg\exists sel' : (n_\emptyset, sel', n_U) \in H' \\ is & \text{sonst} \end{cases}$$

Weitere Transferfunktionen

Für $[x.sel := y]^l$ und
$[x.sel{:=}y.sel']^l$ kann man auf ähnlicherweise die Transferfunktionen erstellen. Siehe Skript

Transferfunktion für $[\text{malloc } p]^l$

Falls p die Form x hat:

- Verbindung von x entfernen
- neue (ungeteilte) Location erstellen und diese an x binden
- $\phi_l^{SA}((S, H, is)) = \{(S' \cup \{(x, n_{\{x\}})\}, H', is')\}$
  mit $(S', H', is') = kill_x(S, H, is)$

Falls p die Form x.sel hat:

- Anweisung wird umgeschrieben:
  $[\texttt{malloc } t]^{l_1}; [x.sel := t]^{l_2}; [t := \texttt{nil}]^{l_3}$
  t ist eine neue (temporäre) Variable, und l_1, l_2, l_3 neue Labels
- Transferfunktion: $f_l^{SA} = f_{l_3}^{SA} \circ f_{l_2}^{SA} \circ f_{l_1}^{SA}$
  die alle schon bekannt sind