Derivation of breath first search

COMP2111 assignment 3

Dae Ro Lee z5060887 and Wing Feng z5091907

May 24, 2017

1 Introduction

The derivation of BREATH FIRST SEARCH on a tree using a bounded QUEUE.

2 The Derivation

$$\textbf{proc}\ \textsc{search}(\textbf{value}\ t, \textbf{value}\ N, \textbf{value}\ k, \textbf{result}\ v, \textbf{result}\ f)\cdot$$

$$\llcorner t, N, k, v, f : \left[\begin{array}{l} \forall x \in V_t(x \in \Gamma_t^*(r_t) \land x \notin \Gamma_t^+(x)) \land \max_{i\in\mathbb{N}} |\Gamma_t^i(r_t) \cup \Gamma_t^{i+1}(r_t)| \leq N, \\ (f \land \exists w \in V_{t_0}(k_{t_0}(w) = k_0 \land \lambda_{t_0}(w) = v))\lor \\ (\neg f \land \forall w \in V_{t_0}(k_{t_0}(w) \neq k_0)) \end{array}\right]\lrcorner^{(1)}$$

$$(1) \sqsubseteq \quad \langle \textbf{c-frame} \rangle$$

$$v, f : \left[\begin{array}{l} \forall x \in V_t(x \in \Gamma_t^*(r_t) \land x \notin \Gamma_t^+(x)) \land \max_{i\in\mathbb{N}} |\Gamma_t^i(r_t) \cup \Gamma_t^{i+1}(r_t)| \leq N, \\ (f \land \exists w \in V_t(k_t(w) = k \land \lambda_t(w) = v))\lor \\ (\neg f \land \forall w \in V_t(k_t(w) \neq k)) \end{array}\right]$$

$$\sqsubseteq \quad \langle \textbf{introduce local variable} \rangle$$

$$\textbf{var}\ q, n \cdot \llcorner q, n, v, f : \left[\begin{array}{l} \forall x \in V_t(x \in \Gamma_t^*(r_t) \land x \notin \Gamma_t^+(x)) \land \max_{i\in\mathbb{N}} |\Gamma_t^i(r_t) \cup \Gamma_t^{i+1}(r_t)| \leq N, \\ (f \land \exists w \in V_t(k_t(w) = k \land \lambda_t(w) = v))\lor \\ (\neg f \land \forall w \in V_t(k_t(w) \neq k)) \end{array}\right]\lrcorner^{(2)}$$

We define the loop invariant for BREATH FIRST SEARCH as:

$$I := \left(\begin{array}{l} \left(\begin{array}{l} \neg f \land 0 \leqslant n \leqslant N \land K_t(tmp) \neq k \\ \lor(f \land K_t(tmp) = k \land \lambda_t(tmp) = v) \end{array} \right) \\ \land \forall x \in V_t(x \in \Gamma_t^*(r_t) \land x \notin \Gamma_t^+(x)) \end{array} \right)$$

$(2) \sqsubseteq \qquad \langle \text{seq} \rangle$

$\llcorner q, n, v, f : [pre(2), I] \lrcorner_{(3)}$

$; \llcorner q, n, v, f : [I, post(2)] \lrcorner_{(4)}$

$(3) \sqsubseteq \qquad \langle \text{seq} \rangle$

$\llcorner q, n, v, f : [pre(3), pre(3) \wedge q =<> \wedge \neg f] \lrcorner_{(5)}$

$; \llcorner q, n, v, f : [pre(3) \wedge q =<> \wedge \neg f, I] \lrcorner_{(6)}$

$(5) \sqsubseteq \qquad \langle \text{ass} \rangle$

$q := initialise(N);\; f := \neg f$

$(6) \sqsubseteq \qquad \langle \text{i-loc} \rangle$

$\mathbf{var}\ tmp \cdot \llcorner tmp, q, n, v, f : [pre(3) \wedge q =<> \wedge \neg f, I] \lrcorner_{(7)}$

$\sqsubseteq \qquad \langle \textbf{ass} \rangle$

$q :=< r_t >$

$n := 1$

$(4) \sqsubseteq \qquad \langle \textbf{s-post} \rangle$

$q, n : [I \wedge\ g, I \wedge (f \vee\ q =<>)]$

$\sqsubseteq \qquad \langle \textbf{while} \rangle$

$\mathbf{while}\ n \neq 0 \wedge \neg f\ \mathbf{do}$

$\qquad \llcorner q, n : [I \wedge\ g, I] \lrcorner_{8}$

$\mathbf{od}$

$(8) \sqsubseteq \qquad \langle \textbf{seq} \rangle$

$\llcorner n, q : [g \wedge\ I \wedge\ q =< z, qt >, q = qt \wedge tmp = z] \lrcorner_{(a)}$

$; \llcorner n, q : [q = qt \wedge tmp = z, I] \lrcorner_{(b)}$

$(b) \sqsubseteq \qquad \langle \textbf{if} \rangle$

$\mathbf{if}\ k_t(tmp) = k\ \mathbf{then}$

$\qquad \llcorner f, v : [\neg\ f \wedge k_t(tmp) = k, f \wedge\ v = \lambda_t(tmp)] \lrcorner_{(c)}$

$\mathbf{else}$

$\qquad \llcorner [k_t(tmp) \neq k, q = q_0 \cdot \Gamma\ (tmp)] \lrcorner_{(d)}$

$\mathbf{fi}$

$(a) \sqsubseteq \qquad \langle \textbf{ass} \rangle$

$$
\begin{array}{ll}
 & tmp := dequeue(q, n) \\
(c) \sqsubseteq & \quad \langle \mathbf{ass} \rangle \\
 & f := true; v = \lambda_t(tmp) \\
(d) \sqsubseteq & \quad \langle \mathbf{ass} \rangle \\
 & addallchildrentoq
\end{array}
$$

i-loc tmp $\implies k_t(tmp) \neq$ k, tmp is a variable that is not mapped to anything hence k_t does not exist.
$n = 1$ after assignment and N is the maximum number of nodes available. hence $pre + blah \implies invariant$.

3 Code

Putting the code together we have

$$q := initialise(N) \tag{1}$$
$$f := \neg f \tag{2}$$
$$q :=< r_t > \tag{3}$$
$$n := 1 \tag{4}$$
$$\textbf{while } n \neq 0 \wedge \neg f \textbf{ do} \tag{5}$$
$$\quad tmp := dequeue(q, n) \tag{6}$$
$$\quad \textbf{if } k_t(tmp) = k \textbf{ then} \tag{7}$$
$$\quad\quad f := f \tag{8}$$
$$\quad\quad v := \lambda_t(tmp) \tag{9}$$
$$\quad \textbf{else} \tag{10}$$
$$\quad\quad addAllChildren \tag{11}$$
$$\quad \textbf{fi} \tag{12}$$
$$\textbf{od} \tag{13}$$

Define the following queue operation:
initialise: initialise a queue that can hold up to N elements to the empty queue vlaue.

$$
\begin{array}{l}
\textbf{proc } initialise(N, \textbf{return} q) \cdot \\
\quad q : [true, q =<>]
\end{array}
$$

enqueue: adds an item to a queue if there's a space available

$$\textbf{proc } enqueue(q, \textbf{value } v, n) \cdot$$

$$n, q : [n < N \wedge q = q_0, q = q_0 \cdot vn = n_0 + 1]$$

dequeue: return the oldest item in the queue and remove it form the queue

$$\begin{array}{l} \textbf{proc } \mathit{dequeue}(q, n) \cdot \\ \quad n, q : [n > 0 \wedge q =< s, qp > \wedge q \neq \quad <>, s = s_0 \wedge q = qp] \end{array}$$

isempty: return whether a queue is empty

$$\begin{array}{l} \textbf{return } \mathit{isempty}(n, \textbf{return} f) \cdot \\ \quad n, f : [n = \mathit{length}(q), (\neg f \wedge n \neq 0) \vee (f \wedge n = 0)] \end{array}$$

we need to adjust the n, every time we add or remove from the queue.