UROP: Treiber Stack Proof

Joe Rackham
Felix Hofstätter

March 26, 2020

TaDA primitives specification:

$$\vdash \{\mathsf{emp}\}\ \texttt{alloc}(n)\ \{\circledast_{i=0}^{n}\ \texttt{ret} + i \mapsto \_\}$$

$$\vdash \forall n \in \mathbb{N}. \langle \mathtt{x} \mapsto n \rangle\ \texttt{CAS}(\mathtt{x}, \mathtt{a}, \mathtt{b})\ \langle \text{if } n = a \text{ then } \mathtt{x} \mapsto \mathtt{b} \wedge \texttt{ret} = 1 \text{ else } \mathtt{x} \mapsto \mathtt{n} \wedge \texttt{ret} = 0 \rangle$$

Treiber Stack module implementation:

```
push(x, v) {
  nh := alloc(2)
  [nh] := v
  b := 0
  while(b = 0) {
      h := [x]
      [nh + 1] := h
      b := CAS(x, h, nh)
  }
}
```

```
pop(x) {
  h := 0
  b := 0
  while (b = 0) {
      h := [x]
      if (h ≠ 0) {
          nh := [h + 1]
          b := CAS(x, h, nh)
      }
  }
  v := [h]
  return v
}
```

TaDA Treiber Stack module specification:

$$\vdash \forall ls. \langle \mathsf{emp} \rangle\ \texttt{makeTreiber}()\ \langle \exists s.\ \mathsf{TS}(s, \texttt{ret}, \epsilon) \rangle$$

$$\vdash \forall ls. \langle \mathsf{TS}(s, \mathtt{x}, ls) \rangle\ \texttt{push}(\mathtt{x}, \mathtt{v})\ \langle \mathsf{TS}(s, \mathtt{x}, \mathtt{v} : ls) \rangle$$

$$\vdash \forall ls. \langle \mathsf{TS}(s, \mathtt{x}, ls) \rangle\ \texttt{pop}(\mathtt{x})\ \langle \exists ls'.\ \mathsf{TS}(s, \mathtt{x}, ls') \wedge ls = \texttt{ret} : ls' \rangle$$

Abstract predicate definition and interpretation:

$$\mathsf{TS}(a, x, ls) \triangleq \exists a_h, g_h.\, \mathbf{Treiber}_a(x, (a_h, g_h)) * \lceil \textsc{Change} \rceil_a * \mathsf{PtrCpy}(a_h, ls)$$

$$\begin{aligned} \mathsf{PtrCpy}(\epsilon, \epsilon) &\triangleq True \\ \mathsf{PtrCpy}((x, l) : a_h, l : ls) &\triangleq \mathsf{PtrCpy}(a_h, ls) \end{aligned}$$

$$\mathcal{I}(\mathbf{Treiber}_a(x, (a_h, g_h)) \triangleq \exists h.\, x \mapsto h * \mathsf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a)$$

$$\begin{aligned} \mathsf{Linked}(p, \epsilon) &\triangleq p = 0 \\ \mathsf{Linked}(p, (x, l) : a_h) &\triangleq \exists p'.\, p = x * x \mapsto l, p' * \lceil \textsc{Item}(x, l, p') \rceil_a * \mathsf{Linked}(p', a_h) \end{aligned}$$

(1) $\forall x, l, p. \lceil \textsc{Item}(x, l, p) \rceil_a = \lceil \textsc{Item}(x, l, p) \rceil_a \bullet \lceil \textsc{Item}(x, l, p) \rceil_a$
(2) $\forall x, l, p, l', p'. \lceil \textsc{Item}(x, l, p) \rceil_a \bullet \lceil \textsc{Item}(x, l', p') \rceil_a \;\Leftrightarrow\; l' = l \wedge p' = p$

$\textsc{Change}$: $\forall a_h, g_h, l, x, p.\, (a_h, g_d) \rightsquigarrow ((x, l) : a_h, g_d)$
$\textsc{Change}$: $\forall l, a_h, g_h, x, p.\, ((x, l) : a_h, g_d) \rightsquigarrow (a_h, (x, l) \uplus g_d)$

$\vdash \forall ls. \langle \mathbf{TS}(a, \mathtt{x}, ls) \rangle$

Abstract

$\langle \exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * \lceil \textsc{Change} \rceil_a * \mathbf{PtrCpy}(a_h, ls) \rangle$

MakeAtomic

$a : (a_h, g_h) \rightsquigarrow ((x, l) : a_h, g_h)$

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge\}$

`nh := alloc(2)`

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \_, \_\}$

`[nh] := v`

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \_\}$

`b := 0`

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (ls, a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \_ \wedge \mathtt{b} = 0\}$

`while(b = 0) {`

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \_ \wedge \mathtt{b} = 0\}$

OpenRegion

$\forall h, a_h, g_h$

$\langle \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p. x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) \rangle$

`h := [x]`

$\langle \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p. x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \mathtt{h} = h \rangle$

$\{\exists a_h, g_h, p. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \_ \wedge \mathtt{b} = 0 * \mathtt{h} = p\}$

// To be stable we can only assert that **h** has a value assigned

`[nh + 1] := h`

$\{\exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \mathtt{h} \wedge \mathtt{b} = 0\}$

// Viewshift to introduce new guard, **nh** isn't in the shared region so no contradictory guard can exist

$\{\exists a_h, g_h, p. \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \mathtt{h} * \lceil \textsc{Item}(\mathtt{nh}, \mathtt{v}, \mathtt{h}) \rceil_a \wedge \mathtt{b} = 0\}$

UpdateRegion

$\forall h, a_h, g_h$

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, ls) * \circledast_{(x,l) \in g_h} (\exists p. x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) \\ * \mathtt{nh} \mapsto \mathtt{v}, \mathtt{h} * \lceil \textsc{Item}(\mathtt{nh}, \mathtt{v}, \mathtt{h}) \rceil_a \end{array} \right\rangle$

`b := CAS(x, h, nh)`

$\left\langle \begin{array}{l} \underline{\text{if}}\ b = 0\ \underline{\text{then}}\ \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p. x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) \\ \underline{\text{else}}\ \mathtt{x} \mapsto \mathtt{nh} * \mathbf{Linked}(\mathtt{nh}, (\mathtt{nh}, \mathtt{v}) : a_h) * \circledast_{(x,(l,p)) \in g_h} (\exists p. x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \mathtt{h} = h \end{array} \right\rangle$

// If the update occurs a new head is pre-pended to the linked list used to store the stack

$\left\{ \begin{array}{l} \exists a_h, a'_h, g_h, g'_h.\ \underline{\text{if}}\ b = 0\ \underline{\text{then}}\ \mathbf{Treiber}_a(\mathtt{x}, (a'_h, g'_h)) * a \Rrightarrow \blacklozenge * \mathtt{nh} \mapsto \mathtt{v}, \_ \\ \underline{\text{else}}\ a \Rrightarrow ((a_h, g_h), ((\mathtt{nh}, \mathtt{v}) : a_h, g_h)) \end{array} \right\}$

`}`

$\{\exists a_h, g_h. a \Rrightarrow ((ls, a_h, g_h), ((\mathtt{nh}, \mathtt{v}) : a_h, g_h))\}$

$\langle \exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (\mathtt{nh}, \mathtt{v}) : a_h, g_h)) * \lceil \textsc{Change} \rceil_a * \mathbf{PtrCpy}(a_h, ls) \rangle$

// By application of the definition of **PtrCpy**

$\langle \exists a_h, g_h. \mathbf{Treiber}_a(\mathtt{x}, (\mathtt{nh}, \mathtt{v}) : a_h, g_h)) * \lceil \textsc{Change} \rceil_a * \mathbf{PtrCpy}((\mathtt{nh}, \mathtt{v}) : a_h, \mathtt{v} : ls) \rangle$

$\langle \mathbf{TS}(a, \mathtt{x}, \mathtt{v} : ls) \rangle$

Figure 1: Proof of Treiber Stack `push`

$\vdash \forall ls. \langle \mathbf{TS}(a, \mathtt{x}, ls) \rangle$

Abstract

$\langle \exists a_h, g_h.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * \lceil \textsc{Change} \rceil_a * \mathbf{PtrCpy}(a_h, ls) \rangle$

MakeAtomic

$a : ((x, l) : a_h, g_h) \rightsquigarrow (a_h, (x, l) \uplus g_h)$

$\{\exists a_h, g_h.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge\}$

`h := 0`

$\{\exists a_h, g_h.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge \wedge \mathtt{h} = 0\}$

`b := 0`

$\{\exists a_h, g_h.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge \wedge \mathtt{h} = 0 \wedge \mathtt{b} = 0\}$

`while(b = 0) {`

$\{\exists a_h, g_h.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \mathtt{b} = 0\}$

OpenRegion

$\forall h, a_h, g_h$

$\langle \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) \rangle$

`h := [x]`

$\left\langle \begin{array}{l} \exists h_l, h_p.\, \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \mathtt{h} = h * \\ \mathtt{h} \neq 0 \Rightarrow \lceil \textsc{Item}(h, h_l, h_p) \rceil_a \end{array} \right\rangle$

// From the definition of P, h is either null or points to a binary cell with associated guard
// Use guard axiom 1 to multiply the item guard

$\left\{ \begin{array}{l} \exists a_h, g_h, h_l, h_p.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \\ (\mathtt{h} = 0 \vee ((\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) * \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a)) * \mathtt{b} = 0 \end{array} \right\}$

// This assertion is stable under the transition system, if `h` points to a list cell it can be popped out
// of the list but will remain allocated in the shared heaplet defined by g_h

`if (h ≠ 0) {`

$\left\{ \begin{array}{l} \exists a_h, g_h, h_l, h_p.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a * \\ (\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) * \mathtt{b} = 0 \end{array} \right\}$

OpenRegion

$\forall h, a_h, g_h$

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \\ \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a * (\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) \end{array} \right\rangle$

`nh := [h + 1]`

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \\ \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a * \mathtt{nh} = h_p * (\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) \end{array} \right\rangle$

// `h` still points a binary cell regardless of whether it's still in the list
// by guard axiom 2 we know the value hasn't changed

$\left\{ \begin{array}{l} \exists a_h, g_h, h_l, h_p.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, g_h)) * a \Rrightarrow \blacklozenge * \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a * \\ \mathtt{nh} = h_p * (\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) * \mathtt{b} = 0 \end{array} \right\}$

UpdateRegion

$\forall h, a_h, g_h$

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \\ \lceil \textsc{Item}(\mathtt{h}, h_l, h_p) \rceil_a * \mathtt{nh} = h_p * (\mathtt{h} \in dom(a_h) \vee \mathtt{h} \in dom(g_h)) \end{array} \right\rangle$

`b := CAS(x, h, nh)`

$\left\langle \begin{array}{l} \underline{\mathbf{if}}\ b = 0\ \underline{\mathbf{then}}\ \mathtt{x} \mapsto h * \mathbf{Linked}(h, a_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) \\ \underline{\mathbf{else}}\ \exists a'_h.\, \mathtt{x} \mapsto \mathtt{nh} * \mathbf{Linked}(\mathtt{nh}, a'_h) * \circledast_{(x,l) \in g_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \\ \mathtt{h} \mapsto h_l, \mathtt{nh} * \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a * a_h = (\mathtt{h}, h_l) : a'_h \end{array} \right\rangle$

// If b = 1 then `h` still pointed to the head of the list, we establish the interpretation
// of the region with the first item moved to the garbage heaplet

$\left\{ \begin{array}{l} \exists a_h, a'_h, g_h, g'_h.\, \underline{\mathbf{if}}\ b = 0\ \underline{\mathbf{then}}\ \mathbf{Treiber}_a(\mathtt{x}, (a'_h, g'_h)) * a \Rrightarrow \blacklozenge \\ \underline{\mathbf{else}}\ \exists h_l.\, a \Rrightarrow (((\mathtt{h}, h_l) : a_h, g_h), (a_h, (\mathtt{h}, h_l) \uplus g_h)) * \\ \mathbf{Treiber}_a(\mathtt{x}, (a'_h, (\mathtt{h}, h_l) \cup g'_h)) * \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a) \end{array} \right\}$

`}`

$\left\{ \begin{array}{l} \exists a_h, a'_h, g_h, g'_h.\, \underline{\mathbf{if}}\ b = 0\ \underline{\mathbf{then}}\ \mathbf{Treiber}_a(\mathtt{x}, (a'_h, g'_h)) * a \Rrightarrow \blacklozenge \\ \underline{\mathbf{else}}\ \exists h_l.\, a \Rrightarrow (((\mathtt{h}, h_l) : a_h, g_h), (a_h, (\mathtt{h}, h_l) \uplus g_h)) * \\ \mathbf{Treiber}_a(\mathtt{x}, (a'_h, (\mathtt{h}, h_l) \cup g'_h)) * \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a) \end{array} \right\}$

`}`

$\left\{ \begin{array}{l} \exists a_h, a'_h, g_h, g'_h, h_l.\, a \Rrightarrow (((\mathtt{h}, h_l) : a_h, g_h), (a_h, (\mathtt{h}, h_l) \uplus g_h)) * \\ \mathbf{Treiber}_a(\mathtt{x}, (ls, a'_h, (\mathtt{h}, h_l) \uplus g'_h)) * \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a) \end{array} \right\}$

OpenRegion

$\forall h, a'_h, g'_h$

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, a'_h) * \circledast_{(x,l) \in g'_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \mathtt{h} \mapsto h_l, \mathtt{nh} * \\ \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a \end{array} \right\rangle$

`v := [h]`

$\left\langle \begin{array}{l} \mathtt{x} \mapsto h * \mathbf{Linked}(h, a'_h) * \circledast_{(x,l) \in g'_h} (\exists p.\, x \mapsto l, p * \lceil \textsc{Item}(x, l, p) \rceil_a) * \mathtt{h} \mapsto h_l, \mathtt{nh} * \lceil \textsc{Item}(\mathtt{h}, h_l, \mathtt{nh}) \rceil_a * \\ \mathtt{v} = h_l \end{array} \right\rangle$

$\{\exists a_h, g_h, h_l.\, a \Rrightarrow (((\mathtt{h}, h_l) : a_h, g_h), (a_h, (\mathtt{h}, h_l) \uplus g_h)) * \mathtt{v} = h_l\}$

`return v`

$\{\exists a_h, g_h.\, a \Rrightarrow (((\mathtt{h}, \mathtt{ret}) : a_h, g_h), (a_h, (\mathtt{h}, \mathtt{ret}) \uplus g_h))\}$

$\langle \exists a_h, g_h, ls'.\, \mathbf{Treiber}_a(\mathtt{x}, (a_h, (\mathtt{h}, \mathtt{ret}) \uplus g_h)) * \lceil \textsc{Change} \rceil_a * ls = \mathtt{ret} : ls * \mathbf{PtrCpy}(a_h, ls') \rangle$

$\langle \exists ls'.\, \mathbf{TS}(a, \mathtt{x}, ls') * ls = \mathtt{ret} : ls' \rangle$

Figure 2: Proof of Treiber Stack `pop`