IBM LABORATORY VIENNA, Austria

A TECHNIQUE FOR SHOWING THAT TWO FUNCTIONS PRESERVE A RELATION BETWEEN THEIR DOMAINS

C.B.JONES

ABSTRACT

This note shows how a functional form of a relation between the domains of two functions can be utilized to yield a proof that the relation is preserved by the two functions. An example is given based on Lucas's block theorem.

LR 25.3.067

10 April 1970

1. METHOD

The method proposed below replaces the problem of showing that two functions preserve a relation between their respective domains with that of showing that two functions are equivalent. Thus, the necessity to combine the two original functions as in the "twin machine" method of /1/ is avoided yet many of the advantages of this method are preserved.

Suppose we have two functions $f:D \Longrightarrow D$ and $f':D' \Longrightarrow D'$ and we wish to show that they preserve the relation $\mathcal{R}:D \times D' \Longrightarrow \{T,F\}$. If we can find a particular function $m:D \Longrightarrow D'$ such that:

1) $d' = m(d) \equiv \mathcal{R}(d,d')$
2) $m \circ f \equiv f' \circ m$

Then the required relation is preserved by f and f' (N.B. m need not be one to one).

Justification:

first note $m \circ f:D \Longrightarrow D'$
and $f' \circ m:D \Longrightarrow D'$

Suppose $d \in D$ and $d' \in D'$ such that

3)	$\mathcal{R}(d,d')$	
4)	$d' = m(d)$	1,3
5)	$m \circ f(d) = f' \circ m(d)$	2
6)	$m \circ f(d) = f'(d')$	4,5
7)	$\mathcal{R}(f(d),f'(d'))$	6,1

Thus f and f' preserve the relation $\mathcal{R}$.

The functions in which we are interested (e.g. those in section 2) will be recursive. We can utilize the following result, which depends on their structure, to simplify our proofs.

Suppose:

$$f = p_1 \longrightarrow e_1 f \circ f \circ b_1 f$$
$$\vdots$$
$$p_n \longrightarrow e_n f \circ f \circ b_n f$$

$$f' = p'_1 \longrightarrow e_1 f' \circ f' \circ b_1 f'$$
$$\vdots$$
$$p'_n \longrightarrow e_n f' \circ f' \circ b_n f'$$

then $m \circ f = f' \circ m$

if:

$$p_i = p'_i \circ m$$
$$m \circ b_i f = b_i f' \circ m$$
$$m \circ e_i f = e_i f' \circ m$$

Justification follows by recursion induction.

2. DEFINITIONS

The two techniques of maintaining environments shown below were discussed in /1/. The definitions have been written as pure functions since this facilitates the subsequent proof. Procedure denotations, which cannot be changed by assignment but are used in the proof, are separated from the value part, whose members are changeable by assignment and are not discussed in the proof. The function un creates unique names for an identifier and a state.

2.1 State of the Update Machine

Any state of the machine satisfies the predicate is-u-state.

U1 is-u-state = (<s-bl:is-u-bl>,
<s-val-pt:{<n:is-value> || is-un(n)}>)

U2 is-u-bl = (<s-tx:is-st-l>,
<s-e:is-e>,
<s-p-dn:{<n:is-u-proc-den> ||is-un(n)}>,
<s-d:is-u-bl v is-Ω>)

U3 is-u-proc-den = (<s-body:is-st>,
<s-parm-pt:is-id-list>,
<s-e:is-e>)

U4 is-e = {<id:is-n> || is-id(id)}

An interpretation of some text t will be caused by:

U0 int-st-u(μ_0(<s-tx∘s-bl:[t]>))

We use the abbreviations:
UBL = s-bl(ξ)
UTX = s-tx∘UBL
UE = s-e∘UBL
UPDN = s-pdn∘UBL
UD = s-d∘UBL

2.2 State Transition of the Update Machine

U5 int-st-u(ξ) =
is-block(h∘UTX) ⟶ int-block-u(ξ,h∘UTX)
is-call(h∘UTX) ⟶ int-call-u(ξ,(s-nm∘h∘UTX)(UE)(UPDN), $\overset{\text{length (s-arg-l∘h∘UTX)}}{\underset{i=1}{\text{LIST}}}$ elem(i,s-arg-l∘h∘UTX)(UE))
T ⟶ other-u

U6 int-block-u(ξ,t) = unstack-u(int-st-l-u(inst-bl-u(ξ,t,dec-upd(s-dp(t)))))

U7 inst-bl-u(ξ,t,e) = (ξ;<s-bl:μ_o(<s-tx:s-body(t)>,<s-e:μ(UE;e)>,
<s-pdn:{(UPDN; <un(id):μ(id∘s-dp∘t;<s-e:UE>)> |
is-proc-dec(id∘s-dp(t))})>,<s-d:UBL>)>)

U8 int-st-l-u(ξ) = is-<>(UTX) ⟶ ξ
T ⟶ int-st-l-u(next-u(int-st-u(ξ)))

U9 next-u(ξ) = (ξ;<s-tx∘s-bl:t∘UTX>)

U10 int-call-u(ξ,pd,arg-l) =
unstack-u(int-st-u(inst-proc-u(ξ,s-body(pd),s-e(pd),arg-upd(s-parm-pt(pd),arg-l)))

U11 inst-proc-u(ξ,b,e,me) = μ(ξ;<s-bl:μ_o(<s-tx:b>,<s-e:μ(e;me)>,<s-pdn:UPDN>,<s-d:UBL>)>)

U12 unstack-u(ξ) = μ(ξ;<s-bl:UD>)

U13 dec-upd(dp) = μ_o({<id:un(id)> | id∘dp ≠ Ω})

U14 arg-upd(parm-l,arg-l) = μ_o({<elem(i,parm-l):elem(i,arg-l)> | 1 ≤ i ≤ length(arg-l)})

2.3 State of the Search Machine

Any state of the machine satisfies the predicate is-s-state.

S1 is-s-state = (<s-bl:is-s-bl>,
<s-val-pt:{<n:is-value> || is-un(n)}>)

S2 is-s-bl = (<s-tx:is-st-l>,
<s-e-o:is-e>,
<s-ban:is-un>,
<s-epa:is-un>,
<s-p-dn:{<n:is-s-proc-den> || is-un(n)}>,
<s-d:is-s-bl ∨ is-Ω>)

S3 is-s-proc-den = (<s-body:is-st>,
<s-parm-pt:is-id-list>,
<s-ban:is-un>)

is-e: see U4

An interpretation of some text t will be caused by

S0 int-st-s(μ_0(<s-tx∘s-bl:[t]>))

The following abbreviations are used:
SBL = s-bl(ξ)
STX = s-tx∘SBL
SEO = s-e-o∘SBL
SBAN = s-ban∘SBL
SEPA = s-epa∘SBL
SPDN = s-p-dn∘SBL
SD = s-d∘SBL

2.4 State Transition of the Search Machine

S4 int-st-s(ξ) =
is-block(h∘STX) ⟶ int-block-s(ξ,h∘STX)
is-call(h∘STX) ⟶ int-call-s(ξ,find-n(s-nm∘h∘STX,SBL)∘SPDN, $\overset{length(s\text{-}arg\text{-}l∘h∘STX)}{\underset{i=1}{LIST}}$ find-n(elem(i,s-arg-l∘h∘STX),SBL))
T ⟶ other-s

S5 int-block-s(ξ,t) = unstack-s(int-st-l-s(inst-bl-s(ξ,t,dec-upd(s-dp(t)),SBAN)))

S6 inst-bl-s(ξ,t,e,n,) =
μ(ξ;<s-bl:μ_o(<s-tx:s-body(t)>,<s-e-o:e>,<s-ban:un(#)>,<s-epa:n>,
<s-pdn:μ(SPDN;{<un(id): (id∘s-dp(t);<s-ban:un(#)>)> | is-proc-dec(id∘s-dp(t))},
<s-d:SBL>)>)

S7 int-st-l-s(ξ) = is-<>(STX) ⟶ ξ
T ⟶ int-st-l-s(next-s(int-st-s(ξ)))

S8 next-s(ξ) = μ(ξ;<s-tx∘s-bl:t∘TX>)

S9 int-call-s(ξ,pd,arg-l) =
unstack-s(int-st-s(inst-proc-s(ξ,s-body(pd),arg-upd(s-parm-pt(pd),arg-l),s-ban(pd))))

S10 inst-proc-s(ξ,b,e,n) =
μ(ξ;<s-bl:μ_o(<s-tx:b>,<s-e-o:e>,<s-ban:un(#)>,<s-epa:n>,<s-pdn:SPDN>,<s-d:SBL>)>)

S11 unstack-s(ξ) = $\mu(\xi;<s\text{-}bl:SD>)$

dec-upd:see U13
arg-upd:see U14

S12 find-n(id,bl) = s-ban∘bl = $\Omega \longrightarrow \Omega$
id∘s-e-o∘bl $\neq \Omega \longrightarrow$ id∘s-e-o∘bl
T $\longrightarrow$ find-n(id,find-d(s-epa∘bl,bl))

S13 find-d(n,bl) = s-ban∘bl = $\Omega \longrightarrow \Omega$
s-ban∘bl = n $\longrightarrow$ bl
T $\longrightarrow$ find-d(n,s-d∘bl)

3. PROOF OF EQUIVALENCE

We wish to show that the two machines of section 2 are equivalent in the sense that they are equivalent functions from texts to val-pts. It is clearly sufficient that both interpreters use the same selectors (unique names) for any reference to the value part. We are told that all references in the update machine are made by $id \circ s\text{-}e \circ s\text{-}bl(\xi_u)$ whereas $find\text{-}n(id, s\text{-}bl(\xi_s))$ is used in the search machine. It is therefore sufficient to show these two functions yield the same result.

The following mapping, from states satisfying is-s-state to states satisfying is-u-state, is such that the required property is satisfied (see M3):

M1 $m(\xi) = \mu(\xi; \langle s\text{-}bl: map\text{-}bl(s\text{-}bl(\xi))\rangle)$

M2
$$map\text{-}bl(bl) = is\text{-}\Omega(bl) \longrightarrow \Omega$$
$$T \longrightarrow \mu_o(\langle s\text{-}tx: s\text{-}tx(bl)\rangle, \langle s\text{-}e: mk\text{-}env(bl)\rangle, \langle s\text{-}pdn: map\text{-}pdn(s\text{-}pdn(bl), bl)\rangle, \langle s\text{-}d: map\text{-}bl(s\text{-}d(bl))\rangle)$$

M3 $mk\text{-}env(bl) = \mu_o(\{\langle id: find\text{-}n(id,bl)\rangle \mid find\text{-}n(id,bl) \neq \Omega\})$

M4
$$map\text{-}pdn(pdn,bl) = \mu_o(\{\langle n: \mu_o(\langle s\text{-}body: s\text{-}body \circ n \circ pdn\rangle, \langle s\text{-}parm\text{-}pt: s\text{-}parm\text{-}pt \circ n \circ pdn\rangle, \langle s\text{-}e: mk\text{-}env(find\text{-}d(s\text{-}ban \circ n \circ pdn, bl))\rangle)\rangle \mid is\text{-}proc\text{-}den\text{-}s(n \circ pdn)\})$$

We first introduce two lemmas:

Lemma 1: mk-env(bl) = μ(mk-env(find-d(s-epa(bl),bl));s-e-o(bl))

Proof: from M3, S12 and M3 respectively, both sides of the equality are equal to

$\mu(\{$ <id:find-n(id,find-d(s-epa(bl),bl))> | find-n(id,find-d(s-epa(bl),bl)) $\neq \Omega \}$;s-e-o(bl))

Lemma 2: n $\neq$ s-ban(bl) $\supset$ find-d(n,bl) = find-d(n,s-d(bl))

Proof: immediate from S13

We now show that the required relation holds by proving m$\circ$int-st-s = int-st-n$\circ$m

Proof: noting that int-st-u and int-st-s are both of the form:

f	= $p_1 \longrightarrow f_1$ $p_2 \longrightarrow f_2$	S4,U5
f_1	= $ef_1 \circ f \circ bf_1$	S5,U6
f_2	= $ef_2 \circ f \circ bf_2$	S9,U10

we show a) $p_1 s = p_1 u \circ m$
$p_2 s = p_2 u \circ m$

b) $m \circ bf_1 s = bf_1 u \circ m$
$m \circ ef_1 s = ef_1 u \circ m$

c) $m \circ bf_2 s = bf_2 u \circ m$
$m \circ ef_2 s = ef_2 u \circ m$

a)

1 is-block(h∘s-tx∘s-bl∘map(ξ)) = is-block(h∘STX)

2 is-call(h∘s-tx∘s-bl∘map(ξ)) = is-call(h∘STX)

1 and 2 fulfil a).

b) Let is-s-state(ξ) hold

3 int-bl-s(ξ,h∘STX,dec-upd(s-dp∘h∘STX),SBAN) =

μ(ξ;<s-bl:μ_o(<s-tx:s-body∘h∘STX>, S.6

<s-e-o:dec-upd(s-dp∘h∘STX)>,

<s-ban:un(#)>,

<s-epa:SBAN>,

<s-pdn:μ(SPDN;{<un(id):μ(id∘s-dp∘h∘STX;<s-ban:un(#)>)> |

is-proc-dec(id∘s-dp∘h∘STX)})>,

<s-d:SBL>)>)

Now, using ξ' as a temporary abbreviation for the above state:

4 μ(map-pdn(SPDN),s-bl(ξ'));map-pdn({<un(id):μ(id∘s-dp∘h∘STX;<s-ban:un(#)>)> | M4,L2,M4,L1

is-proc-dec(id∘s-dp∘h∘STX)},s-bl(ξ'))

= μ(map-pdn(SPDN,s-bl(ξ));{<un(id):μ(id∘s-dp∘h∘STX;<s-e:μ(mk-env(SBL);dec-upd(s-dp∘h∘STX))>)> |

is-proc-dec(id∘s-dp∘h∘STX)})

5 map∘int-bl-s = M1,3
μ(ξ;<s-bl:μ_o(<s-tx:s-body∘h∘STX>, M2
<s-e:μ (mk-env(SBL);dec-upd(s-dp∘h∘STX))>, M2,M3,L1
<s-pdn:μ(map-pdn(SPDN,s-bl(ξ));{<un(id):μ(id∘s-dp∘h∘STX; M2,4
<s-e:μ(mk-env(SBL);dec-upd(s-dp∘h∘STX))>)> | is-proc-dec(id∘s-dp∘h∘STX)})>,
<s-d:map-bl(SBL)>)>) M2

6 int-bl-u(map(ξ),h∘s-tx∘s-bl∘map(ξ),dec-upd(s-dp∘h∘s-tx∘s-bl∘map(ξ)) = U7,M1
μ(ξ;<s-bl:μ_o(<s-tx:s-body∘h∘STX>, M2
<s-e:μ(mk-env(SBL);dec-upd(s-dp∘h∘STX))>, M2
<s-pdn:μ(map-pdn(SPDN,s-bl(ξ));{<un(id):μ(id∘s-dp∘h∘STX; M2,M4
<s-e:μ(mk-env(SBL);dec-upd(s-dp∘h∘STX))>)> | is-proc-dec(id∘s-dp∘h∘STX)})>,
<s-d:map-bl(SBL)>)>) M2

From U6 and S5, 5 and 6 conclude the first part of b).

7 unstack-u(map(ξ)) = μ(ξ;<s-bl:map-bl(s-d(ξ))>) U12,M2

8 map(unstack-s(ξ)) = μ(ξ;<s-bl:map-bl(s-d(ξ))>) S11,M2

From U6 and S5, 7 and 8 conclude the remaining part of b).

c) Let is-s-state(ξ) hold

Let pd-s = find-n(s-nm(h(STX)),SBL)∘SPDN

Let arg-l-s = $\overset{\text{length (s-arg-l (h(STX)))}}{\underset{i=1}{\text{LIST}}}$ find-n(elem(i,s-arg-l(h(STX))),SBL)

9 inst-proc-s(ξ,s-body∘pd-s,arg-upd(s-parm-pt∘pd-s,arg-l-s),s-ban∘pd-s) = S10
μ(ξ;<s-bl:μ_o(<s-tx:s-body∘pd-s>,
<s-e-o:arg-upd(s-parm-pt∘pd-s,arg-l-s)>,
<s-ban:un(#)>,
<s-epa:s-ban∘pd-s>,
<s-pdn:SPDN>,
<s-d:SBL>)>)

10 map∘int-proc-s = M1,9
μ(ξ;<s-bl:μ_o(<s-tx:s-body∘pd-s>, M2
<s-e:μ(mk-env(find-d(s-ban∘pd-s,SBL));arg-upd(s-parm-pt∘pd-s,arg-l-s))>, M2,L1
<s-pdn:map-pdn(SPDN,SBL)>, M2,L2
<s-d:map-bl(SBL)>)> M2

11 $\underset{i=1}{\overset{length(s\text{-}arg\text{-}l\circ h\circ s\text{-}tx\circ s\text{-}bl\circ map(\xi))}{LIST}}$ elem(i,s-arg-l(h∘s-tx∘s-bl∘map(ξ))(s-e∘s-bl∘map(ξ)) = arg-l-s M1,M2,M3

12 s-parm-pt∘s-nm(h∘s-tx∘s-bl∘map(ξ))(s-e∘s-bl∘map(ξ))(s-pdn∘s-bl∘map(ξ)) =
s-parm-pt∘pd-s M1,M2,M3,M4

13 s-body∘s-nm(h∘s-tx∘s-bl∘map(ξ))(s-e∘s-bl∘map(ξ))(s-pdn∘s-bl∘map(ξ)) =
s-body∘pd-s M1,M2,M3,M4

14 s-e∘s-nm(h∘s-tx∘s-bl∘map(ξ))(s-e∘s-bl∘map(ξ))(s-pdn∘s-bl∘map(ξ)) =
mk-env(find-d(s-ban∘pd-s,SBL)) M1,M2,M3,M4

15 inst-proc-u(map(ξ);s-body∘pd-s,mk-env(find-d(s-ban∘pd-s,SBL)),
arg-upd(s-parm-pt∘pd-s,arg-1-s)) = U11

μ(ξ;<s-bl:μ_o(<s-tx:s-body∘pd-s>,
<s-e:μ(mk-env(find-d(s-ban∘pd-s,SBL));arg-upd(s-parm-pt∘pd-s,arg-1-s))>,
<s-pdn:map-pdn(SPDN,SBL)>,
<s-d:map-bl(SBL)>)>)

From S9, U10 (noting 11,12,13,14) 10,15 conclude the first part of c.
The second part of c) is identical with the second part of b).
Thus our proof is completed.

REFERENCE

/1/ LUCAS,P.: Two Constructive Realizations of the Block Concept and their Equivalence.-
IBM Lab Vienna, Techn.Report TR 25.085, 28 June 1968.