Universität Dortmund
Lehrstuhl Verteilte Systeme

Dr. Michael Zapf

Michael.Zapf@cs.uni-dortmund.de
Michael.Zapf@uni-kassel.de

vkiv@ls4.cs.uni-dortmund.de

Übung zur Vorlesung „Verteilt-kooperative Informationsverarbeitung“ - SS 2007

Lösungen zu Blatt 5

Aufgabe 5.1 – 20 Punkte

Reduzieren Sie so weit wie möglich folgende Ausdrücke gemäß dem Ambient-Kalkül. Bitte geben Sie alle Zwischenschritte an. (Schlagen Sie ggf. die entsprechenden Definitionen im Skript nach.)

a) in m.P | m[ ]
   keine Änderung
b) k[in m.P] | m[Q]
   → m[k[P] | Q]
c) n[m[out n.R] | R]
   → m[R] | n[R]
d) open r.R | k[r[out k.P]]
   → open r.R | r[P] | k[] → R | P | k[ ]
e) (νn)(m[n[out m.P]] | open n) | n[Q]
   → (νn)(n[P] | m[] | open n) | n[Q]
   → (νn)(P | m[] | 0) | n[Q]
   → (νn)P | m[] | n[Q]
f) (νk)!(open k.P) | k[Q]
   keine Änderung (k ist auf den linken Term beschränkt)
g) n[in m.in k.open m] | m[] | k[Q]
   → m[n[in k.open m]] | k[Q]
h) acquire k.release m.R | release k.acquire m.N
   = open k.(m[] | R) | k[] | open m.N
   → m[] | R | open m.N
   → R | N
i) (νk)k[P] | open k.R | k[Q]
   → (νk)k[P] | R | Q
j) (νw)w[k[out w.in k'.in w] | open k'.open k''.P] | k'[open k.k''[Q]]
   → (νw)(k[in k'.in w] | w[open k'.open k''.P]) | k'[open k.k''[Q]]
   ≡ (νw)(k[in k'.in w] | w[open k'.open k''.P] | k'[open k.k''[Q]])   w∉ fn(Q)
   → (νw)(w[open k'.open k''.P] | k'[k[in w] | open k.k''[Q]])
   → (νw)(w[open k'.open k''.P] | k'[in w | k''[Q]])
   → (νw)(w[k'[k''[Q]] | open k'.open k''.P])
   → (νw)(w[k''[Q] | open k''.P])
   → (νw)(w[Q | P])
k) entrap m | m[P] (unter der Annahme, dass „mv in“ als Primitive gegeben ist)

= (νk)(k[ ] | mv in m.in k.**0**) | m[P]

→ (νk)(k[ ] | mv in m.in k.**0** | m[P])   k∉ fn(P)

→ (νk)(k[ ] | m[in k.**0** | P])

→ (νk)(k[m[P]])

l) $n^{\uparrow}$[mv out n.P]

= n[mv out n.P] | allow exit

= n[(νk)k[out n.exit[out k.open k.P]]] | !open exit

→ (νk)k[exit[out k.open k.P]] | n[] | !open exit

→ (νk)(exit[open k.P] | k[]) | n[] | !open exit

= (νk)(exit[open k.P] | k[] | n[] | !open exit)

→ (νk)(exit[open k.P] | k[] | n[] | open exit | !open exit)

→ (νk)(open k.P | k[] | n[] | !open exit)

→ (νk)(P | n[] | !open exit)

= (νk)(P | $n^{\uparrow}$[])

= P | $n^{\uparrow}$[]   für k∉ fn(P)

m) kanal?.P | kanal!.Q | $kanal^{\uparrow\downarrow}$[]

Kurzform: Siehe Definition von mv in, mv out und $kanal^{\uparrow\downarrow}$ *im Skript. Diese Lösung ist akzeptabel.*

kanal?.P | kanal!.Q | $kanal^{\uparrow\downarrow}$[]

= mv in kanal.acquire rd.release wr.mv out kanal.P | mv in kanal.release rd.acquire wr.mv out kanal.Q | $kanal^{\uparrow\downarrow}$[]

→ $kanal^{\uparrow\downarrow}$[acquire rd.release wr.mv out kanal.P | release rd.acquire wr.mv out kanal.Q]

= $kanal^{\uparrow\downarrow}$[open rd.release wr.mv out kanal.P | release rd.open wr.mv out kanal.Q]

= $kanal^{\uparrow\downarrow}$[open rd.(wr[] | mv out kanal.P) | (rd[] | open wr.mv out kanal.Q)]

→ $kanal^{\uparrow\downarrow}$[wr[] | mv out kanal.P | open wr.mv out kanal.Q]

→ $kanal^{\uparrow\downarrow}$[mv out kanal.P | mv out kanal.Q]

→ P | Q | $kanal^{\uparrow\downarrow}$[ ]

Langform: In dieser Lösung wurde bis auf die grundlegenden Ambientprimitive reduziert.

= (νk) k[in kanal.enter[out k.open k.open rd.wr[] | (νm)m[out kanal.exit[out m.open m.P]]]] | (νn) n[in kanal.enter[out n.open n.rd[] | open wr.(νp)p[out kanal.exit[out p.open p.Q]]]] | kanal[!open enter] | !open exit

(Wir gehen davon aus, dass k,n,m,p ∉ fn(P)∪fn(Q); so können wir die Restriktionen nach vorne ziehen. Diese Namen dienen ohnehin nur zur Implementierung von mv in, mv out.)

= (ν k m n p) (k[in kanal.enter[out k.open k.open rd.wr[] | m[out kanal.exit[out m.open m.P]]]]
| n[in kanal.enter[out n.open n.rd[] | open wr.p[out kanal.exit[out p.open p.Q]]]]
| kanal[!open enter] | !open exit)

→ (ν k m n p) (kanal[k[enter[out k.open k.open rd.wr[] | m[out kanal.exit[out m.open m.P]]]]
| n[enter[out n.open n.rd[] | open wr.p[out kanal.exit[out p.open p.Q]]]] | !open enter
] | !open exit)

→ (ν k m n p)(kanal[enter[open k.open rd.wr[] | m[out kanal.exit[out m.open m.P]]] | k[]
| enter[open n.rd[] | open wr.p[out kanal.exit[out p.open p.Q]]] | n[] | !open enter] | !open exit)

→ (ν k m n p) (kanal[open k.open rd.wr[] | m[out kanal.exit[out m.open m.P]] | k[] | open n.rd[] | open wr.p[out kanal.exit[out p.open p.Q]] | n[] | !open enter] | !open exit)

→ (ν m p) (kanal[open rd.wr[] | m[out kanal.exit[out m.open m.P]] | rd[] | open wr.p[out kanal.exit[out p.open p.Q]] | !open enter] | !open exit)

→ (ν m p) (kanal[wr[] | m[out kanal.exit[out m.open m.P]] | open wr.p[out kanal.exit[out p.open p.Q]] | !open enter] | !open exit)

→ (ν m p) (kanal[m[out kanal.exit[out m.open m.P]] | p[out kanal.exit[out p.open p.Q]] | !open enter] | !open exit)

→ (ν m p)(m[exit[out m.open m.P]] | p[exit[out p.open p.Q]] | kanal[!open enter] | !open exit)

→ (ν m p)(exit[open m.P] | m[] | exit[open p.Q] | p[] | kanal[!open enter] | !open exit)

→ (ν m p)(open m.P | m[] | open p.Q | p[] | kanal[!open enter] | !open exit)

→ (ν m p)(P | Q | kanal[!open enter] | !open exit)

≡ P | Q | kanal[!open enter] | !open exit

= P | Q | $kanal^{\uparrow\downarrow}$[]

n) see n.P | n[Q]

= (ν r s)(r[in n.out n.r be s.P] | open s) | n[Q]

= (ν r s)(r[in n.out n.s[out r.open r.P] | in s] | open s) | n[Q]

= (ν r s)(r[in n.out n.s[out r.open r.P] | in s] | open s | n[Q]) (wenn r,s $\notin fn(P) \cup fn(Q)$)

→ (ν r s)(open s | n[r[out n.s[out r.open r.P] | in s] |Q])

→ (ν r s)(open s | r[s[out r.open r.P] | in s] | n[Q])

→ (ν r s)(open s | s[open r.P] | r[in s] | n[Q]) (*)

→ (ν r s)(open s | s[r[] | open r.P] | n[Q])

→ (ν r s)(r[] | open r.P | n[Q])

→ (ν r s)(P | n[Q]) = P | n[Q]

(*) Alternativ:

(ν r s)(open s | s[open r.P] | r[in s] | n[Q])

→ (ν r s)(open r.P | r[in s] | n[Q])

→ (ν r s)(P | in s.**0** | n[Q])

= (νs)in s.**0** | P | n[Q]

≃ P | n[Q] (der erste Prozess ist inaktiv (das s ist gebunden), aber eben nicht 0)

o) m[(x).in x.P | n[(x).in x.Q] | ⟨n⟩] | n[R]

→ m[(in x.P){x←n} | n[(x).in x.Q]] | n[R]

→ m[in n.(P{x←n}) | n[(x).in x.Q]] | n[R]

→ m[in n.P{x←n} | n[(x).in x.Q]] | n[R]

→ n[m[P{x←n} | n[(x).in x.Q]] | R]

p) ⟨n⟩ | (x).m[in x.P] | n[Q] | r[S]

→ (m[in x.P]){x←n} | n[Q] | r[S]

→ m[in n.P{x←n}] | n[Q] | r[S]

→ n[m[P{x←n}] | Q] | r[S]

q) ⟨open n⟩ | (x).p[x.P | n[Q | open p]]

→ p[open n.P{x←open n} | n[Q{x←open n} | open p]]

→ p[P{x←open n} | Q{x←open n} | open p]

r) cell s w | get s (y).open y | w[ ]

Hinweis: mv in/out und $s^{\uparrow\downarrow}$ siehe Skript

→ $s^{\uparrow\downarrow}$[⟨w⟩] | mv in s.(y).(⟨y⟩ | mv out s.open y) | w[ ]

→ $s^{\uparrow\downarrow}$[⟨w⟩ | (y).(⟨y⟩ | mv out s.open y)] | | w[ ]

→ $s^{\uparrow\downarrow}$[⟨w⟩ | mv out s.open w] | | w[ ]

→ $s^{\uparrow\downarrow}$[⟨w⟩] | open w | w[ ]

→ $s^{\uparrow\downarrow}$[⟨w⟩]

= cell s w

s) packet pkt | route pkt with P to (in m.out m.in n) | m[Q] | n[R]

= pkt[!(x).x | !open route] | route[in pkt.⟨in m.out m.in n⟩ | P] | m[Q] | n[R]

→ pkt[!(x).x | !open route | route[⟨in m.out m.in n⟩ | P]] | m[Q] | n[R]

→ pkt[!(x).x | !open route | ⟨in m.out m.in n⟩ | P] | m[Q] | n[R]

→ pkt[!(x).x | (x).x | !open route | ⟨in m.out m.in n⟩ | P] | m[Q] | n[R]

→ pkt[!(x).x | in m.out m.in n | !open route | P] | m[Q] | n[R]

→ m[pkt[!(x).x | out m.in n | !open route | P] | Q] | n[R]

→ m[Q] | pkt[!(x).x | in n | !open route | P] | n[R]

→ m[Q] | n[pkt[!(x).x | !open route | P] | R]

t) (M,N).p[@M arg⟨a⟩ res(x) N.P] | q[!(open io.(b).res[⟨in b⟩])] | ⟨out p.in q, out q.in p⟩

Wir gehen davon aus, dass M, N ∉ fv(P), daher reicht es, P statt P{M←out p.in q, N←out q.in p} zu schreiben

→ p[@(out p.in q) arg⟨a⟩ res(x) (out q.in p).P] | q[!(open io.(b).res[⟨in b⟩])]

= p[(νn)(io[out p.in q.(⟨a⟩| open res.(x).n[out q.in p.⟨x⟩])] | open n) | (x).P] | q[!(open io.(b).res[⟨in b⟩])]

→ (νn)(io[in q.(⟨a⟩| open res.(x).n[out q.in p.⟨x⟩])] | p[open n | (x).P]) | q[!(open io.(b).res[⟨in b⟩])]

(Wir gehen wieder von n ∉ fn(P) aus, also können wir die Restriktion über alle Terme ziehen.)

= (νn)(io[in q.(⟨a⟩| open res.(x).n[out q.in p.⟨x⟩])] | p[open n | (x).P] | q[!(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[io[⟨a⟩| open res.(x).n[out q.in p.⟨x⟩]] | open io.(b).res[⟨in b⟩] | !(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[⟨a⟩| open res.(x).n[out q.in p.⟨x⟩] | (b).res[⟨in b⟩] | !(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[open res.(x).n[out q.in p.⟨x⟩] | res[⟨in a⟩] | !(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[(x).n[out q.in p.⟨x⟩] | ⟨in a⟩ | !(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[n[out q.in p.⟨in a⟩] | !(open io.(b).res[⟨in b⟩])])

→ (νn)(p[open n | (x).P] | q[!(open io.(b).res[⟨in b⟩])] | n[in p.⟨in a⟩])

→ (νn)(p[n[⟨in a⟩] | open n | (x).P] | q[!(open io.(b).res[⟨in b⟩])])

= (νn)(p[n[⟨in a⟩] | open n | (x).P]) | q[!(open io.(b).res[⟨in b⟩])]

→ p[⟨in a⟩ | (x).P] | q[!(open io.(b).res[⟨in b⟩])]

→ p[P{x←in a}] | q[!(open io.(b).res[⟨in b⟩])]

(Bei diesem Vorgang holt sich das p-Ambient die zum Argument „a" gehörende Berechtigung „in a" bei einem externen q-Ambient ab. Man beachte, dass noch immer gilt, dass man aus der Kenntnis von „in a" nicht auf „a" zurückschließen kann; die Kenntnis von „a" ist aber hinreichend für die Gewinnung eines „in a".)

Aufgabe 5.2 – 10 Punkte

a) Wie unterscheiden sich freie Namen von freien Variablen? Wie werden diese gebunden? (3 Punkte)

Freie Namen werden durch das Auftauchen von Ambients oder zugehörigen Berechtigungen erzeugt und sind stets Benennungen der entsprechenden Ambients. Sie werden durch den ν-Operator gebunden und sind dann nur noch innerhalb des Prozesses gültig, auf den sich der Operator bezieht. Taucht derselbe Name außerhalb der Restriktion auf, dann beeinflussen sich der äußere und der innere Name nicht.

Freie Variablen sind Bezeichner, die innerhalb von Prozessen oder Pfaden auftreten. Sie können durch Belegung (d.h. Bindung an einen Wert) ersetzt werden. Diese Bindung erfolgt über die Eingabeoperation (x), welche von einer gleichrangigen Ausgabeoperation mit einem Wert versorgt wird. Durch die Bindung über (x) wird der Gültigkeitsbereich der Variablen auf den Term eingeschränkt, auf den sich die Eingabeoperation bezieht.

Gebundene Variablen erhalten ihre Bedeutung durch die Reduktionsregel (x).P | ⟨M⟩ → P{x←M}, was eben nicht für Namen gilt (siehe aber die Folie „Polyadische, typisierte Kommunikation": Cardelli hat selbst offenbar die strikte Trennung später aufgehoben).

Da auch Namen als Berechtigung in einem Pfad auftreten können, kann man innerhalb einer Formel freie Namen nicht von freien Variablen unterscheiden. Sie müssen explizit entweder als Elemente von *fn* oder *fv* deklariert werden.

Die Festlegung, dass Namen mit m, n, ... bezeichnet werden, Variablen hingegen mit x, y, ... ist reine Konvention und ergibt sich nicht aus den Ambientregeln.

Beispiel: Ist der Bezeichner k in k[P] ein freier Name, so ist fn(k[P]) = {k} ∪ fn(P), und k kann mittels νk gebunden werden. Ist aber k in k[P] eine freie Variable, so ist fn(k[P]) = fn(P), fv(k[P]) = {k} ∪ fv(P). Dann kann k durch (k) gebunden werden: (k).k[P] und würde in Gegenwart eines Prozesses ⟨n⟩ reduziert werden zu n[P{k←n}].

b) Bestimmen Sie die Namen und Variablen der folgenden Terme und geben Sie an, ob diese frei oder gebunden sind. (3 Punkte)

- m[n[]]

  Freie Namen: m,n (können auch freie Variablen sein); keine gebundenen Namen oder Variablen.

- n[(νw)w[m[]]]

  Freie Namen: m,n; gebundener Name: w (innerhalb n[]), freie Variablen m, n, w; keine gebundenen Variablen.

- n[in m.in k.open m] | m[] | k[]

  Freie Namen/Variablen: k,m,n; keine gebundenen Namen oder Variablen.

- ⟨n⟩ | (x).m[in x.P] | k.n[y.Q] | r[S]

  Freie Namen/Variablen: k, m, n, y, r, fn(P), fn(Q), fn(S), fv(P)\{x}, fv(Q), fv(S); gebundene Namen: keine; gebundene Variablen: x

c) Prüfen Sie, ob der folgende Ausdruck gemäß dem Ambient-Kalkül korrekt ist. Anderenfalls geben Sie bitte an, wo der Fehler steckt; geben Sie eine mögliche Korrektur an. (1 Punkt)

open n.n[(νn)P] | !n[P]

Der Ausdruck ist korrekt: Die Verwendung einer Restriktion bedeutet nicht, dass man den Namen in „höheren" Termen nie mehr verwenden kann. Der Name n außerhalb der Restriktion kann mit dem Namen n innerhalb von P (in der Restriktion) nicht interagieren. Insgesamt hat der Term die Form

A | B

mit A=M.C und B=!D

mit M=open n und C=n[E] und D=n[P]

mit E=(νn)P

d) Reduzieren Sie den in c) gegebenen bzw. von Ihnen korrigierten Ausdruck und geben Sie die Menge der freien Namen für jeden Bestandteil in der Hierarchie an. (3 Punkte)

open n.n[(νn)P] | !n[P] ≡ open n.n[(νn)P] | n[P] | !n[P] → n[(νn)P] | P | !n[P]

R = n[(νn)P] | P | !n[P] :

Stufe 1: R

fn(R) = {n} ∪ fn(P)

Stufe 2: n[...], P, !n[...]

fn(n[(νn)P]) = {n} ∪ (fn(P)\{n}) = {n} ∪ fn(P)

fn(P) = nicht genauer spezifizierbar,

fn(!n[P]) = {n} ∪ fn(P)

Stufe 3: (νn)P, P

fn((νn)P) = fn(P)\{n},

fn(P) = nicht genauer spezifizierbar

Aufgabe 5.3 – 10 Punkte

a) Zeigen Sie, dass Sie mit dem Ambient-Kalkül jede beliebige Logikschaltung repräsentieren können. Nutzen Sie das Standardkalkül ohne Kommunikation. (5 Punkte)

Wir erstellen uns ein NAND. Dieses ist definiert als eine Funktion, die „true" wird, wenn mindestens ein Argument „false" ist. Das Resultat liegt dann in einem eigenen Resultatsambient vor (um nicht mit den Eingaben zu interferieren).

nand P_1 P_2 P_3 ... P_n = see false.open result.open false.result[true[]] | result[false]] | P_1 | P_2 | P_3 | ... | P_n

(wobei see n.P := (ν r s)(r[in n.out n.r be s.P] | open s))

Dann gilt:

$\exists i \in \{1,...,n\}$: $P_i \Downarrow$ false[] $\Rightarrow$ (nand P_1 P_2 P_3 ... P_n) $\Downarrow$ result[true[]] , sonst (nand P_1 P_2 P_3 ... P_n) $\Downarrow$ result[false[]]

(d.h. wenn sich bei mindestens einem P_i ein true-Ambient ganz oben zeigt (ggf. nach Reduktion), dann wird dieses „nand" ein „result[true[]]"-Ambient liefern, sonst ein „result[false[]]"-Ambient. Man beachte, dass das „open true" eines der „true"-Ambients vernichtet, aber da zuvor durch „open result" eines hinzukam, hat sich an der Situation nichts geändert.

b) Stellen Sie die Wirkungsweise der Iteration dar:

rec $(m_1)P_1$ ... $(m_p)P_p$ in Q := (ν m_1...m_p)(!open $m_1.P_1$ | ... | !open $m_p.P_p$ | Q)

Wählen Sie sich dazu ein geeignetes Q und reduzieren Sie die Formel. (5 Punkte)

Sei Q = $m_1[m_2[m_3[]]]$

rec $(m_1)P_1$ (m_2) P_2 $(m_3)P_3$ in ($m_1[m_2[m_3[]]]$)

= (ν m_1 m_2 m_3) (!open $m_1.P_1$ | !open $m_2.P_2$ | !open $m_3.P_3$ | $m_1[m_2[m_3[]]]$)

→ (ν m_1 m_2 m_3) (!open $m_1.P_1$ | !open $m_2.P_2$ | !open $m_3.P_3$ | $m_2[m_3[]]$) | P_1 ($m_1 \notin fn(P_1)$)

→ (ν m_1 m_2 m_3) (!open $m_1.P_1$ | !open $m_2.P_2$ | !open $m_3.P_3$ | $m_3[]$) | P_1 | P_2 ($m_1 \notin fn(P_1)$, $m_2 \notin fn(P_2)$)

→ (ν m_1 m_2 m_3) (!open $m_1.P_1$ | !open $m_2.P_2$ | !open $m_3.P_3$) | P_1 | P_2 | P_3 ($m_1 \notin fn(P_1)$, $m_2 \notin fn(P_2)$, $m_3 \notin fn(P_3)$)

= rec $(m_1)P_1$ $(m_2)P_2$ $(m_3)P_3$ in 0 | P_1 | P_2 | P_3

≃ P_1 | P_2 | P_3

Wenn also Q entsprechend die „Zählstände" schachtelt, wird bei der Iteration der Reihe nach jedes P_i aktiviert. Danach ist Q „aufgebraucht" und die Iteration inaktiv (die relevanten Namen m_1, ..., m_p sind gebunden und können nur über ein Q aktiviert werden).

Aufgabe 5.4 – 10 Punkte

a) Geben Sie den Typ der folgenden Ausdrücke an und nennen Sie jeweils die Bedingung, unter welcher der Ausdruck wohltypisiert ist. (4 Punkte)

- n[Q]

  n[Q]: S, wenn Q:T und n:Amb[T]

  Das Ambient selbst tritt nicht in Interaktion mit der Umgebung und kann daher jeden Typ annehmen; innerhalb des Ambients müssen die Prozesstypen mit dem Ambienttyp übereinstimmen.

- m[in n]

  Nicht unmittelbar wohltypisiert: der Ausdruck „in n" ist kein Prozesstyp, d.h. man kann die Ambientregel M[P] nicht anwenden.

  Die korrekte (ausführliche) Schreibweise lautet m[in n.**0**]:S, wobei S irgendein Typ sein kann, wenn „in n.**0**" vom Typ T ist. m ist dann vom Typ Amb[T]

- open m | m[P]

  Wie oben: „open m" sollte korrekterweise als „open m.**0**" geschrieben werden.

  (open m.**0** | m[P]): T, wenn open m.**0**: T und m[P]:T

  d.h. wenn open m:Cap[T] (**0**:T ist ja möglich) und P:S, m:Amb[S] (das kann ja innerhalb des m-Ambients zunächst noch frei gewählt sein),

  was aber sogleich erzwingt, dass T=S ist (Regel für open!), also m:Amb[T] und P:T. Dies ist dann auch verträglich mit der Beobachtung, dass (open m.0 | m[P]):T → P:T, dass also der Typ des Ausdrucks derselbe ist, wenn man ihn erst reduziert.

b) Reduzieren Sie den folgenden Ausdruck zunächst ohne Beachtung der Typisierung. (2 Punkte)

n[in m.⟨open k⟩] | m[k[Q] | open n.(x:Cap[S]).x]:Shh mit Q:S

n[in m.⟨open k⟩] | m[k[Q] | open n.(x).x.**0**]

→ m[n[⟨open k⟩] | k[Q] | open n.(x).x.**0**]

→ m[⟨open k⟩ | k[Q] | (x).x.**0**]

→ m[k[Q] | open k.**0**]

→ m[Q]

c) Geben Sie an, ob der Ausdruck unter b) wohltypisiert ist. Versuchen Sie, die Typisierung aus den Typregeln zu erschließen. (Hinweis: Leiten Sie die Typen der übrigen Namen schrittweise ab.) (4 Punkte)

Der Ausdruck ist bezüglich des Typsystems nicht wohlgeformt, denn (x:Cap[S]).x ist nicht typisierbar. Es handelt sich zwar um die Kurzform von (x:Cap[S]).x.**0**, aber es gibt keine Regel, die diesen Ausdruck mit einem Typ versehen kann:

Sei

(x:Cap[S]).x.**0** = (x:W).P :T

Dann muss P:T (nach der Input-Typregel) für P=x.**0** gelten, weil sonst der Input nicht korrekt typisiert ist (der Prozess hat denselben Typ wie der gesamte Ausdruck). Dann ist nach derselben Regel auch W=T (der Typ der Eingabe ist gleich mit dem Typ des Ausdrucks). Das heißt:

(x:Cap[S]).x.**0** : Cap[S]

Nach der Aktionsregel definiert man sich P = M.N = x.**0**, damit also (x:Cap[S]).P:Cap[S] und somit

N:Cap[S] (was korrekt ist – 0 nimmt alles) und schließlich M:Cap[Cap[S]]. Damit ist x:Cap[Cap[S]], aber nach Voraussetzung ist x:Cap[S].

Die Bestimmung der Typen der Namen ist damit hinfällig, da der Ausdruck nicht konsistent typisierbar ist.

Intuitiv kann man Folgendes erkennen: Der Ausdruck scheint sich leicht reduzieren zu lassen, wie wir bereits in Aufgabenteil b) sahen:

n[in m.⟨open k⟩] | m[k[Q] | open n.(x).x.**0**]

→ m[n[⟨open k⟩] | k[Q] | open n.(x).x.**0**]

→ m[⟨open k⟩ | k[Q] | (x).x.**0**]   (*)

→ m[k[Q] | open k.**0**]

→ m[Q]

Aber etwas ist faul: Q ist ja nach Voraussetzung vom Typ S, k also vom Typ Amb[S] (sonst ist k[Q] nicht korrekt). Am Ende ist m ein Ambient, in dem S-Nachrichten auftauchen (m:Amb[S]), nach Schritt (*) wurde aber eine Nachricht „open k“ innerhalb des Ambients verwendet, die zum Freisetzen von S-Nachrichten führten. open k ist vom Typ Cap[S], also müsste eigentlich m:Amb[Cap[S]] sein. Das heißt, innerhalb desselben Ambients wurden verschiedene Nachrichtentypen eingesetzt,

– nämlich zum einen der Typ von Q
– und zum anderen die Berechtigung open m, welche Nachrichten vom Typ von Q freisetzt

was also gegen die Typisierung verstößt.

Es spricht nichts gegen die Anwendung von „open k“ innerhalb des Ambients (wie sollte man sonst überhaupt open einsetzen?). Man darf dieses „open k“ aber nicht als *Nachricht* in diesem Ambient herumschicken. Es wäre legitim gewesen, das open k mithilfe eines Ambients über eine „in-m“-Berechtigung zu injizieren.