

Some for derivations of trans

Ed Wimmers
Sept. 14, 1988

```
Def for == C:( [! [! Elt.isfunc , Lim.isfunc !] , Arg.isval !] ->
  ( AA@Elt : ' ( Arg distl' IntsToLimit )
    where Def IntsToLimit == intsto @ ( Lim : ' Arg ) end ) )
```

```
Ind == s2      Arg == s1
```

```
Arg_1 == Arg
Arg_j == Arg @ Arg_(j-1)
Ind_j == Ind @ Arg_(j-1)
```

Ind_1 is the innermost index, Ind_2 is the second most inner index, etc.
Arg_j is the original argument for a for construct that is nested j deep.

```
app @ [ cons @ f , g ] == AA:app @ distr @ [ f , g ]
  ( assuming that f == DEL:isseq @ f )
  (no gamma assumptions are needed.)
```

```
app @ [ AA @ f , g ] == AA:app @ distl @ [ f , g ]
  ( assuming that f == DEL:isseq @ f )
  (no gamma assumptions are needed.)
```

```
app @ [ C:f@g , h ] == f@[g,h]
  (no gamma assumptions are needed.)
```

```
intsto == for:<Ind,id>
```

```
distr @ [ for:<E,h> , f ] == for : < [E,f@Arg] , h >
  ( usual gamma assumptions)
```

```
distl == for:< [ s1@Arg , Rel@[s2@Arg,Ind] ] , len@s2 >
```

```
AA:f @ for:<E,h> == for:<f@E,h>
  ( usual gamma assumptions)
```

```
for:<E,h>@f == for:<E@[f@Arg,Ind],h@f>
  (usual gamma assumptions.)
```

trans

- (1) == (cons @ AA:(AA@C:s1) @ insto @ len @ s1) : ' id
- (2) == app @ [cons @ AA:(AA@C:s1) @ insto @ len @ s1 , id]
- (3) == AA:app @ distr @ [AA:(AA@C:s1) @ insto @ len @ s1 , id]
- (4) == AA:app @ distr @ [AA:(AA@C:s1) @ for:<Ind,id> @ len @ s1 , id]
- (5) == AA:app @ distr @

[AA:(AA@C:s1) @ for:<Ind@[len@s1@Arg,Ind],len@s1> , id]

```

(6) == AA:app @ distr @ [ AA:(AA@C:sel) @ for:<Ind,len@s1> , id ]
(7) == AA:app @ distr @ [ for:< AA@C:sel@Ind , len@s1 > , id ]
(8) == AA:app @ for:< [ AA@C:sel@Ind , id@Arg ] , len@s1 >
(9) == for:< app @ [ AA@C:sel@Ind , Arg ] , len@s1 >
(10) == for:< AA:app @ dist1 @ [ C:sel@Ind , Arg ] , len@s1 >
(11) == for:< AA:app @
      for:<[s1@Arg,Rel@[s2@Arg,Ind]],len@s2> @ [C:sel@Ind,Arg] , len@s1 >
(12) == for:< AA:app @
      for:<[s1@Arg,Rel@[s2@Arg,Ind]]@[[C:sel@Ind,Arg]@Arg,Ind],
          len@s2@[C:sel@Ind,Arg]> , len@s1 >
(13) == for:< AA:app @
      for:< [s1@Arg,Rel@[s2@Arg,Ind]] @ [[C:sel@Ind_2,Arg_2],Ind_1] ,
          len@Arg_1 > , len@s1 >
(14) == for:< AA:app@for:<[C:sel@Ind_2,Rel@[Arg_2,Ind_1]],len@Arg_1>,len@s1>
(15) == for:< for:<app@[C:sel@Ind_2,Rel@[Arg_2,Ind_1]],len@Arg_1 >, len@s1 >
(16) == for:< for : < sel @ [Ind_2,Rel@[Arg_2,Ind_1]],len@Arg_1 > , len@s1 >
(17) == for:< for : < Rel @ [Rel@[Arg_2,Ind_1],Ind_2],len@Arg_1 > , len@s1 >
(18) == for:< for : < /:Rel @ [ Arg_2,Ind_1,Ind_2 ] , len@Arg_1 > , len@s1 >

```

The identity on sequences of equal-length sequences is:

AA:(for:<Rel,len>) @ for:<Rel,len>

```

(1) == for: < for:<Rel,len> @ Rel , len >
(2) == for: < for:< Rel@[Rel@Arg,Ind] , len@Rel > , len >
(3) == for: < for:< Rel@[Rel@Arg,Ind] , len@Rel > @ [ Arg, Ind] , len >
(4) == for: <
      for:<Rel@[Rel@Arg,Ind]@[[Arg,Ind]@Arg,Ind],len@Rel@[Arg,Ind]>,
      len >
(5) == for: <
      for:<Rel@[Rel@Arg,Ind]@[[Arg_2,Ind_2],Ind_1],len@Rel@[Arg,Ind]>,
      len >
(6) == for: <
      for:< Rel @ [Rel@[Arg_2,Ind_2],Ind_1], len@Rel@[Arg_1,Ind] > ,
      len >

```

- (7) == for: <
 for:< /:Rel [Arg_2 , Ind_2 , Ind_1] , len@Rel@[Arg_1,Ind] > ,
 len >
- (8) == for: <
 for:< /:Rel [Arg_2 , Ind_2 , Ind_1] , len@Rel@[Arg_1,K:1] > ,
 len >
- (9) == for: <
 for:< /:Rel [Arg_2, Ind_2, Ind_1] , len@Rel@[id,K:1]@Arg_1 > ,
 len >
- (10) == for: < for:< /:Rel [Arg_2, Ind_2, Ind_1] , len@s1@Arg_1 > , len >

JB's alternate derivation for trans:

trans

- (1) == AA:(AA:app) @ AA:distr @ dist1 @ [AA:(C:Rel) , intsto @ len @ s1]
- (2) == AA:(AA:app) @ AA:distr @ dist1 @ [AA:(C:Rel) , for:<Ind,id>@len@s1]
- (3) == AA:(AA:app) @ AA:distr @ dist1 @
 [AA:(C:Rel),for:<Ind@[len@s1@Arg,Ind],len@s1>]
- (4) == AA:(AA:app) @ AA:distr @ dist1 @ [AA:(C:Rel) , for:<Ind,len@s1>]
- (5) == AA:(AA:app) @ AA:distr @ for : < [AA:(C:Rel)@Arg , Ind] , len@s1 >
- (6) == AA:(AA:app) @ for : < distr @ [AA:(C:Rel)@Arg , Ind] , len@s1 >
- (7) == for : < AA:app @ distr @ [AA:(C:Rel)@Arg , Ind] , len@s1 >
- (8) == for : < AA:app @ distr @ [for:<C:Rel@Rel,len>@Arg , Ind] , len@s1>
- (9) == for:< AA:app @ distr @
 [for:<C:Rel@Rel@[Arg@Arg,Ind],len@Arg>,Ind],len@s1 >
- (10) == for:< AA:app@for:<[C:Rel@Rel@[Arg@Arg,Ind],Ind@Arg],len@Arg>,len@s1>
- (11) == for:<for :< app@[C:Rel@Rel@[Arg@Arg,Ind],Ind@Arg] ,len@Arg >,len@s1>
- (12) == for:<for :< Rel @ [Rel @ [Arg@Arg,Ind],Ind@Arg],len@Arg >,len@s1>
- (13) == for:< for :< /:Rel @ [Arg@Arg, Ind , Ind@Arg] , len@Arg >,len@s1 >
- (14) == for:< for :< /:Rel @ [Arg2 , Ind1 , Ind2] , len@Arg_1 > , len@s1 >