) CHAPTER FOUR

) Actio Systems

Introduction

This section will introduce the concept of an Action System as a set of parameterless mutually recursive procedures. A program written using labels and jumps translates directly into an action system. Note however that if the end of the body of an action is reached, then control is passed to the action which called it (or to the statement following the action system) rather than "falling through" to the next label. The exception to this is a special action called the terminating action, usually denoted **Z**, which when called results in the immediate termination of the whole action system.
First e prove some useful results about loop unrolling and recursion unfolding.

Defn: If **T** occurs in **S** and replacing any occurrence of **T** in **S** by **exit(0)** gives a terminal statement of **S** then we say that the occurrence of **T** is in a terminal position in **S**.

Lemma: If each **exit** statement of **S** is in a terminal position then:
{depth=n}; guard$_n$(S) ≈ {depth=n}; S[depth:=depth−p / exit(p)]

Proof: Note that **S** cannot contain any **do**-loops since such a loop can only be terminated by an **exit** statement which is within the loop and therefore not in a terminal position. Any loop will not terminate so can be replaced by **abort**.

The proof is by induction on the structure of **S**, ignoring the **do... od** case as noted above. The only non-trivial case is **S=S$_1$;S$_2$**: **S$_1$** cannot contain any **exit** statement since such a statement is not in a terminal position in **S**.
So **guard$_n$(S$_1$) ≈ if depth=n then S$_1$ fi** which preserves **depth**. So
{depth=n}; guard$_n$(S) ≈ {depth=n}; S$_1$; {depth=n}; guard$_n$(S$_2$)
≈ {depth=n}; S$_1$; {depth=n}; S$_2$[depth:=depth−p / exit(p)]
≈ {depth=n}; S$_1$; S$_2$[depth:=depth−p / exit(p)]
≈ {depth=n}; (S$_1$;S$_2$)[depth:=depth−p / exit(p)]
since **S$_1$** has no **exit**s
which proves the result.

Selective unrolling is useful when there is a much more efficient version of the loop body which can be used under certain conditions. Selective unrolling is also frequently used as a part of the proof of other transformations.

Lemma: Selective unrolling of while loops:

$$\Delta \vdash \textbf{while B do S od} \approx \textbf{while B do S; if B} \wedge \textbf{Q then S fi od}$$

Proof: We will prove by induction on **n** that:
while B do S; if B $\wedge$ **Q then S fi od**$^n \leqslant$ **while B do S od**.
while B do S; if B $\wedge$ **Q then S fi od**$^{n+1}$
$\approx$ **if B then S; if B** $\wedge$ **Q then S fi; while B do S; if B** $\wedge$ **Q then S fi od**^{n} **fi**
$\leqslant$ **if B then S; if B** $\wedge$ **Q then S fi; while B do S od fi** by ind hyp
$\approx$ **if B then S; if B** $\wedge$ **Q then S; while B do S od**
else while B do S od fi fi by forward expansion of **if**.
$\approx$ **if B then S; if B** $\wedge$ **Q then if B then S; while B do S od fi**
else while B do S od fi fi
$\approx$ **if B then S; if B** $\wedge$ **Q then while B do S od**
else while B do S od fi fi loop rolling.
$\approx$ **if B then S; while B do S od fi** eliminate conditional.
$\approx$ **while B do S od** loop rolling.
Hence **while B do S; if B** $\wedge$ **Q then S fi od** $\leqslant$ **while B do S od**.

The converse is similar.

The next theorem uses this result to prove a more general form of selective unrolling where a copy of the body of a loop (with an **if** guard) is inserted after certain of its terminal statements. Note that different copies of the body may have different guards.

Theorem: Suppose **S** is such that each terminal statement is of the form **exit(k)** for some **k**$\geqslant$**0**. (Remember any **S** can be converted to such a form by replacing each non-**exit** primitive terminal statement **T** by the equivalent **T;exit(0)**). Suppose also Ψ**(n,T)** is a condition for each integer **n** and each **exit** statement **T** and **Q(n,T)** is a formula for each integer **n** and each **exit** statement **T**. Then:
$\Delta \vdash$ **while B do S od** $\approx$
while B do
S[if B $\wedge$ **Q(n,T) then S fi**$+|\mathbf{T}|/$ **(n,T)**$|\Psi$**(n,T)** $\wedge$ **ts(n,T,S)** $\wedge$ τ**(n,T,S)=0**] **od**

Here we substitute **if B** $\wedge$ **Q(n,T) then S fi** $+|\mathbf{T}|$ for the **n**th occurrence of **T** in **S** if this is an **exit** statement with terminal value zero. Note $|$ **exit(k)**$| =$**k**; also, the condition τ**(n,T,S)=0** is not strictly necessary here since a terminal statement in the body of a **while** loop may not have terminal value greater than zero–it will be important for the next theorem though.

This substitution ("Selective Substitution of Terminal Statements") will be useful in other theorems so we will abbreviate it to:
while B do S[if B $\wedge$ **Q then S**$+|\mathbf{T}|$ **else T fi** $/$ **T**$|\Psi \wedge \tau =$**0**] **od**

Examples: Ψ can be any condition on integers and **exit** statements for example:

$$\Psi(\mathbf{n,T}) \equiv \mathbf{false}$$

means that no substitution takes place: the theorem is trivial in this case.

$$\Psi(\mathbf{n,T}) \equiv \mathbf{true}$$

means that every allowed substitution takes place: if $\mathbf{Q(n,T)}$ gives the same formula for each $\mathbf{n}$ and $\mathbf{T}$ then this is equivalent to the result of the last theorem with simple absorption applied.

$$\Psi(\mathbf{n,T}) \equiv (\mathbf{n=1} \vee \mathbf{n=3}) \wedge \mathbf{T=T}_0$$

means that the first and third occurrences of $\mathbf{T}_0$ are substituted provided the other conditions hold.

Proof: Let $\mathbf{x}$ be a new variable which does not occur on $\mathbf{S}$, $\mathbf{B}$ or $\mathbf{Q}$. Then

while B **do** S **od**

$\approx$ **beg** x: **while** B **do** S **od end** since $\mathbf{x}$ does not occur in $\mathbf{S}$.

$\approx$ **beg** x: **while** B **do** x:=0; S [**if** Q(n,T) **then** x:=1 **else** x:=0 **fi**; T / (n,T)
$|\Psi(\mathbf{n,T}) \wedge \mathbf{ts(n,T,S)} \wedge \tau(\mathbf{n,T,S})=\mathbf{0}$]
[x:=0;T / T|$\neg\Psi \wedge \tau=$0]
[x:=0;T / T|$\tau>$0] **od end**

since the local variable $\mathbf{x}$ can be freely assigned to.

Note that if the $\mathbf{n}$th occurrence of $\mathbf{T}$ is a terminal statement then: if $\Psi(\mathbf{n,T})$ is satisfied and $\tau(\mathbf{n,T,S})=\mathbf{0}$ the value $\mathbf{1}$ is assigned to $\mathbf{x}$, while the value $\mathbf{0}$ is assigned to $\mathbf{x}$ if $\neg\Psi(\mathbf{n,T})$ or $\tau(\mathbf{n,T,S})>\mathbf{0}$. This covers all the terminal statements of $\mathbf{S}$.

$\approx$ **beg** x: **while** B **do** S [**if** Q(n,T) **then** x:=1 **else** x:=0 **fi**; T / T|$\Psi \wedge \tau=$0]
[x:=0;T / T|$\neg\Psi \wedge \tau=$0]
[x:=0;T / T|$\tau>$0];
if B $\wedge$ x=1 **then** S[...] **fi** **od end**

by selective unrolling (where [...] represents the substitutions).

$\approx$ **beg** x: **while** B **do** S[**if** Q(n,T) **then** x:=1 **else** x:=0 **fi**;
if B $\wedge$ Q $\wedge$ x=1 **then** S[...] **fi** +|T| / T|$\Psi \wedge \tau=$0]
[x:=0; **if** B $\wedge$ Q $\wedge$ x=1 **then** S[...] **fi** +|T| / T|$\neg\Psi \wedge \tau=$0]
[x:=0;T / T|$\tau>$0] **od end**

by simple absorption.

$\approx$ **beg** x: **while** B **do** S[**if** Q(n,T)
then x:=1; **if** B $\wedge$ x=1 **then** S[...] **fi**
else x:=0; **if** B $\wedge$ x=1 **then** S[...] **fi** **fi** +|T|
/ T|$\Psi \wedge \tau=$0]
[x:=0;T / T|$\neg\Psi \wedge \tau=$0][x:=0;T / T|$\tau>$0] **od end**

$\approx$ **beg x: while B do S[if Q(n,T) then x:=1; if B then S[...] fi**
else x:=0 fi $+|\mathbf{T}| \ / \ \mathbf{T}|\Psi \wedge \tau = \mathbf{0}$]
[x:=0;T / T]$|\neg\Psi \wedge \tau = \mathbf{0}]$**[x:=0;T / T]**$|\tau > \mathbf{0}]$ **od end**

by pruning the **if** statements. Now **x** is no longer accessed.

$\approx$ **while B do S[if B** $\wedge$ **Q(n,T) then S fi** $+|\mathbf{T}| \ / \ \mathbf{T}|\Psi \wedge \tau = \mathbf{0}$]
[T / T]$|\neg\Psi \wedge \tau = \mathbf{0}]$**[T / T]**$|\tau > \mathbf{0}]$ **od**

by removing the ghost variable **x**.

$\approx$ **while B do S** [**if B** $\wedge$ **Q(n,T) then S fi** $+|\mathbf{T}| \ / \ \mathbf{T}|\Psi \wedge \tau = \mathbf{0}$] **od**

as required.

The technique used in this proof is to add a "ghost variable" (**x** in the proof) which is set to one value for some of the terminal statements and some of the conditions and to another value for the others and which can then be tested in a following **if** statement so that the branch of the **if** taken depends on which terminal statement was executed. This technique is made use of in several forms of selective unrolling and unfolding.

Theorem: Selective unrolling for unbounded loops. (ie **do...od** loops):
If Ψ, **Q** and **S** are as above :

$$\Delta \vdash \textbf{do S od} \approx \textbf{do S[if Q(n,T) then S fi}+|\mathbf{T}|/\ \mathbf{T}|\Psi \wedge \tau = \mathbf{0}]\ \textbf{od}$$

Proof: Similar to the previous theorem.

Example:
P $\equiv$ **while n**$\geqslant$**2 do**
if even(n) then n:=n/2
else n:=n+1 fi od.

To prove the termination of this directly we need to find a term ϕ**(n)** which is decreased on every step of the loop–which is not particularly easy!

One possibility is: ϕ**(n)**= **n+1−2.((n+1) mod 2)**
which gives: ϕ**(n)** = **n−1** if **n** is even
= **n+1** if **n** is odd.
If **n** is odd then ϕ**(n)**=**n+1** and ϕ**(n+1)**=**n** $<$ ϕ**(n)** since **n+1** is even.
If **n** is even then ϕ**(n)**=**n−1**, ϕ**(n/2)** =**n/2−1** if **n/2** is even
=**n/2+1** if **n/2** is odd.
n/2−1 $<$ **n−1** for all integers **n**$>$**0**.
n/2+1 $<$ **n−1** $\iff$ **n−n/2** $>$ **2** $\iff$ **n**$>$**4**
which is true for all even integers **n**$>$**2** such that **n/2** is odd.
So if **n** is even and **n**$>$**2**:

$\phi(\mathbf{n/2}) = \mathbf{n/2-1} < \mathbf{n-1} = \phi(\mathbf{n})$ if $\mathbf{n/2}$ is even.
$= \mathbf{n/2+1} < \mathbf{n-1} = \phi(\mathbf{n})$ if $\mathbf{n/2}$ is odd.
So $\phi(\mathbf{n})$ is decreased on every step and the program terminates.

As is frequently the case with such proofs of termination, finding a function $\phi(\mathbf{n})$ which is decreased on every iteration required some ingenuity. However if we note that $\mathbf{n+1}$ is even when $\mathbf{n}$ is odd then we see that the next iteration of the loop will select the assignment $\mathbf{n:=n/2}$ and $\mathbf{(n+1)/2}$ is always $<\mathbf{n}$ for odd $\mathbf{n>1}$. To prove termination therefore we use selective unrolling to insert a copy of the body of the loop after the **else** part of the second **if** statement:

P $\approx$ **while** $\mathbf{n\geqslant 2}$ **do**
  if even(n) then n:=n/2
    else n:=n+1;
      if $\mathbf{n\geqslant 2}$
        then if even(n) then n:=n/2
        else n:=n+1 fi fi od.

After the first **if** in the body we must have $\mathbf{n\geqslant 2}$, hence after $\mathbf{n:=n+1}$ we must have $\mathbf{n\geqslant 3}$. Also $\mathbf{n}$ was odd before $\mathbf{1}$ was added so $\mathbf{n}$ must be even:

P $\approx$ **while** $\mathbf{n\leqslant 2}$ **do**
  if even(n) then n:=n/2
    else n:=n+1; $\mathbf{\{n\geqslant 3 \wedge even(n)\}}$**;**
      if $\mathbf{n\leqslant 2}$
        then if even(n) then n:=n/2
        else n:=n+1 fi fi od.

Prune the **if**s after the assertion and merge the two assignments:

P $\approx$ **while** $\mathbf{n\leqslant 2}$ **do**
  if even(n) then n:=n/2
    else n:=(n+1)/2 fi od.

Now $\mathbf{(n+1)/2 < n} \iff \mathbf{n+1 < 2.n} \iff \mathbf{1<n}$
which is true within the loop. Hence $\mathbf{n}$ is decreased on every step and the loop terminates. So, by carrying out some transformations, we have turned a difficult termination proof into a trivial one (and incidentally made the program more efficient).

We now prove a type of "selective unfolding" for recursive statements:

Theorem: Selective Unfolding: $\Delta \vdash$ **proc X** $\equiv$ **S.** $\approx$ **proc X** $\equiv$ **S[if Q then S else X fi / X].**
Proof: We prove by induction on $\mathbf{n}$:
$$\textbf{proc X} \equiv \textbf{S[if Q then S else X fi / X]}.^{n} \leqslant \textbf{proc X} \equiv \textbf{S.}$$

The induction step is:
proc X ≡ S[if Q then S else X fi / X].$^{n+1}$
$\approx$ **S[if Q then S else X fi / X][proc X ≡ S[if Q then S else X fi / X].**^{n} **/ X]**
$\leqslant$ **S[if Q then S else X fi / X][proc X ≡ S. / X]** by induction hypothesis.
$\approx$ **S[if Q then S[proc X ≡ S. / X] else X[proc X ≡ S. / X] fi / X]**
$\approx$ **S[if Q then proc X ≡ S. else proc X ≡ S. fi / X]** by folding.
$\approx$ **S[proc X ≡ S. / X]** redundant test.
$\approx$ **proc X ≡ S.** folding.

Hence **proc X ≡ S[if Q then S else X fi / X]. $\leqslant$ proc X ≡ S.** by induction rule for recursion.

Conversely we prove **proc X ≡ S.**^{n} **$\leqslant$ proc X ≡ S[if Q then S else X fi / X].** by induction.
Let **S′ = proc X ≡ S[if Q then S else X fi / X].**
n=0 trivial
n=1: **proc X ≡ S.**^{1} $\approx$ **S[abort/X] $\leqslant$ S[if Q then S else X fi / X][S′/ X]**
$\approx$ **S′** by folding.
For **n>1**:
proc X ≡ S.$^{n+1}$ $\approx$ **S[proc X ≡ S.**^{n} **/ X]** $\approx$ **S[if Q then proc X ≡ S.**^{n} **else proc X ≡ S.**^{n} **fi / X]**
$\approx$ **S[if Q then S[proc X ≡ S.**$^{n-1}$**/X] else proc X ≡ S.**^{n} **fi / X]** since **n>1**.
$\leqslant$ **S[if Q then S[S′/X] else S′ fi / X]** by induction hyp
$\approx$ **S[if Q then S else X fi / X][S′/X]**
$\approx$ **S′** by folding.

As before, we can generalise this selective unfolding so that, for each **n**, the **n**th **X** in **S** may be replaced by **if Q(n,X) then S else X fi** where **Q** may vary depending on which **X** is substituted.

The following Lemma will be used in several theorems concerned with the transformation of recursive action systems to iterative form:

Lemma: Unroll Last Step of Loop:
If **B′** $\Rightarrow$ **¬B** and $\Delta \vdash$ **{B′}; S′ $\approx$ {B′}; S′; {¬B}** then:
$\Delta \vdash$ **{B}; while B do S; if B′ then S′ fi od $\approx$ {B}; while B do S od; if B′ then S′ fi**
Proof: Use induction on **n** to prove:
{B}; while B do S; if B′ then S′ fi od^{n+1} $\approx$ **{B}; while B do S od**^{n}**; if B′ then S′ fi**
Then use the generalised induction rule for loops.

The reasoning behind normal loop unfolding is that under certain conditions the body of a loop may simplify drastically and hence may be replaced by a much more efficient version. However, in some cases it may be that the condition which allowed us to simplify the loop body is likely still to hold after execution of the simplified body and it makes sense to try and insert a whole loop rather than just the body of the loop. We wish to prove the following theorem:

Theorem: (Entire Loop Unfolding): If $\mathbf{B}' \Rightarrow \mathbf{B}$ then
$\Delta \vdash$ **while B do S od** $\approx$ **while B do S; if Q then while B′ do S od fi od**.
Proof: We use the induction rule for loops:
Let **DO**= **while B do S od**
Let **DO′** = **while B′ do S od**

Claim(i): For $\mathbf{n} < \omega$: **while B do S; if Q then DO′ fi od**$^n \leqslant$ **DO**.
Proof of Claim(i): $\mathbf{n}=\mathbf{0}$ is trivial. Suppose result holds for **n**:
while B do S; if Q then DO′ fi od^{n+1}
$\approx$ **if B then S; if Q then DO′ fi; while B do S; if Q then DO′ fi od**^{n} **fi**
$\leqslant$ **if B then S; if Q then DO′ fi; DO fi** by induction hypothesis.
$\approx$ **if B then S; if Q then DO′; DO**
else DO fi forward expansion of **if**
Now $\mathbf{B}' \Rightarrow \mathbf{B}$ gives **DO′; DO** $\approx$ **DO** by loop merging, so this is:
$\approx$ **if B then S; if Q then DO else DO fi**
$\approx$ **if B then S; DO** prune conditional.
$\approx$ **DO** by first step unrolling.
Hence by the induction rule for loops:
while B do S; if Q then DO′ fi od$^n \leqslant$ **DO** for all $\mathbf{n} < \omega$.

Claim(ii): For $\mathbf{n} < \omega$: **DO**$^n \leqslant$ **while B do S; if Q then DO′ fi od**.
Proof of Claim(ii): $\mathbf{n}=\mathbf{0}$ is trivial. Suppose result holds for **n**:
DO^{n+1} $\approx$ **if B then S; DO**^{n} **fi**
$\approx$ **if B then S; if Q then DO**^{n} **else DO**^{n} **fi** by splitting a tautology.
From the proof of loop merging we have **DO**$^n \leqslant$ **DO′** ^{n}**; DO**^{n}, we also have **DO′** $^n \leqslant$**DO′** so **DO**$^n \leqslant$ **DO′; DO**^{n}. So we get:
$\leqslant$ **if B then S; if Q then DO′; DO**^{n} **else DO**^{n} **fi**
$\leqslant$ **if B then S; if Q then DO′; while B do S; if Q then DO′ fi od**
else while B do S; if Q then DO′ fi od fi
by induction hypothesis.
$\approx$ **if B then S; if Q then DO′ fi; while B do S; if Q then DO′ fi od fi**

by forward expansion of **if**.

$\approx$ **while B do S; if Q then DO′ fi od** by first step unrolling.

As was the case for selective unrolling we can use this result to prove a more general version where the unfolded loop is inserted after certain selected terminal statements of the loop:

Theorem: For each **n** and **T** where the **n**th occurrence of **T** is a terminal statement of **S**, if the formula $\mathbf{B'(n,T)}\Rightarrow\mathbf{B}$ holds then:

$\Delta \vdash$ **while B do S od**

$\approx$ **while B do**

S[(if Q(n,T) then while B′(n,T) do S od fi) $+ \delta$**(T) / (n,T)** $|\tau = \mathbf{0} \wedge \Psi]$ **od**

Note that **while B do S od**$+\mathbf{k}$ $\approx$ **while B do S od; exit(k)** since a **while** loop is a terminal statement.

For unbounded loops we have the theorem:

$\Delta \vdash$ **do S od** $\approx$

do S[(if Q(n,T) then (do if Q(n,T) then S else exit fi od)$+\mathbf{1}$ **fi)** $+ \delta$**(T) / (n,T)** $|\tau = \mathbf{0} \wedge \Psi]$ **od**

For double-nested loops we can replace terminal statements with terminal value **0** or **1** by copies of the whole loop:

$\Delta \vdash$ **do do S od od** $\approx$

do do S[(if Q(n,T)

then do do if Q(n,T) then S$+\mathbf{(2,2)}$ **else exit(2) fi od od fi)** $+ \delta$**(T) / (n,T)**

$|(\tau = \mathbf{0} \vee \tau = \mathbf{1}) \wedge \Psi]$ **od od**

This important general transformation is used in the transformation from recursion to iteration in a regular action system (see Theorem A below). The proof is more complex than the other forms of selective unrolling but follows the same pattern:

(i) Add assignments to a new variable at each terminal position, assigning **1** if the unrolling is to occur and **0** if the unrolling is not to occur.

(ii) Unroll the loop, including a test of this variable.

(iii) Absorb the unrolled loop into the terminal positions.

(iv) Simplify the result, removing the variable **x**.

Example: This is a generalisation of an example in [Dij kstra 76].

Let **DO**= **while** B **do**
while B′ ∧B **do** S′ **od;**
S **od**
Let **DO**″ = **while** B **do**
if B′ **then** S′
else **S** **fi** **od**
Then **DO** ≈ **DO**″.

Proof: By entire loop unrolling after **S**′ in **DO**″.

Cor: If **{B** ∧ **B′}; S′** ⩽ **{B** ∧ **B′}; S′; {B}** then:
while B **do** ≈ **while** B **do**
while B′ **do** S′ **od;** **if** B′ **then** S
S **od** **else** S′ **fi** **od**
Proof: Show that **B** is invariant over the inner loop.

Theorem: Elimination of Tail-Recursion:
If **A** does not occur in **S** or $\mathbf{S}_1$ then:
$\Delta \vdash$ **{B}; proc A** ≡ **S; if B then A else** $\mathbf{S}_1$ **fi.**
≈ **{B}; while B do S od;** $\mathbf{S}_1$

Proof: By the definition of the **while** loop we have:
{B}; while B do S od; $\mathbf{S}_1$
≈ **{B}; proc A** ≡ **if B then S; A fi. ;** $\mathbf{S}_1$
For each **n**< ω use absorption to prove:
{B}; proc A ≡ **if B then S; A fi.**^{n} **;** $\mathbf{S}_1$ ≈ **{B}; proc A** ≡ **if B then S; A else** $\mathbf{S}_1$ **fi.**^{n}
Then by unfolding we get:
≈ **{B}; S; proc A** ≡ **if B then S; A else** $\mathbf{S}_1$ **fi.**$^{n-1}$
≈ **{B}; proc A** ≡ **S; if B then A else** $\mathbf{S}_1$ **fi.**^{n} by induction on **n**.
This completes the proof.

Theorem: Tail-Recursion (general form):
$\Delta \vdash$ **proc A** ≡ **if B then S; A else** $\mathbf{S}_1$ **fi.** ≈ **proc A** ≡ **while B do S od;** $\mathbf{S}_1$**.**
Note that in this case **S** or $\mathbf{S}_1$ may contain further calls to **A**. Ths transformation is an example of "replacing a terminal procedure call by a jump" as described in [Knuth 74].
Proof: By induction. The induction step is:
proc A ≡ **if B then S; A else** $\mathbf{S}_1$ **fi.**$^{n+1}$

$\approx$ **if B then S[proc A ≡ if B then S; A else S$_1$ fi.^{n}/A]; proc A ≡ if B then S; A fi.^{n}**
else S$_1$[proc A ≡ if B then S; A else S$_1$ fi.^{n}/A] fi
$\leqslant$ **if B then S[proc A ≡ while B do S od; S$_1$. /A]; proc A ≡ while B do S od; S$_1$.**
else S$_1$[proc A ≡ while B do S od; S$_1$. /A] fi by induction hypothesis.
Let **DO= while B do S od; S$_1$**.
Now **proc A ≡ DO.** $\approx$ **while B do S[proc A ≡ DO./A] od; S$_1$[proc A ≡ DO./A]** by unfolding
$\approx$ **if B then S[proc A ≡ DO./A]; while B do S[proc A ≡ DO./A] od fi;**
S$_1$[proc A ≡ DO./A] by unrolling
$\approx$ **if B then S[proc A ≡ DO./A]; proc A ≡ DO. fi** by folding
Hence **proc A ≡ if B then S; A else S$_1$ fi.$^{n+1}$** $\leqslant$ **proc A ≡ DO.**
Hence by induction **proc A ≡ if B then S; A fi.** $\leqslant$ **proc A ≡ DO.**

For the converse we first show:
while B do S[proc A ≡ if B then S; A else S$_1$ fi./A] od^m;
S$_1$[proc A ≡ if B then S; A else S$_1$ fi./A]
$\leqslant$ **proc A ≡ if B then S; A else S$_1$ fi.**
for all **m**$< \omega$ by induction on **m**. Induction step is:
while B do S[proc A ≡ if B then S; A else S$_1$ fi./A] od^{m+1}; S$_1$[...]
$\approx$ **if B then S[proc A ≡ if B then S; A else S$_1$ fi./A]; while B do... od^m fi; S$_1$[...]**
$\leqslant$ **if B then S[proc A ≡ if B then S; A else S$_1$ fi./A];**
proc A ≡ if B then S; A else S$_1$ fi. fi by induction hypothesis.
$\approx$ **if B then S; A else S$_1$ fi [proc A ≡ if B then S; A fi. /A]**
$\approx$ **proc A ≡ if B then S; A else S$_1$ fi.** by folding.
Hence **while B do S[proc A ≡ if B then S; A else S$_1$ fi./A] od; S$_1$[...]**
$\leqslant$ **proc A ≡ if B then S; A else S$_1$ fi.**.

Now we prove by induction on **n**:
proc A ≡ while B do S od; S$_1$.^{n} $\leqslant$ **proc A ≡ if B then S; A else S$_1$ fi.**.
Induction step:
proc A ≡ while B do S od; S$_1$.$^{n+1}$
$\approx$ **while B do S[proc A ≡ while B do S od.^{n}/A] od; S$_1$[...]**
$\leqslant$ **while B do S[proc A ≡ if B then S; A fi./A] od; S$_1$[...]**
$\leqslant$ **proc A ≡ if B then S; A else S$_1$ fi.** from above.
Hence by induction **proc A ≡ DO.** $\leqslant$ **proc A ≡ if B then S; A else S$_1$ fi.**

Hence **proc A ≡ while B do S od; SS1.** $\approx$ **proc A ≡ if B then S; A else S$_1$ fi.**.

ACTIONS

Our recursive statement does not directly allow the definition of mutually recursive procedures (since all calls to a procedure must occur within the procedure body). However we can define a set of mutually recursive procedures by putting them all within a single procedure. For example suppose we have two statements, $\mathbf{S}_1$ and $\mathbf{S}_2$ both containing statement variables $\mathbf{X}_1$ and $\mathbf{X}_2$ (where $\mathbf{X}_1$ refers to $\mathbf{S}_1$ and $\mathbf{X}_2$ refers to $\mathbf{S}_2$). We can represent these by a single recursive program:
x:=1; proc A ≡ if x=1 → $\mathbf{S}_1$[x:=1;A/$\mathbf{X}_1$][x:=2;A/$\mathbf{X}_2$]
□ x=2 → $\mathbf{S}_2$[x:=1;A/$\mathbf{X}_1$][x:=2;A/$\mathbf{X}_2$] fi.

Here the additional variable **x** records which is the next procedure which is actually being called when the composite procedure **A** is called. This method of representing mutually recursive procedures will be used to develop techniques for reasoning about action systems. Two action systems are defied to be equivalent if they have equivalent representations. This type of transformation to a different representation is an example of a definitional transformation; by defining the new construct in terms of constructs already introduced we can make full use of the results already proved to prove things about the new construct.

Arsac [Arsac 79] and [Arsac 82] uses a restricted definition of actions together with deterministic assignments, the binary **if** statement and **do** loops with **exit**s so there is no place for nondeterminism in his results. He distinguishes between what he calls "syntactic equivalence" (for programs which perform the same sequence of operations and thus have the same operational semantics) and "semantic equivalence" (for programs which give the same final state, if any, for each initial state and so have the same denotational semantics). He uses mainly syntactic transformations together with a few semantic transformations.

We shall give a more general definition of actions in terms of our recursive statement. This will allow us to use all our other control structures, including the nondeterministic ones and general specifications, within our actions.

Defn: An action is a parameterless procedure acting on global variables (cf [Arsac 79], [Arsac 82]). It is written in the form $\mathbf{A} \equiv \mathbf{S}$ where **A** is a statement variable (the name of the action) and **S** is a statement (the action body). A set of (mutually recursive) actions is called an action system. There may sometimes be a special action (usually denoted **Z**), execution of which causes termination of the whole action system even if there are unfinished recursive calls.

The action system: $\mathbf{AS} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1, \mathbf{A}_2 \equiv \mathbf{S}_2, \ldots, \mathbf{A}_n \equiv \mathbf{S}_n \rangle$ (where $\mathbf{S}_1, \ldots, \mathbf{S}_n$ have no terminal statement with terminal value $>\mathbf{0}$) which may be written:

$\mathbf{A}_1 \equiv \mathbf{S}_1$.
$\mathbf{A}_2 \equiv \mathbf{S}_2$.
. . .
$\mathbf{A}_n \equiv \mathbf{S}_n$.

is defined (by a definitional transformation) as follows:

AS $\approx$ **beg action:='A$_1$': (μA.S) end**
where **S= if action='A$_1$'** $\rightarrow$ **action:='O'; guard$_Z$(S$_1$)[action:='A$_i$';A/A$_i$]**
$\Box$ **action='A$_2$'** $\rightarrow$ **action:='O'; guard$_Z$(S$_2$)[action:='A$_i$';A/A$_i$]**
.
.
$\Box$ **action='A$_n$'** $\rightarrow$ **action:='O'; guard$_Z$(S$_n$)[action:='A$_i$';A/A$_i$] fi.**

Here **action** is a new variable which contains the name of the current action and **guard$_Z$(S)** is defined in a similar way to **guard$_n$(S)** so that:
guard$_Z$(Z) $=_{DF}$ **action:='Z'**
guard$_Z$(v:=e) $=_{DF}$ **if action='O' then v:=e fi** etc.
and as soon as **action** is set to **'Z'** no further statements will be executed. This ensures the correct operation of the "halting" action. Note that **'A$_1$'**, ..., **'A$_n$'**, **'O'** and **'Z'** represent a suitable set of **n+2** distinct constant values.

Note that **A** is never called with **action='Z'** (or anything other than **'A$_1$'**,...,**'A$_n$'**). The assignment **action:='O'** is not really needed because the variable **action** will be assigned again before its value is tested; it is added so that we can distinguish the following three cases depending on the value of **action**:

1/ Currently executing an action: **action='O'**.
2/ About to call another (or the same) action (other than the terminating action):
action= one of **'A$_1$'**,...,**'A$_n$'**.
3/ Have called the terminating action, all outstanding recursive calls are terminated without any statements being executed: **action='Z'**.

Defn: An action is regular if every execution of the action leads to an action call. (This is similar to a regular rule in a Post production system [Post 43]).

Defn: An action system is regular if every action in the system is regular. Any algorithm defined by a flowchart or a program which contains labels and **goto**s but no procedure calls in non-terminal positions can be expressed as a regular action system.

TRANSFORMATIONS INVOLVING ACTIONS

We will now prove some basic properties of action systems which are frequently used in program manipulation. "Substitution" is a form of procedure call unfolding, "Identification" is used in [Arsac 82], we are able to prove this and many other transformations using our "Induction rule for

Action Systems".

Substitution:

We can always replace any occurrence of an action call by the body of the action.
Conversely any statement can be replaced by a new action which has that statement as definition.
More formally:

(a) $\langle \mathbf{A}_1 \equiv \mathbf{S}_1 \text{ ,..., } \mathbf{A}_i \equiv \mathbf{S}_i \text{ ,..., } \mathbf{A}_n \equiv \mathbf{S}_n \rangle \approx \langle \mathbf{A}_1 \equiv \mathbf{S}_1 \text{ ,..., } \mathbf{A}_i \equiv \mathbf{S}_i[\mathbf{S}_j/(\mathbf{m},\mathbf{A}_j)] \text{ ,..., } \mathbf{A}_n \equiv \mathbf{S}_n \rangle$

Where $\mathbf{S}_i[\mathbf{S}_j/(\mathbf{m},\mathbf{A}_j)]$ means $\mathbf{S}_i$ with the $\mathbf{m}$th occurrence of $\mathbf{A}_j$ replaced by $\mathbf{S}_j$.

(b) $\langle \mathbf{A}_1 \equiv \mathbf{S}_1 \text{ ,..., } \mathbf{A}_i \equiv \mathbf{S}_i \text{ ,..., } \mathbf{A}_n \equiv \mathbf{S}_n \rangle \approx \langle \mathbf{A}_1 \equiv \mathbf{S}_1 \text{ ,..., } \mathbf{A}_i \equiv \mathbf{S}_i[\mathbf{A}_{n+1}/(\mathbf{m},\mathbf{S}_{n+1})] \text{ ,..., } \mathbf{A}_n \equiv \mathbf{S}_n \text{ , } \mathbf{A}_{n+1} \equiv \mathbf{S}_{n+1} \rangle$

Where $\mathbf{S}_{n+1}$ is any statement and $\mathbf{A}_{n+1}$ is a new action name.

Proof: (a) **LHS** is equivalent (by the definitional transformation) to:
beg action:='$\mathbf{A}_1$': (μA.S) end
where **S= if action='$\mathbf{A}_1$' → action:='O'; guard$_Z$($\mathbf{S}_1$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]**
□ action='$\mathbf{A}_2$' → action:='O'; guard$_Z$($\mathbf{S}_2$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]
. . .
□ action='$\mathbf{A}_i$' → action:='O'; guard$_Z$($\mathbf{S}_i$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]
. . .
□ action='$\mathbf{A}_n$' → action:='O'; guard$_Z$($\mathbf{S}_n$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$] fi.
where the substitution [action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$] is short for:
[action:='$\mathbf{A}_1$';A, action:='$\mathbf{A}_2$';A, . . . , action:='$\mathbf{A}_n$';A / $\mathbf{A}_1$,$\mathbf{A}_2$,...,$\mathbf{A}_n$]

RHS is similarly equivalent to:
beg action:='$\mathbf{A}_1$': (μA.S′) end
where **S′ = if action='$\mathbf{A}_1$' → action:='O'; guard$_Z$($\mathbf{S}_1$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]**
□ action='$\mathbf{A}_2$' → action:='O'; guard$_Z$($\mathbf{S}_2$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]
. . .
□ action='$\mathbf{A}_i$' → action:='O'; guard$_Z$($\mathbf{S}_i$[$\mathbf{S}_j$/(m,$\mathbf{A}_j$)])[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]
. . .
□ action='$\mathbf{A}_n$' → action:='O'; guard$_Z$($\mathbf{S}_n$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$] fi.

By selective unrolling of the recursive statement we can replace the $\mathbf{m}$th occurrence of **action:='$\mathbf{A}_j$';A** in **guard$_Z$($\mathbf{S}_i$)[action:='$\mathbf{A}_k$';A/$\mathbf{A}_k$]** in **LHS** by **action:='$\mathbf{A}_j$';S**. Then, by pruning the inner **if**, we get:
beg action:='$\mathbf{A}_1$': (μA.S) end
where **S= if...**

□ **action=‘$\mathbf{A}_i$’** $\rightarrow$
action:=‘O’;
guard$_Z$($\mathbf{S}_i$)[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]
[action:=‘$\mathbf{A}_j$’; action:=‘O’;
guard$_Z$($\mathbf{S}_j$)[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$] / (m,action:=‘$\mathbf{A}_j$’;A)]
... fi.

Here the **m**th occurrence of $\mathbf{A}_j$ is first replaced by **action:=‘$\mathbf{A}_j$’;A** which is then replaced by **action:=‘O’; $\mathbf{S}_j$[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]**. The **guard$_Z$** outside means that the assignment **action:=‘O’** can be removed. Thus the **m**th occurrence of $\mathbf{A}_j$ is replaced by **guard$_Z$($\mathbf{S}_j$)[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]**.

The same result is achieved by replacing the **m**th occurrence of $\mathbf{A}_j$ by **guard$_Z$($\mathbf{S}_j$)** and then carrying out the substitution **[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]**. Thus we get:
beg action:=‘$\mathbf{A}_1$’: (μA.S) end
where **S= if...**
□ **action=‘$\mathbf{A}_i$’** $\rightarrow$ **guard$_Z$($\mathbf{S}_i$)[guard$_Z$($\mathbf{S}_j$)/(m,$\mathbf{A}_j$)][action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]**
... fi.
This is equivalent (by the definition of **guard$_Z$**) to the RHS.

(b) Add an extra component to the **if** statement of the form:
□ **action=‘$\mathbf{A}_{n+1}$’** $\rightarrow$ **action:=‘O’; guard$_Z$($\mathbf{S}_{n+1}$)[action:=‘$\mathbf{A}_k$’;$\mathbf{A}/\mathbf{A}_k$]**
which is never executed since **action$\neq$‘$\mathbf{A}_{n+1}$’** is invariant. By part (a) we can replace an occurrence of $\mathbf{S}_{n+1}$ in $\mathbf{S}_i$ by $\mathbf{A}_{n+1}$ which gives the result.

Induction Rule for Action Systems:

Defn: If $\mathbf{AS} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1, \mathbf{A}_2 \equiv \mathbf{S}_2, \ldots, \mathbf{A}_p \equiv \mathbf{S}_p \rangle$ is an action system and $\mathbf{1} \leqslant \mathbf{i} \leqslant \mathbf{p}$ then we define
$\mathbf{AS}^0(\mathbf{i}) =_{DF}$ **abort**
$\mathbf{AS}^n(\mathbf{i}) =_{DF}$ **action:=‘O’; guard$_Z$($\mathbf{S}_i$)[$\mathbf{AS}^{n-1}(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{p}$]** for $\mathbf{0} < \mathbf{n} < \omega$.
This is a generalised version of **proc X** $\equiv$ **S.**^{n} with provision for the terminating action.
$\mathbf{AS}^n(\mathbf{i})$ is the “**n**th truncation” of the **i**th action in the system.

Theorem: Induction rule for action systems:
If $\mathbf{AS} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1, \mathbf{A}_2 \equiv \mathbf{S}_2, \ldots, \mathbf{A}_p \equiv \mathbf{S}_p \rangle$ is an action system and $\mathbf{S}'$ a statement such that
$\mathbf{AS}^n(\mathbf{1}) \leqslant \mathbf{S}'$ for all $\mathbf{n} < \omega$
Then $\mathbf{AS} \leqslant \mathbf{S}'$, ie $\mathbf{A}_1 \leqslant \mathbf{S}'$ (where we are using the name of the first action to denote the whole action system).

Proof: $\mathbf{AS} \approx \underline{\mathbf{beg}}\ \mathbf{action{:=}`A_1'}:\ (\mu\mathbf{A.S})\ \underline{\mathbf{end}}$
where $\mathbf{S} = \underline{\mathbf{if}}\ \mathbf{action{=}`A_1'} \rightarrow \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_1)[\mathbf{action{:=}`A_k';A/A}_k]$
$\Box$ **...**
$\Box\ \mathbf{action{=}`A_n'} \rightarrow \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_n)[\mathbf{action{:=}`A_k';A/A}_k]\ \underline{\mathbf{fi}}$
$\mathbf{action{:=}`A_i';\ (\mu A.S)}^n \approx$ **abort** if $\mathbf{n{=}0}$
$\approx \mathbf{action{:=}`A_i';\ S}[(\mu\mathbf{A.S})^{n-1}/\mathbf{A}]$ if $\mathbf{n{>}0}$
$\approx \mathbf{action{:=}`A_i';\ action{:=}`O';\ guard}_Z(\mathbf{S}_i)[\mathbf{action{:=}`A_k';A/A}_k][(\mu\mathbf{A.S})^{n-1}/\mathbf{A}]$
by pruning conditional
$\approx \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_i)[\mathbf{action{:=}`A_k';\ (\mu A.S)}^{n-1}/\mathbf{A}_k]$

Claim: $\mathbf{AS}^n\mathbf{(i)} \approx \mathbf{action{:=}`A_i';\ (\mu A.S)}^n$ for $\mathbf{1{\leqslant}i{\leqslant}p}$.
Proof: by induction on $\mathbf{n}$, trivial for $\mathbf{n{=}0}$.
Suppose the result holds for $\mathbf{n}$,
$\mathbf{AS}^{n+1}\mathbf{(i)} \approx \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_i)[\mathbf{AS}^n\mathbf{(j)}/\mathbf{A}_j|\mathbf{1{\leqslant}j{\leqslant}p}]$ by definition
$\approx \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_i)[\mathbf{action{:=}`A_j';\ (\mu A.S)}^n/\mathbf{A}_j|\mathbf{1{\leqslant}j{\leqslant}p}]$ by induction hyp
$\approx \mathbf{action{:=}`A_i';\ action{:=}`O';\ guard}_Z(\mathbf{S}_i)[\mathbf{action{:=}`A_j';\ (\mu A.S)}^n/\mathbf{A}_j|\mathbf{1{\leqslant}j{\leqslant}p}]$
by adding a redundant assignment.
$\approx \mathbf{action{:=}`A_i';\ (\mu A.S)}^{n+1}$
which proves the claim by induction.

From the premise we have $\mathbf{AS}^n\mathbf{(1)} \leqslant \mathbf{S}'$ for all $\mathbf{n} < \omega$
Thus $\mathbf{action{:=}`A_1';\ (\mu A.S)}^n \leqslant \mathbf{S}'$
So by the general induction rule for recursion we have
$\mathbf{action{:=}`A_1';\ (\mu A.S)} \leqslant \mathbf{S}'$ ie $\mathbf{AS{\leqslant}S}'$ as required.

Identification:

If two actions $\mathbf{X}$ and $\mathbf{Y}$ differ only in that replacing $\mathbf{Y}$ by $\mathbf{X}$ in both definitions gives the same result then we can identify them and replace $\mathbf{Y}$ by $\mathbf{X}$ everywhere and remove $\mathbf{Y}$ from the system.

Adding unnecessary statements to actions which are similar may enable them to be identified: this may make the system less efficient but with a simplified structure. In our development of the Schorr-Waite algorithm we add a single test which enables us to identify all four actions in a system which collapses to a single tail-recursive procedure and thence to an iterative equivalent.

Proof: By re-ordering the actions we may assume $\mathbf{X{=}A}_p$ and $\mathbf{Y{=}A}_m$ where there are $\mathbf{m}$ actions in the original system.
Let $\mathbf{L} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1, \ldots, \mathbf{A}_p \equiv \mathbf{S}_p, \ldots, \mathbf{A}_m \equiv \mathbf{S}_m \rangle$ where $\mathbf{1{\leqslant}p{<}m}$

and $\mathbf{R}$= $\langle \mathbf{A}_1 \equiv \mathbf{S}\mathrm{J}[\mathrm{Ap}/\mathbf{A}_m]$,...,$\mathbf{A}_p \equiv \mathbf{S}_p[\mathbf{A}_p/\mathbf{A}_m]$,...,$\mathbf{A}_{m-1} \equiv \mathbf{S}_{m-1}[\mathbf{A}_p/\mathbf{A}_m]\rangle$
be action systems where $\mathbf{S}_p[\mathbf{A}_p/\mathbf{A}_m] \approx \mathbf{S}_m[\mathbf{A}_p/\mathbf{A}_m]$. We want to prove $\mathbf{L} \approx \mathbf{R}$.
Use the induction rule for action systems:
Claim: $\mathbf{L}^n(\mathbf{k}) \approx \mathbf{R}^n(\mathbf{k})$, $\mathbf{1} \leqslant \mathbf{k} \leqslant \mathbf{m}-\mathbf{1}$ and $\mathbf{L}^n(\mathbf{m}) \approx \mathbf{R}^n(\mathbf{p})$ for $\mathbf{n} < \omega$,
$\mathbf{L}^0(\mathbf{k}) \approx \mathbf{R}^0(\mathbf{k})$ trivially, suppose the result holds for $\mathbf{n}$:
$\mathbf{L}^{n+1}(\mathbf{k}) \approx \mathbf{S}_k[\mathbf{L}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}]$ for $\mathbf{1} \leqslant \mathbf{k} \leqslant \mathbf{m}$.
By induction hypothesis $\mathbf{L}^n(\mathbf{m}) \approx \mathbf{R}^n(\mathbf{p}) \approx \mathbf{L}^n(\mathbf{p})$. So:
$\mathbf{L}^{n+1}(\mathbf{k}) \approx \mathbf{S}_k[\mathbf{L}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}][\mathbf{L}^n(\mathbf{p})/\mathbf{L}^n(\mathbf{m})]$
$\mathbf{L}^{n+1}(\mathbf{k}) \approx \mathbf{S}_k[\mathbf{A}_p/\mathbf{A}_m][\mathbf{L}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}]$
$\approx \mathbf{S}_k[\mathbf{A}_p/\mathbf{A}_m][\mathbf{L}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}-\mathbf{1}]$ since there are no $\mathbf{A}_m$s for the second substitution.
$\approx \mathbf{S}_k[\mathbf{A}_p/\mathbf{A}_m][\mathbf{R}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}-\mathbf{1}]$ by induction hypothesis
$\approx \mathbf{R}^{n+1}(\mathbf{k})$ for $\mathbf{1} \leqslant \mathbf{k} \leqslant \mathbf{m}-\mathbf{1}$
$\mathbf{L}^{n+1}(\mathbf{m}) \approx \mathbf{S}_m[\mathbf{A}_p/\mathbf{A}_m][\mathbf{R}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}-\mathbf{1}]$
$\approx \mathbf{S}_p[\mathbf{A}_p/\mathbf{A}_m][\mathbf{R}^n(\mathbf{j})/\mathbf{A}_j | \mathbf{1} \leqslant \mathbf{j} \leqslant \mathbf{m}-\mathbf{1}]$ by premise
$\approx \mathbf{R}^{n+1}(\mathbf{p})$
which proves the claim.
Hence **beg action:='$\mathbf{A}_1$': $\mathbf{L}^n(\mathbf{1})$ end** $\approx$ **beg action:='$\mathbf{A}_1$': $\mathbf{R}^n(\mathbf{1})$ end** $\leqslant \mathbf{R}$
and $\mathbf{L} \leqslant \mathbf{R}$ by induction rule for action systems.
Similarly **beg action:='$\mathbf{A}_1$': $\mathbf{R}^n(\mathbf{1})$ end** $\approx$ **beg action:='$\mathbf{A}_1$': $\mathbf{L}^n(\mathbf{1})$ end** $\leqslant \mathbf{L}$
and $\mathbf{R} \leqslant \mathbf{L}$. Hence $\mathbf{L} \approx \mathbf{R}$ as required.

Recursion Removal:

Let $\mathbf{X} \equiv \mathbf{S}$ be an action which depends only on $\mathbf{X}$ and $\mathbf{Y}$ (ie the only action calls in $\mathbf{S}$ are to $\mathbf{X}$ or to $\mathbf{Y}$). We want to remove the recursion in $\mathbf{X}$ by enclosing $\mathbf{S}$ in a loop, replacing $\mathbf{X}$ by a statement which causes re-execuion of the loop, replacing $\mathbf{Y}$ by a statement which causes termination of the loop and putting a single $\mathbf{Y}$ after the loop. Thus for example:

$$\mathbf{X} \equiv \mathbf{S} \approx \mathbf{X} \equiv \textbf{do } \mathbf{S}[\textbf{skip, exit / X,Y}] \textbf{ od; } \mathbf{Y}$$

Arsac [Arsac 82], describes this transformation for the case when the action system is regular and all the calls to $\mathbf{X}$ are in terminal positions. For our action systems we are able to prove two generalisations of this recursion removal:

Theorem(A): If the action system **AS** containing $\mathbf{X}$ (as above) is regular then:

$$\mathbf{X} \equiv \mathbf{S} \approx \mathbf{X} \equiv \textbf{do do } \mathbf{S}[\textbf{exit, exit(2) / X,Y}] \textbf{ od od; } \mathbf{Y}$$

Cor: If $\mathbf{X}$ is tail-recursive (ie all the calls to $\mathbf{X}$ in $\mathbf{S}$ are in terminal positions) then:

$$\mathbf{X} \equiv \mathbf{S} \approx \mathbf{X} \equiv \textbf{do } \mathbf{S}[\textbf{skip, exit / X,Y}] \textbf{ od; } \mathbf{Y}$$

Proof of Cor:
If all calls to **X** are terminal then **S[exit, exit(2) / X,Y]** is reducible since the only terminal statements with terminal value one are the **exit**s which replace the calls of **X** and which are in terminal positions. (**S** cannot have a terminal statement with terminal value $>\mathbf{0}$ since it is the body of an action).

$$\mathbf{S[exit,\ exit(2)\ /\ X,Y]{-}1 = S[skip,\ exit\ /\ X,Y]}$$

so by double iteration:

$$\mathbf{do\ do\ S[exit,\ exit(2)\ /\ X,Y]\ od\ od\ \approx\ do\ S[skip,\ exit\ /\ X,Y]\ od}.$$

Theorem(B): If all calls (to **X** and **Y**) in **S** are in terminal positions and all terminal statements of **S**
are action calls (to **X** or **Y**) then:

$$\mathbf{X \equiv S\ \approx\ X \equiv\ do\ S[skip,\ exit\ /\ X,Y]\ od;\ Y}$$

Note: A more general version of Theorem(B) which allows non-terminal calls to **X** or **Y** in **S** will fail since if **S** contains $\mathbf{X;S}_1$ or $\mathbf{Y;S}_1$ then on termination of the call to **X** or **Y** we should execute $\mathbf{S}_1$ but if we replace **X** or **Y** by **exit**s then any statement after them (ie $\mathbf{S}_1$) will never be executed. This does not matter in Theorem(A) because the fact that the system is regular means that an action call can only ever be terminated by a call to **Z** (the terminating action) and this causes immediate termination of the whole system. Thus any statements following an action call in a regular action system will never be executed.

Proof of Theorem B:
Let $\mathbf{A}_q$ represent **X**, $\mathbf{A}_r$ represent **Y**, $\mathbf{S}_q$ represent **S**. We wish to prove the following action systems to be equivalent:

$\mathbf{AS} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1\ \mathbf{,...,}\ \mathbf{A}_q \equiv \mathbf{S}_q\ \mathbf{,...,}\ \mathbf{A}_p \equiv \mathbf{S}_p \rangle$

$\mathbf{AS'} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1\ \mathbf{,...,}\ \mathbf{A}_q \equiv \mathbf{do}\ \mathbf{S}_q[\mathbf{skip,exit/A}_q\mathbf{,A}_r]\ \mathbf{od;}\ \mathbf{A}_r\ \mathbf{,...,}\ \mathbf{A}_p \equiv \mathbf{S}_p \rangle$
where $\mathbf{1 \leqslant q \leqslant p}$ and $\mathbf{1 \leqslant r \leqslant p}$ and $\mathbf{q \neq r}$.

Claim: $\mathbf{AS}^n \leqslant \mathbf{AS'}$.

Proof: By induction on **n**:
For $\mathbf{i \neq q}$ this is easy since:

$\mathbf{AS}^{n+1}\mathbf{(i)}\ \approx\ \mathbf{action{:=}`O';\ S}_i[\mathbf{AS}^n\mathbf{(j)/A}_j] \leqslant \mathbf{action{:=}`O';\ S}_i[\mathbf{AS'(j)/A}_j]\ \approx\ \mathbf{AS'(i)}$ by folding.

Let $\mathbf{S'} = \mathbf{do}\ \mathbf{S}_q[\mathbf{skip,exit/A}_q\mathbf{,A}_r]\ \mathbf{od;}\ \mathbf{A}_r$.

$\mathbf{AS'} \approx\ \mathbf{action{:=}`A_1';\ A}$ **where**
proc **A** ≡
if $\mathbf{action{=}`A_1'} \rightarrow\ \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S}_1)[\mathbf{action{:=}`A}_i\mathbf{';A/A}_i|\mathbf{1 \leqslant i \leqslant p}]$
. . .
□ $\mathbf{action{=}`A}_q\mathbf{'} \rightarrow\ \mathbf{action{:=}`O';\ guard}_Z(\mathbf{S'})[\mathbf{action{:=}`A}_i\mathbf{';A/A}_i|\mathbf{1 \leqslant i \leqslant p}]$
. . .

$\Box$ **action='$\mathbf{A}_p$' $\rightarrow$ action:='O'; guard$_Z$($\mathbf{S}_p$)[action:='$\mathbf{A}_i$';$\mathbf{A}/\mathbf{A}_i|1\leqslant i\leqslant p$] fi.**

All action calls in $\mathbf{S}_q$ are in terminal positions so all the **if** statements introduced by **guard**$_Z$ can be pruned away.

guard$_Z$($\mathbf{S}'$)[action:='$\mathbf{A}_i$';$\mathbf{A}/\mathbf{A}_i|1\leqslant i\leqslant p$]
$\approx$ depth:=1; while depth=1 do
guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od; $\mathbf{A}_r$[action:='$\mathbf{A}_i$';$\mathbf{A}/\mathbf{A}_i|1\leqslant i\leqslant p$]
$\approx$ depth:=1; while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od; action:='$\mathbf{A}_r$';A

Claim(i):
{depth=1}; while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^n; action:='$\mathbf{A}_r$';A
$\leqslant$ {depth=1}; depth:=0; $\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$].
Proof: of Claim(i) by induction on **n**. Trivial for **n=0**. Assume true for **n**:
{depth=1}; while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^{n+1}; action:='$\mathbf{A}_r$';A
$\approx$ {depth=1}; guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]);
if depth=1 then while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^n fi;
action:='$\mathbf{A}_r$';A
$\approx$ {depth=1}; guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]);
if depth=1
then {depth=1}; while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^n;
action:='$\mathbf{A}_r$';A
else action:='$\mathbf{A}_r$';A fi (forward expansion of conditional).
$\approx$ {depth=1}; $\mathbf{S}_q$[skip,depth:=depth$-$1/$\mathbf{A}_q$,$\mathbf{A}_r$]);
if depth=1
then {depth=1}; while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^n;
action:='$\mathbf{A}_r$';A
else action:='$\mathbf{A}_r$';A fi (by removing the **guard** on $\mathbf{S}_q$).
$\approx$ {depth=1}; $\mathbf{S}_q$[if...fi, depth:=depth$-$1; if...fi/$\mathbf{A}_q$,$\mathbf{A}_r$]
by simple absorption since $\mathbf{S}_q$ is regular and $\mathbf{A}_q$,$\mathbf{A}_r$ are the only terminal statements with terminal value zero.
$\approx$ {depth=1}; $\mathbf{S}_q$[{depth=1};
while depth=1 do guard$_1$($\mathbf{S}_q$[skip,exit/$\mathbf{A}_q$,$\mathbf{A}_r$]) od^n;
action:='$\mathbf{A}_r$';A,
depth:=depth$-$1; action:='$\mathbf{A}_r$';A /$\mathbf{A}_q$,$\mathbf{A}_r$] by pruning the **if**s.
$\approx$ {depth=1}; $\mathbf{S}_q$[{depth=1}; depth:=0;
$\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$],
depth:=depth$-$1; action:='$\mathbf{A}_r$';A /$\mathbf{A}_q$,$\mathbf{A}_r$] by ind hyp.

$\approx$ **{depth=1}; depth:=0; $\mathbf{S}_q$[$\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$],**
action:='$\mathbf{A}_r$';A /$\mathbf{A}_q$,$\mathbf{A}_r$]
since these are the only occurrences of **depth**.
$\approx$ **{depth=1}; depth:=0; $\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$]** by action folding.
Which proves Claim(i) by induction.
Hence by the induction rule for loops:
S'[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$]
$\leqslant$ **depth:=1; depth:=0; $\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$]**
$\approx$ **$\mathbf{S}_q$[action:='$\mathbf{A}_q$';A, action:='$\mathbf{A}_r$';A / $\mathbf{A}_q$,$\mathbf{A}_r$]**
since **depth=0** always holds at the beginning of an action.
Hence **AS'** $\leqslant$**AS**.
Conversely:

<u>**Claim(ii):**</u> $\mathbf{AS}^n(\mathbf{i}) \leqslant \mathbf{AS'}^{\,n}(\mathbf{i})$ for $\mathbf{1} \leqslant \mathbf{i} \leqslant \mathbf{p}$.
By induction on **n**:
$\mathbf{AS}^{n+1}(\mathbf{q}) \leqslant$ **depth:=1; <u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip,depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>$^{n+1}$; $\mathbf{AS'}^{\,n}(\mathbf{r})$**.
Since
$\mathbf{AS}^{n+1}(\mathbf{q}) \leqslant \mathbf{S}_q[\mathbf{AS}^n(\mathbf{q}),\mathbf{AS}^n(\mathbf{r})/\mathbf{A}_q,\mathbf{A}_r]$
$\leqslant$ **$\mathbf{S}_q$[depth:=1; <u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>^{n}; $\mathbf{AS'}^{\,n-1}(\mathbf{r})$,**
$\mathbf{AS'}^{\,n}(\mathbf{r})$ / $\mathbf{A}_q$,$\mathbf{A}_r$] by induction hypothesis.
$\approx$ **depth:=1; $\mathbf{S}_q$[<u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>^{n}; $\mathbf{AS'}^{\,n-1}(\mathbf{r})$,**
depth:=0; $\mathbf{AS'}^{\,n}(\mathbf{r})$ / $\mathbf{A}_q$,$\mathbf{A}_r$]
$\leqslant$ **depth:=1; $\mathbf{S}_q$[<u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>^{n}; $\mathbf{AS'}^{\,n}(\mathbf{r})$,**
depth:=0; $\mathbf{AS'}^{\,n}(\mathbf{r})$ / $\mathbf{A}_q$,$\mathbf{A}_r$]
since $\mathbf{AS'}^{\,n-1}(\mathbf{r}) \leqslant \mathbf{AS'}^{\,n}(\mathbf{r})$.
$\approx$ **depth:=1; $\mathbf{S}_q$[<u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>^{n}; $\mathbf{AS'}^{\,n}(\mathbf{r})$,**
depth:=0; <u>while</u> depth=1 <u>do</u>... <u>od</u>^{n}; $\mathbf{AS'}^{\,n}(\mathbf{r})$ / $\mathbf{A}_q$,$\mathbf{A}_r$]
$\approx$ **depth:=1; $\mathbf{S}_q$[skip,depth:=0 / $\mathbf{A}_q$,$\mathbf{A}_r$];**
<u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>^{n}; $\mathbf{AS'}^{\,n}(\mathbf{r})$
by absorption.
$\approx$ **depth:=1; <u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip, depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>$^{n+1}$; $\mathbf{AS'}^{\,n}(\mathbf{r})$**.
Hence $\mathbf{AS}^{n+1}(\mathbf{q}) \leqslant$ **depth:=1; <u>while</u> depth=1 <u>do</u> $\mathbf{S}_q$[skip,depth:=0/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>; $\mathbf{AS'}^{\,n}(\mathbf{r})$**
$\approx$ **depth:=1; <u>while</u> depth=1 <u>do</u> guard$_1$($\mathbf{S}_q$[skip,<u>exit</u>/$\mathbf{A}_q$,$\mathbf{A}_r$]) <u>od</u>; $\mathbf{AS'}^{\,n}(\mathbf{r})$**
$\approx$ **guard$_0$(<u>do</u> $\mathbf{S}_q$[skip,<u>exit</u>/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>); $\mathbf{AS'}^{\,n}(\mathbf{r})$**
$\approx$ **guard$_0$(<u>do</u> $\mathbf{S}_q$[skip,<u>exit</u>/$\mathbf{A}_q$,$\mathbf{A}_r$] <u>od</u>; $\mathbf{A}_r$); [$\mathbf{AS'}^{\,n}(\mathbf{r})$/$\mathbf{A}_r$]**
$\approx$ $\mathbf{AS'}^{\,n+1}(\mathbf{r})$ since $\mathbf{S}_q$ contains calls to $\mathbf{A}_q$ and $\mathbf{A}_r$ only.

Hence $\mathbf{AS}^n\mathbf{(i)} \leqslant \mathbf{AS'}^{\,n}\mathbf{(i)}$ for $\mathbf{1 \leqslant i \leqslant p}$ and $\mathbf{n} < \omega$.

Hence by induction $\mathbf{AS} \leqslant \mathbf{AS'}$ and thus $\mathbf{AS} \approx \mathbf{AS'}$ as required.

Regular Action Systems:

Theorem: Translation to Iterative Form: The regular action system:
$\mathbf{AS} = \langle \mathbf{A}_1 \equiv \mathbf{S}_1, \mathbf{A}_2 \equiv \mathbf{S}_2, \ldots, \mathbf{A}_n \equiv \mathbf{S}_n \rangle$ with terminating action $\mathbf{Z}$ is equivalent to the iterative statement:

$\mathbf{AS'}$ = **action:=‘$\mathbf{A}_1$’;**
do do if action=‘$\mathbf{A}_1$’ → $\mathbf{S}_1$[action:=‘$\mathbf{A}_i$’;exit / $\mathbf{A}_i$|1⩽i⩽n][exit(2)/Z]
□ action=‘$\mathbf{A}_2$’ → $\mathbf{S}_2$[action:=‘$\mathbf{A}_i$’;exit / $\mathbf{A}_i$|1⩽i⩽n][exit(2)/Z]
□ . . .
□ action=‘$\mathbf{A}_n$’ → $\mathbf{S}_n$[action:=‘$\mathbf{A}_i$’;exit / $\mathbf{A}_i$|1⩽i⩽n][exit(2)/Z] fi od od.

Proof:
Let **IF** be: **if action:=‘$\mathbf{A}_1$’ → $\mathbf{S}_1$**
□ action:=‘$\mathbf{A}_2$’ → $\mathbf{S}_2$
□ . . . fi
Then by the definitional transformation:
AS ≈ action:=‘$\mathbf{A}_1$’; A where
proc A ≡ action:=‘O’; guard$_Z$(IF[action:=‘$\mathbf{A}_i$’;A / $\mathbf{A}_i$|1⩽i⩽n][action:=‘Z’ / Z].
We want to prove that this is equivalent to:
action:=‘$\mathbf{A}_1$’; do do IF[action:=‘$\mathbf{A}_i$’;exit / $\mathbf{A}_i$|1⩽i⩽n][exit(2) / Z] od od
Note that we don’t care about the final value of **action**.
Hence we need to prove that the procedure:
proc A ≡ action:=‘O’; guard$_Z$(IF[action:=‘$\mathbf{A}_i$’;A / $\mathbf{A}_i$|1⩽i⩽n][action:=‘Z’ / Z].
is equivalent to the double loop:
do do IF[action:=‘$\mathbf{A}_i$’;exit / $\mathbf{A}_i$|1⩽i⩽n][exit(2) / Z] od od
The proof uses the following lemma:

Lemma: Induction rule for double iteration.
For the double loop **do do S od od** we define the **n**th truncation by:
do do S od od^0 = abort
do do S od od^{n+1} = action:=‘O’; guard$_Z$(S-(2,3))[do do S od od^n/(n,T)|τ =0 ∨ τ =1]
[action:=‘Z’/(n,T)|τ =2]
where **action** is a new variable.
Then if **do do S od od^n ⩽ S'** for all $\mathbf{n} < \omega$ then **do do S od od ⩽ S'**.

Proof: The **S-(2,3)** will reduce all the terminal statements with terminal value **>0** in the original loop by two (to compensate for the fact that they have been moved out of two loops), the $\mathbf{guard}_Z$ is to ensure that terminal statements with terminal value **2** are dealt with correctly and terminal statements of **S** which do not terminate the loop are replaced by the next lower truncation.

A rigorous proof involves translating the double loop to a double **while** loop (using the definitional transformations), transforming the double **while** loop to a single loop using lemma B in the previous Chapter, and showing that the **n**th truncations of this **while** loop are equivalent to **do do S od od**^{n}. The details are omitted for brevity.

We can easily show that the **n**th truncation:

proc A ≡ action:='O'; guard$_Z$**(IF[action:='A**$_i$**';A / A**$_i$**|1⩽i⩽n][action:='Z' / Z].**^{n}

is equivalent to:

do do IF[action:='A$_i$**';exit / A**$_i$**|1⩽i⩽n][exit(2) / Z] od od**

by induction on **n** since the only terminal statements of **IF** are action calls (because the original action system was regular).
Hence the only terminal statements of **IF[action:='A**$_i$**';exit / A**$_i$**|1⩽i⩽n][exit(2) / Z]** with terminal value zero or one are the **exit**s which arise from the substitution of action calls.

This Lemma then completes the proof of the theorem. We can use this theorem to prove Theorem (A):

Proof of Theorem(A):
Recall that the action system is regular and $\mathbf{S}_q$ contains calls to $\mathbf{A}_q$ and $\mathbf{A}_r$ only.
By the previous theorem, since **AS** is regular:

AS ≈ action:='A$_1$**';**
do do if action='A$_1$**' → S**$_1$**[action:='A**$_i$**'; exit/A**$_i$**|1⩽i⩽p]**
. . .
□ **action='A**$_q$**' → S**$_q$**[action:='A**$_i$**'; exit/A**$_i$**|1⩽i⩽p]**
. . .
□ **action='A**$_p$**' → exit(2) fi od od**

where $\mathbf{A}_q$ represents **X**, $\mathbf{A}_r$ represents **Y** (**1⩽q<p**, **1⩽r⩽p**, **q≠r**) and $\mathbf{A}_p$ is the unique terminating action.
From the premises we deduce that all the terminal statements in $\mathbf{S}_q$ are of the form:

action:='A$_q$**'; exit** or **action:='A**$_r$**'; exit**.

This is a double-nested loop so we can use selective unrolling on selected terminal statements with terminal value one. We will unroll on the statements **action:='A**$_q$**'; exit** and use **action='A**$_q$**'** as the extra condition on the inserted loop. The body of the inserted loop can then be simplified to give (for the **q**th condition):

□ **action=‘$\mathbf{A}_q$’**
→ **$\mathbf{S}_q$[action:=‘$\mathbf{A}_r$’; exit /$\mathbf{A}_r$]**
[action:=‘$\mathbf{A}_q$’; do do if action=‘$\mathbf{A}_q$’
then $\mathbf{S}_q$[action:=‘$\mathbf{A}_r$’; exit /$\mathbf{A}_r$][action:=‘$\mathbf{A}_q$’; exit /$\mathbf{A}_q$]
else exit(2) fi od od]

The **else exit(2)** can be replaced by **else exit(3)** since the inserted loop is in a terminal position in the body of the outer double loop. Use selective unrolling on the inserted loop to replace **action:=‘$\mathbf{A}_r$’; exit** by **action:=‘$\mathbf{A}_r$’; exit(3)**. Now the predicate **action=‘$\mathbf{A}_q$’** is preserved by the loop body and is true initially so we can remove the test and remove the statement **action:=‘$\mathbf{A}_q$’** replacing $\mathbf{A}_q$.

□ **action=‘$\mathbf{A}_q$’**
→ **$\mathbf{S}_q$[action:=‘$\mathbf{A}_r$’; exit /$\mathbf{A}_r$]**
[action:=‘$\mathbf{A}_q$’; do do $\mathbf{S}_q$[action:=‘$\mathbf{A}_r$’; exit(3) /$\mathbf{A}_r$][exit /$\mathbf{A}_q$] od od]

Loop rolling gives:

□ **action=‘$\mathbf{A}_q$’**
→ **do do $\mathbf{S}_q$[action:=‘$\mathbf{A}_r$’; exit(3) /$\mathbf{A}_r$][exit /$\mathbf{A}_q$] od od**

Finally we move the statement **action:=‘$\mathbf{A}_r$’** outside the loop:

□ **action=‘$\mathbf{A}_q$’**
→ **do do $\mathbf{S}_q$[exit(2) /$\mathbf{A}_r$][exit /$\mathbf{A}_q$] od od; action:=‘$\mathbf{A}_r$’**

We can add an **exit** to the end of this statement (since it is at the end of the body of a double loop), then translating the result back to a regular action system completes the proof.