) CHAPTER THREE

= Transformations Involving exiE Statements

Introduction

In this chapter we build on the results of the last chapter to develop a set of transformations of programs which use unbounded loops and **exit** statements. These transformations will allow us to work directly with the programs without having to use the definitional transformations to prove the equivalence of two statements. In this chapter we consider statements to be compounds of “primitive statements” (which are either **exit** statements, or statements which cannot be terminated by the execution of an **exit** statement within them). These correspond loosely to the “basic blocks” of compiler terminology. All the transformations in this section treat primitive statements as atomic, indivisible units.

The1complexity of interpreting **do** loops ks a consequence of the “power” of such control structures; indeed they can be used to simulate any other form f control structure including arbitrary transfers of control. [Bohm & Jacopini 66] showed that any program using any control structures can be transformed to a functionally equivalent one which uses only composition, **if** **then** **else**, and **while** **do** control structures. This has been described as “perhaps the first major result of structured programming” [Ledgard & Marcotty 75].

However if one restricts the transformations to those which increase neither program length nor execution time then these basic control structures are not sufficient. S.R.Kosaraju in [Kosaraju 74] proved that the addition of infinite loops of arbitrary depth and **exit(n)** statements, where **n** can be any integer, is sufficient.

These statements are used by Arsac in [Arsac 79] and [Arsac 82] where several program transformations are developed for recursion removal. [Taylor 84] also discusses them and propounds their inclusion in programming languages as an alternative to current looping syntax.

The following transformations were inspired by Arsac’s work in this area, we have generalised these transformations and proved them within our programming system (which unlike Arsac’s allows nondeterministic programs and specifications as programs as well as a much wider range of control structures):
Incrementation (but not partial incrementation),
Simple Absorption, and a simple converse,
False Iteration, and a simple converse,
Inversion and Proper Inversion,
Loop Doubling,
First Step Unrolling,
Double Iteration,

As well as generalising these transformations, we have added the various "selective unrolling" and "selective unfolding" transformations which (as we shall demonstrate) cannot be derived from Arsac's set of transformations.

The rest of this section will concentrate on our proofs of these transformations together with the proofs of some lemmas which will be needed in the next chapter. First we give some notation:

Defn: Since an **exit** statement is not allowed to leave a block or a recursive procedure or loop (other than a **do** loop) we can regard such statements as single indivisible wholes as far as this section is concerned. Hence we define a primitive statement to be either an **exit** statement, an assignment, an assertion a block, a recursive procedure, a **while** loop, a **for** loop or a nondeterministic iteration. Thus a primitive statement cannot be terminated by the execution of an **exit** statement (unless it is an **exit** statement). All other statements are compound statements.

Primitive Statements: eg:
Exit Statements **exit(k)**
Assignments **x:=x′.Q**
Assertions **{Q}**
Blocks **beg x:S end**
Recursive Procedures **proc X ≡ S**
Counted Repetition **for i:=b step s to f do S od**
Deterministic Iteration **while B do S od**
Nondeterministic iteration **do** $\mathbf{B}_1 \rightarrow \mathbf{S}_1 \square \ldots \square \mathbf{B}_n \rightarrow \mathbf{S}_n$ **od**
Abstraction **rep x/y.Q:S per** and **beg x/y.Q:S per**

Compound Statements: eg:
Composition $\mathbf{S}_1$;$\mathbf{S}_2$
Deterministic Selection **if B then** $\mathbf{S}_1$ **else** $\mathbf{S}_2$ **fi**
Nondeterministic Selection **if** $\mathbf{B}_1 \rightarrow \mathbf{S}_1 \square \ldots \square \mathbf{B}_n \rightarrow \mathbf{S}_n$ **fi**
Unbounded loops **do S od**
Nondeterministic Choice **oneof** $\mathbf{S}_1 \vee \mathbf{S}_2$ **foeno**

Defn: For any statements **T, S** where **T** is primitive the function **occ(T,S)** (the number of occurrences of **T** in **S**) is defined as follows:
If **T=S** then **occ(T,S)=1**.

Otherwise if **S** is primitive then **occ(T,S)=0**, (since in this section we regard all primitive statements as indivisible, so occurrences of a statement within a primitive statement are not counted).

Otherwise $\mathbf{occ(T,(S_1;S_2)) = occ(T,S_1)+occ(T,S_2)}$

$\mathbf{occ(T,\ \underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi}) = occ(T,S_1)+occ(T,S_2)}$

$\mathbf{occ(T,\ \underline{oneof}\ \ S_1 \vee S_2\ \underline{foeno}) = occ(T,S_1)+occ(T,S_2)}$

$\mathbf{occ(T,\underline{if}\ B_1\ \rightarrow S_1 \square \ldots \square B_n\ \rightarrow S_n\ \underline{fi})} = \sum_{1\leqslant i\leqslant n}\mathbf{occ(T,S}_i\mathbf{)}$

$\mathbf{occ(T,\underline{do}\ S_1\ \underline{od}) = occ(T,S_1)}$

We will use the following simple program to illustrate the transformations developed in this section. The program sets **r** to the sum of all integers from **1** to **n**:

```
Prog= r:=0; i:=1;
  do  if i>n then exit fi;
      t:=i;
      do   r:=r+1; t:=t−1;
        if t=0 then exit fi od;
      i:=i+1 od.
```

This program has primitive statements: **r:=0**, **i:=1**, **t:=i**, **r:=r+1**, **t:=t−1**, **i:=i+1** each of which occurs once, and **exit** which occurs twice.

Defn: For any statements **T,S** (where **T** is primitive) and integers **n,d** the predicate **ts(n,T,S,d)** (which is interpreted "the **n**th occurrence of **T** in **S** is a terminal statement of **S** which will leave **d** enclosing loops") is defined recursively as follows:

If **S** is a primitive non-**exit** statement then $\mathbf{ts(n,T,S,d)} \iff \mathbf{n=1} \wedge \mathbf{d=0} \wedge \mathbf{T=S}$

$\mathbf{ts(n,T,\underline{exit}(k),d)} \iff \mathbf{n=1} \wedge \mathbf{d=k} \wedge \mathbf{T=\underline{exit}(k)}$

$\mathbf{ts(n,T,(S_1;S_2),d)} \iff \mathbf{ts(n,T,S_1,d)} \vee \mathbf{ts(n\text{-}occ(T,S_1),T,S_2,d)}$ if $\mathbf{d>0}$

$\iff \mathbf{ts(n\text{-}occ(T,S_1),T,S_2,d)}$ if $\mathbf{d=0}$

$\mathbf{ts(n,T,\underline{if}\ B\ \underline{then}\ S_1\ \underline{else}\ S_2\ \underline{fi},\ d)} \iff \mathbf{ts(n,T,S_1,d)} \vee \mathbf{ts(n\text{-}occ(T,S_1),T,S_2,d)}$

$\mathbf{ts(n,T,\underline{if}\ B_1\ \rightarrow S_1 \square \ldots \square B_n\ \rightarrow S_n\ \underline{fi},\ d)} \iff \bigvee_{1\leqslant i\leqslant n}\mathbf{ts(n\text{-}}\sum_{1\leqslant j<i}\mathbf{occ(T,S}_j\mathbf{),T,S}_i\mathbf{,d)}$

$\mathbf{ts(n,T,\underline{do}\ S_1\ \underline{od},\ d)} \iff \mathbf{ts(n,T,S_1,d+1)}$

With our example the only terminal statement of **Prog** is the first occurrence of **exit**, which will leave zero enclosing **do** loops thus:

$\mathbf{ts(1,\underline{exit},Prog,0)} \iff \mathbf{true}$ while $\mathbf{ts(2,\underline{exit},Prog,0)} \iff \mathbf{false}$.

We write $\mathbf{ts(n,T,S)}$ for $\mathbf{ts(n,T,S,0)}$ and write $\mathbf{ts(S)}$ for $\{\mathbf{T}|\exists\mathbf{n.ts(n,T,S)}\}$ and $\mathbf{ts(S,d)}$ for $\{\mathbf{T}|\exists\mathbf{n.ts(n,T,S,d)}\}$. Thus $\mathbf{ts(Prog)}=\{\underline{\mathbf{exit}}\}$.

We say **T** is a terminal statement of **S**, or **T** is terminal in **S** if $\exists\mathbf{n.ts(n,T,S)}$.

<u>Theorem:</u> If $\mathbf{ts(S)}=\{\}$ then $\vdash \mathbf{S} \approx \mathbf{abort}$.
ie if **S** has no terminal statement then **S** cannot terminate.
Proof: By induction on **n** and on the structure of **S** prove:
If $\mathbf{ts(S,n)}=\{\}$ then $\vdash\mathbf{depth{:=}n;\ guard}_n\mathbf{(S);\ \{depth}\leqslant\mathbf{0\}} \approx \mathbf{abort}$

<u>Defn:</u> If $\mathbf{occ(T,S)}\geqslant\mathbf{n}\geqslant\mathbf{1}$ then the depth of the **n**th occurrence of the primitive statement **T** in **S**, called $\delta\mathbf{(n,T,S)}$, is defined:
If **T**=**S** then $\delta\mathbf{(1,T,S)=0}$
$\delta\mathbf{(n,T,(S_1;S_2))} = \delta\mathbf{(n,T,S_1)}$ if $\mathbf{occ(T,S_1)}\geqslant\mathbf{n}$
$\qquad\delta\mathbf{(n-occ(T,S_1),T,S_2)}$ otherwise
δ(**n**,**T**,**<u>oneof</u>** $\mathbf{S_1} \vee \mathbf{S_2}$ **<u>foeno</u>**) =
δ(**n**,**T**,**<u>if</u>** **B** **<u>then</u>** $\mathbf{S_1}$ **<u>else</u>** $\mathbf{S_2}$ **<u>fi</u>**) = $\delta\mathbf{(n,T,S_1)}$ if $\mathbf{occ(T,S_1)}\geqslant\mathbf{n}$
$\qquad\delta\mathbf{(n-occ(T,S_1),T,S_2)}$ otherwise
δ(**n**,**T**,**<u>if</u>** $\mathbf{B_1} \to \mathbf{S_1} \Box \ldots \Box \mathbf{B}_n \to \mathbf{S}_n$ **<u>fi</u>**) = $\delta\mathbf{(n\text{-}\sum_{1\leqslant j<i}occ(T,S}_j\mathbf{),T,S}_i\mathbf{)}$ where **i** is the smallest integer such that $\mathbf{occ(T,S}_i\mathbf{)}\geqslant\mathbf{n\text{-}\sum_{1\leqslant j<i}occ(T,S}_j\mathbf{)}$
δ(**n**,**T**,**<u>do</u>** $\mathbf{S_1}$ **<u>od</u>**) = $\delta\mathbf{(n,T,S_1)+1}$

<u>Defn:</u> If **T** is primitive and $\mathbf{ts(n,T,S)}$ holds then the <u>terminal</u> <u>value</u> of the **n**th occurrence of **T** in **S**, $\tau\mathbf{(n,T,S)}$, is defined as $|\mathbf{T}| - \delta\mathbf{(n,T,S)}$ where $|\underline{\mathbf{exit}}\mathbf{(k)}| = \mathbf{k}$ and $|\mathbf{T}| = \mathbf{0}$ for all other primitive statements.
Note that if **T** is terminal then we must have $\tau\mathbf{(n,T,S)}\geqslant\mathbf{0}$.

The terminal value of a statement is the number of enclosing loops which would be terminated by the execution of that statement. This means that the next statement to be executed after the **n**th occurrence of **T** in **S** will be the first statement outside $\tau\mathbf{(n,T,S)}$ loops.
For example: $\delta(\mathbf{1},\underline{\mathbf{exit}},\mathbf{Prog})=\mathbf{1}$, $\delta(\mathbf{2},\underline{\mathbf{exit}},\mathbf{Prog})=\mathbf{2}$, $\delta\mathbf{(1,i{:=}1,Prog)=0}$ etc.
$\tau(\mathbf{1},\underline{\mathbf{exit}},\mathbf{Prog}) = |\underline{\mathbf{exit}}| - \delta(\mathbf{1},\underline{\mathbf{exit}},\mathbf{Prog}) = \mathbf{1-1} = \mathbf{0}$.
If $\tau\mathbf{(n,T,S)=0}$ then the next statement executed after the execution of **T** in **S** will be the statement immediately following **S**. If $\tau\mathbf{(n,T,S)>0}$ then the execution of **T** in **S** will cause the termination of the first τ **<u>do</u>** loops enclosing **S**.

It often occurs that we wish to substitute certain occurrences of primitive statements by other statements, and do all the substitutions simultaneously. Such a substitution is defined as follows:

<u>Defn:</u> Global Substitution: If $\mathbf{P(n,T,S)}$ is a predicate on $\mathbf{n,T}$ and $\mathbf{S}$, and $\mathbf{S'(n,T,S)}$ is a statement for any $\mathbf{n,T}$ and $\mathbf{S}$ then the effect of replacing the $\mathbf{n}$th occurrence of the primitive statement $\mathbf{T}$ in $\mathbf{S}$ by $\mathbf{S'(n,T,S)}$ for every $\mathbf{n}$ and $\mathbf{T}$ such that $\mathbf{P(n,T,S)}$ holds is denoted:

$$\mathbf{S[S'(n,T,S)/(n,T)|P(n,T,S)]}$$

For any statement $\mathbf{S''}$, statement function $\mathbf{S'}$, integer $\mathbf{k}$ and predicate $\mathbf{P}$:
$\mathbf{S''[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$ is defined:

(i) $\mathbf{S''[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad = \mathbf{S''}$ if $\mathbf{S''}$ is primitive and $\mathbf{P(k{+}1,S'',S)}$ fails.

(ii) $\mathbf{S''[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad = \mathbf{S'(k{+}1,S'',S)}$ if $\mathbf{S''}$ is primitive and $\mathbf{P(k{+}1,S'',S)}$ holds.

(iii) $\mathbf{(S_1;S_2)[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad = \mathbf{S_1[S'(n,T,S)/(n{-}k,T)|P(n,T,S)\wedge]\ ;}$
$\quad\ \mathbf{S_2[S'(n,T,S)/(n{-}k{-}occ(T,S_1),T)|P(n,T,S)]}$

(iv) $\underline{\mathbf{if}}$ $\mathbf{B}$ $\underline{\mathbf{then}}$ $\mathbf{S_1}$ $\underline{\mathbf{else}}$ $\mathbf{S_2}$ $\underline{\mathbf{fi}}$, similar to (iii).

(v) $\mathbf{(\underline{if}\ B_1 \ \to S_1 \ \square \ ... \ \square \ B_m \ \to S_m \ \underline{fi})[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad = \mathbf{\underline{if}\ B_1 \ \to S_1[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad \square \ \mathbf{...}$
$\quad \kappa\kappa\kappa\kappa \ \square \ \mathbf{B_i \ \to S_i[S'(n,T,S)/(n{-}k{-}\sum_{1\leqslant j<i} occ(T,S_j),T)|P(n,T,S)] \ \square \ ...\underline{fi}}$

(vi) $\mathbf{(\underline{do}\ S_1\ \underline{od})[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]}$
$\quad = \mathbf{\underline{do}\ S_1[S'(n,T,S)/(n{-}k,T)|P(n,T,S)]\ \underline{od}}$

We will often abbreviate $\delta\mathbf{(n,T,S)}$ by δ, $\mathbf{ts(n,T,S)}\wedge\mathbf{R}(\tau\mathbf{(n,T,S)})$ by $\mathbf{R}(\tau)$ (where $\mathbf{R}$ is a relation on integers) and $\mathbf{(n,T)}$ by $\mathbf{T}$ so for example:

$$\mathbf{S[S'(n,T,S)/(n,T)|ts(n,T,S)}\wedge\ \tau\mathbf{(n,T,S)=0]} \text{ becomes } \mathbf{S[S'(n,T,S)/T|}\tau\ \mathbf{=0]}$$

Such substitutions will be used extensively in the rest of this section to prove properties of statements and transformations of them by induction on their structure.

Note that for $\mathbf{n}_0 < \omega$ the statement $\mathbf{S[S'/(n,T)|n{=}n_0}\ \wedge\ \mathbf{T{=}T_0]}$ is $\mathbf{S}$ with $\mathbf{S'}$ replacing the $\mathbf{n}_0$th occurrence of $\mathbf{T}_0$. We abbreviate this to $\mathbf{S[S'/(n_0,T_0)]}$.

<u>Defn:</u> Incrementation:
An example of global substitution is incrementation: if $\mathbf{S}$ is a statement and $\mathbf{k}$ an integer then $\mathbf{S{+}k}$

denotes the substitution of all terminal statements **<u>exit</u>(m)** by **<u>exit</u>(m+k)** and all other primitive terminal statements **T** by **T;<u>exit</u>(k)**. ie:

$$\mathbf{S{+}k} =_{DF} \mathbf{S[T{+}k/(n,T)|}\tau\mathbf{(n,T,S)}\geqslant\mathbf{0]} = \mathbf{S[T{+}k/T|}\tau \geqslant\mathbf{0]}$$

Thus **Prog+1** is identical to **Prog** except that the first occurrence of **<u>exit</u>** is replaced by **<u>exit</u>(2)**. Note that the second occurrence of **<u>exit</u>** is not a terminal statement of **Proc** so is not incremented.

<u>Example:</u> If **P** is
<u>if</u> B <u>then</u> <u>exit</u> <u>fi</u>;
<u>do</u> <u>do</u> a; <u>if</u> C <u>then</u> <u>exit</u> <u>fi</u> <u>od</u>;
  b; <u>if</u> D <u>then</u> <u>exit</u>(2) <u>fi</u>; <u>od</u>; c.

where **a,b** and **c** are primitive then **P+3** is
<u>if</u> B <u>then</u> <u>exit</u>(4) <u>fi</u>;
<u>do</u> <u>do</u> a; <u>if</u> C <u>then</u> <u>exit</u> <u>fi</u> <u>od</u>;
  b; <u>if</u> D <u>then</u> <u>exit</u>(5) <u>fi</u>; <u>od</u>; c; <u>exit</u>(3).

<u>Defn:</u> Partial Incrementation:
This is similar to incrementation except that only those terminal statements which have a terminal value greater than some given value are incremented.

$$\mathbf{S{+}(k,d)} =_{DF} \mathbf{S[T{+}k/(n,T)|ts(n,T,S)} \wedge \tau\mathbf{(n,T,S)}\geqslant\mathbf{d]}$$

For example:
<u>do</u> S <u>od</u>+(k,d) = <u>do</u> S[T+k/(n,T)|ts(n,T,S) $\wedge$ τ**(n,T,S)⩾d+1] <u>od</u>**
  = <u>do</u> S+(k,d+1) <u>od</u>

Clearly **S+(k,0) = S+k**.
Also $(\mathbf{S_1;S_2})\mathbf{+(k,d)} = \mathbf{S_1{+}(k,d);\ S_2{+}(k,d)}$ if $\mathbf{d>0}$
  $= \mathbf{S_1{+}(k,1);\ S_2{+}(k,d)}$ if $\mathbf{d{=}0}$.
This is because terminal statements of $\mathbf{S_1}$ with terminal value zero lead to the execution of $\mathbf{S_2}$ so are not terminal statements of $\mathbf{S_1;S_2}$ but those with terminal value >**0** <u>are</u> terminal statements of $\mathbf{S_1;S_2}$.
For the other compounds we can apply the **+(k,d)** directly to each component.
For primitive non-**<u>exit</u>** statements **S+(k,d) = S; <u>exit</u>(k)**.
For **<u>exit</u>** statements: **<u>exit</u>(l)+(k,d) = <u>exit</u>(l+k)**.

Using the example above, **P+(3,1)** is:

if B **then** **exit** **fi**;
do **do** a; **if** C **then** **exit** **fi** **od**;
  b; **if** D **then** **exit**(5) **fi**; **od**; c.

For most of our substitutions we will not want to replace the non-**exit** primitive statements but only to add another statement after them. This is to ensure that adding **exit**(0) (ie **skip**) after a non-**exit** primitive does not affect our substitutions. For example the definition of simple absorption is:

$$\mathbf{S;S'} \ \approx\ \mathbf{S[S'}+\delta/\mathbf{T}|\tau=\mathbf{0]}$$

which should be interpreted as:

$$\mathbf{S;S'} \ \approx\ \mathbf{S[S'(n,T,S)/(n,T)|ts(n,T,S)} \ \wedge\ \tau\mathbf{(n,T,S)=0]}$$

where $\mathbf{S'(n,T,S)} = \mathbf{T};\ \mathbf{S'}+\delta\mathbf{(n,T,S)}$ if **T** is a non-**exit** primitive
$= \mathbf{S'}+\delta\mathbf{(n,T,S)}$ otherwise.

TRANSFORMATIONS:

Lemma A: For all integers **d**,**k**
$\Delta \vdash$ **{depth=k+l}; guard**$_{k+l}$**(S+(k,l))**
  $\approx$ **{depth=k+l}; guard**$_{k+l}$**(S); if depth$\leqslant$k then depth:=depth$-$k fi**
This says that if after executing **S** the depth is $\leqslant$**k** then the last terminal statement executed corresponds to a one of the statements of **S+(k,l)** which has been incremented by **l**. To get the same effect we execute **depth:=depth$-$k** in this case. If the depth is >**k** then the corresponding terminal statement of **S+(k,l)** would not be incremented so we do nothing. This lemma will be used in the proofs of several important transformations.

Proof: By induction on the structure of **S**:
Let **IF** = **if depth$\leqslant$k then depth:=depth$-$k fi**
Note that **{depth=k+l}; depth:=depth$-$n** $\approx$ **{depth=k+l}; guard**$_{k+l}$**(exit(n))**

Case (i): **S** is an **exit** statement and **l**$\leqslant$ |**S**| -say **S** is **exit(n)** with **n**$\geqslant$**l**.
{depth=k+l}; guard$_{k+l}$**(S+(k,l))** $\approx$ **{depth=k+l}; depth:=depth$-$(n+k)**
  $\approx$ **{depth=k+l}; depth:=depth$-$n; if depth$\leqslant$k then depth:=depth$-$k fi**

Case (ii): **S** is an **exit** statement and **l**> |**S**| -say **S** is **exit(n)** with **n**<**l**.

$\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}+(\mathbf{k},\mathbf{l}))$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{depth{:=}depth{-}n}$ since the terminal value $\mathbf{n}$ is $<\mathbf{l}$.

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{depth{:=}depth{-}n}$; $\{\mathbf{depth{>}k}\}$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{depth{:=}depth{-}n}$; $\underline{\mathbf{if}}$ $\mathbf{depth \leqslant k}$ $\underline{\mathbf{then}}$ $\mathbf{depth{:=}depth{-}k}$ $\underline{\mathbf{fi}}$

Case (iii): $\mathbf{S}$ is a non-$\underline{\mathbf{exit}}$ primitive and $\mathbf{l{=}0}$. Thus $\mathbf{S}$ does not change **depth**.

$\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}+(\mathbf{k},\mathbf{l}))$ $\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{S}$; $\mathbf{depth{:=}depth{-}k}$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{S}$; $\underline{\mathbf{if}}$ $\mathbf{depth \leqslant k}$ $\underline{\mathbf{then}}$ $\mathbf{depth{:=}depth{-}k}$ $\underline{\mathbf{fi}}$

Case (iv): $\mathbf{S}$ is a non-$\underline{\mathbf{exit}}$ primitive and $\mathbf{l{>}0}$. Thus $\mathbf{S}$ does not change **depth** and the incrementation does not take place.

$\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}+(\mathbf{k},\mathbf{l}))$ $\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{S}$; $\{\mathbf{depth{>}k}\}$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{S}$; $\underline{\mathbf{if}}$ $\mathbf{depth \leqslant k}$ $\underline{\mathbf{then}}$ $\mathbf{depth{:=}depth{-}k}$ $\underline{\mathbf{fi}}$

Case (v): $\mathbf{S{=}S_1{;}S_2}$ and $\mathbf{l{=}0}$.

$\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}((\mathbf{S}_1;\mathbf{S}_2)+(\mathbf{k},\mathbf{l}))$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1+(\mathbf{k},\mathbf{1}))$; $\mathbf{guard}_{k+l}(\mathbf{S}_2+\mathbf{k})$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$; $\mathbf{IF}$;

$\underline{\mathbf{if}}$ $\mathbf{depth{=}k{+}l}$ $\underline{\mathbf{then}}$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_2+\mathbf{k})$ $\underline{\mathbf{fi}}$

by induction hypothesis and properties of **guard**.

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$;

$\underline{\mathbf{if}}$ $\mathbf{depth \leqslant k}$ $\underline{\mathbf{then}}$ $\mathbf{depth{:=}depth{-}k}$; $\{\mathbf{depth \leqslant 0}\}$ $\underline{\mathbf{fi}}$;

$\underline{\mathbf{if}}$ $\mathbf{depth{=}k{+}l}$ $\underline{\mathbf{then}}$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_2)$; $\mathbf{IF}$ $\underline{\mathbf{fi}}$

by induction hypothesis again.

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$;

$\underline{\mathbf{if}}$ $\mathbf{depth \leqslant k}$

$\underline{\mathbf{then}}$ $\mathbf{depth{:=}depth{-}k}$

$\underline{\mathbf{else}}$ $\underline{\mathbf{if}}$ $\mathbf{depth{=}k{+}l}$ $\underline{\mathbf{then}}$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_2)$; $\mathbf{IF}$ $\underline{\mathbf{fi}}$ $\underline{\mathbf{fi}}$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$;

$\underline{\mathbf{if}}$ $\mathbf{depth{=}k{+}l}$ $\underline{\mathbf{then}}$ $\mathbf{guard}_{k+l}(\mathbf{S}_2)$ $\underline{\mathbf{fi}}$; $\mathbf{IF}$

by forward contraction of $\underline{\mathbf{if}}$.

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$; $\mathbf{guard}_{k+l}(\mathbf{S}_2)$; $\mathbf{IF}$

$\approx$ $\{\mathbf{depth{=}k{+}l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1;\mathbf{S}_2)$; $\mathbf{IF}$

Case (vi): $\mathbf{S}=\mathbf{S}_1;\mathbf{S}_2$ and $\mathbf{l}>\mathbf{0}$.

$\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}((\mathbf{S}_1;\mathbf{S}_2)+(\mathbf{k,l}))$

$\approx$ $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1+(\mathbf{k,l}))$; $\mathbf{guard}_{k+l}(\mathbf{S}_2+(\mathbf{k,l}))$

$\approx$ $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1)$; **IF;**

<u>if</u> depth=k+l <u>then</u> $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_2+(\mathbf{k,l}))$ **<u>fi</u>**

by induction hypothesis and properties of **guard**.

$\approx$ $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}(\mathbf{S}_1;\mathbf{S}_2)$; **IF**

as for last case.

Case (vii): **S= <u>if</u>** $\mathbf{B}_1 \rightarrow \mathbf{S}_1 \,\square\, \ldots \,\square\, \mathbf{B}_n \rightarrow \mathbf{S}_n$ **<u>fi</u>**

Use case analysis on $\mathbf{B}_1,\ldots,\mathbf{B}_n$ and the induction hypothesis on $\mathbf{S}_1,\ldots,\mathbf{S}_n$ to get:

$\{\mathbf{depth=k+l}\}$; **<u>if</u>** $\mathbf{B}_1 \rightarrow \mathbf{guard}_{k+l}(\mathbf{S}_1)$; **IF**

$\square$ ...

$\square$ $\mathbf{B}_n \rightarrow \mathbf{guard}_{k+l}(\mathbf{S}_n)$; **IF <u>fi</u>**

Then backward and forward expansion of the outer **<u>if</u>** gives the result.

Case (viii): **S=<u>do</u>** $\mathbf{S}_1$ **<u>od</u>**.

$\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}($**<u>do</u>** $\mathbf{S}_1$ **<u>od</u>**$+(\mathbf{k,l}))$

$\approx$ $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}($**<u>do</u>** $\mathbf{S}_1+(\mathbf{k,l+1})$ **<u>od</u>**$)$ by a previous result

$\approx$ $\{\mathbf{depth=k+l}\}$; **depth:=depth+1;**

<u>while</u> depth=k+l+1 <u>do</u> $\{\mathbf{depth=k+l+1}\}$; $\mathbf{guard}_{k+l+1}(\mathbf{S}_1+(\mathbf{k,l+1}))$ **<u>od</u>**

$\approx$ $\{\mathbf{depth=k+l}\}$; **depth:=depth+1;**

<u>while</u> depth=k+l+1 <u>do</u> $\{\mathbf{depth=k+l+1}\}$; $\mathbf{guard}_{k+l+1}(\mathbf{S}_1)$; **IF <u>od</u>**

by induction hypothesis.

$\approx$ $\{\mathbf{depth=k+l}\}$; **depth:=depth+1;**

<u>while</u> depth=k+l+1 <u>do</u> $\{\mathbf{depth=k+l+1}\}$; $\mathbf{guard}_{k+l+1}(\mathbf{S}_1)$ **<u>od</u>; IF**

by forward expansion of **<u>do</u>**.

$\approx$ $\{\mathbf{depth=k+l}\}$; $\mathbf{guard}_{k+l}($**<u>do</u>** $\mathbf{S}_1$ **<u>od</u>**$)$; **IF**

This proves the Lemma.

<u>Cor</u> <u>A:</u> $\Delta \vdash \{\mathbf{depth=d}\}$; $\mathbf{guard}_d(\mathbf{S+d}) \approx \{\mathbf{depth=d}\}$; **depth:=depth−d;** $\mathbf{guard}_0(\mathbf{S})$

Proof: From Lemma A

$\{\mathbf{depth=d}\}$; $\mathbf{guard}_d(\mathbf{S+d})$

$\approx$ $\{\mathbf{depth=d}\}$; $\mathbf{guard}_d(\mathbf{S})$; **<u>if</u> depth⩽d <u>then</u> depth:=depth−d <u>fi</u>**

$\approx$ $\{\mathbf{depth=d}\}$; $\mathbf{guard}_d(\mathbf{S})$; **depth:=depth−d** since **S** cannot increase **depth**

$\approx$ $\{\mathbf{depth=d}\}$; **depth:=depth−d;** $\mathbf{guard}_0(\mathbf{S})$ from a property of **guard**.

Lemma B: If $\mathbf{S}$ is $\mathbf{d}$-reducible and has no terminal statement with terminal value $\mathbf{d}$ then:
$\Delta \vdash$ **{depth=d+1}; guard**$_{d+1}$**(S)**
$\approx$ **{depth=d+1}; guard**$_{d+1}$**(S[T−1/T**$|\tau \geqslant$**d+1]);**
if depth$\leqslant$**1 then depth:=depth−1 fi**
Proof: By induction on the structure of $\mathbf{S}$ using the same cases as Lemma A.

Simple Absorption: $\Delta \vdash$ **S;S′** $\approx$ **S[S′** $+ \delta$**/T**$|\tau =$**0]**

The statement $\mathbf{S'}$ following $\mathbf{S}$ is "absorbed" into it by replacing all of the terminal statements of $\mathbf{S}$ which would lead to $\mathbf{S'}$ by $\mathbf{S'}$ incremented by the depth of the terminal statement. This is used a great deal when restructuring an unstructured program.

For our example program: **Prog; do r:=r−1; if r=0 then exit fi**
is equivalent to **Prog** with the first occurrence (only) of **exit** replaced by:

do r:=r−1; if r=0 then exit(2) fi

Proof of Simple Absorption: Prove that for all $\mathbf{d}$
{depth=d}; guard$_d$**(S); guard**$_0$**(S′)** $\approx$ **{depth=d}; guard**$_d$**(S[S′** $+ \delta$**+d/T**$|\tau =$**d])**
See the proof of the converse for the cases used.

Defn: $\mathbf{S'}$ is a term of $\mathbf{S}$ if for each $\mathbf{k}$ the replacement of $\mathbf{S'}$+$\mathbf{k}$ by **exit(k)** in $\mathbf{S}$ produces a terminal statement with terminal value zero.
This is a generalisation of Arsac's definition of a term in [Arsac 79]. A further generalisation is:

Defn: For $\mathbf{d} \in \mathbb{N}$, $\mathbf{S'}$ is a $\mathbf{d}$-term of $\mathbf{S}$ if for any $\mathbf{k}$ each replacement of $\mathbf{S'}$+$\mathbf{k}$ by **exit(k)** in $\mathbf{S}$ produces a terminal statement with terminal value $\mathbf{d}$.
Thus a term (defined by the previous definition) is a $\mathbf{0}$-term.

Lemma: $\mathbf{S'}$ is a $\mathbf{d}$-term of **do** $\mathbf{S}_1$ **od** iff $\mathbf{S'}$ is a $\big(\mathbf{d}+\mathbf{1}\big)$-term of $\mathbf{S}_1$.
Proof: If some occurrence **exit(k)** is a terminal statement of **do S od** with terminal value $\mathbf{d}$ then the same occurrence is a terminal statement of $\mathbf{S}$ with terminal value $\mathbf{d}$+$\mathbf{1}$ and vice versa. Thus if replacing some occurrence of $\mathbf{S'}$+$\mathbf{k}$ by **exit(k)** in **do S od** gives a terminal statement with terminal value $\mathbf{d}$ then the same substitution in $\mathbf{S}$ gives a terminal statement with terminal value $\mathbf{d}$+$\mathbf{1}$. Conversely if replacing $\mathbf{S'}$+$\mathbf{k}$ in $\mathbf{S}$ gives a terminal statement with terminal value $\mathbf{d}$+$\mathbf{1}$ then this will be a terminal statement of **do S od** with $\mathbf{S'}$+$\mathbf{k}$ replaced by **exit(k)** having terminal value $\mathbf{d}$. For $\mathbf{S}_1$;$\mathbf{S}_2$, a $\mathbf{d}$-term of $\mathbf{S}_1$ is a $\mathbf{d}$-term of the compound iff $\mathbf{d}$>$\mathbf{0}$. A $\mathbf{d}$-term of $\mathbf{S}_2$ is always a $\mathbf{d}$-term of the compound. For all other compound statements, a $\mathbf{d}$-term of any component is a $\mathbf{d}$-term of the

compound.

<u>Theorem:</u> If $\mathbf{S}'$ is a term of $\mathbf{S}$ and every terminal statement of $\mathbf{S}$ with terminal value zero occurs within an occurrence of $\mathbf{S}'+\mathbf{k}$ in $\mathbf{S}$ (for some $\mathbf{k}$) then

$$\Delta \vdash \mathbf{S} \;\approx\; \mathbf{S}[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}];\mathbf{S}'$$

This is a converse to simple absorption, it is a generalisation of Arsac's version, which makes it more practically useful.

Proof: Prove the following by induction on the structure of $\mathbf{S}$:
If $\mathbf{S}'$ is a $\mathbf{d}$-term of $\mathbf{S}$ and any terminal statement of $\mathbf{S}$ with terminal value $\mathbf{d}$ is within an occurrence of $\mathbf{S}'+\mathbf{k}$ in $\mathbf{S}$ (for some $\mathbf{k}$) then

$$\mathbf{S} = \mathbf{S}[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}].$$

We may assume $\mathbf{S}'$ is not primitive or of the form $\mathbf{S}'_1;\mathbf{S}'_2$ since otherwise we may replace $\mathbf{S}'+\mathbf{k}$ throughout $\mathbf{S}$ by $\underline{\mathbf{if}}$ **true** $\underline{\mathbf{then}}$ $\mathbf{S}'$ $\underline{\mathbf{fi}}$ $+ \mathbf{k}$ which is not primitive and not a sequence.

<u>Case</u> (i): $\mathbf{S}= \mathbf{S}'+\mathbf{n}$ for some $\mathbf{n}$. We must have $\mathbf{n}=\mathbf{d}$ since $\mathbf{S}'$ is a $\mathbf{d}$-term of $\mathbf{S}$:
$\big(\mathbf{S}'+\mathbf{d}\big)[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \underline{\mathbf{exit}}(\mathbf{d})[\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}] = \mathbf{S}'+\mathbf{d}$

<u>Case</u> (ii): $\mathbf{S}$ is primitive. Hence $\mathbf{S}\neq\mathbf{S}'+\mathbf{k}$ for any $\mathbf{k}$. Thus:
$\mathbf{S}[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}] = \mathbf{S}[\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \mathbf{S}$ since $\mathbf{S}$ cannot be $\underline{\mathbf{exit}}(\mathbf{d})$ by premise.

<u>Case</u> (iii): $\mathbf{S}= \mathbf{S}_1;\mathbf{S}_2$.
$\big(\mathbf{S}_1;\mathbf{S}_2\big)[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \big(\mathbf{S}_1[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}];\ \mathbf{S}_2[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}]\big)[\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
since each occurrence of $\mathbf{S}'+\mathbf{k}$ must be within $\mathbf{S}_1$ or $\mathbf{S}_2$ since $\mathbf{S}'+\mathbf{k}$ is not a sequence.
$\quad = \mathbf{S}_1[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}];\ \mathbf{S}_2[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \mathbf{S}_1;\mathbf{S}_2$ by induction hypothesis

<u>Case</u> (iv): $\mathbf{S}=$ $\underline{\mathbf{if}}$ $\mathbf{B}$ $\underline{\mathbf{then}}$ $\mathbf{S}_1$ $\underline{\mathbf{else}}$ $\mathbf{S}_2$ $\underline{\mathbf{fi}}$ and $\mathbf{S}\neq\mathbf{S}'+\mathbf{k}$. Any occurrence of $\mathbf{S}'+\mathbf{k}$ must be within either $\mathbf{S}_1$ or $\mathbf{S}_2$. Use the induction hypothesis and forward expansion of $\underline{\mathbf{if}}$.

<u>Case</u> (v): $\mathbf{S}=$ $\underline{\mathbf{do}}$ $\mathbf{S}_1$ $\underline{\mathbf{od}}$. By Lemma $\mathbf{S}_1$ is a $\big(\mathbf{d}+\mathbf{1}\big)$-term of $\mathbf{S}_1$ so
$\big(\underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}\big)[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \big(\underline{\mathbf{do}}\ \mathbf{S}_1[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}]\ \underline{\mathbf{od}}\big)[\mathbf{S}' + \delta+\mathbf{d}/\mathbf{T}|\tau = \mathbf{d}]$
$\quad = \underline{\mathbf{do}}\ \mathbf{S}_1[\underline{\mathbf{exit}}(\mathbf{k})/\mathbf{S}'+\mathbf{k}][\mathbf{S}' + \delta+\mathbf{d}+\mathbf{1}/\mathbf{T}|\tau = \mathbf{d}+\mathbf{1}]\ \underline{\mathbf{od}}$

$= \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}$ by induction hypothesis
Which proves the Theorem.

<u>False</u> <u>Iteration:</u> $\Delta \vdash \mathbf{S}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S+1}\ \underline{\mathbf{od}}$
Proof: $\mathbf{\{depth{=}1\};\ guard_1(S{+}1)}$
$\approx\ \mathbf{\{depth{=}1\};\ guard_1(S);\ depth{:=}depth{-}1}$
by a previous result.
Thus $\mathbf{\{depth{=}0\};\ depth{:=}1;}\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{\{depth{=}1\};\ guard_1(S{+}1)}\ \underline{\mathbf{od}}$
$\approx\ \mathbf{\{depth{=}0\};\ depth{:=}1;}$
$\underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}$
$\mathbf{\{depth{=}1\};\ guard_1(S);\ depth{:=}depth{-}1}\ \underline{\mathbf{od}}$
By loop unrolling this is:
$\approx\ \mathbf{\{depth{=}0\};\ depth{:=}1;}$
$\underline{\mathbf{if}}\ \mathbf{depth{=}1}$
$\underline{\mathbf{then}}\ \mathbf{\{depth{=}1\};\ guard_1(S);\ \{depth{\leqslant}1\};\ depth{:=}depth{-}1;\ \{depth{\leqslant}0\};}$
$\underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}$
$\mathbf{\{depth{=}1\};\ guard_1(S);\ \{depth{\leqslant}1\};}$
$\mathbf{depth{:=}depth{-}1;\ \{depth{\leqslant}0\}}\ \underline{\mathbf{od}}$
$\approx\ \mathbf{\{depth{=}0\};\ depth{:=}1;\ guard_1(S);\ depth{:=}depth{-}1}$
$\approx\ \mathbf{\{depth{=}0\};\ guard_0(S)}$ by the following Lemma.

<u>Lemma:</u> For any $\mathbf{n} \in \mathbb{N}$ and any statement **S**:
$$\Delta \vdash \mathbf{depth{:=}depth{+}1;\ guard_{n+1}(S);\ depth{:=}depth{-}1}\ \approx\ \mathbf{guard_n(S)}$$
Proof: By induction on the structure of **S**:

<u>Cor:</u> By induction on **m** we have the more general form:
$$\Delta \vdash \mathbf{guard_n(S)}\ \approx\ \mathbf{depth{:=}depth{+}m;\ guard_{n+m}(S);\ depth{:=}depth{-}m}$$

<u>Defn:</u> **S** is <u>reducible</u> if replacing any terminal statement $\underline{\mathbf{exit}}\mathbf{(k)}$, which has terminal value one, by $\underline{\mathbf{exit}}\mathbf{(k{-}1)}$ gives a terminal statement of **S**.

Note that any statement can be made reducible by the use of absorption. For example:
$\underline{\mathbf{if}}\ \mathbf{i{=}n}\ \underline{\mathbf{then}}\ \underline{\mathbf{exit}}\ \underline{\mathbf{fi}}\mathbf{;\ i{:=}i{+}1.}$
is not reducible, but by absorption we get:
$\underline{\mathbf{if}}\ \mathbf{i{=}n}\ \underline{\mathbf{then}}\ \underline{\mathbf{exit}}$
$\underline{\mathbf{else}}\ \mathbf{i{:=}i{+}1}\ \underline{\mathbf{fi}}\mathbf{.}$

which is reducible. It reduces to:

$\underline{\textbf{if}}$ **i=n** $\underline{\textbf{then}}$ **skip**

  $\underline{\textbf{else}}$ **i:=i+1** $\underline{\textbf{fi}}$**.**

<u>Defn:</u> **S** is **d-**<u>reducible</u> if replacing any terminal statement $\underline{\textbf{exit}}\textbf{(k)}$, which has terminal value **d+1**, by $\underline{\textbf{exit}}\textbf{(k}-\textbf{1)}$ gives a terminal statement of **S**. Note that this is always the case for **d>0**.

<u>Theorem:</u> If **S** is reducible and all terminal statements of **S** have terminal value greater than zero then:

$$\Delta \vdash \underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \;\approx\; \textbf{S}[\textbf{T}-\textbf{1}/\textbf{T}|\tau > \textbf{0}] = \textbf{S}-\textbf{1}$$

where $\underline{\textbf{exit}}\textbf{(k)}-\textbf{1} = \underline{\textbf{exit}}\textbf{(k}-\textbf{1)}$ for **k>0**. In the substitution **T** must be an $\underline{\textbf{exit}}\textbf{(k)}$ with **k>0** for it to have $\tau >$**0** so **T−1** is always defined. This is a converse to false iteration.

Proof: From Lemma A with **k=1** and **l=d** we have:

$\{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S+(1,d))}$

  $\approx\ \{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S)};\ \underline{\textbf{if}}\ \textbf{depth} \leqslant \textbf{1}\ \underline{\textbf{then}}\ \textbf{depth:=depth}-\textbf{1}\ \underline{\textbf{fi}}$

From Lemma B we have:

$\{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S)}$

  $\approx\ \{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S}-\textbf{(1,d+1))};\ \underline{\textbf{if}}\ \textbf{depth} \leqslant \textbf{1}\ \underline{\textbf{then}}\ \textbf{depth:=depth}-\textbf{1}\ \underline{\textbf{fi}}$

Hence if **S** is **d-**reducible and has no terminal statement with terminal value **d** then:

$\{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{((S}-\textbf{(1,d+1))+(1,d))}$

  $\approx\ \{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S}-\textbf{(1,d+1))};\ \underline{\textbf{if}}\ \textbf{depth} \leqslant \textbf{1}\ \underline{\textbf{then}}\ \textbf{depth:=depth}-\textbf{1}\ \underline{\textbf{fi}}$

  $\approx\ \{\textbf{depth=d+1}\};\ \textbf{guard}_{d+1}\textbf{(S)}$

ie $\textbf{(S}-\textbf{(1,d+1))+(1,d)} \;\approx\; \textbf{S}$.

Putting **d=1** we get: $\textbf{S} \;\approx\; \textbf{(S}-\textbf{(1,1))+1}$

Hence $\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \;\approx\; \underline{\textbf{do}}\ \textbf{(S}-\textbf{(1,1))+1}\ \underline{\textbf{od}} \;\approx\; \textbf{S}-\textbf{(1,1)} = \textbf{S}-\textbf{1}$ (by false iteration).

A consequence of these two results is:

<u>Theorem:</u> If **S′** is a term of **S** then:

$$\Delta \vdash \textbf{S} \;\approx\; \underline{\textbf{do}}\ \big(\textbf{S+1}\big)[\textbf{S}'\textbf{+k+1}/\underline{\textbf{exit}}\textbf{(k)}];\ \textbf{S}'\textbf{+1}\ \underline{\textbf{od}}$$

This can be used to combine multiple copies of a statement into a single copy by putting a false loop around the program, replacing each copy of the statement by an $\underline{\textbf{exit}}$ and putting a single copy at the end of the loop body. Multiple copies of a statement often occur during the removal of recursion and during the restructuring of an unstructured program.

Proof: If **S′** is a term of **S** then **S′+1** is a term of **S+1** and **S+1** has no terminal statement with terminal value zero, so by absorption:

$$\mathbf{S+1} \approx \big(\mathbf{S+1}\big)[\mathbf{S'+k+1}/\underline{\mathbf{exit}}(\mathbf{k})];\ \mathbf{S'+1}$$

And by false iteration: $\mathbf{S} \approx \underline{\mathbf{do}}\ \mathbf{S+1}\ \underline{\mathbf{od}}$.

<u>Defn:</u> **S** is a <u>proper</u> <u>sequence</u> iff every terminal statement of **S** has terminal value zero, ie

$$\forall\mathbf{T,n.\ ts(n,T,S)} \Rightarrow \tau\mathbf{(n,T,S)=0}$$

Thus a proper sequence cannot change **depth**.

If the body of a loop has no terminal statement with terminal value zero then the loop is a "false loop" (the body is only executed once since the execution of any terminal statement in the body will cause termination of the loop). If the body is reducible then the loop can br removed. Note that a statement can always be made reducible by absorption so this is always possible: but the absorption may cause an increase in the program text length. Such "false loops" are useful in "factoring out" several occurrences of a statement into a single occurrence. For example:

$\underline{\mathbf{if}}\ \mathbf{B_1}\ \underline{\mathbf{then}}\ \mathbf{S_1};\ \underline{\mathbf{if}}\ \mathbf{B_2}\ \underline{\mathbf{then}}\ \mathbf{S_2} \approx \underline{\mathbf{do}}\ \underline{\mathbf{if}}\ \mathbf{B_1}\ \underline{\mathbf{then}}\ \mathbf{S_1};\ \underline{\mathbf{if}}\ \mathbf{B_2}\ \underline{\mathbf{then}}\ \mathbf{S_2+1}\ \underline{\mathbf{fi}}\ \underline{\mathbf{fi}};$
$\quad\quad \underline{\mathbf{else}}\ \mathbf{S}\ \underline{\mathbf{fi}} \quad\quad\quad\quad \mathbf{S+1}\ \underline{\mathbf{od}}$
$\quad \underline{\mathbf{else}}\ \mathbf{S}\ \underline{\mathbf{fi}}$

where the second version has only one copy of **S**. If we make the body of the loop on the RHS reducible by absorbing **S+1** and then remove the false iteration we get the LHS.

<u>Loop</u> <u>inversion:</u> $\Delta \vdash \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{S_1};\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_2}\ \underline{\mathbf{fi}}\ \underline{\mathbf{od}}$
$\quad\quad\quad\quad \approx\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}};\ \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{S_2};\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}}\ \underline{\mathbf{od}}$

Proof: Since the assertion **{B}** can be inserted in the loop $\underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ldots\ \underline{\mathbf{od}}$ it is sufficient to prove:

$\Delta \vdash \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}};\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_2}\ \underline{\mathbf{fi}}\ \underline{\mathbf{od}}$
$\quad \approx\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}};\ \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_2}\ \underline{\mathbf{fi}};\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}}\ \underline{\mathbf{od}}$

Let $\mathbf{IF_1} = \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_1}\ \underline{\mathbf{fi}}$, $\mathbf{IF_2} = \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \mathbf{S_2}\ \underline{\mathbf{fi}}$. So we are trying to prove

$$\Delta \vdash \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_1};\ \mathbf{IF_2}\ \underline{\mathbf{od}} \approx \mathbf{IF_1};\ \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_2};\ \mathbf{IF_1}\ \underline{\mathbf{od}}$$

where the two statements inside the loop have been reversed–hence the name of the theorem. Use induction to show that:

$\Delta \vdash \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_1};\ \mathbf{IF_2}\ \underline{\mathbf{od}}^n \leqslant \mathbf{IF_1};\ \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_2};\ \mathbf{IF_1}\ \underline{\mathbf{od}}^n$ for $\mathbf{n>1}$.

Then show that:

$\Delta \vdash \mathbf{IF_1};\ \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_2};\ \mathbf{IF_1}\ \underline{\mathbf{od}}^n \leqslant \underline{\mathbf{while}}\ \mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{IF_1};\ \mathbf{IF_2}\ \underline{\mathbf{od}}^{n+1}$ for $\mathbf{n\geqslant 1}$.

The theorem follows from the general induction rule for iteration.

<u>Inversion:</u>

$\Delta \vdash \underline{\mathbf{do}}\ \mathbf{S_1;S_2}\ \underline{\mathbf{od}} \approx \underline{\mathbf{do}}\ \mathbf{S_1};\ \underline{\mathbf{do}}\ \mathbf{S_2;S_1}\ \underline{\mathbf{od}}\mathbf{+1}\ \underline{\mathbf{od}}$

Proof: Let $\mathbf{IF}_1 = \underline{\mathbf{if}}\ \mathbf{depth{=}1}\ \underline{\mathbf{then}}\ \mathbf{guard}_1(\mathbf{S}_1)\ \underline{\mathbf{fi}}$, $\mathbf{IF}_2 = \underline{\mathbf{if}}\ \mathbf{depth{=}1}\ \underline{\mathbf{then}}\ \mathbf{guard}_1(\mathbf{S}_2)\ \underline{\mathbf{fi}}$

$\{\mathbf{depth{=}0}\};\ \mathbf{guard}_0(\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}})$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S}_1;\mathbf{S}_2)\ \underline{\mathbf{od}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{IF}_1;\ \mathbf{IF}_2\ \underline{\mathbf{od}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{IF}_1;\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{IF}_2;\ \mathbf{IF}_1\ \underline{\mathbf{od}}$

by loop inversion.

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{guard}_1(\mathbf{S}_1);\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S}_2;\mathbf{S}_1)\ \underline{\mathbf{od}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{guard}_1(\mathbf{S}_1);$
$\quad\quad \underline{\mathbf{if}}\ \mathbf{depth{=}1}\ \underline{\mathbf{then}}\ \{\mathbf{depth{=}1}\};\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S}_2;\mathbf{S}_1)\ \underline{\mathbf{od}}\ \underline{\mathbf{fi}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{guard}_1(\mathbf{S}_1);$
$\quad\quad \underline{\mathbf{if}}\ \mathbf{depth{=}1}$
$\quad\quad\ \underline{\mathbf{then}}\ \mathbf{depth{:=}2};\ \underline{\mathbf{while}}\ \mathbf{depth{=}2}\ \underline{\mathbf{do}}\ \mathbf{guard}_2(\mathbf{S}_2;\mathbf{S}_1);$
$\quad\quad\quad\quad\quad\quad \underline{\mathbf{if}}\ \mathbf{depth}\leqslant\mathbf{1}\ \underline{\mathbf{then}}\ \mathbf{depth{:=}depth}-\mathbf{1}\ \underline{\mathbf{fi}};\ \underline{\mathbf{od}};$
$\quad\quad\quad \{\mathbf{depth}<\mathbf{1}\}\ \underline{\mathbf{fi}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{guard}_1(\mathbf{S}_1);$
$\quad\quad \underline{\mathbf{if}}\ \mathbf{depth{=}1}$
$\quad\quad\ \underline{\mathbf{then}}\ \mathbf{depth{:=}2}\ \underline{\mathbf{while}}\ \mathbf{depth{=}2}\ \underline{\mathbf{do}}\ \mathbf{guard}_2\big((\mathbf{S}_2;\mathbf{S}_1)+(\mathbf{1,1})\big)\ \underline{\mathbf{od}};$
$\quad\quad\quad \{\mathbf{depth}<\mathbf{1}\}\ \underline{\mathbf{fi}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \mathbf{guard}_1(\mathbf{S}_1);\ \mathbf{guard}_1(\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1});\ \{\mathbf{depth}<\mathbf{1}\}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};$
$\quad\quad \underline{\mathbf{if}}\ \mathbf{depth{=}1}$
$\quad\quad\ \underline{\mathbf{then}}\ \mathbf{guard}_1(\mathbf{S}_1);\ \mathbf{guard}_1(\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1});$
$\quad\quad\quad \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1})\ \underline{\mathbf{od}}\ \underline{\mathbf{fi}}$

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{depth{:=}1};\ \underline{\mathbf{while}}\ \mathbf{depth{=}1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1})\ \underline{\mathbf{od}}$

by loop rolling.

$\quad\approx\ \{\mathbf{depth{=}0}\};\ \mathbf{guard}_0(\underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \underline{\mathbf{od}})$

Hence: $\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \underline{\mathbf{od}}$ as required.

This transformation is often used in converting a $\underline{\mathbf{do}}$ loop with an $\underline{\mathbf{exit}}$ in the middle into a $\underline{\mathbf{while}}$ loop by moving some statements outside the loop. For example if $\mathbf{S}_1$ and $\mathbf{S}_2$ are proper sequences then:

$\underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \underline{\mathbf{exit}}\ \underline{\mathbf{fi}};\ \mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \underline{\mathbf{exit}}\ \underline{\mathbf{fi}};\ \mathbf{S}_2;\ \mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \underline{\mathbf{od}}$

$\quad\approx\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \underline{\mathbf{if}}\ \mathbf{B}\ \underline{\mathbf{then}}\ \underline{\mathbf{exit}}\ \underline{\mathbf{fi}};\ \mathbf{S}_2;\ \mathbf{S}_1\ \underline{\mathbf{od}}$ (since $\mathbf{S}_1$ is a proper sequence)

$\quad\approx\ \mathbf{S}_1;\ \underline{\mathbf{while}}\ \neg\mathbf{B}\ \underline{\mathbf{do}}\ \mathbf{S}_2;\ \mathbf{S}_1\ \underline{\mathbf{od}}$. (since $\mathbf{S}_1$ and $\mathbf{S}_2$ are proper sequences).

Theorem: If $\mathbf{S}_1$ is reducible then

$$\Delta \vdash \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \mathbf{S}_1[\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\delta+\mathbf{1/T}|\tau=\mathbf{0}]-\mathbf{1}$$

Proof: By inversion:
$\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \underline{\mathbf{od}}$
By absorption $\mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \approx\ \mathbf{S}_1[\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\delta+\mathbf{1/T}|\tau=\mathbf{0}]$

Claim: $\mathbf{S}_1[\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\delta+\mathbf{1/T}|\tau=\mathbf{0}]$ is reducible and has no terminal statement with terminal value zero. To prove this claim, prove the following by induction on the structure of $\mathbf{S}$:
For any $\mathbf{k}$, if $\mathbf{S}$ s d-reducible then $\mathbf{S}[\mathbf{S}'+\mathbf{k}+\delta+\mathbf{d/T}|\tau=\mathbf{d}]$ is also $\mathbf{d}$-reducible and has no terminal statement with terminal value $\mathbf{d}$.
+hen by false iteration: $\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1[\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\delta+\mathbf{1/T}|\tau=\mathbf{0}]\ \underline{\mathbf{od}}$
R $\approx\ \mathbf{S}_1[\underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\delta+\mathbf{1/T}|\tau=\mathbf{0}]-\mathbf{1}$
which proves the theorem.

Proper Inversion:
If $\mathbf{S}_1$ is a proper sequence then: $\Delta \vdash \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}$
Proof: If $\mathbf{S}_1$ is a proper sequence then for any statement $\mathbf{S}$:

$$(\mathbf{S}_1;\mathbf{S})+\mathbf{1} = \mathbf{S}_1+(\mathbf{1},\mathbf{1});\ \mathbf{S}_2+\mathbf{1} = \mathbf{S}_1;\ \mathbf{S}+\mathbf{1}$$

since all of the terminal statements of $\mathbf{S}_1$ have terminal value zero.
Thus $\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}+\mathbf{1}\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ (\mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}})+\mathbf{1}\ \underline{\mathbf{od}}$
$\approx\ \mathbf{S}_1;\ \underline{\mathbf{do}}\ \mathbf{S}_2;\mathbf{S}_1\ \underline{\mathbf{od}}$ by false iteration.

Repetition:
(a) $\Delta \vdash \mathbf{S}_1 \leqslant \mathbf{S}\ \wedge\ \mathbf{S}_2 \leqslant \mathbf{S} \Rightarrow \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}}$
(b) $\Delta \vdash \mathbf{S} \leqslant \mathbf{S}_1\ \wedge\ \mathbf{S} \leqslant \mathbf{S}_2 \Rightarrow \underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}$

Cor: $\Delta \vdash \underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S};\mathbf{S}\ \underline{\mathbf{od}}$ (Loop doubling)
$\Delta \vdash \mathbf{S}_2 \leqslant \mathbf{S}_1 \Rightarrow \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}$
$\Delta \vdash \mathbf{S}_1 \leqslant \mathbf{S}_2 \Rightarrow \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}$

Proof: These are in fact all consequences of **loop doubling:**
(a) If $\mathbf{S}_1 \leqslant \mathbf{S}$ and $\mathbf{S}_2 \leqslant \mathbf{S}$ then $\mathbf{S}_1;\mathbf{S}_2 \leqslant \mathbf{S};\mathbf{S}$ and hence $\underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S};\mathbf{S}\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}}$
(b) If $\mathbf{S} \leqslant \mathbf{S}_1$ and $\mathbf{S} \leqslant \mathbf{S}_2$ then $\mathbf{S};\mathbf{S} \leqslant \mathbf{S}_1;\mathbf{S}_2$ and hence $\underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S};\mathbf{S}\ \underline{\mathbf{od}} \leqslant \underline{\mathbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\mathbf{od}}$

Proof of Loop Doubling:
$\{\mathbf{depth}=\mathbf{0}\};\ \mathbf{guard}_0(\underline{\mathbf{do}}\ \mathbf{S}\ \underline{\mathbf{od}})\ \approx\ \{\mathbf{depth}=\mathbf{0}\};\ \mathbf{depth}:=\mathbf{1};\ \underline{\mathbf{while}}\ \mathbf{depth}=\mathbf{1}\ \underline{\mathbf{do}}\ \mathbf{guard}_1(\mathbf{S})\ \underline{\mathbf{od}}$

Let **IF** $\equiv$ **<u>if</u> depth=1 <u>then</u> guard$_1$(S) <u>fi</u>** $\approx$ **guard$_1$(S)**.
Let **DO$_1$** $\equiv$ **<u>while</u> depth=1 <u>do</u> IF <u>od</u>**
DO$_2$ $\equiv$ **<u>while</u> depth=1 <u>do</u> IF;IF <u>od</u>**
We need to prove **DO$_1$** $\approx$ **DO$_2$**.
Proof is by induction rule for loops:

Claim: $\mathbf{DO}_1^{2n} \leqslant \mathbf{DO}_2$ for all $\mathbf{n} < \omega$.
Trivial for **n=0** so suppose it holds for **n**.
$\mathbf{DO}_1^{2(n+1)} \approx \mathbf{DO}_1^{2n+2} \approx$ **<u>if</u> depth=1 <u>then</u> IF; <u>if</u> d=1 <u>then</u> IF;** $\mathbf{DO}_1^{2n}$ **<u>fi</u> <u>fi</u>**
$\leqslant$ **<u>if</u> depth=1 <u>then</u> IF; <u>if</u> d=1 <u>then</u> IF; DO$_2$ <u>fi</u> <u>fi</u>** by induction hyp
$\approx$ **<u>if</u> depth=1 <u>then</u> IF; IF; DO$_2$ <u>fi</u>** by case analysis on inner **<u>if</u>**
$\approx$ **DO$_2$** by loop rolling.

Claim: $\mathbf{DO}_2^{n} \leqslant \mathbf{DO}_1$ for all $\mathbf{n} < \omega$.
Trivial for **n=0** so suppose it holds for **n**.

$\mathbf{DO}_2^{n+1} \approx$ **<u>if</u> depth=1 <u>then</u> IF; IF;** $\mathbf{DO}_2^{n}$ **<u>fi</u>**
$\kappa\kappa\kappa \leqslant$ **<u>if</u> depth=1 <u>then</u> IF; IF; DO$_1$ <u>fi</u>** by induction hypothesis
$\approx$ **<u>if</u> depth=1 <u>then</u> IF; <u>if</u> depth=1 <u>then</u> IF; DO$_1$ <u>fi</u> <u>fi</u>** by case analysis
$\approx$ **<u>if</u> depth=1 <u>then</u> IF; DO$_1$ <u>fi</u>** by loop rolling
$\approx$ **DO$_1$**.
Hence by induction rule for loops **DO$_1$** $\approx$ **DO$_2$**.

<u>Arsac's Version of the Repetition Transformation</u>
Arsac in [Arsac 79] gives the following, more general, version of this transformation:

<u>do</u> S$_1$ <u>od</u> $\approx$ **<u>do</u> S$_2$ <u>od</u>** $\iff$ **<u>do</u> S$_1$ <u>od</u>** $\approx$ **<u>do</u> S$_1$;S$_2$ <u>od</u>**

However, this fails in general, to see this take **S$_2$** =**skip**. Then the **RHS** is:
<u>do</u> S$_1$ <u>od</u> $\approx$ **<u>do</u> S$_1$; skip <u>od</u>** which holds for any **S$_1$** while the **LHS** is:
<u>do</u> S$_1$ <u>od</u> $\approx$ **<u>do</u> skip <u>od</u>** $\approx$ **abort** which is clearly not true in general.

The other implication, namely:
<u>do</u> S$_1$ <u>od</u> $\approx$ **<u>do</u> S$_2$ <u>od</u>** $\Rightarrow$ **<u>do</u> S$_1$ <u>od</u>** $\approx$ **<u>do</u> S$_1$;S$_2$ <u>od</u>**
looks more convincing but also fails in general. Take
S$_1$ = **<u>if</u> x$\leqslant$0 <u>then</u> <u>exit</u> <u>fi</u>; x:=x$-$1**

$\mathbf{S}_2 = \underline{\textbf{if}}\ \mathbf{x \leqslant 0}\ \underline{\textbf{then}}\ \underline{\textbf{exit}}\ \underline{\textbf{fi}}\textbf{; x:=x}-\mathbf{2}$
Then it is easy to see that: $\{\mathbf{x \geqslant 0} \wedge \textbf{even(x)}\} \vdash \underline{\textbf{do}}\ \mathbf{S}_1\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \mathbf{S}_2\ \underline{\textbf{od}} \approx \textbf{x:=0}$
But if $\textbf{x=2}$ initially then $\underline{\textbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\textbf{od}} \approx \textbf{x:=}-\mathbf{1}$
So $\underline{\textbf{do}}\ \mathbf{S}_1\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \mathbf{S}_1;\mathbf{S}_2\ \underline{\textbf{od}}$ cannot be true.

Arsac uses this transformation to derive a version of "selective unrolling" (see next Chapter) which cannot in fact be derived from the transformations he gives. This invalidates his claim that his set of transformations is "complete" in the sense that any syntactic transformation (ie a transformation that preserves the sequence of states) can be derived from them. For example, the following example of selective unrolling cannot be derived from the transformations given by Arsac:

$$\begin{array}{lcl} \underline{\textbf{do}}\ \ \underline{\textbf{if}}\ \textbf{B}\ \underline{\textbf{then}}\ \mathbf{S}_1 & \approx & \underline{\textbf{do}}\ \underline{\textbf{if}}\ \textbf{B}\ \underline{\textbf{then}}\ \mathbf{S}_1;\ \underline{\textbf{if}}\ \textbf{B}\ \underline{\textbf{then}}\ \mathbf{S}_1\ \underline{\textbf{fi}} \\ \qquad \underline{\textbf{else}}\ \mathbf{S}_2\ \underline{\textbf{fi}}\ \underline{\textbf{od}}\textbf{.} & & \qquad\qquad \underline{\textbf{else}}\ \mathbf{S}_2\ \underline{\textbf{fi}}\ \underline{\textbf{od}}\textbf{.} \end{array}$$

This is because each of his transformations preserves the property that the number of copies of $\mathbf{S}_1$ within any loop equals the number of copies of $\mathbf{S}_2$. This property holds for the first version, but for the second version we have two copies of $\mathbf{S}_1$ and only one of $\mathbf{S}_2$. Hence no sequence of Arsac's transformations will convert the LHS into the RHS.

<u>First</u> <u>Step</u> <u>Unrolling:</u>
$\Delta \vdash \underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \textbf{S;}\ \underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}}\mathbf{+1}\ \underline{\textbf{od}}$
Proof: $\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \textbf{S;S}\ \underline{\textbf{od}}$ loop doubling.
$\approx \underline{\textbf{do}}\ \textbf{S;}\ \underline{\textbf{do}}\ \textbf{S;S}\ \underline{\textbf{od}}\mathbf{+1}\ \underline{\textbf{od}}$ inversion.
$\approx \underline{\textbf{do}}\ \textbf{S;}\ \underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}}\mathbf{+1}\ \underline{\textbf{od}}$ loop doubling.
If **S** is reducible then $\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \approx \textbf{S}[\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}}\mathbf{+1}+\delta/\mathbf{T}|\tau\ \mathbf{=0}]\mathbf{-1}$

Proof: $\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \textbf{S;S}\ \underline{\textbf{od}}$ loop doubling.
$\approx \textbf{S}[\underline{\textbf{do}}\ \textbf{S;S}\ \underline{\textbf{od}}\mathbf{+1}+\delta/\mathbf{T}|\tau\ \mathbf{=0}]\mathbf{-1}$ inversion.
$\approx \textbf{S}[\underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}}\mathbf{+1}+\delta/\mathbf{T}|\tau\ \mathbf{=0}]\mathbf{-1}$ loop doubling.

<u>Double</u> <u>Iteration:</u>
If **S** is reducible then: $\Delta \vdash \underline{\textbf{do}}\ \underline{\textbf{do}}\ \textbf{S}\ \underline{\textbf{od}}\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \textbf{S}[\mathbf{T-1/T}|\tau\ \mathbf{>0}]\ \underline{\textbf{od}} \approx \underline{\textbf{do}}\ \mathbf{S-1}\ \underline{\textbf{od}}$

Note that any statement can be made reducible by the repeated application of absorption, hence any double loop can be converted to a single loop if required. However in general this will cause an increase in program text length. The choice of whether to use a single loop or a double loop can be made on the basis of which version best expresses the solution of the problem.

Defn: If $\mathbf{k} \leqslant \mathbf{d}$ then $\mathbf{S}-(\mathbf{k},\mathbf{d}) =_{DF} \mathbf{S}[\mathbf{T}-\mathbf{k}/\mathbf{T}|\tau \geqslant \mathbf{d}]$.
Note $\big(\mathbf{S}+(\mathbf{k},\mathbf{d})\big)-(\mathbf{k},\mathbf{d}) = \mathbf{S}$, in fact $\big(\mathbf{S}+(\mathbf{k},\mathbf{d})\big)-(\mathbf{k},\mathbf{d}') = \mathbf{S}$ holds for any $\mathbf{d}'$ with $\mathbf{d} \leqslant \mathbf{d}' < \mathbf{d}+\mathbf{k}$.
The converse $\big(\mathbf{S}-(\mathbf{k},\mathbf{d})\big)+(\mathbf{k},\mathbf{d}) = \mathbf{S}$ is not valid: for example $\mathbf{S}-(\mathbf{k},\mathbf{k})$ is the same as $\mathbf{S}-\mathbf{k}$.

Lemma A: If $\mathbf{k} \leqslant \mathbf{d}$ then

$$\{\mathbf{depth}=\mathbf{d}\};\ \mathbf{guard}_d(\mathbf{S}-(\mathbf{k},\mathbf{d}))\ \approx\ \mathbf{guard}_d(\mathbf{S});\ \underline{\mathbf{if}}\ \mathbf{depth} \leqslant \mathbf{0}\ \underline{\mathbf{then}}\ \mathbf{depth{:=}depth}+\mathbf{k}\ \underline{\mathbf{fi}}$$

The proof is similar to the corresponding Lemma for partial incrementation.

The following Lemma is also used in the next Chapter in the theorem on transforming a regular action system to iterative form:

Lemma B: The following are equivalent:
(i) **{x=a}; <u>while</u> x=a <u>do</u> x:=b; S; <u>if</u> x=b <u>then</u> x:=a <u>fi</u> <u>od</u>**.
(ii) **{x=a}; <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u>**.
(iii) **{x=a}; <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>**.
Proof: For (i) ≈ (ii) prove that the **n**th truncations are equivalent and use the induction rule for iteration.

For (iii) ⩽ (ii) we also use induction to prove
{x=a}; <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>^{n}
  ⩽ **{x=a}; <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u>** -the base case (**n=1**) is trivial.
{x=a}; <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>$^{n+1}$
  ≈ **{x=a}; <u>if</u> x=a <u>then</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u>;**
    <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>^{n} **<u>fi</u>**
$\mathbf{n} \geqslant \mathbf{1}$ so {x≠a} ⊢ **<u>while</u> x=a <u>do</u>... <u>od</u>**^{n} ≈ **skip**. So this is
  ≈ **{x=a}; x:=b; <u>while</u> x=b <u>do</u> S <u>od</u>; {x≠b};**
   <u>if</u> x=a <u>then</u> <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>^{n} **<u>fi</u>**
  ⩽ **{x=a}; x:=b; <u>while</u> x=b <u>do</u> S <u>od</u>; {x≠b};**
   <u>if</u> x=a <u>then</u> <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u> <u>fi</u>
by induction hypothesis.
  ≈ **{x=a}; x:=b; <u>while</u> x=b <u>do</u> S <u>od</u>; {x≠b};**
   <u>if</u> x=a ∨ x=b <u>then</u> <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u> <u>fi</u>
  ≈ **{x=a}; x:=b; S; <u>while</u> x=b <u>do</u> S <u>od</u>; <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u>**
by loop unrolling and removing the **<u>if</u>**.
  ≈ **{x=a}; x:=b; S; <u>while</u> x=a ∨ x=b <u>do</u> x:=b; S <u>od</u>** by loop merging.

$\approx$ **{x=a}; <u>while</u> x=a $\vee$ x=b <u>do</u> x:=b; S <u>od</u>** by loop unrolling.

Finally to prove (ii) $\leqslant$ (iii) use induction to prove
{x=a}; <u>while</u> x=a $\vee$ x=b <u>do</u> x:=b; S <u>od</u>^{n}
$\leqslant$ **{x=a}; <u>while</u> x=a <u>do</u> x:=b; <u>while</u> x=b <u>do</u> S <u>od</u> <u>od</u>**

Proof: of double iteration:
{depth=0}; guard$_0$(<u>do</u> <u>do</u> S <u>od</u> <u>od</u>)
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> depth:=2; <u>while</u> depth=2 <u>do</u> guard$_2$(S) <u>od</u> <u>od</u>
{depth=0}; guard$_0$(<u>do</u> S <u>od</u>−1)
$\approx$ **{depth=0}; guard$_0$(<u>do</u> S[T−1/T|τ >0] <u>od</u>)**
$\approx$ **{depth=0}; depth:=1; <u>while</u> depth=1 <u>do</u> guard$_1$(S[T−1/T|τ >0]) <u>od</u>**

Let **IF** $\equiv$ **<u>if</u> depth$\leqslant$0 <u>then</u> depth:=depth+1 <u>fi</u>**. Then by Lemma A:
{depth=0}; guard$_0$(<u>do</u> S <u>od</u>−1)
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> guard$_1$(S); <u>if</u> depth$\leqslant$0 <u>then</u> depth:=depth+1 <u>fi</u> <u>od</u>
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> depth:=2; guard$_2$(S); depth:=depth−1;
<u>if</u> depth$\leqslant$0 <u>then</u> depth:=depth+1 <u>fi</u> <u>od</u>
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> depth:=2; guard$_2$(S);
<u>if</u> depth>1 <u>then</u> depth:=depth−1 <u>fi</u> <u>od</u>
by case analysis on **depth:=depth−1; <u>if</u> depth$\leqslant$0 <u>then</u> depth:=depth+1 <u>fi</u>**.
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> depth:=2; guard$_2$(S);
<u>if</u> depth=2 <u>then</u> depth:=depth−1 <u>fi</u> <u>od</u>
since **depth** can only be decreased.
$\approx$ **{depth=0}; depth:=1;**
<u>while</u> depth=1 <u>do</u> depth:=2; <u>while</u> depth=2 <u>do</u> guard$_2$(S) <u>od</u> <u>od</u>
by Lemma B.
$\approx$ **guard$_0$(<u>do</u> <u>do</u> S <u>od</u> <u>od</u>)**.

<u>Cor:</u> For <u>any</u> **S**:
$\Delta \vdash$ **<u>do</u> <u>do</u> <u>do</u> S <u>od</u> <u>od</u> <u>od</u>** $\approx$ **<u>do</u> <u>do</u> S <u>od</u>−1 <u>od</u>**

Proof: any terminal statement of $\underline{\mathbf{do}}$ **S** $\underline{\mathbf{od}}$ with terminal value one must be a terminal statment of **S** with terminal value two. So reducing such a statement by one gives a terminal statement with terminal value one of **S** and hence a terminal statement of $\underline{\mathbf{do}}$ **S** $\underline{\mathbf{od}}$. Hence $\underline{\mathbf{do}}$ **S** $\underline{\mathbf{od}}$ is reducible and we can apply the last result.
Thus more than two nested $\underline{\mathbf{do}}$ loops around the same statement are never needed.

<u>Loop</u> <u>Absorption:</u>
If $\mathbf{S}_1$ is reducible then
$\Delta \vdash \underline{\mathbf{do}}\ \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}\mathbf{;}\ \mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1[\mathbf{S}_2+\delta\mathbf{+1/T}|\tau=\mathbf{1}]\mathbf{-1}\ \underline{\mathbf{od}}$
Proof: $\underline{\mathbf{do}}\ \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}\mathbf{;}\ \mathbf{S}_2\ \underline{\mathbf{od}}$
$\approx\ \underline{\mathbf{do}}\ \big(\underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}\big)[\mathbf{S}_2+\delta\mathbf{/T}|\tau=\mathbf{0}]\ \underline{\mathbf{od}}$ by absorption.
$\approx\ \underline{\mathbf{do}}\ \underline{\mathbf{do}}\ \mathbf{S}_1[\mathbf{S}_2+\delta\mathbf{+1/T}|\tau=\mathbf{1}]\ \underline{\mathbf{od}}\ \underline{\mathbf{od}}$
Now $\mathbf{S}_1[\mathbf{S}_2+\delta\mathbf{+1/T}|\tau=\mathbf{1}]$ is reducible and has no terminal statement with terminal value zero (by a previous Lemma). So we can apply double iteration to get:
$\underline{\mathbf{do}}\ \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}\mathbf{;}\ \mathbf{S}_2\ \underline{\mathbf{od}}\ \approx\ \underline{\mathbf{do}}\ \mathbf{S}_1[\mathbf{S}_2+\delta\mathbf{+1/T}|\tau=\mathbf{1}]\mathbf{-1}\ \underline{\mathbf{od}}$

This can often be used to replace a double-nested loop by a single loop. For example, suppose we have:
$\underline{\mathbf{do}}\ \mathbf{S}_1\mathbf{;}\ \underline{\mathbf{do}}\ \mathbf{S}_2\ \underline{\mathbf{od}}\mathbf{;}\ \mathbf{S}_3\ \underline{\mathbf{od}}$ where $\mathbf{S}_1$ and $\mathbf{S}_2$ are reducible.
Move the inner loop to the beginning by proper inversion to give:
$\mathbf{S}_1\mathbf{;}\ \underline{\mathbf{do}}\ \underline{\mathbf{do}}\ \mathbf{S}_2\ \underline{\mathbf{od}}\mathbf{;}\ \mathbf{S}_3\mathbf{;}\ \mathbf{S}_1\ \underline{\mathbf{od}}$
Apply loop absorption to give:
$\mathbf{S}_1\mathbf{;}\ \underline{\mathbf{do}}\ \mathbf{S}_2[(\mathbf{S}_3\mathbf{;}\mathbf{S}_1)+\delta\mathbf{+1/T}|\tau=\mathbf{1}]\ \underline{\mathbf{od}}\ \mathbf{-1}$
which is now a single loop.

This can also be used to combine several copies of one statement inside a single loop by transforming it to a double loop. We replace the single loop by a double loop, replace each copy of the statement by an $\underline{\mathbf{exit}}$ of the inner loop and put a single copy of the statement between the two loops, after the inner loop

<u>Lemma</u> <u>1:</u> If $\mathbf{S}'$ is a term of **S** then:
$\Delta \vdash \mathbf{S+1}\ \approx\ \big(\mathbf{S+(1,1)}\big)[\underline{\mathbf{exit}}\mathbf{(k+1)/S'+(1,1)+k}][\mathbf{S}'+\delta\mathbf{+1/T}|\tau=\mathbf{1}][\mathbf{T+1/T}|\tau=\mathbf{0}]$
Proof: We use induction on the structure of **S** and prove the more general result:
$\mathbf{S+1}\ \approx\ \big(\mathbf{S+(1,d)}\big)[\underline{\mathbf{exit}}\mathbf{(k+1)/S'+(1,1)+k}][\mathbf{S}'+\delta\mathbf{+d/T}|\tau=\mathbf{d}][\mathbf{T+1/T}|\tau<\mathbf{d}]$
where $\mathbf{d}\geqslant\mathbf{1}$ and $\mathbf{S}'$ is a $\big(\mathbf{d-1}\big)$-term of **S**. As usual we may assume $\mathbf{S}'$ is compound and not a sequence since otherwise we may replace $\mathbf{S'+k}$ by
$\underline{\mathbf{if}}$ **true** $\underline{\mathbf{then}}$ $\mathbf{S'+k}$ $\underline{\mathbf{fi}}$.

Let $[\mathbf{A}] = [\underline{\mathbf{exit}}(\mathbf{k+1})/\mathbf{S'}+(\mathbf{1,1})+\mathbf{k}]$, $[\mathbf{B}] = [\mathbf{S'}+\delta+\mathbf{d}/\mathbf{T}|\tau=\mathbf{d}]$, $[\mathbf{C}] = [\mathbf{T}+\mathbf{1}/\mathbf{T}|\tau<\mathbf{d}]$.
The only difficult case is:
<u>Case</u> (iv): $\mathbf{S} = \mathbf{S}_1;\mathbf{S}_2$ and $\mathbf{d}>\mathbf{1}$.
$\big(\mathbf{S}+(\mathbf{1,d})\big)[\mathbf{A}][\mathbf{B}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\tau<\mathbf{d}]$
$\quad\approx\ \big(\mathbf{S}_1+(\mathbf{1,d})\big)[\mathbf{A}][\mathbf{B}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\mathbf{0}<\tau<\mathbf{d}];\ \big(\mathbf{S}_2+(\mathbf{1,d})\big)[\mathbf{A}][\mathbf{B}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\tau<\mathbf{d}]$
Claim: If $\mathbf{d}>\mathbf{1}$ then $\big(\mathbf{S}_1+(\mathbf{1,d})\big)[\mathbf{A}][\mathbf{B}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\mathbf{0}<\tau<\mathbf{d}]\ \approx\ \mathbf{S}_1+(\mathbf{1,1})$
Then by induction hypothesis
$\big(\mathbf{S}+(\mathbf{1,d})\big)[\mathbf{A}][\mathbf{B}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\tau<\mathbf{d}]$
$\quad\approx\ \mathbf{S}_1+(\mathbf{1,1});\ \mathbf{S}_2+\mathbf{1}\ \approx\ \big(\mathbf{S}_1;\mathbf{S}_2\big)+\mathbf{1}\ \approx\ \mathbf{S}+\mathbf{1}$.
Proof of Claim: Prove by induction on the structure of $\mathbf{S}$ that:
If $\mathbf{d}>\mathbf{1}$ and $\mathbf{0}<\mathbf{l}<\mathbf{d}$ and $\mathbf{S'}$ is a $\big(\mathbf{d}-\mathbf{1}\big)$-term of $\mathbf{S}$ then
$\quad\Delta\vdash\big(\mathbf{S}+(\mathbf{1,d})\big)[\underline{\mathbf{exit}}(\mathbf{k+1})/\mathbf{S'}+(\mathbf{1,1})+\mathbf{k}][\mathbf{S'}+\delta+\mathbf{d}/\mathbf{T}|\tau=\mathbf{d}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\mathbf{l}<\tau<\mathbf{d}]\ \approx\ \mathbf{S}+(\mathbf{1,l})$

This claim is also used in <u>Case</u> (vi): $\mathbf{S} = \underline{\mathbf{do}}\ \mathbf{S}_1\ \underline{\mathbf{od}}$.

<u>**Lemma**</u> <u>**2:**</u> <u>**do**</u> <u>**do**</u> **S;** <u>**exit**</u> <u>**od**</u> <u>**od**</u> $\approx$ <u>**do**</u> <u>**do**</u> **S** <u>**od**</u> <u>**od**.</u>
Proof: Consider:
{depth=0}; depth:=1; <u>**while**</u> **depth=1** <u>**do**</u>
depth:=2; <u>**while**</u> **depth=2** <u>**do**</u> **guard**$_2$**(S);**
<u>**if**</u> **depth=2** <u>**then**</u> **depth:=1** <u>**fi**</u> <u>**od**</u> <u>**od**</u>**.**
By Lemma B ((iii)⇒(i))this is equivalent to:
{depth=0}; depth:=1; <u>**while**</u> **depth=1** <u>**do**</u>
depth:=2; guard$_2$**(S);** <u>**if**</u> **depth=2** <u>**then**</u> **depth:=1** <u>**fi**</u>**; {depth≠2}**
<u>**if**</u> **depth=2** <u>**then**</u> **depth:=1** <u>**fi**</u> <u>**od**</u>**.**
The second <u>**if**</u> can be removed and then Lemma B ((i)⇒(iii)) gives:
{depth=0}; depth:=1; <u>**while**</u> **depth=1** <u>**do**</u>
depth:=2; <u>**while**</u> **d=2** <u>**do**</u> **guard**$_2$**(S)** <u>**od**</u> <u>**od**</u>**.**
Hence the result.

<u>**Loop**</u> <u>**Expansion:**</u>
If $\mathbf{S'}$ is a term of $\mathbf{S}$ then
$\Delta\vdash$ <u>**do**</u> **S** <u>**od**</u> $\approx$ <u>**do**</u> <u>**do**</u> $\big(\mathbf{S}+(\mathbf{1,1})\big)[\underline{\mathbf{exit}}(\mathbf{k+1})/\mathbf{S'}+(\mathbf{1,1})+\mathbf{k}]$ <u>**od**</u>**; S′** <u>**od**</u>
Proof: <u>**do**</u> **S** <u>**od**</u> $\approx$ <u>**do**</u> <u>**do**</u> **S+1** <u>**od**</u> <u>**od**</u> by false iteration.
$\approx$ <u>**do**</u> <u>**do**</u> $(\mathbf{S}+(\mathbf{1,1}))[\mathbf{A}][\mathbf{S'}+\delta+\mathbf{1}/\mathbf{T}|\tau=\mathbf{1}][\mathbf{T}+\mathbf{1}/\mathbf{T}|\tau=\mathbf{0}]$ <u>**od**</u> <u>**od**</u> by Lemma 1.
$\approx$ <u>**do**</u> <u>**do**</u> $(\mathbf{S}+(\mathbf{1,1}))[\mathbf{A}][\mathbf{S'}+\delta+\mathbf{1}/\mathbf{T}|\tau=\mathbf{1}]$**;** <u>**exit**</u> <u>**od**</u> <u>**od**</u> by the inverse of absorption.
$\approx$ <u>**do**</u> <u>**do**</u> $(\mathbf{S}+(\mathbf{1,1}))[\mathbf{A}][\mathbf{S'}+\delta+\mathbf{1}/\mathbf{T}|\tau=\mathbf{1}]$ <u>**od**</u> <u>**od**</u> by Lemma 2.

$\approx$ **<u>do</u> <u>do</u> (S+(1,1))[A] <u>od</u>; S′ <u>od</u>** by the inverse of absorption.

The following Lemma provides a form of induction rule for unbounded loops:

<u>Lemma:</u> If $\Delta \vdash$ **<u>do</u> (S;)^n; abort <u>od</u>** $\leqslant$ **S′** for all **n**$< \omega$ then **<u>do</u> S <u>od</u>** $\leqslant$ **S′**.
Proof: Prove by induction on **n**:
{depth=k}; guard$_k$(<u>do</u> (S;)^n; abort <u>od</u>) $\approx$ **{depth=k}; depth:=k+1;**
<u>while</u> depth=k+1 <u>do</u> guard$_{k+1}$(S) <u>od</u>^{n}
Then the induction rule for loops gives the result.