Acta Informatica 24, 1–31 (1987)

Sometime = Always + Recursion ≡ Always on the Equivalence of the Intermittent and Invariant Assertions Methods for Proving Inevitability Properties of Programs

Patrick Cousot[1] and Radhia Cousot[2]

[1] Ecole Polytechnique, Centre de Mathématiques Appliquées, F-91128 Palaiseau Cedex, France
[2] Université de Paris-Sud, Centre d'Orsay, LRI, Bat. 490, F-91405 Orsay Cedex, France

Summary. We propose and compare two induction principles called "always" and "sometime" for proving inevitability properties of programs. They are respective formalizations and generalizations of Floyd invariant assertions and Burstall intermittent assertions methods for proving total correctness of sequential programs whose methodological advantages or disadvantages have been discussed in a number of previous papers. Both principles are formalized in the abstract setting of arbitrary nondeterministic transition systems and illustrated by appropriate examples. The "sometime" method is interpreted as a recursive application of the "always" method. Hence "always" can be considered as a special case of "sometime". These proof methods are strongly equivalent in the sense that a proof by one induction principle can be rewritten into a proof by the other one. The first two theorems of the paper show that an invariant for the "always" method can be translated into an invariant for the "sometime" method even if every recursive application of the later is required to be of finite length. The third and main theorem of the paper shows how to translate an invariant for the "sometime" method into an invariant for the "always" method. It is emphasized that this translation technique follows the idea of transforming recursive programs into iterative ones. Of course, a general translation technique does not imply that the original "sometime" invariant and the resulting "always" invariant are equally understandable. This is illustrated by an example.

1. Introduction

We compare two induction principles (that we call "always" and "sometime") for proving inevitability properties of transition systems, the nondeterminism of which can be unbounded. These induction principles are formalizations of program proof methods independently of a particular programming language, of a particular inevitability property and of a particular style of presentation of proof methods. Thanks to these abstract and concise formalizations we can make more rigorous comparisons of program proof methods which for some time have always been compared using methodological arguments based on examples [8, 12].

We introduce the induction principle called "always" in [4]. It is a generalization of those program proof methods in the class of Floyd [7] invariant assertions method for proving total correctness of sequential programs. We introduce the induction principle called "sometime" in [5]. It is an abstract generalization of Burstall [2] intermittent assertions method for proving total correctness of sequential programs.

The "always" induction principle involves an induction along execution traces whereas the "sometime" induction principle involves a combination of an induction along (parts of) execution traces (connected to Burstall's "hand simulation") and recursion induction (related to Burstall's "induction on the data"). Hence the "always" induction principle corresponds to the particular case of the "sometime" induction principle when recursion is not used.

The "always" and "sometime" induction principles are sound and semantically complete (see the proofs respectively in [4, 5]) hence weakly equivalent in the sense that when a proof exists by one method, a proof must exist by the other method. Here we prove a stronger equivalence result, in the sense that whenever a proof exists by one method it can be rewritten into a proof by the other method. Intuitively this is because recursion can be eliminated from proofs in favour of induction in just the same way as it can be eliminated from programs in favour of iteration.

We illustrate our techniques by examples.

2. Inevitability Properties of Programs Represented as Transition Systems

To introduce abstract formalizations of program proof methods, we view programs as (nondeterministic) transition systems $\langle S, t\rangle$ [9] where S is a nonempty set of states and the transition relation t between a state and its possible successors is represented as a function $t \in (S \times S \to \{tt, ff\})$ from pairs of states into truth values (tt is true and ff is false).

We say that the nondeterminism of t is *bounded* when states can only have a finite number of successor states that is to say when $\forall s \in S. |\{s' \in S: t(s, s')\}| \in \omega$ (where $|E|$ is the cardinal of a class E and ω is the set of natural numbers) and *unbounded* otherwise. There is no need for the nondeterminism of t to be bounded. Hence we can represent parallel programs and even fair ones as transition systems by decomposition of these programs into their atomic actions. This decomposition usually destroys syntactic structure of programs and is therefore a debatable way of proceeding. However we neither suggest that program proofs should make a direct use of our induction principles nor that programs should be represented as transition systems before initiating the proofs. On the contrary we attach a great importance to the presentation of proofs. In [3] we have proposed a systematic method for constructing proof methods from induction principles, so that, in particular, the structural information about the original program need not be lost in the proof method (although it might have been in case of direct use of the induction principle).

Example 2.1 (The transition system corresponding to the tree traversing program). The following sequential program to traverse a binary tree and count its tips is taken (literally) from Burstall [2]. It will be used as example throughout the paper.

The value of variable Tr of type tree is either nil or (lf(Tr).rg(Tr)) where lf(Tr) and rg(Tr) are trees, the value of Co of type nat is a natural number and the value of variable St of type stack is either ( ) or (hd(St).tl(St)) where hd(St) is a tree and tl(St) is a stack.

```
Start: St:=( ); Co:=0;
Loop:  if Tr ≠ nil
          then begin Push Tr onto St;
                     Tr:=lf(Tr); goto Loop
               end
          else begin Co:=Co+1;
                     if St=( ) then goto Finish;
                     Pop Tr from St;
                     Tr:=rg(Tr); goto Loop
               end;
Finish:
```

We will use capital letters Tr, ... for program variables and small letters possibly with quotes tr, tr′, tr″, ... for their values at different time instants.

The transition system $\langle S, t\rangle$ corresponding to that program is defined by:

$$S = \{\text{Start}, \text{Loop}, \text{Finish}\} \times \text{tree} \times \text{nat} \times \text{stack}$$

$t(\langle l', \text{tr}', \text{co}', \text{st}'\rangle, \langle l'', \text{tr}'', \text{co}'', \text{st}''\rangle) =$
$[(l' = \text{Start} \wedge l'' = \text{Loop} \wedge \text{tr}'' = \text{tr}' \wedge \text{co}'' = 0 \wedge \text{st}'' = ())$
$\vee (l' = \text{Loop} \wedge \text{tr}' \neq \text{nil} \wedge l'' = \text{Loop} \wedge \text{tr}'' = \text{lf}(\text{tr}') \wedge \text{co}'' = \text{co}' \wedge \text{st}'' = (\text{tr}'.\text{st}'))$
$\vee (l' = \text{Loop} \wedge \text{tr}' = \text{nil} \wedge \text{st}' \neq () \wedge l'' = \text{Loop} \wedge \text{tr}'' = \text{rg}(\text{hd}(\text{st}')) \wedge$
$\text{co}'' = \text{co}' + 1 \wedge \text{st}'' = \text{tl}(\text{st}'))$
$\vee (l' = \text{Loop} \wedge \text{tr}' = \text{nil} \wedge \text{st}' = () \wedge l'' = \text{Finish} \wedge \text{tr}'' = \text{tr}' \wedge \text{co}'' = \text{co}' + 1$
$\wedge \text{st}'' = \text{st}')]$. □

Executions of a program $\langle S, t\rangle$ shall be modelled by its set $\Sigma\langle S, t\rangle$ of complete execution traces $p_0, p_1, p_2, \ldots$ where the p_i are states, there is a transition from p_i to p_{i+1} and the sequence is either infinite or terminates in a state p_n which has no possible successor. More formally,

— 0 is the empty set or zero,
— If $n \in \omega$ and $n \neq 0$ then n will denote $\{0, \ldots n-1\}$,
(so that $m \in n$ is equivalent to $m < n$)

— If E is a set then $E \sim x = \{y \in E : y \neq x\}$,
— $D \dashrightarrow R$ is the class of partial functions from D into R,
— $D \to R$ is the class of total functions from D into R,
— $\Sigma^0\langle S, t\rangle = 0$
Empty traces are not considered,
— $\Sigma^n\langle S, t\rangle = \{p \in (n \to S) : \forall i \in (n-1). t(p_i, p_{i+1}) \wedge \forall s \in S. \neg t(p_{n-1}, s)\}$
Finite traces of length $n > 0$ (p_i is short for $p(i)$),
— $\Sigma^\omega\langle S, t\rangle = \{p \in (\omega \to S) : \forall i \in \omega. t(p_i, p_{i+1})\}$
Infinite traces,
— $\Sigma\langle S, t\rangle = \bigcup_{n \in \omega} \Sigma^n\langle S, t\rangle \cup \Sigma^\omega\langle S, t\rangle$
Complete traces.

A relation $\psi \in (S \times S \to \{tt, ff\})$ is inevitable for a program $\langle S, t\rangle$ if any program execution eventually leads to a state that is related to the initial state by ψ.

More formally, $\psi \in (S \times S \to \{tt, ff\})$ is *inevitable* for $\langle S, t\rangle$ if and only if

$$\forall p \in \Sigma\langle S, t\rangle . \exists i \in Dom(p). \psi(p_0, p_i) \tag{1}$$

(where $Dom(p) = \{x : \exists y. (y = p(x))\}$ is the domain of function p).

Example 2.2 (*Total correctness as an inevitability property*). Total correctness of program 2.1 can be specified by the inevitability of ψ such that:

$$\psi((l, \text{tr}, \text{co}, \text{st}), (l', \text{tr}', \text{co}', \text{st}')) = [(l = \text{Start}) \Rightarrow (l' = \text{Finish} \wedge \text{co}' = \text{tips}(\text{tr}))]$$

where

$$\text{tips}(\text{tr}) = \textit{if } \text{tr} = \text{nil } \textit{then } 1 \textit{ else } \text{tips}(\text{lf}(\text{tr})) + \text{tips}(\text{rg}(\text{tr})). \quad \square$$

Extending Dijkstra's definitions of strong and weak termination [6] (see also Back [1], Gries [8]), we say that inevitability is *strong* when the number i of program steps necessary for reaching the "final" state p_i is bounded by an integer k depending only on the "initial" state p_0.

More formally, $\psi \in (S \times S \to \{tt, ff\})$ is *strongly inevitable* for $\langle S, t\rangle$ if and only if

$$\forall s \in S. \exists k \in \omega. \forall p \in \Sigma\langle S, t\rangle . [(p_0 = s) \Rightarrow (\exists i \leq k. \psi(p_0, p_i))]$$

$\psi \in (S \times S \to \{tt, ff\})$ is *weakly inevitable* for $\langle S, t\rangle$ if and only if ψ is inevitable for $\langle S, t\rangle$ but not strongly.

3. The "Always" Induction Principle Generalizing Floyd's Invariant Assertions Proof Method

Floyd's total correctness proof method for sequential programs [7] was generalized to inevitability proofs for nondeterministic transition systems $\langle S, t\rangle$ and formalized by a number of strongly equivalent induction principles in Cousot and Cousot [4], the most abstract one being:

$$[\exists \Gamma \in Ord, J \in (\Gamma \times S \times S \to \{tt, ff\}). \tag{2}$$

$$\begin{array}{ll} (\text{F.1}) & (\forall s \in S . \exists \gamma \in \Gamma . J(\gamma, s, s)) \\ (\text{F.2}) & \wedge (\forall s, s' \in S, \gamma' \in \Gamma . \\ & \quad J(\gamma', s, s') \Rightarrow \\ (\text{F.2.a}) & \quad [(\exists s'' \in S . t(s', s'') \wedge \forall s'' \in S . [t(s', s'') \Rightarrow \exists \gamma'' < \gamma' . J(\gamma'', s, s'')]) \\ (\text{F.2.b}) & \quad \vee \psi(s, s')])] \end{array}$$

where $(Ord, <)$ is the class of ordinals.

This induction principle is better understood by first considering the case when the nondeterminism of t is bounded. Then one can choose $\Gamma = \omega$ (instead of $\Gamma \in Ord$) in induction principle (2) and prove strong inevitability. More precisely, the induction hypothesis $J(\gamma', s, s')$ is true when "current" state s' is a descendant of "initial" state s and from s' on, execution will lead in at most γ' steps to some "final" state s'' satisfying $\psi(s, s'')$. In particular by F.1, for any initial state s there is an integer γ such that execution for this initial state s is guaranteed to lead to some final state (s'' satisfying $\psi(s, s'')$) in less than γ steps. Moreover by F.2, a descendant s' of an initial state s which is not a final state must have a successor state s'' and all its successor states must be closer to the goal ψ.

We now show that induction principle (2) is a model of Floyd's "always" proof method using the following:

Example 3.1 (*Proof of the tree traversing program using the "always" induction principle*). 1. Floyd's method requires three separate proofs to establish total correctness, one to show partial correctness, the other to show termination and the last to show absence of run-time errors.

(i) The partial correctness proof of program 2.1 first consists in discovering intermediate assertions attached to the control points of the program. These assertions should be local invariants that is to say express relationships that hold between the initial values tr, co, st of variables Tr, Co, St and the values tr′, co′, st′ of these variables Tr, Co, St whenever control passes through the corresponding points:

$$I_{\text{Start}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') = [\text{tr}' = \text{tr} \wedge \text{co}' = \text{co} \wedge \text{st}' = \text{st}]$$

$$I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') = [(\text{tips}(\text{tr}') + \text{co}' + Sum(\text{tips} \circ \text{rg}, \text{st}')) = \text{tips}(\text{tr})]$$

$$I_{\text{Finish}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') = [\text{co}' = \text{tips}(\text{tr})]$$

where

$$f \circ g(x) = f(g(x)) \quad \text{and if } f \in (\text{tree} \to \omega) \text{ and st is a stack then}$$

$$Sum(f, \text{st}) = \mathit{if}\ \text{st} = (\,)\ \mathit{then}\ 0\ \mathit{else}\ f(\text{hd}(\text{st})) + Sum(f, \text{tl}(\text{st})).$$

Observe that these local invariants on values of program variables are equivalent to the following global invariant on program states:

$$I(\langle l, \text{tr}, \text{co}, \text{st}\rangle, \langle l'\ \text{tr}', \text{co}', \text{st}'\rangle) = [(l = \text{Start}) \Rightarrow I_{l'}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}')]$$

in the sense that during execution $I(s, s')$ holds between the initial state s and the current state s'. More precisely, if we let t^* be the reflexive transitive closure of t, we have

$$\forall s, s' \in S . [t^*(s, s') \Rightarrow I(s, s')].$$

These assertions are then shown to satisfy verification conditions which (by induction on the number of program execution steps) imply that they are invariants:

– The local assertion attached to the program entry point Start must hold when the current state $\langle \text{tr}', \text{co}', \text{st}' \rangle$ is equal to the initial state $\langle \text{tr}, \text{co}, \text{st} \rangle$ of the variables so that

$$I_{\text{Start}}(\text{tr}, \text{co}, \text{st}, \text{tr}, \text{co}, \text{st})$$

must be true.

– If the local assertion attached to program point l' holds and control goes from point l' to l'' then the assertion attached to l'' must hold, so that (according to definition 2.1 of t) we obtain the following verification conditions:

$$I_{\text{Start}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \Rightarrow I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{tr}', 0, ())$$

$$[I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \wedge \text{tr}' \neq \text{nil}] \Rightarrow I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{lf}(\text{tr}'), \text{co}', (\text{tr}' . \text{st}'))$$

$$[I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \wedge \text{tr}' = \text{nil} \wedge \text{st}' \neq ()] \Rightarrow$$
$$I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{rg}(\text{hd}(\text{st}')), \text{co}' + 1, \text{tl}(\text{st}'))$$

$$[I_{\text{Loop}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \wedge \text{tr}' = \text{nil} \wedge \text{st}' = ()] \Rightarrow I_{\text{Finish}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}' + 1, \text{st}').$$

– Finally, the local assertion attached to the program exit point must imply the input-output specification:

$$I_{\text{Finish}}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \Rightarrow [\text{co}' = \text{tips}(\text{tr})].$$

Observe that these local verification conditions are equivalent to the following global one:

$$(\forall s \in S . I(s, s))$$
$$\wedge (\forall s, s', s'' \in S . [I(s, s') \wedge t(s', s'')] \Rightarrow I(s, s'')).$$

(ii) According to Floyd [7], "proofs of termination are dealt with by showing that each step of the program decreases some entity which cannot decrease indefinitely". Therefore we associate with each point l of the program a function f_l of the values tr, co, st of the program variables Tr, Co, St taking its values in the well-founded set $(\omega, <)$:

$$f_{\text{Start}}(\text{tr}, \text{co}, \text{st}) = \text{size}(\text{tr}) + 1$$

$$f_{\text{Loop}}(\text{tr}, \text{co}, \text{st}) = \text{size}(\text{tr}) + Sum(\text{size} \circ \text{rg}, \text{st})$$

$$f_{\text{Finish}}(\text{tr}, \text{co}, \text{st}) = 0$$

where

$$\text{size}(\text{tr}) = \textit{if}\ \text{tr} = \text{nil}\ \textit{then}\ 1\ \textit{else}\ 1 + \text{size}(\text{lf}(\text{tr})) + \text{size}(\text{rg}(\text{tr}))$$

and show that after each execution of a command (i.e. after each transition $t(\langle l', \text{tr}', \text{co}', \text{st}'\rangle, \langle l'', \text{tr}'', \text{co}'', \text{st}''\rangle)$) the current value of the $f_{l''}$-function associated with the exit state $\langle l'', \text{tr}'', \text{co}'', \text{st}''\rangle$ is less than the prior value of the $f_{l'}$-function associated with the entrance state $\langle l', \text{tr}', \text{co}', \text{st}'\rangle$:

$$f_{\text{Start}}(\text{tr}', \text{co}', \text{st}') > f_{\text{Loop}}(\text{tr}'', \text{co}'', \text{st}'')$$
$$\text{because } [\text{tr}'' = \text{tr}' \wedge \text{co}'' = 0 \wedge \text{st}'' = ()]$$

$$f_{\text{Loop}}(\text{tr}', \text{co}', \text{st}') > f_{\text{Loop}}(\text{tr}'', \text{co}'', \text{st}'')$$
$$\text{because } [\text{tr}' \neq \text{nil} \wedge \text{tr}'' = \text{lf}(\text{tr}') \wedge \text{co}'' = \text{co}' \wedge \text{st}'' = (\text{tr}'.\text{st}')]$$
$$\text{or } [\text{tr}' = \text{nil} \wedge \text{st}' \neq () \wedge \text{tr}'' = \text{rg}(\text{hd}(\text{st}')) \wedge \text{co}'' = \text{co}' + 1 \wedge \text{st}'' = \text{tl}(\text{st}')]$$

$$f_{\text{Loop}}(\text{tr}', \text{co}', \text{st}') > f_{\text{Finish}}(\text{tr}'', \text{co}'', \text{st}'')$$
$$\text{because } [\text{tr}' = \text{nil} \wedge \text{st}' = () \wedge \text{tr}'' = \text{tr}' \wedge \text{co}'' = \text{co}' + 1 \wedge \text{st}'' = \text{st}'].$$

Then each transition decreases the entity $\gamma' = f(\langle l', \text{tr}', \text{co}', \text{st}'\rangle) = f_{l'}(\text{tr}', \text{co}', \text{st}')$.

(iii) When programs contain partial operations, a proof of absence of runtime errors must show that they do not lead to undefined results. If we use the convention that states on which partial operations would lead to undefined results have no successors by the transition relation t then a proof of absence of run-time errors amounts to a proof that accessible states which are not final ones do have at least one successor. The same way total correctness of parallel programs excludes global permanent deadlocks. Again in such a case we must prove the absence of blocking states (that is to say states with no possible successor).

Since program 2.1 contains no partial operation, it obviously never leads to a blocking state so that for $l' \in \{\text{Start}, \text{Loop}\}$ we have:

$$I_{l'}(\text{tr}, \text{co}, \text{st}, \text{tr}', \text{co}', \text{st}') \Rightarrow \exists l'', \text{tr}'', \text{co}'', \text{st}''.\, t(\langle l', \text{tr}', \text{co}', \text{st}'\rangle, \langle l'', \text{tr}'', \text{co}'', \text{st}''\rangle).$$

2. To sum up (i), (ii) and (iii), Floyd's verification conditions are equivalent to (2) choosing

$$J(\gamma', \langle l, \text{tr}, \text{co}, \text{st}\rangle, \langle l', \text{tr}', \text{co}', \text{st}'\rangle) = [(l = \text{Start}) \Rightarrow (\gamma' = f(\langle l', \text{tr}', \text{co}', \text{st}'\rangle)$$
$$\wedge I(\langle l, \text{tr}, \text{co}, \text{st}\rangle, \langle l', \text{tr}', \text{co}', \text{st}'\rangle))]. \quad \square$$

Observe that in induction principle (2) the entity γ' which is decreased by each program step depends on the current state s' but also on the initial state s. The dependence of γ' on s is a necessity as shown by the following:

Example 3.2. (*Termination functions*). If we choose $S = \omega$, $t(s', s'') = [s'' = s' + 1]$ and $\psi(s, s') = [s' = 2s]$ there are no well-founded class $(\omega, \prec)$ and termination function $f \in (S \to \omega)$ such that $t(s', s'')$ implies $f(s'') \prec f(s')$ since otherwise $\forall x \in \omega.(f(x+1) \prec f(x))$.

However we can prove that ψ is inevitable for $\langle S, t\rangle$ using (2) with $\Gamma=\omega$ and $J(\gamma', s, s')=[\gamma'=(2s-s')\wedge s\leq s'\leq 2s]$. $\square$

We use the class *Ord* of ordinals as an abstraction of common properties of well-founded classes. Roughly speaking, the ordinals are obtained by extending the sequence of natural numbers: $0<1<2\ldots<\omega<\omega\dot{+}1<\omega\dot{+}2<\ldots<\omega\dot{+}\omega=\omega\dot{\times}2<\omega\dot{\times}2\dot{+}1<\omega\dot{\times}2\dot{+}2<\ldots<\omega\dot{\times}2\dot{+}\omega=\omega\dot{\times}3<\omega\dot{\times}3\dot{+}1<\ldots<\omega\dot{\times}\omega<\ldots$ Let us recall [13] that for all ordinals $\alpha, \beta\in Ord$ we have $\alpha=\{\beta\in Ord: \beta<\alpha\}$ and $\beta<\alpha$ if and only if $\beta\in\alpha$. Let X be a class of ordinals. Sup $X=\cup X$ will denote the least upper bound of X and $\mathrm{Sup}^+ X$ will denote the least strict upper bound of X. Ordinal addition $\dot{+}$ and multiplication $\dot{\times}$ correspond to operations on well-orderings. $\alpha\dot{+}\beta$ is ordered by first putting all elements of α in their natural order and then adjoining the elements of β in their natural order. Consequently, if $\alpha<\beta$ then $\gamma\dot{+}\alpha<\gamma\dot{+}\beta$ whereas $\alpha\dot{+}\gamma\leq\beta\dot{+}\gamma$. $\alpha\dot{\times}\beta$ is ordered by substituting a copy of α for each element of β so that $(\alpha\dot{\times}\beta, <)$ is isomorphic to $(\alpha\times\beta, <_2)$ where the right lexicographic ordering $<_2$ on $\alpha\times\beta$ is defined by $\langle y', x'\rangle <_2 \langle y, x\rangle$ if and only if $(x'<x)\vee(x'=x\wedge y'<y)$.

If $(W, \prec)$ is a well-founded class, the rank function ρ on W is defined by transfinite recursion as:

$$\rho(\gamma)=\mathrm{Sup}^+\{\rho(\gamma'): \gamma'\prec\gamma\}.$$

Then obviously by transfinite induction on W, $\rho(\gamma)$ is an ordinal for every γ and $\gamma'\prec\gamma$ implies $\rho(\gamma')<\rho(\gamma)$.

We can now show that Floyd's use of a global invariant $I\in(S\times S\to\{tt, ff\})$ and a termination function $f\in(S\times S\to W)$ on a well-founded class $(W, \prec)$ is equivalent to induction principle (2).

On one hand, we define $\Gamma=\mathrm{Sup}^+\{\rho(f(s, s')): s, s'\in S\}$ and $J(\gamma', s, s')=[I(s,s')\wedge\gamma'=\rho(f(s,s'))]$ so that $f(s,s'')\prec f(s,s')$ implies $\gamma''=\rho(f(s,s''))<\rho(f(s,s'))=\gamma'$.

On the other hand, we choose $(W, \prec)$ as $(\Gamma, <)$ and define $f(s, s')=0$ if there is no $\gamma'\in\Gamma$ such that $J(\gamma', s, s')$ holds or else $f(s, s')=\cap\{\gamma'\in\Gamma: J(\gamma', s, s')\}$ is the smallest $\gamma'\in\Gamma$ such that $J(\gamma', s, s')$ holds. $I(s, s')$ is chosen as $[\exists\gamma'\in\Gamma.J(\gamma', s, s')]$. It follows that $I(s, s')\wedge t(s', s'')$ implies $\exists\gamma'\in\Gamma.J(\gamma', s, s')$ whence $J(f(s, s'), s, s')$ so that by (2) we have $\exists\gamma''<f(s, s').J(\gamma'', s, s'')$ and therefore $I(s, s'')$ and $f(s, s'')<f(s, s')$ hold.

When the nondeterminism of t is bounded, we may prove strong termination using $\Gamma=\omega$. Then $J(\gamma', s, s')$ means that if execution starts in state s and reaches state s' then, a goal state s'' such that $\psi(s, s'')$ holds will be reached in at most γ' steps. When the nondeterminism of t is unbounded, we may only be able to prove weak inevitability using well-founded classes $(W, \prec)$ of order type $\mathrm{Sup}\{\rho(\gamma): \gamma\in W\}$ greater than ω. Such an example is provided by Dijkstra [6]:

Example 3.3 (*Weak termination*). The following program (where X_c and Y_c denote natural constants):

```
X, Y := X_c, Y_c;
do X > 0 → X, Y := X − 1, any natural number
□  Y > 0 → Y := Y − 1
od
```

does not enjoy the property of strong termination, because for $X_c > 0$ no upper bound for Y can be given.

This program can be represented as a transition system:

$$S = \omega \times \omega$$

$$t(\langle x, y\rangle, \langle x', y'\rangle) = ((x > 0 \wedge x' = x - 1) \vee (y > 0 \wedge x' = x \wedge y' = y - 1)).$$

Weak termination of this program follows from the fact that $t(\langle x, y\rangle, \langle x', y'\rangle) \Rightarrow \langle x', y'\rangle \ {}_2\!< \ \langle x, y\rangle$ when the left lexicographic ordering ${}_2\!<$ is defined by $(x' < x) \vee (x' = x \wedge y' < y)$.

Equivalently and more abstractly, since $(\omega \times \omega, {}_2\!<)$ is isomorphic to $(\omega \dot{\times} \omega, <)$ for the isomorphism $F(\langle x, y\rangle) = (\omega \dot{\times} x) \dot{+} y$ we can apply induction principle (2) choosing:

$$\psi(\langle x, y\rangle, \langle x', y'\rangle) = [x' = 0 \wedge y' = 0]$$

$$\Gamma = \omega \dot{\times} \omega$$

$$J(\gamma', \langle x, y\rangle, \langle x', y'\rangle) = [\gamma' = (\omega \dot{\times} x') \dot{+} y'].$$

So as to prove (F.2.a) we observe that when $x > 0$ and $x' = x - 1$ we have $y' < \omega$ so that $\omega \dot{\times} x' \dot{+} y' = \omega \dot{\times} (x - 1) \dot{+} y' < \omega \dot{\times} (x - 1) \dot{+} \omega = \omega \dot{\times} x \leqq \omega \dot{\times} x \dot{+} y$. When $y > 0$, $x' = x$ and $y' = y - 1$ we have $y - 1 < y$ so that $\omega \dot{\times} x' \dot{+} y' = \omega \dot{\times} x \dot{+} (y - 1) < \omega \dot{\times} x \dot{+} y$. □

When considering unbounded nondeterminism, we shall subsequently misuse terms relevant to bounded nondeterminism. However the analogy is often a good support for intuition as in "$J(\gamma', s, s')$ implies that if execution goes on from state s', a goal state s'' satisfying $\psi(s, s'')$ will be reached in at most γ' steps".

It is proved in Cousot and Cousot [4] that induction principle (2) is sound i.e. (2) ⇒ (1) and semantically complete i.e. (1) ⇒ (2).

4. The "Sometime" Induction Principle Generalizing Burstall's Intermittent Assertions Proof Method

Burstall's total correctness proof method for sequential programs [2] was generalized to inevitability proofs for nondeterministic transition systems $\langle S, t\rangle$ and formalized by a number of strongly equivalent induction principles in Cousot and Cousot [5], the one we shall subsequently use being:

$$[\exists \Lambda \in Ord, \theta \in (\Lambda \to (S \times S \to \{tt, ff\})), \pi \in \Lambda, \Delta \in Ord, \quad I \in (\Delta \times S \times S \to \{tt, ff\})). \tag{3}$$

(B.1) $\qquad \theta_\pi = \psi$

(B.2) $\qquad \wedge (\forall \lambda \in \Lambda. \forall s \in S. \exists \delta \in \Delta. I_\lambda(\delta, s, s))$

(B.3) $\wedge(\forall \lambda \in \Lambda, s, s' \in S, \delta' \in \Delta.$
$I_\lambda(\delta', s, s') \Rightarrow$
(B.3.a) $[(\exists s'' \in S . t(s', s'') \wedge \forall s'' \in S . [t(s', s'') \Rightarrow$
$(\exists \delta'' < \delta' . I_\lambda(\delta'', s, s''))])$
(B.3.b) $\vee(\exists \lambda' < \lambda . \forall s'' \in S . [\theta_{\lambda'}(s', s'') \Rightarrow (\exists \delta'' < \delta' . I_\lambda(\delta'', s, s''))])$
(B.3.c) $\vee\, \theta_\lambda(s, s')]])$.

The above induction principle (3) consists in proving a theorem of inevitability of ψ for a transition system $\langle S, t\rangle$ by proving a number of inevitability theorems for θ_λ, $\lambda \in \Lambda$, one of these being ψ (B.1). If for the moment, we ignore (B.3.b) we observe that (B.2) corresponds to (F.1) and (B.3) to (F.2) so that the inevitability of each θ_λ, $\lambda \in \Lambda$ can simply be established using induction principle (2). However (B.3.b) introduces the possibility of using "already" proved theorems for $\theta_{\lambda'}$ when proving the inevitability theorem for θ_λ. Since $\lambda' < \lambda$ and $(\Lambda, <)$ is a well-ordering we take precautions for avoiding circular proofs. By (B.3) we prove that a descendant s' of an initial state s which is not a final state must have a successor state s'' and all its successor states must be closer to the goal ψ. In (B.3.a) we consider all successors s'' of current state s' after a single execution step as described by t. In (B.3.b) we consider all successors s'' of current state s' after zero or more steps as described by the inevitability theorem for $\theta_{\lambda'}$. In both cases the successor state s'' is closer to the goal since $\delta'' < \delta'$. Notice that the existence of at least one successor state s'' when the goal is not yet reached must be established in (B.3.a) whereas it has already been proved in (B.3.b) (when $\theta_{\lambda'}$ was shown to be inevitable).

This induction principle is a generalization of Burstall's intermittent assertions method for proving total correctness of sequential programs. The proof of each inevitability theorem for θ_λ starts from (B.2) and proceeds by successive deductions by "hand-simulation" (B.3.a) or using "a little induction" (B.3.b) (i.e. by referring to theorems for $\theta_{\lambda'}, \lambda' < \lambda$) until the theorem for θ_λ can be deduced (B.3.c). Burstall left implicit the fact that valid proofs should be finite. This is guaranteed in induction principle (3) by the presence of δ belonging to the well-ordering $(\Delta, <)$. Also, Burstall's method makes use of "induction upon data". Up to an isomorphism, this is modelled in induction principle (3) by induction upon ordinals $\lambda \in \Lambda$. This is better shown by means of examples:

Example 4.1 (Weak termination). Continuing example 3.3, we can prove that $\psi(\langle x, y\rangle, \langle x', y'\rangle) = [x' = y' = 0]$ is inevitable for $S = \omega \times \omega$, $t(\langle x, y\rangle, \langle x' y'\rangle) = [(x > 0 \wedge x' = x - 1) \vee (y > 0 \wedge x' = x \wedge y' = y - 1)]$ using Burstall's proof method.

By induction on pairs $\langle x, y\rangle \in \omega \times \omega$ ordered by the left lexicographic ordering ${}_2<$, we prove that execution starting from initial state $\langle x, y\rangle$ inevitably reaches a final state $\langle x', y'\rangle$ such that $x' = y' = 0$. This is obvious when $x = y = 0$ choosing $\langle x', y'\rangle = \langle x, y\rangle$. Else a transition $t(\langle x, y\rangle, \langle x', y'\rangle)$ leads to a state $\langle x', y'\rangle$ satisfying $\langle x', y'\rangle\ {}_2<\ \langle x, y\rangle$ so that by induction going on from $\langle x', y'\rangle$ execution inevitably reaches a state $\langle x'', y''\rangle$ such that $x'' = y'' = 0$.

In induction principle (3), Burstall's induction upon data $\langle x, y\rangle \in \omega \times \omega$ ordered by ${}_2<$ is expressed in a problem independent manner using induction upon ordinals $\lambda \in \Lambda$ naturally ordered by $<$. This is equivalent up to the isomorphism $\lambda = \omega \dot{\times} x + y$. Observe that Burstall's theorem "for all $\langle x, y\rangle \in \omega \times \omega$, execution starting from $\langle x, y\rangle$ inevitably reaches a state $\langle x'', y''\rangle$ such that $x'' = y'' = 0$" is used inductively in its proof for a given instance $\langle x', y'\rangle$ of $\langle x, y\rangle$. In induction principle (3) this symbolic logic sentence is transposed to a set theoretic interpretation (itself presented using first-order logic sentence). More precisely, for each given state $\langle x, y\rangle \in \omega \times \omega$ or equivalently for each $\lambda \in \omega \dot{\times} \omega$, we state that $\theta_\lambda(\langle x, y\rangle, \langle x', y'\rangle) = [(\lambda = \omega \dot{\times} x + y) \Rightarrow (x' = y' = 0)]$ is inevitable for $\langle S, t\rangle$. Hence we consider inevitability theorems for θ_λ, $\lambda \in \omega \dot{\times} \omega$, each of which is an instance of Burstall's symbolic theorem for the value of $\langle x, y\rangle$ such that $\lambda = \omega \dot{\times} x + y$. Then the recursive use of Burstall's theorem in its inductive proof can be interpreted as the use of previously proved inevitability theorems for $\theta_{\lambda'}$ with $\lambda' < \lambda$ in the inevitability proof for θ_λ. Finally it is implied in Burstall's proof that once a proof has been done for each given initial state $\langle x, y\rangle$, we can conclude for all $\langle x, y\rangle \in \omega \times \omega$. This is expressed in induction principle (3) by the use of $\theta_\pi(\langle x, y\rangle, \langle x', y'\rangle) = [x' = y' = 0]$ the inevitability of which results from the inevitability of one of the θ_λ, $\lambda \in \omega \dot{\times} \omega$.

It follows that Burstall's proof consists, up to a morphism, in applying induction principle (3) choosing $\Lambda = \omega \dot{\times} \omega + 1$, $\pi = \omega \times \omega$, $\theta_\pi = \psi$, $\theta_\lambda(\langle x, y\rangle, \langle x', y'\rangle) = [(\lambda = \omega \dot{\times} x + y) \Rightarrow (x' = y' = 0)]$ for $\lambda \in \omega \dot{\times} \omega$ and I defined by cases as follows (we indicate after each case which verification condition of (3) is satisfied): for $\lambda \in \omega \dot{\times} \omega$, $I_\lambda(2, \langle x, y\rangle, \langle x', y'\rangle) = [(\lambda = \omega \dot{\times} x + y) \Rightarrow (x' = x \wedge y' = y)]$ (B.2, if $\lambda = 0$ then B.3.c else B.3.a with $\delta'' = 1$), $I_\lambda(1, \langle x, y\rangle, \langle x', y'\rangle) = [(\lambda = \omega \dot{\times} x + y > 0) \Rightarrow t(\langle x, y\rangle, \langle x', y'\rangle)]$ (B.3.b with $\lambda' = \omega \dot{\times} x' + y'$ and $\delta'' = 0$), $I_\lambda(0, s, s') = \theta_\lambda(s, s')$ (B.3.c), $I_\pi(2, s, s') = ff$ (B.3), $I_\pi(1, \langle x, y\rangle, \langle x', y'\rangle) = [x' = x \wedge y' = y]$ (B.2, B.3.b with $\lambda' = \omega \dot{\times} x + y$ and $\delta'' = 0$), $I_\pi(0, s, s') = \theta_\pi(s, s')$ (B.3.c). $\square$

Example 4.2 (*Proof of the tree traversing program using the "sometime" induction principle*). Burstall's total correctness proof of program 2.1 [2] can be rephrased using proof charts (formally defined in Cousot and Cousot [5] and generalizing Lamport's proof lattices [10]). Arrows $\Rightarrow$ from A to $B_1, \ldots, B_n$ mean that A implies $B_1 \vee \ldots \vee B_n$. We write $A \rightarrow B$ to state that B derives from A by "hand-simulation" and $A \rightsquigarrow B$ if B derives from A using "a little induction". Moreover

$$\begin{tikzcd}
& A \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
A \wedge \neg B \arrow[d] & & A \wedge B \arrow[d, Rightarrow] \\
C & & D
\end{tikzcd}$$

is abbreviated as

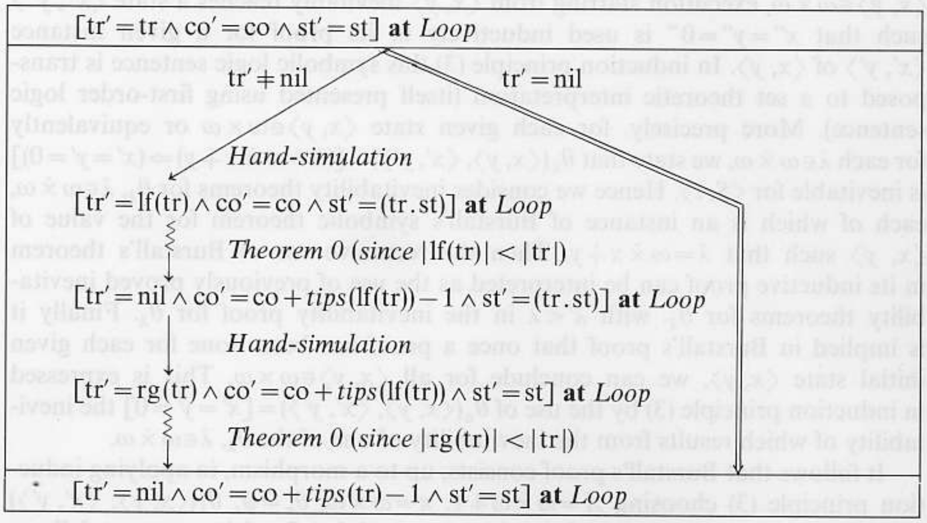
$$\begin{tikzcd}
& A \arrow[dl, "\neg B"'] \arrow[dr, Rightarrow, "B"] & \\
C & & D
\end{tikzcd}$$

Assertions usually contain free variables denoting values of program variables at different time instants. For example, tr, co, st denote the initial values of Tr, Co, St and tr′, co′, st′ their current values. We let

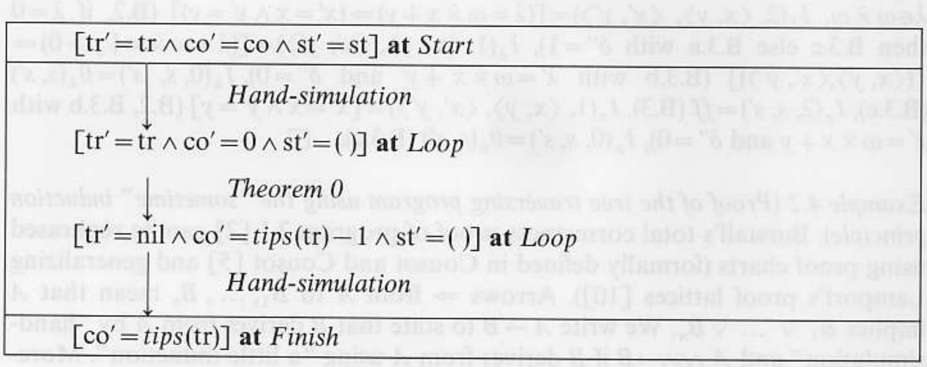
$$|\mathrm{tr}| = \textit{if}\ \mathrm{tr} = \mathrm{nil}\ \textit{then}\ 0\ \textit{else}\ 1 + |\mathrm{lf}(\mathrm{tr})| + |\mathrm{rg}(\mathrm{tr})|$$

denote the weight of tree tr.

Theorem 0. (by induction on $|\text{tr}|$)


[tr′ = tr ∧ co′ = co ∧ st′ = st] **at** *Loop*
tr′ ≠ nil
tr′ = nil
Hand-simulation
[tr′ = lf(tr) ∧ co′ = co ∧ st′ = (tr.st)] **at** *Loop*
Theorem 0 (since |lf(tr)| < |tr|)
[tr′ = nil ∧ co′ = co + *tips*(lf(tr)) − 1 ∧ st′ = (tr.st)] **at** *Loop*
Hand-simulation
[tr′ = rg(tr) ∧ co′ = co + *tips*(lf(tr)) ∧ st′ = st] **at** *Loop*
Theorem 0 (since |rg(tr)| < |tr|)
[tr′ = nil ∧ co′ = co + *tips*(tr) − 1 ∧ st′ = st] **at** *Loop*

Theorem 1.


[tr′ = tr ∧ co′ = co ∧ st′ = st] **at** *Start*
Hand-simulation
[tr′ = tr ∧ co′ = 0 ∧ st′ = ()] **at** *Loop*
Theorem 0
[tr′ = nil ∧ co′ = *tips*(tr) − ∧ st′ = ()] **at** *Loop*
Hand-simulation
[co′ = *tips*(tr)] **at** *Finish*

– The proof of *Theorem i*, $i = 0, 1$ starts from an assumption of the form:

$$[\text{tr}' = \text{tr} \wedge \text{co}' = \text{co} \wedge \text{st}' = \text{st}] \textbf{ at } L$$

and proceeds by successive deductions of intermittent assertions by hand-simulation, by application of a previously proved theorem or by inductive application of *Theorem i*. This proof must be finite so that the graph underlying the corresponding proof chart (PC_i) must have a finite number of vertices n_i. Moreover proof charts exactly corresponding to Burstall's intermittent assertions method do not contain cycles. Consequently, every path in the graph is of finite length. It follows that each intermittent assertion appearing in (PC_i) is at a finite maximal distance $d \in [0, n_i[$ from the exit vertex of (PC_i). (d is the maximum number of deductions before concluding from the intermittent assertion). Hence this intermittent assertion can be written in the form:

$$[\text{tr}' = f_d(\text{tr}) \wedge \text{co}' = g_d(\text{tr}, \text{co}) \wedge \text{st}' = h_d(\text{tr}, \text{st})] \textbf{ at } L_d$$

and should be understood as a shorthand for the more precise: "if execution of the program reaches control point L with values tr, co, st of the variables Tr, Co, St then execution will inevitably reach program point L_d with values tr′, co′, st′ of Tr, Co, St such that $\mathrm{tr}'=f_d(\mathrm{tr})$, $\mathrm{co}'=g_d(\mathrm{tr},\mathrm{co})$ and $\mathrm{st}'=h_d(\mathrm{tr},\mathrm{st})$".

The proof chart (PC) itself can be expressed in relational form when grouping these intermittent assertions into disjunctive form:

$$PC_i(\delta',\langle l,\mathrm{tr},\mathrm{co},\mathrm{st}\rangle,\langle l',\mathrm{tr}',\mathrm{co}',\mathrm{st}'\rangle)=$$
$$\left[(l=L)\Rightarrow\left(\bigvee_{d=0}^{n_i-1}[\delta'=d\wedge l'=L_d\wedge\mathrm{tr}'=f_d(\mathrm{tr})\wedge\mathrm{co}'=g_d(\mathrm{tr},\mathrm{co})\wedge\mathrm{st}'=h_d(\mathrm{tr},\mathrm{st})]\right)\right]$$

More precisely we have:

$$\begin{aligned}&PC_0(\delta',\langle l,\mathrm{tr},\mathrm{co},\mathrm{st}\rangle,\langle l',\mathrm{tr}',\mathrm{co}',\mathrm{st}'\rangle)=\\&\quad[(l=\mathrm{Loop})\Rightarrow\\&\quad\quad([\delta'=4\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{tr}\wedge\mathrm{co}'=\mathrm{co}\wedge\mathrm{st}'=\mathrm{st}]\\&\quad\quad\vee[\delta'=3\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{lf}(\mathrm{tr})\wedge\mathrm{co}'=\mathrm{co}\wedge\mathrm{st}'=(\mathrm{tr}.\mathrm{st})]\\&\quad\quad\vee[\delta'=2\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{nil}\wedge\mathrm{co}'=\mathrm{co}+\mathrm{tips}(\mathrm{lf}(\mathrm{tr}))-1\wedge\mathrm{st}'=(\mathrm{tr}.\mathrm{st})]\\&\quad\quad\vee[\delta'=1\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{rg}(\mathrm{tr})\wedge\mathrm{co}'=\mathrm{co}+\mathrm{tips}(\mathrm{lf}(\mathrm{tr}))\wedge\mathrm{st}'=\mathrm{st}]\\&\quad\quad\vee[\delta'=0\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{nil}\wedge\mathrm{co}'=\mathrm{co}+\mathrm{tips}(\mathrm{tr})-1\wedge\mathrm{st}'=\mathrm{st}])]\end{aligned}$$

$$\begin{aligned}&PC_1(\delta',\langle l,\mathrm{tr},\mathrm{co},\mathrm{st}\rangle,\langle l',\mathrm{tr}',\mathrm{co}',\mathrm{st}'\rangle)=\\&\quad[(l=\mathrm{Start})\Rightarrow\\&\quad\quad([\delta'=3\wedge l'=\mathrm{Start}\wedge\mathrm{tr}'=\mathrm{tr}\wedge\mathrm{co}'=\mathrm{co}\wedge\mathrm{st}'=\mathrm{st}]\\&\quad\quad\vee[\delta'=2\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{tr}\wedge\mathrm{co}'=0\wedge\mathrm{st}'=(\,)]\\&\quad\quad\vee[\delta'=1\wedge l'=\mathrm{Loop}\wedge\mathrm{tr}'=\mathrm{nil}\wedge\mathrm{co}'=\mathrm{tips}(\mathrm{tr})-1\wedge\mathrm{st}'=(\,)]\\&\quad\quad\vee[\delta'=0\wedge l'=\mathrm{Finish}\wedge\mathrm{co}'=\mathrm{tips}(\mathrm{tr})])].\end{aligned}$$

. In the proof of *Theorem 1* for program 2.1 we use *Theorem 0* and in the proof of *Theorem 0* for a tree tr, we assume that *Theorem 0* holds for trees tr′ such that $|\mathrm{tr}'|<|\mathrm{tr}|$. Hence the proof is by induction along the well-ordering $\prec$ on $\{1\}\cup\{\{0\}\times\omega\}$ defined by $(0,n)\prec 1$ for all $n\in\omega$ and $(0,n')\prec(0,n)$ if and only if $n'<n$. Burstall [2] speaks of "induction on data" and does not explicitly take into account the order in which theorems should be proved because he implicitely relies upon the mathematical culture of his readers to avoid errors such as circular proofs.

Induction principle (3) ensures that induction is used correctly because an inevitability theorem for $\theta_{\lambda'}$ can be used in the inevitability proof of θ_λ only if the inevitability of $\theta_{\lambda'}$ has been proved before that of θ_λ i.e. $\lambda'<\lambda$. All particular cases such as inductive or mutually inductive proofs of theorems in Burstall's method can be taken into account by a suitable choice of the well-ordering $(\Lambda,<)$. Up to an isomorphism we can always choose $\Lambda\in Ord$.

For example, instead of the well-ordering $(W,\prec)$ where $W=\{1\}\cup\{\{0\}\times\omega\}$ we can use $(\Lambda,<)$ where $\Lambda=\mathrm{Sup}\{\rho(x)\colon x\in W\}=\omega+1$ up to the isomorphism $\rho\in(W\to Ord)$ defined by $\rho(x)=\mathrm{Sup}^+\{\rho(y)\colon y\prec x\}$ so that $\rho(1)=\omega$ and $\rho(0,n)=n$.

. The reader can now check that Burstall's proof [2] of program 2.1 exactly consists in applying induction principle (3) with:

$\Lambda = \omega + 1$
$\theta_\lambda((\langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle) =$
$([(\lambda = \omega \wedge l = \mathrm{Start}) \Rightarrow (l' = \mathrm{Finish} \wedge \mathrm{co}' = \mathrm{tips}(\mathrm{tr}))]$
$[(|\mathrm{tr}| = \lambda < \omega \wedge l = \mathrm{Loop}) \Rightarrow (l' = \mathrm{Loop} \wedge \mathrm{tr}' = \mathrm{nil} \wedge \mathrm{co}' = \mathrm{co} + \mathrm{tips}(\mathrm{tr}) - 1 \wedge \mathrm{st}' = \mathrm{st})])$

$\pi = \omega$

$\Delta = 5$
$I_\lambda(\delta', \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle) =$
$([(\lambda = \omega) \Rightarrow PC_1(\delta', \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle)]$
$\wedge [(|\mathrm{tr}| = \lambda < \omega) \Rightarrow PC_0(\delta', \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle)])$

For example, from

$I_{|\mathrm{tr}|}(3, \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l, \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle)$
$= [(l = \mathrm{Loop}) \Rightarrow (l' = \mathrm{Loop} \wedge \mathrm{tr}' = \mathrm{lf}(\mathrm{tr}) \wedge \mathrm{co}' = \mathrm{co} \wedge \mathrm{st}' = (\mathrm{tr}.\mathrm{st}))]$

we checked that $|\mathrm{tr}'| = |\mathrm{lf}(\mathrm{tr})| < |\mathrm{tr}|$ and using theorem

$\theta_{|\mathrm{tr}'|}(\langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle \langle l'', \mathrm{tr}'', \mathrm{co}'', \mathrm{st}''\rangle)$
$= [(l' = \mathrm{Loop}) \Rightarrow (l'' = \mathrm{Loop} \wedge \mathrm{tr}'' = \mathrm{nil} \wedge \mathrm{co}'' = \mathrm{co}' + \mathrm{tips}(\mathrm{tr}') - 1 \wedge \mathrm{st}'' = \mathrm{st}')]$

we derived

$[(l = \mathrm{Loop}) \Rightarrow (l'' = \mathrm{Loop} \wedge \mathrm{tr}'' = \mathrm{nil} \wedge \mathrm{co}'' = \mathrm{co} + \mathrm{tips}(\mathrm{lf}(\mathrm{tr})) - 1 \wedge \mathrm{st}'' = (\mathrm{tr}.\mathrm{st}))]$
$= I_{|\mathrm{tr}|}(2, \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l'', \mathrm{tr}'', \mathrm{co}'', \mathrm{st}''\rangle).$

Also from

$I_{|\mathrm{tr}|}(2, \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l' \mathrm{tr}' \mathrm{co}', \mathrm{st}'\rangle)$
$= [(l = \mathrm{Loop}) \Rightarrow (l' = \mathrm{Loop} \wedge \mathrm{tr}' = \mathrm{nil} \wedge \mathrm{co}' = \mathrm{co} + \mathrm{tips}(\mathrm{lf}(\mathrm{tr})) - 1 \wedge \mathrm{st}' = (\mathrm{tr}.\mathrm{st}))]$

and

$t(\langle \mathrm{Loop}, \mathrm{nil}, \mathrm{co}', (\mathrm{tr}.\mathrm{st})\rangle, \langle \mathrm{Loop}, \mathrm{rg}(\mathrm{tr}), \mathrm{co}' + 1, \mathrm{st}\rangle)$

we derived

$[(l = \mathrm{Loop}) \Rightarrow (l'' = \mathrm{Loop} \wedge \mathrm{tr}'' = \mathrm{rg}(\mathrm{tr}) \wedge \mathrm{co}'' = \mathrm{co} + \mathrm{tips}(\mathrm{lf}(\mathrm{tr})) \wedge \mathrm{st}'' = \mathrm{st})]$
$= I_{|\mathrm{tr}|}(1, \langle l, \mathrm{tr}, \mathrm{co}, \mathrm{st}\rangle, \langle l', \mathrm{tr}', \mathrm{co}', \mathrm{st}'\rangle)$

the one difference is that program 2.1 is total, so that absence of run-time errors and more generally of blocking states $(I_\lambda(\delta', s, s') \Rightarrow \exists s'' \in S . t(s', s''))$ need not be explicitly checked. □

Induction principle (3) is sound (since (3)⇒(1)) and semantically complete (since (1)⇒(3)) even when we choose $\Delta \in \omega$ instead of $\Delta \in Ord$ in (3) (see the proofs in Cousot and Cousot [5]).

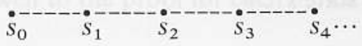
5. Sometime = Always + Recursion

As shown by example 4.1 a major difference between Burstall's [2] method (and its followers such as Owicki and Lamport [14] or Manna and Pnueli [11] who present proofs using proof lattices) and induction principle (3) is that Burstall insists upon a finite presentation of proofs (equivalently this consists in considering proof charts without cycles and $\Delta \in \omega$ instead of $\Delta \in Ord$ in (3)) whereas (3) only insists upon well-foundedness. In particular, (3) can be presented using proof charts with cycles along which some entity (modelled by δ in induction principle (3)) has to be strictly decreased.

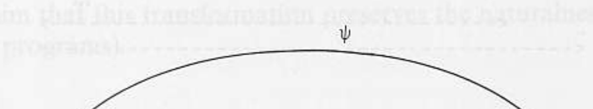
The importance of this remark is to point out that when conveniently generalized into induction principle (3), Burstall's method contains Floyd's method (more precisely induction principle (2)) as a particular case. This is because in induction principle (3) we can choose $\Lambda = 1 = \{0\}$, $\theta_0 = \psi$, $\pi = 0$ so that (B.1) is always true and (B.3.b) never applies, in which case (3) exactly amounts to (2).

The clear advantage of induction principle (3) over (2) is that (3) introduces the possibility of inductive proofs not present in (2). This clearly formalizes Burstall's [2] remark that "recursive" proofs can be retained for iterative versions of recursive programs.

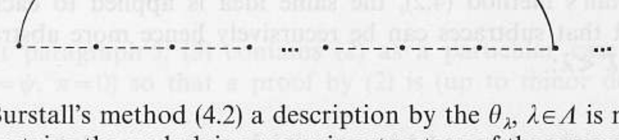
This point can be illustrated pictorially using the total correctness proofs 3.1 and 4.2 of program 2.1. We represent an execution trace of the program as:



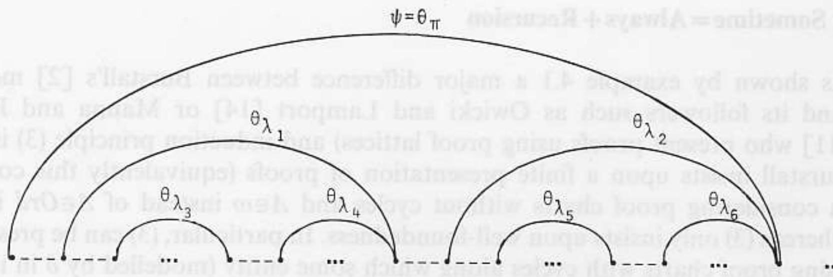
where s_0 is an initial state and there is a transition $t(s_i, s_{i+1})$ from state s_i to state s_{i+1}. The fact that $I(s_i, s_j)$ is true will be represented as:



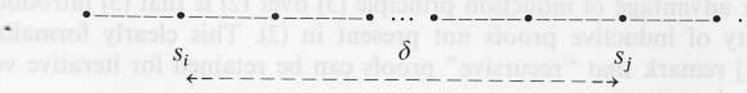
With Floyd's method (3.1) properties of execution traces are specified by ψ:



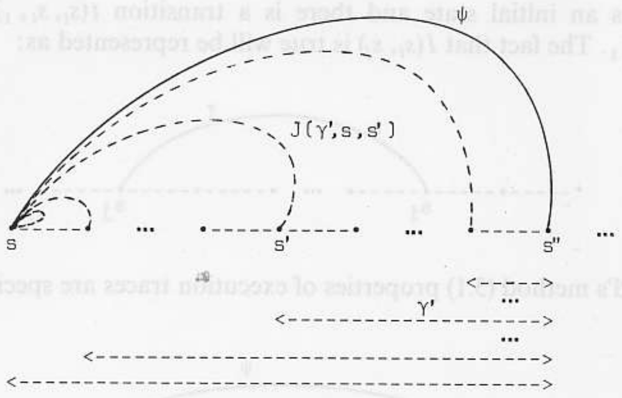
whereas with Burstall's method (4.2) a description by the θ_λ, $\lambda \in \Lambda$ is much more precise since it retains the underlying recursive structure of the program:



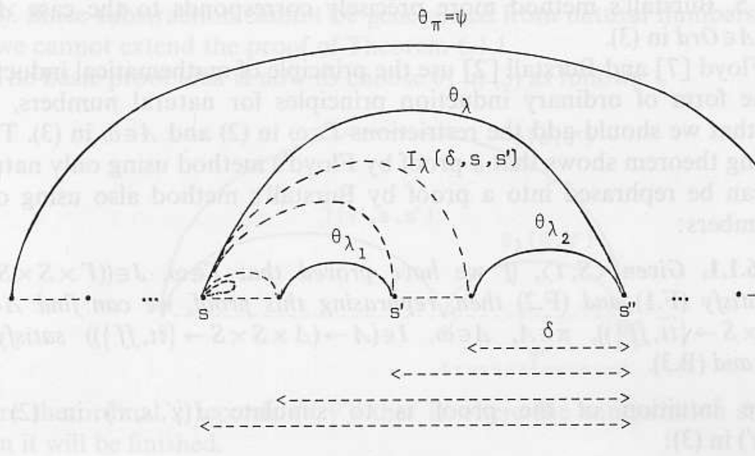
Much more importantly, using (3) a proof can be successively decomposed into independent subproofs whereas (2) requires a global proof. To illustrate this point we represent the fact that state s_j can be reached from state s_i in at most δ steps as:



With Floyd's method (3.1), the global invariant $J(\gamma', s, s')$ relates the current state s' to the initial state s and provides a bound γ' on the number of steps before reaching the final state s'':



With Burstall's method (4.2), the same idea is applied to each θ_λ, $\lambda \in \Lambda$ but for the fact that subtraces can be recursively hence more abstractly described by the $\theta_{\lambda'}$, $\lambda' < \lambda$:



A traditional "separation of concern" argument in favour of induction principle (2) is the usual decomposition of (2) into partial correctness, termination, absence of blocking states proofs. As noticed by Gries [8] all parts of the complete proof are packed together in Burstall's method. However this is only a matter of presentation of (3) which can be easily remedied since the decomposition into independent partial correctness termination, absence of blocking states (and interference freeness checks for partitioned transition systems) proofs is applicable to (2) but as well to the proof for each θ_λ, $\lambda \in \Lambda$ in (3).

6. Sometime ≡ Always

Since (2)⇔(1)⇔(3), induction principles (2) and (3) are equivalent. This means that when a proof exists by one method, a proof must exist by the other method. However from a practical point of view it has sometimes been regarded as a challenge to use (2) on a program which has a "natural" proof by (3) (see Manna and Waldinger [12] and Gries [8]). We now prove a stronger equivalence result which shows that this challenge can always be met because a proof of (3) can be systematically rephrased into a proof by (2) and viceversa. The transformation between the two proofs is very similar to recursion elimination in programs and recursive presentation of iterative programs. Hence naturally, we do not claim that this transformation preserves the naturalness of the proofs (so we do for programs).

6.1. Always ⇒ Sometime

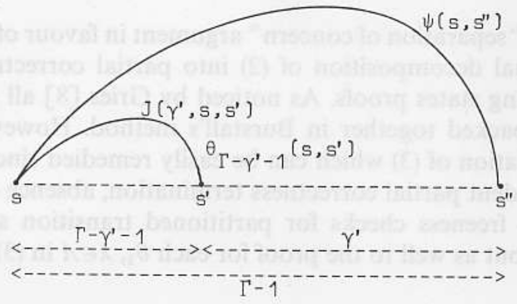
As observed at paragraph 5, (3) contains (2) as a particular case (choosing $\Lambda = 1 = \{0\}$, $\theta_0 = \psi$, $\pi = 0$) so that a proof by (2) is (up to minor details) also a proof by (3).

However we need the more powerful theorem 6.1.1 to make a fair comparison between Burstall's and Floyd's methods. This is because, as also observed at paragraph 5, Burstall's method more precisely corresponds to the case $\Lambda \in \omega$ instead of $\Lambda \in Ord$ in (3).

Since Floyd [7] and Burstall [2] use the principle of mathematical induction only in the form of ordinary induction principles for natural numbers, one can claim that we should add the restrictions $\Gamma \in \omega$ in (2) and $\Lambda \in \omega$ in (3). Then the following theorem shows that a proof by Floyd's method using only natural numbers can be rephrased into a proof by Burstall's method also using only natural numbers:

Theorem 6.1.1. *Given* $\langle S, t \rangle$, *if we have proved that* $\Gamma \in \omega$, $J \in ((\Gamma \times S \times S) \rightarrow \{tt, ff\})$ *satisfy* (F.1) *and* (F.2) *then, rephrasing this proof, we can find* $\Lambda \in \omega$, $\theta \in (\Lambda \rightarrow (S \times S \rightarrow \{tt, ff\}))$, $\pi \in \Lambda$, $\Delta \in \omega$, $I \in (\Lambda \rightarrow (\Delta \times S \times S \rightarrow \{tt, ff\}))$ *satisfying* (B.1), (B.2) *and* (B.3).

Proof. The intuition of the proof is to simulate $J(\gamma', s, s')$ in (2) by $\theta_{\Gamma-\gamma'-1}(s, s')$ in (3):



More precisely, we choose $\Lambda = \Gamma + 1 = \{0, \dots \Gamma\}$, $\pi = \Gamma$, $\theta_\pi = \psi$ (B.1), $\Delta = 3 = \{0, 1, 2\}$, $I_\pi(2, s, s') = ff$ (B.3), $I_\pi(1, s, s') = [s = s']$ (B.2 and B.3.b with $\lambda' = \Gamma - 1$ and $\delta'' = 0$ since $J(0, s, s') \Rightarrow \psi(s, s')$ by F.2), $I_\pi(0, s, s') = \theta_\pi(s, s')$ (B.3.c), for $\lambda < \Gamma$, $\theta_\lambda(s, s') = (\exists \gamma' < (\Gamma - \lambda) . J(\gamma', s, s')]$, $I_\lambda(2, s, s') = [s = s']$ (B.2 and B.3.b with $\lambda' = \lambda - 1$ and $\delta'' = 1$), $I_\lambda(1, s, s') = [\exists \gamma' \leq (\Gamma - \lambda) . J(\gamma', s, s')]$ (B.3.c by F.2.b or B.3.a with $\delta'' = 0$ by F.2.a), $I_\lambda(0, s, s') = \theta_\lambda(s, s')$ (B.3.c). (Verification conditions satisfied by the I_λ, $\lambda \in \Lambda$ have been indicated between parentheses). □

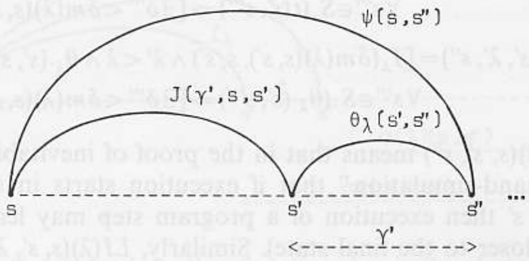
If the nondeterminism is unbounded and induction is restricted to natural numbers the Floyd's method is not complete (Dijkstra [6]). In this case induction principle (2) is therefore not semantically complete with $\Gamma \in \omega$. The same way, (3) is not semantically complete with $\Lambda \in \omega$.

However, considering $\Gamma \in Ord$ in (2) and $\Lambda \in Ord$ in (3) as respective generalizations of Floyd's and Burstall's methods, we claim that a proof by Floyd's method can be always rephrased into a proof by Burstall's method. This is a consequence of the following Theorem 6.1.2. (where (3) is still restricted to finitely presented proofs i.e. $\Delta \in \omega$):

Theorem 6.1.2. *Given* $\langle S, t \rangle$, *if we have proved that* $\Gamma \in Ord$, $J \in ((\Gamma \times S \times S) \rightarrow \{tt, ff\})$ *satisfy* (F.1) *and* (F.2) *then, rephrasing this proof, we can find* $\Lambda \in Ord$, $\theta \in (\Lambda \rightarrow (S \times S \rightarrow \{tt, ff\}))$, $\pi \in \Lambda$, $\Delta \in \omega$, $I \in (\Lambda \rightarrow (\Delta \times S \times S \rightarrow \{tt, ff\}))$ *satisfying* (B.1), (B.2) *and* (B.3).

Proof. Since substraction cannot be generalized from natural numbers to ordinals, we cannot extend the proof of Theorem 6.1.1.

The basic proof idea is now to choose θ_λ in (3) as follows:



where the ordinal λ records $\langle s, \gamma' \rangle$ that is where the computation started and when it will be finished.

By the counting principle (an equivalent of the axiom of choice) there is an ordinal π and a bijection b that maps π onto $S \times \Gamma$. Hence we can define $\underline{s} \in (\pi \dot{\to} S)$ and $\underline{\gamma} \in (\pi \to \Gamma)$ such that $b(\langle \underline{s}(\lambda), \underline{\gamma}(\lambda) \rangle) = \lambda$ for all $\lambda \in \pi$.

Choose

. $\Lambda = \pi \dotplus 1 \; (= \pi \cup \{\pi\})$
. $\theta_\lambda(s, s'') = ([\lambda = \pi \wedge \psi(s, s'')] \vee [\lambda < \pi \wedge (J(\underline{\gamma}(\lambda), \underline{s}(\lambda), s) \Rightarrow \psi(\underline{s}(\lambda), s''))])$
. $\Delta = 3$
. $I_\pi(2, s, s') = ff$
$I_\pi(1, s, s') = [\exists \gamma \in \Gamma . J(\gamma, s, s') \wedge s' = s]$
$I_\pi(0, s, s') = \psi(s, s')$

For all $\lambda < \pi$ (or equivalently $\lambda \in \pi$),
. $I_\lambda(2, s, s') = [J(\underline{\gamma}(\lambda), \underline{s}(\lambda), s) \Rightarrow (s = s')]$
$I_\lambda(1, s, s') = [J(\underline{\gamma}(\lambda), \underline{s}(\lambda), s) \Rightarrow (t(s, s') \wedge \neg \psi(\underline{s}(\lambda), s))]$
$I_\lambda(0, s, s') = \theta_\lambda(s, s')$. □

6.2. *Sometime ⇒ Always*

As a consequence of the following Theorem 6.2.1, a proof by Burstall's method can be rephrased into a proof by Floyd's method. Not surprisingly the technique is analogous to the transformation of a recursive program into an equivalent iterative one (by a compiler using a run-time stack). To turn this comparison into a caricature this is also analogous to the replacement of all theorems (and their proofs) in a mathematical book by a single proposition (and its proof)!

Theorem 6.2.1. *If we have proved that* $\Lambda \in Ord$, $\theta \in (\Lambda \to (S \times S \to \{tt, ff\}))$, $\pi \in \Lambda$, $\Delta \in Ord$, $I \in (\Lambda \times S \times S \to \{tt, ff\})$ *satisfy* (B.1), (B.2) *and* (B.3) *for a transition system* $\langle S, t \rangle$, *then rephrasing this proof, we can find* $\Gamma \in Ord$, $J \in ((\Gamma \times S \times S) \to \{tt, ff\})$ *satisfying* (F.1) *and* (F.2).

Proof. (This rather long proof is illustrated by example 6.1 below).

(a) We define $\delta m \in (\Lambda \to (S \times S \to \Delta))$ such that $\delta m(\lambda)(s, s')$ is the smallest δ such that $I_\lambda(\delta, s, s')$ holds if $\exists \delta \in \Delta . I_\lambda(\delta, s, s')$; otherwise $\delta m(\lambda)(s, s')$ is 0.

(b) We let $HS \in (\Lambda \to (S \times S \times S \to \{tt, ff\}))$ and $LI \in (\Lambda \to (S \times S \times \Lambda \times S \to \{tt, ff\}))$ be

$$HS(\lambda)(s, s', s'') = [I_\lambda(\delta m(\lambda)(s, s'), s, s') \wedge t(s', s'') \wedge \forall s''' \in S.(t(s', s''') \Rightarrow [\exists \delta''' < \delta m(\lambda)(s, s').I_\lambda(\delta''', s, s''')])]$$

$$LI(\lambda)(s, s', \lambda', s'') = [I_\lambda(\delta m(\lambda)(s, s'), s, s') \wedge \lambda' < \lambda \wedge \theta_{\lambda'}(s', s'') \wedge \forall s''' \in S.(\theta_{\lambda'}(s', s''') \Rightarrow [\exists \delta''' < \delta m(\lambda)(s, s').I_\lambda(\delta''', s, s''')])]$$

Informally, $HS(\lambda)(s, s', s'')$ means that in the proof of inevitability for θ_λ it can be shown by "hand-simulation" that if execution starts in state s and later on reaches state s' then execution of a program step may lead to s'' (and all such states are closer to the final state). Similarly, $LI(\lambda)(s, s', \lambda', s'')$ means that in the proof of inevitability for θ_λ it can be shown by "a little induction" that if execution starts in state s and later on reaches state s' then according to the inevitability theorem for $\theta_{\lambda'}$ it may lead to state s'' (and all such states are closer to the final state). (Whenever $\forall \delta \in \Lambda. \neg I_\lambda(\delta, s, s')$ we have in particular $\neg I_\lambda(0, s, s')$ so that by definition of $\delta m(\lambda)(s, s')$, $HS(\lambda)(s, s', s'')$ and $LI(\lambda)(s, s', \lambda', s'')$ do not hold).

(c) We define the relation $\succ$ on $\Lambda \times S \times S$ such that

$$\langle \lambda_1, s_1, s_1' \rangle \succ \langle \lambda_2, s_2, s_2' \rangle$$

if and only if

$$[(\lambda_2 = \lambda_1 \wedge s_2 = s_1 \wedge HS(\lambda_1)(s_1, s_1', s_2')) \vee (\lambda_2 = \lambda_1 \wedge s_2 = s_1 \wedge \exists \lambda' < \lambda_1.LI(\lambda_1)(s_1, s_1', \lambda', s_2')) \vee (\lambda_2 < \lambda_1)].$$

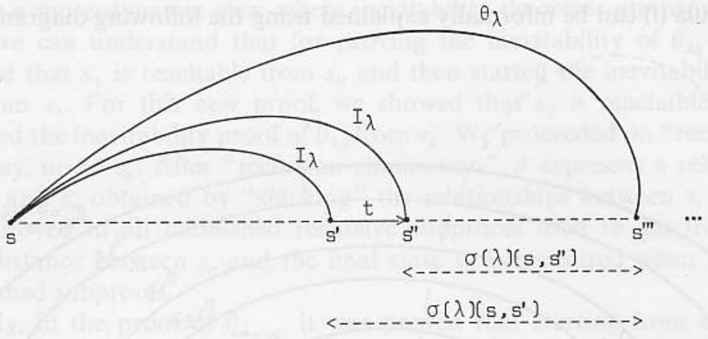
(d) The relation $\succ$ on $\Lambda \times S \times S$ is well-founded.

By reductio ad absurdum, if there were an infinite chain such that $\forall i \geq 0.(\langle \lambda_i, s_i, s_i' \rangle \succ \langle \lambda_{i+1}, s_{i+1}, s_{i+1}' \rangle)$ then by (c) and (b) the chain $\langle \lambda_i, \delta m(\lambda_i)(s_i, s_i') \rangle$, $i \geq 0$ would be strictly decreasing for the left lexicographic ordering on pairs of ordinals, a contradiction.

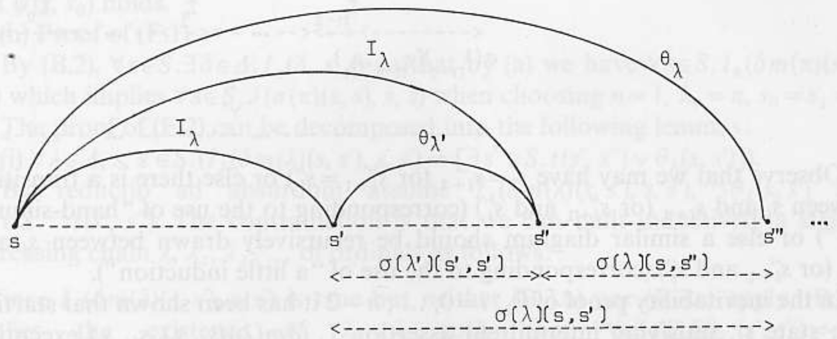
(e) It follows from (d) that we can define $\sigma \in (\Lambda \to (S \times S \to Ord))$ by transfinite recursion such that for all $\lambda \in \Lambda$, $s, s' \in S$ we have:

$$\sigma(\lambda)(s, s') = \mathrm{Sup}\{\alpha + 1 : \neg \theta_\lambda(s, s') \wedge ([\exists s'' \in S.(HS(\lambda)(s, s', s'') \wedge \alpha = \sigma(\lambda)(s, s''))] \vee [\exists \lambda' < \lambda.(\exists s'' \in S.LI(\lambda)(s, s', \lambda', s'') \wedge \alpha = \mathrm{Sup}\{\sigma(\lambda)(s, s''): LI(\lambda)(s, s', \lambda', s'')\} + \sigma(\lambda')(s', s'))])\}.$$

Intuitively, assume that an execution starting in state s reaches state s' and that we have proved that from state s' execution will inevitably reach some final state s''' satisfying $\theta_\lambda(s, s''')$. Then $\sigma(\lambda)(s, s')$ is defined so that there are "at most $\sigma(\lambda)(s, s')$ steps" between s' and s'''. If $\theta_\lambda(s, s')$ holds no further step is necessary so that $\sigma(\lambda)(s, s') = 0$. Else the inevitability proof has been done using "hand-simulation" or "a little induction" or both. If $HS(\lambda)(s, s', s'')$ holds then (by induction) s''' will be reached "in at most $\alpha = \sigma(\lambda)(s, s'')$ steps" from s'' hence "in at most $\alpha + 1$ steps" from s' since $t(s', s'')$ holds. A supremum is obtained by considering all possible states s'':



If $LI(\lambda)(s, s', \lambda', s'')$ holds then the situation is similar but for the fact that $\theta_{\lambda'}(s, s'')$ holds (instead of $t(s', s'')$):



Considering all possible states s'' we have "at most $\operatorname{Sup}\{\sigma(\lambda)(s, s''): LI(\lambda)(s, s', \lambda', s'')\}$ steps" between any such state s'' and the final state s'''. Moreover there are "at most $\sigma(\lambda')(s', s')$ steps" between s' and s''. Hence there are "at most $\alpha = \operatorname{Sup}\{\sigma(\lambda)(s, s''): LI(\lambda)(s, s', \lambda', s'')\} + \sigma(\lambda')(s', s')$ steps" between s' and s'''. Since this α corresponds to a given λ' we than consider the supremum of these α for all possible λ' adding 1 not to rule out the case when $\theta_{\lambda'}$ is identity.

We choose:

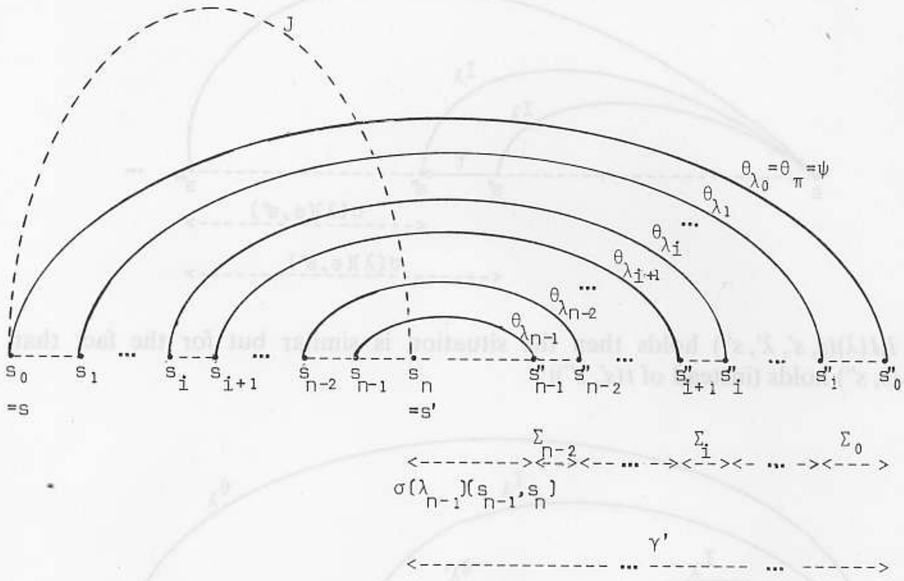
(f) $J(\gamma', s, s') = [\exists n \in (\omega \sim 0), \lambda \in (n \to \Lambda), s_0 \in S, \ldots, s_n \in S, \Sigma \in (n-1 \to Ord).$

$$\begin{aligned}
&((s_0 = s) \wedge (\lambda_0 = \pi) \wedge (s_n = s')\\
&\wedge \forall i \in (n \sim 0).\, \exists s_i'' \in S.\, LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s_i'')\\
&\wedge I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s_n), s_{n-1}, s_n)\\
&\wedge \forall i \in (n \sim 0).\, \neg\theta_{\lambda_i}(s_i, s_{i+1})\\
&\wedge \forall i \in (n \sim 0).\, \Sigma_{i-1} = \operatorname{Sup}\{\sigma(\lambda_{i-1})(s_{i-1}, s''): LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s'')\}\\
&\wedge \gamma' = \Sigma_0 + \ldots + \Sigma_{n-2} + \sigma(\lambda_{n-1})(s_{n-1}, s_n))].
\end{aligned}$$

(g) $\Gamma = \operatorname{Sup}^+\{\gamma' \in Ord: \exists s, s' \in S.\, J(\gamma', s, s')\}$.

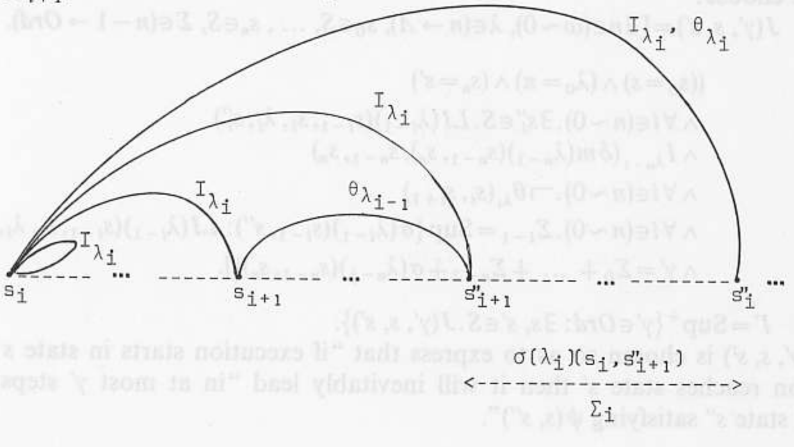
$J(\gamma', s, s')$ is chosen so as to express that "if execution starts in state s and later on reaches state s' then it will inevitably lead "in at most γ' steps" to a final state s'' satisfying $\psi(s, s'')$".

Formula (f) can be informally explained using the following diagram:



Observe that we may have $s_i = s_{i+1}$ (or $s''_{i+1} = s''_i$) or else there is a transition between s_i and s_{i+1} (or s''_{i+1} and s''_i) (corresponding to the use of "hand-simulation") or else a similar diagram should be recursively drawn between s_i and s_{i+1} (or s''_{i+1} and s''_i) (corresponding to the use of "a little induction").

In the inevitability proof of θ_{λ_i}, $i = 0, \ldots, n-2$ it has been shown that starting from state s_i, satisfying intermittent assertion $I_{\lambda_i}(\delta m(\lambda_i)(s_i, s_i), s_i, s_i)$ execution will lead to state s_{i+1} such that $I_{\lambda_i}(\delta m(\lambda_i)(s_i, s_{i+1}), s_i, s_{i+1})$ holds. Applying the inevitability theorem for $\theta_{\lambda_{i+1}}$ at this point, it has been shown that execution will inevitably lead to some state s''_{i+1} such that $\theta_{\lambda_{i+1}}(s_{i+1}, s''_{i+1})$ holds and satisfies intermittent assertion $I_{\lambda_i}(\delta m(\lambda_i)(s_i, s''_{i+1}), s_i, s''_{i+1})$. Therefore $LI(\lambda_i)(s_i, s_{i+1}, \lambda_{i+1}, s''_{i+1})$ holds. Next it has been derived from the last intermittent assertion that $I_{\lambda_i}(\delta m(\lambda_i)(s_i, s''_i), s_i, s''_i)$ holds, which implies $\theta_{\lambda_i}(s_i, s''_i)$ and terminates the inevitability proof for θ_{λ_i}. Moreover execution goes from s''_{i+1} to s''_i "in at most $\sigma(\lambda_i)(s_i, s''_{i+1})$ steps" hence "in at most Σ_i steps" when considering all possible states s''_{i+1}:



Using a more dynamic view where inevitability theorems are proved when needed, we can understand that for proving the inevitability of θ_{λ_0} from s_0, we proved that s_1 is reachable from s_0 and then started the inevitability proof of θ_{λ_1} from s_1. For this new proof, we showed that s_2 is reachable from s_1 and started the inevitability proof of θ_{λ_2} from s_2. We proceeded on "recursively" in this way, up to s_n. After "recursion elimination", J expresses a relation between s_0 and s_n obtained by "stacking" the relationships between s_i and s_{i+1} already proved in all unfinished recursive subproofs used to get from s_0 to s_n. The distance between s_n and the final state s''_0 is measured when returning from finished subproofs.

Finally, in the proof of $\theta_{\lambda_{n-1}}$, it was proved that starting from state s_{n-1} execution will lead to state $s_n = s'$ and from s_n "in at most $\sigma(\lambda_{n-1})(s_{n-1}, s_n)$ steps" to s''_{n-1}.

We conclude that if execution starts from $s = s_0$ and reaches $s_1, \ldots, s_{n-1}, s_n = s'$ then "in at most γ' steps" it will go through states $s''_{n-1}, \ldots, s''_1, s''_0$ such that $\psi(s, s''_0)$ holds.

(h) Proof of (F.1).

By (B.2), $\forall s \in S. \exists \delta \in \Delta. I_\pi(\delta, s, s)$ so that by (a) we have $\forall s \in S. I_\pi(\delta m(\pi)(s, s), s, s)$ which implies $\forall s \in S. J(\sigma(\pi)(s, s), s, s)$ when choosing $n = 1$, $\lambda_0 = \pi$, $s_0 = s_1 = s$.

The proof of (F.2) can be decomposed into the following lemmas:

(i) $\forall \lambda \in \Lambda, s, s' \in S. (I_\lambda(\delta m(\lambda)(s, s'), s, s') \Rightarrow [\exists s'' \in S. t(s', s'') \vee \theta_\lambda(s, s')])$.

By reductio ad absurdum assume $I_\lambda(\delta m(\lambda)(s, s'), s, s'), \neg \theta_\lambda(s, s')$ and $\forall s'' \in S. \neg t(s', s'')$. The contradiction is that we can built inductively a strictly decreasing chain $\lambda, \lambda_1, \lambda_2, \ldots$ of ordinals as follows:

. Since $I_\lambda(\delta m(\lambda)(s, s'), s, s')$ is true but neither (B.3.a) nor (B.3.c) apply, (B.3.b) implies the existence of $\lambda_1 < \lambda$ such that $\forall s'' \in S. [\theta_{\lambda_1}(s', s'') \Rightarrow \exists \delta'' < \delta m(\lambda)(s, s'). I_\lambda(\delta'', s, s'')]$. By (B.2) and (a), $I_{\lambda_1}(\delta m(\lambda_1)(s', s'), s', s')$ holds and we cannot have $\theta_{\lambda_1}(s', s')$ since otherwise $\exists \delta'' < \delta m(\lambda)(s, s'). I_\lambda(\delta'', s, s')$ contrary to (a).

. Assume that we have built a finite strictly decreasing sequence $\lambda, \lambda_1, \ldots, \lambda_i$, $i \geqq 0$ such that $I_{\lambda_i}(\delta m(\lambda_i)(s', s'), s', s') \wedge \neg \theta_{\lambda_i}(s', s')$ holds. It can be prolonged by $\lambda_{i+1} < \lambda_i$ because neither (B.3.a) nor (B.3.c) apply, so that (B.3.b) implies the existence of some $\lambda_{i+1} < \lambda_i$ such that $\forall s'' \in S. [\theta_{\lambda_{i+1}}(s', s'') \Rightarrow \exists \delta'' < \delta m(\lambda_i)(s', s''). I_{\lambda_i}(\delta'', s', s'')]$ hence $\neg \theta_{\lambda_{i+1}}(s', s')$. Moreover $I_{\lambda_{i+1}}(\delta m(\lambda_{i+1})(s', s'), s', s')$ follows from (B.2) and (a). Q.E.D.

(j) $\forall \gamma' \in \Gamma, s, s' \in S. [(J(\gamma', s, s') \wedge \neg \psi(s, s')) \Rightarrow \exists s'' \in S. t(s', s'')]$.

When $J(\gamma', s, s')$ holds we have $I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s_n), s_{n-1}, s_n)$. Moreover $\neg \theta_{\lambda_{n-1}}(s_{n-1}, s_n)$ holds when $n > 1$ but also when $n = 1$ because $\psi(s, s') = \theta_\pi(s, s') = \theta_{\lambda_0}(s_0, s_1)$. Now (j) follows from (i). Q.E.D.

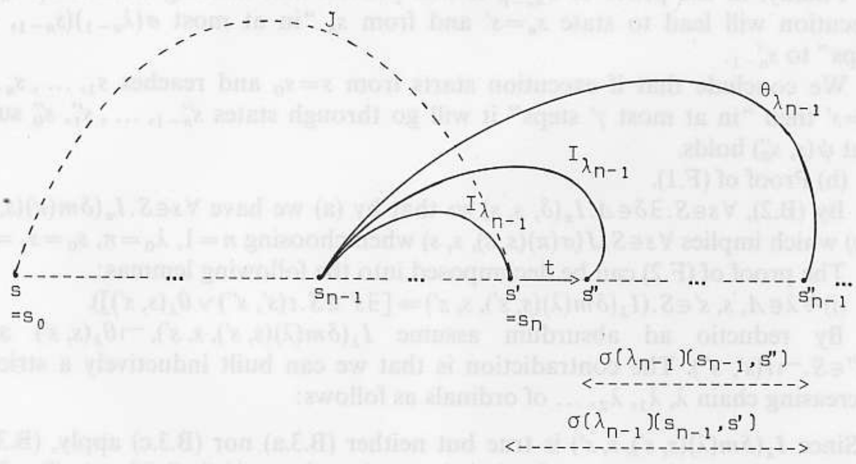
(k) $\forall \lambda \in \Lambda, \delta \in \Delta, s, s' \in S. [I_\lambda(\delta, s, s') \Rightarrow \exists s'' \in S. \theta_\lambda(s, s'')]$.

By transfinite induction on λ, assume (k) holds for all $\lambda' < \lambda$. We show that (k) holds for λ by reductio ad absurdum i.e. assuming $\forall s'' \in S. \neg \theta_\lambda(s, s'')$. We have $I_\lambda(\delta_0, s, s'_0)$ with $\delta_0 = \delta$ and $s'_0 = s'$. Assume we have built a chain $\delta_0 > \ldots > \delta_k$ with $I_\lambda(\delta_k, s, s'_k)$. Since $\neg \theta_\lambda(s, s'_k)$ it follows from (B.3.a) or (B.3.b) and the induction hypothesis that $\exists \delta_{k+1} < \delta_k, s'_{k+1}. I_\lambda(\delta_{k+1}, s, s'_{k+1})$. The contradiction is that, in this way, we can built an infinite strictly decreasing chain of ordinals. Q.E.D.

(l) $\forall \gamma' \in \Gamma, s, s', s'' \in S.[[J(\gamma', s, s') \wedge \neg\psi(s, s') \wedge t(s', s'')] \Rightarrow \exists \gamma'' < \gamma'.J(\gamma'', s, s'')]$.

Assuming $[J(\gamma', s, s') \wedge \neg\psi(s, s') \wedge t(s', s'')]$ we have by (f) that $s_n = s'$, $I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s_n), s_{n-1}, s_n)$ and $\neg\theta_{\lambda_{n-1}}(s_{n-1}, s_n)$ where n is the number of the innermost of all unfinished recursive subproofs used to get from s to s'. Since (B.3.c) does not apply, only two cases have to be considered:

Case 1. If s'' is considered in the inevitability proof for $\theta_{\lambda_{n-1}}$, we have:



(B.3.a) applies to $I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s_n), s_{n-1}, s_n)$ or (B.3.b) applies to it for some $\lambda' < \lambda_{n-1}$ such that $\theta_{\lambda'}(s_n, s'')$.

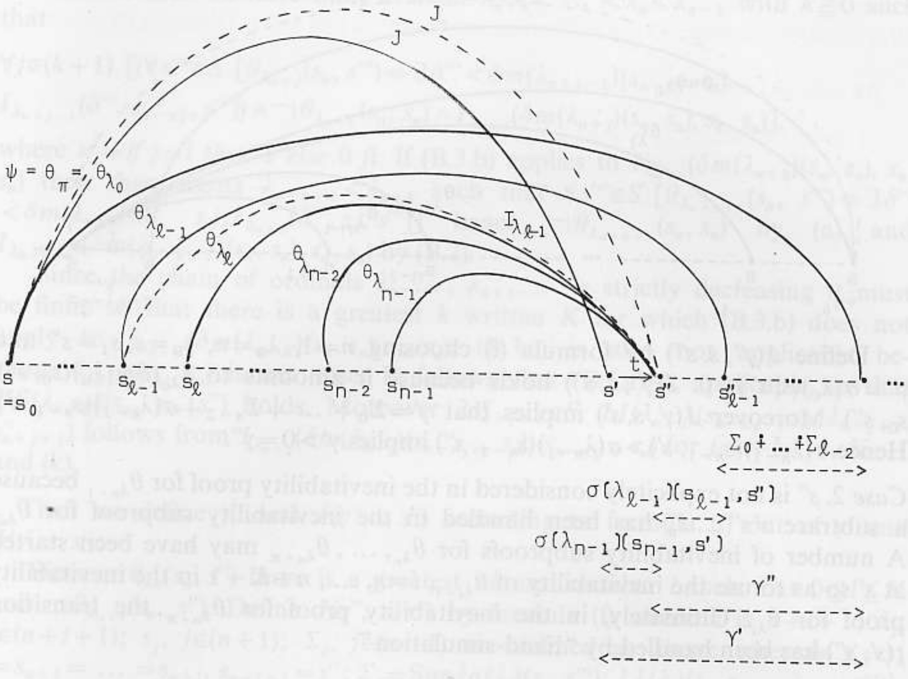
In the first case we have $HS(\lambda_{n-1})(s_{n-1}, s', s'')$ and in the second $LI(\lambda_{n-1})(s_{n-1}, s', \lambda', s'')$. In both cases we observe that (b) implies $I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s''), s_{n-1}, s'')$ and from $\neg\theta_{\lambda_{n-1}}(s_{n-1}, s')$ and (e) we derive $\sigma(\lambda_{n-1})(s_{n-1}, s') > \sigma(\lambda_{n-1})(s_{n-1}, s'')$.

Case 1.1. $\neg\theta_{\lambda_{n-1}}(s_{n-1}, s'')$ (s'' is not a final state for $\theta_{\lambda_{n-1}}$ (so that $s'' \neq s''_{n-1}$ in the above diagram)).

We have just shown that $\sigma(\lambda_{n-1})(s_{n-1}, s') > \sigma(\lambda_{n-1})(s_{n-1}, s'')$ hence $\gamma'' = \Sigma_0 + \ldots + \Sigma_{n-2} + \sigma(\lambda_{n-1})(s_{n-1}, s'') < \Sigma_0 + \ldots + \Sigma_{n-2} + \sigma(\lambda_{n-1})(s_{n-1}, s') = \gamma'$. If we let $J(\gamma'', s, s'')$ be $J(\gamma', s, s')$ with γ'', s'' substituted for γ', s' we have $\gamma'' < \gamma' \wedge J(\gamma'', s, s'')$.

Case 1.2. $\theta_{\lambda_{n-1}}(s_{n-1}, s'')$ (s'' is a final state for $\theta_{\lambda_{n-1}}$ (so that $s'' = s''_{n-1}$ in the above diagram)).

Let l be the smallest natural number such that $l < n$ and $\theta_{\lambda_l}(s_l, s'')$ holds so that if $l > 0$ then we have $\neg\theta_{\lambda_{l-1}}(s_{l-1}, s'')$. Intuitively, this is the case when s'' is a final state for $\theta_{\lambda_{n-1}}, \ldots, \theta_{\lambda_l}$. So, getting from s' to s'' causes the end of unfinished recursive subproofs for $\theta_{\lambda_l}, \ldots, \theta_{\lambda_{n-1}}$ used to get from s to s':



First observe that by (e), we have $\sigma(\lambda_j)(s_j, s'')=0$ for $j=l, \ldots, n-1$.

Let us show that $I_{\lambda_j}(\delta m(\lambda_j)(s_j, s''), s_j, s'')$ holds for $j=\max(l-1, 0), \ldots, n-1$. The case $j=n-1$ has already been considered. If $\max(0, l-1) \leqq j < n-1$ then $J(\gamma', s, s')$ implies $\exists s''' \in S.LI(\lambda_j)(s_j, s_{j+1}, \lambda_{j+1}, s''')$ hence (b), $\theta_{\lambda_{j+1}}(s_{j+1}, s'')$ and (a) imply $I_{\lambda_j}(\delta m(\lambda_j)(s_j, s''), s_j, s'')$.

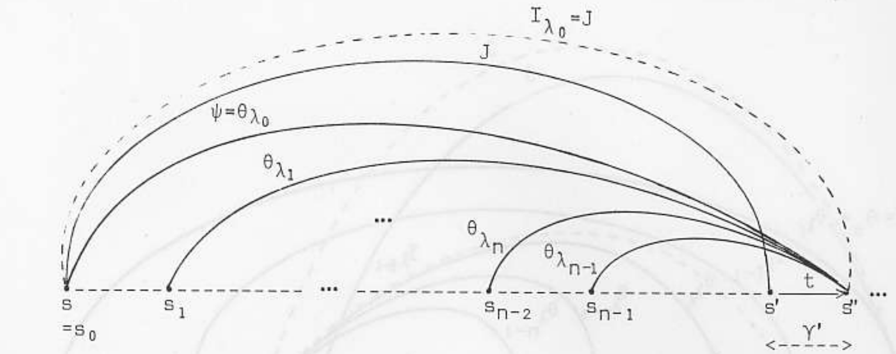
Case 1.2.1. $l>0$ hence $n>1$ (end of the inevitability subproofs for $\theta_{\lambda_{n-1}}, \ldots, \theta_{\lambda_l}$).

Define $J(\gamma'', s, s'')$ by formula (f) choosing $n=l, \lambda_0, \ldots, \lambda_{l-1}, s_0, \ldots, s_{l-1}$ and $\Sigma_0, \ldots, \Sigma_{l-2}$ as defined by $J(\gamma', s, s')$, $s_l=s''$ and $\gamma''=\Sigma_0 \dotplus \ldots \dotplus \Sigma_{l-2} \dotplus \sigma(\lambda_{l-1})(s_{l-1}, s'')$. Then $J(\gamma'', s, s'')$ holds because we have already proved that $I_{\lambda_{l-1}}(\delta m(\lambda_{l-1})(s_{l-1}, s_l), s_{l-1}, s_l)$ is true and $\neg\theta_{\lambda_{l-1}}(s_{l-1}, s_l)$ follows from the definition of l and s_l.

From $\sigma(\lambda_{n-1})(s_{n-1}, s') > \sigma(\lambda_{n-1})(s_{n-1}, s'')=0$ we derive $\Sigma_l \dotplus \ldots \dotplus \Sigma_{n-2} \dotplus \sigma(\lambda_{n-1})(s_{n-1}, s')>0$. We know that $\theta_{\lambda_l}(s_l, s'')$ holds and $J(\gamma', s, s')$ implies $\exists s_l'' \in S.LI(\lambda_{l-1})(s_{l-1}, s_l, \lambda_l, s_l'')$ hence by (b) we have $LI(\lambda_{l-1})(s_{l-1}, s_l, \lambda_l, s'')$. By definition of Σ_{l-1} we derive that $\Sigma_{l-1} \geqq \sigma(\lambda_{l-1})\ (s_{l-1}, s'')$. If follows that $\gamma''=(\Sigma_0 \dotplus \ldots \dotplus \Sigma_{l-2} \dotplus \sigma_{l-1}(\lambda_{l-1})(s_{l-1}, s'')) \leqq (\Sigma_0 \dotplus \ldots \dotplus \Sigma_{l-2} \dotplus \Sigma_{l-1})$
$<(\Sigma_0 \dotplus \ldots \dotplus \Sigma_{l-1} \dotplus \Sigma_l \dotplus \ldots \dotplus \Sigma_{n-2} \dotplus \sigma(\lambda_{n-1})(s_{n-1}, s'))=\gamma'$.

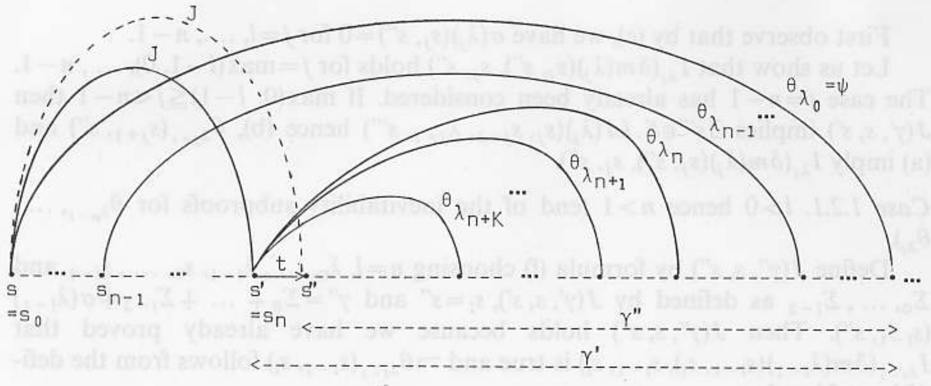
Case 1.2.2. $l=0$ (end of the inevitability proof for ψ).

Intuitively, this is the case when s'' is a final state for $\theta_{\lambda_{n-1}}, \ldots, \theta_{\lambda_0}=\psi$ so that the whole proof is finished. Therefore the above diagram becomes:



Define $J(\gamma'', s, s'')$ by formula (f) choosing $n=1$, $\lambda_0=\pi$, $s_0=s$, $s_1=s''$ and $\gamma''=\sigma(\lambda_0)(s, s'')=0$. $J(\gamma'', s, s'')$ holds because it amounts to $I_{\lambda_0}(\delta m(\lambda_0)(s_0, s''), s_0, s'')$. Moreover $J(\gamma', s, s')$ implies that $\gamma'=\Sigma_0+\ldots+\Sigma_{n-2}+\sigma(\lambda_{n-1})(s_{n-1}, s')$. Hence $\sigma(\lambda_{n-1})(s_{n-1}, s')>\sigma(\lambda_{n-1})(s_{n-1}, s'')$ implies $\gamma'>0=\gamma''$.

Case 2. s'' is not explicitely considered in the inevitability proof for $\theta_{\lambda_{n-1}}$ because a subtrace $s's''\ldots s''_n$ has been handled in the inevitability subproof for θ_{λ_n}. A number of inevitability subproofs for $\theta_{\lambda_n}, \ldots, \theta_{\lambda_{n+K}}$ may have been started at s' so as to use the inevitability of $\theta_{\lambda_{i+1}}$, $i=n, \ldots, n+K-1$ in the inevitability proof for θ_{λ_i}. Ultimately, in the inevitability proof for $\theta_{\lambda_{n+K}}$ the transition $t(s', s'')$ has been handled by "hand-simulation":



Intuitively we get $\gamma''<\gamma'$ such that $J(\gamma'', s, s'')$ holds from what is specified by $J(\gamma', s, s')$ and by recording the above diagram (where all inevitability proofs for $\theta_{\lambda_n}, \ldots, \theta_{\lambda_{n+K}}$ are started at s' and possible useless $\theta_{\lambda'}$ such that $\theta_{\lambda'}(s', s')$ or $\theta_{\lambda'}(s', s'')$ holds are omitted).

. First, we formally define $\theta_{\lambda_n}, \ldots, \theta_{\lambda_{n+K}}$:

In this case (B.3.b) applies to $I_{\lambda_{n-1}}(\delta m(\lambda_{n-1})(s_{n-1}, s_n), s_{n-1}, s_n)$ for some $\lambda'<\lambda_{n-1}$ such that $\neg\theta_{\lambda'}(s_n, s'')$. If we define λ_n as λ', then we have $\lambda_n<\lambda_{n-1}$ and $\forall s'''\in S.[\theta_{\lambda_n}(s_n, s''')\Rightarrow\exists\delta'''<\delta m(\lambda_{n-1})(s_{n-1}, s_n).I_{\lambda_{n-1}}(\delta''', s_{n-1}, s''')]$. It follows that $\neg\theta_{\lambda_n}(s_n, s_n)$ since otherwise $\exists\delta'''<\delta m(\lambda_{n-1})(s_{n-1}, s_n).I_{\lambda_{n-1}}(\delta''', s_{n-1}, s_n)$ in contradiction with the definition (a) of $\delta m(\lambda_{n-1})(s_{n-1}, s_n)$. Moreover (B.2) implies $I_{\lambda_n}(\delta m(\lambda_n)(s_n, s_n), s_n, s_n)$.

Assume that we have built a chain $\lambda_{n+k} < \ldots < \lambda_n < \lambda_{n-1}$ with $k \geqq 0$ such that

$\forall j \in (k+1).[(\forall s''' \in S.[\theta_{\lambda_{n+j}}(s_n, s''') \Rightarrow \exists \delta''' < \delta m(\lambda_{n+j-1})(s_{n-\kappa_j^0}, s_n).$
$I_{\lambda_{n+j-1}}(\delta''', s_{n-\kappa_j^0}, s''')) \wedge \neg \theta_{\lambda_{n+1}}(s_n, s_n) \wedge I_{\lambda_{n+j}}(\delta m(\lambda_{n+j})(s_n, s_n), s_n, s_n)]$,

where $\kappa_j^l =$ *if* $j = l$ *then* 1 *else* 0 *fi*. If (B.3.b) applies to $I_{\lambda_{n+k}}(\delta m(\lambda_{n+k})(s_n, s_n), s_n, s_n)$ then there exists $\lambda_{n+k+1} < \lambda_{n+k}$ such that $\forall s''' \in S.[\theta_{\lambda_{n+k+1}}(s_n, s''') \Rightarrow \exists \delta''' < \delta m(\lambda_{n+k})(s_n, s_n). I_{\lambda_{n+k}}(\delta''', s_n, s''')]$ hence $\neg \theta_{\lambda_{n+k+1}}(s_n, s_n)$ by (a) and $I_{\lambda_{n+k+1}}(\delta m(\lambda_{n+k+1})(s_n, s_n), s_n, s_n)$ by (B.2).

Since the chain of ordinals $\lambda_n, \ldots, \lambda_{n+k}, \ldots$ is strictly decreasing it must be finite so that there is a greatest k written K for which (B.3.b) does not apply to $I_{\lambda_{n+K}}(\delta m(\lambda_{n+K})(s_n, s_n), s_n, s_n)$. (B.3.c) is also not applicable because $\theta_{\lambda_{n+K}}(s_n, s_n)$ is not true. It follows that (B.3.a) is applicable so that $HS(\lambda_{n+K})(s_n, s_n, s'')$ holds. Moreover $\exists s''_{n+j+1} \in S.\ LI(\lambda_{n+j})(s_{n-\kappa_j^{-1}}, s_n, \lambda_{n+j+1}, s''_{n+j+1})$ follows from $I_{\lambda_{n+j}}(\delta m(\lambda_{n+j})(s_{n-\kappa_j^{-1}}, s_n), s_{n-\kappa_j^{-1}}, s_n)$ for $j = -1, \ldots, K-1$ and (k).

. Then we define γ'' and $J(\gamma'', s, s'')$ according to formula (f) out of γ' and $J(\gamma', s, s')$:

Since $\neg \theta_{\lambda_n}(s_n, s'')$ there is a greatest natural number l such that $0 \leqq l \leqq K$ and $\neg \theta_{\lambda_{n+l}}(s_n, s'')$. Define $J(\gamma'', s, s'')$ by formula (f) where n is $n+l+1$; λ_j, $j \in (n+l+1)$; s_j, $j \in (n+1)$; Σ_j, $j \in (n-1)$ are defined as above whereas $s' = s_n = s_{n+1} = \ldots = s_{n+l}$, $s_{n+l+1} = s''$; $\Sigma_j = \mathrm{Sup}\{\sigma(\lambda_j)(s_j, s'''): LI(\lambda_j)(s_j, s_{j+1}, \lambda_{j+1}, s''')\}$, $j = n-1, \ldots, n+l-1$ and $\gamma'' = \Sigma_0 \dotplus \ldots \dotplus \Sigma_{n+l-1} \dotplus \sigma(\lambda_{n+l})(s_{n+l}, s_{n+l+1})$.

. Next we have to prove that $J(\gamma'', s, s'')$ holds:

Observe that we have already proved that $\forall i = n, \ldots, n+l, \ldots, n+K$, $\exists s''_i \in S. LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s''_i)$ and $\forall i = n, \ldots, n+l-1, \ldots, n+K-1$. $\neg \theta_{\lambda_i}(s_i, s_{i+1})$. Moreover we have $\neg \theta_{\lambda_{n+l}}(s_{n+l}, s_{n+l+1})$ by definition of l, s_{n+l} and s_{n+l+1}. Also $I_{\lambda_{n+l}}(\delta m(\lambda_{n+l})(s_{n+l}, s_{n+l+1}), s_{n+l}, s_{n+l+1})$ holds because it is implied by $HS(\lambda_{n+K})(s_n, s_n, s'')$ when $l = K$ or else $l < K$ and $\theta_{\lambda_{n+l+1}}(s_n, s'')$ implies $\exists \delta'''. I_{\lambda_{n+l}}(\delta''', s_{n-\kappa^0_{l+1}}, s'')$ hence by (a) $I_{\lambda_{n+l}}(\delta m(\lambda_{n+l})(s_n, s''), s_n, s'')$ is true. We conclude that $J(\gamma'', s, s'')$ holds.

. Finally, let us show that $\gamma'' < \gamma'$:

We have shown that for all $j = -1, \ldots, K-1$ there is some s''_{n+j+1} such that $LI(\lambda_{n+j})(s_{n-\kappa_j^{-1}}, s_n, \lambda_{n+j+1}, s''_{n+j+1}) = LI(\lambda_{n+j})(s_{n+j}, s_{n+j+1}, \lambda_{n+j+1}, s''_{n+j+1})$ holds. Moreover $\forall i = 1, \ldots, n+l$ we have $\neg \theta_{\lambda_i}(s_i, s_{i+1})$ so that by (e) it follows that $\sigma(\lambda_{n+j})(s_{n+j}, s_{n+j+1}) > \mathrm{Sup}\{\sigma(\lambda_{n+j})(s_{n+j}, s'''): LI(\lambda_{n+j})(s_{n+j}, s_{n+j+1}, \lambda_{n+j+1}, s''')\} \dotplus \sigma(\lambda_{n+j+1})(s_{n+j+1}, s_{n+j+1}) = \Sigma_{n+j} \dotplus \sigma(\lambda_{n+j+1})(s_{n+j+1}, s_{n+j+1})$ for all $j = -1, \ldots, l-1$. Thanks to this inequality and $s_{n+j+1} = s_{n+j+2}$ for $j = -1, \ldots, l-2$, we get $\sigma(\lambda_{n-1})(s_{n-1}, s_n) > \Sigma_{n-1} \dotplus \sigma(\lambda_n)(s_n, s_{n+1}) > \ldots > \Sigma_{n-1} \dotplus \ldots \dotplus \Sigma_{n+l-1} \dotplus \sigma(\lambda_{n+l})(s_{n+l}, s_{n+l})$. If $l = K$ then $HS(\lambda_{n+K})(s_n, s_n, s'')$, $s_{n+l} = s_n$, $\neg \theta_{\lambda_{n+l}}(s_n, s'')$ and (e) imply $\sigma(\lambda_{n+l})(s_{n+l}, s_{n+l}) > \sigma(\lambda_{n+l})(s_{n+l}, s'')$. Otherwise $l < K$ and we have $\neg \theta_{\lambda_{n+l}}(s_n, s_n)$, $LI(\lambda_{n+l})(s_n, s_n, \lambda_{n+l+1}, s'')$, $s_{n+l} = s_n$ so that by (e) $\sigma(\lambda_{n+l})(s_{n+l}, s_{n+l}) = \sigma(\lambda_{n+l})(s_n, s_n) > \sigma(\lambda_{n+l})(s_n, s'') = \sigma(\lambda_{n+l})(s_{n+l}, s'')$. In both cases we conclude that $\gamma' = \Sigma_0 \dotplus \ldots \dotplus \Sigma_{n-2} \dotplus \sigma(\lambda_{n-1})(s_{n-1}, s_n) > \Sigma_0 \dotplus \ldots \dotplus \Sigma_{n-2} \dotplus \Sigma_{n-1} \dotplus \ldots \dotplus \Sigma_{n+l-1} \dotplus \sigma(\lambda_{n+l})(s_{n+l}, s'') = \gamma''$. Q.E.D.

To conclude, (F.2) is an immediate consequence of (j) and (l). $\square$

Example 6.1. (*Illustration of the proof of Theorem* 6.2.1). The transition system $\langle S, t\rangle$ corresponding to the following program (taken from Dijkstra [6]):

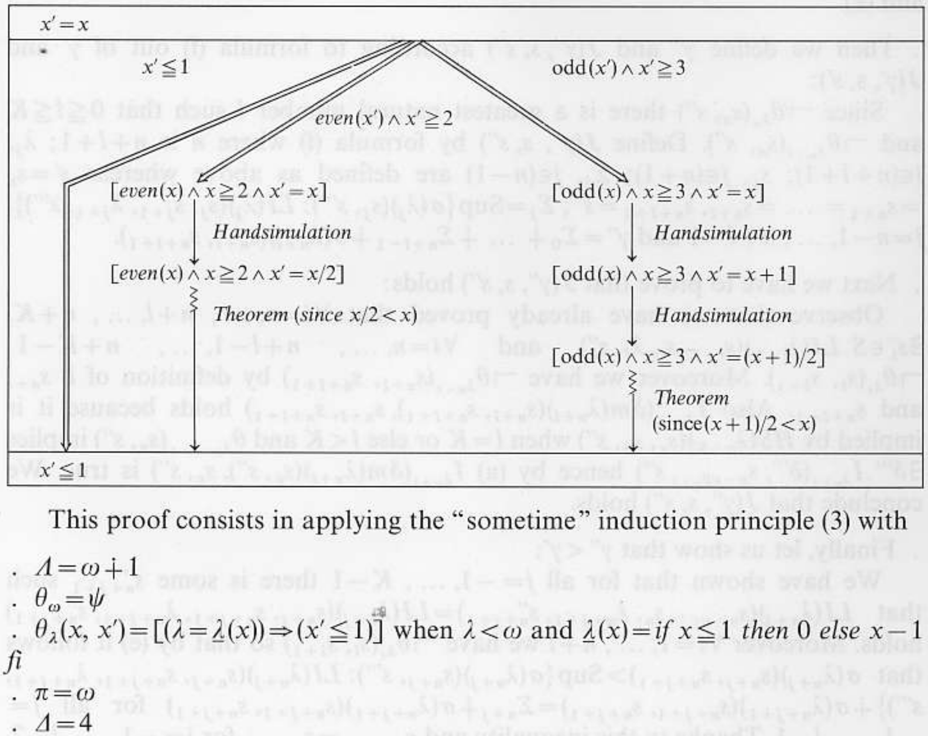
```
do odd (X) and X ≥ 3 → X := X + 1
□ even (X) and X ≥ 2 → X := X/2
od
```

is defined by:

. S is the set of integers,
. $t(x, x') = [(\text{odd}(x) \wedge x \geq 3 \wedge x' = x+1) \vee (\text{even}(x) \wedge x \geq 2 \wedge x' = x/2)]$.

A proof that $\psi(x, x') = [x' \leq 1]$ is inevitable is given by the following proof chart (where x denotes the initial value and x' the current value of program variable X):

Theorem. (by induction on x (when $x > 1$))


$x' = x$
$x' \leq 1$
$\text{odd}(x') \wedge x' \geq 3$
$\mathit{even}(x') \wedge x' \geq 2$
$[\mathit{even}(x) \wedge x \geq 2 \wedge x' = x]$
Handsimulation
$[\mathit{even}(x) \wedge x \geq 2 \wedge x' = x/2]$
Theorem (*since* $x/2 < x$)
$[\text{odd}(x) \wedge x \geq 3 \wedge x' = x]$
Handsimulation
$[\text{odd}(x) \wedge x \geq 3 \wedge x' = x+1]$
Handsimulation
$[\text{odd}(x) \wedge x \geq 3 \wedge x' = (x+1)/2]$
Theorem
(since $(x+1)/2 < x$)
$x' \leq 1$

This proof consists in applying the "sometime" induction principle (3) with

. $\Lambda = \omega + 1$
. $\theta_\omega = \psi$
. $\theta_\lambda(x, x') = [(\lambda = \underline{\lambda}(x)) \Rightarrow (x' \leq 1)]$ when $\lambda < \omega$ and $\underline{\lambda}(x) =$ *if* $x \leq 1$ *then* 0 *else* $x - 1$ *fi*
. $\pi = \omega$
. $\Delta = 4$
. $I_\omega(\delta, x, x') = [(\delta = 1 \wedge x = x') \vee (\delta = 0 \wedge x' \leq 1)]$
. $I_\lambda(\delta, x, x') = [(\lambda = \underline{\lambda}(x)) \Rightarrow ((\delta = 3 \wedge x' = x) \vee (\delta = 2 \wedge t(x, x')) \vee (\delta = 1 \wedge \text{odd}(x) \wedge t^2(x, x')) \vee (\delta = 0 \wedge x' \leq 1))]$.

(a) By definition of $\delta m \in (\Lambda \to (S \times S \to \Delta))$ we have:

$$\begin{aligned}\delta m(\lambda)(x, x') &= 3 \quad \text{iff } [\lambda < \omega \wedge x = x' \wedge x' > 1]\\ &= 2 \quad \text{iff } [\lambda < \omega \wedge t(x, x') \wedge x' \geq 1]\\ &= 1 \quad \text{iff } [(\lambda = \omega \wedge x = x' > 1) \vee (\lambda < \omega \wedge \text{odd}(x) \wedge t^2(x, x') \wedge x' > 1)]\\ &= 0 \quad \text{otherwise}\end{aligned}$$

(b) By definition of HS we have:

$$HS(\omega)(x, x', x'') = [x = x' \wedge t(x', x'') \wedge x'' \leqq 1]$$

and when $\lambda < \omega$,

$$HS(\lambda)(x, x', x'') = [[(x = x') \vee (t(x, x') \wedge [(\lambda = \underline{\lambda}(x)) \Rightarrow (\mathrm{odd}(x) \vee x'' \leqq 1)]) \vee (\mathrm{odd}(x) \wedge t^2(x, x') \wedge [(\lambda = \underline{\lambda}(x)) \Rightarrow (x'' \leqq 1)])] \wedge t(x', x'')]$$

The same way for LI:

$$LI(\omega)(x, x', \lambda', x'') = [x' = x > 1 \wedge \lambda' = \underline{\lambda}(x') \wedge x'' \leqq 1]$$

and when $\lambda < \omega$,

$$\begin{aligned} &LI(\lambda)(x, x', \lambda', x'') \\ &= [[(x = x') \vee t(x, x') \vee (\mathrm{odd}(x) \wedge t^2(x, x'))] \wedge x' > 1 \wedge \lambda > \lambda' = \underline{\lambda}(x') \wedge x'' \leqq 1] \end{aligned}$$

(Observe that when (B.3.a) and (B.3.b) are both true (e.g. when $x = x' = 2$ and $x'' = 1$) so are $HS(\lambda)(x, x', x'')$ and $LI(\lambda)(x, x', \lambda', x'')$ because there is no means of knowing whether hand-simulation or induction has been used in the proof).

(e) We can now determine $\sigma(\lambda)(x, x')$. Because $LI(\lambda)(x, x', \lambda', x'')$ implies $\theta_\lambda(x, x'')$ hence $\sigma(\lambda)(x, x'') = 0$, formula (e) amounts to:

$$\sigma(\lambda)(x, x') = \mathrm{Sup}\{\alpha + 1 : \neg\theta_\lambda(x, x') \wedge ([HS(\lambda)(x, x', x'') \wedge \alpha = \sigma(\lambda)(x, x'')] \vee [\exists \lambda' < \lambda, x'' \in S . LI(\lambda)(x, x', \lambda', x'') \wedge \alpha = \sigma(\lambda')(x', x')])\}.$$

As in the proof of Theorem 6.2.1, it is not necessary to look for a direct (non-recursive) definition of σ because we shall only need the following properties:

. $t(x, x') \Rightarrow [\neg\theta_{\underline{\lambda}(x)}(x, x) \wedge HS(\underline{\lambda}(x))(x, x, x')] \Rightarrow [\sigma(\underline{\lambda}(x))(x, x') < \sigma(\underline{\lambda}(x))(x, x)]$

. $[\mathrm{even}(x) \wedge t(x, x') \wedge t(x', x'')] \Rightarrow [\neg\theta_{\underline{\lambda}(x')}(x', x')$
$\wedge HS(\underline{\lambda}(x'))(x', x', x'') \wedge \neg\theta_{\underline{\lambda}(x)}(x, x')$
$\wedge \exists x''' . LI(\underline{\lambda}(x))(x, x', \underline{\lambda}(x'), x''')] \Rightarrow [\sigma(\underline{\lambda}(x'))(x', x'') < \sigma(\underline{\lambda}(x'))(x', x')$
$< \sigma(\underline{\lambda}(x))(x, x')]$

. $[\mathrm{odd}(x) \wedge t(x, x') \wedge t(x', x'')] \Rightarrow [(\exists x''' . LI(\underline{\lambda}(x))(x, x'', \underline{\lambda}(x''), x''') \vee \theta_{\underline{\lambda}(x)}(x, x''))$
$\wedge HS(\underline{\lambda}(x))(x, x', x'')] \Rightarrow [(\sigma(\underline{\lambda}(x''))(x'', x'') < \sigma(\underline{\lambda}(x))(x, x'') \vee \sigma(\underline{\lambda}(x''))(x'', x'') = 0)$
$\wedge \sigma(\underline{\lambda}(x))(x, x'') < \sigma(\underline{\lambda}(x))(x, x')] \Rightarrow [\sigma(\underline{\lambda}(x''))(x'', x'') < \sigma(\underline{\lambda}(x))(x, x')]$

. $[\mathrm{odd}(x) \wedge t^2(x, x') \wedge t(x', x'')] \Rightarrow [\neg\theta_{\underline{\lambda}(x')}(x', x') \wedge HS(\underline{\lambda}(x'))(x', x', x'')$
$\wedge \neg\theta_{\underline{\lambda}(x)}(x, x')$
$\wedge \exists x''' . LI(\underline{\lambda}(x))(x, x', \underline{\lambda}(x'), x''')] \Rightarrow [\sigma(\underline{\lambda}(x'))(x', x'') < \sigma(\underline{\lambda}(x))(x, x')]$.

(f) In the definition of $J(\gamma', x, x')$ the case $n = 1$ reduces to $[(x = x' \vee x' \leqq 1) \wedge \gamma' = \sigma(\omega)(x, x')]$. Otherwise $n > 1$ and for all $i \in (n \sim 0)$ we have $\Sigma_{i-1} = 0$ because $LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s'')$ implies $s'' \leqq 1$ whence $\sigma(\lambda_{i-1})(s_{i-1}, s'') = 0$. Moreover λ is a function of s because $\lambda_0 = \omega$ and $LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s_i'')$ implies $\lambda_i = \underline{\lambda}(s_i)$ for all $i \in (n \sim 0)$. When $i = 1$, this also implies $s_0 = s_1 = x > 1$. When $i \in (2, \ldots, n-1)$

the terms $\exists s_i'' \in S. LI(\lambda_{i-1})(s_{i-1}, s_i, \lambda_i, s_i'')$ are of the form:

$$\begin{aligned}\underline{\lambda}(s_{i-1}) > \underline{\lambda}(s_i) \wedge s_i > 1 \wedge [(s_{i-1} = s_i) \vee t(s_{i-1}, s_i) \vee (\mathrm{odd}(s_{i-1}) \wedge t^2(s_{i-1}, s_i))]\\ = s_i > 1 \wedge [(\mathrm{even}(s_{i-1}) \wedge t(s_{i-1}, s_i)) \vee (\mathrm{odd}(s_{i-1}) \wedge t^2(s_{i-1}, s_i))]\end{aligned}$$

because $s_{i-1} = s_i$ or $s_i = s_{i-1} + 1$ (when $\mathrm{odd}(s_{i-1}) \wedge t(s_{i-1}, s_i)$) are not compatible with $\underline{\lambda}(s_{i-1}) > \underline{\lambda}(s_i) \wedge s_i > 1$. The term $\neg \theta_{\lambda_i}(s_i, s_{i+1})$ amounts to $s_{i+1} > 1$ for $i \in (n \sim 0)$. It follows that:

$$\begin{aligned}J(\gamma', x, x') = [&(x = x' \vee x' \leqq 1) \wedge \gamma' = \sigma(\omega)(x, x')]\\ &\vee [\exists n > 1, s \in (n+1 \to S). x = s_0 = s_1 > 1 \wedge s_n = x' > 1\\ &\wedge \forall i \in (2, \dots, n-1). [(\mathrm{even}(s_{i-1}) \wedge t(s_{i-1}, s_i))\\ &\qquad\qquad \vee (\mathrm{odd}(s_{i-1}) \wedge t^2(s_{i-1}, s_i))]\\ &\wedge [(s_{n-1} = s_n) \vee t(s_{n-1}, s_n) \vee (\mathrm{odd}(s_{n-1}) \wedge t^2(s_{n-1}, s_n))]\\ &\wedge \gamma' = \sigma(\underline{\lambda}(s_{n-1}))(s_{n-1}, s_n)]\end{aligned}$$

If we let $t'(x, x')$ be $[(\mathrm{even}(x) \wedge t(x, x')) \vee (\mathrm{odd}(x) \wedge t^2(x, x'))]$ this can be written more simply as:

$$\begin{aligned}J(\gamma', x, x') = [&\exists x'' \in S. t'^*(x, x'') \wedge (x' > 1) \Rightarrow\\ &[(x'' = x') \vee t(x'', x') \vee (\mathrm{odd}(x'') \wedge t^2(x'', x'))] \wedge \gamma' = \sigma(\underline{\lambda}(x''))(x'', x')]\end{aligned}$$

where t'^* is the reflexive transitive closure of t'.

Observe that this formula captures the essence of the proof by Burstall's method which consists in considering one step for even states and two steps for odd states. It remains to show that $J(\gamma', x, x')$ satisfies (F.1) (this is obvious) and (F.2). Obviously, if $\neg\psi(x, x')$ then $x' > 1$ whence $\exists x''' \in S. t(x', x''')$. But also if $t(x', x''')$ then four cases have to be considered:

. If $x' = x''$ then $t'^*(x, x') \wedge t(x', x''')$ implies $J(\sigma(\underline{\lambda}(x'))(x', x'''), x', x''')$ and $\sigma(\underline{\lambda}(x'))(x', x''') < \sigma(\underline{\lambda}(x'))(x'', x')$,

. If $\mathrm{even}(x'')$ and $t(x'', x')$ then $t'^*(x, x'') \wedge (\mathrm{even}(x'') \wedge t(x'', x')) \wedge t(x', x''')$ implies $t'^*(x, x') \wedge t(x', x''')$ hence $J(\sigma(\underline{\lambda}(x'))(x', x'''), x, x''')$ and $\sigma(\underline{\lambda}(x'))(x', x''') < \sigma(\underline{\lambda}(x''))(x'', x')$,

. If $\mathrm{odd}(x'')$ and $t(x'', x')$ then $t'^*(x, x'') \wedge (\mathrm{odd}(x'') \wedge t(x'', x') \wedge t(x', x'''))$ implies $t'^*(x, x''') \wedge (x''' = x''')$ hence $J(\sigma(\underline{\lambda}(x'''))(x''', x'''), x, x''')$ and $\sigma(\underline{\lambda}(x'''))(x''', x''') < \sigma(\underline{\lambda}(x''))(x'', x')$,

. If $\mathrm{odd}(x'')$ and $t^2(x'', x')$ then $t'^*(x, x'') \wedge (\mathrm{odd}(x'') \wedge t^2(x'', x')) \wedge t(x', x''')$ implies $t'^*(x, x') \wedge t(x', x''')$ hence $J(\sigma(\underline{\lambda}(x'))(x', x'''), x, x''')$ and $\sigma(\underline{\lambda}(x'))(x', x''') < \sigma(\underline{\lambda}(x''))(x'', x')$.

The translation technique used in the proof of Theorem 6.2.1 has to be very general. Hence for particular examples the resulting "always" invariant may not be the simplest one that can be imagined. In the above example, the invariant:

$$\begin{aligned}J(\gamma', x, x') = &\mathit{if}\ x' \leqq 1\ \mathit{then}\ \gamma' = 0\\ &\mathit{elsif}\ \mathrm{even}(x')\ \mathit{then}\ \gamma' = x' - 1\\ &\mathit{else}\ \gamma' = x' + 1\ \mathit{fi}\end{aligned}$$

would be much more understandable. So we do not pretend that this general translation technique preserves "naturalness" of proofs. □

7. Conclusion

We have formalized and generalized Floyd's and Burstall's inevitability proof methods in such a way that the "always" induction principle is a particular case of the "sometime" induction principle. This is not a definitive advantage for the "sometime" method since we have also proved that any proof by one method can be systematically rewritten into a proof by the other method. The advantage of "sometime" over "always" is the ability to decompose the proof of a theorem into independent proofs of lemmas. However the principle of separation of concerns has been clearly adhered to in most presentations of the "always" induction principle. The same approach can be applied to the "sometime" induction principle. This should lead to a different explanation of Burstall's proof method, a necessary step before concluding that "sometime" is always better than "always".

Acknowledgements: We wish to thank the anonymous referees for their helpful comments.

References

1. Back, R.J.: Semantics of unbounded nondeterminism, in Proc. 7th ICALP, LNCS 85, pp. 51–63. Berlin, Heidelberg, New York: Springer 1980
2. Burstall, R.M.: Program proving as hand simulation with a little induction. Information Processing 74, pp. 308–312. Amsterdam: North-Holland 1974
3. Cousot, P., Cousot, R.: Induction principles for proving invariance properties of programs. In: Tools and notions for program construction. (D. Neel, ed.), pp. 75–119. Cambridge: University Press 1982
4. Cousot, P., Cousot, R.: "A la Floyd" induction principles for proving inevitability properties of programs. In: Algebraic methods in semantics. (M. Nivat, J.C. Reynolds, eds.), pp. 277–312. Cambridge: University Press 1985
5. Cousot, P., Cousot, R.: "A la Burstall" induction principles for proving inevitability properties of programs, Research Report LRIM-83-08, Univ. of Metz, France, 1983
6. Dijkstra, E.W.: A sequel to EWD 592, EWD 600, Burroughs Corp., Nuemen, The Netherlands 1977
7. Floyd, R.: Assigning meaning to programs. In: Proc. Symp. Appl. Math., 19. (Schwartz J.T. (ed.)), Am. Math. Soc., pp. 19–32, Providence, 1967
8. Gries, D.: Is SOMETIME ever better than ALWAYS? ACM TOPLAS, **1**, 258–265 (1979)
9. Keller, R.M.: Formal verification of parallel programs, **19**, 371–384 (1976)
10. Lamport, L.: Proving the correctness of multiprocess programs, IEEE Trans. Soft. Eng., **3**, 125–143 (1977)
11. Manna, Z., Pnueli, A.: How to cook a temporal proof system for your pet language, ACM POPL, **10**, 141–154 (1983)
12. Manna, Z., Waldinger, R.: Is SOMETIME sometimes better than ALWAYS? Intermittent assertions in proving program correctness, **21**, 159–172 (1978)
13. Monk, J.D.: Introduction to set theory, New York: McGraw-Hill 1969
14. Owicki, S., Lamport, L.: Proving liveness properties of concurrent programs. ACM TOPLAS **4**, 455–495 (1982)

Received December 5, 1983/August 18, 1986