stichting mathematisch centrum

MC

AFDELING ZUIVERE WISKUNDE ZN 42/72 APRIL

P. VAN EMDE BOAS

GAP AND OPERATOR GAP

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat 49, Amsterdam.

Introduction

This note contains a generalisation of Borodins gap theorem [3] and Constables operator gap theorem [4]. This generalisation does not only present these gap phenomena in situations dealing with complexity class-like resource bound classes of recursive functions (like honesty classes for example) but also indicates that the gap phenomenon depends only on the second Blum axiom [2] (the property of being a measured set). In fact our result shows that any measured set contains arbitrary large composition and operator gaps.

We present an axiomatisation of the notion of an "acceptance relation". This notion can be equivalently translated into the notion of a measured set but we have introduced it to restrict it in the future by some further axiom related to the first Blum axiom. As we shall see this first axiom of Blum plays no part in the proof of gap theorems so we leave this question open for the present discussion.

The proofs given are derived from proofs by P. Young [11]. The essential modification is a computation "one stage in advance" in the case of the operator gap algorithm. This is needed to produce "closed local gap sections".

As corollaries we have gap and operator gap theorems for several classes of abstract resource bound classes like honesty classes, complexity classes modulo a recursive enumerable class of sets of exeptional points, "summed complexity classes", and the result mentioned above about measured sets.

Conventions and notations

By a function we mean a partial recursive function. Whenever the function is defined for all values of the argument we explicitely call it a total function. The domain under consideration consists of the natural numbers (with zero). The set $\{x \mid a \leq x \leq b\}$ is denoted as $[a,b]$.
For total functions the inequality $f \leq g$ means $\forall x[f(x) \leq g(x)]$. By an

operator we denote a total effective operator (cf. [8]). In working with an operator Γ we consider Γ to be some procedure which allows us to compute $\Gamma(t)(x)$ whenever some machine for t is given; moreover $\Gamma(t)(x)$ is total whenever $t(x)$ is total and to compute $\Gamma(t)(x)$ we only use the values of t at some finite set of arguments of t called the <u>support</u> of (the computation of) $\Gamma(t)(x)$; finally the result is independent of the program used for t. We have the following important observation: whenever $\Gamma(t)(x)$ has its support contained within the interval $[a,b]$ and u is some other (eventually partial) function which satisfies $t|[a,b] = u|[a,b]$ then $\Gamma(t)(x) = \Gamma(u)(x)$.

In our discussions we have some complexity measure in our mind which is denoted by $\{\{\phi_i\},\{\Phi_i\}\}$. (cf. [2,6]).

The reader should be warned that our concept of a complexity class is distinct from the concept as defined in other papers dealing with this subject. In the first place we allow partial functions to be contained in a complexity class, even if the name is total. Furthermore we allow our names to be partial without putting any restriction on the domains of the functions contained in the complexity class. Our precise concept will become clear from the sequel.

If $P(x)$ is a predicate we write $\overset{\infty}{\forall} x\ P(x)$ to express that $P(x)$ holds for all natural x with at most a finite number of exceptions. $\overset{\infty}{\exists} x\ P(x)$ means that $P(x)$ is true for an infinite number of values x.

In discussions we use a fixed set of pairing and unpairing functions. $\langle x,y\rangle$ is the pair index of the pair x,y and $\pi_1(x)$, $\pi_2(x)$ are total recursive functions computing the first and second coordinate of a pair.

1. Abstract resource bound classes and acceptance relations

1.1. Informal discussion

The definition of a complexity class C_t can be given as:

$$\phi_i \in C_t \quad \underline{\text{iff}} \quad \overset{\infty}{\forall}_x \; \Phi_i(x) \leq t(x)$$

In principle the inequality $\Phi_i(x) \leq t(x)$ is undefined whenever $\Phi_i(x)$ or $t(x)$ is undefined. In practice one uses the following interpretation:

The value $t(x)$ should be considered to be some test value. If the computation of $t(x)$ fails to converge we have no test value and hence no test either. This means that in testing the program ϕ_i no test is performed on the argument x.

If $t(x)$ converges and yields a value z we remind ourselves that $\Phi_i(x) \leq z$ stands for the finite disjunction:

$$\Phi_i(x) = 0 \quad \underline{\text{or}} \quad \Phi_i(x) = 1 \quad \underline{\text{or}} \quad \ldots \quad \underline{\text{or}} \quad \Phi_i(x) = z$$

which can be tested component wise. In this concept the case $\Phi_i(x) \leq z$ with $\Phi_i(x)$ diverging is to be considered <u>false</u>.

There is a difference between the case with finite $\Phi_i(x)$ and the case that $\Phi_i(x)$ diverges.
In the first case $\Phi_i(x) \leq z'$ will become true for sufficiently large z' and in the second case it will never become true. The reader should keep this artificial distinction in mind while reading the discussion below.

Next we consider a possible definition of a Honesty class.

ϕ_i is R-honest if it satisfies nearly everywhere the R-honesty condition "$\Phi_i(x) \leq R(x,\phi_i(x))$".

There is a large number of distinct interpretations one could give of the above assertion "does satisfy the R-honesty condition".

Interpretation 1: Enumerate the graph of $\phi_i(x)$ and for each pair $<x,\phi_i(x)>$ produced compute $R(x,\phi_i(x))$. If this converges (R does not need to be total) test whether $\Phi_i(x) \leq R(x,\phi_i(x))$. The number of violations has to be finite.

This interpretation is consistent with the notion of honesty as it is normally used. It is a bad interpretation if one is interested in defining a bound-function R depending on the behaviour of the ϕ_i.

An interpretation restricted to a single argument x is the following:

Interpretation 2: Compute $\phi_i(x)$ and if this converges compute $R(x,\phi_i(x))$. If this converges also test whether $\Phi_i(x) \leq R(x,\phi_i(x))$. If this is not the case ϕ_i is not R-honest at x. ϕ_i should not be not-R-honest at x for infinitely many x.

The above interpretation does still not allow to isolate the bound function R from the honesty condition. A local interpretation is the following:

Interpretation 3: We say that ϕ_i satisfies the honesty condition with value z at the argument-pair $<x,y>$ when we have $\phi_i(x) = y$ and $\Phi_i(x) \leq z$. We say that ϕ_i violates the honesty condition with value z at the argument-pair $<x,y>$ if $\Phi_i(x) = y$ and $\Phi_i(x) > z$. Otherwise the honesty condition (with value z) does not apply to ϕ_i at the argument-pair $<x,y>$. ϕ_i should not violate the honesty condition with value $R(x,y)$ at infinitely many arguments-pairs $<x,y>$.

It is natural to have ϕ_i not violating a honesty condition (at x) if ϕ_i does not converge. In interpretation 1 such a point does not appear in the enumeration; in interpretation 2 the corresponding point x is never marked as an argument where ϕ_i is not R-honest while in the third case the honesty condition is found to be not applicable at $<x,y>$ for all y.

It is also natural to have ϕ_i not violating a honesty condition at x if R(x,y) happens to be undefined for $y = \phi_i(x)$. In the first two interpretations this leads to the non-termination of a computation and therefore to the non-discovery of a violation while in the third interpretation we have as in the case with complexity classes no test value and hence no test.

Furthermore we can formulate the following properties of the "three valued" predicate Hon(i,x,y,z) (where Hon(i,x,y,z) denotes ϕ_i does respect the honesty condition with value z at <x,y>)

a) If z' > z then Hon(i,x,y,z) holds implies that Hon(i,x,y,z') holds also.

b) If Hon(i,x,y,z) does not apply then for no value of z' Hon(i,x,y,z') will apply.

c) Hon(i,x,y,z) is violated then there exist a z' > z such that Hon(i,x,y,z') is true.

d) The quadruples <i,x,y,z> for which Hon(i,x,y,z) holds form a recursive set.

e) The quadruples <i,x,y,z> for which Hon(i,x,y,z) is false form a recursive enumerable set.

The properties a) ... e) represent in fact everything we need about the honesty relation in order to prove an operator gap theorem. They are axiomatized in the notion of an acceptance relation.

It should be clear that in the case of a honestly class a number of problems arise which are not present in the case of a complexity class. We conclude however this introduction by presenting a rather twisted interpretation which shows that the concept of an acceptance relation (as suggested above) is applicable in the case of a complexity class as well.

Let Cpl be a "three valued"-predicate defined by

$$\underline{\text{Cpl}}(i,x,z) = \begin{cases} \underline{\text{if}}\ \Phi_i(x) \leq z\ \underline{\text{then}}\ \text{true} \\ \underline{\text{if}}\ \Phi_i(x) < \infty\ \underline{\text{and}}\ \Phi_i(x) > z\ \underline{\text{then}}\ \text{false} \\ \underline{\text{if}}\ \Phi_i(x) = \infty\ \underline{\text{then}}\ \text{not applicable.} \end{cases}$$

This three valued predicate has the same properties a) ... e) as the predicate Hon above. The difference appears however in the definition of the complexity class. In order to be in the honesty class the program ϕ_i should not violate the condition Hon infinitely often which could be called a weak restriction. However in order to be in the complexity class the function ϕ_i should respect the condition Cpl almost everywhere which is a strong restriction. As we shall conclude in the sequel it is exactly this distinction between weak and strong restriction which produces a number of relevant differences between honesty- and complexity classes.

1.2. Formal definitions and examples

Definition 1.1. An acceptance relation A is a set theoretical total function with three natural ~~arguments~~ (say i, x and z) and values in the three element set {true,false,void} which satisfies the following axioms.

A1. Monotonicity: If $z < z'$ then

(A1a) $A(i,x,z) = \underline{\text{true}} \Longrightarrow A(i,x,z') = \underline{\text{true}}$

(A1b) $A(i,x,z) = \underline{\text{void}} \Longrightarrow A(i,x,z') = \underline{\text{void}}$

furthermore

(A1c) $A(i,x,z) = \underline{\text{false}} \Longrightarrow \exists_{z'}\ A(i,x,z') = \underline{\text{true}}$

A2. Computability:

(A2) The predicate $A(i,x,z) = \underline{\text{true}}$ is recursive in i, x, z.

Remarks

1) One should think the arguments i, x and z to play the role of "index", "argument", and "testvalue". We shall have however interpretations where both i and x encode more information than an index of a program or some argument.

2) For future use one is suggested to introduce a third axiom espressing the fact that $A(i,x,z) = \underline{true}$ forces some computation to terminate. In this report we do not need such an axiom.

Definition 1.2. Let A be an acceptance relation, and let t be a function (which might be partial).
The set of indices strongly A-restricted by t notated $F_S^A(t)$ is the set

$$F_S^A(t) = \{i \mid \overset{\infty}{\forall}_x\ [t(x) < \infty \rightarrow A(i,x,t(x)) = \underline{true}]\}$$

The set of indices weakly A-restricted by t notated $F_S^A(t)$ is the set

$$F_S^A(t) = \{i \mid \overset{\infty}{\forall}_x\ [t(x) < \infty \rightarrow A(i,x,t(x)) \neq \underline{false}]\}$$

Examples

1) Let Cpl (i,x,z) be the acceptance relation defined by

$$\underline{Cpl}\ (i,x,z) = \begin{cases} \underline{true} & \text{if } \Phi_i(x) \leq z \\ \underline{false} & \text{if } z < \Phi_i(x) < \infty \\ \underline{void} & \text{if } \Phi_i(x) = \infty \end{cases}$$

This relation is called the complexity relation. The definition of a complexity class becomes

$$\phi_i \in C_t \quad \underline{if} \quad i \in F_S^{\underline{Cpl}}(t).$$

The complexity class C_t is the set of functions having some index which is in the set strongly Cpl-restricted by t.

2) Let $\{\gamma_i\}$ be some measured set [*)] of functions. We define a corresponding acceptance relation Γ

$$\Gamma(i,x,z) = \begin{cases} \underline{\text{true}} & \text{if } z \geq \gamma_i(x) \\ \underline{\text{false}} & \text{if } z < \gamma_i(x) < \infty \\ \underline{\text{void}} & \text{if } \gamma_i(x) = \infty \end{cases}$$

Given some t the indices i strongly restricted by t are precisely the indices of functions γ_i which are almost everywhere bounded by t. The indices i weakly restricted by t are the indices of functions γ_i which are almost everywhere bounded by t when they converge but also are allowed to diverge (also on arguments where $t(x) < \infty$).

3) Let Hon be the acceptance relation

$$\underline{\text{Hon}}(i,x,z) = \begin{cases} \underline{\text{true}} & \text{if } \Phi_i(\pi_1(x)) \leq z \ \ \underline{\text{and}} \ \ \phi_i(\pi_1(x)) = \pi_2(x) \\ \underline{\text{false}} & \text{if } \Phi_i(\pi_1(x)) > z \ \ \underline{\text{and}} \ \ \phi_i(\pi_1(x)) = \pi_2(x) \\ \underline{\text{void}} & \text{otherwise} \end{cases}$$

The honesty class H_R is the set of functions having some index which is weakly Hon-restricted by R.

4) Let $E = \{B_j\}_j$ be a recursive enumerable class of sets of exceptional points. (cf. [1]). This means that all sets B_j are recursive, $\{B_j\}_j$ is closed under finite unions and contains all finite sets, and finally $\mathbb{N} \notin \{B_j\}_j$.

---

*) i.e.: A sequence of functions such that the relation $\gamma_i(x) = y$ is decidable.

Let Cplex be the acceptance relation

$$\underline{\text{Cplex}}(i,x,z) = \begin{cases} \underline{\text{true}} \text{ if } & x \in B_{\pi_2(i)} \ \underline{\text{or}} \ \Phi_{\pi_1(i)}(\overset{\circ}{x}) \leq z \\ \underline{\text{false}} \text{ if } & x \notin B_{\pi_2(i)} \ \underline{\text{and}} \ z < \Phi_{\pi_1(i)} < \infty \\ \underline{\text{void}} \text{ if } & x \notin B_{\pi_2(i)} \ \underline{\text{and}} \ \Phi_{\pi_1(i)} = \infty. \end{cases}$$

The complexity class C(t) modulo the class of sets of exceptional points E is the set of functions f for which there exist a program index j and a set index k such that the pair <j,k> is strongly Cplex-restricted by t.

5) Let Scpl be the acceptance relation

$$\underline{\text{Scpl}}(i,x,z) = \begin{cases} \underline{\text{true}} \text{ if } & \sum_{y \leq x} \Phi_i(y) \leq z \\ \underline{\text{false}} \text{ if } & z < \sum_{y \leq x} \Phi_i(y) < \infty \\ \underline{\text{void}} & \text{otherwise} \end{cases}$$

The summed complexity class SC(t) is the set of all functions having an index which is strongly Scpl-restricted by t.

Note that whenever the domain of t is infinite, all functions within SC(t) are total which is certainly not the case for normal complexity classes.

It has to be remarked that the set of indices weakly-Cpl restricted by t is not a natural concept. This represent the class of functions which are computable within t(x) steps or are divergent. The only interpretation one might give of such a resource bound class is the interpretation from the point of view of the director of a company which only pays for work which is completed. An employee which has been instructed to perform an infinite job is going to be very cheap.

The following lemma gives some trivial consequences of the axioms.

Lemma 1.3. Suppose $z < z'$ then we have:

(1.3.a) $A(i,x,z') = \underline{\text{false}} \Longrightarrow A(i,x,z) = \underline{\text{false}}$

(1.3.b) $A(i,x,z') = \underline{\text{void}} \Longrightarrow A(i,x,z) = \underline{\text{void}}$

(1.3.c) $A(i,x,z) = \underline{\text{void}} \Longrightarrow \forall_w\, A(i,x,w) = \underline{\text{void}}$

(1.3.d) $A(i,x,z) = \underline{\text{false}}$ and $A(i,x,z') = \underline{\text{true}} \Longrightarrow z < z'$

(1.3.e) $\neg(A(i,x,z) = \underline{\text{false}}$ and $A(i,x,z') = \underline{\text{void}})$

$\neg(A(i,x,z) = \underline{\text{true}}$ and $A(i,x,z') = \underline{\text{void}})$

(1.3.f) $A(i,x,0) = \underline{\text{void}}$ or $\mu z[A(i,x,z) = \underline{\text{true}}] < \infty$

(1.3.g) the predicate $A(i,x,z) = \underline{\text{false}}$ is recursive enumerable

Proof.

(1.3.a) By (A1a) $A(i,x,z) \neq \underline{\text{true}}$ and by (A1b) $A(i,x,z) \neq$ void
Now (1.3.a) follows by the fact that A is total.

(1.3.b) By (A1a) $A(i,x,z) \neq \underline{\text{true}}$. Suppose $A(i,x,z) = \underline{\text{false}}$. Then there exists a w such that $A(i,x,w) =$ true.
Now $A(i,x, \underline{\max}(w,z'))$ is $\underline{\text{true}}$ by (A1a) and void by (A1b) which gives a contradiction.

(1.3.c) By (1.3.b) and (A1b).

(1.3.d) By (A1a) we cannot have $z \geq z'$.

(1.3.e) Directly from (1.3.c).

(1.3.f) If $A(i,x,0) \neq \underline{\text{void}}$ then $A(i,x,0) = \underline{\text{true}}$ in which case the μ-operator yields 0 or $A(i,x,0) = \underline{\text{false}}$ in which case convergence is guaranteed by (A1c).

(1.3.g) By (Aic) and (1.3.c) we have
$A(i,x,z) = \underline{\text{false}}$ iff $A(i,x,z) \neq \underline{\text{true}}$ and $\exists_w A(i,x,w) = \underline{\text{true}}$
The right side clearly is recursive enumerable.

<u>Remark</u>. In general the predicate $A(i,x,z) = \underline{\text{void}}$ will be a Π_2-predicate. If it happens to be also $\sum_1$ (rec. enumerable) then the whole A becomes a recursive function.

1.3. <u>Equivalence of acceptance relations and measured sets</u>

We have seen in example 2) that the notion of a measured set can be expressed by an acceptance relation. We can also attach to an acceptance relation a measured set which contains exactly the same information.

<u>Proposition 1.4</u>. There exist a 1.1 correspondence between measured sets and acceptance relations.

<u>Proof</u>. To the measured set $\{\gamma_i\}$ we let correspond the acceptance relation Γ as given in example 2:

$$\Gamma(i,x,z) = \begin{cases} \underline{\text{true}} \text{ if } & \gamma_i(x) \leq z \\ \underline{\text{false}} \text{ if } z < & \gamma_i(x) < \infty \\ \underline{\text{void}} \text{ if } & \gamma_i(x) = \infty \end{cases}$$

Now let A be an acceptance relation. We define the functions

$$\alpha_i(x) = \mu z[A(i,x,z) = \underline{\text{true}}].$$

This is a measured set. In order to see whether $\alpha_i(x) = z$ we test whether $A(i,x,z) = \underline{\text{true}}$ <u>and</u> $(A(i,x,z-1) \neq \underline{\text{true}}$ <u>or</u> $z = 0)$.

It is clear that by going back-and-forewards in this correspondence we get back our old acceptance relation respectively our old measured set.

It is possible to give a more formal presentation of <u>prop. 1.4</u>. An index for a measured set $\{\gamma_i\}$ can be thought of to be an index j for the program ϕ_j^3 such that ϕ_j^3 is a total 0-1 function which is zero at arguments (i,x,z) for which $\gamma_i(x) = z$. Also an index for an acceptance relation A can be thought of to be an index of the characteristic function of the predicate $A(i,x,z) = $ true (as all other information can be derived from this predicate).

Now let M be the collection of indices of measured sets and let N be the collection of indices of acceptance relations then M and N are recursively equivalent.

Prop. 1.4*. $M \equiv N$

Proof.

Let $g(k)$ be an index of the program

```
$\phi^3_{g(k)}(i,x,z)$ = if $\phi^3_k(i,x,z) = 0$ and $(z=0$ or $\phi^3_k(i,x,z-1)=1)$ then 0
                          elsf $\phi^3_k(i,x,z) = 1$ and $z>0$ and $\phi^3_k(i,x,z-1)=0$ then 0
                          elsf $\phi^3_k(i,x,z) > 1$ then 2
                          else 1
                          fi
```

Now $\phi^3_{g(k)}$ is partial whenever ϕ^3_k is and $\phi^3_{g(k)}$ will fail to be a characteristic function if ϕ_k does.

If ϕ^3_k is a characteristic function then $\phi^3_{g(k)}$ will be the characteristic function of all those triples (i,x,z) for which $\phi_k(i,x,z) = 0$ and $z = 0$ or $\phi_k(i,x,z) \neq \phi_k(i,x,z-1)$.

For fixed i and x $\phi_{g(k)}(i,x,z)$ will be zero at all those z where the value of $\phi_k(i,x,z)$ changes. If $\phi_{g(k)}(i,x,z) = 0$ should hold for at most one z then there should be at most one change of value. If this is however a change of 0 to 1 the $\phi_{g(k)}(i,x,z)$ will be zero also for $z = 0$.

This shows $g(k)$ is index for a measured set iff k is index for an acceptance relation.

Hence we have $N \leq_m M$ by g.

Conversely let $h(k)$ be an index for the program

```
$\phi^3_{h(k)}(i,x,z)$ = if $\exists y \leq z \quad \phi^3_k(i,x,y) = \infty$ then undefined
             elsf $\exists y \leq z \quad \phi^3_k(i,x,y) > 1$ then 2
             elsf $\exists y_1, y_2 \leq z \quad \phi^3_k(i,x,y_1) = 0$ and $\phi^3_k(i,x,y_2) = 0$
                          and $y_1 \neq y_2$ then 1
             elsf $\exists y \leq z \quad \phi^3_k(i,x,y) = 0$ then 0
             else 1
             fi
```

Again $\phi^3_{h(k)}$ is partial whenever ϕ^3_k is. Also $\phi^3_{h(k)}$ is a characteristic function iff ϕ^3_k is. Further if so then $\phi^3_{h(k)}$ is the characteristic function of the set (i,x,z) for which there is exactly one $y \leq z$ such $\phi^3_k(i,x,y) = 0$.

From this one has $M \leq_m N$ by h.

By the normal padding technique *) one concludes

$$M \leq_1 N \quad \text{and} \quad N \leq_1 M \quad \text{whence } M \equiv N.$$

This proposition shows that the theory of acceptance relations is equivalent to the theory of measured sets. The possibility of introducing a third axiom in the future prevents the author from rejecting the notion of an acceptance relation as being useless.

*) A technique to make h and g 1-1. Cf. Rogers [9] or McCreight [7].

2. The gap theorems

2.1. Informal discussion

In this section we prove the following generalisation of the gap theorems of Borodin [3] and Constable [4]. Let A be an acceptance relation and let G be a recursive total function in two variables and let Λ be a total effective operator. Furthermore we suppose $G(x,y) \geq y$ and $\Gamma(t)(x) \geq t(x)$. Then there are arbitrary large total functions t such that the class of indices strongly (weakly) A-restricted by t is equal to the class of indices strongly (weakly) A-restricted by $\lambda_x G(x,t(x))$ respectively $\Gamma(t)$.

This statement contains four gap-theorems, three of which are proved by a straightforward translation of the known proofs for the complexity class case. Only the proof of the operator-gap theorem for weakly-A-restricted classes of functions presents difficulties which are overcome by computing one stage in advance in the construction as described by P. Young [11].

The result is based on a construction of a total function t for which the following holds: whenever for infinitely many x we have $A(i,x,t(x))$ = false and $A(i,x,\Gamma(t)(x))$ = true then we have for infinitely many x $A(i,x,\Gamma(t)(x))$ = false. This is a stronger condition than the condition which results from the construction in the proof of P. Young where we have:

$$\overset{\infty}{\exists}_x\, A(i,x,t(x)) \neq \underline{\text{true}} \Longrightarrow \overset{\infty}{\exists}_x\, A(i,x,\Gamma(t)(x)) \neq \underline{\text{true}}.$$

To understand the difference we briefly describe Young's construction cf. [11].
Let t be defined up to a certain point (say y_0) at the beginning of stage x. Now perform the following computations:

1) Generate $x+1$ programs for functions t_j extending $t|[0,y_0]$ such that $t_{j+1} > \Gamma(t_j)$ on $[y_0+1,\infty)$.

2) Generate x+1 integers $z_j > y_0$ such that for $v \in [y_0, z_{j+1}]$. $\Gamma(t_j)(v)$ can be computed from the values of t_j on $[0, z_j]$. (The z_j are computed for $j = x, x-1, \ldots, 0$).

3) For each $i < x, 0 \leq j \leq x$ test whether there exist a point $z \in [y_0+1, z_j]$ such that $A(i,z,t_j(z)) \neq \underline{true}$. If so we say that i <u>transgresses the extension t_j</u>.

4) If i transgresses the extension t_j and does not transgress the extension t_{j+1} we declare the gap section $<t_j, t_{j+1}>$ unsafe for i.

   Note that each index i has at most one gap-section which is declared unsafe for it. As we have x gapsections and x-1 functions considered we safely may execute 5)

5) Select a gap section t_j which is not declared unsafe for any index. Extend t by t_j and put $y_0 = z_j$ and proceed to the next stage.

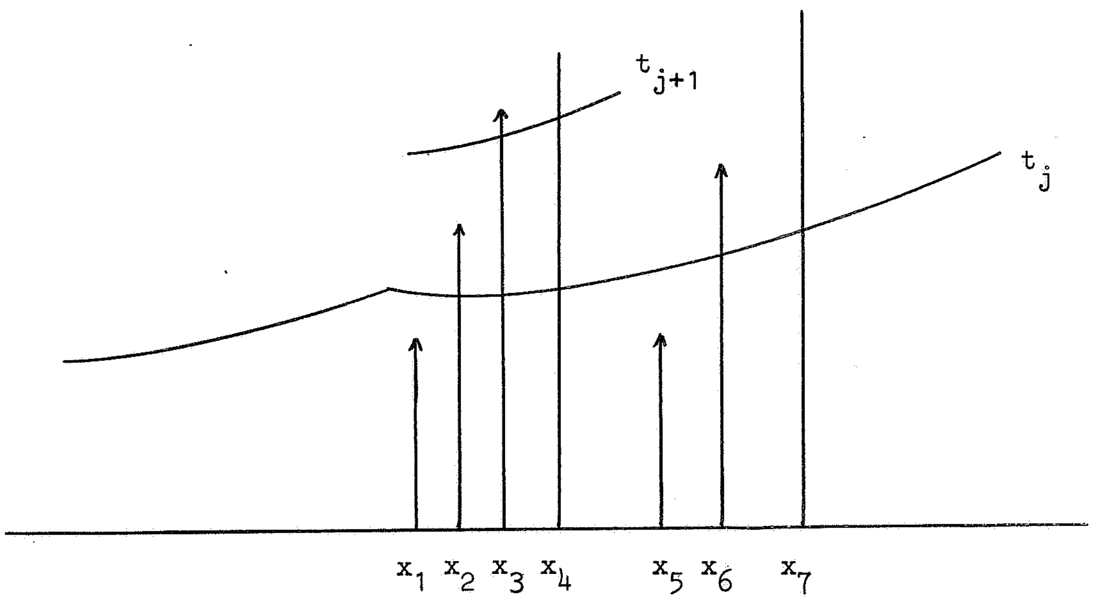
One easily verifies that the function t constructed in this way satisfies the condition

$$\overset{\infty}{\exists}_x\ A(i,x,t(x)) \neq \underline{true} \implies \overset{\infty}{\exists}_x\ A(i,x,\Gamma(t)(x)) \neq \underline{true}$$

and the operator gap theorem for strong-restriction is a straightforward result.

In weak restriction the problem is that we must accept the case $A(i,x,t(x)) = \underline{void}$ and reject the case $A(i,x,t(x)) = \underline{false}$. The only mechanism which allows v_j to discriminate between those cases is the finding of a value $z > t(x)$ such that $A(i,x,z) = \underline{true}$.

The following diagram shows the behaviour of a single index i at different points of an open local gap section t_j, t_{j+1}. The vertical lines represent the set of points (x,y) where $A(i,x,y) \neq \underline{true}$. Note that an unbounded vertical line represents points (x,y) with $A(i,x,y) = \underline{void}$ and that a bounded line represents points with $A(i,x,y) = \underline{false}$.



At x_1 both $A(i,y,t_j(x))$ and $A(i,x,t_{j+1}(x)$ are <u>true</u>, no problem.

At x_2 $A(i,x,t_j(x)) \neq$ <u>true</u> and $A(i,x,t_{j+1}(x)) =$ <u>true</u>, hence $A(i,x,t_j(x)) =$ <u>false</u> and we have that i spoils the gap section $<t_j,t_{j+1}>$ at x_2.

At x_3,x_4 $A(i,x,t_j(x) \neq$ <u>true</u> and $A(i,x,t_{j+1}(x)) \neq$ <u>true</u> (at x_3 (x_4) both are <u>false</u> (<u>void</u>)). The gap is unspoiled. However the behaviour at x_3 makes the gap section $<t_j,t_{j+1}>$ safe again for i and the behaviour at x does not so. Note that it is not possible with the information about t_j and t_{j+1} to discriminate whether we have behaviour as in point x_3 or x_4.

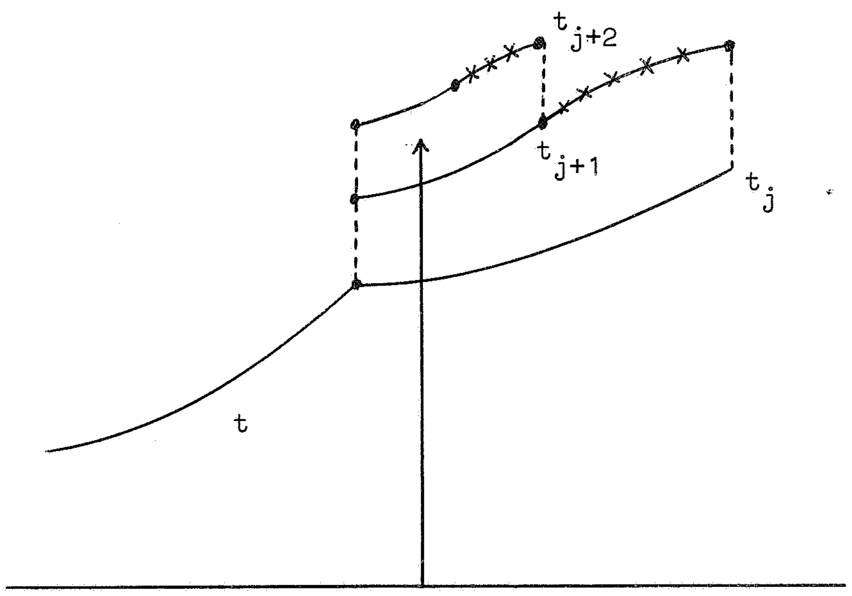
At x_5 $A(i,x,t_j(x)) =$ <u>true</u> $t_{j+1}(x)$ is undefined. No problem.

At x_6 and x_7 $A(i,x,t_j(x)) \neq$ <u>true</u> $t_{j+1}(x)$ is undefined. Now $A(i,x_t,t_j(x_6)) =$ <u>false</u> and $A(i,x_7,t_j(x_7)) =$ <u>void</u>. As no upper bound for $\Gamma(t)(x)$ is given for $x = x_6$ and $x = x_7$ it might happen that i will spoil the gap at x_6 after all, but at x_7 this will never happen.

Again we cannot discriminate whether we have a situation like in point x_6 or in point x_7.

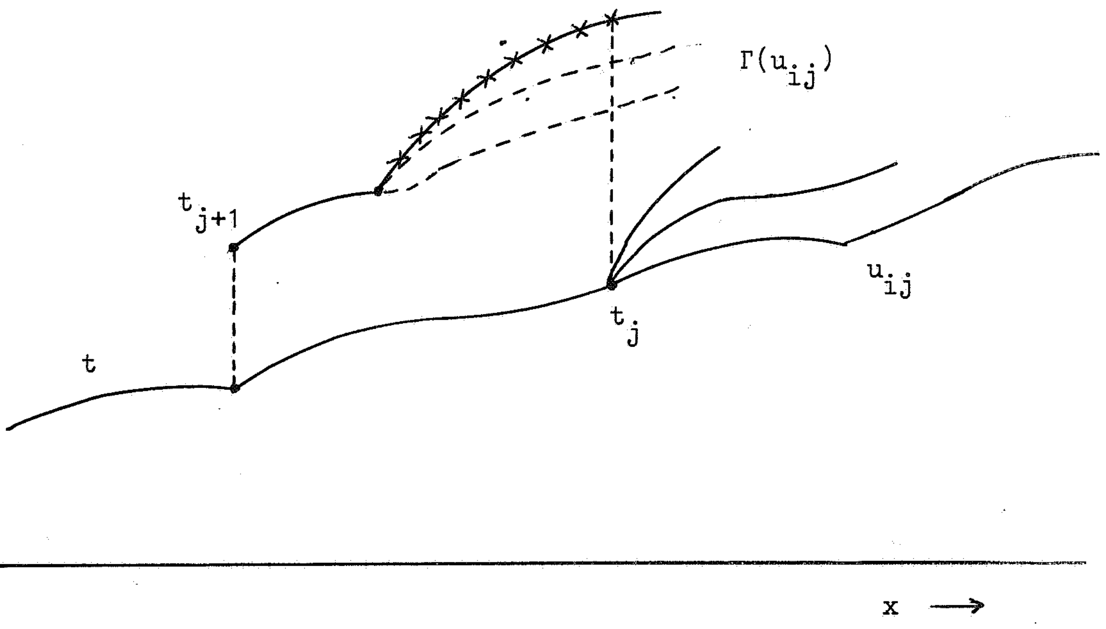
Suppose now that we are given a trick to extend t_{j+1} over the domain of t_j. This way we may forget about the behaviour like in the points x_5, x_6 and x_7. Now the only remaining case where a gapsection is spoiled is in the situation as shown by x_2. But this situation is decidable. Thus it becomes possible to decide whether a closed gap section is spoiled by some index or not.

The other advantage is the following. Suppose that we have detected that i spoils the gap section $<t_{j+1},t_{j+2}>$. Then we have decided at the same moment that for a certain point x with respect to the gap section $<t_j,t_{j+1}>$ the situation is like in x_3 and not like in x_4. (see the diagram below). Hence an index i spoiling a gap section $<t_j,t_{j+1}>$ will be safe for all lower gap sections. This property makes it again to define a safety condition in such a manner that any index will be declared unsafe for at most one gap section. After this the proof goes on as before.



To generate an upperbound for $\Gamma(t)$ over the domain of t_j we use the following observation: The only reason that t_{j+1} is not yet extended over the domain of t_j is that we have not fixed the values of t outside the domain of t_j (which makes it impossible to evaluate $\Gamma(t)$ for all the points where we need its value).

By selecting t_j we restrict ourselves however in the possible extensions of t. For t shall be one of the x+2, extensions u_{ij} of t_j generated during stage x+1. We may assume that all extensions u_{ij} shall be defined over a domain sufficiently large to define $\Gamma(u_{ij})(z)$ for all z in the domain of t_j. So in order to generate an upperbound for $\Gamma(t)$ on the domain of t_j based on the assumption that t_j is selected at <u>stage</u> x we just have to compute <u>one stage in advance</u> the x+2 extensions u_{ij} and to compute the $\Gamma(u_{ij})(z)$ for the z in the domain of t_j. This way we produce closed local gap sections and we are safe. (See the diagram below).



The above informal discussion contains all essential ideas behind the operator-gap theorem for weakly-restricted classes of indices. We present below an informal description of the program together with a proof that it yields the operator-gap we intended to prove. (In §3 the reader will find a formal definition of the program which looks like an ALGOL 68 program (although it is in fact not one) which might answer all questions raised by the vagueness of the informal description.)

2.2. Proof of the theorems

Lemma 2.1. Let A be an acceptance relation and let Γ be a total effective operator such that $\Gamma(t) > t$. Finally let a be a total recursive function. Then there exist a total function $t \geq a$ such that for all i we have

$$\overset{\infty}{\exists}_x\ [A(i,x,t(x)) \neq \underline{true} \wedge A(i,x,\Gamma(t)(x)) = \underline{true}]$$

implies

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{false}].$$

Proof.

At the beginning of stage k we suppose that t(x) is already defined over the interval $[0,y_0]$. Further we have k+1 programs of functions $t_0,\ldots,t_k$ which satisfy:

i) $t_i(x) = t(x)$ for $x \leq y_0$.

ii) $t_i(x) \geq \Gamma(t_i)(x)$ for $x > y_0$, $0 \leq i \leq k$.

iii) $t_i(x) \geq a(x)$ for all x, $0 \leq i \leq k$.

iv) $t_i(x)$ is (monotoneous) non-decreasing in x.

Further we have k+1 pointers z_i $(0 \leq i \leq k)$ which satisfy:

v) $z_k \geq y_0+1$; $z_j \geq z_{j+1}$ for $0 \leq j < k$.

vi) the support of $\Gamma(t_k)$ on $[0,y_0+1]$ is contained in $[0,z_k]$

vii) for $0 \leq j < k$ the support of $\Gamma(t_j)$ on $[0,z_{j+1}]$ is contained in $[0,z_j]$.

Note that the above situation is equivalent to the results of the computations of 1) and 2) in the algorithm of Young.

The computations of <u>stage</u> k can be described as follows:

1) For $0 \leq j \leq k$ we construct k+2 next-stage extensions $u_{j,l}$ $l = 0,1,\ldots,k+1$, which are defined by

```
i)  $u_{j,0}(x)$ = if $x \leq z_j$ then $t_j(x)$
                                  else max $\{a(x),t_j(x-1)\}$.
    and for $l > 0$.
ii) $u_{j,l}(x)$ = if $x \leq z_j$ then $t_j(x)$
                              else max $\{a(x),u_{j,l}(x-1),u_{j,l-1}(x),\Gamma(u_{j,l-1})(x)\}$.
```

2) For $0 \leq j \leq k$, $0 \leq l \leq k+1$ we construct a pointer $v_{j,l}$ which satisfies:

iii) The support of $\Gamma(u_{j,k+1})$ on $[0,z_j+1]$ is contained within $[0,v_{j,k+1}]$.

iv) For $0 \leq l \leq k$ the support of $\Gamma(u_{j,l})$ on $[0,v_{j,l+1}]$ is contained within $[0,v_{j,l}]$.

<u>Comment</u>. Note that these computations again are the equivalent of 1) and 2) in the algorithm of Young;

3) For $j = 0,\ldots,k$ we compute a function segment g_j defined on the interval $[y_0+1,z_j]$ which is defined by

```
$g_j(z)$ = if $z \notin [y_0+1, z_j]$ then skip
           else if $z \leq z_{j+1}$ then $t_{j+1}(z)$
           else max $\{\Gamma(u_{j,l})(z) \mid l = 0,1,\ldots,k+1\}$ fi fi;
```

Comment. Note that by 2) the support of $\Gamma(u_{j,l})$ on $[y_0+1, z_j]$ is contained in $[0, v_{j,l}]$. Further by assumption $\Gamma(t_j)(z) = \Gamma(u_{j,l})(z) \leq t_{j+1}(z)$ for $z \leq z_{j+1}$;

4) For $j = 0,\ldots,k$ and $i = 0,\ldots,k-1$ we check whether it occurs that for some $z \in [y_0+1, z_j]$ we have $A(i,z,t_j(z)) \neq$ true and $A(i,z,g_j(z)) =$ true. If so we say that index i enters the j-th local gap section.

5) For $i = 0,\ldots,k-1$ select the largest j such that index i enters the j-th local gap section. If such a j exist (say j_i) we declare the j_i-th local gap section unsafe for i.

6) Select some (the lowest) j such that the j-th local gap section is not declared unsafe for any i, $0 \leq i \leq k-1$. Such a j^* exists as we have $k+1$ local gap sections and only k indices i each having at most one local gap section declared unsafe for i.

7) Extend t by the values of t_{j^*} on $[y_0+1, z_{j^*}]$; put y_0 equal to z_{j^*}; put t_j equal to $u_{j^*,j}$ for $j = 0,\ldots,k+1$; put z_j equal to $v_{j^*,j}$ for $j = 0,\ldots,k+1$;

8) Proceed to stage $k+1$.

Note that after 7) the assumptions made about the situation before the entering of stage k now are satisfied before entering stage $k+1$.

We still have to explain how the program is initiated. This is performed by having step 1) and 2) executed once with $k = 0$ working on the empty function. This yields two programs t_0 and $t_1 \geq a$ and two pointers z_0 and z_1 such that t_0 and t_1 are monotoneous non-decreasing $t_1 \geq \Gamma(t_0)$, the support of $\Gamma(t_1)(0)$ is contained within $[0,z_1]$ and the support of $\Gamma(t_0)$ on $[0,z_1]$ is contained within $[0,z_0]$. Put $y_0 = -1$. These functions and pointers satisfy the assumptions assumed to hold before entering <u>stage</u> 1.

This completes the informal description of the program. It remains to show that the constructed function t has indeed the property that for all i

$$\overset{\infty}{\exists}_x\ [A(i,x,t(x)) \neq \underline{\text{true}}\ \underline{\text{and}}\ A(i,x,\Gamma(t)(x)) = \underline{\text{true}}]$$

implies

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{\text{false}}].$$

First it should be clear from the description that at any stage after a finite computation t is extended by a finite segment. Hence we can restrict ourselves to the infinitely many values x for which t(x) is defined during stages k with $k > i$.

Now let t(x) be defined at stage k and let $A(i,x,t(x)) \neq \underline{\text{true}}$ and $A(i,x,\Gamma(t)(x)) = \underline{\text{true}}$.

By abuse of notations we give y_0, z_i, t_i, u_{ij}, v_{ij} and g_i the meaning they had during execution of <u>stage</u> k. Hence we know that on $[0,z_{j^*}]$ is equal to $t_{j^*}(x)$ for a j^* such that the j^*-th local gap section was not declared unsafe for index i.

From this fact we derive that either the index i did not enter the j^*-th local gap section or otherwise i did enter the $\tilde{j}$-th local gap section for some $\tilde{j} > j^*$.

The first possibility yields:

$$\forall_{z\in[y_0+1,z_{j^*}]}\ [\neg(A(i,z,t_{j^*}(z)) \neq \underline{\text{true}}\ \underline{\text{and}}\ A(i,z,g_{j^*}(z)) = \underline{\text{true}})].$$

This is however not possible: For some $l^* \leq k+1$ we have

$$t \mid [0,v_{j^*,1^*}] = u_{j^*,1^*} \mid [0,v_{j^*,1^*}].$$

Furthermore we have that the support of $\Gamma(u_{j^*,1^*})$ on $[0,z_{j^*}]$ tained within $[0,v_{j^*,1^*}]$ hence

$$\Gamma(t)(x) = \Gamma(v_{j^*,1^*})(x) \leq g_{j^*}(x).$$

So $A(i,x,t_{j^*}(x)) \neq \underline{\text{true}}$ $\underline{\text{and}}$ $A(i,x,g_{j^*}(x)) = \underline{\text{true}}$. This is a contradiction.

We have therefore the case that the index i enters also the j-th local gap section for some $\tilde{j} > j^*$. This means that for some $\tilde{z} \in [y_0+1,z_{\tilde{j}}]$ we have

$$A(i,\tilde{z},t_{\tilde{j}}(\tilde{z})) = \underline{\text{false}}.$$

As however $\Gamma(t)(\tilde{z}) = \Gamma(t_{j^*})(\tilde{z}) \leq t_{j^*+1}(\tilde{z}) \leq t_{\tilde{j}}(\tilde{z})$.

We conclude $A(i,\tilde{z},\Gamma(t)(\tilde{z})) = \underline{\text{false}}$ for some $\tilde{z} > y_0$.

Our conclusion is that for each stage k where values t(x) are defined in such a way that $A(i,x,t(x)) \neq \underline{\text{true}}$ and $A(i,x,\Gamma(t)(x)) = \underline{\text{true}}$ there are also values $t(\tilde{z})$ defined for which $A(i,\tilde{z},\Gamma(t)(\tilde{z})) = \underline{\text{false}}$.

Again using the fact that only finitely many values of t are defined at a single stage we conclude that

$$\overset{\infty}{\exists}_x\, [A(i,x,t(x)) \neq \underline{\text{true}}\ \underline{\text{and}}\ A(i,x,\Gamma(t)(x)) = \underline{\text{true}}]$$

implies

$$\overset{\infty}{\exists}_x\, [A(i,x,\Gamma(t)(x)) = \underline{\text{false}}]$$

which completes the proof.

It is not hard to prove from the lemma both operator-gap theorems.

Theorem 2.2. [Operator-gap for weak restriction]: Let A be an acceptance relation, Γ a total effective operator with $\Gamma(t) \geq t$ and a a total function. Then there exist a total function $t \geq a$ such that $F_W^A(t) = F_W^A(\Gamma(t))$.

Proof.

Let t be constructed as in the lemma. As $\Gamma(t) \geq t$ the inclusion $F_W^A(t) \subseteq F_W^A(\ (t))$ is trivial. Now suppose $i \notin F_W^A(t)$ and $i \in F_W^A(\Gamma(t))$. Then we have

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{\text{true}}\ \underline{\text{and}}\ A(i,x,t(x)) = \underline{\text{false}}]$$

but now by the lemma we have

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{\text{false}}]$$

so $i \notin F_W^A(t)$. Contradiction.

Theorem 2.3. [Operator-gap for strong restriction]: Let A be an acceptance relation, Γ a total effective operator with $\Gamma(t) \geq t$ and a a total function. Then there exist a total function $t \geq a$ such that $F_S^A(t) = F_S^A(\Gamma(t))$.

Proof.

First we have to remark that both Constable and Young have given a proof which yields the above theorem after a suitable translation. It is also an easy consequence of our lemma 2.1.

Let t be constructed as in the lemma. Again the inclusion $F_S^A(t) \subseteq F_S^A(\Gamma(t))$ is trivial. Next suppose

$$i \notin F_S^A(t) \qquad \text{i.e.} \qquad \overset{\infty}{\exists}_x\ A(i,x,t(x)) \neq \underline{\text{true}}.$$

If there are infinitely many x such that $A(i,x,t(x)) = \underline{\text{void}}$ we have $A(i,x,\Gamma(t)(x))$ void for the same x and we conclude $i \notin F_S^A(\Gamma(t))$. So we may assume that $A(i,x,t(x)) = \underline{\text{false}}$ for infinitely many x. Now suppose that $A(i,x,\Gamma(t)(x)) = \underline{\text{false}}$ for only finitely many x. Then we have again

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{\text{true}}\ \underline{\text{and}}\ A(i,x,t(x)) \neq \underline{\text{true}}]$$

which by Lemma 2.1 yields

$$\overset{\infty}{\exists}_x\ [A(i,x,\Gamma(t)(x)) = \underline{\text{false}}].$$

A contradiction.

The composition-gap theorems could be considered to be special cases of the more powerful operator gap theorems. It has however to be remarked that the old proof of Boredin [3] is correct for both the strong and the weak case. Also the constructive versions of this proof remain true for both cases [5,11]. This results from the fact that in the composition gap construction a "local gap section" is defined over a domain consisting of a single point and is therefore automatically closed.

For completeness sake we give a proof:

<u>Theorem 2.4</u>. [Composition gap theorem]: Let A be an acceptance relation, let G be a total function satisfying $G(x,y) \geq y$ and let a be a total recursive function. Then there exist a function $t \geq a$ such that both $F^A_W(t) = F^A_W(\lambda_x G(x,t(x))$ and

$$F^A_S(t) = F^A_S(\lambda_x G(x,t(x))).$$

<u>Proof</u>

Define $t(x,k) = \underline{\text{if}}\ k=0\ \underline{\text{then}}\ a(x)\ \underline{\text{else}}\ G(x,t(x,k-1))+1$.
Next define

$$t(x) = t(x,\mu k[\underset{i<x}{\forall}[A(i,x,t(x,k)) = \underline{\text{true}}\ \underline{\text{or}}\ A(i,x,t(x,k+1)) \neq \underline{\text{true}}]])$$

Note that the values $\{t(x,k)\}_k$ form an increasing sequence for any fixed x. Furthermore for every i there is at most one k such that $A(i,x,t(x,k) \neq$ true and $A(i,x,t(x,k+1)) =$ true. Hence the μ-operator applied on this predicate yields a value $k \leq x$. Now we can prove about this function t(x) the strong assertion:

for all $x \geq i+1$ we have $A(i,x,t(x)) = A(i,x,G(x,t(x)))$.

The theorem follows straightforwards.

As a corollary we have gap and operator gap theorems for all the acceptance relations treated in the examples in §1. Taking in particular the acceptance relation Cpl we have the old theorems back. Taking the acceptance relation Hon yields the gap theorems for honesty classes.

3. Formal description of algorithm

Readers unfamiliar with the programming language ALGOL 68 are advised to skip this section.

Consider the following ALGOL 68 particular program.

```
begin proc twist = (proc (int,int) int p) proc (int,int) int:
                        ((int x,y) int: p(y,x)) ;
    proc (int,int) int sum = (int x,y) int: (x+y) ;
    proc (int,int) int mus; mus:= twist (sum) ;
    print (mus(2,2))
end
```

At first glance it seems that the programmer wants to verify in a stupid way that 2+2 = 4. In fact the program can be shown to yield an undefined eleboration. The undefinedness results from the fact that by the scope conditions inherent in ALGOL 68 it is nearly impossible to write a routine which has some formal parameter and which yields as value a routine which is dependent of the value of the parameter. In the above example the identifier twist possesses as value the routine:

```
(proc(int,int) int p=~; proc(int,int) int: ((int x, int y) int: p(y,x)))
```

and consequently the call twist(sum) is elaborated by elaborating the closed clause.

```
(proc(int,int) int p=sum; proc(int,int) int: ((int x, int y) int: p(y,x))).
```

Elaboration of this closed clause gives as value the routine possessed by the routine denotation ((int x,int y) int: p(y,x)) which is the routine

```
        (int x = ~, int y = ~; int: p(y,x)).
```

Now the scope of this routine is bounded by the scope of the identifier p (cf. [10], 2.2.4.2.b). As the scope of the name possessed by mus is the whole particular program, the assignation mus:= twist(sum) yields an elaboration which is undefined (cf. [19], 8.3.1.2.c).

The reader of the program should keep in mind that the elaboration of the program is undefined in the semantics of ALGOL 68 as defined in [10]. There are however reasons to assume that this trouble is going to be solved in some way in a future revision of the definition of the language [12, p.25].

To understand the program one should change the semantics of the procedure call in such a manner that in the above example the result of the call twist(sum) should yield the routine

```
(proc(int,int) int p=sum; int x = ~; int y = ~; int: p(y,x))
```

i.e. the formal parameters are transformed into identity declarations defining the formal parameters to be equal to the actual parameters and transferred in this way to the delivered routine.

Although the above discussion is far from complete the author hopes that it is sufficient to explain the meaning of the program for the operator-gap-construction.

The program describes an algorithm which does not terminate. During the infinite computation controlled by the simple do loop do stage(sn+:=1,dp) the computed values of t are loaded into the infinite vector computed value; Although the program is no procedure denotation we have accepted the analogy to represent the values which control the program (the operator, acceptance relation and lowerbound which has to be surpassed by the function which is to be computed) by skip-symbols. The unimaginable user should insert at this place his own values.

The program rather strictly follows the informal description given in §2.2. It is the opinion of the author that it should be possible to present formal definitions of algorithms described in recursion theory using modern high-level programming languages in order to prevent ambiguities and errors which might arise from the informal descriptions which are commonly given. The informal description should explain the algorithm but not define it.

```
begin
     mode fun = proc(int) int;
     mode operator = proc(fun) fun;
     mode acrel = proc(int,int,int) bool;

   ¢ As explained before it is hard to produce some non-trivial
     value of the mode operator in regular ALGOL 68 ¢

     fun lowerbound = ~ ;
     operator gamma = ~ ;
     acrel acceptance = ~ ;

     [0:1 flex] int locub, computed value;
     [0:1 flex] fun local extension;
     int sn, dp;

   ¢ locub, local extension, sn and dp play the role of (zj), (tj),
     k and y0 in the informal description ¢

     priority ∧:=  = 1;
     op ∧:=  = (ref bool a, bool b) ref bool: a:= a∧b;

   ¢ readers unwilling to accept ∧:=  as an indicant should consider
     this declaration to be part of the standard prelude [12.p.49] ¢

     op max = ([ ] int ar) int:
              begin int l = ⌊ar, u = ⌈ar;
                   (l > u | 0 |: l = u | ar[l]|
                          int k = ar[l], kk = max ar[l+1:u];
                          (kk < k | k | kk)
                   )
              end ;
```

```
proc extend = (int x, low, fun t, ref[0: ] fun tj):

    begin int ub = ⌈tj; (ub < x | goto incorrect);
          tj[0]:= (int i) int: if i ≤ low then t(i)
                               else max (t(i-1), lowerbound (i)+1)
                               fi;
          for j to x do
          tj[j]:= (int i) int: if i ≤ low then t(i)
                               else max (tj[j](i-1),tj[j-1](i),
                                         gamma(tj[j-1])(i))
                               fi
    end;

  ¢ The above procedure is the bad animal in this program; its
    elaboration in Regular ALGOL 68 yields undefined results whenever
    called with actual parameters ¢

proc support = (int x, fun f, operator g) int:

    begin int ub:= 0;
          fun fstar = (int i) int: ((i > ub | ub:= i); f(i));
          g(fstar)(x);
          ub
    end;

  ¢ As assumed the operator g works by issuing calls of the function
    it works on, the result being independent of the actual computation
    it invokes  but only dependent of the delivered value ¢

proc suponint = (int x, y, fun f, operator g) int:

    begin (x > y | goto incorrect);
          int out:= y+1;
          for z from x to y do
              (int p = support (z,f,g); (p > out | out:= p));
          out
    end;
```

```
proc domains = (int low, [0: ] fun tj) [  ] int:

    begin int up = ⌈tj; [0:up] int yj; int lastyj:= low + 1;
          for j from up by -1 to 0 do
              lastyj:= yj[j]:= suponint(0,lastyj,tj[j],gamma);
              yj
    end;

  ¢ domains is the procedure which computes the pointers vj,l in part 2
    of the algorithm ¢

proc entergap = (int nof, low, up, fun bottom, roof) bool:

    begin bool safe:= true
          for z from low + 1 to up while safe do
          safe ∧:= acceptance (nof,z,bottom(z)) = acceptance(nof,z,roof(z));
          safe
    end;

  ¢ entergap tests whether the index nof enters the local gap section
    determined by bottom and roof over the interval [low+1,up]. Note
    that the gap section is closed ¢

proc unsafegap = (int nof, k, low, [0: ] int up, [0: ] fun bottom, roof) int:

    begin (k > ⌈up ∨ k > ⌈bottom ∨ k > ⌈roof | goto incorrect);
          bool untouched:= true; int p:= k;
          for j from k by -1 to 0 while untouched do
          (untouched ∧:= entergap(nof,low,up[j],bottom[j],roof[j])|p-:=1);
          p
    end;

  ¢ unsafe gap seeks the highest entered gap section; if not present
    it yields value -1 ¢
```

```
proc stage = (int stagenumber, last defined):

     begin int st = stagenumber, ld = last defined;
           [0:st,0:st+1] int next stage locub;
           [0:st,0:st+1] fun next stage extension;
           [0:st] fun giant;

           for j from 0 to st do
           begin extend (st+1,locub j,local extension[j],
                          nextstage extension[j]);
                 ¢ informal description 1) ¢

                 nextstage locub[j]:= domains(locub[j],
                                             nextstage extension[j]);
                 ¢ informal description 2) ¢

                 giant[j]:= (int i) int:

                    if i ≤ ld ∨ i > locub[j] then skip
                    else int great;
                         ((j=st | goto upperunknown);
                                 (i > locub[j+1] | goto upperunknown);
                         great:= local extension[j+1](i) .
                    upperunknown: great:= local extension[j](i)+1;
                         for k from 0 to st+1 do
                         (int p = gamma(nextstage extension[j,k])(i);
                               (p > great | great:= p)
                         ) ; great
                    fi
                 ¢ informal description 3) ¢
           end;
                 [0:st] bool safegap;
                 for j from 0 to st do safegap[j]:= true;

                 for nof from 0 to st-1 do
                    (int k = unsafegap(nof,st,ld,locub,local extension,
                                          giant);
                         (k ≥ 0 | safegap[k]:= false)
                    );
                 ¢ informal description 4) and 5) ¢
```

```
              int select:= 0;
                while ¬ safegap[select] do select +:= 1;
              ¢ informal description 6) ¢

              dp:= locub select ;
              [0:dp] int xx; for j from 0 to ld do
                                   xx[j]:= computed value[j];
                             for j from ld+1 to dp do
                                   xx[j]:= local extension[select](j);
                             computed value:= xx;

              local extension:= nextstage extension[select];
              locub:= nextstage locub[select]
              ¢ informal description 7) ¢

      end; ¢ end of procedure stage; informal description 8) ¢

             sn:= 0; dp:= -1;
             extend(1,-1,(int i) int: 0, local extension);
             locub:= domains(-1,local extension);

             ¢ these instructions perform the initialisation as
               described in the informal description ¢

             do stage(sn+:=1,dp);

             ¢ the program loops forever unless halted when it
               enters one of the few illegal situations which are
               tested for ¢

             incorrect: skip

end
```

References

[1] L. Bass, P. Young. Hierarchies based on computational complexity and irregularities of class determining measured sets. Report CSD TR 58. (July 71), Purdue University.

(also presented at 2nd ACM Symp. on the theory of computing, May 1970, Northampton Mass. See also the thesis of L. Bass at Purdue University).

[2] M. Blum. A machine independent theory of the complexity of recursive functions. JACM 14 (1967), 322-336.

[3] A. Borodin. Computational complexity and the existence of complexity gaps. JACM 19 (1972), 158-174.

[4] R.L. Constable. The Operator Gap. JACM 19 (1972), 175-183.

[5] P. van Emde Boas. A note on the McCreight-Meyer naming theorem in the theory of computational complexity. Report ZW 7/71 (August 71), Math. Centre, Amsterdam.

[6] J. Hartmanis, J.E. Hopcroft: An overview of the theory of computational complexity. JACM 18 (1971), 444-475.

[7] E.M. McCreight. Classes of computable functions defined by bounds on computations. Thesis (1969), Carnegy Mellon Univ. Pittsburg. Penn.

[8] H. Rogers Jr. Theory of recursive functions and effective computability. McGraw-Hill, New York (1967).

[9] H. Rogers Jr. Gödel numbering of partial recursive functions. Jour. Symb. Logic. 23 (1958) 331-341.

[10] A. van Wijngaarden ed., B.J. Mailloux, J.E.L. Peck and C.H.A. Koster. Report on the Algorithmic Language ALGOL 68. Numerische Mathematik 14 (1969) 79-218.

[11] P. Young. Easy constructions in complexity theory: Speed up and gap theorems. Report CSD TR 57. (July 71) Purdue University.

[12] ALGOL Bulletin 33 (March 72).