Qüestiió - V. 9, n.º 1 (març 1985) pp. 7-34

THE ECOGEN LANGUAGE FOR GENETIC SIMULATION

JORDI OCAÑA
UNIVERSITAT DE BARCELONA

This paper tries to show, in a condensed form, the main features of ECOGEN, an event-schedulling simulation language oriented to the Population Genetics. It is based on the Pascal Language.

Keywords: GENETIC SIMULATION, PASCAL, EVENT-SCHEDULLING, POPULATION GENETICS, EVOLUTIONARY ECOLOGY.

1. INTRODUCTION AND MAIN CONCEPTS.

The ECOGEN Language is an extension of the Pascal Language oriented to facilitate discrete event digital simulation in Evolutionary Ecology and Population Genetics. Interested readers, unfamiliar with these topics may see /7/ for a good introductory text.

ECOGEN is simply the Pascal Language with some additional features, basically:

1) A standard, event schedulling oriented, simulation control program like in GASP-II. Although ECOGEN is in this "dynamical" aspect similar to other event-oriented simulation languages, it is perhaps worth pointing out that it may be quite distinct under a "conceptual" or "organizative" view: while in some event-schedulling oriented languages emphasis is on events (for example, there is specifically a general events list) the "main characters" of ECOGEN are entities (that in a given model can represent, for example, gametes, individuals, ecosystems or parts of the environment). As , in general, a given event will affect directly a given concrete entity (for example "death", event, of a given "individual", entity) in ECOGEN there is a strong association between events and entities. Roughly speaking, they are taken as something like "atributes" of entities, their value being the scheduled time (real value) of occurrence. Every concrete entity may be viewed as having an associated events list, the list of its own associated events, with its own more immediate event. Concrete entities can be grouped in a wide variety of lists, among them there is even an increasing-time-of-its-most-immediate-event list, managed by the control program. See /6/ for a more complete discussion of the ECOGEN "world view".

2) Some additional standard constants, types, variables, procedures and functions try to give an adequate framework to define and to handle concepts like "genotype" or "kinship".

3) An additional declarative part, labelled entity to be placed between main program type and var parts, allows the simulated kinds of entities (populations, indivuduals ...) "profile" definition, i.e. associated attributes, taking the attribute concept in a very wide sense including associated events, metric (real valued) attributes, lists of references to other concrete entities, the genotype, etc.

- Jordi Ocaña - Universitat de Barcelona - Facultat de Biologia - Dep. Bioestadística - Av. Diagonal, 637 Barcelona.

- Article rebut el març de 1985.

4) Some additional syntax (simulating clause) to specify that a given event is simulated by a given procedure. This procedure/event association is a common trait of every event-schedulling oriented simulation language.

5) A set of twelve basic entity handling sentences, associated to correspondingly additional reserved words, namely cancel, create, delete, identify, forget, choose, assign, join, separ, link, unlink, and schedule. Although it was perfectly possible to substitute them by standard (more than eleven) procedures, we ultimately favoured the first possibility, mainly for two reasons:

   a) some of them are syntactically complex; more than eleven standard procedures would be necessary, some of them with many parameters without obvious ordering and meaning.

   b) it is desirable to avoid possible user-redefinition of such "delicate" sentences.

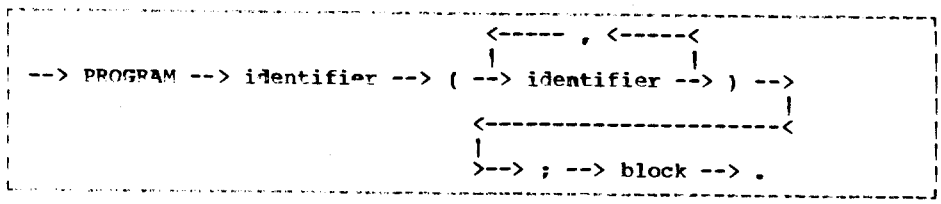
If ECOGEN were based on a more elaborated language (at least in protection and procedure calling aspects) like Ada, the second possibility might be preferred (to define a specific "package" and no additional language syntax).

Use of simulation techniques in Biology, particularly in Population Genetics, is not a recent subject. Interested readers may see /1/, /2/, /3/, for a review on the subject.

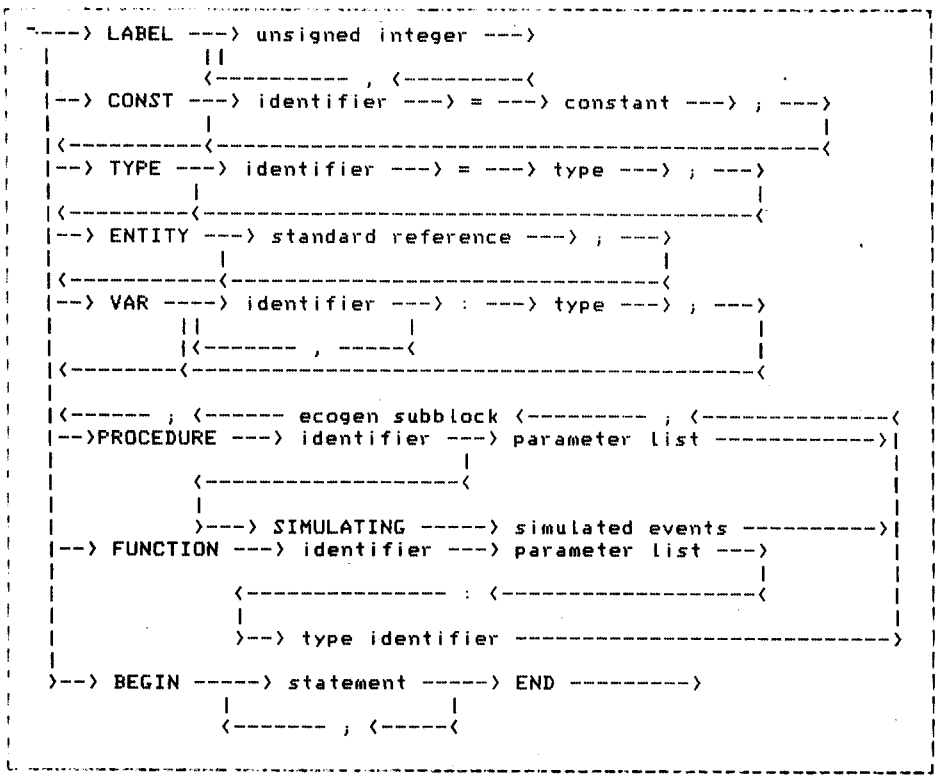
Genetic simulation is not a specially distinct kind of simulation. It is simply systems simulation devoted to a special kind of systems and problems, "genetical" in the sense that their component entities (for example "individuals") are characterized by some genetical attributes (among other) with the possibility of "transfering" their value to other entities (descendants) by processes associated to specific mechanisms like sexual reproduction. These and other features, all associated to the fact that we are simulating living systems, give some particularities to genetic simulation. It is perfectly suitable to use a general purpose or a general simulation language like SIMPSCRIPT to perform genetic simulation. The only purpose of ECOGEN is to do it more easily and more clearly, at least for Pascal programmers.

2. STRUCTURE OF AN ECOGEN PROGRAM

A program written in ECOGEN is structured in a similar way to a Pascal program. The following syntax diagram defines de general structure of an ECOGEN program. A more detailed syntatical description is found in section 5 and in the appendix A.



where "block " is defined as



All the Pascal parts are defined as is usual. As is explained in section 5, the profile - of the main elements of the model is established in the entity declarative part. This means things like to establish the attributes associated to every class of entity or the admissible classes of entities in every list.

The construct procedure <identifier> simulating <simulated events> associates a procedure with an events list. Every time one of the specified events is going to happen (it is scheduled for the most inmediate time) the control program will activate this procedure.

"ecogen sublock" is defined as Jensen & Wirth /4/ "block", that is, lacking entity part and simulating clauses.

Executive statements in main program and sublocks are all Pascal statements plus the additional ECOGEN statements.

3. SPECIAL SYMBOLS IN ECOGEN

The basic vocabulary of the ECOGEN language is the same as that of Pascal, plus the following:

1) Quotation marks " are used to delimit the constant values of the standard types -- chromatid, homologs and genotype (see -- next section) appearing in the text of a program.

2) Word-delimiters or reserved words:

as	identify
assign	join
at	kind
branch	link
by	mode
cancel	own
choose	relating
containing	schedule
create	separ
delete	simulating
entity	unlink
forget	value
from	

4. STANDARD IDENTIFIERS IN ECOGEN.

In addition to standard Pascal identifiers, the ECOGEN language provides some standard constants, types, variables, procedures and functions. Some of them are related to its

character of simulation language, the others try to express biological and genetical concepts.

4.1. STANDARD ECOGEN TYPES.

Among ECOGEN standard types, there are nine scalar types, all predefined as the integer subrange 0..15 (except the type allele).

These are

1. class: expressing all the possible entity classes. If a given class value represents an entity class that is going to really participate in the simulation, it is necessary to associate it with an adequate profile block in the entity declarative part.

2. event: expressing all the possible events changing the state of the system. Every event that really can "happen" in a model must be associated with a specific procedure by means of a simulating clause and to one or more classes of entities in its corresponding profile blocks. Sometimes there are events not associated to any entity, but ECOGEN view imposes the definition of "dummy" entities characterized by, and probably only by, these events.

3. name: at every moment during simulation there are none, one or more "representatives" or "concrete entities", really -- existing (usually assimilable to " registers" in main core), of every class of entity. Access to them is performed by means of "names". When a concrete entity is "created", "chosen" (for example at random from a group of existing concrete entities) and in general "accessed", it (its register) is associated with a name (that one specified in the corresponding accessing sentence). Until this association is overridden (for example by associating this name with another concrete entity) any action specifying this name affects its associated concrete entity. In a given moment, a concrete entity can be associated to (or designated by) one or more (or none) name values, but a given name value can designate at most only one concrete entity.

4. metric: corresponding to the "metric" attributes of entities, those attributes taking numerical real values, as "size" or "blood volume".

5. list: entities may have lists of references of other concrete entities as an attribute. This kind of relational attribute whose value will be changed adding or removing references, is useful in defining entities like "ecosystems" which (jointly with other more standard attributes as "numerical" 'mean time between rainy days' or 'total food resources') are characterized by the fact that they are "groups" of other entities (individuals, stones...).

6. relation corresponding to other possible relational attributes. Its value in a concrete entity is the pattern of other "related" or "linked" entities.

7. gene: corresponding to the basic hereditary units. They may in reality correspond to a wide variety of concepts like a classical "gene" coding for the 'white' trait, a specific DNA segment coding for a protein or only a codon.

8. allele: corresponding to the possible values of gene atributes. The standard definition of this type is

   allele=boolean;

   assuming that every gene has only two possible allelic states.

9. chromosome: gene attributes are logically linked in high level units, the linkage groups or chromosomes, usually corresponding to physically observable organules like the eukariotic chromosomes.

Although all these scalar types have their standard definitions it will frequently be convenient to redefine all or some of them, for example to improve program legibility. Namely, it would be adequate to specify -- that entity classes are

class=(larva, adult, population, environment);

or that names
name= (father, mother, son, sister, brother, partner, migrant1, migrant2);

Nothing more is altered by these possible and frequently desirable redefinitions.

There are two additional scalar standard types:

10. modelist=(decreasing, increasing, first, last, preceding, following, minimum, maximum, atrandom);

specifying modes of access to list attributes, with two standard subrange types

modejoin=decreasing..following;

specifying joining strategies, and

modechoose=first..atrandom;

specifying choosing entities strategies. See /6/ for a discussion on the meaning of these options.

11. locus=1..maxloc; where maxloc is a standard constant specifying the maximum number of genes in each chromosome or linkage unit. Its standard definition is

maxloc=16

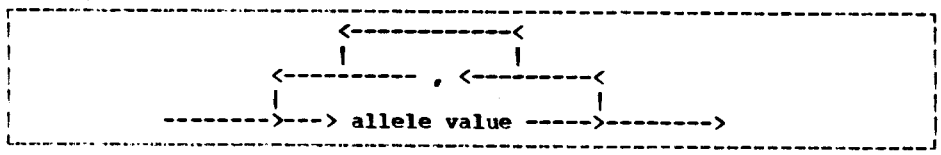
redefinable without problem. The type locus corresponds to the place or rank order of every concrete gene in every concrete chromosome (it is not necessarily fixed as it is subject to change by "structural mutations").

Finally, there are three structured standard types:

12.

```
chromatid= record
               gen: packed array[locus] of allele;
               len: 0..maxloc
           end;
```

As allele correspon to the possible values of single gene attributes, the standard type chromatid correspond to the higher level chromosome attribute values. As has been stated, chromatid constant values appearing in a program must be enclosed in double quotes. A chromatid constant value is simply a list of constant allele values separated by commas or blanks, diagramatically



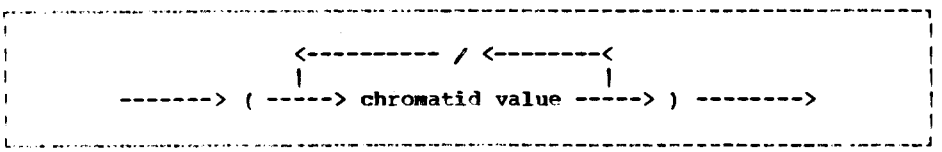
with repetition factor determining length (len) of this chromatid and order determining the locus of each allele. When the allele type is a subrange of the char type it may be represented by a string, as an admissible alternative to the above syntax.

```
homologs= record
              chrom : array[1..maxploidy] of chromatid;
              ploidy: 0..maxploidy
          end;
```

in correspondence to the fact that (depending on the ploidy number) each chromosome is represented by some "homologs" (one in an haploid, two in a diploid ...) maxploidy is a standard constant

maxploidy=2;

also redefinable. It specifies the maximum ploidy number, i.e. the maximum number of homolog chromatids for a given chromosome. An homologs constant value is a list of chromatid constant values separated by slash signs and enclosed in parenthesis (and in double quotes when appearing in a program), diagrammatically

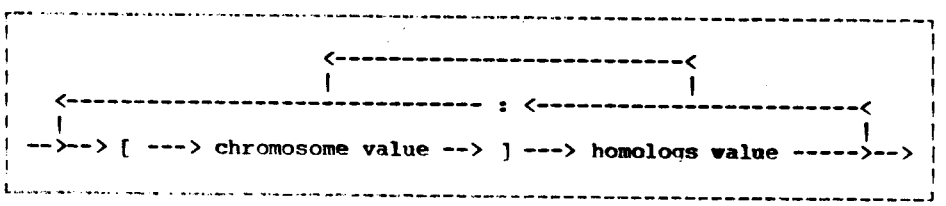


with repetition factor "/" specifying ploidy.

14.

genotype=array[chromosome] of homologs;

finally, the standard type genotype expresses the value of the individuals full pattern of homologs for all chromosomes. A genotype constant value is a list of chromosome constant values enclosed in brackets, followed by a corresponding homologs constant value and separated by colon signs or by blanks. Diagramatically



4.2. STANDARD ECOGEN VARIABLES.

They provide helpful values, taken from standard types definition and entity declarative part.

First there are firstone and lastone, defined as

```
firstone, lastone:   record
                       nam: name;
                       cla: class;
                       eve: event;
                       met: metric;
                       lis: list;
                       rel: relation;
                       chr: chromosome;
                       gen: gene;
                       all: allele
                     end;
```

giving first and last value of standard scalar ECOGEN types.

Variable proper contains a summary of entity declarative part (names associated to every entity clas,... , joining strategy to every list... It is defined as

```
proper:
     record
       cla: array[class] of record
                               nam: set of name;
                               eve: set of event;
                               met: set of metric;
                               chr: set of
                                                    chromosome;
                               lis: set of list;
                               rel: set of
                                                    relation
                             end;
       chr: array[chromosome] of record
                                   gen: set of
                                                        gene
                                 end;
       gen: array[gene] of record
                              all: set of allele
                            end;
       lis: array[list] of record
                             cla: set of class;
                            case mjoin: modejoin
                            of
                             decreasing,
                             increasing:(met:metric);
                             first,last:();
                             preceding,
                             following:(nam:name)
                            end;
       rel: array[relation] of record
                                 cla: set of
                                                    class;
                                 branches: integer
                               end;
       eve: array[event] of record
                               minim,maxim: real
                             end;
       met: array[metric] of record
                               minim,maxim: real
                             end
     end;
```

Finally, variable cod contains strings representing the values of scalar standard ECOGEN types. It is defined as

```
cod: record
nam: array[name]   of stringident;
cla: array[class]  of stringident;
eve: array[event]  of stringident;
met: array[metric] of stringident;
lis: array[list]   of stringident;
rel: array[relation] of stringident;
chr: array[chromosome] of stringident;
gen: array[gene]   of stringident;
all: array[allele] of stringident
(  ....  )end;
```

(assuming stringident=packed array[1..lident] of char; end being lident the implementation dependent maximal length of identifiers).

It would be useful, for exemple, on input/ output.

4.3. STANDARD ECOGEN PROCEDURES AND FUNCTIONS.

Boolean functions informing on the state of concrete entities

All these functions return a boolean value, true if the expressed condition is valid and false if not.

1. function exists(na: name): boolean;

   if name na is associated to a really existing concrete entity.

2. function has(na: name; ge: genotype): boolean;

   if na has genotype ge.

3. function includes(na: name, lst: list; ele: name): boolean;

   if na includes the concrete entity ele in its list attribute lst.

4. function top(na: name, lst: list; ele: name): boolean;

   if ele is the first element of the list lst in owner entity na.

5. function bottom(na: name; lst: list; ele: name): boolean;

   if it is the last.

6. function related(na1: name, rel: relation; na2: name; br1, br2: integer): boolean;

   if na1 and na2 are related by relation rel by its respective "branches" br1 and br2.

Value of entity attributes and other characteristics of entities.

1. function valmetric(na: name; me: metric): real;

   value of attribute me in concrete entity designated by na.

2. function card(na: name; lst: list):integer;

   number of elements in list lst of concrete entity na.

3. procedure valchrom(na: name; chr: chromosome; var ho: homologs);

   returns, in variable ho, all the homolog chromatids for chromosome chr in entity na.

4. procedure valown(na: name; attribute, destination);

   returns the value of a user defined attribute in variable destination of the adequate type (see section 5 for a brief discussion on user defined attributes).

5. function valclass(na: name): class;

   returns the entity class of concrete entity na.

6. function time(na: name; ev: event): real;

   returns the internal time value at which event ev is tabulated (remember that in ECOGEN language there is a strong association between events and entities).

Chromatid, homologs and genotype data handling

Given its standard definition, handling of these genetical types is very easy. But careless performing operations like inserting a new chromatid value into a homologs may be "dangerous" if programmers do not take care of things like attribute ploidy. It is preferable to use the following standard:

1. procedure joinchtd(ct: chromatid; i: integer; var ho: homologs);

   joins a new homolog chromatid ct in place i of homologs ho.

2. procedure delechtd(i: integer; var ho: homologs);

   deletes chromatid i in homologs ho.

3. procedure copychtd(ctl: chromatid; l1, l2: locus, var ct2: chromatid);

   a fragment of ctl, from locus l1 to l2 (both included), is assigned on ct2.

4. procedure concchtd(ctl,ct2: chromatid; var ct: chromatid);

   concatenation of ctl and ct2 in ct.

5. procedure codechtd(s: string; ls: integer; var ct: chromatid);

   string s of length ls is codified as a chromatid value. It is useful on input.

6. procedure codehmlg(s: string; ls: integer, var ho: homologs);

   idem for homologs.

7. procedure codegeno(s: string: ls: integer; var ge: genotype);

   idem for genotype.

8. procedure bgamete(ge: genotype; var gam: genotype);

9. procedure gamete (ge: genotype; var gam: genotype);

both simulating gametogenesis i.e. generation of a new gamete (assimilable to a genotype with ploidy=1) from genotype ge, by random segregation an possibly recombination. bgamete is a more restrictive (and more efficient) version of gamete, specially designed for the case where ploidy of ge equals 2, the standard definition of type allele is on (allele=boolean) and locus value of every gene in all the chromatid is constant - that is, there is no possibility of chromosome structural changes.

Random number generation.

These are the basic resources for random variate generation. Specific functions for some discrete and continuous distributions are described in /6/.

1. function uniform(a,b: real): real;

   generating uniformly distributed real numbers between a and b, a<b.

2. function rand(i1,i2: integer): integer;

   generating a randomly choosen integer number between i1 and i2, i1<=i2, both included.

Simulation control and output

Except tnow and control they are activated from the control program and from other standard subprograms, not directly from the user program.

1. function tnow: real;

   internal time value.

2. function tmax: real;

   maximum time for current replicate.

3. function numrepli: integer;

   current replicate number.

4. function maxrepli: integer;

   maximum number of replicates.

5. function advance: boolean;

control program advances time while all the following conditions hold: tnow is less than tmax, there are available next scheduled events and function advance returns true value. Its standard definition (redefinable for simulating more complicated conditions) is

```
function advance: boolean;
 begin
  advance := true
 end;
```

6. function replicate: boolean;

control program initiates new replicates while numrepli is less than or equal to maxrepli and this function returns true value.

Its standard definition (redefinable) is

```
function replicate: boolean;
 begin
  replicate: = true
 end;
```

7. function recomb(l1,l2: integer; chromosome)): real;

returns the genetic distance, in recombination units, between loci l1 and l2 in chromosome ch. When l1 equals zero it corresponds to the segregation probabilities corresponding to chromosome ch. It is used from procedure gamete and bgamete. In its standard form it always returns the value 0.5 that is, no linkage. Redefine it for simulating other linkage patterns.

The following three procedures have a "null" standard definition. The user must provide his own version (if needed) in every concrete application.

8. procedure report;

control program periodically activates this procedure to give a "report" of the simulation state.

9. procedure summary;

activated by the control program upon ending every replicate.

10. procedure initialize;

activated by the control program at the beginning of every replicate for initialization. Input in different models will probably be different as well.

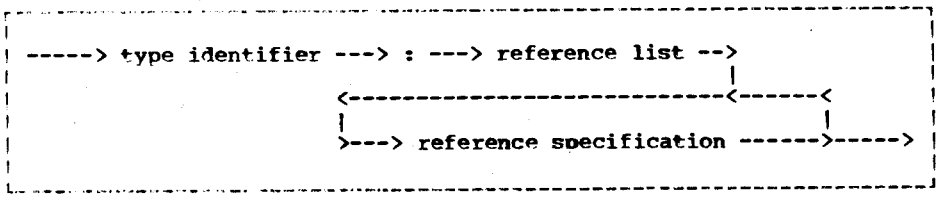
11. procedure control(maxre: integer; initim, maxtim,increport: real; namjob: string);

It is the control program, usually activated in the executive part of main block, after initialization of general simulation parameters -initialize does it for every specific replicate. The control activation properly starts simulation. Its parameters are: maxre (maximum number of replicates, it is the value constantly returned by function maxrepli), initim (internal initial time value of every replicate), maxtim (maximum time length of every replicate), increport (internal time increment between report activations) and namjob (a string identifying the job).

5. ECOGEN SYNTAX AND SENTENCES.

5.1 ENTITY DECLARATIVE PART

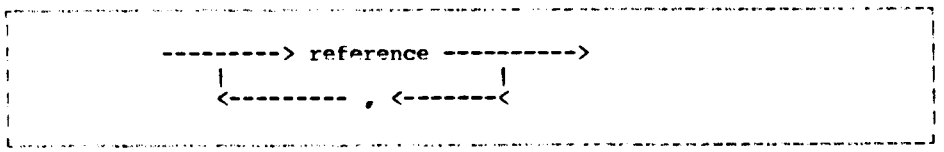
The profile of the entity classes and other model components is defined in part entity of main program declarative part by means of <standard specification> constructs. The general syntactical diagram of (standard specification) is



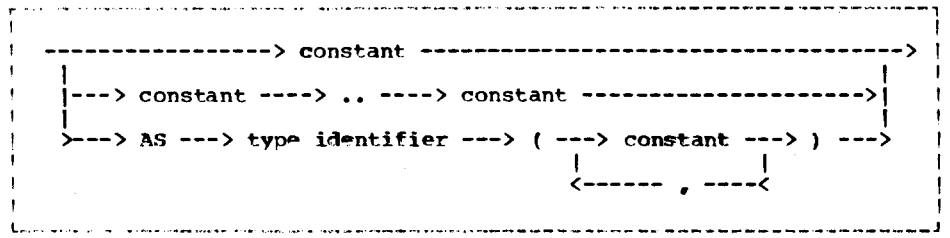
At a semantic level, the tipe identifier may be any of the following, standard scalar -- ECOGEN: class, name, event, metric, list, relation, chromosome, gene and allele.

<reference list> defines a set of values of the previously specified type. The forthcoming <reference specification> constructs refer to all elements in that set and characterize them (names associated to entity classes, genes associated to chromosomes, range of real values for metric attributes, etc.). They must be absent for the name and allele types.

The syntatical diagram of <reference list> is

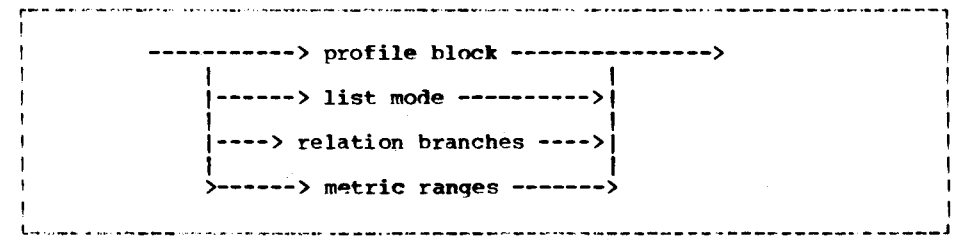


<reference> being defined as:

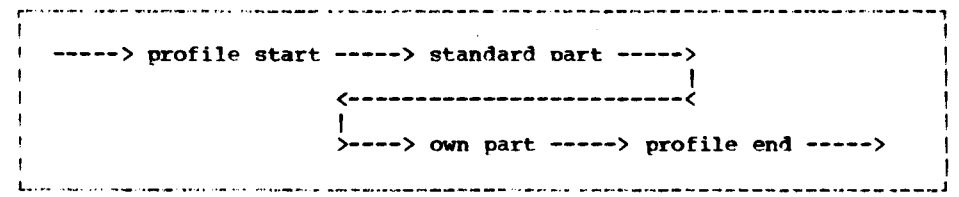


The first two possibilities correspond to the usual Pascal set constructors. The third one will be discussed later. It allows using previously given definitions.

The syntatical diagram of <reference specification> is



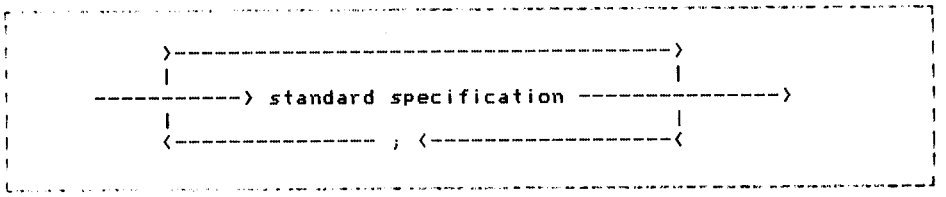
<profile block> is defined as



<profile start> may be either (they are equivalent) the reserved word profile or a left parenthesis. <profile end> may be the reserved word end or a right parenthesis.

The lists of constant values of other types associated to every element of <reference list> are specified in the <standard part> and in the <own part>. It would mean things like events, names or chromosomes associated to given (in <reference list>) entity classes, admissible allelic states for a given set of genes or admissible entity classes in one or more lists. The semantically admissible possibilities are summarized in the fields of the record variable proper (see section 4). In the own part (only for profile blocks defining classes of entities) the user can define entity attributes of any type, not necessarily standard ECOGEN.

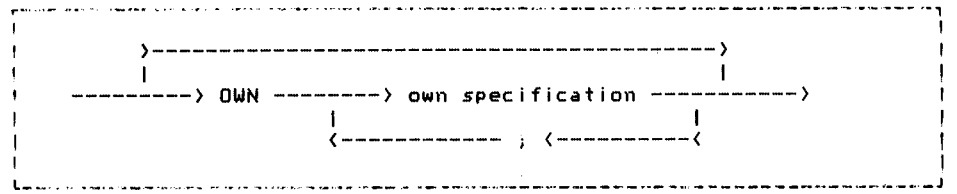
The syntax of <standard part> is



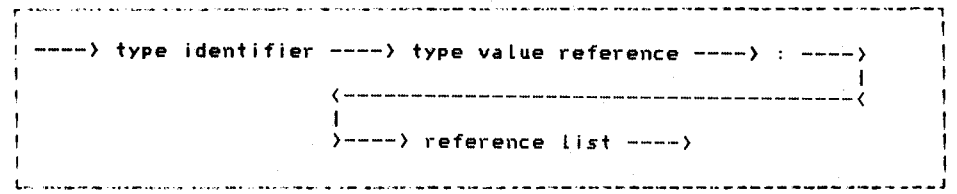
where <standard specification> has been previously defined.

When specifying references of a given type, say T, the construct as
<type identifier>(<constant>,{<constant>})
states that all references of tipe T previously associated to the constants (of the tipe after as) will be included too.

The syntax of <own part> is:



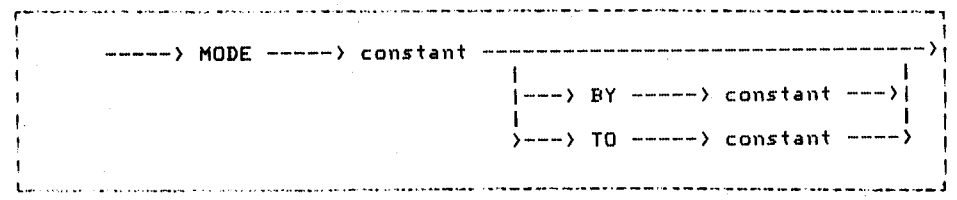
<own specification> id defined as:



and <type value reference>:

associating a specific type of value to the type of attribute.

The syntax of <list mode> is:

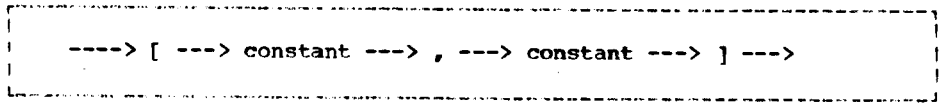


It only applies to type list standard specification. The constant following mode may have any of the possible modejoin type values. It establishes how concrete entities will join the list or lists. If its value is "preceding" or "following" it is necessary to specify to what other concrete entity, formerly in the list, by means of the construct to <constant>, this last constant having a type name value. If its value is "increasing" or "decreasing" it is necessary to specify the corresponding metric attribute, by means of the construct by <constant>, this constant having a type metric value.

The syntax of <relation branches> is:

It only applies to type relation standard specifications, defining how many "branches" they have, that is, the maximum number of concrete entities that can be related to a given concrete entity. The constant must have a non negative integer value.

The syntactic diagram of <metric ranges> is:



Both constants must have a real value. They respectively specify, for type event or type metric standard specifications, the range of possible occurrence times or the range of admissible values.

All the preceding specifications will be usually made once. If repeated, they must be equivalent. No assumptions are made by defect, except for the range values for time of events (zero to an implementation dependent maximum value) and for the real range values of metric attributes.

The remainder of this section is an illustrative example. Assume the following definitions:

```
class=(larva,pupa,adult);
name=(father,mother,partner,son);
list=(family);
chromosome=(autosome,X);
gene=(yellow,white,purple,vestigial);
allele=(wildyellow,mutantyellow,wildwhite,
       mutantwhite,wildpurple,mutantpurple,
       wildvestigial,mutantvestigial);
event=(birth,reproduction,death);
metric=(size,weight);
```

To specify that:

1. all entity classes share the specified chromosomes.
2. admissible genes in chromosome "autosome" are "purple" and "vestigial", admissible genes in chromosome "X" are "white" and "yellow".
3. "wildyellow" and "mutantyellow" are the admissible allelic states of gene "yellow" and so on.
4. names "father" to "partner" apply to "adult" class (that is, are used to designate concrete entities of that class) and "son" to classes "larva" and "pupa"
5. class "adult" has the list attribute "family".
6. the admissible "family" elements are of all classes.
7. they join "family" following the concrete entity named "mother" (in the moment of effectivelly joining)
8. "birth" event refers to "larva" and "pupa" "reproduction to "adult" and "death" to all classes
9. all entity classes share all metric attributes
10. their admissible values range from 0 to 1000

the following specifications should be made:

```
entity
    class: larva,pupa profile
                      chromosome: autosome,X;
                      name      : son;
                      event     : birth,death;
                      metric    : size,weight [0,1E3]
                     end;
    class: adult profile
                  chromosome: as class(larva);
                  name      : father..partner;
                  event     : death,reproduction;
                  metric    : as class(larva);
                  list      : family
                 end;
    list: family mode following to mother
                 profile
                  class: larva..adult
                 end;
    chromosome: autosome profile
                          gene: purple,vestigial
                         end;
    chromosome: X profile
                   gene: purple,vestigial
                  end;
    gene: yellow profile
                  allele: wildyellow,mutantyellow
                 end;
    ( using (,) instead of profile end )
    gene: white (allele: wildwhite,mutantwhite);
    gene: purple (allele: wildpurple,mutantpurple);
    gene: vestigial (allele: wildvestigial,mutantvestigial);
```

As an alternative possibility, the "family" definition can be nested in that or "adult"

```
entity
    ...
    ...
    class: adult profile
                  ...
                  list: family
                        mode following to mother
                        profile
                         class: larva..adult
                        end
                 end;
    ...
```

assume finally the definitions:

```
type
   onelabel=packed array[1..20]
                              of char;
   position=record
              x,y: real
            end;
   marking=(mark1,mark2);
   recollection=(lastrecollection);
```

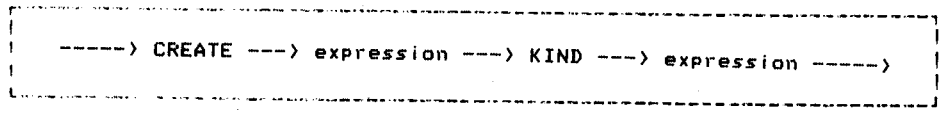
to specify that entities of class adult have the non standard attributes "mark1" and "mark2 of "marking" type, taking "onelabel" type values, and the attribute "lastrecollection" of "recollection" type, with "position" type values, we should specify at the end of the class adult profile clock:

```
...
class: adult profile
          ...
          list: family
       own
        marking(label): mark1,mark2;
        recollection(position): lastrecollection
       end;
...
```

5.2. ECOGEN EXECUTIVE STATEMENTS

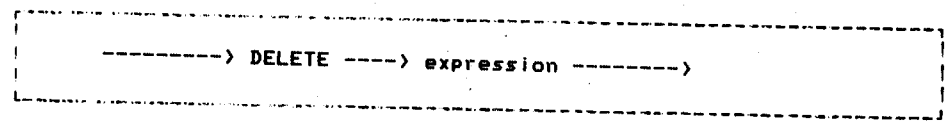
Concrete entity creation and deletion

To create a new concrete entity, ECOGEN provides the create statement, with syntax



The first expression must have a name tipe result. It is the name that will be associated to the new created entity (until a forget or delete statement is reached or until this name value being associated to another concrete entity, see next sections). The second expression must have a class type result. It is the class of the newly created concrete entity.

To delete an existing concrete entity, ECOGEN provides the delete statement, with very simple syntax

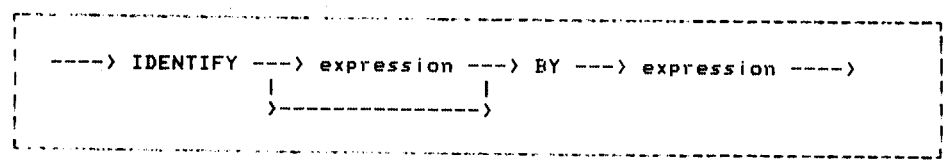


The expression result must be of type name, the name value designing the concrete entity to be eliminated.

Control of access to concrete entities

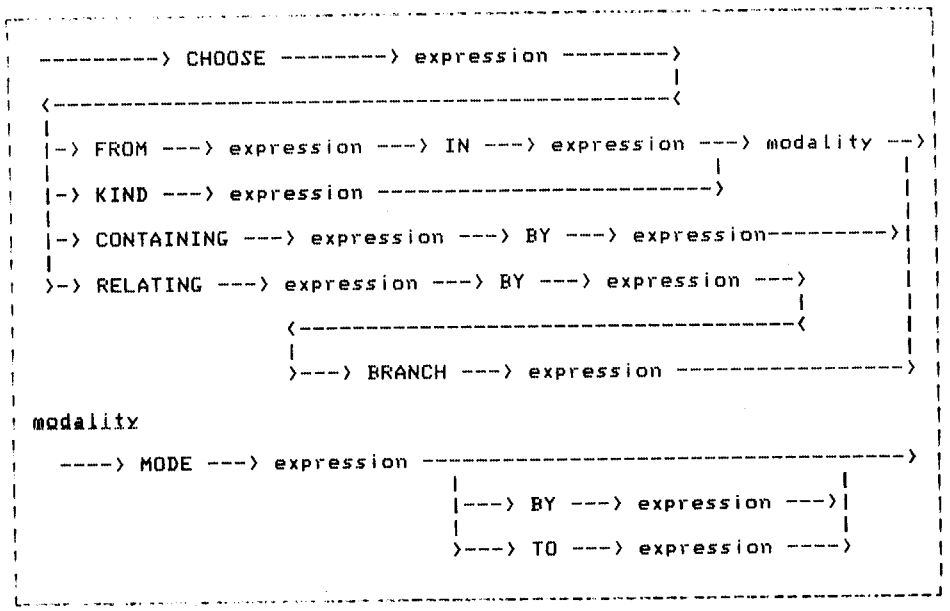
Run time changing of name association to concrete entities is possible using the following ECOGEN sentences

1. identity, with syntax



both expressions must give a name type result. First expression must designate a concrete entity. The second operand gives a new name value to be associated to this preceding concrete entity. The first name still remains as a possibility for designating this entity. If first operand is omitted, name value after by is associated to the last accessed concrete entity, that one most recently created (create), chosen (choose) -see next- or that one associated to the event now activated by the control program.

2. choose, "chooses" a concrete entity, associating it with a name. Its syntax is:



First operand, after choose, must have a name result. It is the name that will be associated to the chosen concrete entity -until this name is used to designate another concrete entity, for example by means of a posterior identify sentence.

Reserved words from, kind, containing and relating specify the choosing modality.

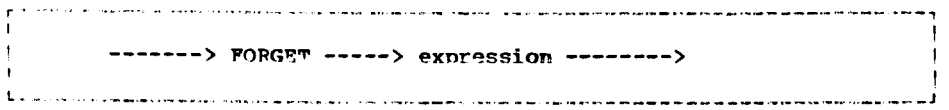
Using from a concrete entity is chosen from all belonging to a list (list value resulting from operand after from) that is an attribute of another concrete entity, designated by the name value resulting from expression after in. How to choose it in the list is specified by a modechoose type value resulting from expression after mode. If this value is first, last or atrandom, nothing more is needed. If this value es maximum or minimum it is necessary to specify a metric attribute (metric value resulting from expression after by). The concrete entity showing the maximum or minimum value for this attribute will be chosen from the list. If the corresponding modechoose type value is preceding or following, it is necessary to specify to what other concrete entity (also in the list) the chosen entity is inmediately preceeding or following. This is specified by the name value resulting from expression after to.

Using kind option, a concrete entity is chosen among all the concrete entities of the class specified by the class values resulting from expression after kind. As all concrete entities of the same class are lineraly ordered (in an order strictly determined by its creation time) forming something like a list, mode of choosing is specified as before.

When using option containing an entity is chosen that has a list attribute (specified by the list value returning expression after by) in which the concrete entity specified by the name resulting from expression after containing is contained as an element.

Finally, if option relating is used, the concrete entity related to the concrete entity specified by the name expression after relating is chosen. The relation is specified by the relation type resulting expression after by. "Branch" number is specified by the integer expression after branch.

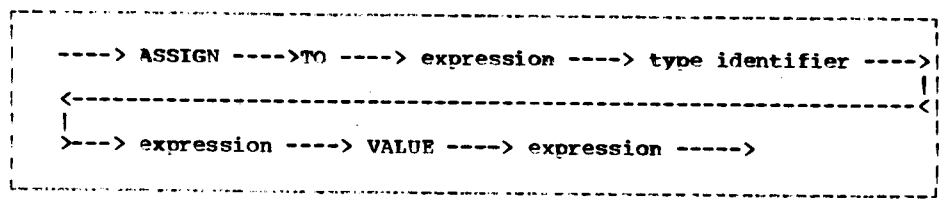
3. forget, with syntax



The association between the name value resulting from the expression and a concrete entity is overridden. The concrete entity still exists but can't be designated by this name.

Changing entity attribute values

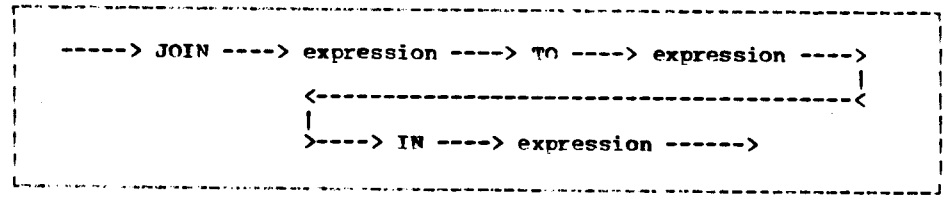
1. **assign**, with syntax



Assigns value resulting from expression after keyword value to the attribute of type (type identifier) (metric, chromosome or an own defined attribute) designated by second expression in concrete entity designated by name value resulting from first expression. The last expression must give a real, homologs or an own defined type attribute value corresponding, respectively, to the result of second expression which must be a metric, chromosome or an own defined type attribute.

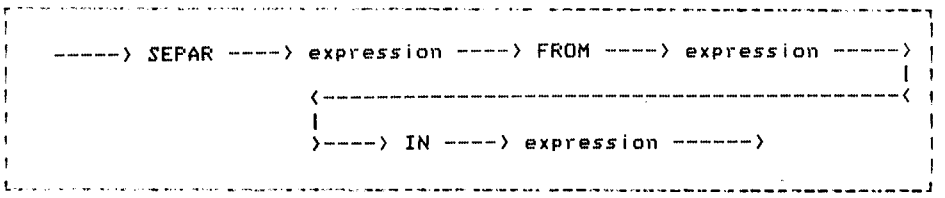
Changing relational attributes

1. **join**, with syntax



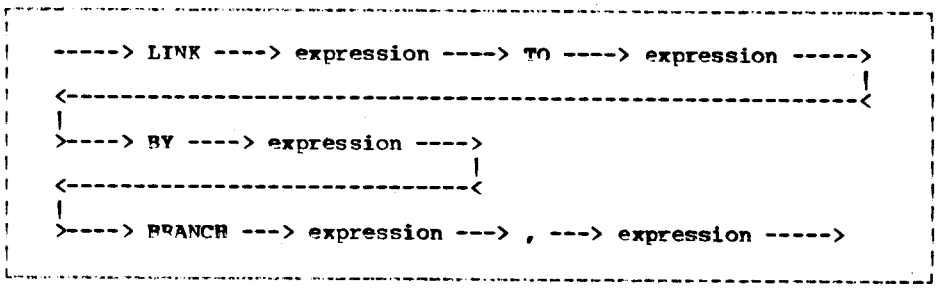
Adds a new concrete entity designated by the name value resulting from the first expression, to the list attribute designated by the list type resulting second expression, that was defined as a list attribute for the concrete entity designated by name expression after in. Mode of joining is that previously specified in entity part.

2. **separ**, with syntax



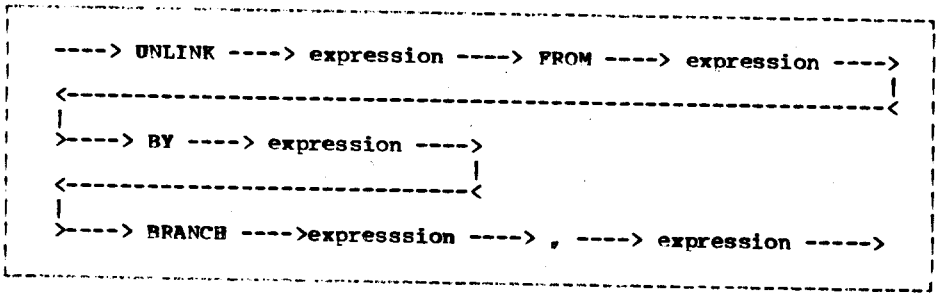
Separates a concrete entity from a given list. All operands have the same meaning as before.

3. **link**, with syntax



The first two expressions must give a name result. They designate two concrete entities to be related by relation resulting from third expression. The numbers of relation "branches" "coming" respectively from first and second concrete entities are designated by the last two integer expressions.

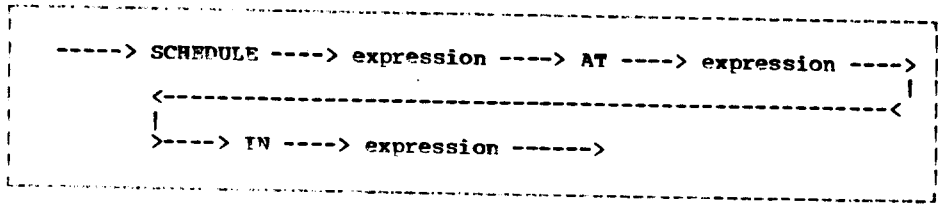
4. **unlink**, with syntax



Performs the inverse operation to that of link. All operands have the same meaning.

Event control

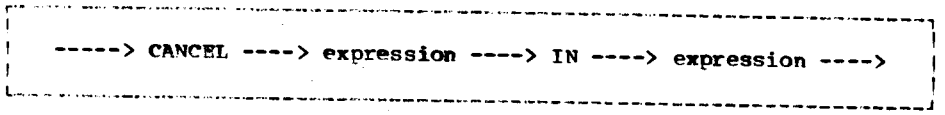
1. **schedule**, with syntax



First expression must have an event type result. It is scheduled, that is, it is going to "happen", at time value specified after at by the second expression, that must evaluate to a real type or compatible result, in the concrete entity designated by the name value resulting from the expression after in.

To try to schedule a still scheduled event will cause a run time error. Previously use the following statement:

2. cancel, with syntax



First operand must have an event type value result. Second operand must specify a concrete entity name. If the event was scheduled in the events list associated to that concrete entity, it will be removed. Cancelling non-scheduled events has no effect.

6. AN EXAMPLE

This is a rather artificial example designed to present a wide variety of ECOGEN elements without being too complex.

We are interested in simulating a random mating, diploid population with no sexual differentiation. Only one autosomic locus with two possible alleles (false and true, but now we call them A and a for clarity) is studied.

To represent the life cycle of individuals we assume that population is composed of two classes of living entities "adult" and "larva", with a chromosome attribute carrying only one gene with the alleles before mentioned.

Adult entities have a random life time, generated from an exponential distribution with constant mean 4. Once (or never) during its life time, they enter in a reproductive state. Time until entering this state is exponentially distributed with constant mean 2. If there are not other waiting, reproductively active adults, the new active adult joints a group of reproductively active ones, waiting for sexual reproduction, until it succeeds in mating or until its associated event "endofsex" happens. "endofsex" is scheduled on entering the reproductively active-waiting group, with a time increase exponentially distributed with mean 2. When an adult begins its active period, if there are available active partners (adults that have previously joined the "active group") mating can really happen or not, depending on the boolean function "mating". In case of successful mating, a partner is chosen at random from the active group, sexual reproduction occurs and the partner leaves the active group. In case of no mating the new active adult enters in the active group, as before.

When two adults mate they produce a fixed number of five new genotypes by sexual reproduction. These genotypes, assimilable to "eggs" experience some kind of natural selection, a differential viability defined by the probabilities of survival to "larva":

0.7 for genotype AA
0.9 for genotype Aa or aA
0.6 for genotype aa.

If a newly produced genotype "survives" (a "success" in the random decision with these associate probabilities) it produces a new larva.

Larvae entities can experience only the event "birth" implying its transformation in an adult. Time until birth is exponentially distributed with mean 0.5. To properly simulate the existence of the waiting-for-reproduction group of adults, we define an additional entity class named "group" with only one representative or concrete entity, named "reproducers", the concrete group of all active adults. If for example

there were differentiated males and females, perhaps it would be convenient to create two concrete representatives of this class, the group of all active males and the group of all active females. The only attribute associated to class "group" is the list "members".

The initial population is composed of five larvae all with heterozygotic genotype Aa.

Relevant output is made only at the end of every replicate, giving a complete inventory of final genotype frequencies. This is made by an appropiate redefinition of standard procedure summary.

The text of the ECOGEN program corresponding to this example can be found in appendix B. Output from this program is:

```
ECOGEN example
Beginning replicate  1 at time   0.0
ECOGEN example/Replicate:  1/Time   2.00
ECOGEN example
End of replicate  1 at time   4.2370
Genotype frequencies are
larva:
homozygotes A/A: 0
heterozygotes  : 0
homozygotes a/a: 0
adult:
homozygotes A/A: 2
heterozygotes  : 4
homozygotes a/a: 1
```

This final output can be reproduced by hand from:

1. Successive values generated by function exponential were:

   a. For betabirth=0.5 parameter:

      0.0136 1.1359 2.0166 0.1246 0.2972 0.2411 0.0202 0.3015
      0.1102 0.1028 1.0150 0.1058 0.0833 0.0862 0.02621 0.6582
      0.4346 0.0645 0.4993 1.0468

   b. For betaacti=2 parameter:

      2.5451 0.3574 4.3148 0.4083 1.1128 0.2332 1.7018 7.5712
      4.1815 0.0376 4.1043 0.0372 0.6941 0.0306 0.4588 2.4794
      0.6726 0.5702 4.7934 0.3123

c. For betadeath=4 parameter:

1.5600 2.6358 2.1229 1.1250 3.3851 6.4905 1.2325 3.9991
3.6652 3.9228 5.4852 8.1943 1.8070 0.1796 2.7943 2.9226
1.2436 1.5714 6.8012 0.9791

d. For betaend=2 parameter:

3.4810 0.9102 1.9619 0.4159 4.7994 1.3462 0.4843 1.5107

2. For the possible crosses, descendant genotypes were

a. Crossing AaxAa (or AaxaA, aAxaA, aAxAa)

AA AA aa Aa aa aa AA aa AA Aa Aa Aa Aa Aa

b. Crossing AaxAA (or aAxAA, AAxaA, AAxAa)

Aa Aa AA AA Aa Aa AA AA Aa AA AA Aa AA AA

c. Crossing Aaxaa (or aAxaa, aaxaA, aaxAa)

Aa Aa Aa aa aa aa aa aa Aa aa Aa aa aa aa Aa

(obviously, in absence of mutation, AAxAA always gives AA, aaxaa always gives aa and Aaxaa always gives Aa)

3. Condition

```
UNIFORM(0,1) <=fitness[genotzygo[1].CHROM[1].GEN[1]
                       gam[1].CHROM[1].GEN[1]]
```

were(f=false, t=true)

a. for fitness[A,A]=0.7: f t t f t t f t f f
b. for fitness[a,A]=fitness[A,a]=0.9: t t t t t t t t t f
c. for fitness[a,a]=0.6: f t t f f f t t t t

7. FINAL COMENTS

As this paper is only an overview of ECOGEN language, many questions will still remain unclear. Interested readers may see /6/ for a more comprehensive description of ECOGEN. We are now going to point out some possibly important questions.

The present version of ECOGEN is an evolutionary product of a first, much more baroque and ill-defined ECOGEN project /5/. This fact has been important in the present state of implementation, as we point out later. Clearly it still can not be considered a definitive version as it needs some improvements.

One obvious necessary improvement, corresponding to the capital importance of continuous models in Population Genetics and Evolutionary Ecology, is to extend it to allow handling of continuous or mixed discrete/continuous models, as has been made, for example, with GASP-IV language.

Some omissions are deliberate. There is, for example, the case of random variable generating functions. As they can be more or less easily programmed (knowing the overlying -- theory) and improved or adapted to every particular need (jointly with the possibility of using very good preexisting packages) we have left them to be defined in auxiliary libraries.

Another deliberately ignored point is some software to adjust additive, dominant and episthatic values. It will be presented in the future, as a separate library of auxi-

liar routines .

Implementation will be the subject of a following paper. As a first implementation step we wrote (in Pascal) a translator program, from ECOGEN into Pascal. This slow and bulky program (clearly improvable) was fully running on an Apple II microcomputer, under -- UCSD-Pascal. We had also begun to do something similar for the IBM 4341 under VS-CMS. All this work is now for the most part useless because, as we have seen, ECOGEN definition has been greatly changed. We hope to have very soon another working translator. The job of the ECOGEN into Pascal translator is, mainly, to analyse the const, and type parts of main block, making grow, in a stack like fashion, a chained list of records corresponding to all these definitions. Using this information it analyzes the specifications in the entity part of main block, initializing some standard variables (firstone, lastone, proper, cod, and other "unaccessible" to the user). It also creates an events routine from simulating clauses. Finally, it analyses and substitutes the ECOGEN executive statements (create, choose,...) by specific (also unaccesible directly by the user) Pascal procedure callings.

8. REFERENCES.

/1/ BASELGA, M. and NUEZ, F.: "Simulación. Su Contribución a la Genética". Monografias de la E.T.S. de Ingenieros Agrónomos. Nim. 1. Universidad Politécnica de Valencia (1975).

/2/ CROSBY, J.L.: "Computer Simulation in Genitics". J. Wiley (1973).

/3/ FRASER, A. and BURNELL, D.: "Computer Models in Genetics". McGraw-Hill (1970).

/4/ JENSEN, K. and WIRTH, N.: "Pascal User Manual and Report". Springer-Verlag (1974)

/5/ OCAÑA, J." The SGEN1 package for simulation in Population Genetics. XIth International Biometric Conference. Abstracts: 68. (1982).

/6/ OCAÑA, J., ALONSO, G. and RUIZ DE VILLA, M.C.: "El Lenguaje ECOGEN de simulación genética. Documento preliminar". Publicaciones de Bioestadística y Biomatemática, num. 15. Universitat de Barcelona,(1984).

/7/ ROUGHGARDEN, J.: "Theory of Population Genetics and Evolutionary Ecology" an introduction". McMillan. (1979).

9. APPENDIX A. ECOGEN SYNTAX: CHANGES AN ADDITIONS TO PASCAL SYNTAX.

The following changes must be made to /4/ appendix D to define the ECOGEN Language:

1. Replace ⟨block⟩ definition by:

```
⟨block⟩::=⟨label declaration part⟩
          ⟨constant definition part⟩
          ⟨type definition part⟩
          ⟨entity definition part⟩
          ⟨variable declaration part⟩
          ⟨ecogen sublocks declaration part⟩
          ⟨statement part⟩
```

2. Replace ⟨constant⟩ definition by:

```
⟨constant⟩::=⟨unsigned number⟩|⟨sign⟩⟨usigned number⟩|
             ⟨constant identifier⟩|⟨sign⟩⟨constant identifier⟩|
             ⟨string⟩|
             ⟨genetic value⟩
⟨genetic value⟩::="⟨chromatid value⟩"|⟨homologs value⟩"|
                  "⟨genotype value⟩"
⟨chromatid value⟩::=⟨allele value⟩(,⟨allele value⟩)|
                    ⟨allele value⟩( ⟨allele value⟩)|
                    ⟨string⟩
⟨allele value⟩::=⟨identifier⟩|'⟨character⟩'|
                 ⟨unsigned integer⟩|
⟨homologs value⟩::=(⟨chromatid value⟩(/⟨chromatid value⟩))
⟨genotype value⟩::=[⟨chromosome value⟩]⟨homologs value⟩
                          (:[⟨chromosome value⟩]⟨homologs value⟩)|
                    [⟨chromosome value⟩]⟨homologs value⟩
                          ( [⟨chromosome value⟩]⟨homologs value⟩)
⟨chromosome value⟩::=⟨allele value⟩
```

3. Add the following, between ⟨pointer type⟩ and ⟨variable declaration part⟩ definitions:

```
⟨entity definition part⟩::=⟨empty⟩|
                           entity(⟨standard specification⟩;)
⟨standard specification⟩::=
        ⟨type identifier⟩:⟨reference list⟩(⟨reference specification⟩)
⟨reference list⟩::=⟨reference⟩(,⟨reference⟩)
⟨reference⟩::=⟨constant⟩|⟨subrange⟩|⟨as specification⟩
⟨subrange⟩::=⟨constant⟩..⟨constant⟩
⟨as reference⟩::=as⟨type identifier⟩(⟨constant⟩(,⟨constant⟩))
⟨reference specification⟩::=⟨profile block⟩|⟨list mode⟩|
                            ⟨relation branches⟩|⟨metric ranges⟩
⟨profile block⟩::=⟨profile start⟩
                  ⟨standard part⟩
                  ⟨own part⟩
                  ⟨profile end⟩
⟨profile start⟩::=profile | (
⟨profile end⟩::=end | )
⟨standard part⟩::=⟨empty⟩|
                  ⟨standard specification⟩(;⟨standard specification⟩)
⟨own part⟩::=⟨empty⟩|
               own⟨own specification⟩(;⟨own specification⟩)
⟨own specification⟩::=
               ⟨type identifier⟩⟨type value reference⟩:⟨reference list⟩
⟨type value reference⟩::=⟨emptyt⟩|(⟨type identifier⟩)
⟨list mode⟩::=mode⟨constant⟩|
              mode⟨constant⟩ by⟨constant⟩
              mode⟨constant⟩ to⟨constant⟩
⟨relation branches⟩::=branch ⟨constant⟩
⟨metric ranges⟩::=[⟨constant⟩,⟨constant⟩]
```

4. Replace definitions of ⟨procedure and function declaration part⟩ to ⟨procedure declaration⟩, both included, by:

```
⟨ecogen sublocks declaration part⟩::=(⟨sublock declaration⟩;)
⟨sublock declaration⟩::=⟨event procedure declaration⟩|
                        ⟨procedure or function declaration⟩
⟨event procedure declaration⟩::=⟨event procedure heading⟩
                                ⟨ecogen sublock⟩
⟨event procedure heading⟩::=
                          procedure ⟨identifier⟩
                          simulating ⟨simulated events list⟩
⟨simulated events list⟩::=⟨constant⟩(,⟨constant⟩)
⟨ecogen sublock⟩::=⟨label declaration part⟩
                   ⟨constant declaration part⟩
                   ⟨type definition part⟩
                   ⟨variable declaration part⟩
                   ⟨procedure and function declaration part⟩
                   ⟨statement part⟩
⟨procedure and function declaration part⟩::=
                          (⟨procedure or function declaration⟩;)
⟨procedure or function declaration⟩::=⟨procedure declaration⟩|
                                      ⟨function declaration⟩
⟨procedure declaration⟩::=⟨procedure heading⟩⟨ecogen sublock⟩
```

5. Replace ⟨simple statement⟩ definition by:

```
⟨simple statement⟩::=⟨assignment statement⟩|
                     ⟨procedure statement⟩|
                     ⟨go to statement⟩|
                     ⟨ecogen statement⟩|
                     ⟨empty statement⟩
```

6. Between ⟨empty⟩ and ⟨structured statement⟩ definitions, add the following:

```
⟨ecogen statement⟩::=⟨cancel statement⟩|
                     ⟨create statement⟩|
                     ⟨identify statement⟩|
                     ⟨schedule statement⟩|
                     ⟨assign statement⟩|
                     ⟨join statement⟩|
                     ⟨separ statement⟩|
                     ⟨link statement⟩|
                     ⟨unlink statement⟩|
                     ⟨choose statement⟩|
                     ⟨forget statement⟩|
                     ⟨delete statement⟩
⟨cancel statement⟩::=cancel ⟨event value⟩
                     in ⟨concrete entity name⟩
⟨event value⟩::=⟨expression⟩
⟨concrete entity name⟩::=⟨expression⟩
⟨create statement⟩::=create ⟨concrete entity name⟩
                     kind ⟨entity class⟩
⟨entity class⟩::=⟨expression⟩
⟨identify statement⟩::=identify
                       by ⟨concrete entity name⟩|
                       identify ⟨concrete entity name⟩
                       by ⟨concrete entity name⟩
⟨schedule statement⟩::=schedule ⟨event value⟩
                       at ⟨real value⟩
                       in ⟨concrete entity name⟩
⟨real value⟩::=⟨expression⟩
⟨assign statement⟩::=assign to
                     ⟨concrete entity name⟩
                     ⟨type identifier⟩ ⟨destination attribute⟩
                     value ⟨expression⟩
⟨destination attribute⟩::=⟨expression⟩
⟨join statement⟩::=join ⟨concrete entity name⟩
                   to ⟨list value⟩
                   in ⟨concrete entity name⟩
⟨list value⟩::=⟨expression⟩
⟨separ statement⟩::=separ ⟨concrete entity name⟩
                     from ⟨list value⟩
                     in ⟨concrete entity name⟩
⟨link statement⟩::=link ⟨concrete entity name⟩
                   to ⟨concrete entity name⟩
                   by ⟨relation value⟩ ⟨link mode⟩
⟨relation value⟩::=⟨expression⟩
⟨link mode⟩::=branch ⟨branch number⟩,⟨branch number⟩
⟨branch number⟩::=⟨expression⟩
```

```
<unlink statement>::=unlink <concrete entity name>
                     from <concrete entity name>
                     by <concrete entity name>
<choose statement>::=choose <concrete entity name>
                     <choose way>
<choose way>::=from <list value>
               in <concrete entity name>
               mode <modality>|
               kind <entity class>
               mode <modality>|
               containing <concrete entity name>
               by <list value>|
               relating <concrete entity name>
               by<relation value>
               branch <branch number>
<modality>::=<mode choose>|
             <mode choose> by <metric value>|
             <mode choose> to <concrete entity name>
<mode choose>::=<expression>
<metric value>::=<expression>
<forget statement>::=forget <concrete entity name>
<delete statement>::=delete <concrete entity name>
```

10. APPENDIX B. PROGRAM EXAMPLE.

```
program example(output);
(*.....................................................................*)
(*.........ecogen program example, simple selection model.............*)
(*.....................................................................*)
 const
       maxloc=1;        (* max. number of loci per chromatid is now 1 *)
       betabirth=0.5;   (* mean time until event birth                 *)
       betaacti =2;     (* mean time until event activity              *)
       betadeath=4;     (* mean life time                              *)
       betaend  =2;     (* mean time of sexual activity                *)

 type class=(adult, larva, group);
      name =(newadult, active, partner, tired, deceased,
             inventoried, born, newlarva, reproducers);
      event=(activity, endofsex, death, birth);
      list =(membership);

(* ............ defining the profile of model elements ..............*)
 entity class: adult profile
                      name       : newadult..inventoried;
                      event      : activity..death;
                      chromosome: 1 profile
                                      gene: 1 profile
                                              allele: false,true
                                            end
                                    end
                     end;
        class: larva profile
                      name       : inventoried, newlarva;
                      event      : birth;
                      chromosome:as class(adult)
                     end;
        class: group profile
                      name: reproducers;
                      list: membership profile
                                         class: adult
                                       end
                                       mode last
                     end;
 var fitness: array[allele, allele]
                     of real;

(* ..................................................................... *)
 function exponential(beta:real): real;
  begin
   exponential:=-beta * ln(uniform(0,1))
  end;
```

```
(* ..... *)
 procedure initialize;
  var i: 1..5;
  begin
   create reproducers kind group;
   (* active adults group, init. empty *)
   for i:=1 to 5 do
       begin
        create newlarva kind larva;
        assign to newlarva chromosome 1
                          value "(1/0)";
        (* next birth of initial larvae: *)
        schedule birth
        at tnow + exponential(betabirth)
        in newlarva
       end
  end;

(* ..... *)
 procedure natal simulating birth;
  var homol: homologs;
  begin
   identify by born;
   create newadult kind adult;
   (* genotype of born assigned to newadult: *)
   assign to newadult chromosome 1
                    value homol;
   delete born; (* old larva dessapears *)
   schedule activity
   at tnow + exponential(betaacti) in newadult;
   schedule death
   at tnow + exponential(betadeath)   in newadult
  end;

(* ..... *)
 procedure sexual simulating activity;

  var zygo:1..5;
      genot1, genot2, gam:genotype;

  function mating: boolean;
   begin
    mating:= not empty(reproducers, membership)
    (* all matings succeed if there are available partenaires.
       write a more restrictive mating function for restricted mating *)
   end;

  procedure succedmat;
   begin
    choose partner
    from membership in reproducers
                                  mode atrandom
    valchrom(active, 1, genot1[1]);
    valchrom(partner,1, genot2[1]);
    for zygo:=1 to 5 do
    (* generate 5 offspring *)
        begin
         (* gamete coming from active initializes genotzygo: *)
         bgamete(genot1, genotzygo);
         bgamete(genot2, gam);
         if uniform(0,1)
            <=
            fitness[genotzygo[1].chrom[1].gen[1],
                    gam      [1].chrom[1].gen[1]]
            then (* viable *)
         begin
          (* add gam to genotype of newlarva to make it diploid *)
          joinchtd(gam[1].chrom[1], 2, genotzygo);
          create newlarva kind larva;
          assign to newlarva chromosome 1
                           value genotzygo[1];
          schedule birth
          at tnow + exponential(betabirth) in newlarva
         end
        end
   end;
```

```
  begin (* sexual *)
   identify by active;
   if mating then begin
                   succedmat;
                   cancel endofsex in partner;
                   separ partner from membership
                                    in reproducers
                  end
              else begin
                   join active to membership
                                    in reproducers;
                   schedule endofsex
                   at tnow + exponential(betaend)
                   in active
                  end
  end;

(* ..... *)
 procedure fatigue simulating endofsex;
  begin
   identify by partner;
   separ partner from membership in reproducers
  end;

(* ..... *)
 procedure funebre simulating death;
  begin
   identify by deceased;
   delete deceased
  end;


(* ..... *)
 procedure summary;

  var all1, all2:allele;
      h          :homologs;
      frec       :array[allele,allele] of integer;

  procedure inifrec;
   begin
    for all1:=false to true
        do
        for all2:= false to true
            do frec[all1,all2]:=0
   end;

  procedure countfrec;
   begin
    all1:=h.chrom[1].gen[i];
    all2:=h.chrom[2].gen[i];
    frec[all1,all2]:=frec[all1,all2] + 1
   end;

  procedure display(clase: class);
   begin
    inifrec;
    writeln(output,cod.cla[clase],':');
    choose inventoried kind classe
    mode first;
    while exists(inventoried)
          do
          begin
           valchrom(inventoried, 1, h);
           countfrec;
           choose inventoried kind classe
           mode following to inventoried
          end;
    writeln(output,'    homozygotes 1/1: ',frec[true,true]);
    writeln(output,'    heterozygotes  : ',frec[true,false]
                                           +frec[false,true]);
    writeln(output,'    homozygotes 0/0: ',frec[false,false])
   end;

  begin
   writeln(output, namjob);
   writeln(output, 'end of replicate ',numrepli,' at time ',tnow);
   writeln(output, 'genotype frequencies are ');
   display(larva);
   display(adult)
  end;
```

```
(* ..... *)
 begin
  fitness[true,true]:=0.7;
  fitness[true,false]:=0.9; fitness[false,true]:=0.9;
  fitness[false,false]:=0.6;
  control(1, (* number of replicates *)
          0, (* initial time for every replicate *)
          4, (* and duration *)
          2, (* partial results periodicity (report) *)
          'ecogen example')
 end.
```