The Family Description Language and The Instantiation process of GReAt

Oscar MEZA
Universidad Simón Bolívar
Ap. 89000, Baruta, Caracas, Venezuela
meza@usb.ve

Francis LOSAVIO, Oscar ORDAZ
Centro de Ingeniería de Software y Sistemas (ISYS)
Ap. 47567, Los Chaguaramos 1041-A,
Caracas, Venezuela.

Abstract

GReAt\(^1\) (Graph Researcher Assistant) is an environment for assisting researchers in graph theory. It allows the edition of directed graph families (sets of directed graphs described in terms of their characteristic properties). This paper presents the semantics and graphic notation of the Family Description Language (FDL) used in GReAt for family edition. The family instantiation process is studied, that is, how to obtain a digraph sample from a family given by an FDL expression. Finally, an algorithm to instantiate a family is proposed.

Key Words: graph editor, graph families, graph family instantiation, random graph generation, directed graphs.

---

\(^1\)Research project, undertaken by the Software Engineering Research Center ISYS at the Universidad Central de Venezuela, which is supported by the Consejo de Desarrollo Científico y Humanístico (CDCH) of the Universidad Central de Venezuela, the french Postgraduated Cooperation Program (PCP) in Informatics of the CEFI-CONICIT and the BID-CONICIT New Technologies Program.
1 Introduction

Several automatic tools have been developed to solve specific problems in the field of Graph Theory: The graph editors CABRI [1], AMDI [6, 7] and the conjecture generator system GRAFFITI [4]. Based on these antecedents, the GReAt (Graph Researcher Assistant) environment [8] offers a set of integrated tools to assist the graph researcher, automating its tasks whenever possible. Some tools of GReAt are dedicated to the graphic edition of directed graphs (or digraphs) and the graphic edition of digraph families (i.e., sets of digraphs).

An interesting and unconventional application of the family editor of GReAt is to aid the researcher in the generation of conjectures, building conjectures' counterexamples and bound examples, allowing the construction of families verifying a set of properties, and the random generation of a digraph sample from a digraph family (the instantiation process). Due to the lack of notational standardization and ambiguity, a formal description language (FDL) for describing digraph families have been defined. In this paper we present a characterization of the semantics and graphic notation of the Family Definition Language (FDL) of the family editor of GReAt and we propose an algorithm for the family instantiation process. Besides this introduction and the conclusions, the paper contains three sections. Sections 2 and 3 present respectively the semantics and graphic notation of Family Description Language. Section 4 describes the instantiation process, that is, how to generate at random a digraph sample from a digraph family.

2 Semantics of the Family Description Language (FDL) of GReAt

A family of digraphs is a set of non-labelled digraphs. By a non-labelled digraph we mean the isomorphism class of a digraph (e.g., a circuit with three vertices is the isomorphism class of all the labelled circuits with three vertices). A family of digraphs may be defined by means of a set of invariant constraints (i.e., the digraphs of the family are those satisfying each invariant constraint in the set) or by relationships, or "operations", between some families. For example, the family of digraphs having at most 10 vertices and chromatic number equal to 4 is a family defined by a set of invariants, here the two invariants involved are "number of vertices" and "chromatic number". The family obtained by the "union" (set union) of families F1 and F2 is a family defined by relationships of some families, here the relationship is the "union operation" between families F1 and F2. An Instance of a family is a labelled digraph isomorphic to some digraph in that family. An instantiation process of a family, or "an instantiation of a family" for short, is a process that allows to obtain an instance of the family.

In GReAt, we will not be able to describe all families but a subset of them. The subset of families that can be described in GReAt is determined by the list of invariant constraints (e.g., chromatic number = 3, Hamiltonian = true, etc.) and the list of operations between families (e.g., cartesian product) that
are defined in GReAt. Obviously, this subset can be enlarged. So, we will not present here neither the list of such invariant constraints nor the list of such operations. Instead of that, in what follows we will present some new operations between families, which are inspired from the graphic representations that a researcher usually makes of a family[5, 2]. The definition of the new operations on families will depend on the structure of the families. Thus, we will give an inductive definition of a family and at the same time we will present the new operations. This inductive definition eases implementation issues in GReAt.

In what follows, the symbol's string that represents an operation (e.g., + represents the addition operation) will be called Operator. A family F will be represented by the following structure:

\[ F = \langle Id, Type, Subfamilies, Operators, InvConstraints \rangle \] (1)

\( Id \) is a family identifier (there are not two different families with the same identifier, thus a family is completely determined by its identifier), \( Type \) is an integer, \( Subfamilies \) is a sequence of distinct family identifiers, \( Operators \) is a sequence of operators on families, \( InvConstraints \) is a set of invariant constraints. We denote by \( set(F) \) the set of instances of \( F \).

The \( Type \) attribute represent the family type (i.e., defined by invariant constraints, defined by operations, etc.) and can be 0, 1, or 2:

- A \( Type=0 \) family will be called defined by invariant constraints. In this case, \( Subfamilies \) and \( Operators \) are empty sequences and the family consists of all digraphs that satisfy each invariant constraint in the set \( InvConstraints \).

- A \( Type=1 \) will be called “defined by an operation”. In this case, \( InvConstraints \) is the empty set, \( Operators \) is a sequence having only one element which is an operator associated to some operation on families (e.g., “cartesian product”, “family union”, etc.) and \( Subfamilies \) is the sequence of families (\( Type=0, 1 \) or 2) for which the operation applies.

- A \( Type=2 \) family will be called constrained by formal pattern. \( InvConstraints \) is the empty set. \( Subfamilies \) is a sequence of identifiers of families \( Type=0, 1 \) or 2, these identifiers must be different to \( Id \). If \( Subfamilies \) is the empty sequence, the family represents the empty family. \( Operators \) may be empty. The operations represented by the operators in \( Operators \) will be called Formal Pattern operations and will be described below.

In order to present the Formal Pattern operations, it is necessary to introduce some concepts related to a family. Let \( G,F \) be two families. \( G \) is said to be a descendent of \( F \) if \( G=F \) or \( (Type(F)=2 \) and \( G \) is descendent of some family in \( Subfamilies(F) \)). If \( F \) is a family constrained by formal pattern, we say that a family in \( Subfamilies(F) \) is a son of \( F \). If \( F' \) is a descendent of a family \( F \) and \( D \) is an instance of \( F \) the subset of vertices of \( D \) corresponding to \( F' \) is the set of vertices of the instance of \( F' \) (which is a subgraph of \( D \)).

Let \( (c,r,l,r,l) \) be a boolean vector where: \( c = true \) stands for the sentence “there must exist at least an
arc”. \( r = \text{true} \) stands for “Add with probability 1/2 an arc from the first vertex to the second vertex”. \( l = \text{true} \) stands for “Add with probability 1/2 an arc from the second vertex to the first vertex”. \( rl = \text{true} \) stands for “Add with probability 1/2 two arcs in opposite senses between the first and second vertices”.

For each vector \((c,r,l,rl)\) we will define a Formal Pattern operator, denoted by \(FP\). Thus there will be virtually 16 different Formal Pattern Operators. The Formal Pattern operation corresponding to \(FP\) is an application \( F^3 \to \Gamma \), where \( \Gamma \) is the set of Type 0, 1 or 2 families, and \(FP\) will be defined only for \(F1, F2, F3\) such that \(\text{Type}(F3) = 2\). \(F1\) and \(F2\) are two descendents of \(F3\) belonging respectively to two distinct sons of \(F3\). The family \(FP(F1,F2,F3)\) has Type=2 and its structure can be obtained from \(F3\) as follows:

\[
FP(c,r,l,rl)(F1,F2,F3) = \langle \text{Id,2, Subfamilies}(F3), \text{Operators}(F3) | (FP(c,r,l,rl)(F1,F2)), \emptyset \rangle
\]

Every instance of \(FP(F1,F2,F3)\) is obtained from an instance \(D\) of \(F3\) adding to \(D\) a set of arcs \(A_{FP}\) which depends on \((c,r,l,rl)\). Let \(V_1\) and \(V_2\) be the sets of vertices of \(D\) corresponding to \(F1\) and \(F2\) respectively. For each pair of vertices \(v_1 \in V_1\) and \(v_2 \in V_2\), the following process must be done in order to obtain \(A_{FP}\) (initially, \(A_{FP}\) is empty):

- if \(r=\text{true}\) (i.e., Add with probability 1/2 an arc from the first vertex to the second vertex) then we add with probability 1/2 the arc \((v_1,v_2)\) to \(A_{FP}\),
- if \(l=\text{true}\) (i.e., Add with probability 1/2 an arc from the second vertex to the first vertex) then we add with probability 1/2 the arc \((v_2,v_1)\) to \(A_{FP}\),
- if \(rl=\text{true}\) (i.e., Add with probability 1/2 two arcs in opposite senses between the first and second vertices) then we add with probability 1/2 the two arcs \((v_1,v_2)\) and \((v_2,v_1)\) to \(A_{FP}\).

- if \(c=\text{true}\) (i.e., there must exist at least an arc between \(v_1\) and \(v_2\)) and no arc between \(v_1\) and \(v_2\) has yet been added to \(A_{FP}\), an arc between \(v_1\) and \(v_2\) must be chosen uniformly likely from the subset of \(\{r,l,rl\}\) having \(\text{true}\) values (e.g., if \(r=\text{true}\), \(l=\text{false}\) and \(rl=\text{true}\), the subset of \(\{r,l,rl\}\) having \(\text{true}\) values is \(\{r,rl\}\) thus with probability 1/2 we choose \(r\) or \(rl\). If \(r\) is chosen then we add to \(A_{FP}\) the arc \((v_1,v_2)\), otherwise \(rl\) has been chosen, and thus we add to \(A\) the two arcs \((v_1,v_2)\) and \((v_2,v_1)\)). Hence, this subset must be non-empty.

Finally, the instances of a family \(F\) constrained by formal pattern are obtained as follows: Let \(D\) the disjoint union of an instance of each son of \(F\). For each operator \(FP\) in \(\text{Operators}(F)\), let \(A_{FP}\) the set of arcs obtained as above. An instance of \(F\) is obtained by adding to \(D\) the arcs in each \(A_{FP}\).

Remarks:
(a) Let \(\Delta(D)\) be the set of all instances that can be generated from \(D\) by means of the previous process. Let \(S_r\) be equal to \(\{(v_1,v_2)/v_1 \in V_1\) and \(v_2 \in V_2\}\) if \(r=\text{true}\), \(\emptyset\) otherwise. Let \(S_l\) be equal to \(\{(v_2,v_1)/v_1 \in V_1\) and \(v_2 \in V_2\}\) if \(l=\text{true}\), \(\emptyset\) otherwise. Let \(S_{rl}\) be equal to \(\{(v_1,v_2),(v_2,v_1)/v_1 \in V_1\) and \(v_2 \in V_2\}\) if \(rl=\text{true}\), \(\emptyset\) otherwise. Then:
• If \( c = \text{false} \) then \( \Delta(D) = \{D' / V(D') = V(D), A(D') = A(D) \cup S, S \subseteq S_r \cup S_t \} \)

• If \( c = \text{true} \) then \( \Delta(D) = \{D' / V(D') = V(D), A(D') = A(D) \cup S, S \subseteq S_r \cup S_t \text{ and } \forall v_1, v_2 \in V_1, V_2 \text{ respectively, } (v_1, v_2) \in \mathcal{E} \} \)

(b) \( F_P(\text{false,false,false,false}) \) \( (\text{F}_1, \text{F}_2, \text{F}_3) = \text{F}_3 \) and the vector \( (c, r, l, r_l) = (\text{true, false, false, false}) \) is meaningless.

(c) Let \( F \) be a family Constrained by Formal Pattern. Let \( F' \) be a family such that Type\((F') = 2\), Subfamilies\((F') = \text{Subfamilies}(F)\), Operators\((F') \) is a permutation of the sequence Operators\((F)\). Then set\((F) = \text{set}(F')\).

In order to increase the expressive power of the family description structure in equation (1), we will add four boolean attributes to the structure, namely: Optionality, External Connection, Vertex addition and Arc Addition. "Optionality" and "External Connection" attributes apply to all type of families, "Vertex Addition" and "Arc Addition" attributes apply only to families Constrained by Formal Pattern \( (\text{Type} = 2) \). So, Type\( = 0 \) or 1 families will have Vertex Addition and Arc Addition attributes equal to \text{false}.

A family \( F \) will be represented by the new structure:

\[
F = < \text{Id, Type, Subfamilies, Operators, InvConstraints, Optionality, } \\
\text{External Connection, Vertex Addition, Arc Addition} >
\]  

(2)

Optionality\( = \text{true} \) means that set\((F) \) contains the empty graph.

We associate an attribute to each vertex of an instance of a family. This attribute will be called External Connection Vertex attribute and has one of the three values \text{true, false or undefined}. Now it will be presented the process followed to generate an instance of a family \( F \) given as (2).

Let \( F \) be a family such that VertexAddition\((F) = \text{ArcAddition}(F) = \text{false} \) or Type\((F) = 0 \) or Type\((F) = 1 \). An instance of \( F \) can be obtained from an instance \( D \) of \(< \text{Id, Type, Subfamilies, Operators, InvConstraints}> \) obtained as above, and then modifying the "External Connection Vertex" attribute to each vertex of \( D \) as follows: if ExternalConnection\((F) = \text{true} \) then for each vertex \( v \) of \( D \) with "External Connection Vertex" attribute equal to \text{undefined}, set "External Vertex Attribute" of \( v \) equal \text{true}. If ExternalConnection\((F) = \text{false} \) then set to \text{false} the "External Connection Vertex" attribute of each vertex of \( D \). Notice that a vertex of \( D \), before modifying its External Connection Vertex attribute, will have the External Connection Vertex attribute equal to \text{undefined} if Type\((F) = 0 \) or 1. If Type\((F) = 2 \) then a vertex of an instance obtained from \(< \text{Id, Type, Subfamilies, Operators, InvConstraints}> \) will always have the "External Connection Vertex" attribute set to \text{true} or \text{false} because this vertex corresponds to a son of \( F \).

Let \( F \) be a family such that VertexAddition\((F) = \text{true} \) and ArcAddition\((F) = \text{false} \) and Type\((F) = 2 \). An instance \( D' \) of \( F \) can be obtained from an instance \( D \) of \(< \text{Id, Type, Subfamilies, Operators, InvConstraints}> \) obtained as above. Let \( D' \) be equal to \( D \). Add to \( D' \) some isolated vertices with "External
Let $F$ be a family such that $\text{VertexAddition}(F) = \text{true}$ and $\text{ArcAddition}(F) = \text{true}$ and $\text{Type}(F) = 2$. An instance $D'$ of $F$ can be obtained from an instance $D$ of $<\text{Id}, \text{Type}, \text{Subfamilies}, \text{Operators}, \text{InvConstraints}>$ obtained as above. Let $D'$ be equal to $D$. $\text{VertexAddition}(F) = \text{true}$ means to add to $D'$ some isolated vertices with “External Connection Vertex” attribute equal $\text{true}$. After adding isolated vertices, if $\text{ArcAddition}(F) = \text{true}$ then for each ordered pair of vertices $(v_1, v_2)$ in $D'$ such that $v_1$ and $v_2$ do not correspond to the same son of $F$ and, $v_1$ and $v_2$ have “External Connection Vertex” attribute respectively equal $\text{true}$, then add to $D'$ an arc from $v_1$ to $v_2$ with probability $1/2$. Then modify the “External Connection Vertex” attribute to each vertex of $D'$ as follows. If $\text{ExternalConnection}(F) = \text{false}$ then set to $\text{false}$ the “External Connection Vertex” attribute of each vertex of $D$.

In section 4, an example of the instantiation process is presented in order to illustrate the meaning of all the attributes given in this section.

### 3 Graphic Notation of the Family Description Language of GReAt

In this section, the FDL of section 2 will be described by means of a graphic (or visual) notation. The graphic notation is shown in order to facilitate the reading and should not be considered as a definitive one. The language power should not be judged on this base, since the experience on the first GReAt family editor prototype will permit us to improve this aspect. For interfaces purposes, we will distinguish five types of graphic family constructors: One-node Family, Family defined by Invariant Constraint, Family Constrained by Formal Pattern, Family defined by an Operation, User Defined Family.

The family editor of GReAt allows to save families in a database. Every family in the database previously saved by the user, has a name or identification. Hence, the User Defined Family constructor allows to create a family which refers to another family in the database by its name.
Figure 2: Families defined by invariant constraints and by an operation

Figure 1 shows a family having six subfamilies. Every family different from one-node family is represented by a plain or adorned rectangular frame with white or grey background, the name of the family appears at the upper left corner of the rectangle. The one-node families are represented by a circle. Family 1 is a family constrained by formal pattern as family 1.1 too. Family 1.1.1 is a family constrained by invariant. Family 1.2 is a user defined family. Families 1.3 and 1.4 are one-node families.

3.1 Family defined by Invariant Constraints, One-node Family and User Defined Family

An example of a family defined by Invariant Constraints is the family of digraphs for which the Hamiltonian invariant has true value and $\delta^+\min > 5$. For notational conventions, a list of invariant constraints surrounded by a rectangular frame, denotes the family of digraphs verifying the invariant constraints (see figure 2).

The one-node family constructor corresponds to a family constrained by invariant where the InvConstraints is $n = 1$, i.e., the number of vertices is equal to 1. Hence the only instance of such family is an isolated vertex. The one-node family is represented graphically by a circle.

In order to refer to a family in the database, in other family descriptions, the user must assign a name to each family created and saved in the database. A family created by the User Defined Family constructor, is a family that refers to another previously created family. For example, in figure 1 family 1.2 refers to family XXX. Hence, an User Defined Family F is a family constrained by invariants where the only invariant is “belongs to set($F'$)”, $F'$ being the referred family.

3.2 Family defined by an Operation

A family defined by an operation is represented by a rectangular frame having at upper left corner the name of the family and the name of the operation and at center the frames, placed side by side, of families on which operation apply. As an example, in figure 2 we have family YYY which is the Union of families ZZZ and WWW.
3.3 Family Constrained by Formal Pattern

A digraph family constrained by formal pattern is a family consisting of several subfamilies related each other by means of formal pattern operators also called links. The subfamilies can be created by any constructor. The links relate two descendents of a family. In figure 1 we can see links connecting some descendents of family 1. We can see that there are links between families 1.1.1 and 1.1.2, 1.1 and 1.2, 1.3 and 1.4, 1.1.1 and 1.2 respectively.

A link between two families F₁ and F₂ establishes the connection pattern between each pair of vertices v₁ and v₂ of G₁ and G₂ respectively (G₁ and G₂ are instances of F₁ and F₂ respectively). The mapping between the Formal Pattern operators \( FP_{(c,r,l,rl)}(F₁,F₂,F₃) \) and the corresponding graphic links are showed in figure 3: each meaningful value of \((c,r,l,rl)\) has a graphic representation. We have:

- \( c=\text{true} \), i.e., there must exist at least one arc between v₁ and v₂, is represented graphically by a dotted segment between v₁ and v₂.
- \( r=\text{true} \), i.e., add an arc from v₁ to v₂, is represented graphically by an arc from v₁ to v₂.
- \( l=\text{true} \), i.e., add an arc from v₂ to v₁, is represented graphically by an arc from v₂ to v₁.
- \( rl=\text{true} \), add an arc from v₁ to v₂ and an arc from v₂ to v₁, is represented graphically by a segment between v₁ and v₂.

Now we present the graphic issues concerning the four family attributes “Optionality”, “External Connection”, “Vertex Addition” and “Arc Addition”:

(a) **Optionality**: This attribute stands for the optional character of the family. A family is optional if the instantiation process for this family allows to generate the empty digraph (i.e., the digraph with no vertex). Optionality = \( \text{true} \) is represented by a grey background of the family picture. Optionality = \( \text{false} \) is represented by a white background of the family picture.

(b) **External Connection**: External Connection = \( \text{true} \) is represented by means of a dotted line for the family rectangular frame. External Connection = \( \text{false} \) is represented by means of a straight line for the family rectangular frame.

(c) **Arc Addition**: Arc Addition attribute = \( \text{true} \) is represented by an arrow drawn at the upper right corner of the family rectangular frame. When Arc Addition attribute = \( \text{false} \), the arrow does not appear.

(d) **Vertex Addition**: Vertex Addition attribute = \( \text{true} \) is represented by an small circle drawn at the upper right corner of the family rectangular frame. When Vertex Addition attribute = \( \text{false} \), the small
circle does not appear.

We can set up the attributes for a family created by any of the five family constructors. For certain constructors, some attributes are predefined. Thus, for any constructor other than "constrained by formal pattern" constructor, the "Arc Addition" and "Vertex Addition" attributes are predefined as false (i.e., they cannot be modified) to forbid arc additions and vertex additions. "Optionality" and "External Connection" attributes can be set up for families created by any constructor. All the attributes of a family created with "constrained by formal pattern" constructor, (i.e., a Type=2 family) can be set during the family construction process. In figure 1, family 1.1 is optional (grey background), allows connection with the exterior (dotted frame), allows arc and vertex additions (arrow and small circle at upper right corner respectively). A true value of any of the four attributes means to add a new property to the family that is being defined, for example, "Optionality" stands for adding the empty graph to the family, "Vertex Addition" allows the addition of isolated vertices to each instance of the drawn family, and so on. So, the default value of the four attributes is set to false when we are graphically creating a family. If a user needs to add any of the four properties to the family being created, he must set up explicitly to true the respective attribute.

4 The Instantiation Process

In this section, an algorithm to generate instances of a family is presented. The algorithm takes a family (Type = 0, 1 or 2) and returns an instance sample from that family with a certain probability. The general problem of generating an instance from any type of family is a difficult problem that, in general, may not have solution [4, 7, 10]. So, the algorithm describes the general instantiation process of a family, but essentially it describes the instantiation process of a Type=2 family. For families defined by invariant constrains, in [7] some ad hoc algorithms for a given set of invariants are proposed. We will suppose for families Type = 0 or 1, that an ad hoc algorithm to generate an instance of such family is given. It can be seen that if every instance have a non-zero probability of being generated by the ad hoc algorithms, so will do the algorithm given bellow. Hence, any instance of a family constrained by formal pattern will have a non-zero probability of being generated.

Now, we will illustrate by an example the instantiation process of family 1 in figure 1. Note that family 1, is described by $<1, 2, <1, 1, 2, 1, 3, 1, 4>, <(FP(1, 1, 0, 1), (1, 1, 1, 2)), (FP(0, 0, 0, 1), (1, 1, 1, 2)), (FP(1, 0, 0), (1, 3, 1, 4))>, 0, 0, 0, 0, 0>$. An instance of family 1 may be obtained recursively by obtaining, at first, an instance from each family 1.1, 1.2, 1.3, and 1.4. Suppose that digraphs $G_1, G_2, G_3$ and $G_4$ in figure 4 are respective instances of families 1.1, 1.2, 1.3 and 1.4. Note that some vertices are dotted, that is, their external connection vertex attribute has true value. All the other vertices have external connection attribute equal to false. Now, the operators in $FP(1, 1, 0, 1) (1, 1, 1, 2), FP(0, 0, 0, 1) (1, 1, 1, 2, 1), $
Figure 4: Obtaining an instance of a family

$FP_{(0,1,0,0)} (1.3, 1.4, 1)$ are applied to $G_1 \cup G_2 \cup G_3 \cup G_4$ to obtain the digraph $H_1$. Note that the two first operators will be applied to each pair of vertices of instances of families 1.1.1 and 1.2 respectively. As the “Arc Addition” attribute of family 1 is true, some arcs are added to $H_1$ between vertices with “external connection vertex” attribute equal true (i.e., dotted vertices) to obtain, finally, the instance $H_2$ of family 1.

The algorithm will be recursive and will be presented in a Pascal-like form. It is a function that returns a graph instance of the family in the formal parameter. Besides the External Connection Vertex attribute (see equation 2 in section 2), we introduce another attribute that will be called family ownership attribute. This attribute contains the id (see section 2) of the family that created the vertex at the instantiation. For example, the vertex $v$ of $G_1$ in figure 4 created at the instantiation of the descendent family 1.1.2 in figure 1, has family ownership attribute equal to the family id 1.1.2. Both the external vertex and the family ownership attributes have been defined only for instantiation purposes.

The Instantiation algorithm follows:

Function Instantiation($F$:Family):Digraph;
(1) If $Type(F) = 0$ or 1 then let $G$ be the instance generated (or returned) by the ad hoc algorithm of family $F$.
(2) If $Type(F) = 2$ then let $G_1, G_2, \ldots, G_n$ be the instances obtained from the recursive call of Instantiation applied to the subfamilies $F_1, F_2, \ldots, F_n$ of $F$ respectively. If a subfamily $F_i, 1 \leq i \leq n$, is a Type=0 or Type=1 family, the algorithm will not be called recursively for this family but the ad hoc algorithm for family $F_i$ will be called to obtain the instance $G_i$. Let $G = G_1 \cup G_2 \cup \ldots \cup G_n$ (If $n = 0$, $F$ does not contain subfamilies and $G$ will contain the empty graph).
(2.1) For each operator $(FP_{(e,r,t,f)}, (FV, FW))$ in $Operators(F)$ do:
(2.1.1) For each pair of vertices \((v, w)\) of \(G\) such that \(\text{family-ownership}(v)\) is a descendent of \(F_V\) and \(\text{family-ownership}(w)\) is a descendent of \(F_W\) do:

(2.1.1.1) if \(r=\text{true}\) then we add with probability 1/2 the arc \((v, w)\) to \(G\),

(2.1.1.2) if \(l=\text{true}\) then we add with probability 1/2 the arc \((w, v)\) to \(G\),

(2.1.1.3) if \(rl=\text{true}\) then we add with probability 1/2 the two arcs \((v, w)\) and \((w, v)\) to \(G\).

(2.1.1.4) if \(c=\text{true}\) and no arc between \(v\) and \(w\) has yet been added to \(G\), an arc between \(v\) and \(w\) must be chosen uniformly likely from the subset of \(\{r, l, rl\}\) having \text{true} values (e.g., if \(r=\text{true}\), \(l=\text{false}\) and \(rl=\text{true}\), the subset of \(\{r, l, rl\}\) having \text{true} values is \(\{r, rl\}\) thus with probability 1/2 we choose \(r\) or \(rl\). If \(r\) is chosen then we add to \(G\) the arc \((v, w)\), otherwise \(rl\) has been chosen, and thus we add to \(G\) the two arcs \((v, w)\) and \((w, v)\)).

(3) If \(\text{VertexAddition}(F) = \text{ArcAddition}(F) = \text{false}\) or \(\text{Type}(F) = 0\) or \(\text{Type}(F) = 1\): if \(\text{ExternalConnection}(F) = \text{true}\) then for each vertex \(v\) of \(G\) with “External Connection Vertex” attribute equal to \text{undefined}, set “External Vertex Attribute” of \(v\) equal \text{true}. If \(\text{ExternalConnection}(F) = \text{false}\) then set to \text{false} the “External Connection Vertex” attribute of each vertex of \(G\).

(4) If \(\text{VertexAddition}(F) = \text{true}\) and \(\text{ArcAddition}(F) = \text{false}\) and \(\text{Type}(F) = 2\): Add to \(G\) some isolated vertices with “External Connection Vertex” attribute equal \text{true} and then modify the “External Connection Vertex” attribute to each vertex of \(G\) as follows. If \(\text{ExternalConnection}(F) = \text{false}\) then set to \text{false} the “External Connection Vertex” attribute of each vertex of \(G\).

(5) If \(\text{VertexAddition}(F) = \text{true}\) and \(\text{ArcAddition}(F) = \text{true}\) and \(\text{Type}(F) = 2\): \(\text{VertexAddition}(F) = \text{true}\) means to add to \(G\) some isolated vertices with “External Connection Vertex” attribute equal \text{true}. After adding some isolated vertices to \(G\), if \(\text{ArcAddition}(F) = \text{true}\) then for each ordered pair of vertices \((v, w)\) in \(G\) such that \(v\) and \(w\) do not correspond to the same son of \(F\) and, \(v\) and \(w\) have External Connection Vertex attribute equal \text{true} respectively, then add to \(G\) an arc from \(v\) to \(w\) with probably 1/2. Then modify the “External Connection Vertex” attribute to each vertex of \(G\) as follows. If \(\text{ExternalConnection}(F) = \text{false}\) then set to \text{false} the “External Connection Vertex” attribute of each vertex of \(G\).

(6) returns \(G\). End.

5 Conclusions

The semantics of a language for formal description of digraph families has been presented, a graphic notation for a family has been introduced and an algorithm to obtain an instance of a family has been proposed. In order to study the suitability of such a language to the user needs, we have developed a
prototype of FDL. This first prototype has been written in C++ under Unix², X-Windows³ and Motif⁴ facilities. The prototype is being tested by digraph researchers to adapt it to user requirements.

References


²Unix is a registered trademark of Unix System Laboratories
³X-Windows is a registered trademark of the Massachusetts Institute of Technology
⁴Motif is a registered trademark of Open Software Foundation