2 Checking Algorithm

This algorithm determines whether a combination of (partial entity data types), G_p, conforms to a given EXPRESS schema. G_p might, for example, represent the set of partial entity instances appearing in an entity instance. For that entity instance to be valid, G_p must be valid under the algorithm presented here.

2.1 Construct the subsup graphs

The entities in an EXPRESS schema can be partitioned into disjoint supsub graphs, in which the nodes are entity data types and the edges are (directed) subtype relationships. A supsub graph, *G*, may be constructed as follows:

1. Choose any entity data type, *e*, that has not yet been assigned to a graph. Add entity *e* to the graph *G*.
2. For each entity *e* that is added to graph *G*, add all its immediate supertypes to the graph, deleting any duplicates;
3. For each entity *e* that is added to graph *G*, add all its immediate subtypes to the graph, deleting all duplicates.

Repeat (2) and (3) for each new entity that is added to the graph, until all entities in the graph have been processed.

The members of a supsub graph, *G*, form a set, say $G = \{a, b, c, \ldots n\}$. A supsub graph may consist of a single entity, in which case the entity will be neither a supertype nor a subtype.

2.2 Collect the constraints

This part of the algorithm collects all the constraints applicable to the entities in the supsub graph *G*.

1. For each entity *e* in *G* that is declared SUBTYPE OF (s_1 [, s_2, …, s_n]), create a logical expression of the form:
   NOT *e* OR (s_1 [AND s_2 … AND s_n])
   where s_1 , s_2, …, s_n are the entities appearing in the SUBTYPE clause.

   Note – if the graph *G* is created as a graph and not just a set, then s_1 , s_2, …, s_n are all the entities that are connected by edges *out of* the node *e*.

2. For each entity *e* in *G* that is declared ABSTRACT SUPERTYPE, create a logical expression of the form:
   NOT *e* OR (s_1 [OR s_2 … OR s_n])
   where s_1 , s_2, …, s_n are all the entities that are declared SUBTYPE OF (*e*) in the schema.

   Note – if the graph *G* is created as a graph and not just a set, then s_1 , s_2, …, s_n are all the entities that are connected by edges *into* the node *e*.

3. For each `ANDOR` operator in a subtype constraint expression, where
   - the operator is not part of a subexpression operand of a `ONEOF` expression or an `AND` expression, and
   - one of the operands of the `ANDOR` is an entity reference,
   delete the `ANDOR` operator and the entity reference.

   For example the following expression, where lowercase letters are entity references:
   ```
   a ANDOR b ANDOR ONEOF(c, d, e ANDOR f, g) ANDOR h ANDOR i
   ```
   reduces to
   ```
   ONEOF(c, d, e ANDOR f, g)
   ```

4. Identify the set of root entities in the graph, R_G, i.e. the set of entity data types that are not subtypes of any other entity.

   If R_G contains only one root entity, skip the rest of this step.

   If R_G contains more than one root entity, do the following.

   - Create the set of all distinct pairs of entities in R_G, $P_G = \{ (r_i, r_j) \mid j > i, r_i, r_j \text{ in } R_G \}$. To each such pair in P_G attach a logical expression, initialized to NOT (r_i AND r_j), where (r_i, r_j) is the pair.

     Note – the set P_G will have $n \bullet (n - 1) / 2$ members, where n is the number of root entities in R_G.

- Create the set S_G of all multiply inheriting subtypes in the graph G, i.e. the set of entity nodes that have more than one subtype edge out of them.
- For each member (subtype), s_m, in S_G :

  a. determine the set of all root entities, R_m that can be reached by following the SUBTYPE OF links (directed edge sequences) out of s_m through any number of edges to a root.

  b. If R_m has only one member, do nothing, and go on to the next member of S_G.

  c. If R_m has more than one member, create the set of all distinct pairs of entities in R_m, $P_m = \{ (r_i, r_j) \mid j > i, r_i, r_j \text{ in } R_m \}$.

  d. For each member of P_m, find the corresponding member (root entity pair) of P_G and add "OR s_m" to its attached logical expression. (This modifies the expression to allow that root entity combination when subtype s_m is present in the instance.)

  When all the multiply inheriting subtypes in S_G have been processed, the logical expressions attached to the members of P_G are complete.
- For each member of P_G, add its attached logical expression to the set of logical expressions representing the subtype constraints.

  These expressions are used to invalidate root combinations that are not allowed by the multiply inheriting subtypes.

5. For each entity e_i in the supsub graph G collect all constraint expressions that apply to it. Except for `TOTAL_OVER` constraints, a constraint expression applies to an entity e_i if the expression includes an entity reference e_i.

   For example, the constraint expression
   `ONEOF(c, d, e)`
   applies to entities *c*, *d* and *e* only; it does not apply to any other entities.

   A `TOTAL_OVER` constraint for an entity, say *e*, applies to all immediate subtypes of *e* that are *not* referenced within the expression.

   For example:
   `SUBTYPE_CONSTRAINT sc FOR e; TOTAL_OVER(b,c); END_SUBTYPE_CONSTRAINT;`
   will result in the `TOTAL_OVER` being applied to all the immediate subtypes of *e*, except for subtypes *b* and *c*.

2.3 Check the structure

This stage of the algorithm checks G_p against the EXPRESS requirement that entity data types that are not subtypes (the root nodes of the graphs) cannot be combined unless a specific subtype declaration joins their supsub graphs. Simply stated, a combination G_p is not valid unless it corresponds to exactly one graph G developed from the EXPRESS schema as specified in 2.1 above.

For a given combination G_p, choose any entity e in G_p. By the construction algorithm given in 2.1 entity e appears in exactly one supsub graph, G. G_p is tentatively valid if every entity in G_p is in G, and it is invalid if that is not the case.

If G_p is invalid, stop. If G_p is tentatively valid, continue.

2.4 Evaluate the constraints

At this point every entity in the supsub graph G has a list, possibly empty, of the applicable constraint expressions. This part of the algorithm checks the constraint expressions applicable to the entities in the set G_p.

Given a set of entities, G_p, and a corresponding set of constraint expressions then:

- A subexpression that consists only of an entity reference is TRUE if the entity is a member of G_p, otherwise it is FALSE.

- A `ONEOF` expression is TRUE if no more than one of its subexpressions is TRUE, otherwise it is false.
- An `ANDOR` expression is equivalent to a logical OR expression. It is TRUE if either of its operands is TRUE, otherwise it is FALSE.
- A `TOTAL_OVER` expression is equivalent to the logical expression formed by ORing its subexpressions. It is TRUE if one or more of its subexpressions are TRUE, otherwise it is FALSE.
- An `AND` expression is equivalent to a logical AND expression. It is TRUE if both its operands are TRUE, otherwise it is FALSE.

1. Collect the set of constraint expressions that apply to the entities in G_p.
2. Replace each entity reference in the set of constraint expressions by TRUE if the entity is a member of G_p, otherwise replace the entity reference by FALSE.
3. Treat the resulting set of constraint expressions as logical expressions and evaluate them.
4. If any expression evaluates to FALSE then G_p is invalid, otherwise G_p is valid.

That is the end of the checking algorithm.

3 Evaluated set algorithm

The checking algorithm may be used as a basis for an algorithm to determine the evaluated set of a supsub graph.

1. Calculate the power set $\boldsymbol{P}_g$ of the supsub graph G. This contains all possible entity combinations G_p available for G. For n entities $\boldsymbol{P}_g$ will have 2^{n-1} members. By definition, every G_p in $\boldsymbol{P}_g$ will meet the tentative validity criterion specified in 2.3.
2. For each G_p in $\boldsymbol{P}_g$, evaluate the constraints on G as specified in 2.4. If G_p is invalid, delete it from $\boldsymbol{P}_g$. This results in a reduced set, say $\boldsymbol{E}_g$.

   For efficiency purposes, the set elimination may be performed during the calculation of the power set.

   Depending on the exact structure of the supsub graph the size of the reduced set $\boldsymbol{E}_g$ will be somewhere between n and 2^{n-1}. If the graph is a binary tree, the size of the reduced set will be approximately proportional to 1.5^n.
3. The members of $\boldsymbol{E}_g$ form the evaluated set for G.