Logical deduction
in Gellish formalized natural languages

by

Andries van Renssen
Gellish.net

February 2013
revised August 2019

Table of content

1 Introduction

Gellish formalized natural languages, such as Gellish Formal English, are intended to be suitable for expressing information, knowledge and requirements as well as queries and answers on those queries in a computer interpretable way. Those formal languages are defined primarily by defining the syntax for expressions and secondly by specifying a semantic content in the form of a collection of expressions in the formal language that define concepts and that specify generally valid knowledge about kinds of relations and the roles of concepts in those kinds of relations.

Information etc. can be expressed explicitly in Gellish expressions, but in addition to directly expressed information, there is also implied information which is information that is not directly expressed, but is derived by reasoning on the basis of the information that is directly expressed. Such logical deduction applies the rules of logic. Such rules are a generally valid part of the language definition and thus they are also included in the definition of Gellish formalized languages. This enables that software, such as the Gellish Communicator application, can answer questions for which the answers (conclusions) are inferred by derivation through reasoning on the basis of data in a given database. In Gellish the logic and rules are separate from the reasoning process. Where necessary, the rules are stored in Gellish formal language expressions, whereas the reasoning process is implemented in software.

This document is intended to provide guidance on the modelling of rules as data and guidance for the development of software that implements the reasoning processes. Thus this document explains how rules can be expressed in Gellish and how software can apply those rules. The Gellish Communicator application software demonstrates how most of these deduction processes can be implemented.

2 Implications (if-then)

A reasoning process uses database content (one or more expressions of information) as premises to come to conclusions. In other words expressions may have implications that can be expressed by other expressions.

An implication is a consequence that follows from a state (an expression of a state of affairs or premise).

There are implications for different kinds of states as discussed in the following paragraphs.

2.1 Basic logic implications

The definition of Gellish formalized languages (the language defining ontology) include the definition of logical binary relations. It also includes expressions that specify which kinds of relation are subtypes of the basic logic relations. Those basic logic kinds of relations and their implications are described in the following paragraphs.

2.1.1 Transitive relations

For a transitive relation it holds that: if A is related by a transitive relation of a particular kind to B and B is related by a relation of the same kind to C, then it has as implication that A is related by a relation of the same kind to C.
Whether a relation is transitive can be determined on basically either one of two criteria:

- being a sameness relation in some respect. Such a kind of relation expresses that something is the same as something else in some respect.
  For example, has the same hair colour as, is of the same size as, is a member of the same football club as, etc.
- being an ordering relation on the basis of a particular criterion, such as a comparative relation in size, extent or intensity. Such a kind of relation introduces some kind of order into a set.

For example: is bigger than, is smaller than, is longer than, is darker than, but also: is located in, is ancestor of, etc. Exceptions to this rule are for example: is a father of, because a relation between father of a father of A is a grandfather of A and not a father of A.

For example if a database contains two assertions that state that A is bigger than B and that B is bigger than C. If a query asks whether A is bigger than C, then it can confirm that that is true.

2.1.2 Intransitive relations

An intransitive relation is a binary relation that is not a transitive relation. Thus it does not have the above described implication.

2.1.3 Symmetric relations

A symmetric relation is a binary relation for which holds that: if A is related to B by a symmetric relation of a particular kind in which A has a particular role, then it has as implication that B is related to A by the same kind of relation and in which B has a role of the same kind.
For example: is a colleague of, is a sibling of. Because if A is a colleague of B, then A has a role as colleague and B has also a role as colleague in that relation. Similarly with siblings. Note that in Gellish the two kinds of roles are distinguished whereas the specification of the symmetry declares them to be exchangeable.

2.1.4 Antisymmetric relations

An antisymmetric binary relation is a relation that is not a symmetric relation. In other words, the two related things have roles of different kinds in the relation. For example, a husband and wife marriage relation is an antisymmetric relation because the two related parties in such a relation have different roles in their relation.

2.1.5 Reflexive relations

A reflexive relation is a binary relation that specifies that if something plays the first role in such a relation then it has as implication that the same thing is also the player of the second role in that relation. In other words, it is a relation between an object and itself.
Note that an alias relation (or synonym relation) in Gellish is not a relation between something and itself, because in Gellish it is defined as a relation that relates a relation between something and a name or code to another relation between the same thing and another name or code. Therefore it is not a reflexive relation.

2.1.6 Irreflexive relations

An irreflexive relation is a binary relation in which the two roles are by definition played by different things. It has as implication that the second role is play by another thing than the first role.

2.1.7 Disjointness relations

A disjointness relation is a binary relation between two kinds that specifies that if something is classified to be a member of one of the related kinds then it has as implication that it can and may not be classified by the other kind. In other words, there can be no individual thing that can be correctly classified by both specified related kinds at the same time or are not allowed to be element of the both specified related collections at the same time. The criteria for membership of both kinds exclude each other or the defintion of the nature of the collections does not allow that something is an element of both collections at the same time.

2.2 Implications for subtypes

If something is by definition the case for all things of a particular kind, then it is also the case for things that are classified by a subtype of that kind. In other words kinds inherit all the characteristics and other things that are the case for their supertype kinds.

In Gellish all kinds included in a taxonomic dictionary. Thus all concepts are arranged in a taxonomy. Thus all kind (concepts) are explicitly defined as being a subtype of one or more supertype kinds. Thus if something is classified as a member of a kind, then that implies that it has characteristics, and knowledge that hold for that kind and it also implies that it has characteristics etc. that are inherited from the supertype(s) of that kind.

For example, if kind A is a subtype of kind B and something holds for individual things that are of kind B, then this implies that it also holds for things that are of kind A.
In other words, if kind A is a subtype of kind B and some individual thing is classified as being of kind A, then this implies that it is also of kind B.

2.3 Implications of possibilities

Expressions of generic possibilities express what can be the case. Such expressions are also called existentially quantified expressions, because they specify that there (may) *exist* (= the existential quantifier) some (not all) occasions for which the stated expression is true.

Assume that a Gellish database contains statements about knowledge, being possibilities for things of particular kinds, within cardinality constraints. For example:

- assertion car <can have as part a> wheel (1)

This is a quasi-unconditional statement. However it has a meaning that reveals that it actually is a conditional statement. This is shown by the following interpretation:

> *if there exists an X that is classified as a car, then X can have as part a Y that is classified as a wheel.*

Thus the condition is that there exists an individual thing that is a car.

The language defining database (the Gellish ontology) also contains relations that specify which kind of relation can be a realization or fulfilment of the possibility relation (see the extended language definition section). For example:

- assertion <can have as part a> <can have as realization a> <has as part> (2)

Finally, assume that the database also contains statements about individual things. For example as follows:

- statement C1 <is classified as a> car (3)
- statement W1 <is classified as a> wheel (4)

Then the premises (1) (2) (3) and (4) have as implication the derived implied possibility that:

- possibility C1 <has a part that is classified as a> wheel

and that it is an implied possibility that

- possibility C1 <has as part> W1 (5)

although it is also possible that:

- C1 has not W1 as its part, or
- C1 has (also) another wheel as its part, or
- C1 has no part that is classified as a wheel.

2.4 Implications of requirements

Requirements are a special case of possibilities. Requirements specify that a particular party requires that something shall be the case.

Assume that a Gellish database contains statements about requirements, being what shall be the case for all things of particular kinds, within cardinality constraints and assuming that the validity context is applicable. For example:

- statement (each) car <shall have as part a> wheel (6)

Then, because C1 is classified as a car, the premise (6) and (3) imply that there is a derived implied requirement for C1 that states that

- requirement C1 <has a part that is classified as a> wheel (7)

Assume that the database also contains the statement:

- statement C1 <has as part> W1 (8)

Then from (8), (3) and (4) it can be concluded that for C1 the requirement (6) is satisfied by (8). This implied conclusion can be explicitly expressed as follows:

- statement (8) <is a fulfilment of> (6) (9)

If the cardinality constraints specify a requirement for more than one part, then the requirement (6) is only partially fulfilled. Then only a collection of statements can be a fulfilment of (6).

2.5 Implications of definitions

An expression of a definition is an expression of what is by definition the case for all things of particular kinds, within cardinality constraints. These kinds of expressions are similar to expressions of what is always the case for well-formed things (things that are 'on-spec') of particular kinds. Both kinds of expressions are sometimes called universally quantified expressions or *universally quantified rules*, because they are valid *for all* (= the universal quantifier) things of the kind about which the expressions defines something.

Assume that a Gellish database contains statements about definitions. For example:

- assertion car <has by definition as part a> wheel (10)

An expression of a relation between kinds which kind of relation is denoted by a phrase that start with <has by definition..> means that eavh individual thing of the specified kind has something by definition. More precisely it has (by definition) a realiization of such a relation with an individual thing of the related kind. Thus statement (10) states that each car <has a part> and that part <is a> wheel. Therefore, the premises (10) and (3) have as implied consequence conclusion that

- statement C1 <has a part that is classified as a> wheel (11)

If a statement such as (11) appears in a database we can conclude that (11) is compliant with the definition in statement (10)

From statements (10), (8), (3) and (4) together we can draw the compliancy conclusion:

- assertion (8) <is compliant with> (10).

Often definitions are regarding facts that specify that things of a particular kind have by definition a particular qualitative aspect. In such cases the qualitative aspect is often directly related to its possessor, without the existence of a statement such as (8). This analogous to statement (11), but replacing 'wheel' by a qualitative aspect, such as 'red'. For example:

- assertion person <is by definition> mortal (12)
- statement Socrates <is classified as a> person (13)

To draw a conclusion, the database should also contain statements such as the following:

- assertion <is by definition> <can have as realization a> <is> (14)

According to the Modus Ponens reasoning the premises (12), (13) and (14) have as implied consequence:

- conclusion Socrates <is> mortal (15)

In other words, (15) is compliant with definition (12). Note that (15) is comparable with (11) and not with (8).

2.6 Implications of short-cut relations

The difference between the above two ways of compliancy is caused by the use of a short-cut relation in (12). The short-cut relation does not make explicit that the qualitative aspect mortal is a qualification of mortality and that (12) implies that a person <by definition has as aspect a> mortality. Thus, statement (12) is a short-cut for the two definition statements:

- statement person <has by definition as aspect a> mortality (16)
- statement mortality of a person <is by definition qualified as> mortal (17)

and statement (13) is a short-cut for the statements:

- statement Socrates <has as aspect> mortality of Socrates (18)
- statement mortality of Socrates <is qualified as> mortal (19)

Thus in fact (18) is compliant with definition (16) and (19) is compliant with definition (17). Together (18) and (19) are therefore compliant with the short-cut definition (12).

2.7 Implications of individual states

An individual state is a state of an individual thing. For example, the state of John Doe.

There may also be a state of (elements in) an individual collection of things. For example, the state of a particular collection, population or stock.

Kinds of relations for expressions about the latter are denoted by phrases that start with:

- <each of which is ...>

For example, a database may contain the following statement:

stamp collection A <each of which is classified as a> stamp (20)

This has as consequence that if S is an element of A, then it can be concluded from (20) that S is a stamp.

2.8 Implications of negation

A negation of a relation is denoted in Gellish by an intention with the value denial.

For example:

- denial valve position P <is qualified as> open

The implication of a denial depends on the alternatives for what is denied. In the above example, the values are either open or closed. Thus the denial has as implication:

- assertion valve position P <is qualified as> closed

2.9 Relation chains

The language definition also includes kinds of relations that are defined as relation chains, whereas it is specified which series of relations of specific kinds define the chain. For example an ancestor relation is defined as a (subtype of) relation chain and is further defined as a chain of one or more parent-child relations. This means that if a database contains two relations that state that A is a child of B and that B is a child of C, then software can conclude that C is an ancestor of A.

This can be applied in queries. For example assume the following query is given:

who is an offspring of C

then software will determine that the kind of relation is an inverse of the <in an ancestor> relation, which is a subtype of relation chain. Thus it can find the B as well as A.

Note that if the database contains assertions such as A is a son of B and B is a daughter of C, then the software will find from the language definition that those kinds of relations are subtypes of the <is a child of> kind of relation. Therefor it will also find B and A in that case.

3 Rules

Rules for logical deduction are a special kind of knowledge or requirements. Rules explicitly specify what implications, conclusions or consequences can or should be drawn, given particular conditions (premises). Thus a basic rule relates conditions to a consequence in the following way: If all conditions in a rule are simultaneously true, then the rules states that the conclusion is also true. If at least one of the conditions is false then the consequence is false.

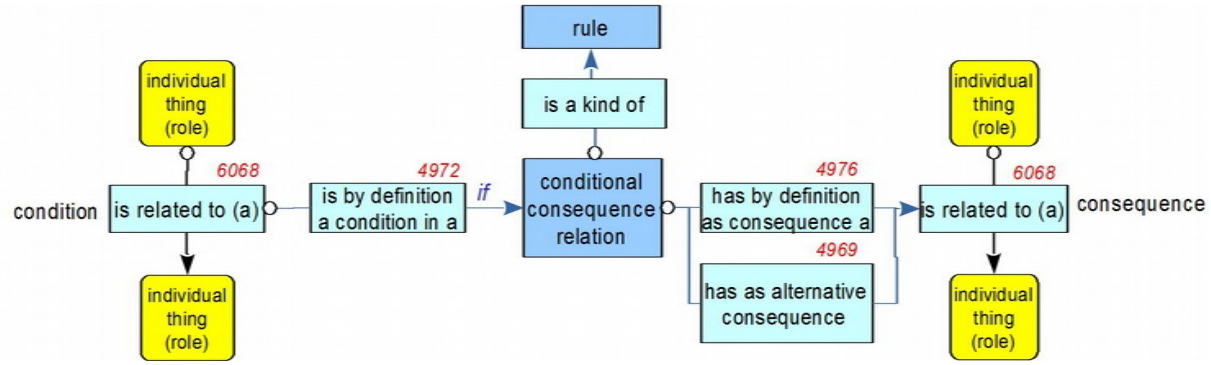
Expressions can express what is the case, but they can also express something valuable, although we don't know whether it is the case or not. Assertions belong to the first category. Assertions are expressions about things that are the case (or not the case). Thus assertions have a 'truth value', being either true or false. Examples of assertions are:

assertion John <is the father of> Bill

assertion man <can be a father of a> person

Both expressions are (assumed to be) true, although the second assertion does not hold for every combination of a man and a person.

Conditions (or premisses) and consequences (of fulfilled conditions) are neither true nor false. They do not express something that is the case, but something that might be the case. Thus they have no 'truth value', or we might say that their truth value is by definition undecided. Conditions and consequences are therefore to be distinguished from assertions. In Gellish conditions and consequences are expressions (relations) that are distinguished by their 'intention' being respectively 'condition' and 'consequence'.
Rules relate conditions to consequences, as is illustrated in the following figure:



Thus a rule can be expressed as follows:

assertion	rule-1 <has by definition as condition a> (1)	
assertion	rule-1 <has by definition as condition a> (2)	
assertion	rule-1 <has by definition as conditional consequence a> (3)	
condition	grandparent <is a parent of> parent	(1)
condition	parent <is a parent of> grandchild	(2)
consequence	grandparent <is a grandparent of> grandchild	(3)

The first condition is to be interpreted as: 'if a person who has a role as grandparent is a parent of a person who has a role as parent'. The second condition has as interpretation: 'if the person who has a role as parent (and who is previously mentioned in this rule) is a parent of a person who has a role as grandchild. And the consequence is to be interpreted as: 'then the person who has a role as grandparent (and who is previously mentioned in this rule) is the grandparent of the person who has a role as grandchild (and who is previously mentioned in this rule)'.

Note that the conditional and consequential expressions have no truth value, whereas the rule will be applied on assertions and if they satisfy the conditions then the reasoning on the basis of the rule will create assertions as conclusion.

The above rule is expressed in Gellish expression format as follows:

UID of left hand object	Name of left hand object	UID of idea	Name of relation type	UID of right hand object	Name of right hand object
			is a grandparent of		
logic:1	rule for grandparent relation	logic:101	is a qualitative kind of	4.959	if-then-else relation
logic:1	rule for grandparent relation	logic:102	has by definition as conditional consequence a	5.970	is a grandparent of
6.296	grandparent	logic:103	is a parent of	5.285	parent
5.285	parent	logic:104	is a parent of	6.123	grandchild
logic:1	rule for grandparent relation	logic:105	has by definition as condition a	logic:103	grandparent-parent relation
logic:1	rule for grandparent relation	logic:106	has by definition as condition a	logic:104	parent-grandchild relation

3.1 Conjunction (and)

A conjunction is a combination of (expressions of) states (facts, assertions, conditions) that together have or by evaluation may produce a 'truth value'. If a conjunction has a truth value of 'true', then the combination is true if and only if all the conjoint components are true. The combination is false if at least one of the conjoint components is false. The combination is probable (with a particular degree of probability) if at least one of the conjoint components is merely probable, and none is false.

Conjunctions are especially applicable in conditions and consequences in conditional consequence relations (rules).

In Gellish this is applied if multiple statements are all mentioned as conditions or consequences of the same conditional consequence relation. The fact that multiple conditions are mentioned for one conditional consequence relation implies that all those conditions have to be satisfied in order to conclude that the conjunction is satisfied.

For example:

- A <is a condition in> R
- B <is a condition in> R

These two conditions shall both be satisfied so that the conjunction is satisfied in order to draw the consequence on satisfied condition(s).

Similarly, the fact that multiple consequences on satisfied conditions are mentioned for the same conditional relation implies that all mentioned consequences are applicable. Such combinations of consequences do not necessarily have a truth value. It may be that they are both executed and thus become realized

- D <is a consequence on satisfied condition in> R
- E <is a consequence on satisfied condition in> R

In other words:

- D <follows from> R and E <follows from> R

Also other relations between expressions are called conjunctions, such as part-whole relations between occurrences (activities), temporal relations and causal relations (The Trivium, page 110).

3.2 Disjunction (or)

A disjunction is a combination of states (facts) that can be applicable or true if only one of the components is applicable or true. A disjunction is also called an alternation.

There are two kinds of disjunction: inclusive disjunction and exclusive disjunction.

The inclusive disjunction (either A or B or both (A and B)) is modelled in Gellish as follows:

- A <is a condition in> R
- B <is an alternative condition in> R

The exclusive disjunction (either A or B but not both (A and B)) can be modelled by specifying:

- A <is an alternative condition in> R
- B <is an alternative condition in> R

3.3 Else (if not)

If a condition is not satisfied it may be that another consequence follows. This can be specified in a separate if-then relation by specifying an inverse condition. However it is also possible to record such a consequence in the same if-then-else relation as follows:

- F <is a consequence on unsatisfied condition in> R

4 Biconditional relations (iff)

A biconditional relation is a binary relation between two ideas in which one is the condition and the other is the consequence, whereas both have the same 'truth value'. This means that for p and q holds that 'if p then q' as well as 'if q then p'. In such a case p is called a necessary and sufficient condition for q and vice versa. It is also called an 'if and only if' or an iff condition.

Biconditional relations are expressed in Gellish by relations between two ideas that are of a kind that is denoted by the phrase <is a bidirectional of>.

Each kind of relation in Gellish is denoted by either a base phrase or an inverse phrase. Thus both phrases denote the same kind of relation. For example the assertion 'John is a son of Jim' and its inverse expression 'Jim is the father of John' are two expressions of the same idea and the expressions use phrases that denote the same kind of relation, being a father-son relation. This implies that the two expressions are related because of the fact that they are expressions of the same idea. That relation could be made explicit as a binary relation that could be called a biconditional relation between the two expressions of the same idea. Thus such biconditional relations are implied already and thus such explicit relations would be superfluous.

5 Firing of rules

Software may execute queries that trigger the search for ideas about which rules are applicable that have conditions that are satisfied. If all conditions for the firing of a rule are satisfied, then the software may fire the rule. Firing a rule means that the implication(s) (action(s) or conclusion(s)) are taken. The inferred idea(s) may be reported and/or added to the database. Software may even check whole database contents on ideas that are satisfied conditions in rules and verify whether the consequences are drawn and are consistent with possible other ideas.

What will happen when a rule fires?

The process works as follows:

1. **Search for potentially firing rules**

   Select an individual thing that is classified by a kind.
   Verify whether the kind appears in a condition of a rule.
   If yes, verify whether the rule is satisfied.
   If yes, verify whether the same rule has other conditions and verify whether they are also satisfied.
   If yes, *potentially fire* the rule for that individual (unless already done) by determining the consequence.
   If not, verify whether an action is specified on an unsatisfied condition.
   If yes, potentially fire the rule by determining the alternative consequence.

2. If several rules *potentially fire*, they are all in a 'conflict set'.
3. If several rules are in a conflict set, a rule is chosen to fire based on a predefined conflict strategy. The conflict strategy may mandate choosing a rule that has a higher priority, or that has been used more recently, for example. (Section 6.2 of the RIF Primer gives some examples of how priorities may be used.)
4. **Firing a rule**

   If a rule of the form *if C then A* fires, then the action *A* is carried out or the consequence A is drawn.

   This causes the creation of either of the following expressions:

   - A derived possibility
   - A derived requirement
   - A fulfillment conclusion
   - A consequence conclusion
   - A compliancy conclusion

5. This cycle repeats until there are no more rules that can fire; that is, a fixpoint is reached.

For example, assume that ...

The resulting conclusions should be reported to a user who should decide what happens with the resulting expressions. They may be added to a database, sent to other parties, etc.

6 Incompatibility constraints

An incompatibility declaration is a statement that specifies an object cannot have two relations of particular kinds at the same time. If the roles that the object plays are of the same kind, then the constraint is specified by a cardinality constraint that specifies the allowed number of simultaneous roles of that kind. If the roles are of different kinds, thus typically the kinds of relations are different, then such a constraint is specified by a relation such as <is incompatible with a>. Thus when a

relation is of a kind that has a compatibility constraint on one of its roles, then this implies that there may not be another relation of the another kind in which it plays a role of the specified kind.