SEMANTIC DATA TYPES

Mykola Rabchevskiy

In AGI sections: CAUSALITY and STRUCTURING THE OBSERVABLE algorithms for detecting cause-and-effect relationships and constructing a structured model of the situation were considered. In both cases, the process is based on making hypotheses and their subsequent verification, and the essence of hypothesizing is reduced to ***constructing an invariant function***. Thus, solving these problems depends on the ability to build potential invariants.

A potential invariant is a function whose arguments are the attributes of several objects from the situation description. When detecting structured objects, the invariant is a ***numerical function*** with a constant value during the observation time of the corresponding object. In the case of searching for a causal relationship between a situation and an event, an invariant is a ***Boolean function*** that always takes the value ***false*** in a situation that does not involve an event and ***true*** in a situation where the event always or with some probability takes place.

Designing functions is reduced to ***combining elementary functions*** into a complex ***hierarchical*** function. The role of elementary functions is played by arithmetic operators (addition, subtraction, multiplication, division), logical functions (comparison of numerical values, logical AND and OR), Euclidean distance calculations, trigonometric functions, and so on.

Obviously, the complexity of the combination and the number of possible combinations are not limited. Several circumstances help to reduce this diversity and thereby simplify the enumeration task.

First, some of the constructed expressions will turn out to be ***constants***, the value of which does not depend on the parameter values (like, for example, ***a < a***). Of course, they should be excluded from the set of potential invariants. Secondly, ***several expressions of different***

structures can represent the same function; they can have the same values with the same arguments.

In particular, many operators/functions do not depend on the ***order of the arguments*** (addition, multiplication, division, etc.). You should choose one from the set of possible expressions that differ only in the order of the arguments (for example, 6 ways to write the product of three factors).

This, however, does not exhaust the possibilities of reducing the volume of enumeration. The most substantial restrictive requirement is the ***meaningfulness of using a particular function/operation, depending on the meaning of the arguments***. Since the arguments are ***attributes of the current situation***, each of the arguments has a ***well-defined meaning*** - coordinate, size, speed, etc. Ignoring the meaning of quantities leads to expressions that do not carry meaning - such as the sum of temperature with a coordinate. We know how valuable the ***dimensions of the quantities*** included in the expressions from the school physics course are. There is, however, an even stronger notion of ***semantic type*** than dimension. This is because a ***suitable combination of dimensions of the operation arguments is not yet a guarantee of the meaningfulness*** of the result; the ***same dimension can have values that are different in meaning*** and therefore not interchangeable in the expression. For example, the Euclidean distance on a plane is calculated as

distance = sqrt(x^2 + y^2)

The dimensions of the ***x*** and ***y*** coordinates are the same, but ***they are not logically interchangeable*** - the distance expression must use ***two different coordinates***; respectively, the two coordinates have a different semantic type, "***X distance***" and "***Y distance***". Different in meaning are also ***absolute coordinates*** and ***coordinate differences***, **time point** and ***duration***, etc.

All ***elementary operations*** are provided with information about which combinations of ***semantic types of arguments*** are allowed and the ***semantic type of the result, depending on the types of arguments***. To use this to exclude expressions that are obviously meaningless to use as invariants, all values used are of the appropriate semantic type. This radically reduces the number of potential invariants.

A natural question arises: how does the use of ***semantic types*** fit in with the requirement of AGI ***generality***? To answer this question, consider a situation where AGI controls some real system. This system is designed to perform a range of missions in an appropriate operating ***environment***. The environment, in turn, determines the composition of ***sensors and actuators***. The design of effectors is based on those ***geometric, physical, and other principles dictated by the environment***. The ***semantic type of the measured values is already explicitly or implicitly represented*** in the system design. Information about the

sensors and actuators used is a natural element of ***setting up*** AGI to control the corresponding system, which does not affect the generality of AGI. Semantic types are not rigidly built-in elements of algorithms - they are part of the ***knowledge about the environment and the managed system***, technically represented ***not by code but by data***. Potential ***invariants are generated following the semantic data types coming from sensors and effector***s.

The number of invariants, which depends on how large the acceptable depth of the hierarchy of expressions of potential invariants, is limited not by the logical possibilities of their generation, but by ***how extensive the available computing resources*** of a particular system are - they should allow ***testing of the potential invariants in real-time***.

The technique for constructing a system of potential invariants is demonstrated by the program available here:

https://github.com/mrabchevskiy/invariants/

For those who are not going to experiment with the code, just look at the ***file with the results*** generated by the program (presented in the repository).

Technological details

To generate functions, it is convenient to use ***postfix*** (parenthesis-free) ***notation*** of expressions, which is ***concatenative***. The expression for a function formed by applying some operation to two `child` functions is reduced to the concatenation of the postfix entries of the expression for the left subfunction, the right subfunction, and the operator. The conversion of a postfix notation into a traditional ***infix*** notation is performed according to the same rules as the calculation of the value of an expression for the given values of the arguments.

A series of calculations are made with random arguments ' values ***to check that the constructed function is not a constant expression***. While not rigorous in nature, this approach is easy to implement, and many random trials make error unlikely. The exact process makes it possible to establish whether ***two constructed functions represent two variants of expression the same function*** in essence.

Subscribe to AGI engineering

By Mykola Rabchevskiy · Launched 2 years ago

AGI: fundamentals, architecture, implementation, source code