Argument from AI summary: How does asemantic computing differ from traditional distributed computing?

Marius Buliga

Institute of Mathematics, Romanian Academy
P.O. BOX 1-764, RO 014700
Bucureşti, Romania
Marius.Buliga@imar.ro , mbuliga@protonmail.ch

20.10.2023

Abstract

We explain what is an argument from AI summary [7] and what is asemantic computing [1]. As an example of such an argument from AI summary we provide the answer given for the question in the title by the LLM from [5].

1 Introduction

An argument from AI summary [7] aims to establish if the written communication of an idea is defective or not, by using an AI tool as a mirror which reflects back the message sent by the human author of the idea. In order to construct such an argument the human author asks an AI to provide a summary of the thesis shared by the author. The summary is then checked by the author. In the case when the author confirms the summary, then the conversation with the AI can be used as proof that the communication of the idea is conformal with the intentions of the author.

With such a proof, the author may argue that if (even) the AI gets the intended message, then probably the message is clear enough for other human readers.

Mind that an argument from AI summary does not suppose that there is an a priori correct meaning of the message, nor that the AI tool has any understanding of the message. The human author inputs the message into the AI tool then verifies that the output reflects the author's intentions. In the case when the author does not think that the AI tool output is conformal with the author's intentions, then such an argument from AI summary indicates that the AI tool is not up to the task or that the author should work more on the message.

In the following is given the AI [5] answer to the question from the title and as proof the archived answer [6]. As the author of [1], I confirm that the AI answer uses my article in a way which corresponds well with what I intended to communicate in [1].

2 Asemantic computing

As explained in Molecular computers which are based on graph rewriting systems like chemlambda, chemSKI or Interaction Combinators, the repository which contains also the article [1], graph rewriting systems are a very promising direction for building decentralized, distributed computing systems, as well as an inspiration for real life molecular computing. Among such graph rewriting systems we list: Lafont Interaction Combinators [2], chemlambda [3], chemSKI [4]. Computing with such systems is local in space and time and therefore there is no need for a global semantics, from the point of view of computation.

Classically there is a 3 stages process which uses graph rewriting. We want to solve a problem, therefore we start from a program, or more generally from a particular class of term rewrite systems (such as for example lambda calculus, but not combinatory logic).

Meaning to structure: The program is then compiled to a graph. This can also be seen as a language to structure problem. The language can be a term rewrite system, the program can be a term, the structure can be an abstract syntax tree. Many other examples exist, which can be put into the form of transforming something which has a meaning (for a human), to something which can be processed by a machine (in this case a graph).

Structure to structure: transform the initial graph into a final graph, by using the graph rewriting system and an algorithm for the choice and order of application of the rewrites. Usually this problem is complicated because we want to have a good translation not only of the meaning to structure, but also of the term reduction into graph reduction. For example, when we write a program we expect it to "do" something, and we would like that the execution of the program by the decentralized machine (ie the structure to structure part) to be a translation of what we imagine the program is doing. As an example, if our program is a lambda term, which is compiled into a graph, we would like the graph rewriting to be compatible with the term rewriting, ie the beta reduction term rewriting rule. These semantic constraints which are put in the use of graph rewriting are incompatible with decentralized computing.

Structure to meaning: transform the final graph into the language, in order to obtain the answer of the problem. For example, if the initial program was a lambda term, we compile it into a graph, then we reduce the graph, then we translate back the final graph into a lambda term. In other examples we just want to translate the final graph into something which has a meaning for us, into the solution of the problem.

In all such applications of graph rewriting, we position at the level of meaning, we descend at the level of structure, where we use graph rewriting as a tool and then we translate back the result into a meaningful output.

In contrast with the classical approach, it is argued that distributed computation with graph rewrite systems should skip the initial and final global semantics conversion. Instead, we should regard as the natural level of computation the one called "structure to structure". The design of the initial structure (if any, or if it makes sense to think about an initial global graph) could use the step called "meaning to structure" only as a tool which serves the structure to structure goal. Likewise, the extraction of meaning from the main computation part can be only based on local in space and time information. Finally, the graph rewriting algorithm should be as well local in space and time. Such a computation style which sits mainly at the local graph rewrite level is called "asemantic computing" in [1].

3 The answer to the question

The conversation with the AI was started by the demand to summarize [1]. Then the AI suggested the question from the title. In the following is reproduced the answer to this question. The whole conversation is archived at [6].

The edits of the answer are only the replacement of the text "geeksforgeeks.org" with [1] for the links provided in the AI answer (only the links name are changed, all links point to [1]).

How does asemantic computing differ from traditional distributed computing?

Answer | Phind V6 Model

Asemantic computing differs from traditional distributed computing in several key ways:

1. Global Control and Meaning: Traditional distributed computing systems require a global control and meaning. This is because they are designed to perform tasks that require a global understanding of the system's state. In contrast, asemantic computing systems do not require a global control or meaning. They operate locally in time and space, without needing to understand the overall state of the system [1].

2. Semantics: Asemantic computing does not have semantics, which are the meanings of symbols in a system. This is a significant difference from traditional distributed computing systems, which rely on semantics to understand and interpret the data. In asemantic computing, the data is processed without any inherent meaning [1].

3. Turing Machines and Graph Rewrite Systems: Asemantic computing uses Turing machines and graph rewrite systems, which are local and do not require global control or meaning. In contrast, traditional distributed computing systems often use term rewrite systems, which are non-local and require global control [1].

4. Confluence: Asemantic computing does not require confluence, which is the property of a graph rewrite system where if we have a graph that can be rewritten into one without any further possible rewrites, then this final state is unique. This is undesirable for life-like distributed computations, where final states are to be interpreted as death and they have to be recycled somehow by another mechanism [1].

5. Extracting Meaning: The meaning we can extract from an asemantic computing system is only local. In traditional distributed computing systems, we can extract global measures of agreement and synchronization. However, in asemantic computing, we cannot extract precise global measures of agreement or synchronization [1].

6. Programming and Usage: Asemantic computing systems are unclear how to program or use them in the old ways. However, they can be used as if they are a living ecosystem, an extension of the meatspace [1].

References

[1] M. Buliga, Asemantic computing, in: chemlambda. (2022). chemlambda/molecular: Molecular computers which are based on graph rewriting systems like chemlambda, chemSKI or Interaction Combinators (v1.0.0). Zenodo. https://doi.org/10.5281/zenodo.7306917 , [github source]

[2] Y. Lafont, Interaction Combinators, *Information and Computation* **137**, 69–101, (1997)

[3] M. Buliga, Artificial life properties of directed interaction combinators vs. chemlambda, arXiv:2005.06060 (2020) https://mbuliga.github.io/quinegraphs/ic-vs-chem.html#icvschem

[4] M. Buliga, chemSKI with tokens: world building and economy in the SKI universe, arXiv:2306.00938, (2023)

[5] Phind, www.phind.com

[6] Archive of Phind answers, https://archive.ph/DUsn8

[7] M. Buliga, Argument from AI summary, (2023)
chorasimilarity.wordpress.com/2023/10/01/argument-from-ai-summary/