A Similarity Criterion For Sequential Programs Using Partial Functions - Turing-machine reducibility and Categorization

Abhinav Aggarwal and Padam Kumar

Indian Institute of Technology Roorkee

Some more properties

The representation of $\mathcal{L}_{C_j}^{P_j}$ as a composition of three functions, or equivalently, concatenation of three programs, may not be unique. All we have to do is find some representation in which the required sub-structure isomorphism can be established.

1. A mechansim to enumerate all programs that are type-1 isomorphic to some given program
2. The commutative diagram, seen as a directed graph, $\mathbb{G}$, contains connected components that are sets of mutually type-1 isomorphic programs.
3. A special component - one containing **infinite loops**. We denote this cluster by $\mathbb{H}_1$. Any program, $\mathcal{L}_C^P$ belonging to $\mathbb{H}_1$, must be sub-structure transformationally isomorphic to an infinite loop, $\mathcal{L}_{True}^{id}$, which means that it must contain some non-terminating component.
4. The only programs in $\mathbb{G}_1 \backslash \mathbb{H}_1$ that do not halt are the **ones that enter the undefined state some time during their execution** and then cease to come out of this state, rendering the machine perform bogus computations forever. The program corresponding to $id_\perp$ must be type-1 isomorphic to all such *outliers* of $\mathbb{G}_1 \backslash \mathbb{H}_1$, and hence, it forms yet another connected component in $\mathbb{G}_1$, say $\mathbb{K}_1$.
5. The programs in $((\mathbb{G}_1 \backslash \mathbb{H}_1) \backslash \mathbb{K}_1)$ are **guaranteed to halt in finite time**.
6. Turing machine reducibility, in general, is a direct consequence of the existence of a type-1 arrows. However, the inverse may not be true.
7. Non-recursive enumerability of establishing the existence of type-1 arrows.

Categorization

Many possible ways to form categories of sequential programs:

1. **Objects** - Blocks in the control flow diagram
   Arrows - Directed links connecting these blocks and depicting control flow
   Identity Arrow - Self loops over blocks
   Composition of arrows - Concatenation of blocks in order along a path
   Every diagram represents a complete computer program, for which the topological alignment of objects and arrows provides the corresponding control-flow
   All the sub-diagrams of a given diagram correspond to programs that are type-1 isomorphic to the program for the original diagram. This categorization fails to capture type-2 isomorphism, and in most cases, type-0 isomorphism as well.
2. **Objects :** State assignments
   Arrows : Computer programs
   In such a model, we say that **there exists an arrow between two objects if there exists a computable function which will transform the state assignment of object-1 into the state assignment of object-2.**
3. Product category, C^k
   Objects : Collections of k-state assignments
   Arrows L Computable bijective functions, mapping each of these k-assignments to its corresponding assignment in the other object
   The different arrows that exist between the same pair of objects represent programs that are type-0 reducible to each other.

Type-2 arrows - Turing-machine Reducibility

Let $P_i, P_j \in \mathcal{P}$ be two computer programs, and, TM_i and TM_j be two Turing machines that accept languages $L(P_i)$ and $L(P_j)$, respectively. Then there exists an **arrow of type-2**, $A_{i,j}^2 \in \mathcal{A}_{i,j}^2$ between P_i and P_j, denoted by $P_i \xrightarrow{A_{i,j}^2} P_j$, if the language $L(P_i)$ is Turing reducible to language $L(P_j)$, i.e. $L(P_i)$ is decidable relative to $L(P_j)$.

$$P_i : \mathcal{S}_{i,\perp_i} \rightarrow \mathcal{S}_{i,\perp_i}$$

$$\Big| \; A_{i,j}^2$$

$$P_j : \mathcal{S}_{j,\perp_j} \rightarrow \mathcal{S}_{j,\perp_j}$$

Figure 1: Diagrammatic representation of type-2 arrow

Properties of type-2 arrows

1. Turing machines corresponding to P_i and P_j to **accept at least** $L(P_i)$ **and** $L(P_j)$, respectively.
2. We do not require $L(TM_i)$ to be Turing reducible to $L(TM_j)$ for a type-2 arrow to exist.
3. The true nature of P_i and P_j is not necessarily simulated in the exact sense these programs are coded, but possibly in some other way such that **the transformations these programs provide to their input state assignments is replicated by the Turing machines on the same set of inputs**.

References

[1] Maarten M. Fokkinga, *A Gentle Introduction to Category Theory.* University of Twente library, 1992.

[2] Samson Abramsky, Achin Jung, *Domain Theory : Corrected and expanded version.* Clanderon Press, Oxford, 1994.

[3] MIT OCW-6.045J, *Automata, Computability and Complexity.*

[4] Piergiorgio Odifreddi, *Classical Recursion Theory: The theory of functions and sets of natural numbers.* Elsevier Science, 1980.

[5] Shmuel Katz, Zohar Manna, *A closer look at termination.* Acta Informatica, Vol. 5, 1975.

[6] Zohar Manna, Nachum Dershowitz, *Proving termination with multiset orderings.* Memo AIM-310, Stanford Intelligence Laboratory, 1978.

[7] Andreas Blass, Yuri Gurevich, *Program termination and well partial orderings.* ACM Transactions on Computational Logic, Vol. 5, 2006.

[8] Michael Sipser, *Introduction to the Theory of Computation.* 2^{nd} Edition, Course Technology, Cengage learning, 2006.

[9] I. N. Herstein, *Topics in Algebra.* 2^{nd} Edition, John Wiley and Sons, Inc. 2006.

[10] Georg Cantor, *Contributions to the founding of the theory of transfinite numbers.* Dover Publications, Inc. 1995.

[11] Donald Knuth, *The Art of Computer Programming : Fundamental Algorithms*, Vol. 1, 3^{rd} Edition, Pearson Education, Inc. 1997.

[12] B. A. Davey, H. A. Priestley, *Introduction to Lattices and Order.* Cambridge University Press, 2002.

[13] Peter G. Hinman, *Recursion-Theoretic Heirarchies.* Springer Verlag, 1978.