Read-Write Reductions

Extended Abstract

Eli Gafni

Department of Computer Science, UCLA, Los Angeles, CA 90095, USA.
eli@cs.ucla.edu

Abstract. The discovery, more than a decade ago, of the relation between Distributed-Computing (DC) and Algebraic-Topology (AT) raised the specter of requiring checking task solvability to be intimately connected to expertise in AT. Yet, in the area of Centralized Algorithms proving a problem to be NP or PSPACE complete requires more algorithmic expertise than complexity one. In analogy, we show that in DC the equivalent of polynomial-time reductions, is read-write reductions. We define the notion of read-write reduction between distributed tasks, and show that all interesting known read-write impossible tasks can be proven impossible via read-write reduction to a task called Symmetry-Breaking (SB). Discovering a read-write reduction requires solely algorithmic expertise.

1 Introduction

Since the introduction of topological arguments [2, 20, 21], they have been used among other uses, to argue the insolvability of a given task in a given model. Do we need now to run and study topology in order to make progress in this area of checking tasks for solvability?

We propose task called Symmetry-Breaking whose role in proving another task to be read-write insolvable is informally but usefully the analogue of the role of SAT in proving NP-completeness. To prove a problem NP-complete one does not need to be an expert in Turing Machine tricks. Once SAT was proven NP-complete from here on NP-completeness is proved by reduction. Similarly, we exhibit that all natural tasks to-date which are known to be insolvable wait-free in the shared-memory read-write model, can be proven so by reduction from the corresponding size SB task.

We do not claim that this analogy to be formal. I.e. we do not claim that if a task A is insolvable wait-free in the read-write model then there exits a reduction. In fact we know that to be false: SB is not solvable even when one is given the use of a Torus task [15] (or for that matter any orientable manifold), thus SB will not suffice to prove that the Torus task is insolvable. Yet we do not know how to pose a Torus task for arbitrary number of processors in a way that will not be "artificial." Thus, we conjecture that there exists a proper "natural" definition of "natural" families of tasks for which SB is the weakest task.

2 Definitions

A task is an input-output relation between sets of processors, each set called a *participating set*, and output tuples, each specifying an output value for each processor in the participating set.

A task is solvable in certain model if there exists a protocol in the model, and a notion of when a processor participates in the protocol, such that for a run in the model with a participating set P, all processor output and halt so that the outputs constitute an allowed output tuple in the task.

The tasks we consider will be families of tasks, each parametrized by n, denoting the maximum size of the participating set. For each task we will consider two families: Non-Comparison, and Comparison. In the former the largest participating set is drawn out of $\{p_0, , p_{n-1}\}$. In the latter, the participating set is any set of processors of cardinality less or equal to n. It is known that the ability to draw the participating processors from a universe large enough is equivalent to the processor identifiers being used only via comparison [12].

Although the task definition is just a mapping from participating sets to output, it encompasses a processor that may wake up with different inputs by considering each input to be associated with different processor identifier and considering the comparison version of the task. Thus w.l.o.g. below we will refer to different inputs, albeit in that case only the comparison version makes sense.

We will say that a task A read-write *implements* B if given any numbers of copies of A and any number of read-write registers than we can wait-free solve B. We than will say that A is potentially stronger than B. If the opposite is also true than we say that A and B are *equivalent*. If we know that B is impossible to solve read-write wait-free, than such an implementation proves that A is insolvable read-write wait-free by *reduction from* B.

3 Task-Families

1. k-Set-Election: The task $SE(k, n)$ says that for all participating sets of size at most n each processor outputs an id of a processor from the participating set, and the number of distinct ids that appear in an output tuple is less or equal to k.
2. k-Strong-Set-Election: The task $SSE(k, n)$ is like Set-Election but in addition if p_i outputs p_j then p_j outputs p_j i.e. itself.
3. k-Set-Consensus: In the task $SC(k, n)$ each processor wakes up with an input tuple of size k, where at position i there is a 0 or 1. It returns as an output an index $1 \leq j \leq k$ and a single bit, 0 or 1. All processors that return the same index j return with it the same bit b. The non-triviality requires that b can be returned together with the index j only if there exist a participating processor who has b in the jth position of its input tuple.
4. k-Test-and-Set: In the task $TAS(k, n),\ n > k$ participating processors output 0 or 1, and each output tuple always contains at least one 0, and at most k 0s. The largest participating set is of cardinality n

5. Symmetry-Breaking: In the task $SB(n)$ processors output 0 or 1. The largest participating set is of size n and the output tuples for this size correspond to all the possible n length bit strings, excluding the all 0s and all 1s strings.
6. n-Adaptive-Renaming: In the task $AR(f(k), m, n)$ processors return positive distinct integers in the range 1 o m. For $k < n$ they return in the range 1 to $f(k)$.
7. n-Renaming: In the task $R(m, n),\ m \geq n$ processors return positive distinct integers in the range 1 to m.

4 Reductions

Below we first outline the sequence of reductions from $SB(n)$ that prove all the above tasks to be read-arite wait-free insolvable given that $SB(n)$. That $SB(n)$ is insolvable is a consequence of its equivalence to comparison $AR(2n-2, n)$. The latter was proved insolvable by direct topological arguments in [20].

In the subsection that follow the outline we elaborate on each item in the outline, respectively. Most of the reductions are almost at the level of folklore. Some are substantial, and then we reference them. No new reduction is introduced here and thus the contribution of the paper is just in organizing all these scattered related known items into a single place.

4.1 Outline of Sequence of Reductions

1. $SB(n)$ and $AR(2n-2, n)$ are trivially solvable in the non-comparison model.
2. Below we show comparison $SB(n)$ and comparison $AR(2n-2, n)$ to be equivalent in the comparison model. The task $AR(2n-2, n)$ was proved to be insolvable in the comparison model in [20].
3. If comparison $SB(n)$ is read-write wait-free solvable then comparison $SB(n+1)$ is [16]. Thus $SB(n+1)$ is weaker than $SB(n)$, but the reduction is white-box rather than black-box reduction. We do not know a black-box reduction for this fact.
4. Non-comparison $TAS(k, n)$ is equivalent to task $TAS(k, n+1)$ and thus to $TAS(k, \infty)$. Below we show that non-comparison $TAS(k, k+1)$ implements $SB(k+1)$, and obviously comparison $TAS(k, k+1)$ implements non-comparison $TAS(k, k+1)$.
5. Non-comparison $SE(k, k+1)$ is equivalent to $SSE(k, k+1)$ which is equivalent to non-comparison $TAS(k, k+1)$. Obviously the comparison versions implement their corresponding non-comparison ones. And obviously $SE(k, k+2)$ implements $SE(k, k+1)$.
6. The non-comparison $SC(k, n)$ is equivalent to $SE(k, n)$.
7. The task $AR(2k-1, 2n-2, n)$ is equivalent to $TAS(n-1, n)$.

4.2 Sketch of Reductions

1. Needs no further comment.

2. (a) Comparison $SB(n)$ implements comparison $AR(2n-2,n)$: Processors use $SB(n)$ to break themselves into two disjoint groups each of which is non-empty when the cardinality of the participating set is n. One group G_0 of cardinality n_0 consists of the processors that output 0 in $SB(n)$ and the other group is G_1 of cardinality n_1. Both groups now use the comparison renaming algorithm $AR(2k-1,2n-1,n)$ in [**?**,8]. Only that G_0 renames from position 1 upwards, while group G_1 renames from position $2n-2$ downward. We observe that $(2n_0-1)+2n_1-1 \leq 2(n_0+n_1)-2 \leq 2n-2$, where the first inequality is true iff both G_0 and G_1 are non-empty. It is easy to see that when the participating set cardinality is less than N the space is enough.
   (b) Comparison $AR(2n-2,n)$ implements comparison $SB(n)$: Processors that obtain values from $AR(2n-2,n)$ in the range 1 to n-1 output 0, while the rest output 1.
3. This was put in to raise the question whether any task A that can be shown to implement B *provided* A *was read-write wait-free solvable*, means that if B is not solvable than B can be reduced to A. We do not know the answer to this question but conjecture the answer to be positive. We challenge the reader to show that comparison $SB(n)$ implements comparison $SB(n+1)$.
4. Trivially non-comparison $TAS(k,n+1)$ implements non-comparison $TAS(k,n)$. To see the reverse put processors p_0 to p_{n-1} through $TAS_1(k,n)$. At most k of them will obtain a 0. They then proceed some to $TAS_2(k,n)$ and some to $TAS_3(k,n)$. They proceed by renaming $AR(2k-1,2n-1,n)$. Those that obtain values in 1 to $n-1$ go to the corresponding ports in $TAS_2(k,n)$, and those that obtain a value j higher than $n-1$ go to port $j-(n-1)$ in $TAS_3(k,n)$. Processor p_n attaches to port n in $TAS_2(k,n)$. Processors that obtain values in $TAS_3(k,n)$ return as final output the negation of their output from $TAS_3(k,n)$, while processor with output from $TAS_2(k,n)$ retain their output. The idea of negation was proposed to us by Rafail Ostrovasky [17].
   To see that non-comparison $TAS(k,k+1)$ implements comparison $SB(n)$ take two copies $TAS_1(k,k+1)$ and $TAS_2(k,k+1)$. The n processors $AR(2k-1,2n-1,n)$ rename into port of $TAS_1(k,k+1)$ and $TAS_2(k,k+1)$ where port i in the latter stand for the integer $i+(k+1)$. processors out of $TAS_2(k,k+1)$ negate their output.
5. (a) The task $SE(k,k+1)$ is equivalent to $SSE(k,k+1)$: Obviously $SSE(k,k+1)$ implements $SE(k,k+1)$. The reverse implementation appears in [2].
   (b) The task $SSE(k,k+1)$ is equivalent to $TAS(k,k+1)$: See [2].
6. Non-comparison $SC(k,n)$ is equivalent to $SE(k,n)$: In this issue [9].
7. The task $AR(2k-1,2n-2,n)$ is equivalent to $TAS(n-1,n)$:
   (a) The task $AR(2k-1,2n-2,n)$ implements $TAS(n-1,n)$: Processors that get values in 1 to $n-1$ output 0, the rest output 1.
   (b) Processors do immediate snapshot [3]. Those that end up with snapshot of size n apply to $TAS(n-1,n)$. They thus divide into three disjoint groups: Group $G_{<n}$ of those that obtain a snapshot of size less than n, Group G_0 of those that obtained 0 in $TAS(n-1,n)$, and the rest are in G_1. Processors in $G_{<n}$ and G_1 $AR(2k-1,2n-1,n)$ rename from 1 upward while those in G_0 rename from $2n-2$ downward [8, 14, 18].

5 Conclusions

We have presented a sequence of reductions/implementations that show how all known interesting insolvable task can be deemed so by reduction from the family SB. This led to the speculation that any "interesting" task is at least as strong as SB. Indeed that speculation led to a renewed push to understand the relation between SB and TAS that has recently resulted in the conclusion that SB is strictly weaker than TAS. We also leave some interesting open problems. If it can be (white-box) shown that the read-write wait-free solvability of A, either by assumption of the existence of read-write code as in the BG-simulation [2, 5], or by considering the topological ramification of such solvability, would imply the read-arite solvability of task B, does it necessarily implies that B can be reduced to A (black-box)? It will be elegant and satisfying if the answer is positive.

Finally, it will be of the utmost interest to capture rigorously what is informally considered a "natural" task family and show that any task of interest at the least breaks symmetry.

Acknowledgment: I am in debt to Sergio Rajsbaum who assigned me to give an invited talk at DISC 2004 Godel celebration session. An assignment that resulted in [8], where the idea of equating SB to SAT was first introduced.

References

1. Afek Y., H. Attiya, Dolev D., Gafni E., Merrit M. and Shavit N., Atomic Snapshots of Shared Memory. *Proc. 9th ACM Symposium on Principles of Distributed Computing (PODC'90)*, ACM Press, pp. 1–13, 1990.
2. Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations *Proc. 25th ACM Symposium on the Theory of Computing (STOC'93)*, ACM Press, pp. 91-100, 1993.
3. Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming (Extended Abstract). *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93)*, ACM Press, pp. 41-51, 1993.
4. Borowsky E. and Gafni E., A Simple Algorithmically Reasoned Characterization of Wait-Free Computations (Extended Abstract). *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, ACM Press, pp. 189–198, 1997.
5. Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG Distributed Simulation Algorithm. *Distributed Computing,* 14(3):127–146, 2001.
6. Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.
7. Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.
8. Gafni E., DISC/GODEL presentation: R/W Reductions (DISC'04), 2004. http://www.cs.ucla.edu/˜ eli/eli/godel.ppt
9. Afek Y., Gafni E., Rajsbaum S., Raynal M. and Travers C., Simultaneous Consensus Tasks: A Tighter Characterization of Set-Consensus, This issue.
10. Gafni E., Group-Solvability. *Proc. 18th Int. Symposium on Distributed Computing (DISC'04)*, Springer Verlag LNCS #3274, pp. 30–40, 2004.
11. Gafni E. and Kouznetsov P., Two Front Agreement with Application to Emulation and Robustness. *to appear*.

12. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., and Reischuk, R., Renaming in an asynchronous environment, J. ACM 37, 3 (Jul. 1990), 524-548.
13. N. Fredrickson and N. Lynch, Electing a Leader in a Synchronous Ring, JACM, January 1987.
14. Gafni E. and Rajsbaum S., Musical Benches. *Proc. 19th Int. Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS # 3724, pp. 63–77, September 2005.
15. Eli Gafni, Sergio Rajsbaum, Maurice Herlihy, Subconsensus Tasks: Renaming is Weaker than Set Agreement, DISC06, Springer Verlag LNCS #4167, Stockholm, Sweeden, September 18-20 , 329-339, 2006.
16. Gafni E., In preparation.
17. Ostrovsky R., Private communication to the author.
18. Gafni E. and Rajsbaum S., Raynal M., Travers C., The Committee Decision Problem. *Proc. Theoretical Informatics, 7th Latin American Symposium (LATIN'06)*, Springer Verlag LNCS #3887, pp. 502-514, 2006.
19. Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on programming Languages and Systems*, 11(1):124-149, 1991.
20. Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.
21. Saks, M. and Zaharoglou, F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.