The $0-1$-Exclusion Family of Tasks

Eli Gafni*

July 20, 2008

Abstract

Interesting tasks are scarce. Yet they are essential as an investigation material, if we are to understand the structure of the tasks world. We propose a new family of tasks called 0-1 Exclusion tasks, and show it is an interesting family.

A 0-1 Exclusion task on n processors is specified by a sequence of $n-1$ bits $b(1), b(2), ..., b(n-1)$. For participating set of size $k,\ 0 < k < n$, each processor is to output 0 or 1 but they should not all output $b(k)$. When the participating set is of size n then they should all output neither all 0's nor all 1's.

Only one member of this family, the one specified by $b(1) = b(2) = ... = b(n-1) = 1$, was implicitly considered in the past and shown to be equivalent to Set-Consensus. In this initial investigation of the whole family we show that not all of its members are created equal. We take the member specified by $b(1) = 1, b(2) = ... = b(n-1) = 0$, and show that it is read-write unsolvable for all n, but is strictly weaker than Set-Consensus for n odd.

We show some general results about the whole family. The family is sandwiched between Set-Consensus from above and Weak-Symmetry-Breaking from below. Any Black-Box of n ports that solves a 0-1 Exclusion task, can be used to solve that task for n processors with ids from unbounded domain.

Finally we show an intriguing relation between Strong-Renaming and the 0-1 Exclusion family, and make few conjectures about the implementations relationships among members of of the family, as well as possibly tasks outside it.

Regular submission.

*Computer Science Department UCLA Los Angles, CA 90095 eli@cs.ucla.edu

1 Introduction

Recently, in a surprising result, it was shown that within the sequence of tasks WSB=$WSB_2, WSB_3, ...$ there are certain values of n for which WSB_n is read-write solvable [2]. To us this indicated that in the first place there was no precise rationale to consider all size instances of WSB simultaneously. The collection is not cohesive. What makes a collection cohesive? We have no answer to this general question, but to get there, we first asked whether we can find a task on n processors which is unsolvable for all n and is strictly weaker than Set-Consensus. In other words, a task to replace the fallen WSB, which was thought to be unsolvable for all n [6], and weaker than Set-Consensus [3].

This is where the 0-1 Exclusion family comes from. We started with a set of tasks which is as cohesive as they come: A Uniformly Solvable task [5]. We took The Uniformly Solvable task specified by an infinite sequence of bits $b(1), b(2), ...$ in which a processor outputs 0 or 1. When the size of the participating set is $k > 0$ then the only constraint is that processor will not all output $b(k)$. Obviously this is a Uniformly Solvable task as there is a single algorithm to solve it: A processor that observes a size of participating set k outputs $1 - b(k)$. We then asked what happens to this task if we impose one additional constraint: When the size of the participating set is some single $n > 1$ processors are not only precluded from all outputing $b(n)$, but they are also precluded from all outputing $1 - b(n)$, i.e. when the size of the participating set is n at least one processor outputs 1 and at least one outputs 0. Notice that we got the the 0-1 member by removing a single n-tuple from the Uniform task. For each n we get a different task on N processors $p_1, ..., p_n$. It is known that when we start with the Uniformly Solvable task in which for all k $b(k) = 0$, we fix an n, then we get a task which is equivalent to Set-Consensus on n processors [8]. But what about other sequences?

The main contribution of this paper is the discovery of this novel family of tasks. In the initial investigation of this family, to show that it is at the least a non-boring family, we concentrate on the sequence $b(1) = 1$ and for all $k > 1$, $b(k) = 0$. We call the resulting sequence of tasks one for each $n > 1$, SEA (Single-Exclusion-Alternation).

We show some broad brush results pertaining the whole family, and some results for SEA in particular. Most importantly, we show that SEA is read-write unsolvable for all n, and for n odd is strictly easier than set consensus. For any specific instance of a member of the family one can evaluate a combinatorial expression to check for its solvability, i.e. the question of the solvability of a member of the class is a decidable question [11]. As yet we did not find a solvable member. We nevertheless have no clue how to make this into an unsolvability proof of the whole family.

Before we enumerate all our 0-1 Exclusion results we need some simple definitions. Lets T_n be an n-processors task, member of the 0-1 Exclusion family. For $0 < k < n$ let $b_{T_n}(k)$ be the bit excluded for participating set of size k. We define two associated tasks, the complement task $C(T_n)$, and the dual task $D(T_n)$, both on n processors. For $C(T_n)$, $b_{C(T_n)}(k) = 1 - b_{T_n}(k)$. I.e it is the same task only replacing 0's with 1's and 1's with 0's. For $D(T_n)$, $b_{D(T_n)}(k) = 1 - b_{T_n}(n - k)$. I.e. if we take the complement of the dual we get a task whose exclusion sequence is the reverse sequence of original exclusion sequence. Obviously, the dual of the dual is the original. Finally, if T_n is a member of the 0-1 Exclusion family, then the task $SYM(T_n)$ is a task on $2n - 1$ processors, that for any participating set of size $0 < k < n + 1$ allows the same set of tuples that T_n allows. The task $SYM(T_n)$ when viewed on n processor of ids 1,...,n essentially requires that a solution is symmetric, processors can only compare their ids rather have, say, a priori different programs for processors whose ids is odd and another one for processors whose ids is even. Thus $SYM(T_n)$ is potentially harder than T_n.

Let T_n be a member of the 0-1 Exclusion family. Here we enumerate the results:

1. The four tasks T_n, $C(T_n)$, $D(T_n)$, and $C(D(T_n))$, are equivalent.

2. $SYM(T_n)$ is not harder (i.e. it is equivalent) than T_n.

3. SEA is unsolvable for all n.

4. SEA is strictly weaker than Set-Consensus for n odd.

5. Any member of the 0-1 Exclusion family is weaker (not necessarily strictly) than Set-Consensus.

6. Any member in the family of 0-1 Exclusion solves WSB.

Thus the 0-1 Exclusion family is sandwiched between Set-Consensus on the top and WSB on the bottom.

We apply the idea of creating an unsolvable task by "shrinking" a Uniform Solvable task, to Strong Renaming. We propose many potentially unsolvable sequences of tasks by choosing a combination of n integers and eliminating all the output n-tuples that contain just them. That is, we choose n distinct integers set $D = \{i_1, ..., i_n\}$ in the range 1 to $2n$-1 and we remove an n-tuple $tuple$ if $\{w|(p_i, w) \in tuple\} = D$. Unlike $(n, 2n-2)$-Strong-Renaming which eliminates an "integer" in the sense that it preclude any processor from outputing it, we eliminate much less, just a single combination of n integers out of the n choose $2n-1$ available. We show that each instance created this way solves a member of the family of the 0-1 Exclusion. Further more, we show that all these instances are not as strong a Set-Consensus. We solve Strong-Renaming instance on 3 processors that exclude certain 3-tuple, by using SEA_3 which we prove is strictly weaker than Set-Consensus. Thus 0-1 exclusion has the potential to be the "weakest" among unsolvable tasks, when the unsolvable tasks are created, in a way yet to be formalized, from Uniformly Solvable one.

The paper is organized as follows: The next section is the Model Section. "Walking the talk," there is very little "topology" in our paper. Most of our argumentation is algorithmic. Nevertheless, in arguing the Unsolvability of SEA, we will assume that the reader is familiar with the definition of subdivided simplex, it faces, etc., a notion that appears in most standard 3rd year Computer-Science Algorithms books. We will just quote a result concerning invariance over the different subdivision of a subdivided simplex. The "weakness" of SEA viz Set-Consensus follow verbatim from [3], we thus do not copy that model section. The "weakness" here is exactly in the same sense that there, it was and is here proved under the round-by-round Hypothesis which proves it in a limited model hypothesized to be equivalent to the general model. Aside from the section on task to get the vocabulary, the results we rely in are just out of other papers. We do not develop any new topological tool in this paper. We nevertheless, have gone the path of unifying terminology in terms of output $tuples$. Since this is a side show of this paper, it may end up to be a bit terse. We then prove our results viz the 0-1 family, and SEA in particular. Following that, we discuss the relation between Strong-Renaming and the 0-1 exclusion family. Finally, we conclude with rehashing the numerous open problems and conjectures left on the table.

2 Model

2.1 tasks

In Distributed Computing, the analogue of a $function$ Turing-Machines compute, is a $task$. A task is independent of the model of computation it is to be solved. A task encompasses the level of coordination it requires. A consensus task requires the strongest coordination, while as task in which processors output their id requires no coordination. Thus given a task it will be solved in a model only if this model provides the level of coordination the task requires.

The only thing we will require from any model of distributed computing is to have the notion of $participating - set$, that it a definition that allows us to take an infinite distributed computation in the model and decide which processors participate in that computation.

To define a task T on n processors, $p_1, ..., p_n$, we consider any subset of the processors as participating set. The specification of the task is a map from each participating set P to a set $\Delta_T(P)$ of $tuple$s where a tuple is a set of $|P|$ pairs $\{(p_i, w \in O) | p_i \in P\}$ for some w in an output values set O.

The interpretation is that if the participating set is P, if all processors in P output, the combination results in a tuple in $\Delta_T(P)$.

For example, in the $k-$Set-Consensus task [] on processor $p_1, ..., p_n$, the output set is the the ids of the processors, $\Delta_T(P)$ is such that for each pair $(p_i, w) \in tuple$, $tuple \in \Delta_T(P)$ implies $w \in P$, and $|\{w|(p_i, w) \in tuple\}| < k + 1$. We will use the shorthand Set-Consensus when when the universe of processors is n processors and $k = n - 1$.

If we want to model a situation when "p_i may start with different inputs," as traditionally tasks are defined [], we just increase the universe of processors and distribute the inputs among processors as to get each processor to have a single input value. With a single input value we can ignore the value as the processor's id incorporates it.

For instance, to define the task $binary - consensus$ [9] we define it on $2n$ processors. The Output set is $O = \{0, 1\}$. The output values in a tuple have to agree, they are either all 0, or all 1. If the participating set is a subset of the first n processors then there is a single output tuple for that participating set, with all output values, say 0, if the participating set is subset of the second n processors, then the only output value possible is 1. If the cardinality of the participating set is strictly bigger than n then anything goes.

The the nth instance of Weak-Symmetry-Breaking (WSB_n) task [3], is a task on $2n - 1$ processors. The Output set is $O = \{0, 1\}$. When the cardinality of the participating set is n, then each output tuple contains some pairs in which the output value is 0, and some pairs in which the output value is 1. For other cardinalities, anything goes.

In the $(2n - 1, q)$-Weak-Renaming task [1], the task is defined on $2n - 1$ processors. The output set is $O = \{1, 2, ..., q\}$. For participating set of size $k = n$, no two pairs in the pairs of a tuple agree on the output value. For other cardinalities, anything goes.

The task (n, q)-Strong-Renaming (also Called Adaptive-Renaming) on n processors [1], the output set is $O = \{1, 2, ..., q\}$. For participating set of size k, no two pairs in the pairs of a tuple agree on the output value, moreover, no output value in a pair is larger then $2k - 1$.

Of particular interest is the Immediate-Snapshot task IS_n on n processors [10]. We define it recursively, as IS is a Uniformly Solvable task []. Task IS_1 on processor p_1 has a singleton tuple $(p_1, \{p_1\})$. Task IS_n extend IS_{n-1} in the following way: For participating set of size $k < n$ that include processor p_n, the output set of k-tuples is isomorphic to the output set for $p_1, ...p_k$ with any fixed one to one correspondence between processor's ids. For participating set of size $k = n$ we take a new values from O say $o = \{p_1, ..., p_n\}$, create the n pairs $(p_1, o), ..., (p_n, o)$. We now create the n-tuples by taking any k-tuple from any participating set of size k and inserting enough new pair to make it syntactically valid n-tuple.

2.2 Tasks Solving Tasks

Let task A and B be defined on the same set of processors. We say that task A $solves$ task B if there is a function s from pairs in A to pairs in B that preserves the first entry of the pair, i.e. the processor associated with the pair, such that every tuple in A is mapped to a tuple in B.

2.3 Sequential-Composition of tasks

Given two two tasks A and B on the same set of processors we define the task BA which we call $A\ followed$ by B. We create the output tuple for participating set P by taking a tuple $\{(p_i, w_a)\}_{p_i \in P} \in \Delta_A(P)$, and a tuple $\{(p_i, w_B)\}_{p_i \in P} \in \Delta_B(P)$, and adding a tuple $\{(p_i, w_A, w_B\} p_i \in P$ to $\Delta_B A(P)$.

A composition may come adjoined with a $termination$ map. A termination map is a partial function on pairs that return $HALT$ or $CONT$. Once a pair $(p_i, *)$ is mapped to $HALT$, processor p_i is not in the participating set of the next task and the composition does not add anything further to the $(p_i, *)$.

Of particular interest is self composition. Consider a self composition of A with itself infinitely many times. If the composition is adjoined with a termination functions such that after finite number of composition every pair has $HALT$ed, then we'll call the composition $well - composed$. A k-stage composition for instance can be modeled by a termination map that $HALT$s a pair once it "accumulates" k output values.

2.4 Well-Composition of IS_n

For the purposes of this paper, every well-composition of IS_n with itself is called an $n-1$ dimensional $subdivided\ simplex$. The pairs of a well composed IS_n are of the form $(p_i, P_1, P_2, ..., P_k)$ where this pair maps to $HALT$ and no prefix $(p_i, P_1, P_2, ..., P_q),\ q < k$ does. Notice that in any tuple all the pairs map to $HALT$. A a k-tuple is called a k-simplex.

A $carrier$ of a pair q, denoted $carrier(q)$, is the union of the participating sets in the sequence in q. All k, $(k-1)$-tuples that are sub-tuple of a k-tuple are its $k-1$-faces. The $k-1$-face of the subdivided simplex defined by a set of processors P such that $|P| = k$, are all the tuples the carrier of each pair of which is a subset equal P. All the $n-1$-tuples whose carriers are not of cardinality n are the $boundary$ of the subdivided-simplex.

Consider IS_n. An n-tuple is of the form $\{p_i, P_i\}$. Let the parity of the tuple be odd or even according to the number of distinct participating sets in the pair of the tuple. It is easy to check that if two n-tuple share $n-2$ face (that is made of $n-1$ processors), then they have different parity. We leave it as an exercise to show that there is a generalization of this property to any subdivided-simplex. I.e. there a rule to give each n-tuple a "parity" such that two adjacent n-tuples are of different parity. We say then that a sub-divided simplex is $orientable$. The parity function that maps n-tuples to "+" or "-" is the orientation.

We will use the following invariance result. It says that a 0,1 coloring of a boundary of a subdivided simplex determines a property of the inside for any legal "inside."

Let S be oriented subdivided simplex. If the nodes on the boundary are colored by 0 or 1, then for any 0,1 coloring of the rest of the nodes, we count in a certain way the number of n-tuples where all the pairs of the tuple are colored by the same color. This number is called the $content$ of the subdivided simplex. We call a tuple whose all pairs are of the same color c, mono-c.

1. If n is odd: Sum the number of n-tuples mono tuple considering the sign it gets from the bipartite graph.

2. If n is even: Do the above only that if a tuple is a mono-1 add an extra "-."

The result we will use is that once the boundary and its coloring is fixed the content is the same no matter how we got the boundary [2]. In particular given a boundary, it was created by some stages of IS's, and some termination rules. In this paper will create a subdivided simplex with the same boundary by terminating any processor p_i after the first stage if the pair is $(p_i, \{p_1, ..., p_n\}$. We then continue with the original termination rule.

If a well-composed IS_n A_n solves a 0-1 Exclusion task T_n, let c be the binary coloring of vertices of A_n induced by the solution, then the content of A_n is 0. This is a corollary of [6] as it implies that to be a solution the induced coloring has to have no mono n-tuple.

Let B_n be a well-composed IS_n. Eliminate some internal n-tuples (an n-tuple that has no vertex on the boundary) to get $B_n(holes)$. If the task $B_n(holes)$ solves a 0-1 Exclusion task T_n, then the binary coloring induced on the boundary of B_n is such that the content of B_n for any coloring of the interior vertices is some a priori fixed number $C(T_n)$.

This follows from the fact that we can count the content by the particular subdivision mentioned above of planting a single internal n-tuple and coloring it 0, then the only possible monos are mono-0's. Whether a combination of faces is the support of a mono-0 n-tuple (that is an n-tuple whose vertices live on these faces and not on less) is possible iff the $b(k)$ associated with each of the faces is 1, and then inductively the numbers of such mono-0 associate with these combination is 1 or -1 depending on the parity of the number of the faces. Thus given a 0-1 Exclusion task T_n it is a simple combinatorial exercise to find $C(T_n)$.

2.5 SWMR-Atomic-SM and Immediate-Snapshots

We will assume the standard asynchronous Atomic SM [7]. In this model processors alternate between writing in their SM cell and reading all SM cells in a snapshot. Cells are initialized to $\perp$. An execution is a sequence of processors ids where the first appearance of a processor is interpreted as a $write$ and its next appearance in the sequence as a $read$ etc. A processor $participates$ if it appears in the sequence. We will assume "full-information" model in which each $write$ appends to its cell what it read in the preceding $read$. Following a $read$ a processor holds a $view$. The view is what would be the content of its cell in the next $write$.

A protocol π is a partial function for each processor from its view to an output.

We will say that a protocol π solves T_n wait-free in the model, if for all infinite executions, all processors that appear infinitely many times have views that map to outputs. Furthermore, if all processors in an execution with a participating set P output, then the tuple $tuple$ they created satisfies $tuple \in \Delta_{T_n}(P)$.

We will use the following result [12]:

Theorem 1: A task T_n is solvable in SM, iff it is solvable by some well-composed IS_n.

2.6 Task Implementation

We say that task A implements task B, if the SM model equipped with A as a procedure, B is solvable.

2.7 The Round-by-Round Hypothesis

The following has not been proved yet, but is believed by all. When we say that SEA is weaker than Set-Consensus it is modulo this Hypothesis.

The Round-by-Round Hypothesis: A implements B, iff some well composition of AIS_n solves B.

3 Properties of 0-1 Exclusion and Family and the SEA Task

3.1 The four tasks T_n, $C(T_n)$, $D(T_n)$, and $C(D(T_n))$, are equivalent

The only nontrivial part is the follows:

Theorem 2: The task T_n and $D(T_n)$ are equivalent.

Proof:

Since $D(D(T_n)) = T_n$, it suffice to show that T_n implements $D(T_n)$.

Let each processor register in shared memory and then each take a snapshot to see the number of registrants. If the number is $k,\ 0 < k < n$, then the processor outputs $b_{T_n}(n-k)$, else, if $k = n$ it goes to T_n and output from it.

To see that this solves $D(T_n)$, notice that if the participating set is of size $k,\ 0 < k < n$, then at least one processor will output $b_{T_n}(n-k)$. Thus the tuple all $1 - b_{T_n}(n-k) = b_{D(T_n)}(k)$, is avoided.

If the participating set is $k = n$, then consider the case that $r,\ n > r > 0$ is the number of processors that went to obtain output from T_n. Since T_n excluded $b_{T_n}(r)$ at least one of the r processors will output $1 - b_{T_n}(r)$. Of the $(n-r)$ processors that did not go to T_n, at least one will see $n-r$ initial registrants and output $b_{T_n}(n-(n-r)) = b_{T_n}(r)$. This one will be the on that executed the last $read$ that resulted in that the number of registrants is less than n. Thus, not all output 0 and not all output 1.

If $r = n$, then by the specification of T_n not all output 0 and not all output 1.

Corollary 3: If there exits a read-write algorithm for T_n then there exists a algorithm by which a processor stops after its first write-read step if it observes a participating set of size $k < n$. It then outputs $1 - b_{T_n}(k)$.

Proof: Consider the implementation of T_n from $D(T_n)$.

3.2 $SYM(T_n)$ is not harder (i.e. equivalent) to T_n

Theorem 4: $SYM(T_n)$ is not harder (i.e. equivalent) than T_n.

Proof:

Take T_n with ports 1 to n. Processors register and then observe a toggle bit called a "gate" which has two states, it can be "open" or "closed." The gate is initially open. A processor that observes the gate open, Strongly Rename [1] to obtain an output we call $port$. If he number of the port obtained is less than $n+1$ it invokes T_n at that port and departs with an output from T_n. If the port it gets is greater than n or initially it observed the gate closed it registers in a $spill - over$ set, and toggles the gate to close.

If all processors renamed into T_n we are done. Else, at least one processor must rename into T_n. If $r,\ 0 < r < q \leq n$ rename into T_n then at least one of those will output the bit $1 - b_{T_n}(r)$. To get the complement bit by at least one processor, let p_j in the $spill - over$ set snapshot and get $spill - over_j$ and $registered_j$. It then outputs $b_{T_n}(|registered_j| - |spill - over_j|)$. Since at least one, the last one to take a snapshot, will obtain $|registered_j| = q$ and $|spill - over_j| = q - r$, then at least one will output $b_{T_n}(q-(q-r)) = b_{T_n}(r)$. Thus at least two processors will output different bits. A crucial observation is that Strong-Renaming will never rename all the processors invoking it to the $spill - over$ set, hence we are allowed to state $r > 0$. The function of the gate is to prevent a processor who arrived after a processor from the $spill - over$ set departed, to go the T_n. This will follow from the fact the processors in the $spill - over$ set close the gate, and no processor ever opens it. We do this in order to maintain that at least one processor from the spill-over set will see the correct final cardinality of the $registered$ set (q), and the final cardinality of the $spill - over$ set ($q - r$). This follows from he fact that after the last processor to register into the $spill - over$ set does so, no process will register, period.

The code appears as algorithm 1.

Algorithm 1 $SYM(T_n)$ from T_n

1: Initially: $REG[1..n] = SPLOVR[1..n] = FALSE$,, $GATE = FALSE$, $PORT[1..n]$ $RREG[1..n], RSPLOVR[1..n]$
2:
3: $REG[i] := TRUE$;
4: **if** $GATE = TRUE$ **then**
5: $PORT[i]$:=**call(Strong-Renaming);**
6: **if** $PORT[i] < n + 1$ **then**
7: **return(call(**T_n **at port** $PORT[i]$**))**
8: **end if**
9: **end if**
10: $GATE := FALSE$;
11: $SPLOVR[i] := TRUE$;
12: $RREG[i] := |\{j|REG[j] = TRUE\}|$;
13: $RSPLOVR[i] := |\{j|RSPLOVR[j] = TRUE\}|$;
14: **return(**$b_{T_n}(RREG[i] - RSPLOVR[i])$**)**

3.3 SEA is unsolvable for all n

Theorem 5: SEA is unsolvable for all $n > 1$.

Proof:

Let NC be a subdivided simplex that solves SEA. To be a solution, the number of mono n-tuples should be zero, while the boundary should satisfy the specification of the task.

It follows from [2], and discussed in the Model Section, that given a boundary of a chromatic subdivided simplex that is colored by 0 and 1 any other chromatic subdivision that agrees with it on the boundary will result in the same Content. Thus we can count the Content using a particularly convenient subdivision. If this subdivision results in a Content that in absolute value different than 0, then it is a proof that no other subdivision will get what we need, which is a count of 0. We will do this by taking the boundary of NC, planting an $n-1$ chromatic simplex in it and coning the appropriate faces of the boundary. We will consider the binary coloring of all the nodes of the middle n-tuple to be 0.

By the specification of the problem aside from the 0-dim face, no proper face has a mono-0 tuple. (Alternatively, by the corollary 2 we can assume w.l.o.g that the only 0's on the boundary are the nodes that are the 0-dim faces.) Since the nodes internal to the $n-1$ face are all 0 by construction, then the only mono n-tuples are the mono-0. Thus it is easy to see that the only mono-0 n-tuple are the middle one, and all n combinations of choosing $n-1$ vertices out of the middle n-tuple and combining it with the 0-dim face across from it. Thus, we get $n+1$ mono-0 simplexes. If the orientation of the middle simplex is +, then all other simplexes share an $n-1$ face with it and therefore are all of orientation -. Consequently the Content is $-(n-1)$. Thus, for $n > 1$ we have that the content is strictly different than 0.

Corollary 6: For n odd if we use the technique in [2] to eliminate a pair of positive and negative mono-0's, and we unify the rest of $n-1$ mono-0 simplexes in pairs, we obtain a pseudo-manifold that solves SEA_n.

3.4 SEA is strictly weaker than Set-Consensus for n odd

Theorem 7: SEA is strictly weaker than Set-Consensus for n odd.

Proof:

The construction in [3] that shows that Renaming is easier than set consensus happen to be binary colored in a way that satisfies not only the requirements of WSB, but also satisfies the requirements of the stronger problem SEA, and therefore it also proves that Set-Consensus is strictly stronger than SEA. (See also Corollary 5.)

3.5 Any member of the 0-1 Exclusion family is weaker (not necessarily strictly) than Set-Consensus

Theorem 8: Any member of the 0-1 Exclusion family is implementable by Set-Consensus.

Proof:

Using $(n, n-1)$-Set-Consensus by which n processor output at most $n-1$ distinct names of participating processors one can solve $(n, 2n-2)$-Strong Renaming where for participating set of size $k < n$ processors rename between 1 and $2k-1$, while for $k = n$, they rename between 1 and $2n-2$ [4].

Consider a member in the 0-1 Exclusion family specified by $b(k),\ k = 1, ..., n-1$. We color of $r = 1, ..., 2n-2$ by 0 and 1. Let the color of i be $CS(i)$. We color as follows:

1. $CS(1) = 1 - b(1)$,
2. If $b(k) = b(k-1), 1 < k < n$ then $CS(2k-1) = 0$, and $CS(2k-2) = 1$,
3. If $b(k) \neq b(k-1), 1 < k < n$ then $CS(2k-1) = CS(2k-2) = b(k-1)$,
4. $CS(2n-2)$ is colored by the color that is minority slots in $CS(1)$ up to $CS(2n-3)$.

It is an easy induction to see that this coloring has the invariant that $abs(|\{0 < i < 2k < 2n-1 | CS(i) = 1\}| - |\{0 < i < 2k < 2n-1 | CS(i) = 0\}|) = 1$, and moreover the majority of colors is $1 - b(k)$. Also if we consider all the integers, the number that is colored by 1, equal the number that is colored 0, equals $n-1$. Consequently, if a processor outputs the color of the integer it obtains we satisfy the specification of the 0-1 Exclusion problem.

3.6 Any member in the family of 0-1 Exclusion solves Impossible Weak-Renaming

Theorem 9: Any member in the family of 0-1 Exclusion solves Impossible Weak-Renaming.

Proof:

Follows from the result that T_n is equivalent to $SYM(T_n)$, and the latter is a restriction of WSB.

4 Strong-Renaming and the 0-1 Exclusion Family

In the task $(n, 2n-1)$-Strong-Renaming if the participating set if of size $1 \leq k \leq n$ then the k processor each return a unique integer from the range 1 to $2k-1$. It is known that $(n, 2n-2)$-Strong-Renaming is equivalent to Set-Consensus [4].

What if we fix n integers and we preclude all n processors to together return these n combination? Precluding processors from returning the integer $2n-1$ is much more, it it precluding them from returning any tuple with a pair the contains the integer $2n-1$. Thus, this is potentially an easier problem. Is there a single n-combination that can be eliminated and the problem can still be read-write solvable.

We only partially answer this question. We show that the elimination of an n-combination results in a problem that is potentially harder than some member of the 0-1 Exclusion. If it will be proven that all members of the 0-1 Exclusion family are unsolvable then the elimination of any n-combination results in an unsolvable task.

On the flip side we show that there exists a collection n-combinations the elimination of any will results in a task which is equivalent to Set-Consensus. Perhaps any single combination elimination results in a task equivalent to set consensus? We answer on the negative. We show that for $(3, 5)$-Strong-Renaming, if we eliminate the combination 2,3,and 5 then SEA on 3 processors solves the problem. Since for 3 processors SEA is shown to be strictly easier than Set-Consensus, it shows the result.

4.1 Reducing Strong-Renaming to 0-1 Exclusion

It is easy to see that if we eliminate the combination $(n, n + 1, ..., 2n - 1)$ then we get a task which is equivalent to set consensus: Set-Consensus solves $(n, 2n - 2)$-Strong-Renaming [4]. The task $(n, 2n - 2)$-Strong-Renaming is harder than the task at hand. To see that our task solves Set-Consensus, let processors that output between 1 and $n - 1$ output 0, and above $n - 1$, output 1. It is easy to see by the specification of Strong-Renaming that for $k < n$, at least one processor must output 0. Once we have a participating set of size n, they cannot all output 0, as there are only $n - 1$ positions of outputing 0. On the other hand they cannot all output 1, as the any tuple that will result it is eliminated. Thus, the elimination of that single combination is equivalent to the elimination of the $2n - 1$'st integer, i.e. all the tuples the contain position $2n - 1$. Obviously this reasoning goes through as long as for any $k < n$ the number of integers in the n-combination whose value is less equal $2k - 1$ is less than k.

What about eliminating other tuples? Suppose we create the task SR_{comb} from $(n, 2n - 1)$-Strong-Renaming by eliminating the combination $comb$. We show how to create a 0-1 Exclusion member E_{comb} such that SR_{tuple} solves E_{tuple}.

Consider all the integers in $comb$ to be colored by 1, and the rest of the integers by 0. Denote the color of integer i by $CS(i)$. Create the 0-1 Exclusion task E_{comb}, such that for participating set $0 < k < n$ one excludes $b(k)$. The bit $b(k)$ is the minority bit of the integers 1,...,$2k - 1$, i.e. if $|\{i|CS(i) = 0, i \leq 2k - 1\}| \leq k - 1$ then $b(k) = 1$, else $b(k) = 0$.

If processors in SR_{comb} return the color of the integer they obtained, then they solved E_{comb}.

4.2 An SR Task equivalent to a weak 0-1 exclusion member

We now show how for 3 processors, $E_{(2,3,5)}$ implements $SR_{(2,3,5)}$: For $E_{(2,3,5)}$ we have $b(1) = 1, b(2) = 0$. The implementation follows. For historical reasons and metaphors we will use the word "slot" as a synonym with integer. Processors obtain an output from $E_{(2,3,5)}$. If a processor obtained 0 it raises a flag for slot 1, if it obtained 1 it raises a flag for slot 2. It then checks whether another processors raises a flag for the same slot (it cannot be that 3 processor will). If not then it return the slot. Else, if it is smaller among the two and its flag is on slot 2, it returns slot 3. If it is the bigger among the two and its flag is on slot 2, it raises a flag for slot 1. If it smaller among the two and its flag is on slot 1 it returns 4, else it returns 5.

This algorithm is a finite state machine and I have checked all possibilities. The idea though is that processors that return 0, start strong renaming on slots colored 0 and processors that returned 1 start strong renaming on slots color 1. They nevertheless are not allowed to go above slot $2k - 1$. Thus if they spill-over they start going backward on the complement colored slots.

5 conclusions

The trigger to this paper is the recent discovery that WSB is solvable for certain values of n [2]. This means that different size instance of WSB do not make a cohesive whole. To make a cohesive collection of solvable tasks we proposed in the past the notion of Uniform Solvability [5]. The idea behind this paper is to create a cohesive unsolvable collection by taking a solvable cohesive collection and eliminating output tuples. It resulted in the discovery of the 0-1 Exclusion family as well as the weaker unsolvable versions of Strong-Renaming.

The main advance we miss is a formalization of this process of deriving what we would term a Uniformly Unsolvable task out of Uniformly solvable one.

For instance, why excluded the all 1's or the all 0's tuple and not a collection of tuples of the type, say, one processor output 0 and all the rest 1's? Why eliminate an n-combination the case of Strong-Renaming, rather than a single n-tuple. We have no good rationale for that as yet.

On the technical side the main question left is the impossibility of all the instances of the 0-1 family for any n, as well as finding the implementation relationships between members of the family. It is easy to see that unlike WSB given an exclusion task on n processors the the Content of any solution is the same over all solutions. Thus we conjecture that given two Exclusion tasks $T1$ and $T2$ on n processors, one with Content a_1 and the other with a_2, then $T1$ implements $T2$ if and only if a_1 divides a_2, otherwise, if none is a divisor of the other, then they are unrelated. Since the Content of Set-Consensus is 1 it solves the whole family.

We also conjecture that once the notion of Uniform Unsolvability is formalized, then any Uniformly Unsolvable task solves some instance of the 0-1 Exclusion family. We have shown this promise via example by taking Strong-Renaming, creating supposedly Uniformly Unsolvable task out of it, and show that indeed, every instance implements some 0-1 Exclusion family member.

For one and half decades Weak-Renaming was the only game in town. When pressed for a task possibly weaker than Set-Consensus only Renaming came to mind. How did I (we) miss the 0-1 Exclusion and Excluding a combination from Strong-Renaming rather than a whole integer, for so long? I guess that is the kind of a question that comes after every interesting discovery.

Acknowledgment Yehuda Afek caught a mistake that brought in the "gate" in the proof of Theorem 4. I am in debt to Armando Castaneda for making himself available for my numerous questions. I owe Sergio for being Sergio.

References

[1] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, Rdiger Reischuk: Renaming in an Asynchronous Environment J. ACM 37(3): 524-548 (1990).

[2] Armando Castaneda and Sergio Rajsbaum: New Combinatorial Topology Upper and Lower Bounds for Renaming, To appear in PODC 08.

[3] Eli Gafni, Sergio Rajsbaum, Maurice Herlihy: Subconsensus Tasks: Renaming Is Weaker Than Set Agreement. DISC 2006: 329-338.

[4] Achour Mostfaoui, Michel Raynal, Corentin Travers: Exploring Gafni's Reduction Land: From Omegak to Wait-Free Adaptive (2p-[p/k])-Renaming Via k-Set Agreement. DISC 2006: 1-15.

[5] Eli Gafni, A Simple Algorithmic Characterization of Uniform Solvability. FOCS 2002: 228-237.

[6] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Eli Gafni, Elias Koutsoupias: Three-Processor Tasks Are Undecidable. SIAM J. Comput. 28(3): 970-983 (1999).

[12] 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, August 1997.