Toward Parameterized Verification of Synchronous Distributed Applications

Sagar Chaki James Edmondson
Carnegie Mellon University, Pittsburgh, PA, USA
{chaki,jredmondson}@sei.cmu.edu

ABSTRACT

We present preliminary results on parameterized verification of distributed applications that assume a synchronous model of computation. Our theoretical results are negative – the problem is undecidable even if each node has a single bit of non-determinism and the property is a 1-index safety property. Further, even if each node is completely deterministic, and the property is again a 1-index safety, parameterized verification cannot be solved via the cutoff method. Empirically, we show how to encode such applications as Array-Based Systems and verify them using existing model checkers. We demonstrate this approach on protocols for distributed mutual exclusion and collision avoidance.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms

Verification

Keywords

Paramterized verification, synchronous systems, array-based systems

1. INTRODUCTION

Distributed applications play crucial, often unseen, roles in our lives. There is also a growing need to incorporate them into safety-critical domains. For example, there are US mandates to incorporate communicating autonomous vehicles on roadways[1]. Indeed, researchers have already started developing intersection protocols [4] that rely on vehicle-to-vehicle communication. Bugs in such distributed protocols and applications have the potential to cause not only financial damage, but also injury and loss of human lives. Consequently, verifying the safety of such distributed protocols and applications before deployment in the real-world has tangible monetary and public safety benefits.

In this paper, we focus on the verification of *synchronous* distributed applications (SDAs). In such applications, each node executes in rounds, and messages sent (or variables written to) by a node in round r are visible to other nodes in round $r+1$. SDAs have been studied widely in the literature [13]. They are also easier to design and verify compared to asynchronous applications. For example, the use of PALS [1] – a "synchronizer" protocol in the hard real-time domain – has been shown to reduce verification time [14] of an avionics protocol from 35 hours to 30 seconds.

We assume shared memory based communication. Specifically, an SDA instance consists of $n \in \mathbb{N}$ nodes. Each node N_i has access to its own unique id $i \in [1, n]$, and a local copy of an array GV with n elements. Each element of GV is a bit-vector of known fixed width W. In each round r, N_i computes a new value of $GV[i]$ based on the current value of all elements of GV. This value is propagated to the other copies of GV prior to start of round $r+1$ by the underlying communication infrastructure. Note that the element $GV[i]$ is modified only by node N_i. Initially, the first Z (for some known $Z \in [0, W]$) bits of each array element are assigned non-deterministically, and the remaining bits are set to $\perp$.

As part of ongoing work [5], we have developed a domain-specific language, called DASL, for programming SDAs and safety requirements. We have also developed a verifying compiler for DASL programs that handles finite instantiations (i.e., when n is fixed and known) using model checking. In this paper, we focus on the problem of parameterized verification of SDAs – i.e., proving their correctness for arbitrary number of nodes. We present the following results:

1. We show that parameterized verification of SDAs is undecidable, even when $Z = 1$, and the property is a 1-index safety property [8], i.e., of the form $\forall i : G(\phi(i))$. Thus, parameterized verification of synchronous systems is as hard as asynchronous ones even though non-parameterized verification is more tractable [14].

2. We show that even for $Z = 0$ (i.e., deterministic SDAs) and 1-index safety properties, parameterized verification cannot be solved via the cutoff method [8].

3. We present preliminary experimental results on verify-

[1] `http://www.nhtsa.gov/About+NHTSA/Press+Releases/2014/USDOT+to+Move+Forward+with+Vehicle-to-Vehicle+Communication+Technology+for+Light+Vehicles`

SPIN '14, July 21–23, 2014, San Jose, CA, USA

ing SDAs by translating them into Array-Based Systems [10] (ABS). To this end, we present an encoding of the synchronous semantics of SDAs over the asynchronous semantics of ABS. We experimented with two ABS verification tools – MCMT [11] and CUBICLE [6]. Both tools were able to verify our simplest examples easily, but failed on more complex ones.

Related Work. Verification of parameterized systems has been widely studied. Due to limited space, we only touch upon some key points. One view of such systems is as a set of automata communicating over a network. Emerson and Namjoshi [8] present cutoff results in the case of ring networks. Delzanno et al. [7] present decidability and undecidability results for a range of network topologies with and without broadcast using the formalism of well-structured transition systems [9]. A good survey of work in this area, and new results, is provided by Aminof et al. [3].

Another view of parameterized systems, closer to ours, is that of an Array-Based System (ABS) [10]. An ABS consists of a set of arrays, and guarded transitions. In each step, one of the enabled transitions is applied to update the array. Such systems have received increased attention over the last few years, and new decision procedures and model checkers [11, 6] have been developed for verifying them. By encoding SDAs as ABSs, we are able to build on this work.

Verification of distributed algorithms is traditionally done manually [13] using invariants, simulation relations etc. John et al. have recently developed automated parameterized verification techniques for fault-tolerant distributed algorithms [12]. They assume asynchronous semantics.

The rest of this paper is organized as follows. Sec. 2 presents preliminary concepts. Sec. 3 presents our theoretical results. Sec. 4 presents experimental results, and concludes.

2. PRELIMINARIES

Formally, a SDA P is a 4-tuple (GV, W, Z, ρ) where: (i) GV is the global array; (ii) each element of GV is a bitvector of width $W \in \mathbb{N}$; (iii) $Z \in [0, W]$ is the number of nondeterministic bits available to each node; and (iv) ρ is a procedure executed by each node in every round.

Syntax. Let IV be a set of id variables, and id be a distinguished variable such that $\{GV\}$, IV and $\{id\}$ are mutually disjoint. The body of ρ is a statement. The "abstract" syntax of statements, lvalues and expressions is given by the following BNF grammar (w is an integer in $[1, W]$):

$$\begin{array}{rcl} stmt & := & skip \mid lval = exp \mid \text{ITE}(exp, stmt, stmt) \\ & \mid & \text{ALL}(IV, stmt) \mid \langle stmt^+ \rangle \\ lval & := & GV[id][w] \\ exp & := & \top \mid \bot \mid lval \mid GV[IV][w] \mid id \mid IV \mid \diamond(exp^+) \end{array}$$

Intuitively, *skip* is a nop, $lval = exp$ is an assignment, ITE is an "if-then-else", $\text{ALL}(v, st)$ executes st iteratively by substituting v with the id of each node, $\langle st_1 ; \ldots ; st_k \rangle$ executes st_1 through st_k in sequence, and $\diamond$ is an operator. ALL enables iteration over all nodes of P without knowing the exact number of such nodes a-priori. IV and id are natural numbers. Statements are well-typed and variables are well-scoped.

Semantics. The instance of SDA P with $n \in \mathbb{N}$ nodes is denoted $P(n)$. Let $A(n)$ be the set of arrays with n elements, each a W-wide bitvector. The semantics of $P(n)$ is the state transition system (S, I, R) where: (i) $S = A(n)$; (ii) $I = \{a \in S \mid \forall i \in [1, n] \,.\, \forall j \in (Z, W] \,.\, a[i][j] = \bot\}$; and $R \subseteq S \times S$ is the relation such that $(s, s') \in R$ iff for all $i \in [1, n]$, $s'[i]$ is the final value of $GV[i]$ if ρ is executed from the state $GV = s \wedge id = i$. Let R^* denote the transitive closure of R.

Sequentialization. Note that $P(n)$ is semantically equivalent to a sequential program $[\![P(n)]\!]$ that maintains two copies of GV – GV_0 and GV_1 – and executes iteratively. Initially, $GV_0 \in I$. In each iteration, for each node N_i, it executes ρ by reading from GV_0 and writing to $GV_1[i]$. After all nodes have been processed, it copies GV_1 back to GV_0 and proceeds with the next iteration. This observation is crucial for modeling and verifying SDAs as Array-Based Programs.

Specification. A specification ϕ is a formula $\forall i \,.\, \Psi(i)$ where $\Psi(i)$ is an expression with the following grammar:

$$exp := * \mid \top \mid \bot \mid lval \mid GV[i][w] \mid \diamond(exp^+)$$

Let the semantics of $P(n)$ be (S, I, R). We say that $P(n)$ satisfies ϕ, denoted $P(n) \models \phi$ iff $\forall a \in A(n) . \forall a_I \in I . (a_I, a) \in R^* \implies \forall j \in [1, n] \,.\, (a, \Psi(j)) = \top$ where $(a, \Psi(j))$ is the evaluation of $\Psi(i)$ after each $GV[i][w]$ has been replaced with $a[j][w]$.

Parameterized Model Checking. The input to the parameterized model checking problem is a SDA P and a specification ϕ. Its output, denoted $\text{PARMODCK}(P, \phi)$ is $\bot$ if $\exists n \in \mathbb{N} \,.\, P(n) \not\models \phi$, and $\top$ otherwise.

3. THEORETICAL RESULTS

We now show that the parameterized model checking problem is undecidable by reducing the Post Correspondence Problem [15] to it. Initially, we assume $Z \in [1, W]$. Subsequently, we show that the undecidability holds even if $Z = 1$.

Post's Correspondence Problem. Let Σ be an alphabet with at least two letters. An instance I of PCP is given by two sequences $U = \langle u_1, \ldots, u_m \rangle$ and $V = \langle v_1, \ldots, v_m \rangle$ of strings $u_i, v_i \in \Sigma^+$. The output, denoted $\text{PCP}(I)$, is $\bot$ if there exists a finite non-empty sequence (known as the solution) $\langle i_1, \ldots, i_p \rangle$ with $i_j \in [1, m]$ for $j \in [1, p]$ such that:

$$u_{i_1} \bullet \ldots \bullet u_{i_p} = v_{i_1} \bullet \ldots \bullet v_{i_p}$$

where $\bullet$ is string concatenation. PCP is undecidable [15].

For example, suppose $U = \langle a, ab, bba \rangle$ and $V = \langle baa, aa, bb \rangle$. Then $\text{PCP}(I) = \bot$ since there exists a solution $\langle 3, 2, 3, 1 \rangle$. This is because $u_3 \bullet u_2 \bullet u_3 \bullet u_1 = bbaabbbaa = v_3 \bullet v_2 \bullet v_3 \bullet v_1$. On the other hand, if $U = \langle aa, aab, baaa \rangle$ and $V = \langle a, bb, abb \rangle$, then $\text{PCP}(I) = \top$ since there is no solution (each u_i has a bigger length than the corresponding v_i).

We show that for every instance I of PCP, there exists a SDA P and specification ϕ such that $\text{PARMODCK}(P, \phi) = \text{PCP}(I)$. For convenience, we assume that each element of GV is a record whose fields are finite datatypes. Specifically, there are five fields: (i) *idu*, *posu*, *idv*, and *posv* are initialized non-deterministically; and (ii) *st* is initialized to 0. For simplicity, we write $f[i]$ to mean $GV[i].f$ where f is a field.

At a high level, the SDA implements a protocol that works as follows. Variable $st[i]$ indicates the overall progress of N_i in the protocol. Node N_1 is a special node that helps in detecting success. Every other node $N_i, i > 1$ represents two letters: (i) the $posu[i]$-th letter in $u_{idu[i]}$, and (ii) the $posv[i]$-th letter in $v_{idv[i]}$. The protocol succeeds iff the sequence of

if $(st[id] = 0)$
  if $(id = 1)$ $st[id] := 2;$
  else if $(1 \leq idu[id] \leq m \wedge 1 \leq posu[id] \leq |u_{idu[id]}| \wedge$
    $1 \leq idv[id] \leq m \wedge 1 \leq posv[id] \leq |v_{idv[id]}| \wedge$
    $u_{idu[id]}[posu[id]] = v_{idv[id]}[posv[id]])$
    $st[id] := 2;$
  else $st[id] := 1;$
else if $(st[id] = 2)$
  if$(id \neq 2 \vee (posu[id] = 1 \wedge posv[id] = 1))$ $st[id] := 3;$
  else $st[id] := 1;$
else if $(st[id] = 3)$
  if$(id \leq 2 \vee (X_1 \wedge X_2 \wedge X_3 \wedge X_4))$ $st[id] := 4;$
  else $st[id] := 1;$

where:

$$
\begin{aligned}
X_1 &:= posu[id] > 1 \implies (idu[id-1] = idu[id] \\
&\quad \wedge posu[id-1] = posu[id] - 1) \\
X_2 &:= posv[id] > 1 \implies (idv[id-1] = idv[id] \\
&\quad \wedge posv[id-1] = posv[id] - 1) \\
X_3 &:= posu[id] = 1 \implies posu[id-1] = |u_{idu[id-1]}| \\
X_4 &:= posv[id] = 1 \implies posv[id-1] = |v_{idv[id-1]}|
\end{aligned}
$$

Figure 1: Code Fragment for First Three Rounds.

letters represented by $\langle N_2, \ldots, N_n \rangle$ is a solution to PCP(I). The procedure ρ works as follows (the code is in Figure 1):

1. First ensures that the letters represented are valid and identical. This is the case *if* $(st[id] = 0)$ in Figure 1. Note that $st[id] = 1$ means no further progress.

2. Next checks that N_2 represents the first letter of a word. This is the case *if* $(st[id] = 2)$ in Figure 1.

3. Next checks that the node before it represents either the previous letter in the same word or the last letter in another word, as appropriate. This is the case *if* $(st[id] = 3)$ in Figure 1.

4. The final stage of ρ – denoted EqCheck– checks that the nodes represent the same sequence of indices in both U and V. Let the sequence of indices in U represented by the nodes be $\langle iu_1, \ldots, iu_q \rangle$, and the sequence of indices on V represented by the nodes by $\langle iv_1, \ldots, iv_r \rangle$. Starting with node N_1, the program first computes iu_1 and iv_1 and compares them, then computes iu_2 and iv_2, and compares them, and so on. The protocol succeeds iff $\langle iu_1, \ldots, iu_q \rangle = \langle iv_1, \ldots, iv_r \rangle$. Success is indicated by N_1 entering a special OK state. We next define EqCheck in more detail.

EqCheck*: High-Level Idea.* Node N_1 sends out two tokens – tu and tv – along the line $\langle N_1, \ldots, N_n \rangle$. If node N_i receives tu there are two cases:

1. If N_i does not represent the last letter of its corresponding U-word (i.e., if $posu[i] \neq |u_{idu[i]}|$), it passes tu to N_{i+1} and moves to the DONE state.

2. If N_i represents the last letter of its U-word (i.e., if $posu[i] = |u_{idu[i]}|$), it waits for N_1 to move to a special GREEN state. In any subsequent round, if N_i detects that N_1 is in the GREEN state, it passes tu to N_{i+1} and moves to the DONE state.

else if $(st[id] = 4)$
  if $(id = 1)$ $tu[id] := 1; tv[id] := 1;$
  else $tu[id] := 0; tv[id] := 0; wtu[id] := 0; wtv[id] := 0;$
  $st[id] =$ INIT;
else if $(st[id] =$ INIT$) \wedge (id = 1)$
  if $(EXISTS(iu, wtu[iu] = 1) \wedge EXISTS(iv, wtv[iv] = 1))$
    if $(iu = iv)$ $st[id] :=$ OK;
    else if $(idu[iu] = idv[iv])$ $st[id] :=$ GREEN;
else if $(st[id] =$ INIT$) \wedge (id \neq 1)$
  if $(wtu[id] \wedge st[1] =$ GREEN$)$
    $wtu[id] := 0; tu[id] := 1; st[id] :=$ DONE;
  else if $(tu[id-1] = 1 \wedge tu[id] = 0)$
    if $(posu[id] = |u_{idu[id]}|)$ $wtu[id] := 1;$
    else $tu[id] := 1; st[id] :=$ DONE
  if $(wtv[id] = 1 \wedge st[1] =$ GREEN$)$
    $wtv[id] := 0; tv[id] := 1; st[id] :=$ DONE;
  else if $(tv[id-1] = 1 \wedge tv[id] = 0)$
    if $(posv[id] = |v_{idv[id]}|)$ $wtv[id] := 1;$
    else $tv[id] := 1; st[id] :=$ DONE
else if $(st[id] =$ GREEN$)$ $st[id] :=$ INIT;

Figure 2: Code Fragment for EqCheck.

A symmetric behavior (involving $v_{idv[i]}$ and $posv[i]$) occurs if N_i receives token tv. Note that N_i may receive both tu and tv in the same round. Node N_1 moves to the GREEN state (for just one round) if it detects that both tokens have reached the ends of words from U and V with equal indices. Specifically, suppose tu is with node N_{iu} and tv is with node N_{iv}. Then, N_1 moves to the GREEN state for just one round if the following condition holds:

$$
\begin{aligned}
posu[iu] = |u_{idu[iu]}| \wedge posv[iv] = |v_{idv[iv]}| \\
\wedge GV[iu].idu = GV[iv].idv
\end{aligned}
$$

Also, N_1 moves to the OK state if, in addition to the above condition, $iu = iv$ holds as well.

EqCheck*: Implementation Details.* To simulate token passing with shared variables, we introduce two extra Boolean fields tu and tv in each element of GV. Then, node N_i has token tu (or tv) iff i is the largest index such that $tu[i-1] = \top$ (or $tv[i-1] = \top$). Thus, N_i passes tu (or tv) to N_{i+1} by setting $tu[i]$ (or $tv[i]$) to $\top$. We also have two more Boolean fields wtu and wtv to indicate that the node has token tu or tv and is waiting for N_1 to move to the GREEN state. In the first round of EqCheck, tu, tv, wtu and wtv are initialized appropriately. The code for EqCheck is shown in Figure 2. Note that *EXISTS* can be implemented using ALL and additional variables.

Consider the specification $\phi = \forall i \,.\, st[i] \neq$ OK. It can be shown that PARMODCK$(P, \phi) =$ PCP(I). Thus we have:

THEOREM 1. PARMODCK(P, ϕ) *is undecidable.*

We now show that PARMODCK(P, ϕ) is undecidable even if $Z = 1$. Specifically, for any SDA $P = (GV, W, Z, \rho)$ where $Z > 1$, there exists another SDA $\tilde{P} = (\tilde{GV}, \tilde{W}, 1, \tilde{\rho})$, such that $\forall n \in \mathbb{N}$, $P(n)$ is simulated by $\tilde{P}(Z \times n)$. In the first round, every Z-th node of $\tilde{P}(Z \times n)$ copies the NDBs from its next $Z-1$ neighbors. Now every Z-th node has Z NDBs. Note that there are n such nodes. Subsequently, node N_i of $\tilde{P}(Z \times n)$ simulates node N_j of $P(n)$ iff $i = (j-1) \times Z + 1$. This implies the next result.

THEOREM 2. PARMODCK(P, ϕ) *is undecidable even when* $P = (GV, W, 1, \rho)$.

Example	MCMT			CUBICLE		
	Time	N	SMT	Time	N	SMT
MUTEX-OK	0.46	21	1158	0.61	668	614
MUTEX-BUG1	1.2	70	2630	1.8	-	-
MUTEX-BUG2	0.52	48	1350	0.45	-	-
COLL-OK	*	*	*	*	*	*
COLL-BUG1	253	1045	195K	248	-	-
COLL-BUG2	*	*	*	2892	-	-

Figure 3: Experimental Results. Times are in second; N = no. of nodes; SMT = no. of calls to SMT solver; - = data not reported; * = timeout.

The cutoff approach [8] for parameterized verification is based on proving that for a certain class of specifications ϕ and parameterized system P, there exists a known $K \in \mathbb{N}$ such that $\forall n > 0 \,.\, P(n) \models \phi \iff \forall n \leq K \,.\, P(n) \models \phi$. We now show that for SDAs, no such cutoff can exist even if each node is completely deterministic.

THEOREM 3. *For each $K \in \mathbb{N}$, there exists a specification ϕ and a SDA P with $Z = 0$ such that $\forall n \leq K \,.\, P(n) \models \phi \wedge P(K+1) \not\models \phi$.*

PROOF. Consider the SDA P where each element of GV has one field st initialized to 0, and following function ρ: *if* $(id > K)$ $st[id] := 2$; *else* $st[id] := 1$. Let $\phi = \forall i . st[i] \neq 2$. Thus, $\forall n \leq K \,.\, P(n) \models \phi \wedge P(K+1) \not\models \phi$. □

4. EXPERIMENTAL RESULTS

Recall that any SDA instance $P(n)$ is semantically equivalent to a sequential program $[\![P(n)]\!]$ that operates over two copies of the array GV. We now show that $[\![P(n)]\!]$ can be encoded as an Array-Based System (ABS). The main challenge is that in each iteration $[\![P(n)]\!]$ processes every array element, while in an ABS, only one enabled transition is executed asynchronously in each step. Our solution is to: (i) implement a "barrier" using "universal guards" [2]; (ii) use the barrier to implement synchronicity via a protocol modeled after "two-phase commit". Due to lack of space, we are unable to provide further details. However, all our examples are available at `http://www.contrib.andrew.cmu.edu/~schaki/misc/spin14.tgz`.

We experimented with two sets of examples – mutual exclusion and collision avoidance between mobile robots. Both protocols use ordering between ids to ensure that at most one node is in the critical section and no two nodes are in the same physical location (a coordinate on a two-dimensional grid). Mutual exclusion requires reserving and acquiring a single lock. Collision avoidance is more complicated and requires two locks – one for a node's current location and the other for the location the node is moving to.

For each example, we created three versions – one correct and two buggy by omitting crucial checks in the protocol. We manually translated each version into the input language of two ABS model checkers – MCMT v2.5 and CUBICLE v0.5. We then applied the two tools – using their default settings – on their corresponding example files. All experiments were done on a 2.3GHz Machine running 64bit Linux with a time limit of 120 minutes and a memory limit of 4GB. Our experimental results are shown in Fig. 3. Both MCMT and CUBICLE performs symbolic backward reachability using an SMT solver and heuristics to prune out unfeasible executions, and detect fixed points. The results indicate that they are effective on all mutual exclusion examples and the buggy collision avoidance examples. However, the safe collision avoidance example is beyond the scope of both tools.

Conclusion. We presented preliminary results and caveats toward parameterized verification of SDAs. We are exploring several next steps: (i) formally defining and proving correctness of the translation from SDAs (written in DASL) to ABSs; (ii) developing verification algorithms that operate on SDAs directly instead of converting them to ABSs; (iii) implementing a robust parameterized model checker for SDAs; and (iv) performing a more comprehensive evaluation.

Acknowledgments. Copyright 2014 ACM. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0001199

5. REFERENCES

[1] A. Al-Nayeem, M. Sun, X. Qiu, L. Sha, S. P. Miller, and D. D. Cofer. A Formal Architecture Pattern for Real-Time Distributed Systems. In *Proc. of RTSS*, 2009.

[2] F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Universal Guards, Relativization of Quantifiers, and Failure Models in Model Checking Modulo Theories. *JSAT*, 8(1/2), 2012.

[3] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized Model Checking of Token-Passing Systems. In *Proc. of VMCAI*, 2014.

[4] S. R. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige. Reliable intersection protocols using vehicular networks. In *Proc. of ICCPS*, 2013.

[5] S. Chaki and J. Edmondson. Model-Driven Verifying Compilation of Synchronous Distributed Applications, 2014. under submission.

[6] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems - Tool Paper. In *Proc. of CAV*, 2012.

[7] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized Verification of Ad Hoc Networks. In *Proc. of CONCUR*, 2010.

[8] E. A. Emerson and K. S. Namjoshi. On Reasoning About Rings. *IJFCS*, 14(4), 2003.

[9] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2), 2001.

[10] S. Ghilardi and S. Ranise. Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis. *LMCS*, 6(4), 2010.

[11] S. Ghilardi and S. Ranise. MCMT: A Model Checker Modulo Theories. In *Proc. of IJCAR*, 2010.

[12] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *Proc. of FMCAD*, 2013.

[13] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[14] S. P. Miller, D. D. Cofer, L. Sha, J. Meseguer, and A. Al-Nayeem. Implementing logical synchrony in integrated modular avionics. In *Proc. of DASC*, 2009.

[15] E. L. Post. A variant of a recursively unsolvable problem, 1946. Bull. Amer. Math. Soc 52.