Logics for reasoning about cryptographic constructions

Russell Impagliazzo*
Department of Computer Science and Engineering
University of California, San Diego
La Jolla CA 92093
russell@cs.ucsd.edu

Bruce M. Kapron†
Computer Science Department
University of Victoria
Victoria, BC
CANADA V8W 3P6
bmkapron@cs.uvic.ca

April 5, 2005

Abstract

We present two logical systems for reasoning about cryptographic constructions which are *sound* with respect to standard cryptographic definitions of security. Soundness of the first system is proved using techniques from nonstandard models of arithmetic. Soundness of the second system is proved by an interpretation into the first system. We also present examples of how these systems may be used to formally prove the correctness of some elementary cryptographic constructions.

1 Introduction

We present two formal deduction systems that can prove cryptographic constructs secure via the standard of complexity-theoretic cryptography: by giving a reduction between the security of the construct and that of the underlying primitives. Proofs in both systems avoid unnecessary quantifiers, so they may eventually be helpful in machine-assisted verification of protocol correctness. The second is simpler, but narrower in application, being usable only to reason about computational indistinguishability. We prove soundness of the more specific system by interpreting it in the general system. In fact, the main application of the general system may be as a "meta-logic" for proving soundness of logics of security.

Motivation. Correctness in security is even more important than for general systems, since attackers will actively seek out and exploit any defect. Unfortunately, it seems also to be even more difficult to design correct cryptographic protocols than to design correct programs. This is because one needs to think of the behaviour of the protocols not just as they are intended to be used, but under any feasible strategy for an attacker.

A central goal of cryptography is to construct functions and protocols solving relatively complex tasks, such as authentication or pseudo-random generation, from simpler building blocks known as cryptographic primitives. Protocols and functions constructed in this way may themselves be used as primitives in more complex constructions. The security of the constructed object depends on both the security of the underlying primitives and the way in which these primitives are used in a construction. A flaw in either the primitive or the construction can make the constructed function vulnerable to attack. In the absence of strong lower bounds on complexity, we cannot absolutely prove that any such constructed functions are secure. However, we would like to certify that the construction itself does not introduce weaknesses that were not present in the primitives used.

There have been several approaches to validating that a cryptographic construction does not introduce weaknesses. The standard in the complexity-theory based cryptographic community is the reduction [1]. A reduction gives a way of

*Research partially supported by NSF Award CCR-0098197 and USA-Israel BSF Grant 97-00188.

†Research partially supported by NSERC Research Grant OGP0138744 and funding from the MITACS NCE. Both authors received partial support from DoD MURI "Semantic Consistency in Information Exchange", ONR Grant N00014-97-1-0505, at Stanford University while completing some of this work

[1]In this paper, we focus primarily on complexity-based, as opposed to information-theoretic security, which does not rely on the reduction paradigm. However, some of the tools we introduce are in fact applicable to proofs of information-theoretic security.

translating a successful attack on the protocol or construction into a successful attack on the primitives used. An important part of proving cryptographic security is precisely defining what constitutes an attack both on the construction and on the primitives. This method has the advantages of clearly identifying both the security properties of the construct and the assumptions made about the primitive. In addition to providing a conceptual foundation for cryptography, reductions can provide precise, quantitative tools for analyzing specific protocols ([4]).

However, it is not clear that systems built using secure cryptographic primitives will be themselves secure, due to flaws that may be introduced in the protocol or construction which uses the primitves. First, seemingly trivial modifications in the protocol can cause it to become totally insecure. Secondly, the cryptographic protocol itself may be only one component of a much more complex communications protocol that needs to be verified as a whole. Finally, the security properties guaranteed need to be transparent to system builders who wish to incorporate the new tool in their work. It is important to provide tools for specification and verification of secure protocols that are understandable to programmers and can be incorporated into existing methodologies of protocol verification. For these reasons, much research has gone into developing formal methods for the security analysis of protocols.

Most work on the formal verification of security protocols (e.g., [8, 15, 1]) tends to deal with security more abstractly than in complexity-based approaches. Instead of asking, “If the construct is insecure, is the primitive insecure?” these verification methods ask, “If the primitive is secure in an ideal way, is the construct secure?”. Application of a primitive is assumed to produce a piece of atomic data without internal structure, which limits the power of an adversary. On the other hand, so called Dolev-Yao assumptions [15] grant adversaries more power by allowing nondeterministic choice between possible attacks. As a result of this idealized approach, it is quite possible that a protocol proved correct in the formal system is insecure when instantiated using a secure primitive. However, these logics have been quite successful in detecting flaws in protocols, and as a basis for automated tools to assist in protocol analysis ([28]). This is largely due to their having simple, intuitive rules that avoid quantifiers and explicit reasoning about probabilities.

Recently, a number of papers have proposed ways to bridge this gap between complexity-theory and logics for security. The papers [21] and [25] give a syntactic characterization of a kind of cryptographic security via equivalence in a probabilistic process calculus. Abadi and Rogaway [2] present a formal system for reasoning indistinguishability of expressions built using a pseudo-random block cipher. They show that this logic is sound in the standard sense: any algorithm for distinguishing two expressions proved indistinguishable can be converted into an attack on the block cipher. (See also [23] for a corresponding completeness result, and [24], [22] for soundness results in the presence of active adversaries.) *Universal composability* (e.g., [10], [11] and related concepts (e.g., [3]) aim to produce cryptographic primitives that are “equivalent to the ideal primitive” for a variety of applications. It would be interesting to explore whether protocols proved secure by formal methods are secure when instantiated with sufficiently strong primitives or in the random oracle model.

In this paper, we take a more direct route to merge these two areas. We present two logics for security that are simplified formalizations of the language and methods of modern cryptography. Of course, all papers in complexity theory are formalizable in a sufficiently strong logical system, such as Peano arithmetic. The challenge is coming up with a formal system that is sound in the above sense, powerful enough to handle a significant fraction of constructions, and simple enough to be a useful tool for verifying that security is maintained as protocols are modified and combined in implementations. The difficulties involved in formulating such a system include: the formulation of security definitions, the use of probabilities in definitions of security, reasoning about random choices and distributional problems, quantifying the computational power of adversaries, and intuitive but dangerous mis-applications of induction. An illustrative example for this last difficulty is given in Section 2.

It is the aim of this paper to show that it is possible to formulate logical systems that handle these issues implicitly. We combine ideas from cryptography, implicit complexity, and proof complexity. By having a term algebra (such as in Cook’s PV [13]) represent feasible functions, quantification over functions implicitly defines the scope of feasible adversaries. A counting quantifier is added to allow simple probabilistic reasoning. Only “small” numbers (polynomial length) can be proved to exist, so one can implicitly define negligible vs. non- negligible probabilities and limit the scope of induction arguments. By eliminating asymptotics and explicit resource bounds, we make proofs largely quantifier-free, consisting of equations and inequalities. The logic is sound, in the standard sense: any security proof for a construction of one cryptographic object from another provides a reduction from an algorithm that breaks the constructed object to one that breaks the underlying primitive.

Soundness is proved using non-standard models of arithmetic, but knowledge of such models is unnecessary for users – all that is required is an intuition regarding “relatively large” values. A similar situation occurs in the setting of nonstandard analysis. It is possible to give an elementary undergraduate presentation of calculus using infinitesimals (see, e.g., [19]). Proving soundness of the axiomatization of infinitesimals requires use of nonstandard models. However, it is not essential to understand the soundness proofs in order to use these systems.

The first logic we introduce is rather general and powerful, but still a little cumbersome. We show how it can be specialized to provide simpler logics for limited types of reasoning. In particular, we give a simple, sound logic for reasoning about computational indistinguishability. The soundness proof for this system follows directly from that of our more general system, by interpreting the specific system in the general one. Indeed, we think the main application of the general system may be as a "meta-logic", to prove the soundness of more specialized logics for security.

Our work is a preliminary step toward provably sound, practical formal methods for security analysis. We believe that the primary purpose of formal methods in cryptography is not to make it easier for cryptographers to produce valid proofs, but to clarify the difference between valid and invalid proofs. By restricting the kind of proofs allowed, we eliminate many forms of common mistakes that, to the non-expert, appear indistinguishable from a valid argument. On the other hand, we feel our logic is both rich enough that it is possible, and simple enough that it is not overly burdensome, to formalize most proofs in our logic. These proofs could be used by implementers to distinguish harmless and harmful modifications and combinations of protocols. Also, it seems possible that the symbolic techniques we are proposing may scale up, with machine-assistance, to provide verification tools for much larger systems which incorporate cryptographic tools.

Our focus is primarily on constructions of cryptographic functions, rather than protocols. However, we will also demonstrate how reasoning about simple protocols is supported in our framework by showing how to model and verify a protocol for flipping coins on a public line [5], which uses the hidden bit commitment protocol of Naor [26] as a subroutine.

2 A fallacious induction argument

We begin with a fallacious induction proof to demonstrate the potential pitfalls of induction applied to cryptographic constructions. By an injudicious application of induction we will prove the correctness of a construction of a pseudo-random generator based on the iteration of Goldreich-Levin style construction of a hard-core predicate. However, we also show that the function which is the result of the construction is not one-way, yielding a contradiction.

Notation. In this section we will use the following notation. Let $\{0,1\}^*$ denote the set of all strings over $\{0,1\}$, and $\{0,1\}^n$ strings of length n. For $a \in \{0,1\}^n$ and $b \in \{0,1\}^m$, $a \circ b \in \{0,1\}^{m+n}$ is the concatenation of a and b. For $a, b \in \{0,1\}^n$, $a \odot b$ denotes mod 2 inner product of a and b (i.e., when a and b are viewed as bit vectors.) For $a \in \{0,1\}^n$, and $1 \leq i < j \leq n$, a_i denotes the ith bit of a and $a_{\{i\ldots j\}}$ denotes $a_i \circ \cdots \circ a_j$. If A is any set, $\#A$ denotes the number of elements in A.

A function f is *one-way* if it is easy to compute f but any computationally bounded adversary has no more than a negligible chance of finding a preimage for $f(X)$ when X is chosen uniformly at random from $\{0,1\}^n$, for sufficiently large n. A 0-1 valued function b is a hard-core predicate for a function f if it is easy to compute b, but any computationally limited adversary has a no better than negligible chance of doing better than guessing $b(X)$ when given $f(X)$, if X is chosen uniformly at random from $\{0,1\}^n$ for sufficiently large n. A function $f : \{0,1\}^n \to \{0,1\}^m$ is a *pseudo-random generator* if any computationally limited adversary has no more than a negligible chance of distinguishing $f(X)$ from Y chosen uniformly at random from $\{0,1\}^m$, whenever X is chosen uniformly at random from $\{0,1\}^n$. The *stretch of* f is $m - n$.

The *Goldreich-Levin theorem* [18] states that if f is any one-way function, then there is a one-way function g defined from f and a 0-1 valued function b such that b is a hard-core predicate for g. In particular, $g(x \circ r) = f(x) \circ r$ and $b(x \circ r) = x \odot r$. Note that we are assuming x and r have the same length. An immediate application of the Goldreich-Levin theorem is a method for constructing a pseudo-random generator with stretch 1 from a one-way permutation. If f is a permutation then so is g, so that $g(x \circ r)$ is uniformly distributed and b is a hard-core predicate for g. Let $f'(x \circ r) = g(x \circ r) \circ b(x \circ r)$, so that f' is pseudo-random (with stretch 1). This f' is also one-way, and so we can repeat this process to obtain another hard-core predicate.

Can we iterate this construction to obtain a pseudo-random generator with stretch n? We now give a fallacious argument that this is the case. Suppose that $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way permutation. We define a sequence of functions $f_0, \ldots, f_n$, where $f_i : \{0,1\}^{n+n^2} \to \{0,1\}^{n+n^2+i}$ as follows: $f_0(x \circ \vec{r}) = f(x) \circ \vec{r}$, and for $0 < i \leq n$, $f_i(x \circ \vec{r}) = f_{i-1}(x) \circ (x \odot r_i)$, where $\vec{r}$ denotes $r_1 \circ \cdots \circ r_n$. Now we have that for each i, $0 < i \leq n$, if f_{i-1}

is pseudo-random, then so is f_i, by an argument similar to that given above, with a slightly modified version of the Goldreich-Levin theorem. But if f is a one-way permutation then f_0 is pseudo-random. Thus we can fallaciously conclude by "induction" that f_n is pseudo-random.

Is f_n really pseudo-random? If it were, it would also be one-way. Now $f(x \circ \vec{r})$ has the form $f(x) \circ \vec{r} \circ \vec{b}$, where $\vec{b} = b_1 \circ \cdots \circ b_n$ and $b_i = x \odot r_i$. This gives rise to the following system of equations over GF(2):

$$\sum_{j=1}^{n} x^j r_i^j = b_i \qquad (1 \leq i \leq n)$$

With non-negligible probability (over the choice of $\vec{r}$) this system has full rank. Then we can efficiently recover x from $f(x \circ \vec{r})$ for any x and $\vec{r}$ by solving this system of equations, contrary to the purported one-wayness of f_n.

Where did our "induction" argument go wrong? One way of looking at the problem is that it hides the fact that at stage i, an adversary's advantage, or chance of success, depends on i. This advantage could double at each stage, and still appear to be negligible given the assumption that the advantage at the previous stage is negligible. However, after n iterations the advantage is increased by a factor of 2^n and is no longer negligible. This is analogous to the familiar situation in which composing a feasible function with itself a feasible number of times results in a function which is not feasible (e.g. exponentiation by repeated squaring.) Clearly, any system for reasoning formally about such constructions will require a careful treatment of induction.

3 A general system for cryptographic constructions

Our first formal system, which we call T, is a system of first-order logic which supports reasoning about probabilities, asymptotics and polynomial-time functions. We begin by giving a high-level presentation of what types of reasoning can be done in the general formal system. In the following, we use "polynomial" as shorthand for "relatively short" or "relatively fast". Our system deals with the following types of objects.

1. There is a *security parameter* presented as two constants: $\mathbf{n}$, a "moderate integer", is the security parameter in dyadic; and $\mathbf{s}$, a "string", is the same security parameter in unary, $\mathbf{n} = |\mathbf{s}|$. This security parameter is assumed to be sufficiently large that any particular asymptotically true statement is true of $\mathbf{n}$.

2. *Strings* are objects that could be inputs, outputs, or random tapes for a probabilistic algorithm that runs in time polynomial in the security parameter. $\mathbf{s}$ is a string, the variables $x, y, z...$ represent strings, and any efficient function applied to a string yields a string. Using a standard embedding such as dyadic notation, we can also view strings as integers. Thus the usual arithmetic operations (except exponentiation) and comparison relations make sense for strings. Strings are implicitly assumed to be of polynomial length; and variables must be defined to be of some "moderate integer" length when quantified over.

3. *Moderate integers* are those polynomially bounded in the security parameter. An integer is moderate if and only if it is the length of some string, so we don't need to have new variables of this type. We use $p(\mathbf{n})$ for p a polynomial or $|t|$ for t a string term to denote a moderate integer. Now, for example, we can express that q is negligible function by: for every z and every x with $|x| = \mathbf{n}$, $q(x) \leq \frac{1}{|z|}$, eliminating the alternation of quantifiers needed for the usual asymptotic definition.

4. *Feasible functions* are polynomial-time functions that take strings to strings. They include certain basic functions, such as a pairing function and arithmetic operations. New feasible functions can be defined from others by composition or certain forms of recursion. It is well-known (see [12]) that in this way we are able to define any particular polynomial-time function. There are also function symbols of each arity, $f, g, ...$, which allow us to quantify over functions. However, quantification over strings is not allowed outside a quantification over functions, so functions cannot depend on a specific input. We use $A, A_1, A_2...$ to represent possible adversaries for cryptographic functions, but formally, these are just the same as other function variables.

5. *Counting integers* are integers that represent sizes of sets of strings. If $\varphi(x_1, ..x_k)$ is a quantifier-free formula in string variables $x_1, ...x_k$ (possibly involving function variables) and $t_1, ..t_k$ are moderate integer terms, we can form the counting integer term $\#(|x_1| = t_1 \wedge ... \wedge |x_k| = t_k)\varphi(x_1, ...x_k)$ to represent the number of strings

that satisfy φ of the given sizes. Arithmetic operations make sense for counting integers, and their lengths are moderate integers (in particular the length of $\#(|x| = t)\varphi(x)$ is bounded by t). However, not being readily computable, they are not allowed to be inputs to function symbols or treated as strings. Thus, we can reason about $C(\mathbf{n}) = \#(|x| = |\mathbf{n}|)(0 \leq x \leq \mathbf{n} - 1 \wedge gcd(x, \mathbf{n}) = 1)$, that is the number of values smaller than the security parameter which are relatively prime to it, but we cannot use $C(\mathbf{n})$ as the input to an adversary (unless we can show how to compute it another way).

6. We can use counting integers implicitly to reason about *probabilities*, rational numbers in $[0, 1]$. $\mathrm{Pr}_{|x|=t}\varphi(x) = \frac{\#(|x|=t)\varphi(x)}{\#(|x|=t)(x=x)}$. Since both numerator and denominator are counting integers, and we can simulate arithmetic operations on rationals by operations on their numerators and denominators, we can also reason about probabilities.

Accepting the existence of a fixed parameter $\mathbf{n}$ that satisfies all asymptotically true arithmetic statements, and a distinction between integers that are "polynomially bounded" in $\mathbf{n}$ and those that are not, requires some suspension of disbelief. Our feeling is that this will mainly bother those that are mathematically but not logically sophisticated. The mathematically unsophisticated will accept it unquestioningly, and the logically sophisticated will realize that any particular proof is only using a fixed set of asymptotic facts and polynomial bounds, and so is sound for sufficiently large n. Those in the middle range might take some comfort from the fact that we have proved soundness for our system, and be willing to suspend their disbelief based on this proof.

Note that, in order to consistently represent strings and moderate integers by the same type of variable, we assume that moderate integers are represented in dyadic notation [29], which gives a 1-1 correspondence between numbers and strings over $\{\mathtt{0}, \mathtt{1}\}$ (in particular, the string x corresponds to the number which is one less thanthe number whose binary representation is obtained by prepending $\mathtt{1}$ to x, so $\epsilon \equiv 0, \mathtt{0} \equiv 1, \mathtt{1} \equiv 2, \mathtt{00} \equiv 3, \ldots$). We will try to follow the following convention: when the intended interpretation of a variable is a string, we use $x, y, z, \ldots$, and when the intended interpretation is a (moderate) integer we use $i, j, k, \ldots$. Also, we use $\mathtt{0}$ and $\mathtt{1}$ to represent bits (i.e. the members of the alphabet of binary strings) and 0 and 1 to represent numbers zero and one. We will rely heavily on definitions by *recursion on dyadic notation* and proofs by *induction on dyadic notation*. Note that in the base case of such inductions, we are considering the string ϵ, which corresponds to the integer 0. In the inductive step, we assume the desired result for a string σ and then try to prove it for the notational successors of σ, namely $\sigma \circ \mathtt{0}$ and $\sigma \circ \mathtt{1}$. Note that if σ represents the integer x, then $\sigma \circ \mathtt{0}$ represents $2x + 1$ and $\sigma \circ \mathtt{1}$ represents $2x + 2$.

3.1 Syntax

We work in a language which extends that commonly used in bounded arithmetic (see, e.g., [9]), with several modifications. First of all, our language is *multi-sorted*, meaning that we have variables which vary over a collection of different domains. In particular, we have variables representing moderate integers, along with variables which represent k-ary polynomial time functions for each $k \geq 1$. Also, we introduce a class of terms, called *counting terms* which are needed to formalize reasoning about probabilities. Since counting is ostensibly a non-feasible operation, we need to take some care in defining the contexts in which counting terms may be used. Due to the restrictions on counting terms, and also on contexts in which we may quantify over function variables, as described above, our description of the syntax of T is somewhat involved. It is necessary to give separate definitions for classes of terms and formulas of increasing syntactic complexity, culminating with the class of *terms* and *formulas*.

Variables We have a collection of *integer variables* $x, y, z, \ldots, i, j, k \ldots$. We also have for each $k > 0$, *function variables of arity k* $f^k, g^k, h^k, \ldots, f_1^k, f_2^k, \ldots$.

Informally, integer variable represent strings or moderate integers. We will usually adopt the convention that when we want to emphasize that we are using such a variable as the length of a string we use $i, j, k, \ldots$ rather than $x, y, z, \ldots$. Function variables of arity k represent k-ary feasible functionals.

Vocabulary We work over a vocabulary $L = L_R \cup L_F$ where

$$L_R = \{\leq, =\} \qquad\qquad L_F = \{0, 1, \mathbf{s}, \mathbf{n}, +, \cdot, \otimes, |x|, \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor\} \cup \{\mathsf{Ap}^k \mid k \geq 1\}$$

Informally, the intended interpretation of the symbols in L_F is as follows: $0, 1, +, \cdot, \lfloor \frac{x \dot{-} 1}{2} \rfloor$ are standard arithmetic constants and operations ($\dot{-}$ denotes *cutoff subtraction*, that is $x \dot{-} y = \max\{x - y, 0\}$); $|x|$ denotes the number of bits in the *dyadic* notation of x; and $x \otimes y$ denotes $2^{|x| \cdot |y|}$ (this *smash* function is often denoted # but we need this symbol for other purposes.) The constant $\mathbf{s}$ represents the security parameter (as a unary string), while $\mathbf{n}$ represents the length of this string. Note that 0 and 1 represent *integer* constants (i.e. 0 is the constant with dyadic representation ϵ and 1 is the constant with dyadic representation $\mathsf{0}$.) Ap^k represents *application* for k-ary functions.

Basic terms are defined as follows:

- 0, 1, $\mathbf{s}$ and $\mathbf{n}$ are basic terms.
- Every string variable is a basic term.
- If r and s are basic terms, then so are $r+s$, $r \cdot s$, $r \otimes s$, $|r|$, $\left\lfloor \frac{r \dot{-} 1}{2} \right\rfloor$.
- If $t_1, \ldots, t_k$ are basic terms and f^k is a function variable, then $\mathsf{Ap}^k(f^k, t_1, \ldots, t_k)$ is a basic term.

Typically, we will abbreviate $\mathsf{Ap}^k(f^k, t_1, \ldots, t_k)$ as $f(t_1, \ldots, t_k)$.

Basic fomulas The class of basic formulas are those which may be formed as propositional combinations of atomic formulas involving basic terms. In particular:

- If s and t are basic terms, then $s = t$ and $s \leq t$ are basic formulas.
- If φ and ψ are basic formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$ and $\varphi \equiv \psi$.

Counting terms For any basic formula φ and basic terms $t_1, \ldots, t_k$, we have a *counting term*

$$\#(|x_1| = |t_1| \wedge \cdots \wedge |x_k| = |t_k|)\varphi.$$

The intended interpretation of such a term is just the number of tuples of strings, with component i having length $|t_i|$ which satisfy φ. We sometimes write counting terms as given above in the form $\#(|\vec{x}| = |\vec{t}|)\varphi$, or

$$\# \begin{pmatrix} {\scriptstyle |x_1| = |t_1|} \\ \cdots \\ {\scriptstyle |x_k| = |t_k|} \end{pmatrix} \varphi$$

Terms Note that by introducing #, we have the possibility of terms which represent functions of (potentially) more than poly-time complexity. We do not want to allow terms built up arbitrarily from counting terms. *Terms* are defined as follows:

- Any basic term or counting term is a term.
- If r and s are terms, then so are $r+s$, $r \cdot s$, $r \otimes s$, $|r|$, $\left\lfloor \frac{r \dot{-} 1}{2} \right\rfloor$.

String formulas *Atomic formulas* have the form $s = t$ or $s \leq t$, where s and t are terms. We then have

- Any atomic formula is a string formula.
- If φ and ψ are string formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$ and $\varphi \equiv \psi$.
- If φ is a string formula and x is a string variable, then $\forall x\varphi$ and $\exists x\varphi$ are string formulas.

If x is a string variable and t is a basic term, $\forall x \leq t\varphi$ is an abbreviation for $\forall x(x \leq t \to \varphi)$ and $\exists x \leq t\varphi$ is an abbreviation for $\exists x(x \leq t \wedge \varphi)$.

Formulas

- Any string formula is a formula.
- If φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$ and $\varphi \equiv \psi$.
- If φ is a formula and f is a function variable, then $\forall f\varphi$ and $\exists f\varphi$ are formulas.

Due to the restrictions on syntax, the usual notion of substitution of a term for a free variable in a formula becomes more complicated. To avoid this complication, we will use the notation $\varphi[t/x]$ where φ is a formula and t is a *basic* term, to denote the substitution of t for all free occurrence of x in φ', where φ' is φ with bound variables renamed to prevent the capture of any free variables in t. Note we also sometimes write $\varphi(\vec{x})$ to indicate that $\vec{x}$ are free variables of φ. In this case, we may also write $\varphi(\vec{t})$ instead of $\varphi[t_1/x_1] \ldots [t_k/x_k]$.

3.2 Axiomatization of the general system

The axioms of the system fall into several groups: *logical axioms*, *BASIC axioms*, *security parameter axioms*, *poly-time function axioms*, *counting axioms* and an *induction axiom*. Note that when we write an axiom as an open formula, we are (implicitly) quantifying universally over all the free variables.

Logical axioms. We begin with an axiomatization of multi-sorted first-order logic with equality [16]. Of course, we need to introduce restrictions which force substitution instances of axioms, as well as conclusions of rules to result in formulas as we have defined above.

Security parameter axioms. Let $\overline{k}$ denote the term $1 + 1 + \cdots + 1$ (k times). The *security parameter axioms* state that that there is a string of "moderate integer" length. For each $k \geq 0$, there is an axiom Sk of the form $|\mathbf{s}| \geq \overline{k}$.

BASIC axioms. We have a collection of axiom schemes which capture the intended interpretation of the basic function symbols in the language. These are based on those given in [9], but adapted to dyadic integers, and also adding axioms for $\dot{-}$ and relating $\mathbf{n}$ to $\mathbf{s}$.

B1 $x \leq y \to x \leq y + 1$

B2 $x \neq x + 1$

B3 $0 \leq x$

B4 $(x \leq y \wedge x \neq y) \to x + 1 \leq y$

B5 $x \neq 0 \to 2x \neq 0$

B6 $x \leq y \vee y \leq x$

B7 $(x \leq y \wedge y \leq x) \to x = y$

B8 $(x \leq y \wedge y \leq z) \to x \leq z$

B9 $|0| = 0$

B10 $|2x + 1| = |x| + 1 \wedge |2x + 2| = |x| + 1$

B11 $x \leq y \to |x| \leq |y|$

B12 $(x \neq 0 \wedge y \neq 0) \to |x \otimes y| = |x| \cdot |y|$

B13 $0 \otimes x = 1$

B14 $(1 \otimes 2x + 1) = 2 \cdot (1 \otimes x) \wedge (1 \otimes 2x + 2) = 2 \cdot (1 \otimes x)$

B15 $x \otimes y = y \otimes x$

B16 $|x| = |y| \to x \otimes z = y \otimes z$

B17 $|x| = |y| + |z| \to x \otimes d = (y \otimes d) \cdot (z \otimes d)$

B18 $x \leq x + y$

B19 $(x \leq y \wedge x \neq y) \to (2x+2 \leq 2y+1 \wedge 2x+2 \neq 2y+1)$

B20 $x + y = y + x$

B21 $x + 0 = x$

B22 $x + (y + 1) = (x + y) + 1$

B23 $(x + y) + z = x + (y + z)$

B24 $x + y \leq x + z \to y \leq z$

B25 $x \cdot 0 = 0$

B26 $x \cdot (y + 1) = x \cdot y + x$

B27 $x \cdot y = y \cdot x$

B28 $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

B29 $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

B30 $1 \leq x \to ((x \cdot y \leq x \cdot z) \equiv (y \leq z))$

B31 $\left\lfloor \frac{0 \dot{-} 1}{2} \right\rfloor = 0$

B32 $x \neq 0 \to |x| = \left| \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor \right| + 1$

B33 $y \neq 0 \to \left(x = \left\lfloor \frac{y \dot{-} 1}{2} \right\rfloor \equiv (2x+1 = y \vee 2x+2 = y)\right)$

B34 $y \leq z \to y \dot{-} z = 0$

B35 $z \leq y \to (x = y \dot{-} z \equiv y = x + z)$

B36 $\mathbf{n} = |\mathbf{s}|$

Poly-time function axioms. First of all, for each basic function symbol in our language, we include an axiom stating that there is a poly-time function defined by that function symbol, e.g., $\exists f \forall x \forall y (f(x, y) = x \circ y)$. We also have, for each k and i, $1 \leq i \leq k$ an axiom PROJ(k,i) asserting the existence of the ith k-ary projections function: $\exists f^k \forall \vec{x} (f(\vec{x}) = x_i)$ and, for each k, the axiom CONST(k), asserting the existence of a function which ignores its k inputs and returns the constant 0: $\exists f^k \forall \vec{x} (f(\vec{x}) = 0)$. We also have axioms expressing closure properties of poly-time functions. For each k and l we have the axiom COMP(k,l):

$$\forall g^{k+1} \forall h^{k+l} \exists f^{k+l} \forall \vec{x} \forall \vec{y} (f(\vec{x}, \vec{y}) = g(\vec{x}, h(\vec{x}, \vec{y})))$$

which states that the poly-time functions are closed under composition. Finally, for each k we have an axiom LRN(k) expressing closure under limited (i.e. size bounded) recursion on dyadic notation:

$$\begin{aligned}
&\forall g^k \forall h^{k+2} \forall b^{k+1} \exists f^{k+1} \forall \vec{x} \forall y(\\
&\quad f(\vec{x}, 0) = g(\vec{x}) \wedge \\
&\quad |h_1(\vec{x}, y, f(\vec{x}, y))| \leq |b(\vec{x}, 2y+1)| \rightarrow (f(\vec{x}, 2y+1) = h_1(\vec{x}, y, f(\vec{x}, y))) \wedge \\
&\quad |h_1(\vec{x}, y, f(\vec{x}, y))| > |b(\vec{x}, 2y+1)| \rightarrow (f(\vec{x}, 2y+1) = b(\vec{x}, 2y+1)) \wedge \\
&\quad |h_2(\vec{x}, y, f(\vec{x}, y))| \leq |b(\vec{x}, 2y+2)| \rightarrow (f(\vec{x}, 2y+2) = h_2(\vec{x}, y, f(\vec{x}, y))) \wedge \\
&\quad |h_2(\vec{x}, y, f(\vec{x}, y))| > |b(\vec{x}, 2y+2)| \rightarrow (f(\vec{x}, 2y+2) = b(\vec{x}, 2y+2)))
\end{aligned}$$

Counting axioms. The three counting axioms are based on Kolmogorov's probability axioms (for discrete probability spaces).

C1 $\#(|x| = |y|)(\varphi \vee \neg\varphi) = y \otimes 1$

We use the following variant of Kolmogorov's additivity axiom, adapted from [17]

C2 $\#(|\vec{x}| = |\vec{y}|)\varphi = \#(|\vec{x}| = |\vec{y}|)(\varphi \wedge \psi) + \#(|\vec{x}| = |\vec{y}|)(\varphi \wedge \neg\psi)$

The next axiom reflects the somewhat complicated situation which arises from the fact that we are not axiomatizing counting over a single domain, but rather over infinitely many domains, namely $\{0,1\}^n$ for every n. This axiom also relates logical implication with bounds on counting. Intuitively, this axiom states that if there is an injective mapping from the values over a domain which satisfy φ to the values over a (possibly different) domain which satisfy ψ, then the number of values satisfying φ is a lower bound on the number satisfying ψ. Here we will restrict to injective mappings corresponding to a poly-time relation. While this is not as general as possible, it suffices for our purposes, and also leads to a somewhat simpler treatment. Poly-time relations may be modeled using binary poly-time functions. Namely, for every poly-time relation R on $(\{0,1\})^k \times (\{0,1\})^l$ there is a $(k+l)$-ary poly-time function r such that $\langle \vec{x}, \vec{y} \rangle \in R$ iff $r(\vec{x}, \vec{y}) = 0$, and vice-versa.

C3 $\exists r(\mathrm{Inj}[\vec{u}, \vec{v}, r] \wedge (\forall |\vec{x}| = |\vec{u}|)(\varphi(\vec{x}) \rightarrow (\exists |\vec{y}| = |\vec{v}|)(\psi(\vec{y}) \wedge r(\vec{x}, \vec{y}) = 0))) \rightarrow \#(|\vec{x}| = |\vec{u}|)\varphi \leq \#(|\vec{y}| = |\vec{v}|)\psi$

Here we have written $\mathrm{Inj}[\vec{u}, \vec{v}, r]$ as an abbreviation for

$$(\forall |\vec{x}| = |\vec{u}|)(\forall |\vec{z}| = |\vec{u}|)(\forall |\vec{y}| = |\vec{v}|)(r(\vec{x}, \vec{y}) = 0 \wedge r(\vec{z}, \vec{y}) = 0 \rightarrow \vec{x} = \vec{z})$$

We have also written $(\forall |\vec{x}| = |\vec{u}|)\varphi$ as an abbreviation for

$$\forall x_1 \ldots \forall x_k (|x_1| = |u_1| \wedge \ldots |x_k| = |u_k| \rightarrow \varphi)$$

(similarly for $(\exists |\vec{x}| = |\vec{u}|)\varphi$) and if $\vec{x} = x_1, \ldots, x_k$, $\vec{z} = z_1, \ldots, z_k$, then we have written $\vec{x} = \vec{z}$ as an abbreviation for $x_1 = z_1 \wedge \ldots x_k = z_k$.

Induction axiom. We limit induction to open (i.e. quantifier-free) formulas. Note that we could generalize this to bounded formulas, but open formulas seem to suffice for our purposes, and lead to a simpler soundness proof. Induction is on the *length* of the dyadic representation of the induction variable. For each polynomial p and open formula $\varphi(x)$, we have the following induction axiom, which we refer to as $\mathrm{LIND}^o(p, \varphi)$:

$$(\varphi(0) \wedge (\forall x < p(|y|)(\varphi(x) \rightarrow \varphi(x+1)))) \rightarrow \forall x \leq p(|y|)\varphi(x)$$

Derivations in T. If φ and ψ are formulas in the language of T, we write $T, \varphi \vdash \psi$ to indicate that ψ may be obtained from φ and instances of any of the logical, *BASIC*, poly-time function, counting, and induction axioms of T, using the rules of T.

Encoding into first-order logic. We may view any formula φ of T as a first-order formula φ_{fo} in a language with one (integer) sort. First of all, since it is possible to index the k-ary polynomial time functions using clocked Turing machines (see, e.g., [20]), we may eliminate function variable sorts, and view k-ary application as a $k+1$-ary function on integers. In particular, we have a function ap^k such that $\mathsf{ap}^k(\langle i,k\rangle, t_1,\dots,t_k)$ represents the application of Turing machine i with clock bound n^k to inputs $t_1,\dots,t_k$. It is then clear that translations of the polynomial time function axioms are consistent (in particular, they hold in the standard model of arithmetic, with an indexing scheme for polynomial time functions as indicated above.) Also, by extending the vocabulary of T by introducing for each basic formula φ and each $k>0$, a function ct^k_φ such that $\mathsf{ct}^k_\varphi(t_1,\dots,t_k)$ represents the number of tuples $\langle x_1,\dots,x_k\rangle$ such that $|x_i|\le|t_i|$ and $\varphi(x_1,\dots,x_k)$. Moreover, it is possible to define ct^k_φ and ap^k, using standard techniques, in Peano arithmetic. So in fact, every formulas of T may in fact be encoded as a formula in the language of Peano arithmetic.

3.3 Soundness of the general system

The results on this section rely on ideas from nonstandard models of arithmetic. We begin by describing what models for the language of T look like. Any such model will have one domain for interpreting string variables, and a domain for interpreting each of the function variable sorts. In the *standard model*, the first domain is $\mathbb{N}$, and the kth function domain consists of all polynomial time functions $\alpha:\mathbb{N}^k\to\mathbb{N}$. Function and constant symbols (other than $\mathbf{s}$ and $\mathbf{n}$ – the standard model is not able to satisfy all the security parameter axioms) are given their standard interpretation, and this interpretation extends naturally to basic terms. At this point we can define satisfaction of basic formulas in the standard way, and this allows for the interpretation of counting terms in the obvious way. Interpretation of arbitrary terms and satisfaction of general formulas may then be defined via the usual induction.

By a *bounded* formula we mean a formula in which quantifiers occur only in the form $\forall x(|x|\le|y|\to\varphi)$ or $\exists x(|x|\le|y|\wedge\varphi)$, where x is a string variable (no function quantifiers are allowed in such formulas.)

Definition 3.3.1 Suppose that $\varphi(\vec{f},\vec{z},x)$ is a formula (where $\vec{f},\vec{z}$ and x are all the free variables occurring in φ.) For any sequence $\vec{\alpha}$ of poly-time functions, we say that φ *holds asymptotically in* $\mathbb{N}$ if for any sequence $\vec{p}$ of polynomials, there is an $n_0\in\mathbb{N}$ such that for all $\vec{s},t\in\mathbb{N}$ with $|t|>|n_0|$ and $|s_i|=p_i(|x_i|)$, φ holds in $\mathbb{N}$ when the f_i's are interpreted as α_i's, the z_i's are interpreted as s_i's and x is interpreted as t.

Theorem 3.3.2 (Soundness Theorem) *Suppose that φ_1, φ_2 and ψ are bounded formulas such that*

$$T,\varphi_2(\vec{f},\vec{f'}),\forall g\forall\vec{z}\varphi_1(\vec{f},g,\vec{z},\mathbf{s})\vdash\forall g\forall\vec{z}\psi(\vec{f'},g,\vec{z},\mathbf{s}) \qquad (*)$$

where $\vec{f}$ and $\vec{f'}$ are sequences of function variables. We then have the following for all sequences $\vec{\alpha},\vec{\alpha'}$ of poly-time functions: if $\vec{\alpha},\vec{\alpha'}$ satisfy φ_2 in $\mathbb{N}$ and for every poly-time function β, φ_1 holds asymptotically in $\mathbb{N}$ then for every poly-time function β, ψ holds asymptotically in $\mathbb{N}$.

Significance of the Soundness Theorem. We want to use the soundness theorem to prove the correctness of a construction based on some cryptographic primitive. Suppose that f is a function symbol representing this primitive, and that f' is defined using f. The formula φ_2 has f and f' occurring as free variables and gives the definition of f' in terms of f, while φ_1 formalizes the security of f and ψ formalizes the security of f'. The security assumption on f states that any poly-bounded adversary has no more than a negligible chance of breaking f. Thus we have a formula of the form $\forall g\dots$, where g represents an arbitrary poly-time adversary (in the sequel, we will typically denote such an adversary function by A). The formula ψ has f' as a free variable and formalizes the security of f', again by using a formula of the form $\forall g\dots$. The asymptotic assumption about φ is then equivalent to a standard complexity-based assumption about the security of f. By (*), we can formally derive that f' is secure, under the formal assumption that f is secure. The Soundness Theorem, which gives us an asymptotic conclusion about ψ, allows us to conclude that f' is cryptographically secure. In the next section we will give an example which spells out this approach in more detail.

A proof of the Soundness Theorem is given below The ideas behind this proof go back to work by Parihk [27] on subsystems of Peano arithmetic without exponentiation. We use some simple results of model theory which follow Wilkie's paper [30].

Definition 3.3.3 Let M be a model of T. Let $I=\langle I,\mathcal{F}^I\rangle$ be a submodel of M with the following properties:

- $\mathcal{F}^I$ is a the smallest class of functions on I which contains the interpretation in M of the function symbols of T, restricted to I, and which is closed under projection, composition and LRN,

- I is closed under all functions in $\mathcal{F}^I$, and
- if $u \in I$ and $|v| \leq |u|$ then $v \in I$.

I is called an *initial segment* of M. If I is an initial segment of M and $\alpha \in \mathcal{F}^I$, there is a natural extension of α to M, which we denote α^*. These functions agree on all inputs from I.

Note that if M is a model of T, it must contain an element which is the interpretation of the constant symbol **s**. By the security parameter axioms, this means that it must contain an element of nonstandard length. Of course, by the conditions defining an initial segments, any initial segment I must also contain such a nonstandard element. The following facts regarding initial segments are used in the proof of the Soundness Theorem.

Proposition 3.3.4 *Suppose I is an initial segment of a model M of T. Let $t(\vec{f},\vec{x})$ be a term, where $\vec{f}$ and $\vec{x}$ are the only free variables in t. For any tuple $\vec{u}$ of elements of I and $\vec{\alpha}$ of elements of $\mathcal{F}^I$, $t(\vec{\alpha},\vec{u}) \in I$.*

Proof: Proceed by induction on t. For terms built up using function symbols and variables, this follows from the closure of I under $\mathcal{F}^I$. Suppose that t is $\#(|x| = |t'|)\varphi$. Since $|\#(|x| = |t'|)\varphi| \leq |(t' \otimes 1)|$ and, by assumption, $t'(\vec{\alpha},\vec{u}) \in I$, the result follows from the closure properties of I. The case for more complex counting terms, involving counting over several variables, is similar. ■

Proposition 3.3.5 *Suppose I is an initial segment of a model M. Let $\varphi(\vec{f},\vec{x})$ be a bounded formula with all free variables displayed. For any tuple $\vec{u}$ of elements of I and $\vec{\alpha}$ of elements of $\mathcal{F}^I$,*

$$I \models \varphi(\vec{\alpha},\vec{u}) \text{ iff } M \models \varphi(\vec{\alpha^*},\vec{u})$$

Proof: Proceed by induction on the structure of φ. The result is immediate for open formulas. The inductive case for Boolean combinations is also immediate. So suppose that the result holds for φ' and φ has the form $\forall|x| \leq |t|\varphi'$. If $M \models \varphi(\vec{\alpha^*},\vec{u})$ then $I \models \varphi(\vec{\alpha},\vec{u})$, as universal formulas are preserved by submodels, and $\vec{\alpha^*}$ restricted to I is just $\vec{\alpha}$. Now suppose $I \models \varphi(\vec{\alpha},\vec{u})$, Consider any $v \in M$ with $|v| \leq |t(\vec{\alpha},\vec{v})|$. By 3.3.4 and the closure properties of I, $v \in I$ and so $I \models \varphi'(\vec{\alpha},\vec{u},v)$. By the induction hypothesis $M \models \varphi'(\vec{\alpha^*},\vec{u},v)$. So $M \models \varphi(\vec{\alpha^*},\vec{u})$. The argument for bounded existential formulas follows by the equivalence of $\exists x\varphi$ and $\neg\forall x\neg\varphi$. ■

Lemma 3.3.6 *If $M \models T + \Sigma$, where all formulas in Σ are bounded, and I is an initial segment of M, then $I \models T + \Sigma$.*

Proof: Since the basic axioms and first two counting axioms are just universal closures of open formulas, satisfaction in M implies satisfaction in I due to the preservation of universal formulas by submodels. Poly-time function axioms follow from the closure properties of $\mathcal{F}^I$. Formulas in Σ follow from proposition 3.3.5. It remains to show the result for C3 and LIND$^o(p,\varphi)$. For C3, suppose that $\rho \in \mathcal{F}^I$ and for sequences $\vec{v}$,$\vec{u}$ of elements of I,

$$I \models \text{Inj}[\rho,\vec{u},\vec{v}] \wedge (\forall|\vec{x}| = |\vec{u}|)(\varphi(\vec{x}) \rightarrow (\exists|\vec{y}| = |\vec{v}|)(\psi(\vec{y} \wedge \rho(\vec{x},\vec{y}) = 0)).$$

Since by taking ρ^* to be the extension of ρ to M, we get

$$M \models \text{Inj}[\rho^*,\vec{u},\vec{v}] \wedge (\forall|\vec{x}| = |\vec{u}|)(\varphi(\vec{x}) \rightarrow (\exists|\vec{y}| = |\vec{v}|)(\psi(\vec{y} \wedge \rho^*(\vec{x},\vec{y}) = 0)),$$

and so

$$M \models \exists r(\text{Inj}[r,\vec{u},\vec{v}] \wedge (\forall|\vec{x}| = |\vec{u}|)(\varphi(\vec{x} \rightarrow (\exists|\vec{y}| = |\vec{v}|)(\psi(\vec{y} \rightarrow r(\vec{x},\vec{y}) = 0))$$

By assumption, this gives

$$M \models \#(|\vec{x}| = |\vec{u}|)\varphi \leq \#(|\vec{y}| = |\vec{v}|)\psi,$$

and then since $\vec{u}$,$\vec{v}$ are elements of I,

$$I \models \#(|\vec{x}| = |\vec{u}|)\varphi \leq \#(|\vec{y}| = |\vec{v}|)\psi.$$

To show the results for LIND$^o(p,\varphi)$, fix $u \in I$. Suppose that $I \models \varphi(0)$ and that for every $v < p(|u|)$

$$I \models (\varphi(v) \rightarrow \varphi(v+1)).$$

Since φ is bounded, it follows by proposition 3.3.5 that $M \models \varphi(0)$ and that for every $v < p(|u|)$

$$M \models (\varphi(v) \to \varphi(v+1))$$

Since $M \models \text{LIND}^o(p, \varphi)$, it follows that for every $w \leq p(|u|)$, $M \models \varphi(w)$. Now since $u \in I$ and I is an initial segment, it follows that $w \in I$ whenever $w \leq p(|u|)$. So by proposition 3.3.5, $I \models \varphi(w)$ for every $w \leq p(|u|)$. ■

Proposition 3.3.7 *Suppose that I is an initial segment of a model M of T and $\alpha \in \mathcal{F}^I$ is a k-ary function. Then there is a polynomial p such that for all $u_1, \ldots, u_k \in I$, $|\alpha(\vec{u})| \leq p(|\vec{u}|)$.*

Proof: We use induction on the structure of α. For basic function symbols, we can prove in T that they are bounded by some polynomial in the size of their arguments. For a function defined by projection the result is immediate. For functions defined by LRN we use the induction hypothesis to obtain a polynomial bound for the bounding function. For functions defined by composition, we use the induction hypothesis and simple arithmetic manipulations on compositions of polynomials which must hold in I. ■

Proof (of the Soundness Theorem): Suppose that φ_1, φ_2 and ψ are bounded formulas satisfying (*). Let $\vec{\alpha}$ and $\vec{\alpha'}$ be sequences of poly-time functions satisfying φ_2 such that in $\mathbb{N}$, for every poly-time function β and polynomials $\vec{p}$, there is an n_0 for that for all $x, \vec{z}$ with $|x| \geq n_0$, and $|z_i| = p_i(|x|)$, $\varphi_1(\vec{\alpha}, \vec{\beta}, \vec{z}, x)$. For the sake of contradiction suppose also that it is not the case that for every poly-time function β and polynomial p, there is an n_0 so that for all $x, \vec{z}$ with $|x| \geq n_0$, and $|z_i| = p_i(|x|)$, $\psi(\vec{\alpha'}, \beta, \vec{z}, x)$.

Let β_0 be a poly-time function and $\vec{q}$ a sequence of polynomials which witness the failure of the conclusion. Expand the language so that there is a function symbol for every poly-time function. To avoid introducing more notation, we will just write α as the function symbol for the poly-time function α. Let T' be obtained by adding to T, for every function symbol β and sequence $\vec{p}$ of polynomials, a formula of the form $\varphi(\vec{\alpha}, \beta, 2^{p_1(\mathbf{n})}, \ldots, 2^{p_k(\mathbf{n})}, \mathbf{s})$, along with $\neg\psi(\vec{\alpha'}, \beta_0, 2^{q_1(\mathbf{n})}, \ldots, 2^{q_k(\mathbf{n})}, \mathbf{s})$. We may now translate T' into a collection of first-order formulas as described a the end of the preceding section. In particular, we replace each polynomial time function symbol by an index for the corresponding function, Ap^k by ap^k, and counting terms by counting functions. Call this new collection of formulas T''. Every finite subset of T'' is satisfiable because of the asymptotic assumptions about φ and ψ. As noted, T'' may be viewed as a set of formulas in the (first-order) language of Peanon arithmetic. By compactness T'' has a model M (which will be a nonstandard model of arithmetic). Define a submodel I with domain consisting of all $u \in M$ such that $|u|$ is polynomial in $\mathbf{n}$. Let $\mathcal{F}^I$ be the smallest class containing the initial functions on I and closed under projection, composition and bounded primitive recursion on lengths. By proposition 3.3.7 I is an initial segment of M, and so by 3.3.6, $I \models T' \supseteq T$. Furthermore by the construction of I, $I \models \forall g \forall \vec{z} \varphi_1(\vec{f}, g, \vec{z}, \mathbf{s})$, but $I \not\models \forall g \forall x \forall z \psi(\vec{f'}, g, \vec{z}, \mathbf{s})$, giving a contradiction. ■

Extensions of the Soundness Theorem. We note that this proof can be strengthened in order to obtain the existence of a reduction which takes an adversary breaking the construction to an adversary which breaks the primitive. Suppose that the assumption of the theorem holds. We introduce a function symbol A and add an axiom which states that A breaks the construction. We also add function symbols for every function which is poly-time definable from A, and for each of these function symbols, an axiom which states that the function does not break the original primitive. By compactness we obtain a contradiction. If every finite subset of this expanded set of axioms is consistent for arbitrarily large n, we can apply compactness and obtain a contradiction to the soundness theorem. So it must be the case that there is a finite collection $M_1, \ldots, M_k$ of poly-time oracle TM's (with an oracle for A) so that for sufficiently large n, and for any adversary A which breaks the construction, there is an i such that M_i^A is a successfully adversary for the primitive. Although i may depend on n, we can combine a finite number of attacks into a single attack by estimating the success probability of each attack and using the best one. With this stronger soundness result, it follows that our system is also sound with respect to an adversarial model where adversaries are modeled by nonuniform families of circuits.

3.4 Extending the general system

We would like to be able to extend T, by introducing new function constants which represent poly-time functions. Suppose that $T \vdash \exists f \varphi$, for a k-ary function variable f. Then we can expand the vocabulary of T with a new function symbol, say f. That is, we set $L'_F = L_F \cup \{\mathsf{f}\}$, and extend the set of terms to include terms of the form $\mathsf{f}(t_1, \ldots, t_k)$ where the t_i's are basic terms. If we then set $T' = T \cup \{\varphi[\mathsf{f}/f]\}$, it is clear that T' is a conservative extension of T. Suppose that T' is such an extension of T, and that g,h_1,h_2,b are function symbols which have been introduced as part of this extension. Now suppose that we introduce a new function symbol f, and *define* f by *recursion on notation* from g,h_1 and h_2, as follows:

$$\begin{aligned}\mathsf{f}(\vec{x}, \epsilon) &= \mathsf{g}(\vec{x})\\ \mathsf{f}(\vec{x}, y\mathtt{0}) &= \mathsf{h}_1(\vec{x}, y, f(\vec{x}, y))\\ \mathsf{f}(\vec{x}, y\mathtt{1}) &= \mathsf{h}_2(\vec{x}, y, f(\vec{x}, y))\end{aligned}$$

Here we have written ϵ in place of 0 and $y\mathtt{0}$ and $y\mathtt{1}$ in place of $2y+1$ and $2y+2$, respectively, to emphasize the sense in which our scheme is indeed recursion on *notation*. Call the conjunction of these equations φ, and suppose that

$$T \vdash \varphi \to \forall\vec{x}\forall y(|\mathsf{f}(\vec{x}, y)| \leq |\mathsf{b}(\vec{x}, y)|) \tag{$*$}$$

It then follows by the LRN axiom that

$$T \vdash \exists f(f(\vec{x}, 0) = \mathsf{g}(\vec{x}) \wedge f(\vec{x}, 2y+1) = \mathsf{h}_1(\vec{x}, y, f(\vec{x}, y)) \wedge f(\vec{x}, 2y+2) = \mathsf{h}_2(\vec{x}, y, f(\vec{x}, y)))$$

It then follows that $T \vdash \varphi$, and so we may add f and its defining equations to T. Since for most definitions establishing $(*)$ will be very straightforward, we will typically just assert the bound with the understanding that it may be easily derived in T. As a result, we will freely introduce new function symbols defined by limited recursion on notation, and extend T by the defining equations for the new function symbol. Similar arguments may be given for functions defined in simpler ways (e.g., by composition). In some cases, we simplify definitions by LRN even further. Suppose that $\mathsf{h}_1 = \mathsf{h}_2 = \mathsf{h}$. In this case it makes sense to think of the recursion parameter as being presented in unary, so that recursion on notation may be expressed as primitive recursion on the length of the recursion parameter. So we sometimes write definitions by LRN in this form, i.e.,

$$\begin{aligned}\mathsf{f}(\vec{x}, 0) &= \mathsf{g}(\vec{x})\\ \mathsf{f}(\vec{x}, i+1) &= \mathsf{h}(\vec{x}, y, f(\vec{x}, y))\\ |\mathsf{f}(\vec{x}, i)| &\leq |\mathsf{b}(\vec{x}, i)|\end{aligned}$$

In the remainder of this section, we will introduce a number of new poly-time functions using the methodology described above, and also state counting principles C'4-C'11 which may be derived in T. Proof outlines for each of these counting principles are given in Appendix A. In the sequel, if C is a counting term of the form $\#(|\vec{x}| = |\vec{t}|)\varphi$, then by $\#(|\vec{u}| = |\vec{s}|)C$ we denote the term

$$\#(|\vec{u}| = |\vec{s}| \wedge |\vec{x}| = |\vec{t}|)\varphi.$$

We first introduce two counting principles which allow us to prove results involving counting terms via LIND^o. Let $C(x)$ be a counting term.

C$'$4 $\#(|x| = 0)C = C(0)$

C$'$5 $\#(|x| = |y|+1)C = \#(|x| = |y|)C(2x+1) + \#(|x| = |y|)C(2x+2)$

We also have the following principles providing some "boundary" values for counting terms:

C$'$6 $\#(|x| = |y|)C = (y \otimes 1) \cdot C$ (x not free in C)

C$'$7 $|z| = |y| \to (\#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi) = \#(|\vec{x}| = |\vec{y}|)\varphi(z))$

We also have an inclusion-exclusion principle which is easily derived from C2:

C$'$8 $\#(|x| = |y|)(\varphi \vee \psi) = \#(|x| = |y|)\varphi + \#(|x| = |y|)\psi - \#(|x| = |\vec{y}|)(\varphi \wedge \psi)$

Before introducing the next counting principle, we require some definitions.

Definition 3.4.1 The function symbols $\circ$ and drop are defined using LRN, as follows:

$$
\begin{aligned}
&x \circ 0 = x && \mathsf{drop}(x,0) = x \\
&x \circ (2y+1) = 2 \cdot (x \circ y) + 1 && \mathsf{drop}(x, 2y+1) = \left\lfloor \frac{\mathsf{drop}(x,y) \dot{-} 1}{2} \right\rfloor \\
&x \circ (2y+2) = 2 \cdot (x \circ y) + 2 && \mathsf{drop}(x, 2y+2) = \left\lfloor \frac{\mathsf{drop}(x,y) \dot{-} 1}{2} \right\rfloor \\
&|x \circ y| \leq |x| + |y| && |\mathsf{drop}(x,y)| \leq |x|
\end{aligned}
$$

Define the function lv as follows:

$$\mathsf{lv}(x,y) = x \dot{-} \mathsf{drop}(x,y) \cdot (1 \otimes y)$$

We now have the following counting principle, where $C(x_1, x_2)$ is any counting term:

C$'$9 $\#(|x_1| = |y_1| \wedge |x_2| = |y_2|)C = \#(|x| = |y_1 \circ y_2|)C(\mathsf{drop}(x, y_2), \mathsf{lv}(x, y_2))$

We also note the following fact about $\circ$:

$$T \vdash |x \circ y| = |x| + |y|$$

Using this fact, along with axioms B10 and B12, we may show that if p is any polynomial with positive coefficients, there is a term t such that for any x, $T \vdash p(|x|) = |t(x)|$. We will write $\#(|x| = p(|t|))\varphi$ as an abbreviation for $\#(|x| = |t'|)\varphi$ where t' is a term such that $|t'| = p(|t|)$. We also introduce the following abbreviation: if x represents a string of length k, then for $0 \leq i \leq k$, we define

$$x_{\{1\ldots i\}} = \begin{cases} x & \text{if } i \geq k; \\ \mathsf{drop}(x, (k-i) \otimes 1) & \text{otherwise.} \end{cases}$$

We also define

$$x_{\{i\ldots k\}} = \begin{cases} 0 & \text{if } i > k; \\ \mathsf{lv}(x, (k-(i-1)) \otimes 1) & \text{otherwise.} \end{cases}$$

By C3, it is clear that any permutation of a domain will preserve counting terms over that domain. An important special case is the permutation which on input x returns $x \oplus v$ where $\oplus$ denotes bitwise exclusive-or, and v is any fixed value such that $|v| = |x|$. We first require a definition of $\oplus$ by LRN. This is not entirely straightforward, as it requires an extension of LRN to simultaneous multi-variable definitions. In fact, such definitions are admissible, but the proof of this fact requires considerable extra machinery, which is beyond the scope of this paper. The reader may consult [14] for a very detailed development.

Definition 3.4.2 The function $\oplus$ is defined as follows:

$$
\begin{aligned}
0 \oplus 0 &= 0 \\
2x+1 \oplus 2y+1 &= 2 \cdot (x \oplus y) + 1 \\
2x+1 \oplus 2y+2 &= 2 \cdot (x \oplus y) + 2 \\
2x+2 \oplus 2y+1 &= 2 \cdot (x \oplus y) + 2 \\
2x+2 \oplus 2y+2 &= 2 \cdot (x \oplus y) + 1 \\
|x \oplus y| &\leq |x|
\end{aligned}
$$

We then have the following counting principle for any counting term $C(x)$:

C$'$10 $\forall v(|v| = |y| \to \#(|x| = |y|)C = \#(|x| = |y|)C(x \oplus v))$

We introduce terms of the form $\#(i \leq j \wedge |\vec{x}| = |\vec{t}|)\varphi$ which is used to count the number of tuples $\langle i, \vec{x} \rangle$ for which φ holds where $i \in \{0, \ldots, j\}$ and $|\vec{x}| = |\vec{t}|$. To do this, we first define the auxiliary function bin.

Definition 3.4.3 The function bin is defined by

$$\mathsf{bin}(x) = x - (x \otimes 1) + 1.$$

Intuitively, $\mathsf{bin}(x)$ gives the integer whose (possibly left-zero-padded) *binary* notation equals the dyadic notation of x. For example, the dyadic notation of 9 is 010, which is the (left-zero-padded) binary notation of 2, and $\mathsf{bin}(9) = 9 - (9 \oplus 1) + 1 = 9 - 2^3 + 1 = 2$. Note that in general, the length of the binary notation of a number might be one greater than the length of its dyadic notation. For example, the dyadic notation of 8 is 001, while its binary notation is 1000. With these considerations in mind, we may write

$$\#(i \leq j \wedge |\vec{x}| = |\vec{t}|)\varphi$$

as an abbreviation for

$$\#(|x| = |j| + 1 \wedge |\vec{x}| = |\vec{t}|)(\mathsf{bin}(x) \leq j \wedge \varphi[\mathsf{bin}(x)/i]).$$

We have the following recursion-like principle for terms of this type. Suppose that $C(i)$ is a counting term.

C′11 $\#(i \leq k+1)C = C(k+1) + \#(i \leq k)C$

The last counting principle we describe captures the notion of telescoping sums used in hybrid arguments

C′12 $\forall m \leq |z|(\forall i < m(C(i) = C'(i+1)) \rightarrow \#(i \leq m)C - \#(i \leq m)C' = C(m) - C'(0))$

3.5 Sample proofs

We will now use the soundness theorem to prove the correctness of a construction for "stretching" the output of a pseudo-random generator (PRG). The reader should note that the primary purpose of this example is to demonstrate the nuts-and-bolts of proofs in the system T. It is not our contention that such proofs are particularly perspicuous, or amenable to automation. In fact, our ultimate goal is to develop systems which allow more direct reasoning about cryptographic constructions (such a system is presented in Section 4). In this setting, the system T can be viewed as an intermediate step. The techniques presented here can be extended to give general soundness results for the system of Section 4. Below we also prove the correctness of a coin-flipping protocol, demonstrating that T may also be used to reason about simple protocols.

To begin, we must show how the definition of PRG can be formalized in our framework. Recall that for any positive polynomial p, a poly-time function $g : \{0,1\}^* \rightarrow \{0,1\}^*$ is a *pseudo-random generator with stretch* $p(n)$ if for all x, $|g(x)| = |x| + p(|x|)$ and if for every poly-time function $A : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$, and all polynomials r and q, there is an n_0 such that for all $n \geq n_0$,

$$\mathrm{Pr}_{R,X}[A(R, g(X)) = 1] - \mathrm{Pr}_{R,Y}[A(R, Y) = 1] \leq 1/q(n)$$

where $R \in_{\mathcal{U}} \{0,1\}^{r(n)}$, $X \in_{\mathcal{U}} \{0,1\}^n$, and $Y \in_{\mathcal{U}} \{0,1\}^{n+p(n)}$.

Statements having this form are formalized in the language of T as follows:

$$\forall A \forall z_1 \forall z_2 \left(\frac{\#\left(\substack{|r|=|z_2| \\ |x|=\mathbf{n}}\right)(A(r, g(x)) = 1)}{2^{|z_2|+\mathbf{n}}} - \frac{\#\left(\substack{|r|=|z_2| \\ |y|=\mathbf{n}+p(\mathbf{n})}\right)(A(r, y) = 1)}{2^{|z_2|+\mathbf{n}+p(\mathbf{n})}} \leq \frac{1}{|z_1|} \right)$$

Where, for a polynomial q, we write $2^{q(\mathbf{n})}$ for $t(\mathbf{s}) \otimes 1$, where t is a term such that $|t(\mathbf{s})| = q(\mathbf{n})$, and $\frac{s_1}{s_2} - \frac{t_1}{t_2} \leq \frac{u_1}{u_2}$ is an abbreviation for $s_1 \cdot t_2 \cdot u_2 - t_1 \cdot s_2 \cdot u_2 \leq u_1 \cdot s_2 \cdot t_2$. In the sequel, we will sometimes write formulas using standard probability notation for the sake of readability. Such formulas may be formalized in the language of T, as described here. Up to this point, as we have been focusing on the soundness of T, we do not want to introduce a more complex syntax involving symbols such as Pr, as this would only complicate the soundness proof. When applying the system, of course, we want a richer syntax, which can easily be built up on top of T. We will use this richer syntax in our examples of proofs in T.

A few more words on methodology. In this setting, we start by *assuming* the existence of some *primitive* f, which is a poly-time function satisfying some security property. The security property for f is given in the assumption $\forall A \forall r \forall z \varphi_1(f, A, r, z)$ for the Soundness Theorem. Here, f is an unbound function variable in φ, signifying that it is an (unspecified) poly-time function satisfying the security property. Based on this primitive, we construct define a new object f' which, under the assumptions about f, is a poly-time function satisfying some security property. Note that the equations used to define f' from f are also given as part of the assumption in the Soundness Theorem in the formula

φ_2. It is important to remember that there are two orthogonal concerns related to the correctness of the construction. The first is that the constructed object is indeed a polynomial time function. Here we may follow the methodology for introducing new polynomial time functions described in Section 3.4. The second concern is verifying that the constructed ojbect satisfies its security requirements, assuming that the primitive does. This verification depends on an application of the Soundness Theorem.

Definition 3.5.1 Suppose that f is a PRG with stretch 1. We write $f(x)$ as $b(x) \circ r(x)$ where $|b(x)| = 1$ and $|r(x)| = |x|$. Define f', by $f'(x,i) = b'(x,i) \circ r'(x,i)$ where

$$\begin{array}{ll} r'(x,0) = x & b'(x,0) = 0 \\ r'(x,i+1) = r(r'(x,i)) & b'(x,i+1) = \\ & \quad b'(x,i) \circ b(r'(x,i)) \\ |r'(x,i)| \leq |x| & |b'(x,i)| \leq i \end{array}$$

Note that this is the construction of Goldreich and Micali presented in [7], described using recursion on notation. The bounds on the growth of r' and b' are not required for the correctness of the construction, but to ensure that they are feasible functions. This allows us to use the LRN axiom to introduce function symbols r', b' and f' which satisfy the given defining equations.

We now show how, based on our soundness theorem, we can use b' to obtain a PRG with stretch n from a PRG f with stretch 1. (We could in fact obtain a PRG with stretch $p(n)$ for any polynomial p. We are limiting ourselves to stretch n just to simplify the presentation.)

Theorem 3.5.2 *Let $\varphi_1(f, A, r, z)$ and $\psi(b', A, r, z)$ be the formulas of T corresponding, respectively to the following statements:*

$$\mathrm{Pr}_{R,X}[A(R, f(X)) = 1] - \mathrm{Pr}_{R,Y}[A(R,Y) = 1] \leq 1/|z|$$

where $R \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{|r|}$, $X \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{\mathbf{n}}$ and $Y \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{\mathbf{n}+1}$, and

$$\mathrm{Pr}_{R,X}[A(R, b'(X, 2\mathbf{n})) = 1] - \mathrm{Pr}_{R,Y}[A(R,Y) = 1] \leq 1/|z|$$

where $R \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{|r|}$, $X \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{\mathbf{n}}$ and $Y \in_{\mathcal{U}} \{\mathtt{0},\mathtt{1}\}^{2\mathbf{n}}$, and let φ_2 be the formula

$$(\forall |x| \leq \mathbf{n})(\forall i \leq \mathbf{n})\varphi_2'$$

where φ_2' is the conjunction of the equations from Definition 3.5.1. Then $T, \varphi_2, \forall A \forall r \forall z \varphi_1 \vdash \forall A \forall r \forall z \psi$.

From this result and the Soundness theorem for T, it follows immediately that

Corollary 3.5.3 *If f is a PRG with stretch 1, then $\lambda x.b'(x, 2|x|)$ is a PRG with stretch $2|x|$.*

Proof of Theorem 3.5.2: Our proof follows the form of a standard hybrid argument. We will use the more concise probability notation described above. As a further abbreviation, if A is a 0-1 valued function, we write $A(\vec{x})$ for the formula $A(\vec{x}) = 1$. The proof will be by contradiction, so we will actually show that

$$T, \varphi_2, \neg\forall A \forall r \forall z \psi \vdash \neg\forall A \forall r \forall z \varphi_1.$$

To begin we need to put the definition of b' from φ_2 into a form useful for defining an adversary which breaks f. We may show by LIND^o that

$$T, \varphi_2 \vdash \forall x \forall i \leq 2\mathbf{n}(b'(x, i+1) = b(x) \circ b'(r(x), i)) \tag{†}$$

In particular, when $i = 0$ we have

$$b'(x,i+1) = b'(x,1) = b'(x,0) \circ b(r'(x,0)) = 0 \circ b(x) = b(x) \circ 0 = b(x) \circ b'(r(x),0) = b(x) \circ b'(r(x),i)$$

Now assume that the equation holds for all x and for $i = k < 2\mathbf{n}$. When $i = k+1$ we have

$$\begin{aligned} b'(x, i+1) &= b'(x, k+2) \\ &= b'(x, k+1) \circ b(r'(x, k+1)) \\ &= b(x) \circ b'(r(x), k) \circ b(r'(r(x), k) \\ &= b(x) \circ b'(r(x), k+1) \\ &= b(x) \circ b'(r(x), i). \end{aligned}$$

Define the poly-time hybrid function h as follows:

$$h(y, i, u) = \begin{cases} y_{\{1 \ldots 2\mathbf{n}-(i+1)\}} \circ u_1 \circ b'(u_{\{2 \ldots \mathbf{n}+1\}}, i) & \text{if } i \leq 2\mathbf{n}; \\ 0 & \text{otherwise.} \end{cases}$$

From the assumption $\neg \forall A \forall r \forall z \psi$, we have A_0,r_0 and z_0 such that $\neg\varphi_1(b', A_0, r_0, z_0)$. We may now define A_1, the poly-time adversary which will witness that f is not a pseudo-random generator. A_1 is defined as follows:

$$A_1(s, u) = \begin{cases} A_0(s_1, h(s_2, \mathsf{bin}(s_3), u)) & \mathsf{bin}(s_3) < 2\mathbf{n}; \\ 0 & \text{otherwise,} \end{cases}$$

where $s_3 = \mathsf{lv}(s, 2(2\mathbf{n}) + 1)$, $s_2 = \mathsf{lv}(\mathsf{drop}(s, s_3), (2\mathbf{n}) \otimes 1)$ and $s_1 = \mathsf{drop}(s, s_2 \circ s_3)$. In particular, this means that if $s = r \circ y \circ t$ with $|y| = 2\mathbf{n}$ and $|t| = |2(2\mathbf{n}) + 1|$, then $s_1 = r$, $s_2 = y$ and $s_3 = t$ and so by C'9 we have for any u and r_1 such that $|r_1| = |r_0| + 2\mathbf{n} + |2(2\mathbf{n}) + 1|$,

$$\Pr_{|s|=|r_1|}[A_1(s, u)] = \Pr_{\substack{i<2\mathbf{n} \\ |y|=2\mathbf{n}, |r|=|r_0|}}[A_0(r, h(y, i, u))]. \tag{*}$$

Now by (†), C'6 and C'9 we have, for $i < 2\mathbf{n} - 1$,

$$\Pr_{\substack{|r|=|r_0| \\ |x|=\mathbf{n}, |y|=2\mathbf{n}}}[A_0(r, h(y, i, f(x)))] = \Pr_{\substack{|r|=|r_0| \\ |x|=\mathbf{n}+1, |y|=2\mathbf{n}}}[A_0(r, h(y, i+1, x))]. \tag{**}$$

We then have

$$\begin{aligned} &\Pr_{|s|=|r_1|, |x|=\mathbf{n}}[A_1(s, f(x))] - \Pr_{|s|=|r_1|, |x|=\mathbf{n}+1}[A_1(s, x)] \\ &\quad = \Pr_{\substack{|r|=|r_0|, i<2\mathbf{n} \\ |x|=\mathbf{n}, |y|=2\mathbf{n}}}[A_0(r, h(y, i, f(x)))] - \Pr_{\substack{|r|=|r_0|, i<2\mathbf{n} \\ |x|=\mathbf{n}+1, |y|=2\mathbf{n}}} A_0[(r, h(y, i, x))] && \text{(by *)} \\ &\quad = \frac{1}{2\mathbf{n}} \cdot \left(\Pr_{\substack{|r|=|r_0| \\ |x|=\mathbf{n}, |y|=2\mathbf{n}}}[A_0(r, h(y, 2\mathbf{n}-1, f(x))] - \Pr_{\substack{|r|=|r_0| \\ |x|=\mathbf{n}+1, |y|=2\mathbf{n}}}[A_0(r, h(y, 0, x))] \right) \\ &&& \text{(by C'12 and **)} \\ &\quad = \frac{1}{2\mathbf{n}} \cdot \left(\Pr_{|r|=|r_0|, |x|=\mathbf{n}}[A_0(r, b'(x, 2\mathbf{n}))] - \Pr_{|r|=|r_0|, |x|=2\mathbf{n}}[A_0(r, x)] \right) && \text{(by †, C'6 and C'9)} \\ &\quad > \frac{1}{2\mathbf{n}|z_0|} && \text{(by the assumption } \neg\psi(b', A_0, r_0, z_0)) \end{aligned}$$

Choosing z_1 such that $|z_1| = 2\mathbf{n}|z_0|$ (say $z_1 = (\mathbf{s} \circ \mathbf{s}) \otimes z_0$,) we have $\neg\varphi_1(f, A_1, r_1, z_1)$, from which we obtain $\neg\forall A \forall r \forall z \varphi_1(f, A, r, z)$. ■

The framework that we have introduced so far is not general enough to model arbitrary notions of security for protocols. For example, to deal with adversaries with some form of chosen-message attack would require the introduction of functionals. However, we do have sufficient machinery to model protocols in which only one execution of the protocol is analyzed, or in which the adversary is non-adaptive. As an example, we now show how to model and verify a protocol for flipping coins on a public line [5], which uses hidden bit commitment as a subroutine. In order to achieve hidden bit commitment, we use a construction of Naor [26], which uses a pseudo-random generator as a primitive.

The following is a high-level description of the protocol: Alice sends Bob a random $v \in_U \{0,1\}^{3n}$. Bob computes $g(s)$ where s is a random n bit string and g is a pseudo-random generator from n bits to $3n$ bits, and picks a random $b \in_U \{0,1\}$. To commit to $b = 0$ he sends $g(s)$, to a 1, he sends $g(s) \oplus v$. Then (for coin-flipping) Alice sends Bob a random $b' \in_U \{0,1\}$. Bob reveals s and b and outputs $b \oplus b'$. Alice outputs $b \oplus b'$ if the above checks out; otherwise she rejects.

We use lower case for the correct protocol and upper-case for a hypothetical possibly cheating message. We can model the correct protocol as 3 functions for each player: $a_1(v, b') = v$ is Alice's first message, $a_2(v, b', B_1) = b'$ is Alice's second message and $a_{out}(v, b', B_1, (B, S)) = B \oplus b'$ if $g(S) = B_1 \oplus v$ or $g(S) = B_1$ (depending on whether B is 1 or 0), Reject otherwise. Similarly, $b_1(b, s, A_1) = g(s)$ or $g(s) \oplus A_1$ depending on b, $b_2(b, s, A_1, A_2) = (b, s)$, and $b_{out}(b, s, A_1, A_2) = b \oplus A_2$.

An adversary for Bob (i.e., a cheating Alice) is a bit c and a pair of functions $A_2(r, b_1(A_1(r), b, s))$ and $A_1(r)$. The adversary succeeds if probability $b_{out}(b, s, A_1, A_2) = c > \frac{1}{2}+$ a non-negligible function. Similarly, a successful cheating Bob, an adversary for Alice, is a bit c and functions $B_1(r, A_1)$, $B_2(r, A_1, A_2)$ so that $a_{out}(v, b', B_1, B_2) = c$ with probability $> \frac{1}{2}+$ non-negligible amount.

We now outline how to prove in T that the protocol outlined above is correct, i.e. that there are no successful cheating adversaries. We first consider a cheating Alice. Starting with the formal definition of pseudo-random generator given above, along with the easy to establish fact (using counting axiom C$'$10) that for any function A, any r_0 and any v with $|v| = 3 \cdot \mathbf{n}$

$$\# \left({}^{|y|=3\cdot\mathbf{n}}_{|r|=|r_0|} \right) A(r, y \oplus v) = \# \left({}^{|y|=3\cdot\mathbf{n}}_{|r|=|r_0|} \right) A_2(r, y)$$

we can derive that for any function A, any r_0 and any v with $|v| = 3 \cdot \mathbf{n}$

$$\Pr_{\substack{|r|=|r_0|\\|s|=\mathbf{n}}}[A(r, g(s) \oplus v) = 0] - \Pr_{\substack{|r|=|r_0|\\|s|=\mathbf{n}}}[A(g(s)) = 0] \leq \frac{1}{|z|}$$

It then follows that for any adversaries A_1, A_2, and any r_0 and z

$$\Pr_{\substack{|r|=|r_0|\\|s|=\mathbf{n}}}[A_2(r, b_1(A_1(r), 1, s) = 0] - \Pr_{\substack{|r|=|r_0|\\|s|=\mathbf{n}}}[A_2(r, b_1(A_1(r), 0, s) = 0] \leq \frac{1}{|z|}$$

But then for any adversaries A_1, A_2, and any r_0, z and $c \in \{0,1\}$

$$\Pr_{\substack{|s|=\mathbf{n},|r|=|r_0|\\|b|=1}}[b \oplus A_2(r, b_1(A_1(r), b, s) = c] \leq \frac{1}{2} + \frac{1}{|z|}$$

as required.

For a cheating Bob, we want to show that for $c \in \{0,1\}$ and functions $B_1(v, r)$ and $B_2(v, r, a_2(v, b', B_1(v, r)))$ a cheating Bob uses,

$$\Pr_{\substack{|v|=3\mathbf{n},|r|=|r_0|\\|b|=1}}[a_{out}(b, v, B_1(a_1(v, b), r), B_2(v, r, a_2(v, b, B_1(a_1(v, b), r))))] = c] \leq 1/2 + 1/|z|.$$

This is purely a counting argument; no complexity theory assumptions are necessary. Using the definitions of $a_1(v, b) = v, a_2(v, b, B_1) = b$ we can re-write the probability above as

$$\Pr_{\substack{|v|=3\mathbf{n},|r|=|r_0|\\|b|=1}}[a_{out}(b, v, B_1(v, r), B_2(v, r, b)) = c)].$$

Then substituting the definition of a_{out}, and letting $B_2(v, r, b) = B(v, r, b), S(v, r, b)$, we obtain the sequence of equalities and inequalities given in Figure 1. The resulting proof is not completely formalized, but each line follows from the cited counting axioms, or else from BASIC axioms and the definitions of Pr and Exp.

$$
\begin{aligned}
&\Pr_{\substack{|v|=3\mathbf{n},|r|=|r_0|\\|b|=1}}[g(S(v,r,b)) = B_1(v,r) \oplus v \cdot B(v,r,b) \wedge b \oplus B(v,r,b) = c] \\
&= \frac{1}{2}\left(\operatorname*{Exp}_{|v|=3\mathbf{n},|r|=|r_0|}[\#(|b|=1)(g(S(v,r,b)) = B_1(v,r) \oplus v \cdot B(v,r,b) \wedge b \oplus B(v,r,b) = c)] \right) \\
&= \frac{1}{2}\left(1 + \Pr_{|v|=3\mathbf{n},|r|=|r_0|}\left[\begin{smallmatrix} g(S(v,r,0))=B_1(v,r)\oplus v\cdot B(v,r,0)\wedge B(v,r,0)=c \\ \wedge g(S(v,r,1))=B_1(v,r)\oplus v\cdot B(v,r,1)\wedge B(v,r,1)=\neg c \end{smallmatrix} \right] \right) && \text{(C}'\text{8,C1)} \\
&\leq \frac{1}{2}\left(1 + \Pr_{|v|=3\mathbf{n},|r|=|r_0|}[g(S(v,r,0) = B_1(v,r) \oplus v \cdot c \wedge g(S(v,r,1) = B_1(v,r) \oplus v \cdot (1-c)] \right) && \text{(C3)} \\
&\leq \frac{1}{2}\left(1 + \Pr_{|v|=3\mathbf{n},|r|=|r_0|}[g(S(v,r,0)) \oplus g(S(v,r,1)) = v] \right) && \text{(C3)} \\
&\leq \frac{1}{2}\left(1 + \operatorname*{Exp}_{|v|=3\mathbf{n},|r|=|r_0|}\left[\# \left(\begin{smallmatrix} |s_1|=\mathbf{n} \\ |s_2|=\mathbf{n} \end{smallmatrix} \right) (g(S(v,r,0)) \oplus g(S(v,r,1)) = v) \right] \right) && \text{(C}'\text{6)} \\
&\leq \frac{1}{2}\left(1 + \operatorname*{Exp}_{|v|=3\mathbf{n},|r|=|r_0|}\left[\# \left(\begin{smallmatrix} |s_1|=\mathbf{n} \\ |s_2|=\mathbf{n} \end{smallmatrix} \right) (s_1 = S(v,r,0) \wedge s_2 = S(v,r,1) \wedge g(s_1) \oplus g(s_2) = v) \right] \right) && \text{(C3)} \\
&\leq \frac{1}{2}\left(1 + \operatorname*{Exp}_{|v|=3\mathbf{n},|r|=|r_0|}\left[\# \left(\begin{smallmatrix} |s_1|=\mathbf{n} \\ |s_2|=\mathbf{n} \end{smallmatrix} \right) (g(s_1) \oplus g(s_2) = v) \right] \right) && \text{(C3)} \\
&= \frac{1}{2}\left(1 + \frac{1}{2^{3\mathbf{n}}} \cdot \# \left(\begin{smallmatrix} |v|=3\mathbf{n} \\ |s_1|=\mathbf{n} \\ |s_2|=\mathbf{n} \end{smallmatrix} \right) (g(s_1) \oplus g(s_2) = v) \right) && \text{(C}'\text{6)} \\
&= \frac{1}{2}\left(1 + \frac{1}{2^{3\mathbf{n}}} \cdot \# \left(\begin{smallmatrix} |s_1|=\mathbf{n} \\ |s_2|=\mathbf{n} \end{smallmatrix} \right) 0 = 0 \right) && \text{(C}'\text{7)} \\
&= \frac{1}{2}\left(1 + \frac{2^{2\mathbf{n}}}{2^{3\mathbf{n}}} \right) && \text{(C1)} \\
&= \frac{1}{2}\left(1 + \frac{1}{2^{\mathbf{n}}} \right) \leq \frac{1}{2}(1 + 1/|z|)
\end{aligned}
$$

Figure 1: Proof of correctness of the coin-flipping protocol

4 A system for computational indistinguishability

A drawback of the system presented in the preceding section is that it still requires us to reason explicitly about probabilities. In this section we show that by restricting our attention to a particular security property, we are able to eliminate the need for such explicit reasoning. We will present a system for reasoning about *computational indistinguishability*, in which proofs are essentially equational. We will prove soundness for this system by interpreting it in the general system from the previous section and then appealing to the soundness proof that we have already given for this system. In fact, we see this kind of soundness proof as the most important application of the general system: it provides a formal framework for doing "meta-proofs" for systems providing more direct forms of reasoning. While the system we present here could be viewed as an extension of T, it is in fact more self-contained, and does not require most of the machinery of T. As we will see, one rule in this system does require external reasoning, that is, to apply the rule we first need to derive statements of the form $\forall\vec{x}(f(\vec{x}) = g(\vec{x})$. However, we do not need the full power of T to do this.

We work in a language with terms which denote probabilistic poly-time (PPT) functions. More precisely, terms denote PPT function *ensembles*. To this end we have a collection of basic functions and typical closure schemes for defining poly-time functions. We also include terms of the form $\mathsf{let}\ i \leftarrow \mathsf{rand}(p(\mathbf{n}))\ \mathsf{in}\ t$ and $\mathsf{let}\ x \leftarrow \mathsf{rs}(p(\mathbf{n}))\ \mathsf{in}\ t$, for any polynomial p. Intuitively, the meaning of the first kind of term is just the meaning of t where variable i is selected uniformly at random from the set $\{0, \ldots, p(\mathbf{n}) - 1\}$, while the meaning of the second kind of term is t where x is selected uniformly at random from strings of length $p(\mathbf{n})$. So rand is shorthand for "random index" while rs is shorthand for "random string". Note that we only allow the formation of terms of the first (second) form when i is free in t (resp. x is free in t.) In cases where there is no ambiguity, we sometimes write, e.g., $t[\mathsf{rs}(p(\mathbf{n}))/x]$ instead of $\mathsf{let}\ x \leftarrow \mathsf{rs}(p(\mathbf{n}))\ \mathsf{in}\ t$, say when t has only one occurrence of x. Here $t[s/x]$ means term t with all *free* occurrences of x replaced by s. Free variables are those which are not in the scope of a `let`. We always assume that substitution is done in such a way as to avoid variable capture, by changing the name of bound variables if necessary. In general, we refer to $i \leftarrow \mathsf{rand}(\mathbf{n})$ and $x \leftarrow \mathsf{rs}(\mathbf{n})$ as *random bindings*. Suppose $b_1, \ldots, b_k$ is a sequence of bindings. We sometimes abbreviate $\mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ t$ as $\mathsf{let}\ \begin{pmatrix} b_1 \\ \cdots \\ b_k \end{pmatrix}\ \mathsf{in}\ t$ Note that with this syntax, the ordering of bindings from top to bottom is significant because for $1 \leq i < j \leq k$, b_j may refer to the variable bound in b_i. We say that a term is *normal* if it has the form $\mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ t$ where t is just a basic term of T. If

$$v = \mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ t$$

is a normal term and x is a free variable (i.e. not bound by any of the b_i's) occurring in t and

$$u = \mathsf{let}\ c_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ c_k\ \mathsf{in}\ s$$

is any normal term, then by $v\{u/x\}$ we denote the normal term

$$\mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ \mathsf{let}\ c_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ c_k\ \mathsf{in}\ t[s/x],$$

where $t[s/x]$ just denotes the usual substitution for s in x. A normal term is *closed* just in case all the variables occurring in t are bound in one of the $b_1, \ldots, b_k$. Suppose that s and t are closed normal terms. We introduce formulas of the form $s \approx t$, whose intended meaning is that the distribution ensembles represented by s and t are computationally indistinguishable. We would like to reason directly about $\approx$. To this end, we will introduce a number of rule schemes which capture intuitive properties of $\approx$. The soundness of these rules may be shown as follows: we first given a translation of formulas of the form $s \approx t$ into the language of T. We then show that the translation of each of the rules may be proved in T. Soundness then follows from the soundness of T with respect to cryptographic interpretations. Each of the rules given below have fairly simple proofs in T, as we show in Section 4.1.

To begin with, we have rules stating the $\approx$ is reflexive, symmetric and transitive. The remaining rules are given in Figure 2. The (UNIV) rule relates universally quantified equality statements provable in T to $\approx$. In this rule, s and

$$\frac{T \vdash Q_1 Q_2 \ldots Q_k(s = t)}{\mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ s \approx \mathsf{let}\ b_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ b_k\ \mathsf{in}\ t}\ (\mathsf{UNIV})$$

$$\frac{u \approx u'}{v\{u/x\} \approx v\{u'/x\}}\ (\mathsf{SUB}) \qquad \frac{\mathsf{let}\ i \leftarrow \mathsf{rand}(p(\mathbf{n}))\ \mathsf{in}\ u \approx \mathsf{let}\ i \leftarrow \mathsf{rand}(p(\mathbf{n}))\ \mathsf{in}\ u[i+1/i]}{u[0/i] \approx u[p(\mathbf{n})/i]}\ (\mathsf{H\text{-}IND})$$

$$\frac{}{\mathsf{let}\ \begin{pmatrix} \vec{i} \leftarrow \mathsf{rand}(\vec{p}(\mathbf{n})) \\ x \leftarrow \mathsf{rs}\big(\sum_{j=1}^{k}(p_j(\mathbf{n}) - i_j)\big) \end{pmatrix}\ \mathsf{in}\ x \approx \mathsf{let}\ \begin{pmatrix} \vec{i} \leftarrow \mathsf{rand}(\vec{p}(\mathbf{n})) \\ \vec{x} \leftarrow \mathsf{rs}(\vec{p}(\mathbf{n})) \end{pmatrix}\ \mathsf{in}\ \bigcirc_{j=1}^{k} x_{j\{1 \ldots p_j(\mathbf{n}) - i_j\}}}\ (\mathsf{EDIT})$$

Figure 2: Rules for computational indistinguishability

t are terms of T, and $Q_1, \ldots, Q_k$ is a sequence of quantifiers of the form $\forall |x| \leq p(\mathbf{n})$ or $\forall i < p(\mathbf{n})$. Each b_i is a random binding of the form $x \leftarrow \mathsf{rs}(p(\mathbf{n}))$ (resp. $i \leftarrow \mathsf{rand}(p(\mathbf{n}))$), which corresponds to Q_i. All the free variables of s and t must appear in some Q_i. Note that the choice of T in the antecedent of this rule is somewhat arbitrary and was made here for the sake of concreteness. We could actually choose a system without much of the extra machinery introduced in T, say a suitably formulated version of Cook's PV [13].

The (SUB) rule states that we may substitute indistinguishable terms into any probabilistic polynomial time context. Here v is any term in the extended language whose only free variable is x.

The (EDIT) rule enables us to merge, split and/or shorten random strings, the result being indistinguishable from a randomly chosen string of the appropriate length. In the statement of this rule, $\bigcirc_{j=1}^{k} t_i$ denotes $t_1 \circ \cdots \circ t_k$.

The $(\mathsf{H\text{-}IND})$ rule has the flavour of an induction rule, but it also captures the basic fact about computational indistinguishability required for hybrid arguments.

We now repeat the proof from the previous section that the function f' given in Definition 3.5.1 can be used to define a pseudo-random generator with stretch $\mathbf{n}$. Suppose $f(x) = b(x) \circ r(x)$ is a pseudo-random generator with stretch 1. We can capture this with

$$\mathsf{let}\ x \leftarrow \mathsf{rs}(\mathbf{n})\ \mathsf{in}\ f(x) \approx \mathsf{let}\ x \leftarrow \mathsf{rs}(\mathbf{n}+1)\ \mathsf{in}\ x.$$

Using this equivalence and SUB with the context $\mathsf{let}\ i \leftarrow \mathsf{rand}(\mathbf{n})\ \mathsf{in}\ z_{\{1\}} \circ b'(z_{\{2\ldots\mathbf{n}\}}, i)$, we obtain

$$\mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ b(x) \circ b'(r(x), i) \approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}+1) \end{pmatrix}\ \mathsf{in}\ x_{\{1\}} \circ b'(x_{\{2\ldots\mathbf{n}\}}, i) \qquad (\dagger)$$

Suppose that r' and b' are defined as in Definition 3.5.1. We want to show that $\lambda x.b'(x, 2\mathbf{n})$ is a pseudo-random generator with stretch $\mathbf{n}$. Recall that from the proof of Theorem 3.5.2, for any x and $i \leq \mathbf{n}$, we have $b'(x, i+1) = b(x) \circ b'(r(x), i)$. While this proof was given in T, and examination of the proof shows that it uses purely equational reasoning along with induction on notation. Such a proof could be formalized in a system much simpler than T, for example Cook's PV. However, having established this equality, by UNIV we then have

$$\mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ b'(x, i+1) \approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ b(x) \circ b'(r(x), i) \qquad (\dagger\dagger)$$

Define the hybrid function h by $h(z, x, i) = z_{\{1\ldots 2\mathbf{n}-i\}} \circ b'(x, i)$. We then obtain

$$\begin{aligned}
\mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(2\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ h(z, x, i+1) &\approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(2\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ z_{\{1\ldots 2\mathbf{n}-(i+1)\}} \circ b'(x, i+1) && \text{(by } \mathsf{UNIV}\text{)} \\
&\approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(2\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}+1) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ z_{\{1\ldots 2\mathbf{n}-(i+1)\}} \circ x_{\{1\}} \circ b'(x_{\{2\ldots\mathbf{n}\}}, i) \\
& && \text{(by } \mathsf{SUB}, \dagger \text{ and } \dagger\dagger\text{)} \\
&\approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(2\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ z_{\{1\ldots 2\mathbf{n}-i\}} \circ b'(x, i) && \text{(by } \mathsf{EDIT} \text{ and } \mathsf{SUB}\text{)} \\
&\approx \mathsf{let}\ \begin{pmatrix} i\leftarrow\mathsf{rand}(2\mathbf{n}) \\ x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ h(z, x, i) && \text{(by } \mathsf{UNIV}\text{)}
\end{aligned}$$

So it follows by $\mathsf{H\text{-}IND}$ that $\mathsf{let}\ \begin{pmatrix} x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ h(z, x, 0) \approx \mathsf{let}\ \begin{pmatrix} x\leftarrow\mathsf{rs}(\mathbf{n}) \\ z\leftarrow\mathsf{rs}(2\mathbf{n}) \end{pmatrix}\ \mathsf{in}\ h(z, x, 2\mathbf{n})$. But for any z and x with $|z| \leq \mathbf{n}$ we have $h(z, x, 0) = z \circ x$ and $h(z, x, 2\mathbf{n}) = b'(x, 2\mathbf{n})$. By UNIV, EDIT and SUB we then obtain $\mathsf{let}\ x \leftarrow \mathsf{rs}(\mathbf{n})\ \mathsf{in}\ b'(x, 2\mathbf{n}) \approx \mathsf{rs}(2\mathbf{n})$, as required.

Below we give another example in this system, proving that next-bit unpredictability implies pseudorandomness.

4.1 Soundness via interpretation in the general system

We interpret

$$\mathsf{let}\ b_1\ \mathsf{in}\ \ldots \mathsf{let}\ b_k\ \mathsf{in}\ t \approx \mathsf{let}\ c_1\ \mathsf{in}\ \ldots \mathsf{let}\ c_l\ \mathsf{in}\ s$$

as the following formula of T:

$$\forall A \forall z (\Pr_{b_1,\ldots,b_k}[A(t) = 1] - \Pr_{c_1,\ldots,c_l}[A(s) = 1]) \leq \frac{1}{|z|}$$

Soundness of the system for computational indistiguishability is established by proving the interpretation of each of its rules in T. These proofs use the properties of counting terms in an essential way. We explain how soundness for each of the rules may be proved. In some cases we consider a concrete instantiation of the rule rather than its general form in order to make the underlying idea more apparent.

Reflexivity, symmetry and transitivity are obvious.

For UNIV, we have the assumption $Q_1 \dots Q_k(s = t)$. Suppose for concreteness that we have $\forall |x| = \mathbf{n} \forall i < \mathbf{n}(s = t)$. Note that for any φ, it follows by C3 that

$$\#(|x| = \mathbf{n} \wedge i < \mathbf{n})\varphi = \#(|x| = \mathbf{n} \wedge i < \mathbf{n})(|x| = \mathbf{n} \wedge i < \mathbf{n} \wedge \varphi)$$

From this it follows that for any φ with free variable y,

$$\#(|x| = \mathbf{n} \wedge i < \mathbf{n})\varphi[s/y] = \#(|x| = \mathbf{n} \wedge i < \mathbf{n})\varphi[t/y],$$

The interpretation of the conclusion of the UNIV rule follows as a special case.

For SUB, we have from the interpretation of the premise that

$$\forall A \forall z (\Pr_{b_1,\dots,b_k}[A(u) = 1] - \Pr_{c_1,\dots,c_l}[A(u') = 1]) \leq \frac{1}{|z|}$$

For any A, we specialize this formula to poly-time term $\lambda x.A(v(x))$ (such a poly-time function exists, via the COMP axiom of T), obtaining

$$\forall z (\Pr_{d_1,\dots,d_m,b_1,\dots,b_k}[A(v(t)) = 1] - \Pr_{d_1,\dots,d_m,c_1,\dots,c_l}[A(v(s)) = 1]) \leq \frac{1}{|z|}.$$

Since A is arbitrary, we have

$$\forall A \forall z (\Pr_{d_1,\dots,d_m,b_1,\dots,b_k}[A(v(t)) = 1] - \Pr_{d_1,\dots,d_m,c_1,\dots,c_l}[A(v(s)) = 1]) \leq \frac{1}{|z|},$$

giving the interpretation of the conclusion of the rule.

EDIT follows fairly directly from the properties of $\#$. For H-IND, we use the telescoping sum principle C'12. Let t be the probabilistic term in the premise the H-IND rule. For concreteness, suppose that t has the form let $x \leftarrow \mathsf{rs}(\mathbf{n})$ in let $j < \mathbf{n}$ in t'. Let C be the counting term $\#(|x| = n \wedge j < \mathbf{n})(A(t) = 1)$. From C'12 we have

$$C[\mathbf{n}/i] - C[0/i] = \#(i < \mathbf{n})C[i + 1/i] - \#(i < \mathbf{n})C.$$

This is sufficient for establishing the interpretation of H-IND in T.

4.2 A sample proof relating psuedorandomness and next-bit unpredictability

We now give a proof in the system for computational indistinguishability, that next-bit unpredictability implies pseudorandomness. Suppose that $|f(x)| = p(\mathbf{n})$, where $p(\mathbf{n}) > \mathbf{n}$. We say that f is ($p(\mathbf{n})$-)*next bit unpredictable* if

$$\mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(p(\mathbf{n})) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \end{pmatrix} \mathsf{in}\ f(x)_{\{1\dots i+1\}} \approx \mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(p(\mathbf{n})) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ B \leftarrow \mathsf{rs}(1) \end{pmatrix} \mathsf{in}\ f(x)_{\{1\dots i\}} \circ B$$

Note that in order to define this notion in our system, we have deviated from the original definition of [6] which is not presented in terms of computational indistinguishability. However, it is not hard to see that our definition is indeed equivalent to the more standard one.

We now give a proof in our system of the result of [31], that if f is $2\mathbf{n}$-next bit unpredictable then f is a pseudorandom generator with stretch $\mathbf{n}$. Define the hybrid function h by $h(z, x, i) = f(x)_{\{1\dots i\}} \circ z_{\{1\dots 2\mathbf{n}-i\}}$. We then have

$$\begin{aligned}
\mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(2\mathbf{n}) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ z \leftarrow (2\mathbf{n}) \end{pmatrix} \mathsf{in}\ h(z, x, i) &\approx \mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(2\mathbf{n}) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ z \leftarrow (2\mathbf{n}) \end{pmatrix} \mathsf{in}\ f(x)_{\{1\dots i\}} \circ z_{\{1\dots 2\mathbf{n}-i\}} && \text{(by UNIV)} \\
&\approx \mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(2\mathbf{n}) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ z \leftarrow (2\mathbf{n}) \\ B \leftarrow \mathsf{rs}(1) \end{pmatrix} \mathsf{in}\ f(x)_{\{1\dots i\}} \circ B \circ z_{\{1\dots 2\mathbf{n}-(i+1)\}} && \text{(by EDIT)} \\
&\approx \mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(2\mathbf{n}) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ z \leftarrow (2\mathbf{n}) \end{pmatrix} \mathsf{in}\ f(x)_{\{1\dots i+1\}} \circ z_{\{2\mathbf{n}-(i+1)\}} \\
& && \text{(by SUB, since } f \text{ is } p\text{-next bit unpredictable)} \\
&\approx \mathsf{let}\begin{pmatrix} i \leftarrow \mathsf{rand}(2\mathbf{n}) \\ x \leftarrow \mathsf{rs}(\mathbf{n}) \\ z \leftarrow (2\mathbf{n}) \end{pmatrix} \mathsf{in}\ h(x, i+1) && \text{(by UNIV)}
\end{aligned}$$

Now since $h(z, x, 0) = z$ and for z with $|z| = 2\mathbf{n}$, $h(z, x, 2\mathbf{n}) = f(x)$, it follows by H-IND, EDIT and SUB that let $x \leftarrow \mathsf{rs}(\mathbf{n})$ in $f(x) \approx rs(2\mathbf{n})$, as required.

5 Conclusions and further work

The simple examples we have given here demonstrate that it is possible to give "direct" proofs, not involving the use of asymptotic notions, of a standard cryptographic construction. In the case of the system with a primitive relation for computational indistinguishability, we were also able to avoid explicit reasoning about probabilities.

There are a number of ways in which we could strengthen the systems presented here, for example: adding type-two functionals (i.e. functions which take functions as inputs) to the first-order system in order to allow reasoning about protocols and adaptive message attacks through the use of so-called *oracle adversaries*; developing proof-theoretic techniques which allow us to extract quantitative information about reductions from correctness proofs; looking at specialized formal systems for other primitive notions (e.g., "one-wayness"). Other directions for future work include examining the strength of the systems we have presented, e.g., by relating the power of the logic for computational indistinguishability to the Abadi-Rogaway logic, or by providing sufficient conditions for protocols to be provably sound.

One way of viewing the systems that we have presented is as a layer between low-level cryptography and higher-level systems for reasoning about security. It would be interesting to formalize this idea, say by providing a proof of the universal composability [10] of a primitive in one of our systems (or an extension thereof), and then using a more abstract system (essentially using information-theoretic reasoning) to reason about the security of protocols which use the primitive. In this way we could provide an end-to-end, cryptographically sound, formalization of security proofs.

We have not considered questions about the decidability of the systems we presented, nor have we seriously considered practical means for automated reasoning using them. While decidability in itself is not a particularly interesting question, automation is an area which demands further investigation.

Acknowledgements

We would like to thank Jan Krajíček, Daniele Micciancio, Bogdan Warinschi and the anonymous referees for many helpful comments and suggestions.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[3] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR preprint, available at `http://eprint.iacr.org/2003/015/`.

[4] M. Bellare. Practice-oriented provable-security. In E. Okamoto, G. Davida, and M. Mambo, editors, *Proceedings of the 1st International Workshop on Information Security (ISW 97)*, volume 1396 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[5] M. Blum. Coin flipping by telephone: a protocol for solving impossible problems. In *Proceedings of the 24th IEEE Spring Computer Conference, COMPCON*, pages 133–137, 1982.

[6] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[7] R. Boppana and R. Hirschfeld. Pseudo-random generators and complexity classes. In S. Micali, editor, *Advances in Computer Reserch*, volume 5, pages 1–26. JAI Press, 1989.

[8] M. Burrows, M. Abadi, and R Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.

[9] S.R. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986.

[10] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computing*, pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.

[11] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004.

[12] A. Cobham. The intrinsic computational difficulty of functions. In A. Bar Hillel, editor, *Proceedings of the International Conference on Logic, Methodology and Philosophy*, pages 24–30. North Holland, 1965.

[13] S.A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the 7th Annual ACM Symposium on the Theory of Computing*, pages 83–97, 1975.

[14] S.A. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):103–200, 1993.

[15] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.

[16] H.B. Enterton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[17] R. Fagin, J. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[18] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[19] H.J. Keisler. *Elementary Calculus: An Approach Using Infinitesimals*. Prindle, Weber & Schmidt, Boston, MA, second edition, 1986.

[20] D. Kozen. Indexings of subrecursive classes. *Theoretical Computer Science*, 11:277–301, 1980.

[21] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[22] D. Micciancio and S. Panjwani. Adaptive security of symbolic encryption. In J. Kilian, editor, *Proceeedings of the 2005 Theory of Cyptography Conference*, number 3378 in Lecture Notes in Computer Science, pages 169–187. Springer-Verlag, 2005.

[23] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[24] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *Proceedings of the 2004 Theory of Cryptography Conference*, number 2951 in Lecture Notes in Computer Science, pages 133–151. Springer-Verlag, 2004.

[25] J.C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.

[26] M. Naor. Bit commitment using pseudo-randomness. *Jounral of Cryptology*, 4:151–158, 1991.

[27] R. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic*, 36(3):494–508, 1971.

[28] P.Y.A. Ryan and S.A. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Pearson Education Ltd., Harlow, England, 2001.

[29] R. Smullyan. *Theory of Formal Systems*, volume 47 of *Annals of Mathematical Studies*. Princeton University Press, Princeton, NJ, 1961.

[30] A.J. Wilkie. Modèles non standard de l'arithmétique, et complexité algorithmique. In J.P. Ressayre and A.J. Wilkie, editors, *Modèles Non Standard en l'Arithmétique et Théorie des Ensembles*, pages 1–45. Publications Mathématiques de l'Université Paris VII, 1985.

[31] A.C. Yao. Theory and applications of trapdoor functions (extended abstract). In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

A Properties of Counting Terms

In this appendix we give outlines for proofs of the counting principles given in Section 3.4. While the proofs are not fully formalized in T, they are given with a sufficient level of detail to make formalization relatively straightforward.

Proposition A.0.1

$$T \vdash |x| = 0 \to x = 0$$

Proof: Suppose that $x \neq 0$. By B32, $|x| = |\lfloor \frac{x \dot{-} 1}{2} \rfloor| + 1$. Let y denote $|\lfloor \frac{x \dot{-} 1}{2} \rfloor|$. Suppose that $|x| = 0$. Then $y + 1 = 0$, so by B3, $y + 1 \leq y$. Now by B6 and B1, $y \leq y + 1$. So by B7, $y = y + 1$. But this contradicts B2. ■

Proposition A.0.2 *(C'4) For any counting term C,*

$$T \vdash \#(|x| = 0)C = C[0/x]$$

Proof: Say $C = \#(|\vec{x}| = |\vec{y}|)\varphi$. Consider any $x, \vec{x}$ satisfying $|x| = 0 \wedge |\vec{x}| = |\vec{y}| \wedge \varphi$. By A.0.1, $x = 0$, so $|\vec{x}|$ satisfies $\varphi[0/x]$. It follows by C3 that $\#(|x| = 0)C \leq C[0/x]$. It also follows by B9 and C3 that $C[0/x] \leq \#(|x| = 0)C$. ■

Proposition A.0.3 *(C'5) For any counting term C,*

$$T \vdash \#(|x| = |y| + 1)C = \#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y)C[2x + 2/x]$$

Proof: Suppose that C is $\#(|\vec{x}| = |\vec{y}|)\varphi$. We first show that

$$T \vdash \#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y)C[2x + 2/x] \leq \#(|x| = |y| + 1 \wedge |\vec{x}| = |\vec{y}|)\varphi \qquad (*)$$

Assuming

$$|x| = |y| \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[2x + 1/x]$$

we have

$$T \vdash z = 2x + 1 \to (|z| = |y| + 1 \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[z/x] \wedge z = 2 \cdot \left\lfloor \frac{z \dot{-} 1}{2} \right\rfloor + 1).$$

It then follows by C3 (taking $r(x, \vec{x}, z, \vec{y}) = 0$ iff $z = 2x + 1$ and $\vec{y} = \vec{x}$), that

$$T \vdash \#(|x| = |y|)C[2x + 1/x] \leq \#(|x| = |y| + 1 \wedge |\vec{x}| = |\vec{y}|)(\varphi \wedge x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1)$$

Similarly, we may show that

$$T \vdash \#(|x| = |y|)C[2x + 2/x] \leq \#(|x| = |y| + 1 \wedge |\vec{x}| = |\vec{y}|)(\varphi \wedge x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 2)$$

By B33 we then have

$$T \vdash \#(|x| = |y|)C[2x + 2/x] \leq \#(|x| = |y| + 1 \wedge |\vec{x}| = |\vec{y}|)(\varphi \wedge \neg(x = 2 \cdot \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1))$$

The inequality (*) now follows by C2. To obtain equality, we must show

$$T \vdash \#(|x| = |y| + 1)C \leq \#(|x| = |y|)C[2x + 1/x] + \#(|x| = |y)C[2x + 2/x]$$

By C2,

$$T \vdash \#(|x| = |y|+1)C \leq \#(|x| = |y|+1 \wedge |\vec{x}| \leq |\vec{y}|)(\varphi \wedge x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1)$$
$$+\#(|x| = |y|+1 \wedge |\vec{x}| \leq |\vec{y}|)(\varphi \wedge \neg(x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1))$$

But we can show by C3,

$$T \vdash \#(|x| = |y|+1 \wedge |\vec{x}| \leq |\vec{y}|)(\varphi \wedge x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1) \leq C[2x+1/x]$$

and by C3 and B33,

$$T \vdash \#(|x| = |y|+1 \wedge |\vec{x}| \leq |\vec{y}|)(\varphi \wedge \neg(x = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor + 1)) \leq C[2x+2/x]$$

■

Proposition A.0.4 *(C'6) For any counting term C such that x is not free in C,*

$$\#(|x| = |y|)C = (y \otimes 1) \cdot C$$

Proof: By C'4, we have $\#(|x| = |0|)C = C[0/x] = C = (0 \otimes 1) \cdot C$. Assuming that when $|y| = k$ we have $\#(|x| = |y|)C = (y \otimes 1) \cdot C$, we obtain

$$\begin{aligned}
\#(|x| = |2y+1|)C &= \#(|x| = |y|)C[2x+1/x] + \#(|x| = |y|)C[2x+2/x] && \text{(by C'5)}\\
&= \#(|x| = |y|)C + \#(|x| = |y|)C && \text{(since } x \text{ is not free in } C)\\
&= (y \otimes 1) \cdot C + (y \otimes 1) \cdot C && \text{(by assumption)}\\
&= ((2y+1) \otimes 1) \cdot C
\end{aligned}$$

So when $|y| = k+1$ we have $\#(|x| = |y|)C = (y \otimes 1) \cdot C$. The result then follows by LIND^o. ■

Proposition A.0.5 *(C'7)*

$$T \vdash |z| = |y| \rightarrow \#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi) = \#(|\vec{x}| = |\vec{y}|)\varphi[z/x]$$

Proof: From the properties of $=$, it follows immediately from C3 (taking $r(x, \vec{x}, \vec{y}) = 0$ iff $\vec{y} = \vec{x}$ and $x = z$) that for any z,

$$T \vdash \#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi) \leq \#(|\vec{x}| = |\vec{y}|)\varphi[z/x].$$

Similarly, it follows by C3, that

$$T \vdash |z| = |y| \rightarrow \#(|\vec{x}| = |\vec{y}|)\varphi[z/x] \leq \#(|x| = |y| \wedge |\vec{x}| = |\vec{y}|)(x = z \wedge \varphi).$$

■

For C'8, we first need to establish some facts about $\circ$, drop and lv.

Proposition A.0.6

$$T \vdash x \circ y = y + x \cdot (1 \otimes y)$$

Proof: Use induction on $|y|$. When $|y| = 0$, we have $y = 0$ by A.0.1, so that $x \circ y = x = y + x \cdot (1 \otimes y)$. Now suppose the result holds for any z with $|z| = k$. Consider y with $|y| = k + 1$, say $y = 2z + 1$ where $|z| = k$. We then have

$$\begin{aligned}
x \circ y &= 2 \cdot (x \circ z) + 1 \\
&= 2 \cdot (z + x \cdot (1 \otimes z)) + 1 && \text{(by IH)} \\
&= (2z + 1) + 2x \cdot (1 \otimes z) && \text{(by B29, B23, B20)} \\
&= (2z + 1) + x \cdot 2(1 \otimes z) && \text{(by B27, B28)} \\
&= (2z + 1) + x \cdot (1 \otimes (2z + 1)) && \text{(by B14)} \\
&= y + x \cdot (1 \otimes y)
\end{aligned}$$

■

Proposition A.0.7

$$T \vdash \left\lfloor \frac{\mathsf{drop}(x, y) \dot{-} 1}{2} \right\rfloor = \mathsf{drop}\left(\left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor, y \right)$$

Proof: We use induction on $|y|$. When $|y| = 0$, we have $y = 0$ by A.0.1 so that

$$\left\lfloor \frac{\mathsf{drop}(x, y) \dot{-} 1}{2} \right\rfloor = \left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor = \mathsf{drop}\left(\left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor, y \right)$$

Now suppose the result holds for any z with $|z| = k$. Consider y with $|y| = k + 1$, say $y = 2z + 1$ where $|z| = k$. We then have

$$\begin{aligned}
\left\lfloor \frac{\mathsf{drop}(x, y) \dot{-} 1}{2} \right\rfloor &= \left\lfloor \frac{\mathsf{drop}(x, 2z + 1) \dot{-} 1}{2} \right\rfloor \\
&= \left\lfloor \frac{\left\lfloor \frac{\mathsf{drop}(x, y) \dot{-} 1}{2} \right\rfloor \dot{-} 1}{2} \right\rfloor \\
&= \left\lfloor \frac{\mathsf{drop}\left(\left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor, y \right) \dot{-} 1}{2} \right\rfloor && \text{(by IH)} \\
&= \mathsf{drop}\left(\left\lfloor \frac{x \dot{-} 1}{2} \right\rfloor, y \right)
\end{aligned}$$

■

Proposition A.0.8

$$T \vdash |y| = |z| \to \mathsf{drop}(x \circ y, z) = x$$

Proof: Use induction on $|z|$. When $|z| = 0$, we have $y = z = 0$ by A.0.1, so that $\mathsf{drop}(x \circ y, z) = \mathsf{drop}(x \circ 0, 0) = x$. When $|y| = |z| = k$, say $z = 2u + 1$, $y = 2v + 1$, $|u| = |v| = k$,

$$\begin{aligned}
\mathsf{drop}(x \circ y, z) &= \mathsf{drop}(x \circ (2v + 1), 2u + 1) \\
&= \left\lfloor \frac{\mathsf{drop}(2 \cdot (x \circ v) + 1, u) \dot{-} 1}{2} \right\rfloor \\
&= \mathsf{drop}\left(\left\lfloor \frac{2 \cdot (x \circ v) + 1 \dot{-} 1}{2} \right\rfloor, u \right) && \text{(by the preceding proposition)} \\
&= \mathsf{drop}(x \circ v, u) && \text{(by B32)} \\
&= x && \text{(by IH)}
\end{aligned}$$

■

Proposition A.0.9

$$T \vdash |y| = |z| \to \mathsf{lv}(x \circ y, z) = y$$

Proof: Suppose that $|y| = |z|$. We have

$$\begin{aligned}
\mathsf{lv}(x \circ y, z) &= (x \circ y) \dot{-} \mathsf{drop}(x \circ y, z) \cdot (1 \otimes z) \\
&= (x \circ y) \dot{-} x \cdot (1 \otimes z) && \text{(by A.0.8)} \\
&= y + x \cdot (1 \otimes y) \dot{-} x \cdot (1 \otimes z) && \text{(by A.0.6)} \\
&= y && \text{(by B15,B16 and B35)}
\end{aligned}$$

■

Proposition A.0.10

$$T \vdash |x| = |y_1 \circ y_2| \to \exists x_1 \exists x_2 (|x_1| = |y_1| \wedge |x_2| = |y_2| \wedge x = x_1 \circ x_2)$$

Proof: Use induction on $|y_2|$, along with axiom B10. ■

Proposition A.0.11

$$T \vdash |x| = |y_1 \circ y_2| \to (|\mathsf{drop}(x, y_2)| = |y_1| \wedge |\mathsf{lv}(x, y_2)| = |y_2|)$$

Proof: Immediate from A.0.10,A.0.8 and A.0.9. ■

Proposition A.0.12 *(C'9)*

$$T \vdash \#(|x_1| = |y_1| \wedge |x_2| = |y_2|)\varphi = \#(|x| = |y_1 \circ y_2|)\varphi[\mathsf{drop}(x, y_2)/x_1, \mathsf{lv}(x, y_2)/x_2]$$

Proof: The proof will use counting axiom C3, along with A.0.8 and A.0.9. Let C_1 denote $\#(|x_1| = |y_1| \wedge |x_2| = |y_2|)\varphi$ and C_2 denote $\#(|x| = |y_1 \circ y_2|)\varphi[\mathsf{drop}(x, y_2)/x_1, \mathsf{lv}(x, y_2)/x_2]$. To begin, we note that it is not hard to show, by induction on $|y_2|$, that if $|x_1| = |y_1|$ and $|x_2| = |y_2|$, then $|x_1 \circ x_2| = |y_1 \circ y_2|$, and that $\circ$ is injective, that is, if $|x| = |x'|$, $|y| = |y'|$ and $x \circ y = x' \circ y'$, then $x = x'$ and $y = y'$. Now to show that $C_1 \leq C_2$ it suffices, as $\circ$ is injective, to show that for every x_1 and x_2 satisfying φ, where $|x_1| = |y_1|$ and $|x_2| = |y_2|$, there is some x with length $|y_1 \circ y_2|$ which satisfies $\varphi[\mathsf{drop}(x, y_2)/x_1, \mathsf{lv}(x, y_2)/x_2]$. Let $x = x_1 \circ x_2$. As stated above, $|x| = |y_1 \circ y_2|$. Also by A.0.8, $\mathsf{drop}(x, y_2) = x_1$ and by A.0.9,$\mathsf{lv}(x, y_2) = x_2$, and $C_1 \leq C_2$ follows by C3, taking $r(x_1, x_2, x) = 0$ iff $x = x_1 \circ x_2$. We now want to show that $C_2 \leq C_1$. Consider any x with $|x| = |y_1 \circ y_2|$ such that x satisfies $\varphi[\mathsf{drop}(x, y_2)/x_1, \mathsf{lv}(x, y_2)/x_2]$. By A.0.10, there are x_1, x_2 with $|x_1| = |y_1|$, $|x_2| = |y_2|$ and $x = x_1 \circ x_2$ ■

For C'10, we require the following fact about $\oplus$:

Proposition A.0.13

$$T \vdash |v| = |x| \to (x \oplus v) \oplus v = x$$

Proof: Use LIND^o on $|v|$. When $|v| = 0$, we have $v = 0$ by A.0.1, and the result is immediate from the definition of $\oplus$. Now suppose that the result holds for any v' with $v' = k$, and consider v with $|v| = k + 1$, say $v = 2v' + 1$. Since $|x| = |v| = k + 1$, it follows from B32 and B33 that for some x' with $|x'| = k$, $x = 2x' + 1$ or $x = 2x' + 2$. Consider the case where $x = 2x' + 2$. We have

$$\begin{aligned}
(x \oplus v) \oplus v &= (2x' + 2 \oplus 2v' + 1) \oplus 2v' + 1 \\
&= (2 \cdot (x' \oplus v') + 2) \oplus 2v' + 1 && \text{(by definition of } \oplus) \\
&= (2 \cdot ((x' \oplus v') \oplus v')) + 2 && \text{(by definition of } \oplus) \\
&= 2x' + 2 && \text{(by IH)} \\
&= x
\end{aligned}$$

The other cases are similar. ■

Corollary A.0.14 $T \vdash x \neq x' \to x \oplus v \neq x' \oplus v$

Proposition A.0.15 *(C'10)*

$$T \vdash |v| = |y| \to \#(|x| = |y|)C = \#(|x| = |y|)C[x \oplus v/x]$$

Proof: We first note that it is not hard to show using LIND^o that if $|v| = |x|$ then $|x \oplus v| = |x|$ (*). Suppose that C has the form $\#(|\vec{x}| = |\vec{y}|)\varphi$. Fix $|v| = |y|$, and assume

$$|x| = |y| \wedge |\vec{x}| = |\vec{y}| \wedge \varphi.$$

By A.0.13 and (*), it follows that

$$T \vdash |x \oplus v| = |y| \wedge |\vec{x}| = |\vec{y}| \wedge \varphi[(x \oplus v) \oplus v/x]$$

So by A and C3,

$$T \vdash \#(|x| = |y|)C \leq \#(|x| = |y|)C[x \oplus v/x].$$

It also follows immediately by A and C3 that

$$T \vdash \#(|x| = |y|)C[x \oplus v/x] \leq \#(|x| = |y|)C$$

■

To prove $C'11$, we will require the following

Lemma A.0.16 *The following are both theorems of T, for any formula φ:*

1. $\forall x((|x| = |2k+1| \wedge \mathsf{bin}(x) \leq k) \to \exists x((|x| = |2(k+1)+1| \wedge \mathsf{bin}(x) \leq k)))$
2. $\forall x((|x| = |2(k+1)+1| \wedge \mathsf{bin}(x) \leq k) \to \exists x((|x| = |2k+1| \wedge \mathsf{bin}(x) \leq k)))$

Proof: We begin by defining the functions zpl ("zero pad left") and zdl ("zero drop left") as follows: $\mathsf{zpl}(x) = x + (x \otimes 1)$, $\mathsf{zdl}(x) = x - (x \otimes 1)$. Also, bin^{-1} is defined such that $\mathsf{bin}^{-1}(l, x)$, assuming that $l \geq |2x+1|$, returns the binary notation of x, zero padded on the left to length l. Clearly, the dyadic representation of $\mathsf{zpl}(x)$ is just the dyadic representation of x padded on the left by a single $\mathsf{0}$, while if the dyadic representation of x starts with a $\mathsf{0}$, then $\mathsf{zdl}(x)$ is the dyadic representation of x with the leading $\mathsf{0}$ removed. Recall that $\mathsf{bin}(x) = x - (x \otimes 1) + 1$. It follows from the *BASIC* axioms that for all x, $\mathsf{bin}(x) = \mathsf{bin}(\mathsf{zpl}(x))$. By the definition of bin and zpl, this is equivalent to showing

$$x - (x \otimes 1) + 1 = x + (x \otimes 1) - ((x + (x \otimes 1)) \otimes 1) + 1.$$

To begin, we may show using LIND^o that $|x + (x \otimes 1)| = |2x+1|$. When $|x| = 0$, we have $x = 0$ by A.0.1, so that

$$\begin{aligned} |x + (x \otimes 1)| &= |0 + (0 \otimes 1)| \\ &= |0 + 1| && \text{(by B13)} \\ &= |2 \cdot 0 + 1| && \text{(by B25)} \\ &= |2x + 1| \end{aligned}$$

Now suppose that this equation holds for y with $|y| = k$. Suppose $|x| = k+1$. If $x = 2y+1$,

$$\begin{aligned} |x + (x \otimes 1)| &= |2y + 1 + ((2y+1) \otimes 1)| \\ &= |2y + 1 + 2 \cdot (y \otimes 1)| && \text{(by B15 and B14)} \\ &= |2(y + (y \otimes 1)) + 1| && \text{(by B20 and B28)} \\ &= |y + (y \otimes 1)| + 1 && \text{(by B10)} \\ &= |2y + 1| + 1 && \text{(by Induction Hytpothesis)} \\ &= |2(y+1) + 1| && \text{(by B10)} \\ &= |2x + 1| \end{aligned}$$

We can give a similar proof when $x = 2y+2$. Similarly, we may show that for any x and k such that $|x| = |2(k+1)+1|$ and $b(x) \leq k$, $\mathsf{bin}(\mathsf{zdl}(x)) = \mathsf{bin}(x)$. The proposition then follows immediately. ■

Proposition A.0.17 *(C'11) For any counting term C, the following is a theorem of T:*

$$\#(i \leq k+1)C = C[k+1/i] + \#(i \leq k)C.$$

Proof: Suppose that C has the form $\#(|\vec{x}| = |\vec{y}|)\varphi$. Let $l = 2(k+1)+1$ and $l' = 2k+1$. We then have

$$\begin{aligned}
&\#(i \leq k+1)C \\
&\quad = \#(|x| = |l| \wedge |\vec{x}| = |\vec{y}|)(\mathsf{bin}(x) \leq k+1 \wedge \varphi[\mathsf{bin}(x)/i]) && \text{(by Definition)} \\
&\quad = \#(|x| = |l| \wedge |\vec{x}| = |\vec{y}|)((x = \mathsf{bin}^{-1}(l, k+1) \vee \mathsf{bin}(x) \leq k) \wedge \varphi[\mathsf{bin}(x)/i]) && \text{(by C3)} \\
&\quad = C[k+1/i] + \#(|x| = |l|)(\mathsf{bin}(x) \leq k \wedge \varphi[\mathsf{bin}(x)/i]) && \text{(by C'7,C'8 and C3)} \\
&\quad = C[k+1/i] + \#(|x| = |l'|)(\mathsf{bin}(x) \leq k \wedge \varphi[\mathsf{bin}(x)/i]) && \text{(by Lemma A.0.16 and C3)} \\
&\quad = C[k+1/i] + \#(i \leq k)C && \text{(by Definition)}
\end{aligned}$$

Proposition A.0.18

$$\forall m \leq |z|(\forall i < m(C(i) = C'(i+1)) \rightarrow \#(i \leq m)C - \#(i \leq m)C' = C(m) - C'(0))$$

Proof. We use LIND^o on m. When $m = 0$, the result follows by C'4. Now suppose that it holds for $m = k$. We will show that the result holds for $m = k+1$. Suppose that for all $i < k+1$, $C(i) = C'(i+1)$ (*). Then

$$\begin{aligned}
&\#(i < m+1)C - \#(i < m+1)C' \\
&\quad = \#(i < k+2)C - \#(i < k+2)C' \\
&\quad = C(k+1) + \#(i < k+1)C - (C'(k+1) + \#(i < k+1)C') && \text{(by C'11)} \\
&\quad = C(k+1) - C'(k+1) + C(k) - C'(0) && \text{(by IH)} \\
&\quad = C(k+1) - C(k) + C(k) - C'(0) && \text{(by *)} \\
&\quad = C(k+1) - C'(0)
\end{aligned}$$

Note that by choosing appropriate C and C', we obtain (via C'6)

Corollary A.0.19

$$\forall m \leq |z|(\forall i < m(2^{|u|}C(i) = 2^{|v|}C'(i+1)) \rightarrow \#(i \leq m)2^{|u|}C - \#(i \leq m)2^{|v|}C' = 2^{|u|}C(m) - 2^{|v|}C'(0))$$

This is the form of the principle that we actually use in the proof of Theorem 3.5.2