A Quantifier-Free String Theory for *ALOGTIME* Reasoning

by

François Pitt

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2000 by François Pitt

Abstract

A Quantifier-Free String Theory for *ALOGTIME* Reasoning

François Pitt
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2000

The main contribution of this work is the definition of a quantifier-free string theory T_1 suitable for formalizing *ALOGTIME* reasoning. After describing L_1—a new, simple, algebraic characterization of the complexity class *ALOGTIME* based on strings instead of numbers—the theory T_1 is defined (based on L_1), and a detailed formal development of T_1 is given.

Then, theorems of T_1 are shown to translate into families of propositional tautologies that have uniform polysize Frege proofs, T_1 is shown to prove the soundness of a particular Frege system $\mathcal{F}$, and $\mathcal{F}$ is shown to provably p-simulate any proof system whose soundness can be proved in T_1. Finally, T_1 is compared with other theories for *ALOGTIME* reasoning in the literature.

To our knowledge, this is the first formal theory for *ALOGTIME* reasoning whose basic objects are strings instead of numbers, and the first quantifier-free theory formalizing *ALOGTIME* reasoning in which a direct proof of the soundness of some Frege system has been given (in the case of first-order theories, such a proof was first given by Arai for his theory *AID*). Also, the polysize Frege proofs we give for the propositional translations of theorems of T_1 are considerably simpler than those for other theories, and so is our proof of the soundness of a particular $\mathcal{F}$-system in T_1. Together with the simplicity of T_1's recursion schemes, axioms, and rules these facts suggest that T_1 is one of the most natural theories available for *ALOGTIME* reasoning.

Acknowledgements

First and foremost, I would like to thank my advisor, Stephen Cook, without whose help, patience, and guidance this work would have been impossible; Alasdair Urquhart, Alan Borodin, and Charles Rackoff (the other members of my advisory committee) for all their suggestions; and Samuel Buss (my external appraiser) for taking the time to read the whole thesis and saying such nice things about it!

I would also like to thank Stephen Bellantoni and Stephen Bloch, for some interesting and fruitful discussions on NC and NC^1; Arnold Rosenbloom, for all the discussions; the staff of the Department of Computer Science, for all their help with administrative matters; and The Fields Institute, for their program on Computational Complexity.

Last, but certainly not least, I would like to thank my parents and my wife, Marie-Josée, for their patience and encouragement over the last six years, as well as for everything else.

Contents

Index of Function Symbols

(*Slanted* page numbers indicate L_1 definitions, **bold** page numbers indicate T_1 definitions, plain page numbers indicate T_1 axioms or theorems.)

Chapter 1

Introduction

The starting point for this work is the following open problem in complexity theory, concerning propositional proof systems (for a good introduction to propositional proof systems, including the basic definitions, see Cook and Reckhow [19]).

Open Problem 1 *Are Frege ("$\mathcal{F}$") and extended Frege ("$e\mathcal{F}$") proof systems p-equivalent?*

To provide some motivation for studying Open Problem 1 and to give an indication of its importance, note its connection to some major open questions in complexity theory through the following facts.

General Fact 1 *If $NP \neq coNP$, then $P \neq NP$.*

General Fact 2 *$NP = coNP$ if and only if* TAUT $\in NP$.

General Fact 3 TAUT $\in NP$ *if and only if there exists a* super (i.e., *polynomially-bounded) proof system for* TAUT.

General Fact 4 *Given two proof systems f_1 and f_2, if f_1 is super and f_2 p-simulates f_1, then f_2 is also super.*

From Cook and Reckhow's paper, we know that p-simulation imposes a partial order on proof systems. Determining the relative position of particular proof systems in this order helps shed some light on their relative power and, because of General Fact 4, on such major open problems as $NP \overset{?}{=} coNP$ or $P \overset{?}{=} NP$. From this point of view, determining the exact position of Frege systems relative to extended Frege systems in this order is one of the most important questions still open in this area. For the rest of this chapter, I will give a short survey of the major results and issues connected with Open Problem 1.

1.1 $e\mathcal{F}$-systems and P

It is traditional in complexity theory to equate "feasible" with "polynomial time". Moreover, there is a close association between polytime and $e\mathcal{F}$-systems since $e\mathcal{F}$-systems can be thought of as reasoning on uniform polysize circuits. For the rest of this work, I will use the traditional notation "P" when referring to the class of polytime decidable languages, and "FP" when referring to the class of polytime computable functions. Over the years, many characterizations of the classes P and FP have been given, most notably Cobham's "$\mathcal{L}$" and Bellantoni and Cook's "B".

- Cobham's $\mathcal{L}$ [16] is the first machine-independent characterization of the class FP using a form of bounded recursion on notation.
- Bellantoni and Cook's B [4] uses a tiered approach (*i.e.*, it distinguishes between "safe" and "normal" parameters) in order to dispense with explicit bounds as in Cobham's scheme.

Also, many logical theories have been proposed to capture polytime reasoning, most notably Cook's "PV", Buss's "S^1_2", Leivant's "$PT(\mathbb{W})$", and various second-order theories.

- Cook's PV [18, 20] is a free-variable equational theory based on Cobham's $\mathcal{L}$. Cook showed that every formula $t = u$ provable in PV gives rise to a family of propositional tautologies which assert the equation and have uniform polysize $e\mathcal{F}$ proofs, that PV can define and prove the soundness of $e\mathcal{F}$, and that if the soundness of a propositional proof system T is provable in PV, then $e\mathcal{F}$ p-simulates T.
- Buss's S^1_2 [7] is a system of Bounded Arithmetic that can define exactly the polytime functions. Buss showed that S^1_2 is Σ^b_1-conservative over PV (when its language is suitably extended to include all the function symbols of PV), which implies that S^1_2 proves the soundness of $e\mathcal{F}$ and that the Σ^b_1-theorems of S^1_2 can be translated into propositional tautologies that have uniform polysize $e\mathcal{F}$-proofs (by the corresponding results for PV).
- Leivant's $PT(\mathbb{W})$ [26] has generative axioms for $\mathbb{W}$ (intuitively, binary strings) and instances of $\mathbb{W}$-induction as its only axioms. It proves the convergence of exactly the polytime functions over $\mathbb{W}$ (when induction is restricted to positive existential formulas). This formalization is conceptually and technically very simple because it does not rely on any particular initial functions, other than the algebra's constructors (in fact, the theory can talk about any computable function).
- Buss's V^1_1 (studied by Razborov [27]) and Leivant's $L_2(QF^+)$ [25] are two of the most notable examples of second-order theories for P.

1.2 $\mathcal{F}$-systems and *ALOGTIME*

The computational power of $\mathcal{F}$-systems seems to be captured by the uniform class NC^1, since $\mathcal{F}$-systems can be thought of as reasoning on polysize formulas, which are the same as logarithmic-depth circuits. Recall that NC^1 is the class of languages decidable by families of logarithmic-depth circuits (FNC^1 is the functional equivalent, using multi-output circuits), and by results of Ruzzo [28], U_{E^*}-uniform $NC^1 = ALOGTIME$, where $ALOGTIME$ is the class of languages decidable in logarithmic time by a random access alternating Turing machine. The functional class *FALOGTIME* can be defined in two different ways: if functions are thought of as operating on integers in binary notation, we get a "numerical" version of the class, whereas if functions are thought of as operating on strings of bits (which is closer to the circuit model), we get a "string" version of the class. Fortunately, with a suitable interpretation of numbers as strings (or of strings as numbers), both versions are equivalent.

Therefore, for the rest of this work, I will use *ALOGTIME* and NC^1 interchangeably, always referring to the uniform version of the class (unless otherwise specified). Also, I will use "*FALOGTIME*" (or "FNC^1") to refer to the functional version of the class. Various characterizations of *FALOGTIME* have been given over the years, most notably Clote's "N_0" and "N_0'", and Bloch's string algebra.

- Clote's N_0 and N_0' [12, 13, 14] are "numerical" characterizations that use restricted forms of Cobham's recursion on notation. Unfortunately, N_0 includes a complete function for *FALOGTIME* as a base function, and N_0' depends on Barrington's deep result about bounded-width branching programs [3], so neither algebra is as natural for FNC^1 as Cobham's $\mathcal{L}$ is for FP.

- Bloch's algebra [5, 6] is a "string" characterization that uses the "safe" versus "normal" parameter idea together with a form of recursion similar to Allen's "divide and conquer recursion" (DCR) [1]. Bloch recognized that Allen's scheme of DCR (which Allen used to characterize uniform NC) is particularly well-suited to characterizing uniform parallel complexity classes. Combining this with the tiered approach allows him to dispense with explicit bounds on the rate of growth of functions and to give an elegant characterization that uses only simple base functions and one natural scheme of recursion.

Based on Bloch's ideas but incorporating some of Clote's, I will introduce in Chapter 2 a new simple string algebra L_1 that characterizes *FALOGTIME* using very few simple base functions and two simple schemes of recursion (CRN and TRN, to be defined there). It appears to us that L_1 is simpler than previous characterizations because it has fewer, simpler base functions, and no need for explicit bounds on the growth of functions or for different types of parameters.

Based on the characterizations of $ALOGTIME$ given above, a number of theories to capture $ALOGTIME$ reasoning have been defined, most notably Clote's "ALV" and "ALV'", Takeuti and Clote's "TNC^0", and Arai's "AID" (all of which are based on "numerical" characterizations of $ALOGTIME$).

- Clote's ALV and ALV' [13, 14] are free-variable equational theories based on his characterizations of $ALOGTIME$ mentioned above and on Cook's PV. Clote showed that theorems of ALV and ALV' give rise to families of tautologies which have polysize $\mathcal{F}$-proofs, but did not show that either of his theories can prove the soundness of $\mathcal{F}$-systems. Also, the proof that the propositional translations of theorems of ALV or ALV' have polysize $\mathcal{F}$-proofs is fairly involved, and properties of even simple functions (such as "parity" or "majority") are difficult to prove.

- Takeuti and Clote's TNC^0 [15] (first defined by Takeuti [29]) is a first-order theory similar to Buss's S_2^1 that was shown to be conservative over ALV' (when suitably extended to include every function symbol of ALV'). Unfortunately, this theory needs to use a fairly complex form of inference called *bounded successive nomination*, because of its implicit dependence on Barrington's result (through Clote's characterization of $ALOGTIME$), which detracts greatly from its simplicity.

- Arai's AID [2] is a system of bounded arithmetic inspired by Buss's consistency proof for $\mathcal{F}$-systems [9], which proves the soundness of $\mathcal{F}$ and whose Σ_0^b-theorems have polysize $\mathcal{F}$-proofs when suitably translated. Moreover, Arai shows that AID is equivalent to a quantified version of Clote's ALV, and hence that ALV can prove the soundness of $\mathcal{F}$.

Unlike the situation for P, there is no quantifier-free theory for $ALOGTIME$ which has the simplicity and naturalness of PV. I claim that T_1 fills that role, its axioms and induction schemes being based directly on L_1's simple base functions and natural recursion operations. Moreover, the proofs that propositional translations of the theorems of T_1 have uniform polysize $\mathcal{F}$-proofs and that T_1 can prove the soundness of $\mathcal{F}$-systems are much simpler than the corresponding proofs for other theories in the literature.

1.3 Overview

Now that I have provided some context and motivation for studying Open Problem 1, let me give a brief overview of the rest of the thesis. In Chapter 2, I will introduce the string algebra L_1, followed in Chapter 3 by the quantifier-free theory T_1 (including a formal development of the theory, showing how to prove the pigeonhole principle in T_1). In Chapter 4, I will define propositional translations for theorems of T_1 and show that they have polysize $\mathcal{F}$-proofs, while in Chapter 5, I will show that T_1 proves the soundness of $\mathcal{F}$, by formalizing an algorithm for the "Boolean Sentence Value Problem" (BSVP) in T_1, and that $\mathcal{F}$ provably p-simulates any proof system whose soundness can be proved in T_1. Finally, in Chapter 6, I will compare T_1 with various other formalisms for $ALOGTIME$ reasoning, most notably Arai's AID.

Chapter 2

The String Algebra L_1

In this chapter, we define L_1 and show that it contains exactly the functions in *FALOGTIME*. We also give many examples of natural L_1 definitions for simple *FALOGTIME* functions.

2.1 Basic definitions

The basic objects of the algebra are strings over the alphabet $\{0, 1\}$. The set of all such strings can be defined inductively: ε (the empty string), 0, 1 are strings, and if x and y are strings, then so is xy. Together with a wish for simplicity, this inductive definition motivates our choice of base functions.

The reader should keep in mind that our definitions in this chapter are based on, and guided by, the idea of computation by uniform families of circuits. In particular, all our functions will be *length-determined*, *i.e.*, the length of a function depends only on the lengths of the arguments, not their values. Also, the starting point for our algebra L_1 is Bloch's paper [6], where he carries out a similar function-algebraic characterization of *FALOGTIME*, so we will borrow many concepts and definitions from there. (We also borrow certain concepts and definitions from Clote's work [12, 13, 14].)

Now, we define the base functions and the basic operators that we will use to construct new functions. We use $|x|$ to denote the length of x (*i.e.*, the number of symbols (bits) in the string x), $\vec{x}_k$ to denote a k-tuple of variables, and $\vec{x}$ to denote an arbitrary tuple of variables.

BASE: The set of *base functions* consists of (in order of increasing arity):

$$\begin{aligned}
\varepsilon, 0, 1 &= \text{empty string, 0-bit, and 1-bit (constants)},\\
\blacktriangleright x &= \text{the } \lceil |x|/2 \rceil \text{ rightmost bits of } x \text{ (“right half”)},\\
x \lhd y &= x \text{ with } |y| \text{ bits removed from the right (“right chop”)},\\
x \cdot y &= x \text{ followed by } y \text{ (“concatenation”)},
\end{aligned}$$

$$x \mathbin{?} (y, z_0, z_1) = \begin{cases} y & \text{if } x = \varepsilon, \\ z_0 & \text{if } x = w \cdot 0 \text{ for some } w, \\ z_1 & \text{if } x = w \cdot 1 \text{ for some } w, \end{cases} \qquad (\text{“conditional”})$$

$$\mathcal{I}^n_k(x_1, \ldots, x_n) = x_k \quad \text{for any } 1 \le k \le n \quad (\text{“identity” or “projection”}).$$

REMARK 2.1.1 In the definition of $x \mathbin{?} (y, z_0, z_1)$, it is assumed that $|z_0| = |z_1|$. If that is not the case, then the value returned will be padded on the left with as many 0's as are necessary to make z_0 and z_1 the same length (the length of y does not change).

COMP: f is defined from g and $h_1, \ldots, h_k$ by *composition* if

$$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_k(\vec{x})).$$

CRN: f is defined from h by *concatenation recursion on notation* on x if $h(x, \vec{y}) \in \{0, 1\}$ for all $x, \vec{y}$ and

$$\begin{aligned} f(\varepsilon, \vec{y}) &= \varepsilon, \\ f(xi, \vec{y}) &= f(x, \vec{y}) \cdot h(xi, \vec{y}) \quad \text{for } i = 0, 1. \end{aligned}$$

TRN: f is defined from g, h, h_ℓ, and h_r by *tree recursion on notation* on x if

$$f(x, z, \vec{y}) = \begin{cases} g(x, z, \vec{y}) & \text{if } x = \varepsilon, 0, 1, \\ h\big(x, z, \vec{y}, f(x\blacktriangleleft, h_\ell(z), \vec{y}), f(\blacktriangleright x, h_r(z), \vec{y})\big) & \text{otherwise,} \end{cases}$$

where $x\blacktriangleleft = x \lhd \blacktriangleright x$ (the $\lfloor |x|/2 \rfloor$ leftmost bits of x). In what follows, we will omit the parameter z when neither g nor h depend on it (in which case the functions h_ℓ and h_r are irrelevant and will not be specified); we will refer to this form of TRN as *simple TRN*.

REMARK 2.1.2 Our “right half” function was called “back half (*Bh*)” by Allen [1] and Bloch [6]. We introduce the new nomenclature because we feel that it is more representative of the action of the function, and the new notation to serve as a graphical reminder of that action (picture the black triangle cutting into the left part of x). Similarly, our “right chop” function was called “chop” by Cook [20] and “most significant part (*Msp*)” by Allen and Bloch. Our new notation should serve as a useful graphical mnemonic for the function's purpose and action (picture the bits of y cutting into the bits of x from the right—in the direction pointed to by the function symbol). Our scheme of CRN is based on the operation of the same name in Clote's work [12, 13, 14], except that our version has been simplified by eliminating the function g from the base case (without loss of generality since we can simply concatenate $g(\vec{y})$ to the left of our functions to get Clote's). Our scheme of TRN is based on Bloch's "very safe DCR", which is itself based on Allen's "DCR" (for "divide-and-conquer recursion"), except that our base case is simpler (defined for $x = \varepsilon, 0, 1$ instead of when $|x| \le |b|$ for some extra parameter b), and we have added the functions h_ℓ and h_r that allow parameter z to vary during recursive calls (hence, TRN is technically a scheme of "recursion with replacement").

DEFINITION 2.1.1 If we let $\mathrm{TRN}\big|_{L'}^{L}$ represent the operation of TRN restricted to functions $g \in L$ and $h, h_\ell, h_r \in L'$, for function classes L and L', then

- L_0 is the closure of BASE under COMP and CRN;
- L_1 is the closure of L_0 under COMP, CRN, and $\mathrm{TRN}\big|_{L_0}^{L_1}$, defined recursively.

The next few sections contain mainly function definitions, where the following notational conventions will be used.

- For any constant string c, $\vec{c}_k$ represents the tuple consisting of k copies of c.
- Unary functions have higher precedence than binary functions and binary functions have higher precedence than functions of higher arity (keep in mind that "?" has arity 4). Concatenation has higher precedence than any other binary function when represented by juxtaposition; it has lower precedence than any other binary function when represented by "$\cdot$".
- i and j represent arbitrary fixed single bits, whereas k, ℓ, m, and n represent arbitrary fixed non-zero natural numbers. When 2^k is used, k ranges over all natural numbers (including zero), and similarly for 2^ℓ.
- The notation $k \times x$ stands for $\overbrace{x \cdots x}^{k}$ (*i.e.*, x concatenated with itself k times). We let $0 \times x = \varepsilon$ and use $\widehat{k}$ as an abbreviation for $k \times 1$, *i.e.*, the unary string representing k.

2.2 Functions in L_0

In this section, we define many functions in L_0 and show that many useful generalizations of CRN can be simulated in L_0. We are motivated by two goals: to define the machinery necessary to prove that L_1 contains all of FNC^1, and to show that many useful functions have simple definitions in our algebra.

2.2.1 Basic functions

First, we define a few simple variations on some of the BASE functions. The rightmost bit of x: $x' = x \mathbin{?} (\varepsilon, 0, 1)$; x with its rightmost bit removed: $x{\lessdot} = x \lhd 1$; the $\lfloor |x|/2 \rfloor$ leftmost bits of x: $x\blacktriangleleft = x \lhd \blacktriangleright x$.

Next, a function that reverses the bits of x can be defined by first using CRN to define a function $\mathsf{reverse}(x, y)$, which returns the $|y|$ rightmost bits of x reversed:

$$\begin{aligned} \mathsf{reverse}(x, \varepsilon) &= \varepsilon, \\ \mathsf{reverse}(x, yi) &= \mathsf{reverse}(x, y) \cdot (x \lhd y)' \quad \text{for } i = 0, 1. \end{aligned}$$

Then, $\mathsf{rev}(x) = \mathsf{reverse}(x, x)$ returns the reverse of x. Using this function, we can now define symmetric counterparts to some of the earlier functions:

$$y \rhd x = \mathsf{rev}(\mathsf{rev}(x) \lhd \mathsf{rev}(y)), \qquad {\gtrdot}x = \mathsf{rev}(\mathsf{rev}(x){\lessdot}), \qquad {}^{\backprime}x = \mathsf{rev}(x)'.$$

Now, let us introduce a generalization of CRN: a function f is defined from h by *left CRN* (or *reverse CRN*) on x if $h(x, \vec{y}) \in \{0, 1\}$ for all $x, \vec{y}$ and

$$\begin{aligned} f(\varepsilon, \vec{y}) &= \varepsilon, \\ f(ix, \vec{y}) &= h(ix, \vec{y}) \cdot f(x, \vec{y}) \quad \text{for } i = 0, 1. \end{aligned}$$

If f is defined from h by left CRN on x, then we can use CRN to define

$$\begin{aligned} \mathsf{aux\_}f(\varepsilon, \vec{y}) &= \varepsilon, \\ \mathsf{aux\_}f(xi, \vec{y}) &= \mathsf{aux\_}f(x, \vec{y}) \cdot h(\mathsf{rev}(xi), \vec{y}) \quad \text{for } i = 0, 1, \end{aligned}$$

and $f(x, \vec{y}) = \mathsf{rev}(\mathsf{aux\_}f(\mathsf{rev}(x), \vec{y}))$, using COMP. In what follows, we will use the notational conventions outlined before this section and we will no longer include the trivial base case $f(\varepsilon, \vec{y}) = \varepsilon$ or write "for $i = 0, 1$" when using CRN to define new functions.

2.2.2 String manipulation functions

Now, we will define useful functions for manipulating strings. First, two simple functions that returns a string of the same length as its input, but consisting entirely of 0's or entirely of 1's:

$${}_j(xi) = {}_jx \cdot j \qquad \text{(by CRN).}$$

Next, we can define a number of functions to compare the lengths of strings (these function symbols will be distinguished by putting a superscript "^{L}" next to them). First, it is useful to have a conditional that tests for the length of a string: $x \mathbin{?}^{ZL} (y, z) = x \mathbin{?} (y, z, z)$ is equal to y if x is empty and equal to z otherwise. Because $?^{ZL}$ distinguishes only between "empty" and "non-empty", we will define the "length-relational" functions below so that they return ε when the relation holds and some fixed non-empty string (like "1") otherwise. Accordingly, we define a simple signum function that returns ε if its argument is empty and 1 otherwise: $\approx^L x = x \, ?^{ZL} \, (\varepsilon, 1)$ and a corresponding "negation": $\neg^L x = x \, ?^{ZL} \, (1, \varepsilon)$. Now, we are ready to define the comparison functions.

$$x \geq^L y = \approx^L(x \rhd y) \qquad x \leq^L y = \approx^L(x \lhd y) \qquad x =^L y = \approx^L((x \rhd y) \cdot (x \lhd y))$$
$$x >^L y = \neg^L(x \leq^L y) \qquad x <^L y = \neg^L(x \geq^L y) \qquad x \neq^L y = \neg^L(x =^L y)$$
$$\mathsf{max}^L(x, y) = (x \geq^L y) \, ?^{ZL} \, (x, y) \qquad \mathsf{max}^L_1(x) = x$$
$$\mathsf{max}^L_{k+1}(x, \vec{x}_k) = \mathsf{max}^L\big(x, \mathsf{max}^L_k(\vec{x}_k)\big)$$

We can also define functions to manipulate the lengths of strings, namely $\mathsf{div}^L_{2^k}(x)$ that returns a string of 1's whose length is $\lfloor |x|/2^k \rfloor$ and a corresponding $\mathsf{mod}^L_{2^k}(x)$ function satisfying ${}_1x = 2^k \times \mathsf{div}^L_{2^k}(x) \cdot \mathsf{mod}^L_{2^k}(x)$.

$$\mathsf{div}^L_1(x) = {}_1x \qquad \mathsf{div}^L_{2k}(x) = \mathsf{div}^L_k(x\blacktriangleleft)$$
$$\mathsf{mod}^L_{2^k}(x) = \big(2^k \times \mathsf{div}^L_{2^k}(x)\big) \rhd {}_1x$$

(Interestingly, there does not seem to be a way to define a div^L_k function for arbitrary k without using TRN.) Following this, we define functions to perform simple bit manipulations on strings (extract single bits or substrings, pad to a certain length).

- "Left bit": $\mathsf{lb}(x, y) = y \, ?^{ZL} \, (\varepsilon, {}^{\backprime}(\gtrdot y \rhd x))$ returns bit number $|y|$ of x from the left; "right bit": $\mathsf{rb}(x, y) = y ?^{ZL} (\varepsilon, (x \lhd y \lessdot)')$ returns bit number $|y|$ of x from the right (both are equal to ε if $y = \varepsilon$ or $|y| > |x|$). For convenience, we also define $\mathsf{lb}^B(x, y) = \mathsf{lb}(x, y) \; ? \; (0, 0, 1)$ and $\mathsf{rb}^B(x, y) = \mathsf{rb}(x, y) \; ? \; (0, 0, 1)$ which return 0 or 1 for all arguments.

- "Left cut": $\mathsf{lc}(x, y) = x \lhd (y \rhd x)$ returns the $|y|$ leftmost bits of x; "right cut": $\mathsf{rc}(x, y) = (x \lhd y) \rhd x$ returns the $|y|$ rightmost bits of x (both return ε if $y = \varepsilon$ and x if $|y| \geq |x|$).

- "Left pad": $\mathsf{lp}_j(x, y) = {}_j(y \lhd x) \cdot x$ returns x padded on the left with j's so that $|\mathsf{lp}_j(x, y)| \geq |y|$; "right pad": $\mathsf{rp}_j(x, y) = x \cdot {}_j(x \rhd y)$ returns x padded on the right with j's so that $|\mathsf{rp}_j(x, y)| \geq |y|$ (both return x if $|y| \leq |x|$).

- "Left adjust": $\mathsf{la}_j(x, y) = {}_j(y \lhd x) \cdot ((x \lhd y) \rhd x)$ returns x either chopped or padded on the left so that $|\mathsf{la}_j(x, y)| = |y|$; "right adjust": $\mathsf{ra}_j(x, y) = (x \lhd (y \rhd x)) \cdot {}_j(x \rhd y)$ returns x either chopped or padded on the right so that $|\mathsf{ra}_j(x, y)| = |y|$.

Finally, we have all the functions we need to define a tuple function ($\langle \vec{x}_k \rangle_k$) and corresponding projection functions ($\pi^k_\ell(x)$). To form tuples, we simply concatenate the arguments together after padding them on the left so that they all have the same length. The projection functions are then defined easily using ◀ and ▶. One small complication arises because we can only divide the length of a string by a power of 2, so we need to form tuples that always have a power of 2 elements even when there are fewer of them that are actually input values. The definitions follow and are inspired by similar definitions in Bloch's paper [6]. (The tuple function is defined in terms of an auxiliary function tuple that has an extra parameter specifying the length to which each value should be padded.)

$$\begin{aligned}
\mathsf{tuple}_1(x, z) &= \mathsf{lp}_0(x, z)\\
\mathsf{tuple}_{2k}(\vec{x}_k, \vec{y}_k, z) &= \mathsf{tuple}_k(\vec{x}_k, z) \cdot \mathsf{tuple}_k(\vec{y}_k, z)\\
\mathsf{tuple}_{2k+1}(\vec{x}_k, \vec{y}_{k+1}, z) &= \mathsf{tuple}_{k+1}(\varepsilon, \vec{x}_k, z) \cdot \mathsf{tuple}_{k+1}(\vec{y}_{k+1}, z)\\
\langle \vec{x}_k \rangle_k &= \mathsf{tuple}_k\big(\vec{x}_k, \mathsf{max}^L_k(\vec{x}_k)\big)\\
\pi^1_1(y) &= y\\
\pi^{2k}_\ell(y) &= \begin{cases} \pi^k_\ell(y◀) & \text{if } \ell \le k\\ \pi^k_{\ell-k}(▶y) & \text{if } \ell > k \end{cases}\\
\pi^{2k+1}_\ell(y) &= \begin{cases} \pi^{k+1}_{\ell+1}(y◀) & \text{if } \ell \le k\\ \pi^{k+1}_{\ell-k}(▶y) & \text{if } \ell > k \end{cases}
\end{aligned}$$

Note that these functions satisfy the following relations:

$$\begin{aligned}
\pi^k_\ell(\langle x_1, \ldots, x_k \rangle_k) &= \mathsf{lp}_0(x_\ell, \mathsf{max}^L_k(\vec{x}_k)),\\
\left\langle \pi^{2^k}_1(y), \ldots, \pi^{2^k}_{2^k}(y) \right\rangle_{2^k} &= y
\end{aligned}$$

(unfortunately, $\langle \pi^k_1(y), \ldots, \pi^k_k(y) \rangle_k \neq y$ for arbitrary k because of the way the tuple function is defined).

2.2.3 Generalizations of CRN

We now introduce a generalization of CRN where the recursion is defined on several variables at once. (We assume that $x_1, \ldots, x_m$ all have the same length, or are appropriately padded on the left with 0's to make them all the same length.)

DEFINITION 2.2.1 (CRN$_m$) We say that f is defined from h by *CRN$_m$ on* $x_1, \ldots, x_m$ if

$h(\vec{x}_m, \vec{y}) \in \{0,1\}$ for all $\vec{x}_m, \vec{y}$ and

$$\begin{aligned} f(\vec{\varepsilon}_m, \vec{y}) &= \varepsilon, \\ f(x_1 i_1, \ldots, x_m i_m, \vec{y}) &= f(x_1, \ldots, x_m, \vec{y}) \cdot h(x_1 i_1, \ldots, x_m i_m, \vec{y}) \quad \text{for } i_1 = 0,1; \ldots; i_m = 0,1. \end{aligned}$$

(We can also define *left* CRN$_m$ similarly to left CRN.) If f is defined from h by CRN$_m$ on $x_1, \ldots, x_m$, then we can define f using CRN as follows: We will define an auxiliary function $\mathsf{aux\_}f$ by CRN on a parameter z; this function will mimic the recursion on $x_1, \ldots, x_m$ by using lc to extract the correct substrings of $x_1, \ldots, x_m$ based on the length of z. Then, f is easily defined from $\mathsf{aux\_}f$ by COMP.

$$\begin{aligned} \mathsf{aux\_}f(zi, \vec{x}_m, \vec{y}) &= \mathsf{aux\_}f(z, \vec{x}_m, \vec{y}) \cdot h\big(\mathsf{lc}(x_1, zi), \ldots, \mathsf{lc}(x_m, zi), \vec{y}\big) \\ f(\vec{x}_m, \vec{y}) &= \mathsf{aux\_}f\big(\mathsf{max}^L_m(\vec{x}_m), \mathsf{lp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)), \ldots, \mathsf{lp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m)), \vec{y}\big) \end{aligned}$$

When a function $f(x, \vec{y})$ is defined by CRN on x from h, every bit in the output corresponds to one bit from x. Now, we will show how to define a function where every bit of x corresponds to two bits in the output, and then generalize this to arbitrary values (where every group of 2^k bits in the input corresponds to a group of 2^n bits in the output, which we will call "2^k-to-2^n-CRN", or "$(2^k, 2^n)$-CRN").

Following the notation mentioned above, we say that a function f is defined from h by *1-to-2-CRN ($(1,2)$-CRN) on x* if $|h(x, \vec{y})| = 2$ for all $x, \vec{y}$ and

$$\begin{aligned} f(\varepsilon, \vec{y}) &= \varepsilon, \\ f(xi, \vec{y}) &= f(x, \vec{y}) \cdot h(xi, \vec{y}) \quad \text{for } i = 0,1. \end{aligned}$$

(We can also define *left* $(1,2)$-CRN.) If $f(x, \vec{y})$ is defined from h by $(1,2)$-CRN on x, we can define f using CRN as follows: We will first define an auxiliary function $\mathsf{aux\_}f(z, x, \vec{y})$ by CRN on z, to return the $|z|$ leftmost bits of $f(x, \vec{y})$ and then define f from $\mathsf{aux\_}f$ by COMP. Intuitively, $\mathsf{aux\_}f(z, x, \vec{y})$ uses $\mathsf{div}^L_2(z)$ to determine which bits of x to give as input to h and $\mathsf{mod}^L_2(z)$ to determine which bit of h to output next.

$$\begin{aligned} \mathsf{aux\_}f(zi, x, \vec{y}) &= \mathsf{aux\_}f(z, x, \vec{y}) \cdot \mathsf{lb}^B\big(h\big(\mathsf{lc}(x, \mathsf{div}^L_2(z) \cdot i), \vec{y}\big), \mathsf{mod}^L_2(z) \cdot i\big) \\ f(x, \vec{y}) &= \mathsf{aux\_}f(x \cdot x, x, \vec{y}) \end{aligned}$$

Now, we can introduce the generalization mentioned above.

DEFINITION 2.2.2 ($(2^k, 2^n)$-CRN) We say that f is defined from g and h by *2^k-to-2^n-CRN ($(2^k, 2^n)$-CRN) on x* if $|h(x, \vec{y})| = 2^n$ for all $x, \vec{y}$ and

$$\begin{aligned} f(x, \vec{y}) &= g(x, \vec{y}) && \text{if } |x| < 2^k, \\ f(x \cdot z, \vec{y}) &= f(x, \vec{y}) \cdot h(x \cdot z, \vec{y}) && \text{for } z \in \{0,1\}^{2^k}. \end{aligned}$$

(As before, we can also define *left* $(2^k, 2^n)$-CRN.) If f is defined from g and h by $(2^k, 2^n)$-CRN on x, then we can define f using CRN as follows. (The intuition is similar to that for $(1,2)$-CRN given above.)

$$\mathsf{aux\_}f(zi, x, \vec{y}) = \mathsf{aux\_}f(z, x, \vec{y}) \cdot \mathsf{lb}^B\Big(h\big(\mathsf{lc}\big(x, \mathsf{mod}^L_{2^k}(x) \cdot 2^k \times (\mathsf{div}^L_{2^n}(z) \cdot i)\big), \vec{y}\big), \mathsf{mod}^L_{2^n}(z) \cdot i\Big)$$

$$f(x, \vec{y}) = g\big(\mathsf{lc}(x, \mathsf{mod}^L_{2^k}(x)), \vec{y}\big) \cdot \mathsf{aux\_}f\big(2^n \times \mathsf{div}^L_{2^k}(x), x, \vec{y}\big)$$

By combining the two generalizations above, we can show that any function defined by "$(2^k, 2^n)$-CRN$_m$" can be defined using CRN and COMP alone, which gives us a relatively powerful way to define many more useful functions.

2.2.4 Boolean functions

The next functions we will introduce are the Boolean operators, *i.e.*, the standard connectives together with some useful functions for comparing bits (these function symbols will be distinguished by putting a superscript "^{B}" next to them). First, we will define a "Boolean test" function, which tests for the *truth-value* of its argument (where a string's truth-value is determined by its rightmost bit by convention, with $1 = \texttt{true}$ and $0 = \texttt{false}$ — ε is treated the same way as 0): $x \,?^B (y, z) = x \,? (z, z, y)$ is equal to y if x is "true"; z if x is "false" (according to the convention above). Then, $\approx^B x = x \,?^B (1, 0)$ returns the truth-value of x and we can define the boolean connectives in the usual way.

$$\begin{array}{lll}
\neg^B x = x \,?^B (0, 1) & x \wedge^B y = x \,?^B (\approx^B y, 0) & x \vee^B y = x \,?^B (1, \approx^B y) \\
x \to^B y = x \,?^B (\approx^B y, 1) & x \leftrightarrow^B y = x \,?^B (\approx^B y, \neg^B y) & x \oplus^B y = x \,?^B (\neg^B y, \approx^B y) \\
x \geq^B y = y \to^B x & x \leq^B y = x \to^B y & x =^B y = x \leftrightarrow^B y \\
x <^B y = \neg^B (x \geq^B y) & x >^B y = \neg^B (x \leq^B y) & x \neq^B y = \neg^B (x =^B y)
\end{array}$$

Using CRN$_m$, we can now easily define the following useful functions that perform bitwise operations on their arguments.

$$\begin{aligned}
\mathsf{not}^B(xi) &= \mathsf{not}^B(x) \cdot \neg^B i \\
\mathsf{and}^B_k(x_1 i_1, \ldots, x_k i_k) &= \mathsf{and}^B_k(x_1, \ldots, x_k) \cdot (i_1 \wedge^B \cdots \wedge^B i_k) \\
\mathsf{or}^B_k(x_1 i_1, \ldots, x_k i_k) &= \mathsf{or}^B_k(x_1, \ldots, x_k) \cdot (i_1 \vee^B \cdots \vee^B i_k) \\
\mathsf{xor}^B_k(x_1 i_1, \ldots, x_k i_k) &= \mathsf{xor}^B_k(x_1, \ldots, x_k) \cdot (i_1 \oplus^B \cdots \oplus^B i_k) \\
\mathsf{iff}^B_k(x_1 i_1, \ldots, x_k i_k) &= \mathsf{iff}^B_k(x_1, \ldots, x_k) \cdot \big((i_1 \leftrightarrow^B i_2) \wedge^B \cdots \wedge^B (i_{k-1} \leftrightarrow^B i_k)\big)
\end{aligned}$$

And following Buss [8], we can define functions that implement *carry-save addition*: CScar to compute the carry bits and CSadd to compute the addition bits. Note that these functions are defined so that $\mathsf{CScar}(x, y, z, w) + \mathsf{CSadd}(x, y, z, w) = x + y + z + w$ (a fact that will be proved rigorously in Chapter 3.)

$$\begin{aligned}
\mathsf{CScar}_3(i_1x_1, i_2x_2, i_3x_3) &= \big((i_1 \wedge^B i_2) \vee^B (i_2 \wedge^B i_3) \vee^B (i_3 \wedge^B i_1)\big) \cdot \mathsf{CScar}_3(x_1, x_2, x_3)\\
\mathsf{CSadd}_3(x_1, x_2, x_3) &= \mathsf{xor}_3^B(0x_1, 0x_2, 0x_3) = 0 \cdot \mathsf{xor}_3^B(x_1, x_2, x_3)\\
\mathsf{CScar}(x_1, x_2, x_3, x_4) &= \mathsf{CScar}_3\big(\mathsf{CScar}_3(x_1, x_2, x_3) \cdot 0, \mathsf{CSadd}_3(x_1, x_2, x_3), 0x_4\big) \cdot 0\\
\mathsf{CSadd}(x_1, x_2, x_3, x_4) &= \mathsf{CSadd}_3\big(\mathsf{CScar}_3(x_1, x_2, x_3) \cdot 0, \mathsf{CSadd}_3(x_1, x_2, x_3), 0x_4\big)
\end{aligned}$$

2.3 Functions in L_1

In this section, we define many functions in L_1 and show that some useful generalizations of TRN can be simulated in L_1. Again, we are motivated by two goals: to define the machinery necessary to prove that L_1 contains all of FNC^1, and to show that many useful functions have simple definitions in our algebra.

2.3.1 Basic functions

Recall that the operation of TRN is restricted in L_1 so that we cannot define a function by TRN from functions that are themselves defined by TRN. Hence, it will be useful to be able to define more than one function simultaneously by TRN.

DEFINITION 2.3.1 (TRN$_k$) The functions f_i $(1 \le i \le k)$ are defined from functions g_i, h_i, h_ℓ, and h_r by *TRN$_k$ on x* if $|f_1(x, z, \vec{y})| = \cdots = |f_k(x, z, \vec{y})|$ for all $x, z, \vec{y}$, and for every $1 \le i \le k$,

$$f_i(x, z, \vec{y}) = \begin{cases} g_i(x, z, \vec{y}) & \text{if } x = \varepsilon, 0, 1,\\ h_i\big(x, z, f_1(x\blacktriangleleft, h_\ell(z), \vec{y}), f_1(\blacktriangleright x, h_r(z), \vec{y}), \ldots, & \\ \qquad f_k(x\blacktriangleleft, h_\ell(z), \vec{y}), f_k(\blacktriangleright x, h_r(z), \vec{y}), \vec{y}\big) & \text{otherwise.} \end{cases}$$

If $\vec{f_k}$ are defined from $\vec{g}_k$, $\vec{h}_k$, h_ℓ, and h_r by TRN$_k$, we can define the k-tuple $F(x, z, \vec{y}) = \langle \vec{f_k}(x, z, \vec{y})\rangle_k$ by TRN as follows.

$$F(x, z, \vec{y}) = \begin{cases} \langle \vec{g}_k(x, z, \vec{y})\rangle_k & \text{if } x = \varepsilon, 0, 1,\\ \Big\langle \vec{h}_k\big(x, z, \pi_1^k(F(x\blacktriangleleft, h_\ell(z), \vec{y})), \pi_1^k(F(\blacktriangleright x, h_r(z), \vec{y})), \ldots, & \\ \qquad \pi_k^k(F(x\blacktriangleleft, h_\ell(z), \vec{y})), \pi_k^k(F(\blacktriangleright x, h_r(z), \vec{y})), \vec{y}\big) \Big\rangle_k & \text{otherwise.} \end{cases}$$

Then, a simple composition gives $f_i(x, z, \vec{y}) = \pi_i^k(F(x, z, \vec{y}))$ for $1 \le i \le k$.

Now, we can define some functions by TRN (actually, by simple TRN). The first two perform Boolean operations on all the bits of their input; $|x|$ returns the length of x, expressed as a binary number; $x \# y$ returns $|x|$ copies of y concatenated together (so that $|x \# y| = |x| \times |y|$);

$\mathsf{div}^L_k(x)$ returns a string whose length is equal to $\lfloor |x|/k \rfloor$ (where we use the notation "$\mathsf{mod}^L_k(x)$" as a shorthand for the term $(k \times \mathsf{div}^L_k(x)) \rhd {}_1x)$; and the last two functions are defined by TRN$_2$ and will be used to count the number of 1's in the string x, using the carry-save technique of Buss [8] (this will be done below).

$$\mathsf{AND}(x) = \begin{cases} x & \text{if } x = \varepsilon, 0, 1 \\ \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x) & \text{otherwise} \end{cases}$$

$$\mathsf{OR}(x) = \begin{cases} x & \text{if } x = \varepsilon, 0, 1 \\ \mathsf{OR}(x\blacktriangleleft) \vee^B \mathsf{OR}(\blacktriangleright x) & \text{otherwise} \end{cases}$$

$$|x| = \begin{cases} {}_1x & \text{if } x = \varepsilon, 0, 1 \\ x\blacktriangleleft \rhd \blacktriangleright x \;?^{ZL} \big(|x\blacktriangleleft| \cdot 0, |x\blacktriangleleft| \cdot 1\big) & \text{otherwise} \end{cases}$$

$$x \# y = \begin{cases} x \;?^{ZL} (\varepsilon, y) & \text{if } x = \varepsilon, 0, 1 \\ (x\blacktriangleleft \# y) \cdot (\blacktriangleright x \# y) & \text{otherwise} \end{cases}$$

$$\mathsf{div}^L_k(x) = \begin{cases} x \rhd \widehat{k} \;?^{ZL} ({}_1x, \varepsilon) & \text{if } x = \varepsilon, 0, 1 \\ \mathsf{div}^L_k(x\blacktriangleleft) \cdot \mathsf{div}^L_k(\blacktriangleright x) \cdot \big((\mathsf{mod}^L_k(x\blacktriangleleft) \cdot \mathsf{mod}^L_k(\blacktriangleright x)) \rhd \widehat{k} \;?^{ZL} (1, \varepsilon)\big) & \text{otherwise} \end{cases}$$

$$\mathsf{CAR}(x) = \begin{cases} {}_0x & \text{if } x = \varepsilon, 0, 1 \\ \mathsf{CScar}(\mathsf{CAR}(x\blacktriangleleft), \mathsf{CAR}(\blacktriangleright x), \mathsf{ADD}(x\blacktriangleleft), \mathsf{ADD}(\blacktriangleright x)) & \text{otherwise} \end{cases}$$

$$\mathsf{ADD}(x) = \begin{cases} x & \text{if } x = \varepsilon, 0, 1 \\ \mathsf{CSadd}(\mathsf{CAR}(x\blacktriangleleft), \mathsf{CAR}(\blacktriangleright x), \mathsf{ADD}(x\blacktriangleleft), \mathsf{ADD}(\blacktriangleright x)) & \text{otherwise} \end{cases}$$

Note that $|\mathsf{CAR}(x)| = |\mathsf{ADD}(x)|$ (easy to show inductively) so the definition by TRN$_2$ is correct. Also note that with div^L_k and mod^L_k, we can now define "(k, ℓ)-CRN" similarly to $(2^k, 2^\ell)$-CRN, but for blocks of bits of arbitrary fixed lengths. Interestingly, it does not seem possible to define a more general div^L function that would take two parameters, and thus to define a general form of CRN where the lengths of the input and output blocks of bits are specified by extra parameters (we discuss this issue further in Chapter 3).

2.3.2 Numerical functions

Unfortunately, the fact that functions in L_1 are length-determined makes it harder to define "numerical" functions, *i.e.*, functions that treat their inputs as binary notation for numbers (ignoring leading 0's). For example, the definition of $|x|$ given above is quite simple whereas the definition of $|x|^N$ given below relies on some more complex functions.

Now, we will define a number of "numerical" functions (distinguished by putting a superscript "^{N}" next to them). We start with an equality operator for numbers, and also one for strings.

$$x =^N y = \mathsf{AND}(1 \cdot \mathsf{iff}_2^B(x, y)) \qquad x =^S y = \neg^B(x =^L y) \wedge^B (x =^N y)$$

Note that the extra "1" is necessary in the definition of $=^N$ for $\varepsilon =^N \varepsilon$ to be true, and the value of $x =^N y$ is independent of the lengths of x and y, *i.e.*, the function really does behave as though its string inputs were binary representations of numbers. Also note that "$\neg^B$" is necessary in the definition of $=^S$ because of our convention that $=^L$ returns ε for "true" and 1 for "false".

Next, we define a successor and a predecessor function, both by left CRN. The successor function is defined in terms of an auxiliary function that simply replaces each bit by its negation until it encounters a 0, which it replaces by 1, and then outputs each bit unchanged (*e.g.*, 11010 becomes 11011, 1011 becomes 1100, 111 becomes 000). The successor function first adds a 0 to the front (left) of its argument before calling the auxiliary function, in case the string consists of all 1's (*e.g.*, 111 correctly becomes 1000 and not just 000). The predecessor function performs a similar computation, except replacing bits by their negation until it encounters a 1, and first checking that the string does not consist of all 0's before calling the auxiliary function.

$$\mathsf{aux\_succ}^N(ix) = \big(\mathsf{AND}(1x) \;?^B\; (\neg^B i, i)\big) \cdot \mathsf{aux\_succ}^N(x) \qquad \mathsf{succ}^N(x) = \mathsf{aux\_succ}^N(0x)$$
$$\mathsf{aux\_pred}^N(ix) = \big(\mathsf{OR}(x) \;?^B\; (i, \neg^B i)\big) \cdot \mathsf{aux\_pred}^N(x) \qquad \mathsf{pred}^N(x) = \mathsf{OR}(x) \;?^B\; (\mathsf{aux\_pred}^N(x), x)$$

Note that one unfortunate side-effect of the fact that the functions are length-determined is that the successor function always appends a bit to the left of its argument. So starting from ε and applying succ^N repeatedly, we get a series of strings that represent $0, 1, 2, \ldots$ in binary, but whose lengths are also $0, 1, 2, \ldots$ Next, we define the numerical predicate $<^N$, which together with $=^N$ allows us to define all other relational operators on numbers using the Boolean connectives. Note that to define less^N, we use Clote's "programming trick" [12] of making a sweep through the bits of the strings x and y, appending a 1 when some condition is met so that the final composition with OR yields 1 iff the condition was met at some position. Using AND, we could similarly define functions that test for some condition on every bit of their inputs. Also recall that functions defined by CRN$_2$ (such as less^N below) first pad their arguments on the left with 0's so they have the same length.

$$\mathsf{less}^N(xi, yj) = \mathsf{less}^N(x, y) \cdot \big((i <^B j) \wedge^B (x =^N y)\big) \qquad x <^N y = \mathsf{OR}(\mathsf{less}^N(x, y))$$

The next function we want to define is $\mathsf{bit}^N(x, z)$, which returns bit number z of x, starting at 1 and counting from the right, where z is interpreted as a binary number. The easiest way to do this is by defining a function $\mathsf{pow}^N(z, x)$ that returns a string of length $|x|$ consisting entirely of 0's except at bit position z (from the right), if $1 \leq^N z \leq^N |x|$. Then, we define a function $\mathsf{maskbit}^N(x, y)$ that treats y as a mask to determine which bit of x to return.

$$\mathsf{pow}^N(z, ix) = \big(|ix| =^N z\big) \cdot \mathsf{pow}^N(z, x)$$
$$\mathsf{maskbit}^N(x, y) = \mathsf{OR}(\mathsf{and}_2^B(x, y))$$
$$\mathsf{bit}^N(x, z) = \mathsf{maskbit}^N(x, \mathsf{pow}^N(z, x))$$

Next, we want to define addition. This will require only a few more definitions. First, in order to simulate a function that strips leading ones (or zeros) from x, we can define functions $\mathsf{first}_j(x)$ that return a mask which is 1 on the leftmost bit of x equal to j and 0 elsewhere. We can also define a function that returns a mask which is 1 on every significant bit of x (*i.e.*, every bit to the right of the first "1" in x) and 0 elsewhere.

$$\mathsf{first}_1(xi) = \mathsf{first}_1(x) \cdot \big(\mathsf{OR}(x)\ ?^B\ (0, i)\big) \qquad \mathsf{first}_0(xi) = \mathsf{first}_0(x) \cdot \big(\mathsf{AND}(1x)\ ?^B\ (\neg^B i, 0)\big)$$
$$\mathsf{mask}^N(xi) = \mathsf{mask}^N(x) \cdot \big(\mathsf{OR}(x)\ ?^B\ (1, i)\big)$$

Then, we can define a function which computes the carry bits and an addition function, as follows.

$$\mathsf{carry}^N(ix, jy) = \mathsf{maskbit}^N\big(\mathsf{and}_2^B(ix, jy), \mathsf{first}_0(\mathsf{xor}_2^B(ix, jy))\big) \cdot \mathsf{carry}^N(x, y)$$
$$x +^N y = \mathsf{xor}_3^B(\mathsf{carry}^N(x, y) \cdot 0, x, y)$$

Finally, using the addition function, we can define a function that counts the number of ones in a string: $\mathsf{sum}(x) = \mathsf{CAR}(x) +^N \mathsf{ADD}(x)$ and using this function, define the "numerical" length of x: $|x|^N = \mathsf{sum}(\mathsf{mask}^N(x))$. (Note that we could also have defined $|x| = \mathsf{sum}({}_1x)$ instead of directly using TRN.)

2.4 L_1 and *FALOGTIME*

In this section, we prove the following claim.

CLAIM 2.4.1 *L_1 = FALOGTIME (= uniform FNC^1).*

To be precise, we say that a k-ary function f belongs to *FALOGTIME* if there exists an integer polynomial p_f such that $|f(\vec{x}_k)| \leq p_f(|x_1|, \ldots, |x_k|)$ for every $\vec{x}_k$, and if the language $\{\langle \vec{x}_k, i, b\rangle :$ the i-th bit of $f(\vec{x}_k)$ is equal to $b\}$ is recognizable by an ATM running in time $\mathcal{O}\big(\max\{\log|x_1|, \ldots, \log|x_k|\}\big)$.

2.4.1 *FALOGTIME* is contained in L_1

To prove that *FALOGTIME* $\subseteq L_1$, we show how to simulate the computation of an ATM using functions in L_1. Then, if $f \in$ *FALOGTIME*, there exists a term $t_f \in L_1$ such that $|t_f(\vec{x}_k)| = p_f(|\vec{x}_k|)$, so we can use CRN on $t_f(\vec{x}_k)$ to compute each bit of f by simulating the ATM on the appropriate input. (Technically speaking, this works only if $|f(\vec{x}_k)| = |t_f(\vec{x}_k)|$ for all $\vec{x}_k$, but if that is not the case, we can simply use Bloch's idea of "length masks" to compute a mask $b_f(\vec{x}_k)$ of length $t_f(\vec{x}_k)$ that has a 1 in every bit position where $f(\vec{x}_k)$ is defined and a 0 elsewhere.)

Now, without loss of generality, let the ATM have the following properties.

1. There is a function $t \in L_1$ such that the ATM runs for no more than $||t(\vec{x})|| = \mathcal{O}(\log |t(\vec{x})|)$ steps on inputs $\vec{x}$ (always possible when the ATM runs in logarithmic time since t can use $\#$ and $\cdot$ to output a string whose length is an arbitrary polynomial in the lengths of the inputs). Also, universal states of the ATM are given *even* numbers and existential states, *odd* numbers. Moreover, we assume that the function t is defined so that $|t(\vec{x})|$ is always a power of 2 and $2^{2|t(\vec{x})|}$ is greater than the number of states of the ATM for any inputs $\vec{x}$ (in other words, a string of length $2|t(\vec{x})|$ is long enough to encode the state of the ATM).

2. The ATM has n read-only input tapes represented by strings $x_1, \ldots, x_n$ and k worktapes represented by pairs of strings $y_1^\ell, y_1^r, \ldots, y_k^\ell, y_k^r$, each of length exactly $2|t(\vec{x})|$, where y_i^ℓ represents the content of tape number i to the left of the tape head and y_i^r represents the content to the right, with the head scanning the rightmost symbol of y_i^ℓ. Each of the three possible worktape symbols (1, 0, or blank) is encoded using two bits (11 for 1, 10 for 0, and 00 for blank). Initially, the worktapes are blank.

3. The computation tree of the ATM is a complete binary tree (each non-leaf node has exactly two successor configurations, a left successor and a right successor, and every leaf occurs at the same level).

4. Access to the input occurs only at the leaves of the computation tree and is of the form "accept iff symbol number y_i^ℓ (interpreted as a binary number) on input tape number j is equal to b", where i, j, and b are encoded in the current state of the ATM.

Then, if we let $\text{CON} = \langle s, y_1^\ell, y_1^r, \ldots, y_k^\ell, y_k^r \rangle_{2k+1}$ represent a configuration of the ATM when in state s, we can define $l\text{CON}$ and $r\text{CON}$, the left and right successor configurations of CON, as follows:

$$\ell\text{CON} = \left\langle \mathsf{lstate}(\text{CON}), \mathsf{ltape}_1^\ell(\text{CON}), \mathsf{ltape}_1^r(\text{CON}), \ldots, \mathsf{ltape}_k^\ell(\text{CON}), \mathsf{ltape}_k^r(\text{CON}) \right\rangle_{2k+1},$$

$$r\text{CON} = \left\langle \mathsf{rstate}(\text{CON}), \mathsf{rtape}_1^\ell(\text{CON}), \mathsf{rtape}_1^r(\text{CON}), \ldots, \mathsf{rtape}_k^\ell(\text{CON}), \mathsf{rtape}_k^r(\text{CON}) \right\rangle_{2k+1},$$

where each of $\mathsf{lstate}, \mathsf{ltape}_i^\ell, \mathsf{ltape}_i^r, \mathsf{rstate}, \mathsf{rtape}_i^\ell, \mathsf{rtape}_i^r$ is easily seen to be in L_0, involving only simple string manipulations and finite table lookup on the state s (for example, $\mathsf{ltape}_i^\ell(\textsc{con}) = d_i \;?^B \big(00 \rhd ((y_i^\ell \lhd 00 \cdot b_i) \cdot \mathsf{lc}(y_i^r, 00)), (00 \cdot y_i^\ell) \lhd 00\big)$ computes the contents of tape i to the left of the head in the left successor of $\textsc{con}$, where d_i (the direction of movement for head i) and b_i (the tape symbol to write on tape i) are obtained from the state and tape contents of $\textsc{con}$ using the conditional function). Moreover, if we let select be defined by $(2,1)$-CRN to output every second bit of its input string, the function $\mathsf{input}(\textsc{con}, \vec{x}) = \mathsf{bit}^N(x_j, \mathsf{select}(y_i^\ell)) \leftrightarrow^B b$ (where i, j, and b are extracted from the state s) is equal to the accept state of the given input configuration and is in L_1. Finally, we let $\textsc{con}_0 = \langle s_0, t(\vec{x}) \# 00, \ldots, t(\vec{x}) \# 00\rangle_{2k+1}$ denote the initial configuration, where s_0 is the initial state of the ATM.

Now, we can easily use TRN to define a function eval that evaluates the computation tree of the ATM, so that the result of the entire computation is given by $\mathsf{eval}(t(\vec{x}), \textsc{con}_0, \vec{x})$:

$$\mathsf{eval}(z, \textsc{con}, \vec{x}) = \begin{cases} \mathsf{input}(\textsc{con}, \vec{x}) & \text{if } z = \varepsilon, 0, 1, \\ \pi_1^{2k+1}(\textsc{con})' \;?^B \big(\mathsf{eval}(z\blacktriangleleft, \ell\textsc{con}, \vec{x}) \vee^B \mathsf{eval}(\blacktriangleright z, r\textsc{con}, \vec{x}), & \\ \qquad\qquad \mathsf{eval}(z\blacktriangleleft, \ell\textsc{con}, \vec{x}) \wedge^B \mathsf{eval}(\blacktriangleright z, r\textsc{con}, \vec{x})\big) & \text{otherwise.} \end{cases}$$

Note that in the recursive call, $\pi_1^{2k+1}(\textsc{con})$ simply extracts the current state from the given configuration, and the rightmost bit of the state number is used to determine whether the configuration is universal or existential. Also note that this half of the proof is considerably simpler than the corresponding proofs in Bloch [6] and Clote [13, 14]. This seems to be because our scheme of TRN encapsulates the sort of computation carried out by ATM's more directly than the schemes considered by Bloch and Clote, especially by its use of the parameter replacement functions h_ℓ and h_r.

2.4.2 L_1 is contained in *FALOGTIME*

To prove that $L_1 \subseteq \mathit{FALOGTIME}$, we argue that every function in L_0 can be computed by a family of circuits in uniform FNC^0, and that every function in L_1 can be computed by a family of circuits in uniform FNC^1, where we use Bloch's notion of *mapping-uniformity*, defined in [6], which generalizes U_{E^*}-uniformity to make sense for circuits of constant depth. As will be seen, the facts that functions in L_0 have constant depth circuits and that functions in L_1 have logarithmic depth circuits are quite simple to prove; the technical difficulties arise mainly from uniformity considerations.

First, we give bounds on the rate of growth of functions in L_0 and L_1.

LEMMA 2.4.2 *For every n-ary function $f \in L_0$, there exist constants $a_0^f, a_1^f, \ldots, a_n^f \in \mathbb{N}$ such that $|f(x_1, \ldots, x_n)| \leq a_0^f + a_1^f|x_1| + \cdots + a_n^f|x_n|$ for all strings $x_1, \ldots, x_n$.*

PROOF The result is proved by induction on the definition of f.

- If $f = \varepsilon$, then $a_0^f = 0$ since $|\varepsilon| = 0$.
- If $f = 0$ or $f = 1$, then $a_0^f = 1$ since $|0| = |1| = 1$.
- If $f(x) = \blacktriangleright x$, then $a_0^f = 0, a_1^f = 1$ since $|\blacktriangleright x| = \lceil |x|/2 \rceil \leq |x|$.
- If $f(x,y) = x \lhd y$, then $a_0^f = 0, a_1^f = 1, a_2^f = 0$ since $|x \lhd y| = |x| \dot{-} |y| \leq |x|$.
- If $f(x,y) = x \cdot y$, then $a_0^f = 0, a_1^f = 1, a_2^f = 1$ since $|x \cdot y| = |x| + |y|$.
- If $f(x,y,z,w) = x \mathbin{?} (y,z,w)$, then $a_0^f = 0, a_1^f = 0, a_2^f = 1, a_3^f = 1, a_4^f = 1$ since $|x \mathbin{?} (y,z,w)| \leq \max\{|y|,|z|,|w|\} \leq |y| + |z| + |w|$.
- If $f(x_1,\dots,x_n) = \mathcal{I}_k^n(x_1,\dots,x_n)$, then $a_0^f = 0, \dots, a_{k-1}^f = 0, a_k^f = 1, a_{k+1}^f = 0, \dots, a_n^f = 0$ since $|\mathcal{I}_k^n(x_1,\dots,x_n)| = |x_k|$.
- If f is defined by CRN from h, then $|f(x,\vec{y})| = |x|$ so $a_0^f = 0, a_1^f = 1, a_2^f = 0, \dots, a_{n+1}^f = 0$.
- If f is defined by COMP from g and $h_1,\dots,h_k$, then

$$\begin{aligned}
|f(\vec{x})| &= |g(h_1(\vec{x}),\dots,h_k(\vec{x}))| \\
&\leq a_0^g + a_1^g|h_1(\vec{x})| + \cdots + a_k^g|h_k(\vec{x})| \\
&\leq a_0^g + \sum_{1\leq i\leq k} a_i^g\Big(a_0^{h_i} + \sum_{1\leq j\leq n} a_j^{h_i}|x_j|\Big) \\
&\leq \Big(a_0^g + \sum_{1\leq i\leq k} a_i^g a_0^{h_i}\Big) + \sum_{1\leq j\leq n}\Big(\sum_{1\leq i\leq k} a_i^g a_j^{h_i}\Big)|x_j|
\end{aligned}$$

so $a_0^f = a_0^g + a_1^g a_0^{h_1} + \cdots + a_k^g a_0^{h_k}$, and $a_i^f = a_1^g a_i^{h_1} + \cdots + a_k^g a_i^{h_k}$ for $1 \leq i \leq n$. □

LEMMA 2.4.3 *For every n-ary function $f \in L_1$, there exists a polynomial $p_f \in \mathbb{N}[x_1,\dots,x_n]$ such that $|f(x_1,\dots,x_n)| \leq p_f(|x_1|,\dots,|x_n|)$ for all strings $x_1,\dots,x_n$.*

PROOF The result is proved by induction on the definition of f, where we use the notation $|\vec{x}_n|$ to stand for the list $|x_1|,\dots,|x_n|$.

- If $f \in L_0$, then by the preceding lemma, $p_f(|\vec{x}_n|) = a_0^f + a_1^f|x_1| + \cdots + a_n^f|x_n|$.
- If f is defined by CRN from h, then as in the preceding lemma, $p_f(|x|,|\vec{y}|) = |x|$.
- If f is defined by COMP from g and $h_1,\dots,h_k$, then

$$\begin{aligned}
|f(\vec{x}_n)| &= |g(h_1(\vec{x}_n),\dots,h_k(\vec{x}_n))| \\
&\leq p_g\big(|h_1(\vec{x}_n)|,\dots,|h_k(\vec{x}_n)|\big) \\
&\leq p_g\big(p_{h_1}(\vec{x}_n),\dots,p_{h_k}(\vec{x}_n)\big)
\end{aligned}$$

(since polynomials in $\mathbb{N}[\vec{x}]$ are non-decreasing), so $p_f(|\vec{x}_n|) = p_g\big(p_{h_1}(|\vec{x}_n|), \dots, p_{h_k}(|\vec{x}_n|)\big)$.

- If f is defined by TRN form g, h, h_ℓ, h_r, where we assume without loss of generality that
  - $|g(x, z, \vec{y})| \leq p_g(|x|, |z|, |\vec{y}|)$,
  - $|h_\ell(z)| \leq c|z|$ and $|h_r(z)| \leq c|z|$,
  - $|h(x, z, \vec{y}, v_\ell, v_r)| \leq a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|) + a(|v_\ell| + |v_r|)$,

then intuitively, at each level of the recursion, the length of the second argument is multiplied by c so that at the bottom level (after $\lg|x|$ steps), the second argument has length $\mathcal{O}(c^{\lg|x|}|z|)$. At the same time, the lengths of each recursive call to f are multiplied by a, which means that the total length of f (bounded by the length of g in the base case) is multiplied by $a^{\lg|x|}$. More precisely, we show that

$$\begin{aligned}
p_f(|x|, |z|, |\vec{y}|) &= \Big(a_0 + \sum(b_i|y_i|)\Big) \cdot \sum_{j=0}^{\lceil\lg|x|\rceil-1} (2a)^j + a_1|x| \cdot \sum_{j=0}^{\lceil\lg|x|\rceil-1} a^j \\
&\quad + a_2|z| \cdot \sum_{j=0}^{\lceil\lg|x|\rceil-1} (2ac)^j + (2a)^{\lceil\lg|x|\rceil} \cdot p_g\big(1, c^{\lceil\lg|x|\rceil}|z|, |\vec{y}|\big) \\
&\leq 2|x|^{\lceil\lg a\rceil+1} \Big[a_0 + \sum(b_i|y_i|) + a_1|x| + a_2|z||x|^{\lceil\lg c\rceil} \\
&\qquad + a \cdot p_g\big(1, c|z||x|^{\lceil\lg c\rceil}, |\vec{y}|\big)\Big].
\end{aligned}$$

Technically speaking, we need to use $\max\{1, |z|\}$ everywhere that $|z|$ appears in this expression, but this does not change the proof substantially besides making it longer to write down. Also, we need to deal separately with special cases such as when $a = 0$ or when the lengths of h_ℓ and h_r are constants independent of $|z|$, but all of these cases simplify the proof so we present the general case only.

Now, if $x = \varepsilon, 0, 1$, then $|f(x, z, \vec{y})| = |g(x, z, \vec{y})| \leq p_g(|x|, |z|, |\vec{y}|) \leq p_g(1, |z|, |\vec{y}|) \leq a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|) + p_g(1, |z|, |\vec{y}|) = p_f(1, |z|, |\vec{y}|)$. If $|x| > 1$, then we consider two subcases. If $|x|$ is even, then $\lceil\lg(|x|/2)\rceil = \lceil\lg|x|\rceil - 1$, so

$$\begin{aligned}
|f(x, z, \vec{y})| &= \big|h\big(x, z, \vec{y}, f(x\blacktriangleleft, h_\ell(z), \vec{y}), f(\blacktriangleright x, h_r(z), \vec{y})\big)\big| \\
&\leq a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|) \\
&\quad + a\big(|f(x\blacktriangleleft, h_\ell(z), \vec{y})| + |f(\blacktriangleright x, h_r(z), \vec{y})|\big) \\
&\leq a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|) \\
&\quad + a\big(p_f(|x\blacktriangleleft|, |h_\ell(z)|, |\vec{y}|) + p_f(|\blacktriangleright x|, |h_r(z)|, |\vec{y}|)\big) \\
&\leq a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|) + 2a\,p_f(|x|/2, c|z|, |\vec{y}|)
\end{aligned}$$

$$
\begin{aligned}
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +2a\Bigg[\Big(a_0+\sum(b_i|y_i|)\Big)\cdot \sum_{j=0}^{\lceil\lg|x|\rceil-2}(2a)^j + a_1\frac{|x|}{2}\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}a^j\\
&\qquad +a_2c|z|\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}(2ac)^j + (2a)^{\lceil\lg|x|\rceil-1}\cdot p_g\big(1, c^{\lceil\lg|x|\rceil-1}c|z|, |\vec{y}|\big)\Bigg]\\
&\le p_f(|x|,|z|,|\vec{y}|).
\end{aligned}
$$

If $|x|$ is odd, then $\lceil\lg((|x|-1)/2)\rceil \le \lceil\lg|x|\rceil - 1$ and $\lceil\lg((|x|+1)/2)\rceil = \lceil\lg|x|\rceil - 1$, so

$$
\begin{aligned}
|f(x,z,\vec{y})| &= \big|h\big(x,z,\vec{y}, f(x\blacktriangleleft, h_\ell(z),\vec{y}), f(\blacktriangleright x, h_r(z),\vec{y})\big)\big|\\
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +a\big(|f(x\blacktriangleleft, h_\ell(z),\vec{y})| + |f(\blacktriangleright x, h_r(z),\vec{y})|\big)\\
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +a\big(p_f(|x\blacktriangleleft|, |h_\ell(z)|, |\vec{y}|) + p_f(|\blacktriangleright x|, |h_r(z)|, |\vec{y}|)\big)\\
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +a\big(p_f((|x|-1)/2, c|z|, |\vec{y}|) + p_f((|x|+1)/2, c|z|, |\vec{y}|)\big)\\
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +a\Bigg[\Big(a_0+\sum(b_i|y_i|)\Big)\cdot \sum_{j=0}^{\lceil\lg|x|\rceil-2}(2a)^j + a_1\frac{|x|-1}{2}\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}a^j\\
&\qquad +a_2c|z|\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}(2ac)^j + (2a)^{\lceil\lg|x|\rceil-1}\cdot p_g\big(1, c^{\lceil\lg|x|\rceil-1}c|z|, |\vec{y}|\big)\Bigg]\\
&\quad +a\Bigg[\Big(a_0+\sum(b_i|y_i|)\Big)\cdot \sum_{j=0}^{\lceil\lg|x|\rceil-2}(2a)^j + a_1\frac{|x|+1}{2}\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}a^j\\
&\qquad +a_2c|z|\cdot\sum_{j=0}^{\lceil\lg|x|\rceil-2}(2ac)^j + (2a)^{\lceil\lg|x|\rceil-1}\cdot p_g\big(1, c^{\lceil\lg|x|\rceil-1}c|z|, |\vec{y}|\big)\Bigg]\\
&\le a_0 + a_1|x| + a_2|z| + \sum(b_i|y_i|)\\
&\quad +\Big(a_0+\sum(b_i|y_i|)\Big)\cdot \sum_{j=1}^{\lceil\lg|x|\rceil-1}(2a)^j + a_1|x|\cdot\sum_{j=1}^{\lceil\lg|x|\rceil-1}a^j\\
&\quad +a_2|z|\cdot\sum_{j=1}^{\lceil\lg|x|\rceil-1}(2ac)^j + (2a)^{\lceil\lg|x|\rceil}\cdot p_g\big(1, c^{\lceil\lg|x|\rceil}|z|, |\vec{y}|\big)\\
&\quad +\frac{-a_1}{2}\sum_{j=1}^{\lceil\lg|x|\rceil-1}a^j + \frac{a_1}{2}\sum_{j=1}^{\lceil\lg|x|\rceil-1}a^j \le p_f(|x|,|z|,|\vec{y}|). \qquad \square
\end{aligned}
$$

Now, we are ready to discuss circuits. For the sake of completeness, we summarize here Bloch's definitions and results, suitably modified to apply to our setting. We will be working with circuit families that compute *functions* instead of relations, *i.e.*, circuits will generally have multiple output gates. In this setting, we define FNC^0 to be the class of functions computed by constant depth circuit families (because the circuits have multiple output gates, this class contains interesting functions, unlike the relational counterpart NC^0), and FNC^1 to be the class of functions computed by logarithmic depth circuit families.

We assume that the gate set for our circuits consists of constants 0 and 1, unary identity ($\approx$) and negation ($\neg$), and binary conjunction ($\wedge$), disjunction ($\vee$), left projection (π_L), and right projection (π_R) (this could be reduced at the cost of longer proofs). Given a circuit family composed of such gates, we identify gates in the circuits by pairs $\langle \text{OUT}, \sigma \rangle$, where OUT is the number of an output gate of the circuit (in binary) and $\sigma \in \{L, R, S\}^*$ represents a path in the circuit from the given output gate, where L and R indicate the left and right inputs of a binary gate, respectively, and S indicates the only input of a unary gate. (Note that we number the output gates from right to left, starting with 1, and similarly for the input gates of each input.)

Now, we want to work with *uniform* families of circuits. Unfortunately, the standard notion of U_{E^*}-uniformity defined by Ruzzo [28] does not make sense for constant depth circuits (it would require the extended connection language for the circuits to be recognizable by an ATM in constant time, which is not even enough time for the ATM to examine a gate number or an input length). To remedy this, Bloch defines a notion of *mapping-uniform* circuits, where the uniformity computation is divided in two phases. The main purpose of the uniformity computation is to be able to recognize connections in the circuit (*i.e.*, given a gate and a path, what gate is at the end of the path?) and gate information (*i.e.*, given a gate number, what type is that gate?). Because of our numbering scheme for gates, determining the descendant of a gate $\langle \text{OUT}, \sigma \rangle$ along a path σ' is easy: the answer is simply $\langle \text{OUT}, \sigma\sigma' \rangle$. Determining the type of an internal gate is also not difficult, as we will see. The hard part, requiring the "two-phase" approach, is to determine which input bit is tied to an input gate (which is necessary to fully specify the "type" of that input gate).

Here is some intuition behind the two phases of the uniformity computation. Essentially, given an input gate $\langle \text{OUT}, \sigma \rangle$ in a circuit, the first phase must use σ to identify *how* to compute the number of the input bit as a function of the output gate number, but without carrying out that computation. This will be done by a deterministic Turing machine running in time proportional to $|\sigma|$, which is bounded by the depth of the circuit, and it is this phase of the computation that will be composed or iterated for functions defined by composition or recursion. The second phase of the computation will only be carried out once, by an ATM that combines the information from the first phase together with the rest of the information about the circuit.

More precisely, the first phase will output a term t_σ that may be thought of as mapping output bit positions to input bit positions that "affect" that output bit, *i.e.*, given an input gate $\langle \textsc{out}, \sigma\rangle$ in the circuit tied to bit number r of some input parameter, we want $t_\sigma(\textsc{out}) = r$. For this purpose, we introduce the *mapping language* of a family of circuits, which consists of a set of functions that encapsulate all the "primitive" dependencies that may exist between output and input bit numbers. Besides the natural numbers used to represent bit positions, we also use the symbol "$\perp$" to indicate that a given output bit does not depend on any input bit. For a circuit with k input parameters $y_1, \dots, y_k$, the mapping language contains the following function symbols (each function is implicitly defined to be equal to $\perp$ when its argument is $\perp$).

- $\textsc{undef}(x) = \perp$
- $\textsc{add}_j(x) = x + |y_j|$
- $\textsc{min}_j(x) = \min\{x, |y_j|\}$
- $\textsc{one}(x) = 1$
- $\textsc{sub}_j(x) = x \dot{-} |y_j|$

Before we move back to circuits, we argue that terms t_σ in the mapping language can be computed in alternating logarithmic time (as a function of $m = \max\{|t_\sigma|, \textsc{out}, |y_1|, \dots, |y_k|\}$). Given $t_\sigma, \textsc{out}, |y_1|, \dots, |y_k|$, an ATM can check in parallel if t_σ contains $\textsc{undef}$ and output $\perp$ immediately if this is the case; otherwise, the ATM guesses the position in t_σ where the last "$\textsc{one}$" appears, and replaces the subsequent part of the term with 1. Once these simple checks are done, the ATM can construct lists of numbers from $|y_1|, \dots, |y_k|$ and subterms of t_σ for each block of functions of the form $\textsc{min}_{j_1}(\dots \textsc{min}_{j_\ell}(t') \dots)$ in t_σ, and evaluate each of these blocks in parallel. The rest of the subterms will contain only $\textsc{add}_j$ and $\textsc{sub}_j$ functions, and these lists of terms and numbers from $|y_1|, \dots, |y_k|$ can be added and subtracted (using two's complement) with standard carry-save techniques.

Now, following Bloch, we say that a circuit family is *mapping-uniform* if there exist a deterministic multi-tape Turing machine P and an ATM Q such that for every gate $\langle \textsc{out}, \sigma\rangle$ in a circuit of the family, the following two conditions hold.

1. If $\langle \textsc{out}, \sigma\rangle$ is an input gate tied to bit number r of some input parameter, then P on input σ runs in time $\mathcal{O}(|\sigma|)$ and outputs a term t_σ in the mapping language such that $t_\sigma(\textsc{out}) = r$.

2. Machine Q on input $\textsc{out}, \sigma, |y_1|, \dots, |y_k|, \tau$ runs in logarithmic time, *i.e.*, in alternating time $\mathcal{O}\big(\log(\max\{\textsc{out}, |\sigma|, |y_1|, \dots, |y_k|\})\big)$ and accepts iff $\langle \textsc{out}, \sigma\rangle$ is an internal gate of type τ or $\langle \textsc{out}, \sigma\rangle$ is an input gate tied to bit number r of y_j and $\tau = \langle j, r\rangle$ (in some standard encoding).

A direct argument shows that any mapping-uniform family of circuits of at least logarithmic depth is also U_{E^*}-uniform. Now, we are ready to show that $L_1 \subseteq FNC^1$.

LEMMA 2.4.4 *For every $f \in BASE$, f can be computed by a uniform circuit family in FNC^0.*

PROOF

ε: The empty circuit is the only one computing this function. Hence, on input $\sigma = \varepsilon$, P outputs $\textsc{undef}(x)$; on input $\textsc{out}, \sigma, \tau$, Q accepts iff $\textsc{out} = 0$, $\sigma = \varepsilon$, and $\tau = \varepsilon$.

$0, 1$: Circuits for these functions consist of a single output gate, one of the "constant" gates 0 or 1, appropriately. Hence, on input $\sigma = \varepsilon$, P outputs $\textsc{undef}(x)$; on input $\textsc{out}, \sigma, \tau$, Q accepts iff $\textsc{out} = 1$, $\sigma = \varepsilon$, and $\tau = 0$ or $\tau = 1$, respectively.

$\blacktriangleright y_1$: Circuits for this function connect output gates to input gates directly, using unary identity gates $\approx$. Hence, on input $\sigma = S$, P outputs simply x; on input $\textsc{out}, \sigma, |y_1|, \tau$, Q accepts iff $1 \leq \textsc{out} \leq \lceil |y_1|/2 \rceil$, $\sigma = \varepsilon$ and $\tau = \approx$, or $\sigma = S$ and $\tau = \langle 1, \textsc{out} \rangle$.

$y_1 \lhd y_2$: Again, circuits for this function connect output gates to input gates directly, using unary identity gates $\approx$. Hence, on input $\sigma = S$, P outputs $\textsc{add}_2(x)$; on input $\textsc{out}, \sigma, |y_1|, |y_2|, \tau$, Q accepts iff $1 \leq \textsc{out} \leq |y_1| \dot{-} |y_2|$, $\sigma = \varepsilon$ and $\tau = \approx$, or $\sigma = S$ and $\tau = \langle 1, r \rangle$ for $r = \textsc{add}_2(\textsc{out})$.

$y_1 \cdot y_2$: The simplest circuits to compute this function would use unary identity gates connected directly to the input bits, as in the last two cases. Unfortunately, this would not allow P to know from σ alone which term to output. Therefore, we do something slightly different, as depicted in Figure 2.4.1.

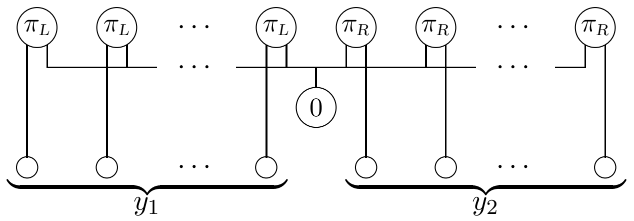


Figure 2.4.1: Uniform circuits for the concatenation function "·".

Now, on input $\sigma = L$, P outputs $\textsc{sub}_2(x)$, while on input $\sigma = R$, P outputs simply x; on input $\textsc{out}, \sigma, |y_1|, |y_2|, \tau$, Q accepts iff

- $1 \leq \textsc{out} \leq |y_2|$ and
  - $\sigma = \varepsilon$, $\tau = \pi_R$, or
  - $\sigma = L$, $\tau = 0$, or
  - $\sigma = R$, $\tau = \langle 2, \textsc{out} \rangle$; or
- $|y_2| + 1 \leq \textsc{out} \leq |y_2| + |y_1|$ and
  - $\sigma = \varepsilon$, $\tau = \pi_L$, or
  - $\sigma = R$, $\tau = 0$, or
  - $\sigma = L$, $\tau = \langle 1, r \rangle$ for $r = \textsc{sub}_2(\textsc{out})$.

y_1 ? (y_2, y_3, y_4): This is the only base function requiring circuits of depth greater than one. There is one consideration making the circuits slightly more complicated than it would seem necessary at first: the shorter of the last two input parameters must be "padded" to the same length as the longer, requiring some extra gates. So, the circuits for ? are of two different kinds: when $|y_1| = 0$, the circuits simply use unary identity gates $\approx$ for output, connected directly to the input gates of y_2, while if $|y_1| > 0$, the circuits are depicted in Figure 2.4.2 (we illustrate the case when $0 < |y_3| < |y_4|$; the other cases are identical except for the obvious modifications to the types of the projection gates).

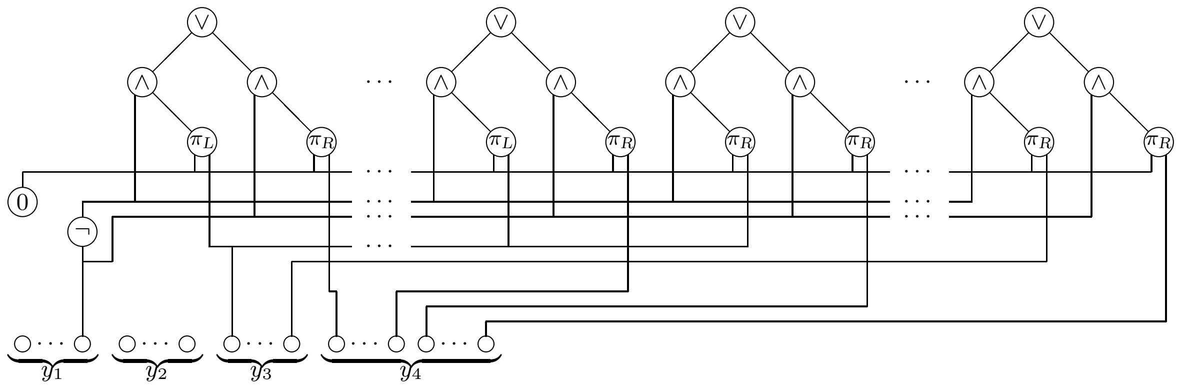


Figure 2.4.2: Uniform circuits for the conditional function "?".

Now, there are only a constant number of possibilities for σ that machine P needs to check. We list each one and the corresponding output for P, as well as a brief explanation indicating which bit of which input parameter is designated by the given path σ, in Table 2.4.1.

σ	output	explanation
S	x	same bit of y_2
RL	ONE(x)	first bit of y_1
LLS	ONE(x)	first bit of y_1 (negated)
LRR	MIN$_3(x)$	bit of y_3 (padded)
RRR	MIN$_4(x)$	bit of y_4 (padded)

Table 2.4.1: Behaviour of machine P for the conditional function.

Next, on input OUT, $\sigma, |y_1|, |y_2|, |y_3|, |y_4|, \tau$, Q accepts iff

- $|y_1| = 0$, $1 \leq$ OUT $\leq |y_2|$, and
  - $\sigma = \varepsilon$, $\tau = \approx$, or
  - $\sigma = S$, $\tau = \langle 2, \text{OUT}\rangle$; or

- $|y_1| > 0$, $1 \le \text{OUT} \le \max\{|y_3|, |y_4|\}$, and
  - $\sigma = \varepsilon$, $\tau = \vee$, or
  - $\sigma = L$, $\tau = \wedge$, or
  - $\sigma = R$, $\tau = \wedge$, or
  - $\sigma = LL$, $\tau = \neg$, or
  - $\sigma = LR$, $\tau = \pi_R$ if $\text{OUT} \le |y_3|$, $\tau = \pi_L$ if $|y_3| < \text{OUT}$, or
  - $\sigma = RL$, $\tau = \langle 1, 1\rangle$, or
  - $\sigma = RR$, $\tau = \pi_R$ if $\text{OUT} \le |y_4|$, $\tau = \pi_L$ if $|y_4| < \text{OUT}$, or
  - $\sigma = LLS$, $\tau = \langle 1, 1\rangle$, or
  - $\sigma = LRL$, $\tau = 0$, or
  - $\sigma = LRR$, $\tau = \langle 3, r\rangle$ for $r = \text{MIN}_3(\text{OUT})$, or
  - $\sigma = RRL$, $\tau = 0$, or
  - $\sigma = RRR$, $\tau = \langle 4, r\rangle$ for $r = \text{MIN}_4(\text{OUT})$.

$\mathcal{I}_k^n(y_1, \ldots, y_n)$: Circuits will use unary identity gates $\approx$ directly connected to the proper input bits. Hence, on input $\sigma = S$, P outputs simply x; on input $\text{OUT}, \sigma, |y_1|, \ldots, |y_n|, \tau$, Q accepts iff $1 \le \text{OUT} \le |y_k|$, $\sigma = \varepsilon$ and $\tau = \approx$, or $\sigma = S$ and $\tau = \langle k, \text{OUT}\rangle$. □

Next, we want to show that functions defined by CRN also have uniform circuit families. For technical reasons (*i.e.*, to simplify the proof), we actually show the result for *left* CRN. (Since L_0 and L_1 remain the same whether CRN or left CRN is used to define them, this is sufficient.)

LEMMA 2.4.5 *If $f(y_1, y_2, \ldots, y_n)$ is defined from h by left CRN on y_1, where h has uniform circuits of depth $d_h(|y_1|, |y_2|, \ldots, |y_n|)$, then f has uniform circuits of depth*

$$\max\left\{d_h(1, |y_2|, \ldots, |y_n|), d_h(2, |y_2|, \ldots, |y_n|), \ldots, d_h(|y_1|, |y_2|, \ldots, |y_n|)\right\}.$$

PROOF A natural circuit for f consists of a series of h-circuits in parallel, one for each output bit of f, where the i-th h-circuit is connected to the first i bits of y_1 (as well as to every other input parameter). Clearly, the depth of this circuit is as stated above. Moreover, given a path σ, machine P simply simulates P_h to get a term t_σ^h and outputs $t_\sigma(x) = t_\sigma^h(\text{ONE}(x))$, while machine Q accepts $\text{OUT}, \sigma, |y_1|, |y_2|, \ldots, |y_n|, \tau$ iff $|y_1| = 0$, $\text{OUT} = 0$, $\sigma = \varepsilon$, $\tau = \varepsilon$, or Q_h accepts $\text{OUT}' = 1, \sigma, |y_1'| = \text{OUT}, |y_2|, \ldots, |y_n|, \tau$. □

Following this, we need to show that the composition of functions computed by uniform families of circuits is also computable by uniform families of circuits, of the right depth.

LEMMA 2.4.6 *If $f(y_1, \ldots, y_n)$ is defined from g and $h_1, \ldots, h_k$ by COMP, where g has uniform circuits of depth $d_g(|z_1|, \ldots, |z_k|)$ and h_i has uniform circuits of depth $d_{h_i}(|y_1|, \ldots, |y_n|)$ (for $1 \leq i \leq k$), then f has uniform circuits of depth*

$$d_g(|h_1(\vec{y}_n)|, \ldots, |h_k(\vec{y}_n)|) + \max\{d_{h_i}(|\vec{y}_n|)\} + \lceil \lg(k) \rceil.$$

PROOF A natural circuit for f consists of a circuit for g whose input gates are connected to the output gates of the corresponding circuits for h_i directly. Unfortunately, machine P cannot tell what term to output just from a path in such a circuit, because P does not have enough time to determine which h_i the path leads into (this would require determining the number of the input gate of g through which the path passes). Therefore, we construct a slightly more complicated circuit by adding a layer of selection gates of depth $\lceil \lg(k) \rceil$ between the circuit for g and the h_i circuits (somewhat like what was done for the concatenation function), in such a way that machine P can easily determine from a path σ which h_i the path goes through. Clearly, the depth of such a circuit is as stated above. Moreover, given a path σ, machine P can break it up into σ_g through g (getting a term t_{σ_g}), followed by σ' through the selection subcircuit (which gives the index i of the function h_i feeding into the g circuit), followed by a final part σ_i through h_i (getting a term t_{σ_i}). P then outputs $t_{\sigma_i}(t_{\sigma_g}(x))$, in time linear in $|\sigma|$. Also, on input $\text{OUT}, \sigma, |y_1|, \ldots, |y_n|, \tau$, machine Q accepts iff $\langle \text{OUT}, \sigma \rangle$ is a gate of type τ in g (by simulating Q_g), or $\langle \text{OUT}, \sigma \rangle$ is a gate within $\lceil \lg(k) \rceil$ steps of an input of g and τ is the correct type of projection gate (computing $t_{\sigma_g}(\text{OUT})$ to figure out which input bit of g the path σ goes through, and then checking the constantly many possibilities for the part of σ through the selection subcircuit), or $\langle \text{OUT}, \sigma \rangle$ is a gate of type τ or an input gate $\tau = \langle j, r \rangle$ for $r = t_{\sigma_i}(t_{\sigma_g}(\text{OUT}))$ within an h_i circuit (computing $t_{\sigma_g}(\text{OUT})$ and tracking the path through the selection subcircuit to figure out the index i, and then simulating Q_{h_i} to verify τ). All this can be done in logarithmic time. □

Finally, we show that functions defined by TRN can be computed by uniform families of circuits.

LEMMA 2.4.7 *If $f(y_1, y_2, y_3, \ldots, y_n)$ is defined from g, h, h_ℓ, and h_r by TRN, where g has uniform circuits of depth $d_g(|y_1|, |y_2|, |y_3|, \ldots, |y_n|)$, h has uniform circuits of depth d_h (a constant), and h_ℓ and h_r have uniform circuits of depth d_ℓ and d_r (constants), then f has uniform circuits of depth*

$$\mathcal{O}\big(d_g(1, |y_2||y_1|^c, |y_3|, \ldots, |y_n|) + \log |y_1| \cdot (d_h + d_\ell + d_r)\big)$$

for some constant $c \in \mathbb{N}$.

PROOF A natural circuit for f consists of a binary tree of h-subcircuits connected to the appropriate bits of the first input and to each other, with a layer of g circuits at the bottom, where the second input of the h and g circuits is connected to subtrees of h_ℓ and h_r circuits. As in the proof for composition, we use layers of selection gates to "glue" together the different subcircuits (between successive h circuits, between h and g circuits, between h or g circuits and the circuits to compute left and right halves of the first input or left and right functions of the second input, as well as between successive "half" functions for the first and second inputs), so that P can tell from the path σ alone which subcircuit a path leads to. The total depth of such a circuit is obviously as stated. Moreover, given a path σ, P can divide the path into portions through h circuits, accumulating the terms for each one, and the final portion of the path through an h, g, h_ℓ, h_r, or "half" circuit, outputting the composition of each term. This can obviously be done in linear time in the length of σ. On input $\text{OUT}, \sigma, |y_1|, |y_2|, |y_3|, \ldots, |y_n|, \tau$, machine Q can break up the path σ into a first part through some number of h circuits and a final part σ' entirely contained inside some h, g, h_ℓ, h_r, or "half" subcircuit. Q can compute the term corresponding to the first part of the path to figure out which output bit of the subcircuit σ passes through, and then simulate the ATM for that subcircuit to verify τ, all in logarithmic time. □

Now, we can put all of these results together.

THEOREM 2.4.1 *For all $f \in L_0$, f can be computed by a uniform family of circuits in FNC^0 (*i.e.*, of constant depth).*

PROOF By induction on the definition of f: Lemma 2.4.4 shows that functions in BASE have uniform constant depth circuits, and Lemmas 2.4.5 and 2.4.6 show that CRN and COMP preserve uniform constant depth. □

THEOREM 2.4.2 *For all $f \in L_1$, f can be computed by a uniform family of circuits in FNC^1 (*i.e.*, of logarithmic depth).*

PROOF By induction on the definition of f.

- If $f \in L_0$, then the preceding theorem shows the result.
- If f is defined by CRN from $h \in L_1$, then Lemma 2.4.5 shows that f can be computed by uniform circuits of the same depth as that of the circuits for h, which shows the result.
- If f is defined by COMP from $g, h_1, \ldots, h_k \in L_1$, then Lemma 2.4.6 and Lemma 2.4.3 (on

the length of functions in L_1) show that f can be computed by uniform circuits of depth

$$\begin{aligned} &d_g(|h_1(\vec{y})|, \ldots, |h_k(\vec{y})|) + \max\{d_{h_i}(|\vec{y}|)\} + \lceil \lg(k) \rceil \\ &\qquad = \mathcal{O}\Big(\log\big(\max\{p_{h_1}(|\vec{y}|), \ldots, p_{h_k}(|\vec{y}|)\}\big) + \log(\max\{|\vec{y}|\})\Big) \\ &\qquad\qquad = \mathcal{O}\big(\log(\max\{|\vec{y}|\})\big). \end{aligned}$$

- If f is defined by TRN from $g \in L_1$ and $h, h_\ell, h_r \in L_0$, then Lemma 2.4.7, together with Lemmas 2.4.2 and 2.4.3 on the length of functions in L_0 and L_1, shows that f can be computed by uniform circuits of depth

$$\begin{aligned} &\mathcal{O}\big(d_g(1, |y_2||y_1|^c, |y_3|, \ldots, |y_n|) + \log|y_1| \cdot (d_h + d_\ell + d_r)\big) \\ &\qquad = \mathcal{O}\big(\log(\max\{|y_2|, \ldots, |y_n|\}) + \log|y_1|\big) \\ &\qquad\qquad = \mathcal{O}\big(\log(\max\{|y_1|, |y_2|, \ldots, |y_n|\})\big). \qquad \square \end{aligned}$$

Chapter 3

The Quantifier-Free Theory T_1

In this chapter, we will define the theory T_1 and give its formal development, including many proofs of simple properties of functions of T_1, as well as many derived rules, and concluding with an illustrative example by proving the pigeonhole principle.

3.1 Definitions

The theory T_1 that we now describe is a *quantifier-free* system, *i.e.*, a free-variable theory with propositional connectives, modeled after Cook's PV [18] but based on the algebra L_1. The language of T_1 consists of the function symbols

$$\{\varepsilon, 0, 1, {}_0, {}_1, \blacktriangleleft, \blacktriangleright, \cdot, \triangleright, \triangleleft, ?\},$$

the function constructors $\{\lambda, \ell\mathrm{CRN}, r\mathrm{CRN}, \mathrm{TRN}\}$, the predicate symbol $\{=\}$, and the usual propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. More precisely, we have the following definitions (where we use the informal notation $x0$ for $(x \cdot 0)$ and $0x$ for $(0 \cdot x)$—similarly for $x1$ and $1x$).

Definition 3.1.1 The *function symbols* and *terms* of T_1 are defined as follows. (The intended interpretation of each function symbol is as given in Chapter 2, where ℓCRN represents "left" (or "reverse") CRN and rCRN represents "right" (or "plain") CRN, and we use the notation introduced there instead of the more formal prefix notation. Also, each function symbol and each term has a *rank* of either 0 or 1—that intuitively indicates which one of L_0 or L_1 the function symbol or term belongs to.)

1. Each variable $x_0, x_1, x_2, \ldots$ is a term of rank 0.

2. If f is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term whose rank is the maximum of the ranks of $f, t_1, \ldots, t_n$ (*i.e.*, the rank of $f(t_1, \ldots, t_n)$ is 0 iff the rank of each one of $f, t_1, \ldots, t_n$ is 0).

3. $\varepsilon, 0, 1$ are 0-place function symbols (constants) of rank 0.

4. ${}_0, {}_1, \blacktriangleleft, \blacktriangleright$ are 1-place function symbols of rank 0.

5. $\cdot, \rhd, \lhd$ are 2-place function symbols of rank 0.

6. ? is a 3-place function symbol of rank 0.

7. If t is a term and $x_1, \ldots, x_n$ is a list of variables including all the variables in t, then $[\lambda x_1 \ldots x_n \,.\, t]$ is an n-place function symbol of the same rank as that of t.

8. If h is an $(n+1)$-place function symbol, then ℓCRN$[h]$ and rCRN$[h]$ are $(n+1)$-place function symbols whose rank is that of h.

9. If g is an $(n+2)$-place function symbol, h is an $(n+4)$-place function symbol of rank 0, and h_ℓ and h_r are 1-place function symbols of rank 0, then TRN$[g, h, h_\ell, h_r]$ is an $(n+2)$-place function symbol of rank 1.

DEFINITION 3.1.2 The *axioms* of T_1 are as follows (except for the propositional and equality axioms, they simply define the function symbols).

0. Any standard, complete set of axioms for the propositional calculus (with equations of the form $x = y$ in place of propositional atoms, for arbitrary variables x and y).

1. (a) $x = x$

   (b) $x = y \;\rightarrow\; y = x$

   (c) $(x = y \wedge y = z) \;\rightarrow\; x = z$

   (d) $(x_1 = y_1 \wedge \cdots \wedge x_k = y_k) \;\rightarrow\; f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$
   (for all k-ary function symbols f, for all $k \geq 1$)

2. $\varepsilon \neq 0 \;\wedge\; 0 \neq 1 \;\wedge\; 1 \neq \varepsilon$

3. (a) $x \cdot \varepsilon = x \;\wedge\; x \cdot y0 = (x \cdot y) \cdot 0 \;\wedge\; x \cdot y1 = (x \cdot y) \cdot 1$

   (b) $x \cdot y = \varepsilon \;\rightarrow\; (x = \varepsilon \wedge y = \varepsilon)$

   (c) $x \cdot y = 0 \;\rightarrow\; (x = \varepsilon \wedge y = 0) \vee (x = 0 \wedge y = \varepsilon)$
   $x \cdot y = 1 \;\rightarrow\; (x = \varepsilon \wedge y = 1) \vee (x = 1 \wedge y = \varepsilon)$

4. (a) $\varepsilon \rhd x = x \;\wedge\; 0y \rhd x = 0 \rhd (y \rhd x) \;\wedge\; 1y \rhd x = 1 \rhd (y \rhd x)$

   (b) $0 \rhd \varepsilon = \varepsilon \;\wedge\; 0 \rhd 0x = x \;\wedge\; 0 \rhd 1x = x$
   $1 \rhd \varepsilon = \varepsilon \;\wedge\; 1 \rhd 0x = x \;\wedge\; 1 \rhd 1x = x$

   (c) $y \rhd x = \varepsilon \;\leftrightarrow\; x \rhd y0 \neq \varepsilon \;\leftrightarrow\; x \rhd y1 \neq \varepsilon$

5. (a) $x = x \lhd \varepsilon \ \wedge \ (x \lhd y) \lhd 0 = x \lhd y0 \ \wedge \ (x \lhd y) \lhd 1 = x \rhd y1$

   (b) $\varepsilon = \varepsilon \lhd 0 \ \wedge \ x = x0 \lhd 0 \ \wedge \ x = x1 \lhd 0$
   $\varepsilon = \varepsilon \lhd 1 \ \wedge \ x = x0 \lhd 1 \ \wedge \ x = x1 \lhd 1$

   (c) $\varepsilon \neq 1y \lhd x \ \leftrightarrow \ \varepsilon \neq 0y \lhd x \ \leftrightarrow \ \varepsilon = x \lhd y$

6. (a) ${}_0\varepsilon = \varepsilon \ \wedge \ {}_0(x0) = {}_0x \cdot 0 \ \wedge \ {}_0(x1) = {}_0x \cdot 0$

   (b) ${}_1\varepsilon = \varepsilon \ \wedge \ {}_1(x0) = {}_1x \cdot 1 \ \wedge \ {}_1(x1) = {}_1x \cdot 1$

7. $\varepsilon\,?\,(x,y,z) = x \ \wedge \ w0\,?\,(x,y,z) = {}_0(z \lhd y) \cdot y \ \wedge \ w1\,?\,(x,y,z) = {}_0(y \lhd z) \cdot z$

8. (a) $(x\blacktriangleleft) \cdot (\blacktriangleright x) = x$

   (b) $(x\blacktriangleleft \lhd \blacktriangleright x) = \varepsilon \ \wedge \ 1 \rhd (x\blacktriangleleft \rhd \blacktriangleright x) = \varepsilon$

9. We use "$x\,?^{EL}\,(y,z)$" as a shorthand notation for $(x\blacktriangleleft \rhd \blacktriangleright x)\,?\,(y,z,z)$ (which equals y if the length of x is even, z if the length of x is odd).

   (a) $(x0)\blacktriangleleft = x\,?^{EL}\big(x\blacktriangleleft, x\blacktriangleleft \cdot ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x)\big) \ \wedge \ (x1)\blacktriangleleft = x\,?^{EL}\big(x\blacktriangleleft, x\blacktriangleleft \cdot ((\blacktriangleright x \cdot 1) \lhd \blacktriangleright x)\big)$

   (b) $\blacktriangleright(x0) = x\,?^{EL}\big(\blacktriangleright x \cdot 0, 1 \rhd (\blacktriangleright x \cdot 0)\big) \ \wedge \ \blacktriangleright(x1) = x\,?^{EL}\big(\blacktriangleright x \cdot 1, 1 \rhd (\blacktriangleright x \cdot 1)\big)$

   (c) $(0x)\blacktriangleleft = x\,?^{EL}\big((0 \cdot x\blacktriangleleft) \lhd 1, 0 \cdot x\blacktriangleleft\big) \ \wedge \ (1x)\blacktriangleleft = x\,?^{EL}\big((1 \cdot x\blacktriangleleft) \lhd 1, 1 \cdot x\blacktriangleleft\big)$

   (d) $\blacktriangleright(0x) = x\,?^{EL}\big((x\blacktriangleleft \rhd (0 \cdot x\blacktriangleleft)) \cdot \blacktriangleright x, \blacktriangleright x\big) \ \wedge \ \blacktriangleright(1x) = x\,?^{EL}\big((x\blacktriangleleft \rhd (1 \cdot x\blacktriangleleft)) \cdot \blacktriangleright x, \blacktriangleright x\big)$

10. $[\lambda x_1 \ldots x_n . t](x_1, \ldots, x_n) = t$

11. (a) $\ell\mathrm{CRN}[h](\varepsilon, \vec{y}) = \varepsilon$
    $\wedge \ \ \ell\mathrm{CRN}[h](0x, \vec{y}) = \big((h(0x,\vec{y}) \cdot 0) \lhd h(0x,\vec{y})\big) \cdot \ell\mathrm{CRN}[h](x,\vec{y})$
    $\wedge \ \ \ell\mathrm{CRN}[h](1x, \vec{y}) = \big((h(1x,\vec{y}) \cdot 0) \lhd h(1x,\vec{y})\big) \cdot \ell\mathrm{CRN}[h](x,\vec{y})$

    (b) $r\mathrm{CRN}[h](\varepsilon, \vec{y}) = \varepsilon$
    $\wedge \ \ r\mathrm{CRN}[h](x0, \vec{y}) = r\mathrm{CRN}[h](x,\vec{y}) \cdot \big(h(x0,\vec{y}) \rhd (0 \cdot h(x0,\vec{y}))\big)$
    $\wedge \ \ r\mathrm{CRN}[h](x1, \vec{y}) = r\mathrm{CRN}[h](x,\vec{y}) \cdot \big(h(x1,\vec{y}) \rhd (0 \cdot h(x1,\vec{y}))\big)$

12. $\mathrm{TRN}[g,h,h_\ell,h_r](x,z,\vec{y}) = x \lhd 1\,?\,(g(x,z,\vec{y}), t, t)$
    where $t = h\big(x, z, \vec{y}, \mathrm{TRN}[g,h,h_\ell,h_r](x\blacktriangleleft, h_\ell(z), \vec{y}), \mathrm{TRN}[g,h,h_\ell,h_r](\blacktriangleright x, h_r(z), \vec{y})\big)$

Remark 3.1.1 By Claim 2.4.1, every function in *FALOGTIME* is represented by some function symbol in T_1, and every function symbol in T_1 represents a function in *FALOGTIME*.

Definition 3.1.3 The *rules of inference* of T_1 are as follows.

0. Any standard, complete set of rules for the propositional calculus.

1. *Substitution* for an arbitrary formula A, variable x, and term t:

$$A \vdash A[t/x]$$

2. *Induction on Notation* (NIND) for an arbitrary formula A and variable x:

   (a) ("left" version) $A[\varepsilon], A[x] \to A[0x], A[x] \to A[1x] \vdash A$

   (b) ("right" version) $A[\varepsilon], A[x] \to A[x0], A[x] \to A[x1] \vdash A$

3. *Tree Induction* (TIND) for an arbitrary formula A, variables x, z, and unary function symbols h_ℓ, h_r of rank 0:

$$A[\varepsilon, z], A[0, z], A[1, z], \big(A[x\blacktriangleleft, h_\ell(z)] \wedge A[\blacktriangleright x, h_r(z)]\big) \to A[x, z] \vdash A$$

3.2 Developing the theory

In this section, we give a formal development of T_1, starting with a few simple theorems and working our way towards multi-variable versions of CRN and NIND. These will be used to define binary addition and "counting" functions, and to prove their properties.

3.2.1 Basic definitions and theorems

CLAIM 3.2.1 $\varepsilon \cdot x = x$

PROOF By NIND on x: $\varepsilon \cdot \varepsilon = \varepsilon$ (by Axiom 3a and Rule 1),

$$\begin{aligned} \varepsilon \cdot x0 &= (\varepsilon \cdot x) \cdot 0 && \text{(Axiom 3a, Rule 1)} \\ &= x \cdot 0 && \text{(Induction Hypothesis, Axiom 1d, Rule 1),} \\ \varepsilon \cdot x1 &= (\varepsilon \cdot x) \cdot 1 && \text{(Axiom 3a, Rule 1)} \\ &= x \cdot 1 && \text{(Induction Hypothesis, Axiom 1d, Rule 1).} \end{aligned}$$ □

CLAIM 3.2.2 $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

PROOF By NIND on z: $x \cdot (y \cdot \varepsilon) = x \cdot y = (x \cdot y) \cdot \varepsilon$ (by Axiom 3a and Rule 1),

$$\begin{aligned} x \cdot (y \cdot z0) &= x \cdot ((y \cdot z) \cdot 0) && \text{(Axioms 3a and 1d, Rule 1)} \\ &= (x \cdot (y \cdot z)) \cdot 0 && \text{(Axiom 3a and Rule 1)} \\ &= ((x \cdot y) \cdot z) \cdot 0 && \text{(Induction Hypothesis)} \\ &= (x \cdot y) \cdot z0 && \text{(Axiom 3a and Rule 1),} \end{aligned}$$

and a similar proof shows $x \cdot (y \cdot z1) = (x \cdot y) \cdot z1$. □

Note that in the proofs that follow, we will not mention explicitly the application of particular axioms, of the induction hypothesis, or of the substitution rule when they are self-evident. Also, when proving a statement by NIND, the cases for $x0$ and $x1$ will often be almost identical (as above) so we will prove both cases at once using "i" to stand for either 0 or 1.

Remark 3.2.1 Be advised that the rest of this section contains a large number of technical claims, together with their proofs, which are included here for the sake of completeness. Most of these claims are of limited interest in themselves, apart from illustrating the style of proofs in T_1 and giving basic properties of functions which will be used in later proofs. For this reason, we recommend that on a first reading, the reader focus mainly on the Definitions, Theorems, and Derived Rules, which contain the essential results.

On "$\triangleright$" and "$\triangleleft$"

We start by defining two functions that will serve as a convenient shorthand notation throughout the rest of this chapter, and prove basic properties of these functions.

Definition 3.2.1 (L) $\gtrdot = [\lambda x \,.\, 1 \triangleright x]$ (R) $\lessdot = [\lambda x \,.\, x \triangleleft 1]$

(To make the notation more consistent with previous usage, we will write "$\gtrdot x$" and "$x \lessdot$" instead of the more formal "$\gtrdot(x)$" and "$\lessdot(x)$", respectively.)

Claim 3.2.3 *(L)* $0 \triangleright x = 1 \triangleright x$ *(R)* $x \triangleleft 1 = x \triangleleft 0$

Proof (L): Immediate from Axiom 4b. (R): Immediate from Axiom 5b. $\square$

Claim 3.2.4 *(L)* $\gtrdot\varepsilon = \varepsilon \wedge \gtrdot(0x) = x \wedge \gtrdot(1x) = x$ *(R)* $\varepsilon\lessdot = \varepsilon \wedge (x0)\lessdot = x \wedge (x1)\lessdot = x$

Proof (L): This is just a restatement of Axiom 4b. (R): This is just a restatement of Axiom 5b. $\square$

Claim 3.2.5 *(L)* $y \triangleright \varepsilon = \varepsilon$ *(R)* $\varepsilon = \varepsilon \triangleleft y$

Proof (L) By NIND on y: $\varepsilon \triangleright \varepsilon = \varepsilon$, $iy \triangleright \varepsilon = i \triangleright (y \triangleright \varepsilon) = i \triangleright \varepsilon = \varepsilon$. (R) By NIND on y: $\varepsilon \triangleright \varepsilon = \varepsilon$, $\varepsilon \triangleleft yi = (\varepsilon \triangleleft y) \triangleleft i = \varepsilon \triangleleft i = \varepsilon$. $\square$

Many of the theorems that follow will be similar to the ones above in having a "left" and "right" version, both of which can be proved in the same way (by using the appropriate version of NIND when necessary). Hence, to avoid unnecessary repetition, we will only give the proof of one version from now on.

CLAIM 3.2.6 *(L)* $\gtrdot(y \rhd x) = y \rhd \gtrdot x$ *(R)* $(x \lhd y)\lessdot = x\lessdot \lhd y$

PROOF (L) By NIND on y: $\gtrdot(\varepsilon \rhd x) = \gtrdot x = \varepsilon \rhd \gtrdot x$, $\gtrdot(iy \rhd x) = \gtrdot\gtrdot(y \rhd x) = \gtrdot(y \rhd \gtrdot x) = iy \rhd \gtrdot x$. $\square$

CLAIM 3.2.7 *(L)* $zy \rhd zx = y \rhd x$ *(R)* $xz \lhd yz = x \lhd y$

PROOF (L) By NIND on z: $\varepsilon y \rhd \varepsilon x = y \rhd x$, $(iz)y \rhd (iz)x = \gtrdot(zy \rhd (iz)x) = zy \rhd \gtrdot(iz)x = zy \rhd zx = y \rhd x$. $\square$

COROLLARY 3.2.8 *(L)* $x \rhd xy = y$ *(R)* $y = yx \lhd x$

COROLLARY 3.2.9 *(L)* $x \rhd x = \varepsilon$ *(R)* $\varepsilon = x \lhd x$

Now, we are ready to define two more functions that will also be used as a convenient shorthand notation for the rest of the chapter.

DEFINITION 3.2.2 (L) $\backprime = [\lambda x \,.\, x \lhd \gtrdot x]$ (R) $' = [\lambda x \,.\, x\lessdot \rhd x]$

(To make the notation more consistent with previous usage, we will write "$\backprime x$" and "x'" instead of the more formal "$\backprime(x)$" and "$'(x)$", respectively.)

CLAIM 3.2.10 *(L)* $\backprime\varepsilon = \varepsilon \wedge \backprime(0x) = 0 \wedge \backprime(1x) = 1$ *(R)* $\varepsilon' = \varepsilon \wedge (x0)' = 0 \wedge (x1)' = 1$

PROOF (L) From the definition, by Claim 3.2.4 and by Corollary 3.2.8: $\backprime\varepsilon = \varepsilon \lhd \gtrdot\varepsilon = \varepsilon \lhd \varepsilon = \varepsilon$, $\backprime(ix) = ix \lhd \gtrdot(ix) = ix \lhd x = i$. $\square$

CLAIM 3.2.11 *(L)* $\backprime x \cdot \gtrdot x = x$ *(R)* $x = x\lessdot \cdot x'$

PROOF (L) By NIND on x: $\backprime\varepsilon \cdot \gtrdot\varepsilon = \varepsilon \cdot \varepsilon = \varepsilon$, $\backprime(ix) \cdot \gtrdot(ix) = i \cdot x = ix$. $\square$

On "·"

CLAIM 3.2.12 *(L)* $\varepsilon \neq ix$ *(R)* $xi \neq \varepsilon$

PROOF (L) By Axioms 2 and 3b, and by taking the contrapositive: $xi = \varepsilon \rightarrow x = \varepsilon \wedge i = \varepsilon \rightarrow i = \varepsilon$. $\square$

CLAIM 3.2.13 *(L)* $0x \neq 1y$ *(R)* $x0 \neq y1$

PROOF (L) By Axiom 2, Claim 3.2.10, Axiom 1d, and by taking the contrapositive: $0x = 1y \rightarrow \backprime(0x) = \backprime(1y) \rightarrow 0 = 1$. $\square$

Claim 3.2.14 *(L)* $\triangleright x = \varepsilon \leftrightarrow x = \varepsilon \lor x = 0 \lor x = 1$ *(R)* $x = \varepsilon \lor x = 0 \lor x = 1 \leftrightarrow \varepsilon = x\triangleleft$

Proof (L) By NIND on x: $\triangleright\varepsilon = \varepsilon \leftrightarrow \varepsilon = \varepsilon$, $\triangleright(ix) = \varepsilon \leftrightarrow x = \varepsilon \leftrightarrow ix = i\varepsilon \leftrightarrow ix = i$. □

Theorem 3.2.1 *(L)* $x = \varepsilon \lor x = 0 \cdot \triangleright x \lor x = 1 \cdot \triangleright x$ *(R)* $x = \varepsilon \lor x = x\triangleleft \cdot 0 \lor x = x\triangleleft \cdot 1$

Proof (L) By NIND on x: $\varepsilon = \varepsilon$, $ix = i \cdot \triangleright(ix)$. □

Note that by Claims 3.2.12 and 3.2.13, we can show that exactly one of the disjuncts holds (*i.e.*, that $x = \varepsilon \rightarrow x \neq 0 \cdot \triangleright x \land x \neq 1 \cdot \triangleright x$ and $x = 0 \cdot \triangleright x \rightarrow x \neq \varepsilon \land x \neq 1 \cdot \triangleright x$ and $x = 1 \cdot \triangleright x \rightarrow x \neq 0 \cdot \triangleright x \land x \neq \varepsilon$).

Corollary 3.2.15 *(L)* $x \neq \varepsilon \leftrightarrow x = 0 \cdot \triangleright x \lor x = 1 \cdot \triangleright x$ *(R)* $x \neq \varepsilon \leftrightarrow x = x\triangleleft \cdot 0 \lor x = x\triangleleft \cdot 1$

Note that Theorem 3.2.1 can easily be generalized to show, for example, $x = \varepsilon \lor x = 0 \lor x = 1 \lor x = 00 \cdot \triangleright\triangleright x \lor x = 01 \cdot \triangleright\triangleright x \lor x = 10 \cdot \triangleright\triangleright x \lor x = 11 \cdot \triangleright\triangleright x$, or, by substituting various terms for x, $xi = i \lor xi = 0 \cdot (\triangleright x)i \lor xi = 1 \cdot (\triangleright x)i$, etc. Because $(A \lor B) \land (A \rightarrow C) \land (B \rightarrow C) \rightarrow C$ is a theorem of T_1, we can use Theorem 3.2.1 together with substitution to define an entire family of "derived rules" in T_1, like the following.

Derived Rule 3.2.1

1. (L) $A[\varepsilon], A[0x], A[1x] \vdash A$ (R) $A[\varepsilon], A[x0], A[x1] \vdash A$
2. $A[\varepsilon], A[0], A[1], A[0x0], A[0x1], A[1x0], A[1x1] \vdash A$

As an example of application of Derived Rule 3.2.1, we prove the following simple claim.

Claim 3.2.16 $(\triangleright x)\triangleleft = \triangleright(x\triangleleft)$

Proof By Derived Rule 3.2.1: $(\triangleright\varepsilon)\triangleleft = \varepsilon = \triangleright(\varepsilon\triangleleft)$, $(\triangleright i)\triangleleft = \varepsilon\triangleleft = \triangleright\varepsilon = \triangleright(i\triangleleft)$, $(\triangleright(ixj))\triangleleft = (xj)\triangleleft = x = \triangleright(ix) = \triangleright((ixj)\triangleleft)$. □

On "$_0$" and "$_1$"

Claim 3.2.17 $_0(x \cdot y) = {}_0x \cdot {}_0y$

Proof By NIND on y: $_0(x\cdot\varepsilon) = {}_0x = {}_0x\cdot\varepsilon = {}_0x\cdot{}_0\varepsilon$, $_0(x\cdot yi) = {}_0(x\cdot y)\cdot 0 = {}_0x\cdot{}_0y\cdot 0 = {}_0x\cdot{}_0(yi)$. □

Note that an identical theorem can be proved with $_1x$ in place of $_0x$. In what follows, we will often need to prove theorems in which "$_0$" or "$_1$" appear, where the particular function used does not matter. We will indicate this by using "$_j$" to stand for either of the above functions.

Claim 3.2.18 ${}_jx \cdot j = j \cdot {}_jx$

Proof By NIND on x: ${}_j\varepsilon \cdot j = \varepsilon \cdot j = j = j \cdot \varepsilon = j \cdot {}_j\varepsilon$, ${}_j(xi) \cdot j = ({}_jx \cdot j) \cdot j = (j \cdot {}_jx) \cdot j = j \cdot ({}_jx \cdot j) = j \cdot {}_j(xi)$. □

Claim 3.2.19 ${}_jx \cdot {}_jy = {}_jy \cdot {}_jx$

Proof By NIND on y, and by Claims 3.2.17 and 3.2.18: ${}_jx \cdot {}_j\varepsilon = {}_jx \cdot \varepsilon = {}_jx = \varepsilon \cdot {}_jx = {}_j\varepsilon \cdot {}_jx$, ${}_jx \cdot {}_j(yi) = {}_jx \cdot {}_jy \cdot j = {}_jy \cdot {}_jx \cdot j = {}_j(yx) \cdot j = j \cdot {}_j(yx) = j \cdot {}_jy \cdot {}_jx = {}_jy \cdot j \cdot {}_jx = {}_j(yi) \cdot {}_jx$. □

Corollary 3.2.20 ${}_j(xy) = {}_j(yx)$

Claim 3.2.21 ${}_jx = {}_j({}_0x)$ *and* ${}_jx = {}_j({}_1x)$

Proof (We will prove only the first property, the second one being almost identical.) By NIND on x: ${}_j\varepsilon = {}_j({}_0\varepsilon)$, ${}_j(xi) = {}_jx \cdot j = {}_j({}_0x) \cdot j = {}_j({}_0x \cdot 0) = {}_j({}_0(xi))$. □

Corollary 3.2.22 ${}_0x = {}_0y \leftrightarrow {}_1x = {}_1y$

Claim 3.2.23 $\gtrdot_j x = {}_jx\lessdot$

Proof By NIND on x, and by Claim 3.2.18: $\gtrdot_j\varepsilon = \gtrdot\varepsilon = \varepsilon = \varepsilon\lessdot = {}_j\varepsilon\lessdot$, $\gtrdot_j(xi) = \gtrdot({}_jx \cdot j) = \gtrdot(j \cdot {}_jx) = {}_jx = ({}_jx \cdot j)\lessdot = {}_j(xi)\lessdot$. □

Claim 3.2.24 *(L)* $\gtrdot({}_jx) = {}_j(\gtrdot x)$ *(R)* ${}_j(x\lessdot) = ({}_jx)\lessdot$

Proof (L) By NIND on x, and by Claim 3.2.17: $\gtrdot({}_j\varepsilon) = \gtrdot\varepsilon = \varepsilon = {}_j\varepsilon = {}_j(\gtrdot\varepsilon)$, $\gtrdot({}_j(ix)) = \gtrdot(j \cdot {}_jx) = {}_jx = {}_j(\gtrdot(ix))$. □

Claim 3.2.25 $x = \varepsilon \leftrightarrow {}_jx = \varepsilon$

Proof By NIND on x: $\varepsilon = \varepsilon \leftrightarrow {}_j\varepsilon = \varepsilon$, $xi = \varepsilon \leftrightarrow {}_jx \cdot j = \varepsilon$. □

Claim 3.2.26 *(L)* $x \rhd y = {}_jx \rhd y$ *(R)* $y \lhd x = y \lhd {}_jx$

Proof (L) By NIND on x: $\varepsilon \rhd y = y = {}_j\varepsilon \rhd y$, $ix \rhd y = \gtrdot(x \rhd y) = \gtrdot({}_jx \rhd y) = j \cdot {}_jx \rhd y = {}_j(ix) \rhd y$. □

Claim 3.2.27 *(L)* $x \rhd {}_jy = {}_j(x \rhd y)$ *(R)* ${}_jy \lhd x = {}_j(y \lhd x)$

Proof (L) By NIND on x, and by Claim 3.2.24: $\varepsilon \rhd {}_jy = {}_jy = {}_j(\varepsilon \rhd y)$, $ix \rhd {}_jy = \gtrdot(x \rhd {}_jy) = \gtrdot_j(x \rhd y) = {}_j(\gtrdot(x \rhd y)) = {}_j(ix \rhd y)$. □

Claim 3.2.28 $\quad {}_j(x \rhd y) = {}_j(y \lhd x)$

Proof $\quad$ By NIND on x, and by Claims 3.2.24 and 3.2.23: ${}_j(\varepsilon \rhd y) = {}_jy = {}_j(y \lhd \varepsilon)$, ${}_j(ix \rhd y) = {}_j(\gtrdot(x \rhd y)) = \gtrdot({}_j(x \rhd y)) = \gtrdot({}_j(y \lhd x)) = ({}_j(y \lhd x))\lessdot = {}_j((y \lhd x)\lessdot) = {}_j(y \lhd ix)$. $\square$

Corollary 3.2.29 $\quad x \rhd {}_jy = {}_jy \lhd x$

Claim 3.2.30 $\quad$ *(L)* ${}^{\backprime}x = x \lhd x\lessdot \qquad$ *(R)* $\gtrdot x \rhd x = x'$

Proof $\quad$ (L) By NIND on x, and by Corollary 3.2.8 and Claims 3.2.26 and 3.2.23: ${}^{\backprime}\varepsilon = \varepsilon = \varepsilon \lhd \varepsilon\lessdot$, ${}^{\backprime}(ix) = i = ix \lhd x = ix \lhd {}_0x = ix \lhd \gtrdot(0_0x) = ix \lhd \gtrdot_0(ix) = ix \lhd {}_0(ix)\lessdot = ix \lhd (ix)\lessdot$.
$\square$

Corollary 3.2.31 $\quad$ *(L)* ${}^{\backprime}(xi) = xi \lhd x \qquad$ *(R)* $x \rhd ix = (ix)'$

Claim 3.2.32

1. *(L)* $y \rhd xi \neq \varepsilon \leftrightarrow y \rhd xi = y \rhd x \cdot i \qquad$ *(R)* $\varepsilon \neq ix \lhd y \leftrightarrow i \cdot x \lhd y = ix \lhd y$
2. *(L)* $y \rhd x \neq \varepsilon \to y \rhd xi \neq \varepsilon \qquad$ *(R)* $\varepsilon \neq x \lhd y \to \varepsilon \neq ix \lhd y$
3. *(L)* ${}_0(y \cdot (y \rhd x)) = {}_0(x \cdot (x \rhd y)) \qquad$ *(R)* ${}_0((x \lhd y) \cdot y) = {}_0((y \lhd x) \cdot x)$

Proof

1. (L) The first direction is proved by Claim 3.2.12: $y \rhd xi = y \rhd x \cdot i \to y \rhd xi \neq \varepsilon$. The other direction is proved by NIND on y: $\varepsilon \rhd xi \neq \varepsilon \to \varepsilon \rhd xi = \varepsilon \rhd x \cdot i$, $jy \rhd xi \neq \varepsilon \to \gtrdot(y \rhd xi) \neq \varepsilon \to y \rhd xi \neq \varepsilon \to y \rhd xi = y \rhd x \cdot i \to \gtrdot(y \rhd xi) = \gtrdot(y \rhd x \cdot i) = y \rhd x \,?^{ZL}\, (\varepsilon, \gtrdot(y \rhd x) \cdot i) \to jy \rhd xi = jy \rhd xi \,?^{ZL}\, (\varepsilon, jy \rhd x \cdot i) = jy \rhd x \cdot i$.

2. (L) By NIND on y and the preceding claim: $\varepsilon \rhd x = x \neq \varepsilon \to \varepsilon \rhd xi = xi \neq \varepsilon$, $jy \rhd x = \gtrdot(y \rhd x) \neq \varepsilon \to y \rhd x \neq \varepsilon \to y \rhd xi \neq \varepsilon \to y \rhd xi = y \rhd x \cdot i \to \gtrdot(y \rhd xi) = \gtrdot(y \rhd x) \cdot i = jy \rhd x \cdot i \neq \varepsilon$.

3. (L) The claim is proved first under the assumption that $x \rhd y = \varepsilon$ (which implies by Axioms 4c and 5c, and by the preceding claims, that $y \rhd xi = y \rhd x \cdot i$, and also implies that $xi \rhd y = \gtrdot(x \rhd y) = \varepsilon$), and then under the assumption that $x \rhd y \neq \varepsilon$ (which implies by Axioms 4c and 5c, and by the preceding claims, that $y \rhd xi = y \rhd x = \varepsilon$). Then, a simple application of modus ponens with the tautology $x \rhd y = \varepsilon \vee x \rhd y \neq \varepsilon$ yields the claim.

   By NIND on x, and under the assumption that $x \rhd y = \varepsilon$: ${}_0(y \cdot (y \rhd \varepsilon)) = {}_0y = {}_0(\varepsilon \cdot (\varepsilon \rhd y))$, ${}_0(y \cdot (y \rhd xi)) = {}_0(y \cdot (y \rhd x) \cdot i) = {}_0(y \cdot (y \rhd x)) \cdot 0 = {}_0(x \cdot (x \rhd y)) \cdot 0 = {}_0x \cdot 0 = {}_0(xi \cdot \varepsilon) = {}_0(xi \cdot (xi \rhd y))$.

By NIND on x, and under the assumption that $x \rhd y \neq \varepsilon$: ${}_0(y \cdot (y \rhd \varepsilon)) = {}_0y = {}_0(\varepsilon \cdot (\varepsilon \rhd y))$, ${}_0(y \cdot (y \rhd xi)) = {}_0(y \cdot \varepsilon) = {}_0(y \cdot (y \rhd x)) = {}_0(x \cdot (x \rhd y)) = {}_0x \cdot {}_0(x \rhd y) = {}_0x \cdot 0 \cdot {>}_0(x \rhd y) = {}_0(xi) \cdot {}_0(xi \rhd y) = {}_0(xi \cdot (xi \rhd y))$. □

On "?" and related functions

Now, we will prove a group of theorems about the conditional function "?". Note that in the statement of some of the theorems below, we will need to express the fact that terms t and u have the same length, something which can be done by the equation ${}_jt = {}_ju$.

First, we introduce two new functions defined in terms of "?" that will be used throughout the rest of this chapter for notational convenience. Whereas the conditional function "?" performs a three-way test on its first argument, the "zero-length conditional" function "$?^{ZL}$" tests whether the length of its first argument is zero or not, and the "even-length conditional" function "$?^{EL}$" tests whether the length of its first argument is even or odd ($?^{EL}$ has already been introduced informally in Axiom 9).

DEFINITION 3.2.3 $?^{ZL} = [\lambda xyz \,.\, x \;?\; (y, z, z)]$

DEFINITION 3.2.4 $?^{EL} = [\lambda xyz \,.\, (x\blacktriangleleft \rhd \blacktriangleright x) \;?^{ZL}\; (y, z)]$

(To make the notation more consistent with previous usage, we will write "$x \;?^{ZL}\; (y, z)$" and "$x \;?^{EL}\; (y, z)$" instead of the more formal "$?^{ZL}(x, y, z)$" and "$?^{EL}(x, y, z)$", respectively.)

CLAIM 3.2.33 $w \;?\; (x, y, z) = w \;?\; \big(x, {}_0(z \lhd y) \cdot y, {}_0(y \lhd z) \cdot z\big)$

PROOF Immediate from Axiom 7. □

CLAIM 3.2.34 $w \;?\; (x, y, z) = w' \;?\; (x, y, z)$

PROOF By NIND on w: $\varepsilon \;?\; (x, y, z) = x = \varepsilon' \;?\; (x, y, z)$, $w0 \;?\; (x, y, z) = {}_0(z \lhd y) \cdot y = 0 \;?\; (x, y, z) = (w0)' \;?\; (x, y, z)$, $w1 \;?\; (x, y, z) = {}_0(y \lhd z) \cdot z = 1 \;?\; (x, y, z) = (w1)' \;?\; (x, y, z)$. □

CLAIM 3.2.35 $wi \;?^{ZL}\; (x, y) = y$

PROOF By Corollary 3.2.8: $wi \;?^{ZL}\; (x, y) = wi \;?\; (x, y, y) = {}_0(y \lhd y) \cdot y = {}_0\varepsilon \cdot y = y$. □

COROLLARY 3.2.36 $iw \;?^{ZL}\; (x, y) = y$

COROLLARY 3.2.37 $w \;?^{ZL}\; (x, y) = {}_jw \;?^{ZL}\; (x, y)$

COROLLARY 3.2.38 $w \;?^{ZL}\; (y, y) = w \;?\; (y, y, y) = y$

Theorem 3.2.2 *For any k-ary function symbol f,*

$$\big({}_0y_1 = {}_0z_1 \wedge \cdots \wedge {}_0y_k = {}_0z_k\big) \rightarrow w\,?\,\big(f(\vec{x}_k), f(\vec{y}_k), f(\vec{z}_k)\big) = f\big(w\,?\,(x_1,y_1,z_1),\ldots,w\,?\,(x_n,y_n,z_n)\big).$$

Proof By NIND on w, and under the assumption that ${}_0y_1 = {}_0z_1 \wedge \cdots \wedge {}_0y_k = {}_0z_k$: $\varepsilon\,?\,\big(f(\vec{x}_k), f(\vec{y}_k), f(\vec{z}_k)\big) = f(\vec{x}_k) = f\big(\varepsilon?(x_1,y_1,z_1),\ldots,\varepsilon?(x_k,y_k,z_k)\big)$, $w0?\big(f(\vec{x}_k), f(\vec{y}_k), f(\vec{z}_k)\big) = f(\vec{y}_k) = f\big(w0\,?\,(x_1,y_1,z_1),\ldots,w0\,?\,(x_k,y_k,z_k)\big)$, $w1\,?\,\big(f(\vec{x}_k), f(\vec{y}_k), f(\vec{z}_k)\big) = f(\vec{z}_k) = f\big(w1\,?\,(x_1,y_1,z_1),\ldots,w1\,?\,(x_k,y_k,z_k)\big)$. □

Claim 3.2.39

$$w\,?\,\big(w\,?\,(x_1,y_1,z_1), w\,?\,(x_2,y_2,z_2), w\,?\,(x_3,y_3,z_3)\big) = w\,?\,\big(x_1, {}_0(z_2 \lhd y_2)\cdot y_2, {}_0(y_3 \lhd z_3)\cdot z_3\big)$$

Proof By NIND on w and by Claim 3.2.32: $\varepsilon?\big(\varepsilon?(x_1,y_1,z_1),\varepsilon?(x_2,y_2,z_2),\varepsilon?(x_3,y_3,z_3)\big) = \varepsilon?(x_1,x_2,x_3) = x_1 = \varepsilon?\big(x_1,{}_0(z_2\lhd y_2)\cdot y_2,{}_0(y_3\lhd z_3)\cdot z_3\big)$, $w0?\big(w0?(x_1,y_1,z_1),w0?(x_2,y_2,z_2),w0?(x_3,y_3,z_3)\big) = w0\,?\,({}_0(z_1\lhd y_1)\cdot y_1, {}_0(z_2\lhd y_2)\cdot y_2, {}_0(z_3\lhd y_3)\cdot y_3) = {}_0\big(({}_0(z_3\lhd y_3)\cdot y_3)\lhd({}_0(z_2\lhd y_2)\cdot y_2)\big)\cdot{}_0(z_2\lhd y_2)\cdot y_2 = {}_0\big(({}_0(y_3\lhd z_3)\cdot z_3)\lhd({}_0(z_2\lhd y_2)\cdot y_2)\big)\cdot{}_0(z_2\lhd y_2)\cdot y_2 = w0\,?\,\big(x_1,{}_0(z_2\lhd y_2)\cdot y_2,{}_0(y_3\lhd z_3)\cdot z_3\big)$, and similarly for $w1$. □

Claim 3.2.40 $w\,?\,(x_0,y_0,z_0)\,?\,(x_1,y_1,z_1) =$

$$w\,?\,\big(x_0\,?\,(x_1,y_1,z_1), ({}_0(z_0\lhd y_0)\cdot y_0)\,?\,(x_1,y_1,z_1), ({}_0(y_0\lhd z_0)\cdot z_0)\,?\,(x_1,y_1,z_1)\big)$$

Proof A straightforward NIND on w, very similar to the proof of Claim 3.2.39. □

Corollary 3.2.41 $w\,?^{ZL}\,(x_0,y_0)\,?^{ZL}\,(x_1,y_1) = w\,?^{ZL}\,\big(x_0\,?^{ZL}\,(x_1,y_1), y_0\,?^{ZL}\,(x_1,y_1)\big)$

Claim 3.2.42 $x\,?^{ZL}\,\big(y\,?^{ZL}\,(z_0,w_0), y\,?^{ZL}\,(z_1,w_1)\big) = y\,?^{ZL}\,\big(x\,?^{ZL}\,(z_0,z_1), x\,?^{ZL}\,(w_0,w_1)\big)$

Proof By NIND on x: $\varepsilon\,?^{ZL}\,\big(y\,?^{ZL}\,(z_0,w_0), y\,?^{ZL}\,(z_1,w_1)\big) = y\,?^{ZL}\,(z_0,w_0) = y\,?^{ZL}\,\big(\varepsilon\,?^{ZL}\,(z_0,z_1), \varepsilon?^{ZL}(w_0,w_1)\big)$, $xi?^{ZL}\big(y?^{ZL}(z_0,w_0), y?^{ZL}(z_1,w_1)\big) = y?^{ZL}(z_1,w_1) = y?^{ZL}\big(xi?^{ZL}(z_0,z_1), xi?^{ZL}(w_0,w_1)\big)$. □

Claim 3.2.43 $w = w\,?\,(\varepsilon, w\lessdot\cdot 0, w\lessdot\cdot 1)$

Proof By NIND on w: $\varepsilon = \varepsilon\,?\,(\varepsilon,\varepsilon\lessdot\cdot 0,\varepsilon\lessdot\cdot 1)$, $w0 = w0\,?\,(\varepsilon, w0, w1) = w0\,?\,(\varepsilon,(w0)\lessdot\cdot 0,(w0)\lessdot\cdot 1)$, $w1 = w1\,?\,(\varepsilon,w0,w1) = w1\,?\,(\varepsilon,(w1)\lessdot\cdot 0,(w1)\lessdot\cdot 1)$. □

Corollary 3.2.44 $w = w\,?^{ZL}\,(\varepsilon, w)$

Theorem 3.2.3 $w?(x,y_0,y_1) = z \leftrightarrow (w=\varepsilon\wedge x=z)\vee(w = w\lessdot\cdot 0\wedge{}_0(y_1\lhd y_0)\cdot y_0 = z)\vee(w = w\lessdot\cdot 1\wedge{}_0(y_0\lhd y_1)\cdot y_1 = z)$

PROOF By NIND on w: $\varepsilon \mathbin{?} (x, y_0, y_1) = z \leftrightarrow x = z \leftrightarrow \varepsilon = \varepsilon \land x = z$, $w0 \mathbin{?} (x, y_0, y_1) = z \leftrightarrow {}_0(y_1 \lhd y_0) \cdot y_0 = z \leftrightarrow w0 = (w0)\lessdot \cdot 0 \land {}_0(y_1 \lhd y_0) \cdot y_0 = z$, $w1 \mathbin{?} (x, y_0, y_1) = z \leftrightarrow {}_0(y_0 \lhd y_1) \cdot y_1 = z \leftrightarrow w1 = (w1)\lessdot \cdot 1 \land {}_0(y_0 \lhd y_1) \cdot y_1 = z$. □

COROLLARY 3.2.45 $w \mathbin{?^{ZL}} (x, y) = z \leftrightarrow (w = \varepsilon \land x = z) \lor (w \neq \varepsilon \land y = x)$

THEOREM 3.2.4 *For any term u, $w \mathbin{?^{ZL}} (u[\varepsilon/w], u) = u$.*

PROOF By induction on the structure of u: if $u = w$, then $w \mathbin{?^{ZL}} (u[\varepsilon/w], u) = w \mathbin{?^{ZL}} (\varepsilon, w) = w$ by Corollary 3.2.44; if $u = x \neq w$, then $w \mathbin{?^{ZL}} (u[\varepsilon/w], u) = w \mathbin{?^{ZL}} (x, x) = x$ by Corollary 3.2.38; if $u = f(t_1, \ldots, t_n)$, then $w \mathbin{?^{ZL}} (u[\varepsilon/w], u) = w \mathbin{?^{ZL}} \big(f(t_1[\varepsilon/w], \ldots, t_n[\varepsilon/w]), f(t_1, \ldots, t_n)\big) = f\big(w \mathbin{?^{ZL}} (t_1[\varepsilon/w], t_1), \ldots, w \mathbin{?^{ZL}} (t_n[\varepsilon/w], t_n)\big) = f(t_1, \ldots, t_n) = u$ by Theorem 3.2.2 and the induction hypothesis. □

CLAIM 3.2.46 *(L)* ${}^\backprime(zx) = z \mathbin{?^{ZL}} ({}^\backprime x, {}^\backprime z)$ *(R)* $(xz)' = z \mathbin{?^{ZL}} (x', z')$

PROOF (L) By NIND on z, and by Corollary 3.2.36: ${}^\backprime(\varepsilon x) = {}^\backprime x = \varepsilon \mathbin{?^{ZL}} ({}^\backprime x, {}^\backprime \varepsilon)$, ${}^\backprime((iz)x) = i = iz \mathbin{?^{ZL}} ({}^\backprime x, {}^\backprime(iz))$. □

CLAIM 3.2.47 *(L)* $\gtrdot(zx) = z \mathbin{?^{ZL}} (\gtrdot x, \gtrdot z \cdot x)$ *(R)* $(xz)\lessdot = z \mathbin{?^{ZL}} (x\lessdot, x \cdot z\lessdot)$

PROOF (L) By NIND on z, and by Corollary 3.2.36: $\gtrdot(\varepsilon x) = \gtrdot x = \varepsilon \mathbin{?^{ZL}} (\gtrdot x, \gtrdot \varepsilon \cdot x)$, $\gtrdot((iz)x) = zx = iz \mathbin{?^{ZL}} (\gtrdot x, \gtrdot(iz) \cdot x)$. □

CLAIM 3.2.48 $v \mathbin{?^{ZL}} (u, t) = u \leftrightarrow (v \neq \varepsilon \to t = u)$

PROOF By NIND on v: $\varepsilon \mathbin{?^{ZL}} (u, t) = u \leftrightarrow u = u \leftrightarrow (\varepsilon \neq \varepsilon \to u = u)$, $vi \mathbin{?^{ZL}} (u, t) = u \leftrightarrow t = u \leftrightarrow (vi \neq \varepsilon \to t = u)$. □

On "◀" and "▶"

CLAIM 3.2.49 *(L)* ${}_j(x\blacktriangleleft) = ({}_jx)\blacktriangleleft$ *(R)* ${}_j(\blacktriangleright x) = \blacktriangleright({}_jx)$

PROOF (L) By NIND on x, Axioms 9, and various theorems proved above: ${}_j(\varepsilon\blacktriangleleft) = \varepsilon = ({}_j\varepsilon)\blacktriangleleft$,

$$\begin{aligned}
{}_j((xi)\blacktriangleleft) &= {}_j\big((x\blacktriangleleft \rhd \blacktriangleright x) \mathbin{?^{ZL}} (x\blacktriangleleft, x\blacktriangleleft \cdot {}^\backprime(\blacktriangleright x \cdot i))\big) \\
&= {}_j(x\blacktriangleleft \rhd \blacktriangleright x) \mathbin{?^{ZL}} \big({}_j(x\blacktriangleleft), {}_j(x\blacktriangleleft \cdot {}^\backprime(\blacktriangleright x \cdot i))\big) \\
&= ({}_j(x\blacktriangleleft) \rhd {}_j(\blacktriangleright x)) \mathbin{?^{ZL}} \big(({}_jx)\blacktriangleleft, {}_j(x\blacktriangleleft) \cdot {}^\backprime{}_j(\blacktriangleright x \cdot i)\big) \\
&= (({}_jx)\blacktriangleleft \rhd \blacktriangleright({}_jx)) \mathbin{?^{ZL}} \big(({}_jx)\blacktriangleleft, ({}_jx)\blacktriangleleft \cdot {}^\backprime(\blacktriangleright({}_jx) \cdot j)\big) \\
&= ({}_jx \cdot j)\blacktriangleleft = ({}_j(xi))\blacktriangleleft \qquad \square
\end{aligned}$$

Basic properties of "$?^{EL}$" can easily be obtained from the basic properties of "$?^{ZL}$", on which it is based. In order to prove properties particular to "$?^{EL}$", we will need the following lemmas.

But first, a few reminders.

- ${}_j(\gtrdot(\blacktriangleright x \cdot i)) = \gtrdot(\blacktriangleright_j x \cdot j) = \gtrdot(j \cdot \blacktriangleright_j x) = {}_j(\blacktriangleright x)$
- $(z \cdot i) \triangleright y = {}_j(z \cdot i) \triangleright y = (j \cdot {}_j z) \triangleright y = \gtrdot({}_j z \triangleright y) = \gtrdot(z \triangleright y)$

Lemma 3.2.50 ${}_0x\blacktriangleleft \triangleright (\blacktriangleright_0 x \cdot 0) = ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \cdot 0$

Proof By Axioms 4c, 5c, and 8b, and by Claim 3.2.32: ${}_0x\blacktriangleleft \triangleleft \blacktriangleright_0 x = \varepsilon \rightarrow {}_0x\blacktriangleleft \triangleright (\blacktriangleright_0 x \cdot 0) \neq \varepsilon \rightarrow {}_0x\blacktriangleleft \triangleright (\blacktriangleright_0 x \cdot 0) = ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \cdot 0$. $\square$

Lemma 3.2.51 ${}_0(xi)\blacktriangleleft \triangleright \blacktriangleright_0(xi) = ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \; ?^{ZL} \; (0, \varepsilon)$

Proof By Lemma 3.2.50:

$$\begin{aligned}
{}_0(xi)\blacktriangleleft \triangleright \blacktriangleright_0(xi) &= x \; ?^{EL} \; \big({}_0x\blacktriangleleft \triangleright (\blacktriangleright_0 x \cdot 0), ({}_0x\blacktriangleleft \cdot {}^{\backprime}(\blacktriangleright_0 x \cdot 0)) \triangleright \gtrdot(\blacktriangleright_0 x \cdot 0)\big) \\
&= x \; ?^{EL} \; \big(({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \cdot 0, ({}_0x\blacktriangleleft \cdot 0) \triangleright \blacktriangleright_0 x\big) \\
&= (x\blacktriangleleft \triangleright \blacktriangleright x) \; ?^{ZL} \; \big({}_0(x\blacktriangleleft \triangleright \blacktriangleright x) \cdot 0, \gtrdot_0(x\blacktriangleleft \triangleright \blacktriangleright x)\big) \\
&= ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \; ?^{ZL} \; \big(0, \varepsilon\big) \qquad \square
\end{aligned}$$

Theorem 3.2.5 $xi \; ?^{EL} \; (y, z) = x \; ?^{EL} \; (z, y)$

Proof By Lemmas 3.2.50 and 3.2.51:

$$\begin{aligned}
xi \; ?^{EL} \; (y, z) &= {}_0(xi) \; ?^{EL} \; (y, z) \\
&= ({}_0(xi)\blacktriangleleft \triangleright \blacktriangleright_0(xi)) \; ?^{ZL} \; (y, z) \\
&= ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \; ?^{ZL} \; (0, \varepsilon) \; ?^{ZL} \; (y, z) \\
&= ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \; ?^{ZL} \; \big(0 \; ?^{ZL} \; (y, z), \varepsilon \; ?^{ZL} \; (y, z)\big) \\
&= ({}_0x\blacktriangleleft \triangleright \blacktriangleright_0 x) \; ?^{ZL} \; (z, y) \\
&= {}_0x \; ?^{EL} \; (z, y) = x \; ?^{EL} \; (z, y) \qquad \square
\end{aligned}$$

Claim 3.2.52 *(L)* $(ixj)\blacktriangleleft = i \cdot x\blacktriangleleft$ *(R)* $\blacktriangleright(ixj) = \blacktriangleright x \cdot j$

PROOF (L) By Axioms 9 and Theorem 3.2.5:

$$\begin{aligned}(ixj)\blacktriangleleft &= xj \mathbin{?^{EL}} \big((i(xj)\blacktriangleleft)\lessdot, i(xj)\blacktriangleleft\big)\\ &= x \mathbin{?^{EL}} \big(i(xj)\blacktriangleleft, (i(xj)\blacktriangleleft)\lessdot\big)\\ &= x \mathbin{?^{EL}} \big(i\cdot x\blacktriangleleft, (i\cdot x\blacktriangleleft\cdot {}^{\backprime}\blacktriangleright x)\lessdot\big)\\ &= x \mathbin{?^{EL}} \big(i\cdot x\blacktriangleleft, i\cdot x\blacktriangleleft\big)\\ &= i\cdot x\blacktriangleleft \qquad \square\end{aligned}$$

CLAIM 3.2.53 *(L)* $(\gtrdot x\lessdot)\blacktriangleleft = \gtrdot(x\blacktriangleleft)$ *(R)* $\blacktriangleright(\gtrdot x\lessdot) = (\blacktriangleright x)\lessdot$

PROOF (L) By Derived Rule 3.2.1 and Claim 3.2.52: $(\gtrdot\varepsilon\lessdot)\blacktriangleleft = \varepsilon = \gtrdot(\varepsilon\blacktriangleleft)$, $(\gtrdot i\lessdot)\blacktriangleleft = \varepsilon = \gtrdot(i\blacktriangleleft)$, $(\gtrdot(ixj)\lessdot)\blacktriangleleft = x\blacktriangleleft = \gtrdot(i\cdot x\blacktriangleleft) = \gtrdot((ixj)\blacktriangleleft)$. $\square$

3.2.2 Further definitions and theorems

In this section, we define many functions in T_1 and prove their basic properties. We also give (and prove) a number of useful derived rules for T_1.

From now on, we will not give proofs that consist only in a straightforward application of NIND. Proof sketches will be given for more complex theorems, and complete proofs are provided in Appendix A for most of the theorems below.

On generalizations of NIND

First, we define some generalizations of NIND based on Derived Rule 3.2.1.

DERIVED RULE 3.2.2
(L) $A[\varepsilon], A[0], A[1], A[x] \to A[00x]\wedge A[01x]\wedge A[10x]\wedge A[11x] \;\vdash\; A$
(R) $A[\varepsilon], A[0], A[1], A[x] \to A[x00]\wedge A[x01]\wedge A[x10]\wedge A[x11] \;\vdash\; A$

PROOF (We will prove only (R), the case for (L) being almost identical.) Let us define a formula $EL[x] : {}_0(x\blacktriangleleft \rhd \blacktriangleright x) = \varepsilon$. By Lemma 3.2.51, we immediately get that $EL[xi] \leftrightarrow \neg EL[x] \leftrightarrow EL[ix]$. To prove that A is true under the given hypotheses, we will show that the hypotheses imply the following two statements:

$$(EL[x] \to A[x]) \wedge (\neg EL[x] \to A[x\lessdot]), \tag{3.2.1}$$

$$(\neg EL[x] \to A[x]) \wedge (EL[x] \to A[x\lessdot]). \tag{3.2.2}$$

Together with the fact that $EL[x]\vee\neg EL[x]$, this will imply that $A[x]\wedge A[x\lessdot]$, *i.e.*, $A[x]$ is true.

We can prove statement 3.2.1 by NIND on x: $(EL[\varepsilon] \rightarrow A[\varepsilon]) \wedge (\neg EL[\varepsilon] \rightarrow A[\varepsilon\lessdot])$ is trivially true since $A[\varepsilon]$ is true by assumption, while

$$(EL[xi] \rightarrow A[xi]) \wedge (\neg EL[xi] \rightarrow A[(xi)\lessdot]) \leftrightarrow (\neg EL[x] \rightarrow A[xi]) \wedge (EL[x] \rightarrow A[x])$$

is true since the second conjunct is true by the induction hypothesis, and so is $(\neg EL[x] \rightarrow A[x\lessdot])$, which, together with the assumption that $A[x] \rightarrow A[xji]$, implies that $A[x\lessdot] \rightarrow A[x\lessdot \cdot ji] \rightarrow A[xi]$.

The same reasoning applies to statement 3.2.2, which concludes the proof. □

Note that this proof can easily be modified to get a similar derived rule for $A[x] \rightarrow A[ixj]$, and it can easily be extended to cover other variations of Theorem 3.2.1.

Next, we want to define simultaneous NIND on two variables. Before we can do this, we need to define a few functions and prove their basic properties.

Definition 3.2.5 (L) $\mathsf{lb} = \big[\lambda xy\,.\, y \,?^{ZL} (\varepsilon, {}^{\backprime}(\gtrdot y \rhd x))\big]$ (R) $\mathsf{rb} = \big[\lambda xy\,.\, y \,?^{ZL} (\varepsilon, (x \lhd y\lessdot)')\big]$

Definition 3.2.6 (L) $\mathsf{lc} = \big[\lambda xy\,.\, x \lhd (y \rhd x)\big]$ (R) $\mathsf{rc} = \big[\lambda xy\,.\, (x \lhd y) \rhd x\big]$

Definition 3.2.7 $\mathsf{min}^L = \big[\lambda xy\,.\, x \lhd y \,?^{ZL} (x, y)\big]$ $\mathsf{max}^L = \big[\lambda xy\,.\, x \rhd y \,?^{ZL} (x, y)\big]$

Claim 3.2.54

1. *(L)* $z \rhd yx = z \rhd y \cdot (z \lhd y) \rhd x$ *(R)* $xy \lhd z = x \lhd (y \rhd z) \cdot y \lhd z$
2. *(L)* $y\lessdot \rhd x = \mathsf{lb}(x, y) \cdot y \rhd x$ *(R)* $x \lhd \gtrdot y = x \lhd y \cdot \mathsf{rb}(x, y)$
3. *(L)* $y \rhd ((x \lhd y) \rhd x) = \varepsilon$ *(R)* $(x \lhd (y \rhd x)) \lhd y = \varepsilon$
4. *(L)* $\mathsf{lc}(\mathsf{rc}(x, y), y) = \mathsf{rc}(x, y)$ *(R)* $\mathsf{rc}(\mathsf{lc}(x, y), y) = \mathsf{lc}(x, y)$
5. *(L)* ${}^{\backprime}(y \rhd x) = y \rhd x \,?^{ZL} \big(\varepsilon, (x \lhd \gtrdot(y \rhd x))'\big)$ *(R)* $(x \lhd y)' = x \lhd y \,?^{ZL} \big(\varepsilon, {}^{\backprime}((x \lhd y)\lessdot \rhd x)\big)$
6. *(L)* $\mathsf{lc}(x, yi) = \mathsf{lc}(x, y) \cdot \mathsf{lb}(x, yi)$ *(R)* $\mathsf{rc}(x, iy) = \mathsf{rb}(x, iy) \cdot \mathsf{rc}(x, y)$
7. *(L)* $\mathsf{lc}(x, y) \cdot y \rhd x = x$ *(R)* $x = x \lhd y \cdot \mathsf{rc}(x, y)$

Now, we can state and prove a derived rule for simultaneous NIND on two variables.

Derived Rule 3.2.3

$$\begin{aligned}
&(LL)\quad A[\varepsilon, y], A[x, \varepsilon], A[x, y] \rightarrow A[0x, 0y] \wedge A[0x, 1y] \wedge A[1x, 0y] \wedge A[1x, 1y] \;\vdash\; A \\
&(LR)\quad A[\varepsilon, y], A[x, \varepsilon], A[x, y] \rightarrow A[0x, y0] \wedge A[0x, y1] \wedge A[1x, y0] \wedge A[1x, y1] \;\vdash\; A \\
&(RL)\quad A[\varepsilon, y], A[x, \varepsilon], A[x, y] \rightarrow A[x0, 0y] \wedge A[x0, 1y] \wedge A[x1, 0y] \wedge A[x1, 1y] \;\vdash\; A \\
&(RR)\quad A[\varepsilon, y], A[x, \varepsilon], A[x, y] \rightarrow A[x0, y0] \wedge A[x0, y1] \wedge A[x1, y0] \wedge A[x1, y1] \;\vdash\; A
\end{aligned}$$

PROOF (We will prove only (RR), the other cases being almost identical.) Under the given assumptions, we will prove $A[x_L \cdot \mathsf{lc}(x_R, z), y_L \cdot \mathsf{lc}(y_R, z)]$ by NIND on z, where

$$x_L = x \lhd \mathsf{min}^L(x, y) \qquad x_R = \mathsf{rc}(x, \mathsf{min}^L(x, y))$$
$$y_L = y \lhd \mathsf{min}^L(x, y) \qquad y_R = \mathsf{rc}(y, \mathsf{min}^L(x, y))$$

Base case: $A[x_L \cdot \mathsf{lc}(x_R, \varepsilon), y_L \cdot \mathsf{lc}(y_R, \varepsilon)] = A[x_L, y_L]$. By the definition of min^L, we know that $\mathsf{min}^L(x, y) = x \vee \mathsf{min}^L(x, y) = y$, which means that $x_L = \varepsilon \vee y_L = \varepsilon$, which implies $(A[x_L, y_L] \leftrightarrow A[x_L, \varepsilon]) \vee (A[x_L, y_L] \leftrightarrow A[\varepsilon, y_L])$, so we know that $A[x_L, y_L]$ is true by the assumptions.

Induction Step: we have that

$$A[x_L \cdot \mathsf{lc}(x_R, zi), y_L \cdot \mathsf{lc}(y_R, zi)] = A[x_L \cdot \mathsf{lc}(x_R, z) \cdot \mathsf{lb}(x_R, zi), y_L \cdot \mathsf{lc}(y_R, z) \cdot \mathsf{lb}(y_R, zi)],$$

which follows directly from the induction hypothesis by the assumptions.

Finally, we know that $A[x_L \cdot \mathsf{lc}(x_R, \mathsf{min}^L(x, y)), y_L \cdot \mathsf{lc}(y_R, \mathsf{min}^L(x, y))] = A[x_L \cdot x_R, y_L \cdot y_R] = A[x, y]$, which completes the proof. $\square$

Note that this rule, and its proof, can easily be extended to more than two variables, giving us a very useful form of NIND on many variables.

On propositional reasoning

Now, we will show how to formalize propositional connectives in T_1. (The definitions are identical to those for L_1, and we use "$x \mathrel{?^B} (y, x)$" instead of the more formal "$?^B(x, y, z)$".)

DEFINITION 3.2.8

1. $?^B = [\lambda xyz \,.\, x \mathrel{?} (z, z, y)]$
2. $\approx^B = [\lambda x \,.\, x \mathrel{?^B} (1, 0)]$
3. $\neg^B = [\lambda x \,.\, x \mathrel{?^B} (0, 1)]$
4. $\wedge^B = [\lambda xy \,.\, x \mathrel{?^B} (\approx^B y, 0)]$
5. $\vee^B = [\lambda xy \,.\, x \mathrel{?^B} (1, \approx^B y)]$
6. $\rightarrow^B = [\lambda xy \,.\, x \mathrel{?^B} (\approx^B y, 1)]$
7. $\leftrightarrow^B = [\lambda xy \,.\, x \mathrel{?^B} (\approx^B y, \neg^B y)]$
8. $\oplus^B = [\lambda xy \,.\, x \mathrel{?^B} (\neg^B y, \approx^B y)]$

The properties of "?" already proven immediately extend to $?^B$ in the obvious way, and the following theorem follows directly from these properties.

Theorem 3.2.6

1. $\approx^B x = 1 \vee \approx^B x = 0$
2. $\approx^B \approx^B x = \approx^B x$
3. $\neg^B x = 1 \leftrightarrow \neg(\approx^B x = 1)$
4. $x \wedge^B y = 1 \leftrightarrow (\approx^B x = 1 \wedge \approx^B y = 1)$
5. $x \vee^B y = 1 \leftrightarrow (\approx^B x = 1 \vee \approx^B y = 1)$
6. $x \rightarrow^B y = 1 \leftrightarrow (\approx^B x = 1 \rightarrow \approx^B y = 1)$
7. $x \leftrightarrow^B y = 1 \leftrightarrow (\approx^B x = 1 \leftrightarrow \approx^B y = 1)$
8. $x \oplus^B y = 1 \leftrightarrow (\approx^B x = 1 \oplus \approx^B y = 1)$

This theorem gives us direct proofs of the usual properties of the defined connectives, from the corresponding properties of the connectives in T_1, and it allows us to introduce the following notation: we will write "t" instead of "$t = 1$" for T_1-terms t. (For example, we could state that "$\approx^B x \oplus^B \neg^B x$" is a theorem.)

On variations of TRN

To define functions by "simple" TRN, we will use STRN$[g, h]$ as shorthand for

$$\big[\lambda x\vec{y}\,.\,\mathrm{TRN}[\lambda xz\vec{y}\,.\,g(x,\vec{y}), \lambda xz\vec{y}v_\ell v_r\,.\,h(x,\vec{y},v_\ell,v_r), \lambda z\,.\,z, \lambda z\,.\,z](x,\varepsilon,\vec{y})\big].$$

The following property is then a direct consequence of the axiom for TRN.

Claim 3.2.55

$$STRN[g,h](x,\vec{y}) = x{<}\,?^{ZL}\Big(g(x,\vec{y}), h\big(x,\vec{y}, STRN[g,h](x\blacktriangleleft,\vec{y}), STRN[g,h](\blacktriangleright x,\vec{y})\big)\Big)$$

On "AND" and "OR"

Definition 3.2.9 $\quad \mathsf{AND} = \mathrm{STRN}[\lambda x\,.\,x, \lambda x v_\ell v_r\,.\,v_\ell \wedge^B v_r]$

Definition 3.2.10 $\quad \mathsf{OR} = \mathrm{STRN}[\lambda x\,.\,x, \lambda x v_\ell v_r\,.\,v_\ell \vee^B v_r]$

We can use TIND to prove the following simple theorem (we give the proof here to illustrate the use of TIND).

Theorem 3.2.7 $\quad \mathsf{AND}({}_1x) = x\ ?^{ZL}\,(\varepsilon, 1)$

PROOF By TIND on x: $\mathsf{AND}({}_1\varepsilon) = \mathsf{AND}(\varepsilon) = \varepsilon = \varepsilon \;?^{ZL}\, (\varepsilon, 1)$, $\mathsf{AND}({}_1 i) = \mathsf{AND}(1) = 1 = i \;?^{ZL}\, (\varepsilon, 1)$, $\mathsf{AND}({}_1 x) = \mathsf{AND}(({}_1 x)\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright({}_1 x)) = \mathsf{AND}({}_1(x\blacktriangleleft)) \wedge^B \mathsf{AND}({}_1(\blacktriangleright x)) = x\blacktriangleleft \;?^{ZL}\, (\varepsilon, 1) \wedge^B \blacktriangleright x \;?^{ZL}\, (\varepsilon, 1) = 1 \wedge^B 1 = 1$. $\square$

Similarly, we can prove that $\mathsf{OR}({}_0 x) = x \;?^{ZL}\, (\varepsilon, 0)$.

Now, we want to prove some more basic properties of AND and OR, the most important of which being $\mathsf{AND}(xy) = \mathsf{AND}(x) \wedge^B \mathsf{AND}(y)$ (and similarly for OR). This property would naturally be proved using NIND since it involves the concatenation of two variables, but AND is defined by TRN which makes it more natural to use TIND. In fact, we will use TIND to prove the property but because of the messy interaction between concatenation recursion and tree recursion, the proof will unfortunately not be as simple as one might expect.

Before we get started, note that it is a simple matter to extend Derived Rule 3.2.2 to give us rules similar to the following ones.

DERIVED RULE 3.2.4 $A[0], A[1], y \neq \varepsilon \wedge A[y] \rightarrow A[y0] \wedge A[y1] \;\vdash\; x \neq \varepsilon \rightarrow A[x]$

DERIVED RULE 3.2.5

$$\begin{aligned} &A[00], A[01], A[10], A[11], A[000], \ldots, A[111], y\blacktriangleleft\lessdot \neq \varepsilon \wedge A[y\blacktriangleleft] \wedge A[\blacktriangleright y] \rightarrow A[y] \\ \vdash\;& x \neq \varepsilon \wedge x \neq 0 \wedge x \neq 1 \rightarrow A[x] \end{aligned}$$

These rules can then be used to prove the following claim and theorem.

LEMMA 3.2.56

1. *(L)* $x \neq \varepsilon \wedge x \neq 0 \wedge x \neq 1 \rightarrow (\gtrdot x \cdot j)\blacktriangleleft = \gtrdot(x\blacktriangleleft) \cdot \backprime\blacktriangleright x$
   (R) $x \neq \varepsilon \wedge x \neq 0 \wedge x \neq 1 \rightarrow \blacktriangleright(\gtrdot x \cdot j) = \gtrdot\blacktriangleright x \cdot j$

2. *(L)* $\backprime x \wedge^B \mathsf{AND}(\gtrdot x \cdot j) = \mathsf{AND}(x) \wedge^B j$ *(R)* $j \wedge^B \mathsf{AND}(x) = \mathsf{AND}(j \cdot x\lessdot) \wedge^B x'$

3. $\backprime x \wedge^B \mathsf{AND}(\gtrdot x) = \mathsf{AND}(x) = \mathsf{AND}(x\lessdot) \wedge^B x'$ *for* $x \neq \varepsilon, 0, 1$

THEOREM 3.2.8 $\mathsf{AND}(xy) = \mathsf{AND}(x) \wedge^B \mathsf{AND}(y)$ *for* $x, y \neq \varepsilon$

PROOF By Lemma 3.2.56 and by Derived Rule 3.2.4 on y: $\mathsf{AND}(xi) = \mathsf{AND}(x) \wedge^B i = \mathsf{AND}(x) \wedge^B \mathsf{AND}(i)$, $\mathsf{AND}(x(yi)) = \mathsf{AND}(xy) \wedge^B i = \mathsf{AND}(x) \wedge^B \mathsf{AND}(y) \wedge^B i = \mathsf{AND}(x) \wedge^B \mathsf{AND}(yi)$.
$\square$

Note that a similar lemma and theorem can be used to show $\mathsf{OR}(xy) = \mathsf{OR}(x) \vee^B \mathsf{OR}(y)$ for $x, y \neq \varepsilon$.

On generalizations of CRN—part I

Now, we will define simultaneous concatenation recursion on notation for many variables, prove its basic properties, and define a few more useful functions based on this generalized CRN. But first, we must prove a number of technical lemmas.

THEOREM 3.2.9 $\quad {}_jx = {}_jy \leftrightarrow x \rhd y = \varepsilon = x \lhd y$

CLAIM 3.2.57 *For $f = r\mathit{CRN}[h]$,*

1. ${}_j(f(x, \vec{y})) = {}_jx$
2. $f(x, \vec{y}) \lhd z = f(x \lhd z, \vec{y})$
3. $\mathsf{lb}(f(x, \vec{y}), z) = x \rhd z \; ?^{ZL} \big((0 \cdot h(\mathsf{lc}(x, z), \vec{y}))', \varepsilon\big)$ *for* $x, z \neq \varepsilon$
4. $\mathsf{lc}(f(x, \vec{y}), z) = f(\mathsf{lc}(x, z), \vec{y})$

(A similar claim can be proved about ℓCRN.)

DEFINITION 3.2.11 $\quad$ (L) $\mathsf{lp}_j = \big[\lambda xy \,.\, {}_j(y \lhd x) \cdot x\big]$ $\qquad$ (R) $\mathsf{rp}_j = \big[\lambda xy \,.\, x \cdot {}_j(x \rhd y)\big]$

DEFINITION 3.2.12

$$\mathsf{max}^L_1 = [\lambda x \,.\, x]$$
$$\mathsf{max}^L_{k+1} = \big[\lambda x\vec{x}_k \,.\, \mathsf{max}^L\big(x, \mathsf{max}^L_k(\vec{x}_k)\big)\big]$$

LEMMA 3.2.58

1. *(L)* $y \rhd x = x \rhd y \; ?^{ZL} (y \rhd x, \varepsilon)$ $\qquad$ *(R)* $x \lhd y = x \rhd y \; ?^{ZL} (x \lhd y, \varepsilon)$
2. *(L)* $(y \lhd (x \rhd y)) \rhd x = y \rhd x$ $\qquad$ *(R)* $x \lhd ((y \lhd x) \rhd y) = x \lhd y$

CLAIM 3.2.59

1. *(L)* $\mathsf{lp}_j(x, \mathsf{max}^L(x, y)) = \mathsf{lp}_j(x, y)$ $\qquad$ *(R)* $\mathsf{rp}_j(x, \mathsf{max}^L(x, y)) = \mathsf{rp}_j(x, y)$
2. *(L)* $\mathsf{lb}\big(\mathsf{lp}_0(xi, \mathsf{max}^L(xi, yj)), \mathsf{max}^L(xi, yj)\big) = i$
   (R) $\mathsf{rb}\big(\mathsf{rp}_0(ix, \mathsf{max}^L(ix, jy)), \mathsf{max}^L(ix, jy)\big) = i$

(Note that this can easily be generalized to more than two variables.)
Now, we are ready to define $\ell\mathrm{CRN}_m$ and $r\mathrm{CRN}_m$.

DEFINITION 3.2.13

$$\begin{aligned}
\ell\mathrm{CRN}_m[h] &= \Big[\lambda\vec{x}_m\vec{y}\,.\,\ell\mathrm{CRN}\big[\lambda z\vec{x}_m\vec{y}\,.\,h(\mathsf{rc}(x_1,z),\dots,\mathsf{rc}(x_m,z),\vec{y})\big] \\
&\qquad\qquad \big(\mathsf{max}^L_m(\vec{x}_m),\mathsf{lp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m)),\dots,\mathsf{lp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m)),\vec{y}\big)\Big] \\
r\mathrm{CRN}_m[h] &= \Big[\lambda\vec{x}_m\vec{y}\,.\,r\mathrm{CRN}\big[\lambda z\vec{x}_m\vec{y}\,.\,h(\mathsf{lc}(x_1,z),\dots,\mathsf{lc}(x_m,z),\vec{y})\big] \\
&\qquad\qquad \big(\mathsf{max}^L_m(\vec{x}_m),\mathsf{rp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m)),\dots,\mathsf{rp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m)),\vec{y}\big)\Big]
\end{aligned}$$

A simple application of Derived Rule 3.2.3 together with Claim 3.2.59, both generalized to the m variables $x_1,\dots,x_m$, suffices to show the following basic theorem about $\ell\mathrm{CRN}_m$ and $r\mathrm{CRN}_m$.

THEOREM 3.2.10

$$\begin{aligned}
&\ell CRN_m[h](x_1,\dots,x_m,\vec{y}) \\
&\quad = x_1\cdot\dots\cdot x_m\ ?^{ZL}\Big(\varepsilon,{}^{\backprime}\big(h\big(\mathsf{lp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m)),\dots,\mathsf{lp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m)),\vec{y}\big)\cdot 0\big) \\
&\qquad\qquad \cdot\,\ell CRN_m[h]\big(\gtrdot\mathsf{lp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m)),\dots,\gtrdot\mathsf{lp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m)),\vec{y}\big)\Big) \\
&r CRN_m[h](x_1,\dots,x_m,\vec{y}) \\
&\quad = x_1\cdot\dots\cdot x_m\ ?^{ZL}\Big(\varepsilon,r CRN_m[h]\big(\mathsf{rp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m))\lessdot,\dots,\mathsf{rp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m))\lessdot,\vec{y}\big) \\
&\qquad\qquad \cdot\,\big(0\cdot h\big(\mathsf{rp}_0(x_1,\mathsf{max}^L_m(\vec{x}_m)),\dots,\mathsf{rp}_0(x_m,\mathsf{max}^L_m(\vec{x}_m)),\vec{y}\big)\big)'\Big)
\end{aligned}$$

Using $\ell\mathrm{CRN}_m$, we can now define some useful functions and prove their basic properties.

DEFINITION 3.2.14

1. $\mathsf{not}^B = \ell\mathrm{CRN}[\lambda x\,.\,\neg^B({}^{\backprime}x)]$

2. $\mathsf{and}^B_m = \ell\mathrm{CRN}_m\big[\lambda\vec{x}_m\,.\,({}^{\backprime}x_1)\wedge^B\cdots\wedge^B({}^{\backprime}x_m)\big]$

3. $\mathsf{or}^B_m = \ell\mathrm{CRN}_m\big[\lambda\vec{x}_m\,.\,({}^{\backprime}x_1)\vee^B\cdots\vee^B({}^{\backprime}x_m)\big]$

4. $\mathsf{xor}^B_m = \ell\mathrm{CRN}_m\big[\lambda\vec{x}_m\,.\,({}^{\backprime}x_1)\oplus^B\cdots\oplus^B({}^{\backprime}x_m)\big]$

5. $\mathsf{iff}^B_m = \ell\mathrm{CRN}_m\big[\lambda\vec{x}_m\,.\,(({}^{\backprime}x_1)\leftrightarrow^B({}^{\backprime}x_2))\wedge^B\cdots\wedge^B(({}^{\backprime}x_{m-1})\leftrightarrow^B({}^{\backprime}x_m))\big]$

CLAIM 3.2.60

1. $\mathsf{and}^B_m(x_1i_1,\dots,x_mi_m) = \mathsf{and}^B_m(\vec{x}_m)\cdot\mathsf{and}^B_m(\vec{i}_m)$

2. ${}_jx_1=\cdots={}_jx_m\wedge{}_jy_1=\cdots={}_jy_m\rightarrow\mathsf{and}^B_m(x_1y_1,\dots,x_my_m)=\mathsf{and}^B_m(\vec{x}_m)\cdot\mathsf{and}^B_m(\vec{y}_m)$.

3. $\mathsf{and}^B_m(\vec{x}_m)\lessdot = \mathsf{and}^B_m(x_1\lessdot,\dots,x_m\lessdot)$

4. $\mathsf{and}^B_m(\vec{x}_m)\lhd y = \mathsf{and}^B_m(x_1\lhd y,\dots,x_m\lhd y)$

And similarly for or^B_m, iff^B_m, and not^B. We can now prove a theorem relating the functions AND and OR to each other.

Theorem 3.2.11 $\neg^B\mathsf{AND}(x) = \mathsf{OR}(\mathsf{not}^B(x))$ *and* $\neg^B\mathsf{OR}(x) = \mathsf{AND}(\mathsf{not}^B(x))$ *for* $x \neq \varepsilon$

On generalizations of CRN—part II

Finally, we are ready to generalize CRN to operate on blocks of bits instead of single bits. For technical reasons to be discussed below, this will be done only for variable-length blocks of bits whose lengths are powers of two.

In order to get such a version of CRN, we first need to define a length-division function. Ideally, we would like to define a function $\mathsf{div}^L(x,y)$ whose length would be equal to $\lfloor |x|/|y| \rfloor$, but this seems to be impossible in T_1. Instead, we can define a function $\mathsf{powdiv}^L(x,y)$ whose length is equal to $\lfloor |x|/2^{\lceil \lg |y| \rceil} \rfloor$, *i.e.*, the function divides the length of x by the smallest power of 2 larger than or equal to the length of y. Luckily, this will be sufficient for our purposes, as will be seen in Chapter 5.

The first functions we define are a function that returns a string whose length is the smallest power of two larger than or equal to the length of its input, and a function that tests whether or not the length of its input is a power of two.

Definition 3.2.15 $\mathsf{pow}^L = \mathrm{STRN}[1, \lambda y v_\ell v_r \,.\, v_r \cdot v_r]$

Definition 3.2.16 $\mathsf{ispow}^L = \mathrm{STRN}[\lambda y \,.\, \varepsilon, \lambda y v_\ell v_r \,.\, y \;?^{EL}\, (v_r, 1)]$

(Note that ispow^L returns ε if the length of its input is a power of two and 1 otherwise.) The basic properties of pow^L and ispow^L are now easy to prove by TIND.

Claim 3.2.61

1. $\blacktriangleright\mathsf{pow}^L(y) = \mathsf{pow}^L(\blacktriangleright y) = \mathsf{pow}^L(y)\blacktriangleleft$ *(for* $y \neq \varepsilon, 0, 1$*)*
2. ${}_1\mathsf{pow}^L(y) = \mathsf{pow}^L({}_1 y) = \mathsf{pow}^L(y)$
3. $\mathsf{pow}^L(\mathsf{pow}^L(y)) = \mathsf{pow}^L(y)$
4. $\mathsf{pow}^L(y) \rhd y = \varepsilon$
5. $\mathsf{ispow}^L(\mathsf{pow}^L(y)) = \varepsilon$

Before we can define the length-division function, we need to define a “length multiplication” function (this is just the “smash” function).

Definition 3.2.17 $\# = \mathrm{STRN}[\lambda xy \,.\, x \;?^{ZL}\, (\varepsilon, y), \lambda xy v_\ell v_r \,.\, v_\ell \cdot v_r]$

These properties of # can then be proven with simple applications of TIND and NIND.

CLAIM 3.2.62

1. $\varepsilon \# y = \varepsilon = x \# \varepsilon$
2. (L) ${}_1(xi \# y) = {}_1(x \# y) \cdot {}_1y$ (R) ${}_1(x \# yi) = {}_1(x \# y) \cdot {}_1x$
3. (L) ${}_1((x \cdot y) \# z) = {}_1(x \# z) \cdot {}_1(y \# z)$ (R) ${}_1(x \# (y \cdot z)) = {}_1(x \# y) \cdot {}_1(x \# z)$
4. ${}_1(x \# y) = {}_1(y \# x)$

Now, we can define the powdiv^L function, and a corresponding powmod^L function.

DEFINITION 3.2.18

$$\mathsf{powdiv}^L = \mathrm{TRN}\big[\lambda yx\,.\, y \mathrel{?^{ZL}} (\varepsilon, {}_1x), \lambda yxv_\ell v_r\,.\, v_r, \blacktriangleleft, \blacktriangleleft\big]$$
$$\mathsf{powmod}^L = \big[\lambda xy\,.\,(\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x,y)) \rhd {}_1x\big]$$

Straightforward applications of TIND then prove these basic properties.

CLAIM 3.2.63

1. $\mathsf{powdiv}^L(x,y) = \mathsf{powdiv}^L({}_1x, {}_1y)$
2. $\mathsf{powdiv}^L(x,y) = \mathsf{powdiv}^L(x, \mathsf{pow}^L(y))$
3. $x \rhd \mathsf{pow}^L(y) \neq \varepsilon \rightarrow \mathsf{powdiv}^L(x,y) = \varepsilon$
4. $\mathsf{powdiv}^L(\mathsf{pow}^L(y) \# z, y) = y \mathrel{?^{ZL}} (\varepsilon, {}_1z)$

And properties of powmod^L follow directly from the properties of powdiv^L.

COROLLARY 3.2.64

1. $\mathsf{powmod}^L(x,y) = \mathsf{powmod}^L({}_1x, {}_1y)$
2. $\mathsf{powmod}^L(x,y) = \mathsf{powmod}^L(x, \mathsf{pow}^L(y))$
3. $x \rhd \mathsf{pow}^L(y) \neq \varepsilon \rightarrow \mathsf{powmod}^L(x,y) = {}_1x$
4. $\mathsf{powmod}^L(\mathsf{pow}^L(y) \# z, y) = \varepsilon$

We need just a few more technical lemmas about powdiv^L and powmod^L before we can define generalized CRN and prove its properties.

Claim 3.2.65

1. $x \rhd (\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x, y)) = \varepsilon$
2. $y \neq \varepsilon \rightarrow \mathsf{powmod}^L(x, y) \rhd \mathsf{pow}^L(y) \neq \varepsilon$
3. $y \neq \varepsilon \rightarrow \mathsf{powdiv}^L(x1, y) = \mathsf{powdiv}^L(x, y) \cdot \big((\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y)\ ?^{ZL}\ (1, \varepsilon)\big)$
   $y \neq \varepsilon \rightarrow \mathsf{powmod}^L(x1, y) = (\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y)\ ?^{ZL}\ \big(\varepsilon, \mathsf{powmod}^L(x, y) \cdot 1\big)$
4. $\mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) = y\ ?^{ZL}\ (\varepsilon, {}_1z) \cdot \mathsf{powdiv}^L(x, y) \wedge$
   $\mathsf{powmod}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) = \mathsf{powmod}^L(x, y)$
5. $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon \rightarrow \mathsf{powdiv}^L(x, y) = \mathsf{powdiv}^L(x \lhd \mathsf{pow}^L(y), y) \cdot 1$
   $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon \rightarrow \mathsf{powmod}^L(x, y) = \mathsf{powmod}^L(x \lhd \mathsf{pow}^L(y), y)$

At last, we are ready to define the generalized versions of ℓCRN and rCRN which we will name "ℓpowCRN" and "$r$powCRN". Functions defined by ℓpowCRN or $r$powCRN take two extra parameters u, v as input, and essentially perform CRN on their first input by replacing blocks of $|\mathsf{pow}^L(u)|$ bits with blocks of $|\mathsf{pow}^L(v)|$ bits. Just like $(2^k, 2^\ell)$-CRN in Chapter 2, we will simulate these generalized forms of CRN by using powdiv^L and powmod^L to extract the correct substring of the first input to pass to h and to output the correct bits of h in sequence. Intuitively, ℓpowCRN$[g, h]$ and $r$powCRN$[g, h]$ will behave as follows (for all strings z such that $|z| = |\mathsf{pow}^L(u)|$).

$$\begin{aligned}
\ell\text{powCRN}[g, h](x, u, v, \vec{y}) &= g(x, u, v, \vec{y}) \qquad (\text{if } |x| < |\mathsf{pow}^L(u)|)\\
\ell\text{powCRN}[g, h](z \cdot x, u, v, \vec{y}) &= \mathsf{la}_0\big(h(z \cdot x, u, v, \vec{y}), \mathsf{pow}^L(v)\big) \cdot \ell\text{powCRN}[g, h](x, u, v, \vec{y})\\
r\text{powCRN}[g, h](x, u, v, \vec{y}) &= g(x, u, v, \vec{y}) \qquad (\text{if } |x| < |\mathsf{pow}^L(u)|)\\
r\text{powCRN}[g, h](x \cdot z, u, v, \vec{y}) &= r\text{powCRN}[g, h](x, u, v, \vec{y}) \cdot \mathsf{ra}_0\big(h(x \cdot z, u, v, \vec{y}), \mathsf{pow}^L(v)\big)
\end{aligned}$$

Where we have used the functions la_j and ra_j, whose definitions and basic properties (easily proved by Derived Rule 3.2.3) appear below.

Definition 3.2.19
(L) $\mathsf{la}_j = \big[\lambda xy\,.\,{}_j(y \lhd x) \cdot ((x \lhd y) \rhd x)\big]$ (R) $\mathsf{ra}_j = \big[\lambda xy\,.\,(x \lhd (y \rhd x)) \cdot {}_j(x \rhd y)\big]$

Claim 3.2.66

1. (L) ${}_1(\mathsf{la}_j(x, y)) = {}_1y$ (R) ${}_1y = {}_1(\mathsf{ra}_j(x, y))$
2. (L) $\mathsf{lc}(\mathsf{la}_j(x, y), y) = \mathsf{la}_j(x, y)$ (R) $\mathsf{ra}_j(x, y) = \mathsf{rc}(\mathsf{ra}_j(x, y), y)$

DEFINITION 3.2.20

$$\begin{aligned}\ell\mathrm{powCRN}[g,h] = \Big[\lambda xuv\vec{y}.\, \ell\mathrm{CRN}\Big[\lambda zxuv\vec{y}.\\ \mathsf{rb}\Big(\mathsf{la}_0\Big(h\big(\mathsf{rc}\big(x, (\mathsf{pow}^L(u) \# (1\cdot \mathsf{powdiv}^L(\gtrdot z, v)))\cdot \mathsf{powmod}^L(x,u)\big), u, v, \vec{y}\big),\\ \mathsf{pow}^L(v)\Big), 1\cdot \mathsf{powmod}^L(\gtrdot z, v)\Big)\Big]\big(\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u), x, u, v, \vec{y}\big)\\ \cdot\, g\big(\mathsf{rc}(x, \mathsf{powmod}^L(x,u)), u, v, \vec{y}\big)\Big]\end{aligned}$$

$$\begin{aligned}r\mathrm{powCRN}[g,h] = \Big[\lambda xuv\vec{y}.\, g\big(\mathsf{lc}(x, \mathsf{powmod}^L(x,u)), u, v, \vec{y}\big)\cdot r\mathrm{CRN}\Big[\lambda zxuv\vec{y}.\\ \mathsf{lb}\Big(\mathsf{ra}_0\Big(h\big(\mathsf{lc}\big(x, \mathsf{powmod}^L(x,u)\cdot(\mathsf{pow}^L(u) \# (\mathsf{powdiv}^L(z\lessdot, v)\cdot 1))\big), u, v, \vec{y}\big),\\ \mathsf{pow}^L(v)\Big), \mathsf{powmod}^L(z\lessdot, v)\cdot 1\Big)\Big]\big(\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u), x, u, v, \vec{y}\big)\Big]\end{aligned}$$

THEOREM 3.2.12 *For $u \neq \varepsilon$ and $v \neq \varepsilon$,*

$\ell powCRN[g,h](x,u,v,\vec{y}) =$

$$x \rhd \mathsf{pow}^L(u) \;?^{ZL} \Big(\mathsf{la}_0(h(x,u,v,\vec{y}), \mathsf{pow}^L(v))\cdot \ell powCRN[g,h](\mathsf{pow}^L(u) \rhd x, u, v, \vec{y}), g(x,u,v,\vec{y})\Big),$$

$rpowCRN[g,h](x,u,v,\vec{y}) =$

$$x \rhd \mathsf{pow}^L(u) \;?^{ZL} \Big(rpowCRN[g,h](x \lhd \mathsf{pow}^L(u), u, v, \vec{y})\cdot \mathsf{ra}_0(h(x,u,v,\vec{y}), \mathsf{pow}^L(v)), g(x,u,v,\vec{y})\Big).$$

PROOF We prove the theorem for $r$powCRN only, the case for ℓpowCRN being almost identical. To start with, if $x \rhd \mathsf{pow}^L(u) \neq \varepsilon$, then Claim 3.2.63 and Corollary 3.2.64 give us the result immediately since $\mathsf{powdiv}^L(x,u) = \varepsilon$ and $\mathsf{powmod}^L(x,u) = {}_1x$.

Next, suppose that $x \rhd \mathsf{pow}^L(u) = \varepsilon$. Then, Claim 3.2.65 implies that $\mathsf{powdiv}^L(x,u) \neq \varepsilon$, which shows that $\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u) = (\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u)\lessdot)\cdot \mathsf{pow}^L(v) = (\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x \lhd \mathsf{pow}^L(u), u))\cdot \mathsf{pow}^L(v)$. The following facts are then direct consequences of preceding claims, and hold for all strings z such that $z \neq \varepsilon \wedge z \lhd \mathsf{pow}^L(v) = \varepsilon$:

- $\mathsf{lc}(x, y\cdot z) = \mathsf{lc}(x,y)\cdot \mathsf{lc}(y \rhd x, z)$ (easy to prove by NIND on z),
- $\mathsf{powmod}^L\big((\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u)\lessdot)\cdot {}_1z\lessdot, \mathsf{pow}^L(v)\big)\cdot 1 = \mathsf{powmod}^L({}_1z\lessdot, \mathsf{pow}^L(v))\cdot 1 = {}_1z\lessdot\cdot 1 = {}_1z$,
- $\mathsf{lc}\Big(x, \mathsf{powmod}^L(x,u)\cdot\Big(\mathsf{pow}^L(u) \# \Big(\mathsf{powdiv}^L\big((\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u)\lessdot)\cdot z\lessdot, v\big)\cdot 1\Big)\Big)\Big) = \mathsf{lc}\big(x, \mathsf{powmod}^L(x,u)\cdot(\mathsf{pow}^L(u) \# (\mathsf{powdiv}^L(x,u)\lessdot\cdot 1))\big) = \mathsf{lc}(x, {}_1x) = x$.

These facts can be used to prove by NIND on z that

$$\begin{aligned}x \rhd \mathsf{pow}^L(u) = \varepsilon \wedge z \lhd \mathsf{pow}^L(v) = \varepsilon \to f\big((\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x,u)\lessdot)\cdot {}_1z, x, u, v, \vec{y}\big) =\\ f\big(\mathsf{pow}^L(v) \# \mathsf{powdiv}^L(x \lhd \mathsf{pow}^L(u), u), x, u, v, \vec{y}\big)\cdot \mathsf{rc}\big(\mathsf{ra}_0(h(x,u,v,\vec{y}), \mathsf{pow}^L(v)), z\big)\end{aligned}$$

(where we use "f" to denote the function defined by rCRN in the definition of $r$powCRN), and putting $z = \mathsf{pow}^L(v)$ in this last fact gives us the theorem. $\square$

3.2.3 Numerical definitions and theorems

In this section, we will give definitions for numerical predicates and functions (*i.e.*, ones that treat their string arguments as encoding binary numbers) and prove their properties.

On "$=^N$" and "$<^N$"

The definitions of "$=^N$" and "$<^N$" inside T_1 are the same as in L_1. Intuitively, $x =^N y$ if the two strings are equal when padded on the left with 0's to the same length.

DEFINITION 3.2.21 $\quad =^N = \big[\lambda xy\,.\,\mathsf{AND}(1 \cdot \mathsf{iff}_2^B(x,y))\big]$

In a similar way, $x <^N y$ if there is a bit position where x has a 0, y has a 1, and the portions of x and y to the left of that position are numerically equal.

DEFINITION 3.2.22

$$<^N = \big[\lambda xy\,.\,\mathsf{OR}\big(r\mathrm{CRN}_2[\lambda xy\,.\,(x' <^B y') \wedge^B (x{\lessdot} =^N y{\lessdot})](\mathsf{lp}_0(x,y), \mathsf{lp}_0(y,x))\big)\big]$$

(Where we used "$i <^B j$" as shorthand for "$\neg^B i \wedge^B j$".)

Now, we prove basic properties of the two predicates just defined. A simple NIND suffices to show the following theorem.

THEOREM 3.2.13

1. $x =^N \varepsilon \leftrightarrow x = {}_0x$

2. $x =^N y \leftrightarrow \mathsf{lp}_0(x,y) = \mathsf{lp}_0(y,x)$

This immediately implies that "$=^N$" is an equivalence relation. If we define $=^L = [\lambda xy\,.\,(x \rhd y) \cdot (x \lhd y) \mathbin{?^{ZL}} (\varepsilon, 1)]$ and $=^S = [\lambda xy\,.\,\neg^B(x =^L y) \wedge^B x =^N y]$, then the theorem we just proved immediately implies that $x =^S y \leftrightarrow x = y$. Together with the facts about propositional connectives proved in Theorem 3.2.6, this means that for any formula A of T_1, there exists a term $\widehat{A}$ of T_1 such that T_1 can prove $A \leftrightarrow \widehat{A}$ (with our usual convention whereby "$\widehat{A}$" stands for the formula $\widehat{A} = 1$).

Now, we prove more properties of $=^N$ and $<^N$. By Theorem 3.2.10, we have the following two theorems.

CLAIM 3.2.67

1. $x0 =^N y0 \leftrightarrow x =^N y \leftrightarrow x1 =^N y1$

2. $\neg^B(x0 =^N y1)$

3. $\neg^B(x1 =^N y0)$

CLAIM 3.2.68

1. $x0 <^N y0 = x <^N y \vee^B (x =^N y \wedge^B 0 <^B 0) = x <^N y$

2. $x0 <^N y1 = x <^N y \vee^B (x =^N y \wedge^B 0 <^B 1) = x \leq^N y$

3. $x1 <^N y0 = x <^N y \vee^B (x =^N y \wedge^B 1 <^B 0) = x <^N y$

4. $x1 <^N y1 = x <^N y \vee^B (x =^N y \wedge^B 1 <^B 1) = x <^N y$

Simple proofs by NIND now suffice to show the following lemma.

CLAIM 3.2.69

1. $\neg^B(x <^N \varepsilon)$

2. $\neg^B(x <^N x)$

3. $\neg^B(\varepsilon <^N {}_0x)$

Using the notation "$x >^N y$" for $y <^N x$, "$x \leq^N y$" for $x <^N y \vee^B x =^N y$, and "$\geq^N$" similarly defined, we have the following theorem. (We give its proof here because it is representative of the kind of proof that will be used for most theorems concerning numerical functions.)

THEOREM 3.2.14

$$\begin{aligned}
&(x <^N y \wedge^B \neg^B(x =^N y) \wedge^B \neg^B(x >^N y))\\
\vee^B\ &(\neg^B(x <^N y) \wedge^B x =^N y \wedge^B \neg^B(x >^N y))\\
\vee^B\ &(\neg^B(x <^N y) \wedge^B \neg^B(x =^N y) \wedge^B x >^N y)
\end{aligned}$$

PROOF By Derived Rule 3.2.3, and the lemma above: When $y = \varepsilon$, the statement of the theorem reduces to $(x =^N \varepsilon \wedge^B \neg^B(x >^N \varepsilon)) \vee^B (\neg^B(x =^N \varepsilon) \wedge^B x >^N \varepsilon)$, which can be proved by regular NIND on x: $\varepsilon =^N \varepsilon \wedge^B \neg^B(\varepsilon >^N \varepsilon)$, $(x0 =^N \varepsilon \wedge^B \neg^B(x0 >^N \varepsilon)) \vee^B (\neg^B(x0 =^N \varepsilon) \wedge^B x0 >^N \varepsilon) = (x =^N \varepsilon \wedge^B \neg^B(x >^N \varepsilon)) \vee^B (\neg^B(x =^N \varepsilon) \wedge^B x >^N \varepsilon)$, $(x1 =^N \varepsilon \wedge^B \neg^B(x1 >^N \varepsilon)) \vee^B (\neg^B(x1 =^N \varepsilon) \wedge^B x1 >^N \varepsilon) = \varepsilon <^N x \vee^B \varepsilon =^N x = \varepsilon \leq^N x$. We can show that the statement holds when $x = \varepsilon$ in the same way. Next, we have four cases to consider:

$$\begin{aligned}
&(x0 <^N y0 \wedge^B \neg^B(x0 =^N y0) \wedge^B \neg^B(x0 >^N y0))\\
\vee^B&(\neg^B(x0 <^N y0) \wedge^B x0 =^N y0 \wedge^B \neg^B(x0 >^N y0))\\
\vee^B&(\neg^B(x0 <^N y0) \wedge^B \neg^B(x0 =^N y0) \wedge^B x0 >^N y0) = \quad (x <^N y \wedge^B \neg^B(x =^N y) \wedge^B \neg^B(x >^N y))\\
&\qquad \vee^B(\neg^B(x <^N y) \wedge^B x =^N y \wedge^B \neg^B(x >^N y))\\
&\qquad \vee^B(\neg^B(x <^N y) \wedge^B \neg^B(x =^N y) \wedge^B x >^N y)
\end{aligned}$$

and similarly for $x1, y1$,

$$\begin{aligned}
&(x0 <^N y1 \wedge^B \neg^B(x0 =^N y1) \wedge^B \neg^B(x0 >^N y1)) \\
&\vee^B(\neg^B(x0 <^N y1) \wedge^B x0 =^N y1 \wedge^B \neg^B(x0 >^N y1)) \\
&\vee^B(\neg^B(x0 <^N y1) \wedge^B \neg^B(x0 =^N y1) \wedge^B x0 >^N y1) = \quad (x \leq^N y \wedge^B 1 \wedge^B \neg^B(x >^N y)) \\
&\qquad\qquad\qquad\qquad \vee^B(\neg^B(x \leq^N y) \wedge^B 0 \wedge^B \neg^B(x >^N y)) \\
&\qquad\qquad\qquad\qquad \vee^B(\neg^B(x \leq^N y) \wedge^B 1 \wedge^B x >^N y) \\
&\qquad\qquad\qquad = \quad (x <^N y \wedge^B \neg^B(x >^N y)) \\
&\qquad\qquad\qquad\qquad \vee^B(x =^N y \wedge^B \neg^B(x >^N y)) \\
&\qquad\qquad\qquad\qquad \vee^B(\neg^B(x <^N y) \wedge^B \neg^B(x =^N y) \wedge^B x >^N y)
\end{aligned}$$

and similarly for $x1, y0$. □

Corollary 3.2.70 $\varepsilon \leq^N x$

Corollary 3.2.71 $x =^N y = x \leq^N y \wedge^B x \geq^N y$

Next, from the fact that $\mathsf{lp}_0(0x, y) = \mathsf{lp}_0(x, y) \ \vee \ \mathsf{lp}_0(0x, y) = 0 \cdot \mathsf{lp}_0(x, y)$ (which can easily be proved by cases depending on the length of $x \rhd y$), simple proofs by Derived Rule 3.2.3 show the following lemma.

Lemma 3.2.72

1. $x =^N y = 0x =^N y = x =^N 0y = 0x =^N 0y$
2. $x <^N y = 0x <^N 0y$
3. $x <^N y = 0x <^N y = x <^N 0y$

This lemma can be used, with a generalization of Derived Rule 3.2.3 to three variables, to show the following theorems and their corollaries (from Theorem 3.2.13).

Theorem 3.2.15 $x =^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x =^N y \wedge y >^N z \rightarrow x >^N z$

Corollary 3.2.73 $x =^N y \wedge y \leq^N z \rightarrow x \leq^N z$ *and* $x =^N y \wedge y \geq^N z \rightarrow x \geq^N z$

Theorem 3.2.16 $x <^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x >^N y \wedge y >^N z \rightarrow x >^N z$

Corollary 3.2.74 $x \leq^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x \geq^N y \wedge y >^N z \rightarrow x >^N z$

Corollary 3.2.75 $x \leq^N y \wedge y \leq^N z \rightarrow x \leq^N z$ *and* $x \geq^N y \wedge y \geq^N z \rightarrow x \geq^N z$

On "$|\cdot|$" and "succ^N"

Now, we define the binary length function "$|\cdot|$" and the numerical successor function "succ^N" as in L_1, and prove some of their basic properties.

DEFINITION 3.2.23

$$\begin{aligned} \mathsf{cuss}^N &= \ell\mathrm{CRN}\big[\lambda x\,.\,\mathsf{AND}(1 \gtrdot x)\ ?^B\ (\neg^{B\backprime}x, {}^{\backprime}x)\big] \\ \mathsf{succ}^N &= [\lambda x\,.\,\mathsf{cuss}^N(0x)] \end{aligned}$$

Simple proofs by NIND show the following theorem (proving the relevant properties first for the auxiliary function cuss^N, and then for succ^N).

CLAIM 3.2.76

$$\begin{aligned} \mathsf{succ}^N(\varepsilon) &= 1 \\ \mathsf{succ}^N(x0) &= 0x1 \\ \mathsf{succ}^N(x1) &= \mathsf{succ}^N(x) \cdot 0 \end{aligned}$$

Using this theorem, a simple NIND will now prove the following properties.

CLAIM 3.2.77

$$\begin{aligned} \mathsf{succ}^N(0x) &= 0 \cdot \mathsf{succ}^N(x) \\ \mathsf{succ}^N(1x) &= {}^{\backprime}\mathsf{succ}^N(x) \cdot \neg^{B\backprime}\mathsf{succ}^N(x) \cdot {\gtrdot}\mathsf{succ}^N(x) \\ {}^{\backprime}\mathsf{succ}^N(x) &= \mathsf{AND}(1x) \\ \mathsf{succ}^N(x) &= \mathsf{AND}(1x)\ ?^B\ \big(1 \cdot {}_0x, 0 \cdot {\gtrdot}\mathsf{succ}^N(x)\big) \end{aligned}$$

Now, we can prove a few theorems involving succ^N together with some of the other numerical functions already defined.

THEOREM 3.2.17

1. $x <^N \mathsf{succ}^N(x)$
2. $x >^N y = x \geq^N \mathsf{succ}^N(y)$

The binary length function is defined in the same way as in L_1, as follows.

DEFINITION 3.2.24 $\quad |\cdot| = \mathrm{STRN}[{}_1, \lambda x v_\ell v_r\,.\,x\ ?^{EL}\ (v_\ell \cdot 0, v_\ell \cdot 1)]$

(To be consistent with previous notation, we will write "$|x|$" instead of the more formal "$||(x)$".)

CLAIM 3.2.78 $|x| = |_j x|$

THEOREM 3.2.18 $_j x = {}_j y \leftrightarrow |x| = |y|$

PROOF One direction ($_j x = {}_j y \to |x| = |y|$) is immediate from the preceding claim. The other is proved by TIND on y (with $h_\ell = \blacktriangleleft$ and $h_r = \blacktriangleright$): $|x| = |\varepsilon| \to |x| = \varepsilon \to x = \varepsilon \to {}_j x = {}_j \varepsilon$, $|x| = |i| \to |x| = 1 \to x = i' \to {}_j x = {}_j i$, and assuming that $|x\blacktriangleleft| = |y\blacktriangleleft| \to {}_j x\blacktriangleleft = {}_j y\blacktriangleleft$ and $|\blacktriangleright x| = |\blacktriangleright y| \to {}_j \blacktriangleright x = {}_j \blacktriangleright y$, we have that

$$\begin{aligned}
|x| = |y| &\to x \mathbin{?^{EL}} \big(|x\blacktriangleleft| \cdot 0, |x\blacktriangleleft| \cdot 1\big) = y \mathbin{?^{EL}} \big(|y\blacktriangleleft| \cdot 0, |y\blacktriangleleft| \cdot 1\big) \\
&\to \big(x\blacktriangleleft \rhd \blacktriangleright x = \varepsilon \wedge y\blacktriangleleft \rhd \blacktriangleright y = \varepsilon \wedge |x\blacktriangleleft| \cdot 0 = |y\blacktriangleleft| \cdot 0\big) \vee \\
&\quad \big(x\blacktriangleleft \rhd \blacktriangleright x \neq \varepsilon \wedge y\blacktriangleleft \rhd \blacktriangleright y \neq \varepsilon \wedge |x\blacktriangleleft| \cdot 1 = |y\blacktriangleleft| \cdot 1\big) \\
&\to \big(x\blacktriangleleft \rhd \blacktriangleright x = \varepsilon \wedge y\blacktriangleleft \rhd \blacktriangleright y = \varepsilon \wedge {}_j x\blacktriangleleft = {}_j y\blacktriangleleft\big) \vee \\
&\quad \big(x\blacktriangleleft \rhd \blacktriangleright x = i \wedge y\blacktriangleleft \rhd \blacktriangleright y = i' \wedge {}_j x\blacktriangleleft = {}_j y\blacktriangleleft\big) \\
&\to {}_j x = {}_j y
\end{aligned}$$

(where the two cases for $|x\blacktriangleleft| \cdot 0 = |y\blacktriangleleft| \cdot 1$ and $|x\blacktriangleleft| \cdot 1 = |y\blacktriangleleft| \cdot 0$ were not included in the disjunction on the second and third lines since they are known to be false). □

The following theorem can be proved with an easy TIND and its corollaries are immediate from previously proved theorems.

THEOREM 3.2.19 $|xi| =^N \mathsf{succ}^N(|x|)$

COROLLARY 3.2.79 $|x| <^N |xi|$

COROLLARY 3.2.80 $x >^N |y\lessdot| \to x \geq^N |y|$

On "masking" functions

In order to define binary addition, and to prove its properties, we will need "masking" functions like the ones that were defined in L_1. We give their definition and basic properties here.

DEFINITION 3.2.25

$$\begin{aligned}
\mathsf{first}_0 &= r\mathrm{CRN}[\lambda x\,.\,\mathsf{AND}(1x\lessdot) \mathbin{?^B} (\neg^B x', 0)] \\
\mathsf{first}_1 &= r\mathrm{CRN}[\lambda x\,.\,\mathsf{OR}(x\lessdot) \mathbin{?^B} (0, x')]
\end{aligned}$$

DEFINITION 3.2.26 $\mathsf{maskbit} = \big[\lambda xy\,.\,\mathsf{OR}(\mathsf{and}_2^B(x, y))\big]$

DEFINITION 3.2.27 $\mathsf{delfirst}_1 = \big[\lambda x\,.\,\mathsf{and}_2^B\big(x, \mathsf{not}^B(\mathsf{first}_1(x))\big)\big]$

The basic theorem below, as well as its corollary, can both be proved with a simple NIND.

THEOREM 3.2.20

$$\mathsf{first}_0(0x) = 1 \cdot {}_0x \qquad \mathsf{first}_0(1x) = 0 \cdot \mathsf{first}_0(x)$$
$$\mathsf{first}_1(0x) = 0 \cdot \mathsf{first}_1(x) \qquad \mathsf{first}_1(1x) = 1 \cdot {}_0x$$

COROLLARY 3.2.81 $\mathsf{first}_0(x) = \mathsf{first}_1(\mathsf{not}^B(x))$

On binary addition

Before we define binary addition and prove its properties, let us make a remark about "numerical" functions. If a formula A contains only terms made up of functions f with the property that $f(x_1, \ldots, x_m) = f\big(\mathsf{lp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)), \ldots, \mathsf{lp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m))\big)$ (which happens to be the case for the numerical functions), then $A[x_1, \ldots, x_m] \leftrightarrow \big({}_jx_1 = \cdots = {}_jx_m \to A[x_1, \ldots, x_m]\big)$. Thus, we can use the following special form of Derived Rule 3.2.3 to prove any such formula A (the rule is stated only for two variables but can easily be extended to more).

DERIVED RULE 3.2.6 $A[\varepsilon, \varepsilon], {}_jx = {}_jy \wedge A[x, y] \to A[0x, 0y] \wedge A[0x, 1y] \wedge A[1x, 0y] \wedge A[1x, 1y] \vdash {}_jx = {}_jy \to A[x, y]$

(The conclusion of the rule can easily be proved from the antecedent by a simple application of Derived Rule 3.2.3.)

Now, binary addition is defined just as in L_1, as follows.

DEFINITION 3.2.28 $\mathsf{carry}^N = \ell\mathrm{CRN}_2\big[\lambda xy \,.\, \mathsf{maskbit}\big(\mathsf{and}^B_2(x, y), \mathsf{first}_0(\mathsf{xor}^B_2(x, y))\big)\big]$

DEFINITION 3.2.29 $+^N = \big[\lambda xy \,.\, \mathsf{xor}^B_3(\mathsf{carry}^N(x, y) \cdot 0, x, y)\big]$

(To make the notation consistent with previous usage, we will write "$x +^N y$" instead of the more formal "$+^N(x, y)$".) The commutativity of "$+^N$" is a direct result of the commutativity of each function involved in its definition.

THEOREM 3.2.21 $x +^N y = y +^N x$

Proving the associativity of "$+^N$" will be slightly more complicated. First, we relate the functions $+^N$ and succ^N through the following lemma and theorem.

LEMMA 3.2.82

$$\mathsf{carry}^N(x0, 1) = \mathsf{carry}^N(x, \varepsilon) \cdot 0 = {}_0x0$$
$$\mathsf{carry}^N(x1, 1) = x \mathbin{?^{ZL}} \big(1, \mathsf{carry}^N(x, 1) \cdot 1\big)$$

Theorem 3.2.22 $\quad x +^N 1 = x \mathbin{?^{ZL}} \big(0 \cdot \mathsf{succ}^N(x), \mathsf{succ}^N(x)\big) =^N \mathsf{succ}^N(x)$

Next, we can state certain facts about the carry function.

Claim 3.2.83 *For* ${}_jx = {}_jy$,

$$\begin{aligned}
\mathsf{carry}^N(x, \varepsilon) &= {}_0x \\
\mathsf{carry}^N(x, x) &= x \\
\mathsf{carry}^N(0x, 0y) &= 0 \cdot \mathsf{carry}^N(x, y) \\
\mathsf{carry}^N(1x, 0y) &= {\backprime}\mathsf{carry}^N(x, y) \cdot \mathsf{carry}^N(x, y) \\
\mathsf{carry}^N(1x, 1y) &= 1 \cdot \mathsf{carry}^N(x, y)
\end{aligned}$$

Note that we omitted the property $\mathsf{carry}^N(0x, 1y) = {\backprime}\mathsf{carry}^N(x, y) \cdot \mathsf{carry}^N(x, y)$ from this theorem since it follows directly by the commutativity of carry^N. This will be the case for many of the theorems and proofs about $+^N$ that we will now present: for the sake of brevity, we will omit statements and proofs that follow directly from previous ones by commutativity. The following claim follows directly from the corresponding properties for carry^N.

Claim 3.2.84 *For* ${}_jx = {}_jy$,

$$\begin{aligned}
x +^N \varepsilon &= 0x \\
x +^N x &= x0 \\
0x +^N 0y &= 0 \cdot (x +^N y) \\
1x +^N 0y &= {\backprime}(x +^N y) \cdot \neg^{B}{\backprime}(x +^N y) \cdot {\gtrdot}(x +^N y) \\
1x +^N 1y &= 1 \cdot (x +^N y)
\end{aligned}$$

Now, although we can use Claim 3.2.84 to prove theorems about $+^N$ by Derived Rule 3.2.6, we will also have need of the following theorem further on.

Claim 3.2.85

$$\begin{aligned}
x0 +^N y0 &= (x +^N y) \cdot 0 \\
x1 +^N y0 &= (x +^N y) \cdot 1 \\
x1 +^N y1 &= {\gtrdot}\mathsf{succ}^N(x +^N y) \cdot 0
\end{aligned}$$

With the help of this theorem, we can now prove the following important properties of $+^N$ with a version of Derived Rule 3.2.6 that concatenates bits to the right instead of to the left.

THEOREM 3.2.23

1. $x +^N \mathsf{succ}^N(y) =^N \mathsf{succ}^N(x +^N y)$
2. $x +^N (y +^N z) =^N (x +^N y) +^N z$
3. $y <^N z \leftrightarrow x +^N y <^N x +^N z$
4. $x =^N y \wedge z =^N w \rightarrow x +^N z =^N y +^N w$
5. $x =^N y \wedge z <^N w \rightarrow x +^N z <^N y +^N w$
6. $x <^N y \wedge z <^N w \rightarrow x +^N z <^N y +^N w$

On iterated sums

The last functions we need to define are iterated sums, defined as in L_1 using Buss's "carry-save" technique.

DEFINITION 3.2.30

$$\begin{aligned}
\mathsf{CScar}_3 &= \ell\mathrm{CRN}_3\big[\lambda x_1x_2x_3\,.\,(({}^\backprime x_1 \wedge^B {}^\backprime x_2) \vee^B ({}^\backprime x_2 \wedge^B {}^\backprime x_3) \vee^B ({}^\backprime x_3 \wedge^B {}^\backprime x_1))\big] \\
\mathsf{CSadd}_3 &= \big[\lambda x_1x_2x_3\,.\,\mathsf{xor}_3^B(0x_1, 0x_2, 0x_3)\big] \\
\mathsf{CScar} &= \big[\lambda x_1x_2x_3x_4\,.\,\mathsf{CScar}_3\big(\mathsf{CScar}_3(x_1,x_2,x_3)\cdot 0, \mathsf{CSadd}_3(x_1,x_2,x_3), 0x_4\big)\cdot 0\big] \\
\mathsf{CSadd} &= \big[\lambda x_1x_2x_3x_4\,.\,\mathsf{CSadd}_3\big(\mathsf{CScar}_3(x_1,x_2,x_3)\cdot 0, \mathsf{CSadd}_3(x_1,x_2,x_3), 0x_4\big)\big]
\end{aligned}$$

The following properties are a direct consequence of these definitions.

CLAIM 3.2.86

$$\mathsf{CScar}_3(x0, y0, z0) = \mathsf{CScar}_3(x,y,z)\cdot 0$$
$$\mathsf{CScar}_3(x1, y0, z0) = \mathsf{CScar}_3(x0, y1, z0) = \mathsf{CScar}_3(x1, y0, z1) = \mathsf{CScar}_3(x,y,z)\cdot 0$$
$$\mathsf{CScar}_3(x0, y1, z1) = \mathsf{CScar}_3(x1, y0, z1) = \mathsf{CScar}_3(x1, y1, z0) = \mathsf{Cscar}_3(x,y,z)\cdot 1$$
$$\mathsf{CScar}_3(x1, y1, z1) = \mathsf{CScar}_3(x,y,z)\cdot 1$$
$${}_0\mathsf{CScar}_3(x,y,z)\cdot 0 = {}_0\mathsf{CSadd}_3(x,y,z) = 0\cdot {}_0\mathsf{max}_3^L(x,y,z) = 0\cdot \mathsf{max}_3^L({}_0x, {}_0y, {}_0z)$$
$$0\cdot \mathsf{max}_3^L({}_0x, {}_0y, {}_0z) = \mathsf{CScar}_3({}_0x, {}_0y, {}_0z)\cdot 0 = \mathsf{CSadd}_3({}_0x, {}_0y, {}_0z)$$
$${}_0\mathsf{CScar}(x,y,z,w) = {}_0\mathsf{CSadd}(x,y,z,w) = 00\cdot {}_0\mathsf{max}_4^L(x,y,z,w) = 00\cdot \mathsf{max}_4^L({}_0x, {}_0y, {}_0z, {}_0w)$$
$$00\cdot \mathsf{max}_4^L({}_0x, {}_0y, {}_0z, {}_0w) = \mathsf{CScar}({}_0x, {}_0y, {}_0z, {}_0w) = \mathsf{CSadd}({}_0x, {}_0y, {}_0z, {}_0w)$$

We can now prove one main lemma and one main theorem about the "carry-save" addition functions.

Lemma 3.2.87

$$(\mathsf{CScar}_3(\mathsf{succ}^N(x), y, z) \cdot 0) +^N \mathsf{CSadd}_3(\mathsf{succ}^N(x), y, z) \\ =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(x, y, z) \cdot 0) +^N \mathsf{CSadd}_3(x, y, z)\big)$$

Theorem 3.2.24 $\mathsf{CScar}(x, y, z, w) +^N \mathsf{CSadd}(x, y, z, w) =^N x +^N y +^N z +^N w$

Finally, we can define the function "sum", that adds all the bits of its argument, as in L_1.

Definition 3.2.31

$$\begin{aligned} \mathsf{CARADD} &= \mathrm{STRN}\big[\lambda x\,.\,{}_0x \cdot x, \lambda x v_\ell v_r\,.\,\mathsf{CScar}(v_\ell◀, ▶v_\ell, v_r◀, ▶v_r) \cdot \mathsf{CSadd}(v_\ell◀, ▶v_\ell, v_r◀, ▶v_r)\big] \\ \mathsf{CAR} &= \big[\lambda x.\,\mathsf{CARADD}(x)◀\big] \qquad \mathsf{ADD} = \big[\lambda x.\,▶\mathsf{CARADD}(x)\big] \\ \mathsf{sum} &= \big[\lambda x.\,\mathsf{CAR}(x) +^N \mathsf{ADD}(x)\big] \end{aligned}$$

The following basic properties of CARADD will be used to prove results about sum and can be proved easily from previous theorems.

Claim 3.2.88

1. $\mathsf{CARADD}({}_0x) =^N 0$
2. $\mathsf{sum}({}_0x) = \mathsf{CAR}({}_0x) +^N \mathsf{ADD}({}_0x) =^N 0 +^N 0 =^N 0$

Theorem 3.2.25 $\mathsf{sum}(x) =^N \mathsf{sum}(x◀) +^N \mathsf{sum}(▶x)$

From this theorem, it is possible to prove that $\mathsf{sum}(xy) =^N \mathsf{sum}(x) +^N \mathsf{sum}(y)$ with a sequence of lemmas and theorems similar to the ones used to show that $\mathsf{AND}(xy) = \mathsf{AND}(x) \wedge^B \mathsf{AND}(y)$. In particular, we have that $\mathsf{sum}(x0) =^N \mathsf{sum}(x) +^N 0 =^N \mathsf{sum}(x)$ and $\mathsf{sum}(x1) =^N \mathsf{sum}(x) +^N 1 =^N \mathsf{succ}^N(\mathsf{sum}(x))$.

Simple proofs by NIND now show the following theorem.

Theorem 3.2.26 $\mathsf{sum}(x) \leq^N \mathsf{sum}({}_1x) =^N |x|$

3.3 Proving the pigeonhole principle in T_1

In this section, we will be working with the following form of the pigeonhole principle, denoted $\mathrm{PHP}_n(f)$ (or simply PHP when n and f are clear from the context): "no map $f : [n+1] \to [n]$ is injective", or equivalently "if f is a map from $[n+1]$ to $[n]$, then there exist $i \neq j \in [n+1]$ such that $f(i) = f(j)$". (Note that we are using the common notation "$[n]$" to represent the set $\{1, 2, \dots, n\}$, for any positive integer n, and in what follows, we will use the term *map* to mean a (possibly) multi-valued function.)

Informally, the proof of PHP goes as follows: Assume for a contradiction that f is a map from $[n+1]$ to $[n]$ and that f is injective (*i.e.*, for every $i \neq j \in [n+1]$, $f(i) \neq f(j)$). Define

$$\text{count}(k, \ell) = \big|\{x \in [\ell] : f(y) = x \text{ for some } y \in [k]\}\big|,$$

i.e., count(k, ℓ) is the number of elements in $[\ell]$ mapped onto from elements in $[k]$ by f. Then, the following facts are easy to prove.

1. count$(n+1, \ell) \leq \ell$ for any $1 \leq \ell \leq n$ (since there are ℓ elements in $[\ell]$).
2. count$(1, n) \geq 1$ (since $f(1) \in [n]$).
3. count$(k+1, n) >$ count(k, n) for $1 \leq k \leq n$ (since $f(k+1)$ must be different from $f(1), \dots, f(k)$ by the assumption that f is injective).

Combining facts 2 and 3, we get that count$(k, n) \geq k$ for all $1 \leq k \leq n+1$. But then, $n+1 \leq$ count$(n+1, n) \leq n$, *i.e.*, $n+1 \leq n$, which is a contradiction. Hence, PHP is true.

In the rest of this section, we will show how the informal proof given above can be formalized in T_1, in a top-down manner. Also, we adopt the following notational convention: when a function is defined through an auxiliary function that is of no interest in itself, the name of the auxiliary function will consist of the function's name spelled backwards (*e.g.*, we will define below a function "map" in terms of an auxiliary function "pam").

3.3.1 Representation of PHP in T_1

The first step in the formalization will be to use the machinery given above to write down L_1-functions that define PHP. Formally, PHP = PHP$_n(f)$ depends on two parameters: n and f; moreover, given n, f can be described by an $[n \times (n+1)]$ binary array whose (i, j)-th entry is equal to 1 if $i = f(j)$ and 0 otherwise, *i.e.*, each row corresponds to one hole and each column to one pigeon. This is essentially the representation we will use to encode the problem.

More precisely, given n, every bit string a can be seen as encoding an $[n \times (n+1)]$ binary array (and therefore a partial map $f_a^n : [n+1] \to [n]$) by either padding a on the right with 0's or chopping off enough bits from the right of a so that its length is $n \cdot (n+1)$, and then reading the array in row-major order, so that the first $n+1$ bits of a (from the left) represent the first row of the array, and so on. For example, the string 1000110010 represents at the same time

$$\text{the } [2 \times 3] \text{ binary array } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad \text{the } [3 \times 4] \text{ binary array } \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \qquad \text{etc.}$$

Then, every such $[n \times (n+1)]$ binary array a represents a partial map $f_a^n : [n+1] \to [n]$.

Now, given a bit string N such that $n = |\mathrm{N}|$, we can define a function $\mathsf{adj}(a, \mathrm{N})$ that "adjusts" the length of the bit string a so that $|\mathsf{adj}(a, \mathrm{N})| = |\mathrm{N}| \cdot (|\mathrm{N}| + 1)$, *i.e.*, $\mathsf{adj}(a, \mathrm{N})$ is exactly the $[n \times (n+1)]$ binary matrix encoded by a, written out in row-major order:

$$\mathsf{adj}(a, \mathrm{N}) = a \lhd ((\mathrm{N}1 \# \mathrm{N}) \rhd a) \cdot {}_0(a \rhd (\mathrm{N}1 \# \mathrm{N})).$$

Next, given a column number K and a row number L in unary, we can easily define a function entry that extracts a single entry (bit) of the matrix:

$$\mathsf{entry}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}) = \big(((\mathrm{N}1 \# \mathrm{L}{<}) \cdot \mathrm{K}{<}) \rhd \mathsf{adj}(a, \mathrm{N})\big)'.$$

Note that the value returned by this function is meaningless unless $1 \le |\mathrm{K}| = k \le n+1$ and $1 \le |\mathrm{L}| = \ell \le n$. In order to simplify the presentation, we will implicitly assume that k and ℓ fall within this range for the rest of the section, where $k = |\mathrm{K}|$ and $\ell = |\mathrm{L}|$ by convention (*i.e.*, functions are implicitly defined by cases to be equal to ε for values outside the meaningful range).

Once we have the function entry, it is easy to define functions col and row that extract columns or rows of the matrix, by CRN:

$$\begin{aligned}
\mathsf{loc}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}) &= \mathsf{loc}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}{<}) \cdot \mathsf{entry}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}) \quad (\text{for } \mathrm{L} \neq \varepsilon)\\
\mathsf{col}(a, \mathrm{N}, \mathrm{K}) &= \mathsf{loc}(a, \mathrm{N}, \mathrm{K}, \mathrm{N});\\
\mathsf{wor}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}) &= \mathsf{wor}(a, \mathrm{N}, \mathrm{K}{<}, \mathrm{L}) \cdot \mathsf{entry}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}) \quad (\text{for } \mathrm{K} \neq \varepsilon)\\
\mathsf{row}(a, \mathrm{N}, \mathrm{L}) &= \mathsf{wor}(a, \mathrm{N}, \mathrm{N}1, \mathrm{L}).
\end{aligned}$$

So that $\mathsf{col}(a, \mathrm{N}, \mathrm{K})$ is the k-th column of a and $\mathsf{row}(a, \mathrm{N}, \mathrm{L})$ is the ℓ-th row of a. Moreover, it is easy to prove in T_1 that

$$\begin{aligned}
\mathsf{lc}(\mathsf{col}(a, \mathrm{N}, \mathrm{K}), \mathrm{L}) &= \mathsf{loc}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}),\\
\mathsf{lc}(\mathsf{row}(a, \mathrm{N}, \mathrm{L}), \mathrm{K}) &= \mathsf{wor}(a, \mathrm{N}, \mathrm{K}, \mathrm{L}),
\end{aligned}$$

directly from the properties of CRN.

Using these functions, we can now define two functions needed to represent $\mathrm{PHP}_n(f)$:

$$\mathsf{map}(a, \mathrm{N}) = \begin{cases} 1 & \text{if } f_a^n \text{ is a map, } \textit{i.e.}, \text{ every column of } a \text{ contains at least one } 1,\\ 0 & \text{otherwise;} \end{cases}$$

$$\mathsf{inj}(a, \mathrm{N}) = \begin{cases} 1 & \text{if } f_a^n \text{ is injective, } \textit{i.e.}, \text{ every row of } a \text{ contains at most one } 1,\\ 0 & \text{otherwise.} \end{cases}$$

To compute map (resp. inj), we will first define a function pam (resp. jni) that returns a bit string with one bit for each column (resp. row) of a indicating whether the constraint is satisfied or not for that column (resp. row); then, we simply take the conjunction of all the bits to get the answer:

$$\begin{aligned}
\mathsf{pam}(a,\mathrm{N},x) &= \mathsf{pam}(a,\mathrm{N},x{\lessdot})\cdot\mathsf{OR}\big(\mathsf{col}(a,\mathrm{N},x)\big) \quad (\text{for } x\neq\varepsilon)\\
\mathsf{map}(a,\mathrm{N}) &= \mathsf{AND}\big(\mathsf{pam}(a,\mathrm{N},\mathrm{N}1)\big),\\
\mathsf{jni}(a,\mathrm{N},x) &= \mathsf{jni}(a,\mathrm{N},x{\lessdot})\cdot\neg^{B}\mathsf{OR}\big(\mathsf{delfirst}_1(\mathsf{row}(a,\mathrm{N},x))\big) \quad (\text{for } x\neq\varepsilon)\\
\mathsf{inj}(a,\mathrm{N}) &= \mathsf{AND}\big(\mathsf{jni}(a,\mathrm{N},\mathrm{N})\big).
\end{aligned}$$

Finally, we can easily define a function that represents PHP:

$$\mathsf{php}(a,\mathrm{N}) = \mathsf{map}(a,\mathrm{N}) \rightarrow^{B} \neg^{B}\mathsf{inj}(a,\mathrm{N}).$$

3.3.2 The T_1-proof of PHP

First, let us define the function $\mathsf{count}(a,\mathrm{N},\mathrm{K},\mathrm{L})$, which returns the number of elements from $[\ell]$ that are mapped onto by elements from $[k]$ according to f_a^n. We do this by first defining $\mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L})$, which returns a string of ℓ bits, one for each of the first ℓ rows of a, where bit j is set to 1 iff row j contains at least one 1 in the first k columns:

$$\mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L}) = \mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L}{\lessdot})\cdot\mathsf{OR}\big(\mathsf{wor}(a,\mathrm{N},\mathrm{K},\mathrm{L})\big) \quad (\text{for } \mathrm{L}\neq\varepsilon)$$

Then, $\mathsf{count}(a,\mathrm{N},\mathrm{K},\mathrm{L}) = \mathsf{sum}\big(\mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L})\big)$. We can depict the situation as follows, where we have represented the submatrix of a consisting of the first k columns for each of the first ℓ rows, and where the value of $\mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L})$ can be read bit by bit, one for each row:

$$\begin{array}{cl}
 & \overbrace{\hspace{9em}}^{k} \hspace{5em} \mathsf{tnuoc}(a,\mathrm{N},\mathrm{K},\mathrm{L}) :\\
\ell\left\{\begin{array}{ccccc} 1&0&0&\cdots&1\\ 0&0&0&\cdots&0\\ \vdots&\vdots&\vdots&\vdots&\vdots\\ 0&0&0&\cdots&1 \end{array}\right. &
\begin{array}{cccl} \longrightarrow & 1 & = & \mathsf{OR}(\mathsf{wor}(a,\mathrm{N},\mathrm{K},1))\\ \longrightarrow & 0 & = & \mathsf{OR}(\mathsf{wor}(a,\mathrm{N},\mathrm{K},11))\\ \vdots & \vdots & \vdots & \quad\vdots\\ \longrightarrow & 1 & = & \mathsf{OR}(\mathsf{wor}(a,\mathrm{N},\mathrm{K},\mathrm{L})) \end{array}
\end{array}$$

Now, we give a formalization in T_1 of the high-level proof of PHP outlined at the beginning of this section. To make the notation easier to read, all theorems are conditional to the fact that k and ℓ are within meaningful range.

Recall the general outline of the proof: under the assumption that $\neg^{B}\mathsf{php}(a,\mathrm{N})$, *i.e.*, that $\mathsf{map}(a,\mathrm{N})$ and $\mathsf{inj}(a,\mathrm{N})$, it is possible to prove the following two facts.

Fact 3.3.1 $\quad \mathsf{count}(a,\mathrm{N},\mathrm{N}1,\mathrm{L}) \leq^{N} \ell \quad$ (where $\ell = |\mathrm{L}|$, by convention)

Fact 3.3.2 $k \le^N \mathsf{count}(a, \mathrm{N}, \mathrm{K}, \mathrm{N})$ (where $k = |\mathrm{K}|$, by convention)

Then, we get that $|\mathrm{N}1| \le^N \mathsf{count}(a, \mathrm{N}, \mathrm{N}1, \mathrm{N}) \le^N |\mathrm{N}|$, so that $|\mathrm{N}1| \le^N |\mathrm{N}|$ (by transitivity of $\le^N$). But since we know that $\neg^B\big(|x1| \le^N |x|\big)$, we get $\mathsf{php}(a, \mathrm{N})$ by contradiction.

Now, we can prove fact 3.3.1:

$$\mathsf{count}(a, \mathrm{N}, \mathrm{N}1, \mathrm{L}) = \mathsf{sum}\big(\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{N}1, \mathrm{L})\big) \le^N \big|\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{N}1, \mathrm{L})\big| \le^N |\mathrm{L}| = \ell$$

since $\mathsf{sum}(x) \le^N |x|$ for any string x and $\big|r\mathrm{CRN}[h](x, \vec{y})\big| = |x|$ for any L_1-function h and any strings x, $\vec{y}$.

To prove fact 3.3.2, we will first show that it is possible to prove the following two facts (corresponding to facts 2 and 3 in the informal proof).

Fact 3.3.3 $\mathsf{count}(a, \mathrm{N}, \varepsilon, \mathrm{N}) =^N 0$

Fact 3.3.4 $\mathsf{count}(a, \mathrm{N}, \mathrm{K}, \mathrm{N}) >^N \mathsf{count}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N})$

Then, we can use NIND to show that $\mathsf{count}(a, \mathrm{N}, \mathrm{K}, \mathrm{N}) \ge^N |\mathrm{K}|$: $\mathsf{count}(a, \mathrm{N}, \varepsilon, \mathrm{N}) \ge^N |\varepsilon|$ by fact 3.3.3 and $\mathsf{count}(a, \mathrm{N}, \mathrm{K}, \mathrm{N}) >^N \mathsf{count}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N}) \ge^N |\mathrm{K}{\lessdot}|$ by fact 3.3.4 and the induction hypothesis, so that $\mathsf{count}(a, \mathrm{N}, \mathrm{K}, \mathrm{N}) \ge^N |\mathrm{K}|$.

Next, to prove fact 3.3.3, we use NIND together with the fact that $\mathsf{wor}(a, \mathrm{N}, \varepsilon, \mathrm{L}) = \varepsilon$ (by definition) to conclude that $\mathsf{tnuoc}(a, \mathrm{N}, \varepsilon, \mathrm{L}) = {}_0\mathrm{L}$: $\mathsf{tnuoc}(a, \mathrm{N}, \varepsilon, \varepsilon) = \varepsilon = {}_0\varepsilon$, and assuming that $\mathsf{tnuoc}(a, \mathrm{N}, \varepsilon, \mathrm{L}{\lessdot}) = {}_0(\mathrm{L}{\lessdot})$, then $\mathsf{tnuoc}(a, \mathrm{N}, \varepsilon, \mathrm{L}) = \mathsf{tnuoc}(a, \mathrm{N}, \varepsilon, \mathrm{L}{\lessdot}) \cdot \mathsf{OR}\big(\mathsf{wor}(a, \mathrm{N}, \varepsilon, \mathrm{L})\big) = {}_0(\mathrm{L}{\lessdot}) \cdot \mathsf{OR}(\varepsilon) = {}_0(\mathrm{L}{\lessdot}) \cdot 0 = {}_0(\mathrm{L})$. Finally, we use the fact that $\mathsf{sum}({}_0x) =^N 0$ for any string x to conclude that fact 3.3.3 holds.

To prove fact 3.3.4, we have to show that $\mathsf{sum}\big(\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}, \mathrm{N})\big) >^N \mathsf{sum}\big(\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N})\big)$. Intuitively, this will be true iff $\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}, \mathrm{N})$ contains more 1's than $\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N})$. Formally, for any two strings x and y, the term $\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(x), y)\big)$ expresses the fact that y has a 1 in every position where x has a 1, and the term $\mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(x), y)\big)$ expresses the fact that there is a position where x has a 0 but y has a 1. Now, since we can prove that $\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(x), y)\big) \wedge^B \mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(x), y)\big) \to^B \mathsf{sum}(x) <^N \mathsf{sum}(y)$ (the proof is given in Appendix A), we only need to show the following facts to complete the proof of fact 3.3.4.

Fact 3.3.5 $\mathsf{AND}\Big(\mathsf{or}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N})), \mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}, \mathrm{N})\big)\Big)$

Fact 3.3.6 $\mathsf{OR}\Big(\mathsf{and}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}{\lessdot}, \mathrm{N})), \mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}, \mathrm{N})\big)\Big)$

It is relatively easy to prove fact 3.3.5 by NIND on the last argument: $\mathsf{tnuoc}(a, \mathrm{N}, \mathrm{K}{\lessdot}, 1) =$

$\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,1))$ and $\mathsf{tnuoc}(a,\textsc{n},\textsc{k},1) = \mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k},1))$ so

$$\begin{aligned}
&\mathsf{AND}\big(\mathsf{or}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a,\textsc{n},\textsc{k}\lessdot,\varepsilon)),\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\varepsilon)\big)\big)\\
&\quad= \neg^B\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,1)) \vee^B \mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k},1))\\
&\quad= \neg^B\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})) \vee^B \mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})\cdot\mathsf{entry}(a,\textsc{n},\textsc{k},\textsc{l})\big)\\
&\quad= \neg^B\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})) \vee^B \mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})) \vee^B \mathsf{entry}(a,\textsc{n},\textsc{k},\textsc{l})\\
&\quad= 1 \vee^B \mathsf{entry}(a,\textsc{n},\textsc{k},\textsc{l}) = 1.
\end{aligned}$$

Also, by the definition of tnuoc, we get that

$$\begin{aligned}
&\mathsf{AND}\big(\mathsf{or}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})),\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{l})\big)\big)\\
&\quad= \mathsf{AND}\Big(\mathsf{or}_2^B\Big(\mathsf{not}^B\big(\mathsf{tnuoc}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l}\lessdot)\cdot\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l}))\big),\\
&\qquad\qquad \mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{l}\lessdot)\cdot\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l}))\Big)\Big)\\
&\quad= \mathsf{AND}\big(\mathsf{or}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l}\lessdot)),\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{l}\lessdot)\big)\big)\\
&\qquad \wedge^B \big(\neg^B\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})) \vee^B \mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l}))\big)\\
&\quad= 1 \wedge^B \big(\neg^B\mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})) \vee^B \mathsf{OR}(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l}))\big) = 1
\end{aligned}$$

(where the third equality holds by the induction hypothesis).

The proof of fact 3.3.6 is the most involved so far. First, by the definition of tnuoc and properties of CRN, we know that

$$\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{n}) = \mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{l}\lessdot)\cdot\mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l})\big)\cdot\big(\textsc{l}\rhd\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{n})\big).$$

Because $\mathsf{OR}(xy) = \mathsf{OR}(x)\vee^B\mathsf{OR}(y)$ and $\mathsf{and}_2^B(xy,wz) = \mathsf{and}_2^B(x,w)\cdot\mathsf{and}_2^B(y,z)$ when $|x| = |w|$ and $|y| = |z|$, we get easy proofs in T_1 that

$$\begin{gathered}
\neg^B\mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})\big) \wedge^B \mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l})\big)\rightarrow^B\\
\mathsf{OR}\big(\mathsf{and}_2^B\big(\mathsf{not}^B(\mathsf{tnuoc}(a,\textsc{n},\textsc{k}\lessdot,\textsc{n})),\mathsf{tnuoc}(a,\textsc{n},\textsc{k},\textsc{n})\big)\big).
\end{gathered}$$

Hence, we can prove fact 3.3.6 by showing that there must exist some value $\textsc{l}$ for which $\neg^B\mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l})\big) \wedge^B \mathsf{OR}\big(\mathsf{wor}(a,\textsc{n},\textsc{k},\textsc{l})\big)$. Now, we can prove that $\neg^B\mathsf{OR}(\mathsf{delfirst}_1(x)) \wedge^B \mathsf{lb}(x,y) \rightarrow^B \neg^B\mathsf{OR}(\mathsf{lc}(x,y\lessdot))$ (see Appendix A), and since $\mathsf{wor}(a,\textsc{n},\textsc{k}\lessdot,\textsc{l}) = \mathsf{lc}(\mathsf{row}(a,\textsc{n},\textsc{l}),\textsc{k}\lessdot)$ and $\neg^B\mathsf{OR}\big(\mathsf{delfirst}_1(\mathsf{row}(a,\textsc{n},\textsc{l}))\big)$ by the assumption that $\mathsf{inj}(a,\textsc{n})$, we only need to show that there is some $\textsc{l}$ for which $\mathsf{lb}\big(\mathsf{row}(a,\textsc{n},\textsc{l}),\textsc{k}\big)$, which is equivalent to showing that there is a value of $\textsc{l}$ for which $\mathsf{entry}(a,\textsc{n},\textsc{k},\textsc{l})$.

Unfortunately, we do not have quantifiers to reason with so to show the existence of $\textsc{l}$, we have to construct it explicitly, *i.e.*, to define a function $\mathsf{pos}(a,\textsc{n},\textsc{k})$ that gives the value of $\textsc{l}$. Because all functions definable in T_1 are length-determined, pos will have to return a *bitmask* to the position of $\textsc{l}$, and this bitmask cannot be used directly with the current definition of row to prove what we need. So, we will define an alternate function mrow whose last argument is a bitmask instead of a unary string and for which we can show $\mathsf{row}(a, \textsc{n}, \textsc{l}) = \mathsf{mrow}\big(a, \textsc{n}, (1 \cdot {}_0(\textsc{l} \rhd \textsc{n}1))^{\lessdot}\big)$:

$$\begin{aligned}\mathsf{mwor}(a, \textsc{n}, \textsc{k}, m) &= \mathsf{mwor}(a, \textsc{n}, \textsc{k}^{\lessdot}, m) \cdot \mathsf{maskbit}^N(\mathsf{col}(a, \textsc{n}, \textsc{k}), m) \quad (\text{for } \textsc{k} \neq \varepsilon)\\ \mathsf{mrow}(a, \textsc{n}, m) &= \mathsf{mwor}(a, \textsc{n}, \textsc{n}1, m)\end{aligned}$$

With this definition, pos can easily be defined as $\mathsf{pos}(a, \textsc{n}, \textsc{k}) = \mathsf{first}_1(\mathsf{col}(a, \textsc{n}, \textsc{k}))$. By the assumption that $\mathsf{map}(a, \textsc{n})$, we know that $\mathsf{OR}(\mathsf{col}(a, \textsc{n}, \textsc{k}))$ and since we can prove that $\mathsf{OR}(x) \leftrightarrow^B \mathsf{maskbit}^N(x, \mathsf{first}_1(x))$ in T_1 (see Appendix A), we get that $\mathsf{maskbit}^N\big(\mathsf{col}(a, \textsc{n}, \textsc{k}), \mathsf{pos}(a, \textsc{n}, \textsc{k})\big)$, which implies immediately that $\mathsf{lb}\big(\mathsf{mrow}(a, \textsc{n}, \mathsf{pos}(a, \textsc{n}, \textsc{k})), \textsc{k}\big)$.

Now, because row and mrow are both defined by CRN on the same parameter $\textsc{k}$, it is sufficient to show that $\mathsf{entry}(a, \textsc{n}, \textsc{k}, \textsc{l}) = \mathsf{maskbit}^N\big(\mathsf{col}(a, \textsc{n}, \textsc{k}), (1 \cdot {}_0(\textsc{l} \rhd \textsc{n}1))^{\lessdot}\big)$ in order to prove that $\mathsf{row}(a, \textsc{n}, \textsc{l}) = \mathsf{mrow}\big(a, \textsc{n}, (1 \cdot {}_0(\textsc{l} \rhd \textsc{n}1))^{\lessdot}\big)$. And because we can prove in T_1 that $\mathsf{lb}^B(x, y) = \mathsf{maskbit}^N\big(x, (1 \cdot {}_0(y \rhd x1))^{\lessdot}\big)$ (see Appendix A), this is equivalent to showing that $\mathsf{entry}(a, \textsc{n}, \textsc{k}, \textsc{l}) = \mathsf{lb}(\mathsf{col}(a, \textsc{n}, \textsc{k}), \textsc{l}) = \mathsf{loc}(a, \textsc{n}, \textsc{k}, \textsc{l})'$, a fact which is immediate by the definition of loc.

Finally, we can redefine tnuoc using mwor instead of wor, as follows:

$$\mathsf{mtnuoc}(a, \textsc{n}, \textsc{k}, \textsc{l}) = \mathsf{mtnuoc}(a, \textsc{n}, \textsc{k}, \textsc{l}^{\lessdot}) \cdot \mathsf{OR}\big(\mathsf{mwor}\big(a, \textsc{n}, \textsc{k}, (1 \cdot {}_0(\textsc{l} \rhd \textsc{n}1))^{\lessdot}\big)\big) \quad (\text{for } \textsc{l} \neq \varepsilon)$$

and using the reasoning given above, it is possible to prove that

$$\mathsf{OR}\big(\mathsf{and}_2^B\big(\mathsf{not}^B(\mathsf{mtnuoc}(a, \textsc{n}, \textsc{k}^{\lessdot}, \textsc{n})), \mathsf{mtnuoc}(a, \textsc{n}, \textsc{k}, \textsc{n})\big)\big).$$

Moreover, because of the equivalence between wor and mwor given above, this implies the corresponding result for tnuoc, which completes the proof.

REMARK 3.3.1 Based on the T_1-proof of the pigeonhole principle just given, it should be possible to prove other, similar combinatorial statements in T_1. One example is Tutte's theorem, which states that a graph has no perfect matching iff it satisfies a certain simple form of decomposition. This would give an alternative proof that "perfect matching" tautologies have short $\mathcal{F}$-proofs, and maybe provide a more precise estimate of the size of these proofs by the results of Chapter 4. (The "perfect matching" tautologies were first discussed in a paper by Impagliazzo, Pitassi, and Urquhart [21], where it was shown that they have polysize $\mathcal{F}$-proofs—note that those proofs were *non*-uniform, unlike the proofs we would obtain through T_1.)

Chapter 4

Theorems of T_1 Have Polysize $\mathcal{F}$-proofs

For every term t of T_1 with free variables $x_1, \ldots, x_k$, we define a *length function* $\mathrm{len}_t(m_1, \ldots, m_k)$ that gives the exact length of t as a function of the lengths of $x_1, \ldots, x_k$ (this function is well-defined because functions in L_1 are length-determined). Then, we define a family of propositional *term formulas* $\langle t\rangle_1^{\vec{m}}, \ldots, \langle t\rangle_{\mathrm{len}_t(\vec{m})}^{\vec{m}}$ that describe the bits of t in terms of the bits of $x_1, \ldots, x_k$ (where $\langle t\rangle_1^{\vec{m}}$ describes the leftmost bit of t), *i.e.*, given any truth-value assignment to the atoms representing the bits of $x_1, \ldots, x_n$, the truth value of $\langle t\rangle_i^{\vec{m}}$ represents the correct value for bit number i of term t. Finally, for any formula A of T_1, we define a family of *propositional translations* $[\![A]\!]^{\vec{m}}$, where $\vec{m}$ lists the lengths of all free variables in A, and show that there are short $\mathcal{F}$-proofs of $[\![A]\!]^{\vec{m}}$ whenever A is a theorem of T_1.

4.1 Length functions

The length functions are defined inductively as follows (where "sg" is the signum function).

$$\mathrm{len}_x(m) = m$$

$$\mathrm{len}_{f(t_1,\ldots,t_k)}(\vec{m}) = \mathrm{len}_{f(x_1,\ldots,x_k)}\big(\mathrm{len}_{t_1}(\vec{m}_1), \ldots, \mathrm{len}_{t_k}(\vec{m}_k)\big)^1$$

$$(\text{ where } x_1, \ldots, x_k \text{ occur in none of } t_1, \ldots, t_k \text{ })$$

$$\mathrm{len}_\varepsilon = 0 \qquad \mathrm{len}_0 = 1 \qquad \mathrm{len}_1 = 1$$

$$\mathrm{len}_{0x}(m) = m \qquad \mathrm{len}_{1x}(m) = m$$

$$\mathrm{len}_{x\blacktriangleleft}(m) = \lfloor m/2 \rfloor \qquad \mathrm{len}_{\blacktriangleright x}(m) = \lceil m/2 \rceil$$

$$\mathrm{len}_{x\lhd y}(m, n) = m \dot{-} n \qquad \mathrm{len}_{y\rhd x}(n, m) = m \dot{-} n$$

[1]Where $\vec{m}_i$ represents the lengths of the variables that occur in t_i.

$$\begin{aligned}
\mathrm{len}_{x\cdot y}(m,n) &= m+n\\
\mathrm{len}_{x?(y,z_0,z_1)}(m,n,p_0,p_1) &= \textbf{if } m=0 \textbf{ then } n \textbf{ else } \max\{p_0,p_1\}\\
\mathrm{len}_{[\lambda\vec{x}.t](\vec{y})}(\vec{n}) &= \mathrm{len}_{t[\vec{y}/\vec{x}]}(\vec{n}')^{2}\\
\mathrm{len}_{\ell\mathrm{CRN}[h](x,\vec{y})}(m,\vec{n}) &= m \qquad\qquad \mathrm{len}_{r\mathrm{CRN}[h](x,\vec{y})}(m,\vec{n}) = m\\
\mathrm{len}_{\mathrm{TRN}[g,h,h_\ell,h_r](x,z,\vec{y})}(m,p,\vec{n}) &= \textbf{if } m\le 1 \textbf{ then } \mathrm{len}_{g(x,z,\vec{y})}(m,p,\vec{n}) \textbf{ else}\\
&\quad \mathrm{len}_{h(x,z,\vec{y},v_0,v_1)}\Big(m,p,\vec{n},\mathrm{len}_{\mathrm{TRN}[g,h,h_\ell,h_r](x\blacktriangleleft,h_\ell(z),\vec{y})}(m,p,\vec{n}),\\
&\qquad \mathrm{len}_{\mathrm{TRN}[g,h,h_\ell,h_r](\blacktriangleright x,h_r(z),\vec{y})}(m,p,\vec{n})\Big)
\end{aligned}$$

4.2 Term formulas

To every variable x of T_1 are associated propositional atoms $\langle x\rangle_1^m,\ldots,\langle x\rangle_m^m$. For other terms of T_1, the term formulas are defined inductively as follows. (When subscript i is used without specifying its range in the definition of $\langle t\rangle_i^{\vec{m}}$, it is implicitly assumed that $1\le i\le \mathrm{len}_t(\vec{m})$.)

$$\langle f(t_1,\ldots,t_k)\rangle_i^{\vec{m}} = \langle f(x_1,\ldots,x_k)\rangle_i^{\mathrm{len}_{t_1}(\vec{m}_1),\ldots,\mathrm{len}_{t_k}(\vec{m}_k)}\Big[\langle t_j\rangle_{i_j}^{\vec{m}_j}/\langle x_j\rangle_{i_j}^{\mathrm{len}_{t_j}(\vec{m}_j)}\Big]_{\substack{1\le j\le k\\ 1\le i_j\le \mathrm{len}_{t_j}(\vec{m}_j)}}$$

$$(\text{ where } x_1,\ldots,x_k \text{ occur in none of } t_1,\ldots,t_k\ )$$

$$\begin{aligned}
\langle 0\rangle_1 &= \bot & \langle 1\rangle_1 &= \top\\
\langle {}_0x\rangle_i^m &= \bot & \langle {}_1x\rangle_i^m &= \top\\
\langle x\blacktriangleleft\rangle_i^m &= \langle x\rangle_i^m & \langle\blacktriangleright x\rangle_i^m &= \langle x\rangle_{i+\lfloor m/2\rfloor}^m\\
\langle x\lhd y\rangle_i^{m,n} &= \langle x\rangle_i^m & \langle y\rhd x\rangle_i^{m,n} &= \langle x\rangle_{i+n}^m
\end{aligned}$$

$$\langle x\cdot y\rangle_i^{m,n} = \begin{cases}\langle x\rangle_i^m & \text{if } i\le m\\ \langle y\rangle_{i-m}^n & \text{if } m<i\end{cases}$$

$$\langle x\ ?\ (y,z_0,z_1)\rangle_i^{m,n,p_0,p_1} = \begin{cases}\langle y\rangle_i^n & \text{if } m=0\\ \big(\neg\langle x\rangle_m^m\wedge\langle{}_0(z_1\lhd z_0)\cdot z_0\rangle_i^{p_0,p_1}\big) & \\ \vee\big(\langle x\rangle_m^m\wedge\langle{}_0(z_0\lhd z_1)\cdot z_1\rangle_i^{p_0,p_1}\big) & \text{if } m>0\end{cases}$$

$$\begin{aligned}
\langle[\lambda\vec{x}.t](\vec{y})\rangle_i^{\vec{n}} &= \langle t[\vec{y}/\vec{x}]\rangle_i^{\vec{n}'}\\
\langle\ell\mathrm{CRN}[h](x,\vec{y})\rangle_i^{m,\vec{n}} &= \langle h(z,\vec{y})\cdot 0\rangle_1^{m-i+1,\vec{n}}\big[\langle x\rangle_{j+i-1}^m/\langle z\rangle_j^{m-i+1}\big]_{1\le j\le m-i+1}\\
&\quad (\text{ where } z \text{ does not occur in } h(x,\vec{y})\ )\quad \text{for } m>0\\
\langle r\mathrm{CRN}[h](x,\vec{y})\rangle_i^{m,\vec{n}} &= \langle 0\cdot h(z,\vec{y})\rangle_{\mathrm{len}_{h(z,\vec{y})}(i,\vec{n})+1}^{i,\vec{n}}\big[\langle x\rangle_j^m/\langle z\rangle_j^i\big]_{1\le j\le i}\\
&\quad (\text{ where } z \text{ does not occur in } h(x,\vec{y})\ )\quad \text{for } m>0
\end{aligned}$$

[2]Where $\vec{n}'$ represents the lengths of the variables from $\vec{y}$ that actually occur in $t[\vec{y}/\vec{x}]$.

$$\big\langle \mathrm{TRN}[g,h,h_\ell,h_r](x,z,\vec{y})\big\rangle_i^{m,p,\vec{n}} = \begin{cases} \langle g(x,z,\vec{y})\rangle_i^{m,p,\vec{n}} & \text{if } m \le 1 \\ \Big\langle h\big(x,z,\vec{y},\mathrm{TRN}[g,h,h_\ell,h_r](x\blacktriangleleft, h_\ell(z),\vec{y}), \\ \qquad \mathrm{TRN}[g,h,h_\ell,h_r](\blacktriangleright x, h_r(z),\vec{y})\big)\Big\rangle_i^{m,p,\vec{n}} & \text{if } 1 < m \end{cases}$$

Remark 4.2.1 Note that "?" is the only primitive function symbol that has non-trivial term formulas (because it is the only function that depends directly on the values of its arguments), so that any non-trivial term formula must depend on ? in some way.

4.3 Propositional translations

The propositional translations of formulas of T_1 are defined inductively as follows (where $\odot$ stands for any one of the binary propositional connectives, and $\vec{m}_0$ and $\vec{m}_1$ represent the lengths of the variables that occur in A and B (or t and u), respectively.)

$$\begin{aligned}
[\![t = u]\!]^{\vec{m}} &= \begin{cases} \bigwedge\limits_{1\le i\le \mathrm{len}_t(\vec{m}_0)} \langle t\rangle_i^{\vec{m}_0} \leftrightarrow \langle u\rangle_i^{\vec{m}_1} & \text{if } \mathrm{len}_t(\vec{m}_0) = \mathrm{len}_u(\vec{m}_1), \\ \bot & \text{otherwise.} \end{cases} \\
[\![\neg A]\!]^{\vec{m}} &= \neg[\![A]\!]^{\vec{m}} \\
[\![A \odot B]\!]^{\vec{m}} &= [\![A]\!]^{\vec{m}_0} \odot [\![B]\!]^{\vec{m}_1}
\end{aligned}$$

4.4 The simulation result

Now, we can prove the following theorem.

Theorem 4.4.1 *If A is provable in T_1, then for any $\vec{m}$, $[\![A]\!]^{\vec{m}}$ has uniform polysize $\mathcal{F}$-proofs.*

Proof The proof is by induction on the number of inferences in the proof of A. If A is an axiom, then Subsection 4.4.1 below shows that $[\![A]\!]^{\vec{m}}$ has linear-size $\mathcal{F}$-proofs. If A is obtained by a derivation, then by the induction hypothesis, the propositional translations of the premises of the last inference all have short $\mathcal{F}$-proofs. Subsection 4.4.2 below shows that in this case also, $[\![A]\!]^{\vec{m}}$ has short $\mathcal{F}$-proofs. Moreover, all these $\mathcal{F}$-proofs are uniform, in the sense that there exists a specific function that takes a theorem of T_1 and the lengths of its variables into a $\mathcal{F}$-proof of the translation of the theorem. This function is described implicitly in the sections that follow, but it can be formalized in T_1 itself, using techniques similar to those developed in Chapter 5 (we do not expect any technical difficulties in doing this but time constraints prevent us from working out the details). $\square$

4.4.1 Axioms

For most axioms of the form $t = u$, just writing down the definitions of $\langle t\rangle_i$ and $\langle u\rangle_i$ is enough to see that the axiom is a theorem with short proofs since $\langle t\rangle_i = \langle u\rangle_i$. We give a more detailed argument only for a few axioms.

0. The axioms for the propositional calculus can obviously be simulated by any $\mathcal{F}$-system.

1. (a) By reflexivity, there are linear-size $\mathcal{F}$-proofs of $\bigwedge\langle x\rangle_i^m \leftrightarrow \langle x\rangle_i^m$ for any variable x.

   (b) If $m \neq n$, then the antecedent of the axiom translates to $\bot$ so the translation of the axiom is a trivial theorem. If $m = n$, then the commutativity of $\leftrightarrow$ gives linear-size $\mathcal{F}$-proofs of $\left(\bigwedge\langle x\rangle_i^m \leftrightarrow \langle y\rangle_i^n\right) \to \left(\bigwedge\langle y\rangle_i^n \leftrightarrow \langle x\rangle_i^m\right)$.

   (c) If $m \neq n$ or $n \neq p$, then one of the antecedents of the axiom translates to $\bot$ so the translation of the axiom is a trivial theorem. If $m = n = p$, then the transitivity of $\leftrightarrow$ gives linear-size $\mathcal{F}$-proofs of $\left(\left(\bigwedge\langle x\rangle_i^m \leftrightarrow \langle y\rangle_i^n\right) \wedge \left(\bigwedge\langle y\rangle_i^n \leftrightarrow \langle z\rangle_i^p\right)\right) \to \left(\bigwedge\langle x\rangle_i^m \leftrightarrow \langle z\rangle_i^p\right)$.

   (d) An easy induction on the structure of the function symbol f, together with properties of $\leftrightarrow$, is sufficient to show that there are short $\mathcal{F}$-proofs of $\left(\bigwedge\langle x_1\rangle_i^{m_1} \leftrightarrow \langle y_1\rangle_i^{n_1} \wedge \cdots \wedge \bigwedge\langle x_k\rangle_i^{m_k} \leftrightarrow \langle y_k\rangle_i^{n_k}\right) \to \bigwedge\langle f(x_1,\ldots,x_k)\rangle_i^{m_1,\ldots,m_k} \leftrightarrow \langle f(y_1,\ldots,y_k)\rangle_i^{n_1,\ldots,n_k}$ when $m_1 = n_1, \ldots, m_k = n_k$ (the axiom's translation becoming a trivial theorem otherwise as one of the antecedents translates to $\bot$). The size of these proofs is linear if f does not contain any function defined by TRN; it is polynomial otherwise (since the size of the term formulas can be polynomial in the lengths of the variables).

2. $\langle 0\rangle_1 = \bot$ and $\langle 1\rangle_1 = \top$.

3. (a) For all $1 \leq i \leq m+0$, $\langle x\cdot\varepsilon\rangle_i^m = \langle x\rangle_i^m$, and for all $1 \leq i \leq m+n+1$,

$$\langle x\cdot y0\rangle_i^{m,n} = \begin{cases} \langle x\rangle_i^m & \text{if } i \leq m \\ \langle y\cdot 0\rangle_{i-m}^n = \begin{cases} \langle y\rangle_{i-m}^n & \text{if } i-m \leq n \\ \langle 0\rangle_{i-m-n} & \text{if } n < i-m \end{cases} & \text{if } m < i \end{cases}$$

$$\langle (x\cdot y)\cdot 0\rangle_i^{m,n} = \begin{cases} \langle x\cdot y\rangle_i^{m,n} = \begin{cases} \langle x\rangle_i^m & \text{if } i \leq m \\ \langle y\rangle_{i-m}^n & \text{if } m < i \end{cases} & \text{if } i \leq m+n \\ \langle 0\rangle_{i-(m+n)} & \text{if } m+n < i \end{cases}$$

(similarly for $x\cdot y1 = (x\cdot y)\cdot 1$).

   (b) $[\![x\cdot y = \varepsilon]\!]^{m,n}$ holds iff $m+n = 0$, and $[\![x = \varepsilon \wedge y = \varepsilon]\!]^{m,n}$ holds iff $m = 0$ and $n = 0$.

   (c) $[\![x\cdot y = 0]\!]^{m,n}$ holds iff $m+n = 1$ and $\langle x\rangle_1^1 = \bot$ or $\langle y\rangle_1^1 = \bot$. Similarly for $x\cdot y = 1$.

4. (a) For $1 \le i \le m$, $\langle \varepsilon \rhd x \rangle_i^m = \langle x \rangle_{i+0}^m$ by definition; also, for $1 \le i \le m - (n+1)$, $\langle (0y) \rhd x \rangle_i^{m,n} = \langle x \rangle_{i+n+1}^m = \langle y \rhd x \rangle_{i+1}^{m,n} = \langle 0 \rhd (y \rhd x) \rangle_i^{m,n}$, and the same reasoning applies to $\langle (1y) \rhd x \rangle_i^{m,n}$.

   (b) $\mathrm{len}_{0 \rhd \varepsilon} = 0 \dot{-} 1 = 0 = \mathrm{len}_\varepsilon$, and for $1 \le i \le m + 1 \dot{-} 1$, $\langle 0 \rhd (0x) \rangle_i^m = \langle 0x \rangle_{i+1}^m = \langle x \rangle_{(i+1)-1}^m = \langle x \rangle_i^m$ (similarly for $\langle 0 \rhd (1x) \rangle_i^m$). The same reasoning applies with "$1 \rhd \ldots$" in place of "$0 \rhd \ldots$".

   (c) $\mathrm{len}_{y \rhd x}(m,n) = m \dot{-} n$ and $\mathrm{len}_{x \rhd y0}(m,n) = \mathrm{len}_{x \rhd y1}(m,n) = (n+1) \dot{-} m$, and $m \dot{-} n = 0 \leftrightarrow m - n \le 0 \leftrightarrow m \le n \leftrightarrow m < n+1 \leftrightarrow 0 < (n+1) - m \leftrightarrow (n+1) \dot{-} m \ne 0$.

5. (a) Similarly to 4a.

   (b) Similarly to 4b.

   (c) Similarly to 4c.

6. (a) $[\![{}_0\varepsilon = \varepsilon]\!] = \top$ since $\mathrm{len}_{0\varepsilon} = \mathrm{len}_\varepsilon = 0$, and for $1 \le i \le m+1$, $\langle {}_0(x0) \rangle_i^m = \langle {}_0(x1) \rangle_i^m = \bot$ and $\langle {}_0x \cdot 0 \rangle_i^m = \begin{cases} \langle {}_0x \rangle_i^m & \text{if } i \le m, \\ \langle 0 \rangle_{i-m} & \text{if } m < i. \end{cases}$

   (b) Similarly to 6a.

7. $\langle \varepsilon \mathbin{?} (x,y,z) \rangle_i^{m,n,p} = \langle x \rangle_i$ for $1 \le i \le m$, by definition; also, for $1 \le i \le \max\{n,p\}$,

$$\begin{aligned}
\langle (w0) \mathbin{?} (x,y,z) \rangle_i^{k,m,n,p} &= \big(\neg \langle w0 \rangle_{k+1}^k \wedge \langle {}_0(z \lhd y) \cdot y \rangle_i^{n,p}\big) \\
&\quad \vee \big(\langle w0 \rangle_{k+1}^k \wedge \langle {}_0(y \lhd z) \cdot z \rangle_i^{n,p}\big) \\
&= \big(\neg\bot \wedge \langle {}_0(z \lhd y) \cdot y \rangle_i^{n,p}\big) \vee \big(\bot \wedge \langle {}_0(y \lhd z) \cdot z \rangle_i^{n,p}\big) \\
&\leftrightarrow \langle {}_0(z \lhd y) \cdot y \rangle_i^{n,p}, \\
\langle (w1) \mathbin{?} (x,y,z) \rangle_i^{k,m,n,p} &= \big(\neg \langle w1 \rangle_{k+1}^k \wedge \langle {}_0(z \lhd y) \cdot y \rangle_i^{n,p}\big) \\
&\quad \vee \big(\langle w1 \rangle_{k+1}^k \wedge \langle {}_0(y \lhd z) \cdot z \rangle_i^{n,p}\big) \\
&= \big(\neg\top \wedge \langle {}_0(z \lhd y) \cdot y \rangle_i^{n,p}\big) \vee \big(\top \wedge \langle {}_0(y \lhd z) \cdot z \rangle_i^{n,p}\big) \\
&\leftrightarrow \langle {}_0(y \lhd z) \cdot z \rangle_i^{n,p}.
\end{aligned}$$

8. (a) For all $1 \le i \le m$,

$$\langle (x\blacktriangleleft) \cdot (\blacktriangleright x) \rangle_i^m = \begin{cases} \langle x\blacktriangleleft \rangle_i^m = \langle x \rangle_i^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle \blacktriangleright x \rangle_{i-\lfloor m/2 \rfloor}^m = \langle x \rangle_{i-\lfloor m/2 \rfloor + \lfloor m/2 \rfloor}^m & \text{if } \lfloor m/2 \rfloor < i \end{cases}$$

   (b) $x\blacktriangleleft \lhd \blacktriangleright x = \varepsilon$ since $\mathrm{len}_{x\blacktriangleleft}(m) \dot{-} \mathrm{len}_{\blacktriangleright x}(m) = \lfloor m/2 \rfloor \dot{-} \lceil m/2 \rceil = 0$;
   $1 \rhd (x\blacktriangleleft \rhd \blacktriangleright x) = \varepsilon$ since $(\mathrm{len}_{\blacktriangleright x}(m) \dot{-} \mathrm{len}_{x\blacktriangleleft}(m)) \dot{-} 1 = \lceil m/2 \rceil \dot{-} \lfloor m/2 \rfloor \dot{-} 1 = 0$.

The last four axioms are easy to prove if we note the following two facts.

(a) *For any term t, $t \lhd t = \varepsilon$ and therefore $\langle_0(t \lhd t) \cdot t\rangle_i^{\vec{m}} = \langle t\rangle_i^{\vec{m}}$ for $1 \le i \le \mathrm{len}_t(\vec{m})$.*

(b) *Within any $\mathcal{F}$-system, the identity $(\neg p \wedge q) \vee (p \wedge q) \leftrightarrow q$ has linear-size proofs.*

Together, these facts show that

$$\langle x \mathbin{?} (y, z, z)\rangle_i^{m,n,p} \leftrightarrow \begin{cases} \langle y\rangle_i^n & \text{if } m = 0, \\ \langle z\rangle_i^p & \text{if } m > 0. \end{cases}$$

9. (a) Note that $\mathrm{len}_{(x0)\blacktriangleleft}(m) = \lfloor (m+1)/2 \rfloor = \lceil m/2 \rceil$ and

$$\begin{aligned}&\mathrm{len}_{x\blacktriangleleft \rhd \blacktriangleright x?(x\blacktriangleleft, x\blacktriangleleft\cdot((\blacktriangleright x\cdot 0)\lhd\blacktriangleright x), x\blacktriangleleft\cdot((\blacktriangleright x\cdot 0)\lhd\blacktriangleright x))}(m)\\ &= \textbf{if } \lceil m/2 \rceil \dot{-} \lfloor m/2 \rfloor = 0 \textbf{ then } \lfloor m/2 \rfloor \textbf{ else } \lfloor m/2 \rfloor + (\lceil m/2 \rceil + 1 - \lceil m/2 \rceil)\\ &= \lceil m/2 \rceil,\end{aligned}$$

so both terms do have the same length.

Now, for $1 \le i \le \lceil m/2 \rceil$, $\langle (x0)\blacktriangleleft\rangle_i^m = \langle x0\rangle_i^m = \langle x\rangle_i^m$ and

$$\begin{aligned}&\langle x\blacktriangleleft \rhd \blacktriangleright x \mathbin{?} \big(x\blacktriangleleft, x\blacktriangleleft \cdot ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x), x\blacktriangleleft \cdot ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x)\big)\rangle_i^m\\ &\leftrightarrow \begin{cases} \langle x\blacktriangleleft\rangle_i^m & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \langle x\blacktriangleleft \cdot ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x)\rangle_i^m & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases}\\ &\leftrightarrow \begin{cases} \langle x\rangle_i^m & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \begin{cases} \langle x\blacktriangleleft\rangle_i^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x)\rangle_1^m & \text{if } i > \lfloor m/2 \rfloor \end{cases} & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases}\\ &\leftrightarrow \begin{cases} \langle x\rangle_i^m & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \begin{cases} \langle x\rangle_i^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle x\rangle_i^m & \text{if } i = \lceil m/2 \rceil \end{cases} & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases}\\ &\leftrightarrow \langle x\rangle_i^m\end{aligned}$$

since $\langle ((\blacktriangleright x \cdot 0) \lhd \blacktriangleright x)\rangle_1^m = \langle x\rangle_{1+\lfloor m/2 \rfloor}^m$. The case for $x1$ is identical.

(b) Note that $\mathrm{len}_{\blacktriangleright(x0)}(m) = \lceil (m+1)/2 \rceil = \lfloor m/2 \rfloor + 1$ and

$$\begin{aligned}&\mathrm{len}_{x\blacktriangleleft\rhd\blacktriangleright x?(\blacktriangleright x\cdot 0, 1\rhd(\blacktriangleright x\cdot 0), 1\rhd(\blacktriangleright x\cdot 0))}(m)\\ &= \textbf{if } \lceil m/2 \rceil \dot{-} \lfloor m/2 \rfloor = 0 \textbf{ then } \lceil m/2 \rceil + 1 \textbf{ else } (\lceil m/2 \rceil + 1) - 1\\ &= \lfloor m/2 \rfloor + 1,\end{aligned}$$

so both terms do have the same length.

Now, for $1 \le i \le \lfloor m/2 \rfloor + 1$,

$$\langle \blacktriangleright (x0) \rangle_i^m = \langle x0 \rangle_{i+\lfloor (m+1)/2 \rfloor}^m = \begin{cases} \langle x \rangle_{i+\lceil m/2 \rceil}^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle 0 \rangle_1 & \text{if } i = \lfloor m/2 \rfloor + 1 \end{cases}$$

and

$$\begin{aligned} &\langle x\blacktriangleleft \rhd \blacktriangleright x \mathbin{?} \big(\blacktriangleright x \cdot 0, 1 \rhd (\blacktriangleright x \cdot 0), 1 \rhd (\blacktriangleright x \cdot 0)\big)\rangle_i^m \\ \leftrightarrow &\begin{cases} \langle \blacktriangleright x \cdot 0 \rangle_i^m & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \langle 1 \rhd (\blacktriangleright x \cdot 0) \rangle_i^m & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases} \\ \leftrightarrow &\begin{cases} \begin{cases} \langle \blacktriangleright x \rangle_i^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle 0 \rangle_1 & \text{if } i > \lfloor m/2 \rfloor \end{cases} & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \langle \blacktriangleright x \cdot 0 \rangle_{i+1}^m & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases} \\ \leftrightarrow &\begin{cases} \begin{cases} \langle \blacktriangleright x \rangle_i^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle 0 \rangle_1 & \text{if } i > \lfloor m/2 \rfloor \end{cases} & \text{if } \lceil m/2 \rceil = \lfloor m/2 \rfloor \\ \begin{cases} \langle \blacktriangleright x \rangle_{i+1}^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle 0 \rangle_1 & \text{if } i > \lfloor m/2 \rfloor \end{cases} & \text{if } \lceil m/2 \rceil > \lfloor m/2 \rfloor \end{cases} \\ \leftrightarrow &\begin{cases} \langle x \rangle_{i+\lceil m/2 \rceil}^m & \text{if } i \le \lfloor m/2 \rfloor \\ \langle 0 \rangle_1 & \text{if } i = \lfloor m/2 \rfloor + 1 \end{cases} \end{aligned}$$

The case for $x1$ is identical.

(c) Similarly to 9a.

(d) Similarly to 9b.

10. $\langle [\lambda \vec{x} \,.\, t](\vec{x}) \rangle_i^{\vec{m}} = \langle t[\vec{x}/\vec{x}] \rangle_i^{\vec{m}'} = \langle t \rangle_i^{\vec{m}'}$ for all $1 \le i \le \text{len}_t(\vec{m}')$.

11. (a) Since $\text{len}_{\ell\text{CRN}[h](\varepsilon,\vec{y})}(\vec{n}) = 0$, $[\![\ell\text{CRN}[h](\varepsilon, \vec{y}) = \varepsilon]\!]^{\vec{n}} = \top$.

Also, because $\text{len}_{(z\cdot 0)\lhd z}(p) = p + 1 \dot{-} p = 1$, we have that

$$\begin{aligned} \langle \ell\text{CRN}[h](0x, \vec{y}) \rangle_1^{m,\vec{n}} &= \langle h(z, \vec{y}) \cdot 0 \rangle_1^{m+1,\vec{n}} \big[\langle 0x \rangle_j^m / \langle z \rangle_j^{m+1}\big]_{1 \le j \le m+1} \\ &= \langle h(0x, \vec{y}) \cdot 0 \rangle_1^{m,\vec{n}} \\ &= \langle (h(0x, \vec{y}) \cdot 0) \lhd h(0x, \vec{y}) \rangle_1^{m,\vec{n}} \\ &= \langle \big((h(0x, \vec{y}) \cdot 0) \lhd h(0x, \vec{y})\big) \cdot \ell\text{CRN}[h](x, \vec{y}) \rangle_1^{m,\vec{n}} \end{aligned}$$

(similarly for $1x$),

and for $1 < i \le m+1$,

$$\begin{aligned}
\langle \ell\mathrm{CRN}[h](0x,\vec{y})\rangle_i^{m,\vec{n}} &= \langle h(z,\vec{y})\cdot 0\rangle_1^{m+2-i,\vec{n}}\big[\langle 0x\rangle_{j+i-1}^{m}/\langle z\rangle_j^{m+2-i}\big]_{1\le j\le m+2-i}\\
&= \langle h(z,\vec{y})\cdot 0\rangle_1^{m+1-(i-1),\vec{n}}\\
&\qquad \big[\langle x\rangle_{j+(i-1)-1}^{m}/\langle z\rangle_j^{m+1-(i-1)}\big]_{1\le j\le m+1-(i-1)}\\
&= \langle \ell\mathrm{CRN}[h](x,y)\rangle_{i-1}^{m,\vec{n}}\\
&= \langle\big((h(0x,\vec{y})\cdot 0)\lhd h(0x,\vec{y})\big)\cdot \ell\mathrm{CRN}[h](x,\vec{y})\rangle_i^{m,\vec{n}}
\end{aligned}$$

(similarly for $1x$).

(b) Similarly to 11a.

12. For all $1 \le i \le \mathrm{len}_{\mathrm{TRN}[g,h,h_\ell,h_r](x,z,\vec{y})}(m,p,\vec{n})$, and by the remark above,

$$\begin{aligned}
\langle x\lhd 1\ ?\ (g(x,z,\vec{y}),t,t)\rangle_i^{m,p,\vec{n}} &= \begin{cases}\langle g(x,z,\vec{y})\rangle_i^{p,\vec{n}} & \text{if } m\dot{-}1=0\\ \big(\neg\langle x\lhd 1\rangle_{m\dot{-}1}^{m}\wedge\langle_0(t\lhd t)\cdot t\rangle_i^{m,p,\vec{n}}\big) & \\ \vee\big(\langle x\lhd 1\rangle_{m\dot{-}1}^{m}\wedge\langle_0(t\lhd t)\cdot t\rangle_i^{m,p,\vec{n}}\big) & \text{if } m\dot{-}1>0\end{cases}\\
&\leftrightarrow \begin{cases}\langle g(x,z,\vec{y})\rangle_i^{p,\vec{n}} & \text{if } m\dot{-}1=0\\ \langle t\rangle_i^{m,p,\vec{n}} & \text{if } m\dot{-}1>0\end{cases}
\end{aligned}$$

where $t = h\big(x,z,\vec{y},\mathrm{TRN}[g,h,h_\ell,h_r](x\blacktriangleleft,h_\ell(z),\vec{y}),\mathrm{TRN}[g,h,h_\ell,h_r](\blacktriangleright x,h_r(z),\vec{y})\big)$.

4.4.2 Rules of inference

For all the rules in Definition 3.1.3, if one of the premises contains an equation of the form $[\![t=u]\!]^{\vec{m}}$ that degenerates to $\perp$ because $\mathrm{len}_t(\vec{m}_0)\neq\mathrm{len}_u(\vec{m}_1)$, then the rule becomes trivial. We therefore assume that none of the propositional translations of atomic formulas of T_1 are degenerate cases. Also, when we use the notation "$\bigwedge\!\!\!\bigwedge$" with no subscript, we implicitly assume that the conjunction is over all relevant values of the index of the term formulas involved.

0. Any standard, complete set of rules for the propositional calculus can be p-simulated within any $\mathcal{F}$-system.

1. We have short $\mathcal{F}$-proofs of $[\![A]\!]^{m,\vec{n}}$. Substituting $\langle t\rangle_i^{\vec{p}}$ for $\langle x\rangle_i^{m}$ throughout these proofs yield short $\mathcal{F}$-proofs of $[\![A[t/x]]\!]^{\vec{p},\vec{n}}$.

2. (a) First, a few observations. Let $F = (x = \varepsilon \vee x = 0 \cdot {\gtrdot} x \vee x = 1 \cdot {\gtrdot} x)$. Then,

$$\llbracket x = \varepsilon \rrbracket^m = \begin{cases} \top & \text{if } m = 0 \\ \bot & \text{if } m > 0 \end{cases}$$

$$\begin{aligned} \llbracket x = 0 \cdot {\gtrdot} x \rrbracket^m &= \bigwedge \langle x \rangle_i^m \leftrightarrow \langle 0 \cdot {\gtrdot} x \rangle_i^m \\ &= \langle x \rangle_1^m \leftrightarrow \langle 0 \rangle_1 \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle {\gtrdot} x \rangle_{i-1}^m \\ &= \langle x \rangle_1^m \leftrightarrow \bot \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle x \rangle_i^m \\ \llbracket x = 1 \cdot {\gtrdot} x \rrbracket^m &= \langle x \rangle_1^m \leftrightarrow \top \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle x \rangle_i^m \end{aligned}$$

so that

$$\begin{aligned} \llbracket F \rrbracket^m &= \begin{cases} \top \vee \bot \vee \bot & \text{if } m = 0 \\ \bot \vee \big(\langle x \rangle_1^m \leftrightarrow \bot \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle x \rangle_i^m\big) & \\ \vee \big(\langle x \rangle_1^m \leftrightarrow \top \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle x \rangle_i^m\big) & \text{if } m > 0 \end{cases} \\ &\leftrightarrow \begin{cases} \top & \text{if } m = 0 \\ \big(\big(\langle x \rangle_1^m \leftrightarrow \bot \vee \langle x \rangle_1^m \leftrightarrow \top\big) \wedge \bigwedge_{1<i} \langle x \rangle_i^m \leftrightarrow \langle x \rangle_i^m\big) & \text{if } m > 0 \end{cases} \\ &\leftrightarrow \begin{cases} \top & \text{if } m = 0 \\ \neg \langle x \rangle_1^m \vee \langle x \rangle_1^m & \text{if } m > 0 \end{cases} \end{aligned}$$

Therefore, $\llbracket F \rrbracket^m$ has linear-size $\mathcal{F}$-proofs.

Now, we have short $\mathcal{F}$-proofs of

$$\begin{aligned} A_\varepsilon &= \llbracket A[\varepsilon] \rrbracket^{\vec{n}}, \\ A_0 &= \llbracket A[x] \rrbracket^{m,\vec{n}} \to \llbracket A[0x] \rrbracket^{m,\vec{n}}, \\ \text{and } A_1 &= \llbracket A[x] \rrbracket^{m,\vec{n}} \to \llbracket A[1x] \rrbracket^{m,\vec{n}}. \end{aligned}$$

If $m = 0$, then by Axiom 1d, there are short proofs of

$$\llbracket x = \varepsilon \rrbracket^m \to \big(\llbracket A \rrbracket^{m,\vec{n}} \leftrightarrow \llbracket A[\varepsilon] \rrbracket^{\vec{n}}\big),$$

which shows that there are short proofs of

$$\llbracket x = \varepsilon \rrbracket^m \to \big(\llbracket A[\varepsilon] \rrbracket^{\vec{n}} \to \llbracket A \rrbracket^{m,\vec{n}}\big). \tag{4.4.1}$$

If $m > 0$, then substituting $1 \rhd x$ for x in A_0 gives short proofs of

$$\llbracket A[1 \rhd x] \rrbracket^{m,\vec{n}} \to \llbracket A[0 \cdot {\gtrdot} x] \rrbracket^{m,\vec{n}};$$

moreover, by Axiom 1d, there are short proofs of

$$[\![x = 0 \cdot \gtrdot x]\!]^m \to \big([\![A]\!]^{m,\vec{n}} \leftrightarrow [\![A[0 \cdot \gtrdot x]]\!]^{m,\vec{n}}\big),$$

and therefore, by transitivity, of

$$[\![x = 0 \cdot \gtrdot x]\!]^m \to \big([\![A[1 \rhd x]]\!]^{m,\vec{n}} \to [\![A]\!]^{m,\vec{n}}\big).$$

A similar argument shows that there are short proofs of

$$[\![x = 1 \cdot \gtrdot x]\!]^m \to \big([\![A[1 \rhd x]]\!]^{m,\vec{n}} \to [\![A]\!]^{m,\vec{n}}\big),$$

which, together with (4.4.1), implies that there are short proofs of

$$[\![F]\!]^m \to \big([\![A[1 \rhd x]]\!]^{m,\vec{n}} \to [\![A]\!]^{m,\vec{n}}\big).$$

Applying modus ponens to this and $[\![F]\!]^m$ gives short proofs of

$$[\![A[1 \rhd x]]\!]^{m,\vec{n}} \to [\![A]\!]^{m,\vec{n}}. \tag{4.4.2}$$

Repeated substitutions of $1 \rhd x$ for x in the proof of (4.4.2) give short proofs of

$$\begin{gathered}[\![A[11 \rhd x]]\!]^{m,\vec{n}} \to [\![A[1 \rhd x]]\!]^{m,\vec{n}} \\ \vdots \\ [\![A[\widehat{m} \rhd x]]\!]^{m,\vec{n}} \to [\![A[\widehat{m-1} \rhd x]]\!]^{m,\vec{n}}\end{gathered}$$

(where we remind the reader that "$\widehat{k}$" is a shorthand for $\overbrace{1 \cdots 1}^{k}$).
Since $\widehat{m} \rhd x = \varepsilon$, using Axiom 1d and modus ponens gives short proofs of

$$[\![A[\varepsilon]]\!]^{\vec{n}} \to [\![A[\widehat{m} \rhd x]]\!]^{m,\vec{n}},$$

and using transitivity $m+1$ times now gives short proofs of

$$[\![A[\varepsilon]]\!]^{\vec{n}} \to [\![A]\!]^{m,\vec{n}}.$$

A final application of modus ponens with A_ε gives the short proofs of $[\![A]\!]^{m,\vec{n}}$ we wanted: as can easily be seen, the size of this proof is $\mathcal{O}(m \cdot p(m,\vec{n}))$, where $p(m,\vec{n})$ was the maximum size of the proofs of A_ε, A_0, A_1.

(b) The same reasoning as for part (a) applies.

3. We have short proofs of $[\![A[\varepsilon, z]]\!]^{p,\vec{n}}$, $[\![A[0, z]]\!]^{p,\vec{n}}$, $[\![A[1, z]]\!]^{p,\vec{n}}$, and

$$A' = \big([\![A[x\blacktriangleleft, h_\ell(z)]]\!]^{m,p,\vec{n}} \wedge [\![A[\blacktriangleright x, h_r(z)]]\!]^{m,p,\vec{n}}\big) \to [\![A[x, z]]\!]^{m,p,\vec{n}}.$$

If $m = 0$, using modus ponens twice on formula (4.4.1) gives a short proof of $[\![A]\!]^{m,p,\vec{n}}$. If $m > 0$, then by repeatedly substituting first $x\blacktriangleleft, h_\ell(z)$ and then $\blacktriangleright x, h_r(z)$ for x, z in the proof of A', we get a binary tree of short proofs of formulas of the form of A', where the formula at the root is $[\![A[x, z]]\!]^{m,p,\vec{n}}$, the formula at each node is implied by the conjunction of the formulas at its children nodes, and at the leaves, the terms being substituted for u can all be proved to be equal to 0 or 1 (single bits of x).

For example, if $m = 3$, the tree would have the form depicted in Figure 4.4.1 (where we've indicated only the consequent of the formula being proved at each node, so that a node B with children C and D represents a proof of the formula $(C \wedge D) \to B$ and a node B with one child C represents a proof of the formula $C \to B$).

$$\begin{tikzcd}
 & & A[x, z] \arrow[dll, dash] \arrow[dr, dash] & & \\
A[x\blacktriangleleft, h_\ell(z)] \arrow[d, dash] & & & A[\blacktriangleright x, h_r(z)] \arrow[dl, dash] \arrow[dr, dash] & \\
A[0, z'] \wedge A[1, z'] & & A\big[(\blacktriangleright x)\blacktriangleleft, h_\ell(h_r(z))\big] \arrow[d, dash] & & A\big[\blacktriangleright\blacktriangleright x, h_r(h_r(z))\big] \arrow[d, dash] \\
 & & A[0, z'] \wedge A[1, z'] & & A[0, z'] \wedge A[1, z']
\end{tikzcd}$$

Figure 4.4.1: Proof tree for $m = 3$.

Therefore, the proofs of $[\![A[0, z]]\!]^{p,\vec{n}}$ and $[\![A[1, z]]\!]^{p,\vec{n}}$ can be used with Rule 1 (substituting the right terms for z) and modus ponens to prove the formulas at the first level, and going up level by level using modus ponens, we obtain a proof of the consequent of the formula at the root of the tree, *i.e.*, $[\![A]\!]^{m,p,\vec{n}}$. Moreover, if $q(m, p, \vec{n})$ is the maximum proof size of the premises and c is a constant satisfying $|h_\ell(z)| \leq 2^c|z|$ and $|h_r(z)| \leq 2^c|z|$, then the size of this proof is $\mathcal{O}(m \cdot q(m, p \cdot m^c, \vec{n}))$ since the tree has depth no more than $\lceil \log_2 m \rceil$ and thus size no more than $2m$.

Remark 4.4.1 Note that the estimates on the size of $\mathcal{F}$-proofs for T_1's theorems given above might be used to get more precise upper bounds than are currently known for the size of $\mathcal{F}$-proofs of certain families of tautologies. For example, the family of tautologies arising from the pigeonhole principle was first shown to have polysize $\mathcal{F}$-proofs by Buss [8] (whose estimate of the size of the proofs was $\mathcal{O}(n^{20})$); a careful analysis of the T_1-proof given in Chapter 3 together with the results of this chapter could provide a better bound (the details would be somewhat tedious but straightforward), and the same could be done for the other families of tautologies mentionned at the end of Chapter 3.

Chapter 5

T_1 Proves the Soundness of $\mathcal{F}$

In this chapter, we will show how to formalize a particular $\mathcal{F}$-system in T_1, how to formalize Buss's algorithm for the *Boolean Sentence Value Problem* (BSVP) [10] in T_1, and how to use the BSVP algorithm to prove the soundness of the given $\mathcal{F}$-system in T_1. Then, we show that $\mathcal{F}$ provably p-simulates any proof system S whose soundness can be proved in T_1.

5.1 Formalizing $\mathcal{F}$-systems

Because any two $\mathcal{F}$-systems p-simulate each other, we will focus on the particular $\mathcal{F}$-system below:

language: variables $p_1, p_2, \ldots$, constants $\top$ and $\bot$, connective $\rightarrow$, brackets ( )

formulas: $\top$, $\bot$, p_i (for any $i \geq 1$), and recursively, $(A \rightarrow B)$ for any formulas A and B

axiom schemes (for any formulas A, B, C):

1. $(A \rightarrow (B \rightarrow A))$
2. $\big((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))\big)$
3. $\big(((B \rightarrow \bot) \rightarrow (A \rightarrow \bot)) \rightarrow (A \rightarrow B)\big)$
4. $\top$

rule (modus ponens): $A, (A \rightarrow B) \vdash B$

5.1.1 Formulas

Given a formula A of $\mathcal{F}$, we will encode A into a string $\#_\ell A$ in the following way: Let $w_A = \max\{i : p_i \text{ appears in } A\}$ and $\ell_A = 1 + \lfloor \lg(w_A + 1) \rfloor$ (one more than the binary length of w_A). Then, for $2^\ell \geq \ell_A + 2$, $\#_{2^\ell} A$ is obtained from A by using the following Gödel-numbering scheme (where $(i)_2^{2^\ell-2} \in \{0,1\}^{2^\ell-2}$ represents i in binary using exactly $2^\ell - 2$ bits—note that because of our choice of 2^ℓ, this will always start with a 0).

symbol	code
p_i	$00(i)_2^{2^\ell-2}$
$\perp$	$0010^{2^\ell-3}$
$\top$	$0011^{2^\ell-3}$
(	$100^{2^\ell-2}$
$\to$	$110^{2^\ell-2}$
)	$010^{2^\ell-2}$

In fact, because we can count in T_1, it is possible to define a slightly more complicated encoding where the codes for "(", "$\to$", and ")" include information about the logical depth of the symbol (making sure that ℓ_A is adjusted to be the maximum of its old value and the logical depth of A). In what follows, we will use "$(_j$", "$\to_j$", and "$)_j$" to represent the corresponding symbol at a logical depth of j.

For example, the formula $A = \big(p_2 \to \big((p_{10} \to \perp) \to p_2\big)\big)$ can be rewritten as $A = (_0 p_2 \to_0 (_1 (_2 p_{10} \to_2 \perp)_2 \to_1 p_2)_1)_0$ by including the depth information for each symbol except logical variables and constants, which would be encoded as follows (where the string was split in two to fit on the page, and a little bit of space was added between blocks of bits representing each symbol, for readability):

$$\begin{aligned}\#_8 A = {} & 10000000\ 00000010\ 11000000\ 10000001\ 10000010\ 00001010 \cdots \\ & \cdots 11000010\ 00100000\ 01000010\ 11000001\ 00000010\ 01000001\ 01000000.\end{aligned}$$

With this encoding, it is easy to write a function $\mathsf{formula}(x, z)$ in T_1 that returns 1 if $x = \#_{2^\ell} A$ for some formula A such that $|\mathsf{pow}^L(z)| = 2^\ell \geq \ell_A + 2$, or 0 otherwise. The function is defined by $r$powCRN and simply checks that x is one of "$\#_{2^\ell}\perp$", "$\#_{2^\ell}\top$", "$\#_{2^\ell}p_i$", or that x has the form "$(_a \cdots)_a$" and that each symbol in x is preceded by a valid string of symbols, according to the following simple rules (using counting and masking operations):

- "p_i", "$\perp$", and "$\top$" must immediately follow either "(" or "$\to$";
- "$(_j$" must immediately follow either "$(_{j-1}$" or "$\to_{j-1}$";
- "$\to_j$" must either immediately follow "$(_j p_i$", "$(_j \perp$", or "$(_j \top$", or it must follow, in order, "$(_j (_{j+1} \cdots)_{j+1}$" (where everything between the parentheses has depth at least $j+1$);
- "$)_j$" must either immediately follow "$\to_j p_i$", "$\to_j \perp$", or "$\to_j \top$", or it must follow, in order, "$\to_j (_{j+1} \cdots)_{j+1}$" (where everything between the parentheses has depth at least $j+1$).

5.1.2 Proofs

Now, we can encode $\mathcal{F}$-proofs easily. A proof $A_1, \ldots, A_k$ is encoded by a pair of strings: $\langle \#_{2^\ell} A_1, \ldots, \#_{2^\ell} A_k \rangle_k$ and $\langle j_1, \ldots, j_k \rangle_k$, where $2^\ell \geq \max_{1 \leq i \leq k}\{\ell_{A_i}\} + 2$ and

$$j_i = \begin{cases} \langle 0, m \rangle & \text{if } A_i \text{ is an instance of axiom } m, \\ \langle k_1, k_2 \rangle & \text{if } A_{k_2} = A_{k_1} \to A_i. \end{cases}$$

Again, it is straightforward to write a function $\mathsf{proof}(x, y, z, w)$ in T_1 that returns 1 if x, y encode an $\mathcal{F}$-proof for $2^\ell = |\mathsf{pow}^L(z)|$ and $k = |w|$, or 0 otherwise, by using simple masking operations and $r$powCRN. (Technically speaking, the encodings of the formulas in a proof must be padded so they all have the same length, but it is a simple matter to take care of.)

Finally, we can define in T_1 the following function:

$$F(x, y, z, w) = \begin{cases} \pi^{|w|}_{|w|}(x) & \text{if } \mathsf{proof}(x, y, z, w) = 1, \\ \#_{|\mathsf{pow}^L(z)|} \top & \text{otherwise,} \end{cases}$$

that returns the tautology proved by x, y (or some fixed tautology if x, y is not an $\mathcal{F}$-proof).

5.2 Buss's algorithm for the BSVP

For reference purposes, we will now summarize Buss's most recent published *ALOGTIME* algorithm for the BSVP [10]. Actually, we present a slight variation of his algorithm applied to formulas containing only the connective "$\to$" as opposed to "$\wedge$" and "$\vee$". Given such a Boolean sentence, we can represent it as a binary tree with $2^{d+1} - 1$ leaves for some d (we can pad any sentence so that it meets this condition by preceding it with enough copies of "$\top \to \cdots$"), where each leaf stores either "$\top$" or "$\bot$" and each interior node represents the connective "$\to$". For two nodes U and V in this tree, we write "$U \rhd V$" to mean that U is an ancestor of V, and "$U \trianglerighteq V$" to mean $U = V$ or $U \rhd V$. The least common ancestor of U and V is denoted $\mathrm{lca}(U, V)$. By convention, we draw trees with the root at the top and the leaves at the bottom, so that "above" and "below" correspond to "ancestor" and "descendant", respectively. Also, we define a *scarred sentence* as a binary tree whose leaves store $\top$ or $\bot$ and that contains exactly one internal node with only one child (the missing child is called the *scar*). The "value" of a scarred sentence is defined to be a pair of truth-values $(t_\top, t_\bot)$, where $t_\top$ is the value of the Boolean sentence obtained when the scar is replaced by $\top$, and similarly for $t_\bot$.

The algorithm will be described as a *pebbling game* on the formula's tree between two players: the *Pebbler* and the *Challenger*, and it proceeds in *rounds*. During each round, the Pebbler places *pebbles* labelled with a truth-value "0" or "1" on nodes of the tree, representing assertions by the Pebbler that the subformulas rooted at those nodes have the indicated truth-values. Following the Pebbler's move, the Challenger *challenges* one of the pebbled positions U, representing an assertion by the Challenger that the pebble value at U is incorrect (and implicitly, that every pebbled position below U is correct).

Intuitively, the essential feature of the pebbling-challenging game is to break up the work by creating scarred subsentences and evaluating them at the same time as their scar, instead of performing the evaluation sequentially. (For example, we could evaluate $(A \to B) \to C$ by evaluating $(A \to \bigstar) \to C$ in parallel with B, where "$\bigstar$" indicates the position of the scar.) Together with Buss's innovative technique for finding scar positions in a semi-oblivious fashion through distinguished leaves, this feature of the algorithm allows even unbalanced sentences to be evaluated in a logarithmic number of steps.

The game is designed so that the Pebbler has a winning strategy if the value of the sentence is "1"; otherwise, the Challenger has a winning strategy. Many of the rules of the game might seem somewhat arbitrary and strict, but they are designed so that a play of the game can be evaluated in $ALOGTIME$ while preserving the property that the correct player has a winning strategy. For example, the game will never last more than d rounds (when there are $2^{d+1} - 1$ leaves), and since specifying arbitrary pebble positions would require $\mathcal{O}(d)$ bits (which would take us outside $ALOGTIME$), there must be a "semi-oblivious" way of specifying pebble positions using only a constant number of bits per round.

Before giving the details of the pebbling game, we need to introduce a bit more notation. First, leaves will be numbered from left to right, starting with 1, and assigned a *rank* equal to the largest integer k such that 2^k divides the leaf number. Next, in each round $i \geq 1$, there will be distinguished leaves L_i, C_i, and R_i (for "left", "center", and "right", respectively) and distinguished nodes A_i and B_i (for "above" and "below", respectively), satisfying the following conditions (see Figure 5.2.1 for a picture).

1. $A_i \trianglerighteq B_i \trianglerighteq C_i$, with $A_i = B_i$ only if $B_i = C_i$;

2. A_i is the lowest (and latest) challenged node, while B_i is the highest pebbled position satisfying the first condition—or $B_i = C_i$ if there is no pebbled node below A_i (informally, the players have "agreed" at B_i but "disagree" at A_i);

3. L_i and R_i are distinct leaves of rank $d - i$ and C_i is of rank greater than $d - i$;

4. every leaf in the subtree rooted at A_i but outside the subtree rooted at B_i has number in the range
$$(L_i - 2^{d-i}, L_i + 2^{d-i}) \cup (R_i - 2^{d-i}, R_i + 2^{d-i})$$
(where the intervals are open).

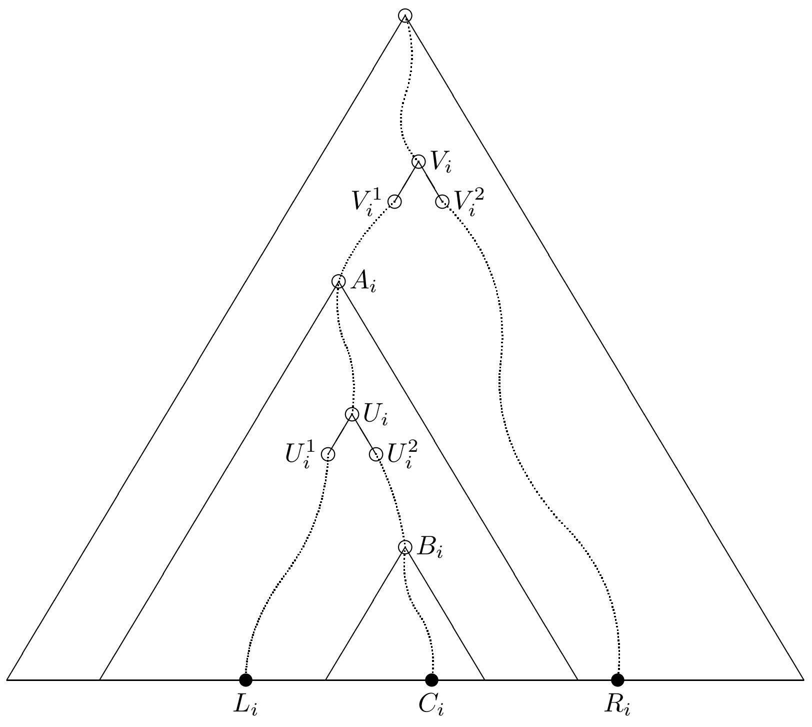


Figure 5.2.1: Distinguished nodes (triangles delineate subtrees; dotted lines indicate paths). Note that this illustrates only *one* of the many configurations possible.

The pebbling game proceeds as follows. In round 0, the Pebbler must pebble the root node with value "1", and the Challenger must challenge the pebble at the root. In preparation for round 1, set A_1 to be the root, $B_1 = C_1$ to be the leaf numbered 2^d, and L_1 and R_1 to be the leaves numbered 2^{d-1} and $2^d + 2^{d-1}$, respectively (see Figure 5.2.2 for an example where each leaf's number and rank is indicated). In round $i \geq 1$, let $U_i = \text{lca}(L_i, C_i)$ and $V_i = \text{lca}(C_i, R_i)$ (note that U_i and V_i are distinct because L_i and R_i are distinct).

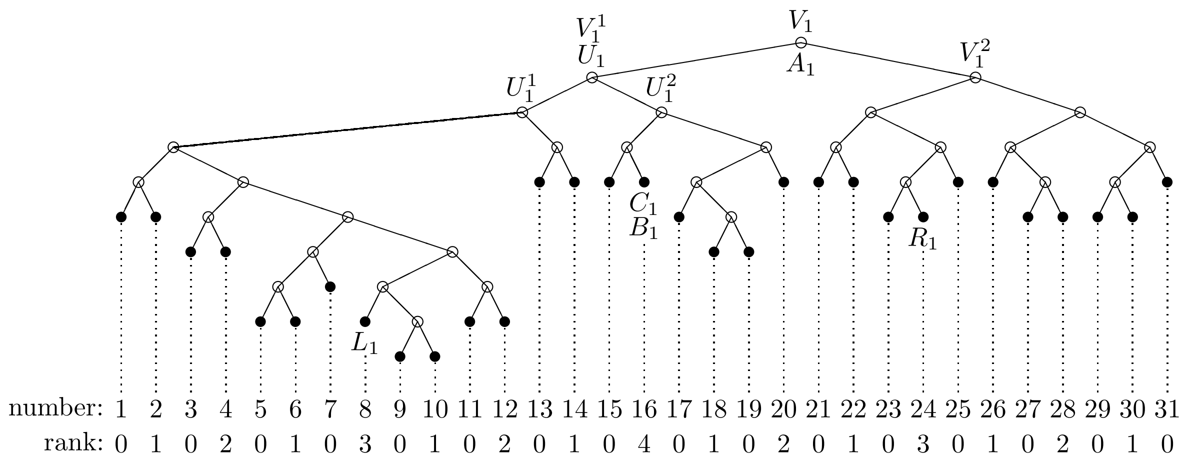

V_1^1
U_1
V_1
A_1
V_1^2
U_1^1
U_1^2
C_1
B_1
R_1
L_1
number: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
rank: 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

Figure 5.2.2: Labelled example of the BSVP algorithm at round 1.

In each round, the Pebbler uses six bits of information (one per node) to pebble U_i, V_i, and their two immediate children (U_i^1, V_i^1 to the left and U_i^2, V_i^2 to the right, respectively). In addition, the Pebbler must use three bits of information to specify the relative positions of A_i, U_i, and V_i (*i.e.*, which nodes are ancestors of which ones—since all three nodes are ancestors of C_i, they all lie on the path from C_i to the root).

The Challenger then challenges one node from among A_i, U_i, U_i^1, U_i^2, V_i, V_i^1, or V_i^2 (using three bits to specify which one), subject to the conditions that the node challenged must be in the subtree rooted at A_i and outside the subtree rooted at B_i.

For round $i+1$, A_{i+1} is set to the node just challenged, B_{i+1} is set to the highest pebbled node below A_{i+1} (or to C_{i+1} if there is no pebbled node below A_{i+1}), and L_{i+1}, C_{i+1}, R_{i+1} are set according to Table 5.2.1.

Challenged Node	Pebbler says $U_i \rhd V_i$	Pebbler says $V_i \rhd U_i$
U_i^1	$C_{i+1} = L_i$ $R_{i+1} = L_i + 2^{d-i-1}$ $L_{i+1} = L_i - 2^{d-i-1}$	$C_{i+1} = L_i$ $R_{i+1} = L_i + 2^{d-i-1}$ $L_{i+1} = L_i - 2^{d-i-1}$
U_i^2	$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{d-i-1}$ $L_{i+1} = L_i + 2^{d-i-1}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{d-i-1}$ $L_{i+1} = L_i + 2^{d-i-1}$
V_i^1	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{d-i-1}$ $L_{i+1} = L_i + 2^{d-i-1}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{d-i-1}$ $L_{i+1} = L_i - 2^{d-i-1}$
V_i^2	$C_{i+1} = R_i$ $R_{i+1} = R_i + 2^{d-i-1}$ $L_{i+1} = R_i - 2^{d-i-1}$	$C_{i+1} = R_i$ $R_{i+1} = R_i + 2^{d-i-1}$ $L_{i+1} = R_i - 2^{d-i-1}$
U_i or V_i	Game Ends	Game Ends

Challenged Node: A_i			
Pebbler says $A_i \rhd U_i, V_i$	Pebbler says $U_i, V_i \unrhd A_i$	Pebbler says $U_i \unrhd A_i \rhd V_i$	Pebbler says $V_i \unrhd A_i \rhd U_i$
$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{d-i-1}$ $L_{i+1} = L_i - 2^{d-i-1}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{d-i-1}$ $L_{i+1} = L_i + 2^{d-i-1}$	$C_{i+1} = C_i$ $R_{i+1} = R_i + 2^{d-i-1}$ $L_{i+1} = L_i + 2^{d-i-1}$	$C_{i+1} = C_i$ $R_{i+1} = R_i - 2^{d-i-1}$ $L_{i+1} = L_i - 2^{d-i-1}$

Table 5.2.1: Next leaf nodes in Buss's BSVP algorithm.

Now, it is easy to show by induction on the number of rounds played that properties 1–4 are preserved for the duration of the algorithm: it is simply a matter of checking case-by-case each possibility in Table 5.2.1 for the values of C_i, L_i, and R_i. For example, suppose that V_1^2 is challenged in Figure 5.2.2, then for round 2, $A_2 = V_1^2$, $B_2 = C_2 = R_1 = 24$, $L_2 = R_1 - 4 = 20$, and $R_2 = R_1 + 4 = 28$ (we refer to leaves by their number), so $A_2 \rhd B_2 = C_2$, A_2 is the lowest challenged node and $B_2 = C_2$, L_2 and R_2 have rank $2 = 4 - 2$ and C_2 has rank $3 > 2$, and every leaf in the subtree rooted at A_2 but outside the subtree rooted at B_2 has number in the range $(20 - 2^{4-2}, 20 + 2^{4-2}) \cup (28 - 2^{4-2}, 28 + 2^{4-2}) = (16, 24) \cup (24, 32)$.

The game ends as soon as one of the players makes an "obvious" mistake, *i.e.*, one from the following list. (Note that by property 4, the game must end by round number d because $(L_d - 2^{d-d}, L_d + 2^{d-d}) \cup (R_d - 2^{d-d}, R_d + 2^{d-d}) = \{L_d, R_d\}$; it is easy to see that in that case, one of the two players will be forced to make a mistake from the list below.)

- Pebbler: when the input nodes of a gate are either leaves or are pebbled and the output node is pebbled incompatibly.
- Challenger: when the output of a gate whose input nodes are either leaves or pebbled is challenged, even though it is correctly pebbled.
- Pebbler: when a leaf is incorrectly pebbled.
- Challenger: when a correctly pebbled leaf is challenged.
- Pebbler: when a node is pebbled with both "0" and "1".
- Pebbler: when an incorrect assertion is made about whether $U_i \rhd V_i$, $A_i \rhd U_i$, $A_i \rhd V_i$.
- Challenger: when the challenged node is above a previously challenged node.
- Challenger: when the challenged node is at or below a previously agreed upon pebble value (a pebble is "agreed upon" if it was placed in an earlier round and in that round, the Challenger challenged an ancestor of that pebble).

It is straightforward to see that the game produces the correct result: if the value of the sentence is "true", the Pebbler can win the game by simply pebbling every node with its correct value and making assertions compatible with the structure of the sentence, while if the value of the sentence is "false", the Challenger can win the game by always challenging the lowest incorrectly pebbled node that is not below a previously agreed upon node.

Moreover, the game can be translated into an *ALOGTIME* algorithm, as follows: First, simulate possible plays of the game using existential moves for the Pebbler and universal moves for the Challenger. Then, for each such game, existentially guess the first mistake made and universally verify that no earlier mistake was made. Note that from a play of the game, it is easy to determine the last round when L_j was computed from R_j and to compute the appropriate sum of powers of 2 to add to R_j in order to get L_i (the same goes for R_i). As for C_i, simply find the last round when C_j was equal to L_j or R_j and we know that $C_i = C_j$. Finally, because it is possible to count in *ALOGTIME*, ancestors can readily be computed to find U_i, V_i and thus determine A_i and B_i.

5.3 Formalizing the BSVP

The algorithm that we will use to solve the BSVP in T_1 is simply a formalization inside our theory of the algorithm described in the previous section. To formalize this algorithm inside T_1, we will define a function BSVP that decomposes and evaluates the sentence, using TRN to perform the work in parallel, disjunction (OR) over all possible Pebbler guesses to emulate existential moves, and composition and implication to emulate Challenger's universal moves.

First, it is easy to define by $r$powCRN a function $\mathsf{sentence}(v, x, z)$ that takes as argument a truth-value assignment v (represented simply by a string whose 1^{st} bit is the value of p_1, whose 2^{nd} bit is the value of p_2, etc.) and the encoding of a formula $x = \#_{|\mathsf{pow}^L(z)|}A$, and returns the sentence obtained by substituting the given truth-values for the variables in x.

Next, we define the function $\mathsf{BSVP}(d, h, m, s)$ that does the work according to Buss's algorithm. There will be one variation: because we want the function to apply to arbitrary sentences, but a sentence must have a power of 2 minus 1 leaves in the algorithm, we will pad sentences so they have $2^{d+1} - 1$ leaves and remember the position of the root of the original sentence inside the padded version as "M". (The algorithm needs to be changed so that it takes the distinguished node M into account at the same time as A, B, L, C, R, but the changes are easy to make since M remains fixed for the duration of the algorithm.) In what follows, the parameter s is fixed and encodes a superformula of the Boolean sentence we are evaluating (padded so it always has a power of two minus one leaves), the parameter m is fixed and indicates the root M of the subformula of s whose value we are interested in, the parameter h varies and represents the history of the game so far, as a sequence of blocks $b_1 a_1 \cdots b_i a_i$ (each block a constant-length string encoding Pebbler's guesses on the relative positions of the nodes A, U, V, B, and M (in b_j), as well as Challenger's chosen node (in a_j)), and d varies and represents 2 to the power of the current round number, in unary. The function $\mathsf{BSVP}(d, h, m, s)$ returns a *truth-triplet* $(c, t_\top, t_\bot)$, where c is a check-bit indicating whether h is a valid description of the structure of s or not, and $(t_\top, t_\bot)$ is the value of the possibly scarred subformula of M picked out by h. The function will be defined by TRN on d, following Buss's algorithm, but first, we must specify how truth-triplets can be combined in various ways.

We generalize disjunction and implication to truth-triplets, and define composition of truth-triplets, as follows:

$$\begin{aligned}
(c^1, t^1_\top, t^1_\bot) \Vvee (c^2, t^2_\top, t^2_\bot) &= \big(c^1 \vee c^2, (c^1 \wedge t^1_\top) \vee (c^2 \wedge t^2_\top), (c^1 \wedge t^1_\bot) \vee (c^2 \wedge t^2_\bot)\big),\\
(c^1, t^1_\top, t^1_\bot) \twoheadrightarrow (c^2, t^2_\top, t^2_\bot) &= \big(c^1 \wedge c^2, t^1_\top \to t^2_\top, t^1_\bot \to t^2_\bot\big),\\
(c^1, t^1_\top, t^1_\bot) \circ (c^2, t^2_\top, t^2_\bot) &= \big(c^1 \wedge c^2, t^1_{t^2_\top}, t^1_{t^2_\bot}\big)
\end{aligned}$$

(where $t^1_{t^2_\top}$ is equal to $t^1_\top$ if $t^2_\top = \top$ and $t^1_\bot$ if $t^2_\top = \bot$, and similarly for $t^1_{t^2_\bot}$).
Then, for $h = b_1 a_1 \cdots b_i a_i$, we can define $\mathsf{BSVP}(d, h, m, s)$ as follows:

$$\mathsf{BSVP}(d,h,m,s) = \bigvee_{\substack{\text{all Pebbler}\\ \text{guesses } b}} \begin{bmatrix}\text{composition of } \mathsf{BSVP}(\blacktriangleright d, hba, m, s)\text{'s using } \circ \text{ and } \twoheadrightarrow,\\ \text{based on structure induced by } b \text{ and where } a \text{ picks out}\\ \text{different subformulas at the current round}\end{bmatrix}.$$

Because b has a fixed length, the disjunction actually represents a fixed number of cases, each one of which has a unique structure determined by the value of b. We will not list all possible cases here (they can easily be written down from the description of Buss's algorithm and Table 5.2.1), but we give two illustrative examples based on the sentence depicted in Figure 5.2.2.

1. Consider the sentence depicted in Figure 5.3.1, where we have filled-in the unique interior node that represents "m" and each leaf that falls inside the correct intervals around L_1 and R_1. At round 1, we have that $\mathsf{BSVP}(d, h, m, s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^1}, m, s) \twoheadrightarrow \mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, m, s)$, where "$a_{U^1}$" and "$a_{U^2}$" are fixed-length strings representing which subsentence is selected, and "b" is the unique fixed-length string representing the correct structure of the formula. The other parts of the formula (under V^2_1) are of no interest because they fall outside "m".

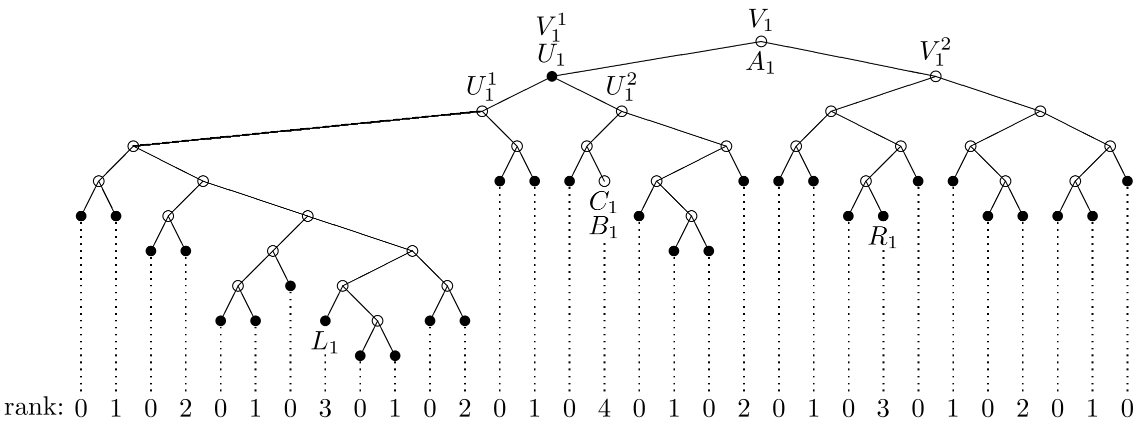
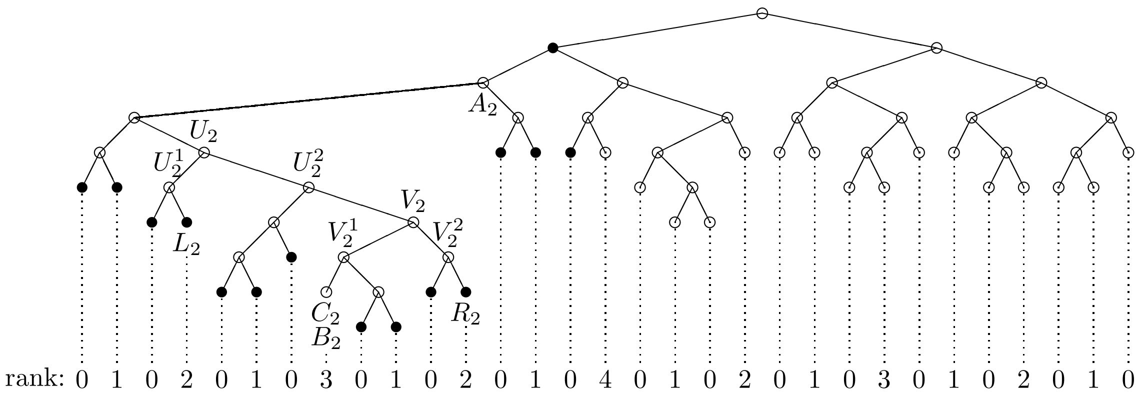

V_1^1
U_1
V_1
A_1
V_1^2
U_1^1
U_1^2
C_1
B_1
R_1
L_1
rank: 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

Figure 5.3.1: Labelled example of the BSVP algorithm at round 1.

2. If we look at the first recursive call of BSVP in the preceding case, we have the situation depicted in Figure 5.3.2, in which case it is easy to see that $\mathsf{BSVP}(d, h, m, s) =$

$$\mathsf{BSVP}(\blacktriangleright d, hba_A, m, s) \circ \Big(\mathsf{BSVP}(\blacktriangleright d, hba_{U^1}, m, s) \twoheadrightarrow \\ \Big(\mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, m, s) \circ \big(\mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, m, s) \twoheadrightarrow \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, m, s)\big)\Big)\Big).$$



A_2
U_2
U_2^1
U_2^2
V_2
V_2^1
V_2^2
L_2
C_2
B_2
R_2
rank: 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

Figure 5.3.2: Labelled example of the BSVP algorithm at round 2.

At the last round (when $d = 1$), the history h is analyzed and compared with the actual structure of the sentence, and the check-bit c returned is $\top$ iff they agree. The check-bit can be obtained by taking the OR of a bit-string computed by $r$powCRN on h, where the bit output for each block b in h is 1 iff b is correct. Moreover, each of these bits can be computed by finding the positions of the L, C, and R leaves from the first part of the history h (which is easy to do by TRN) and then finding least common ancestors of these leaves (which again can be done easily by TRN).

At the same time, the actual sentence left at the last round will have one of the simple forms shown in Figure 5.3.3, which can all be evaluated trivially since we know the values of L, C, and R, and we can easily check when $B = C$.

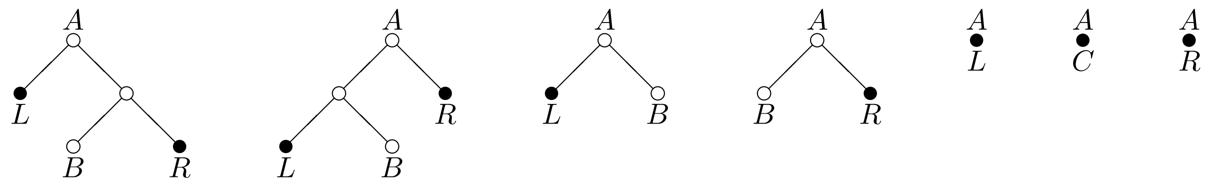


Figure 5.3.3: Base cases for BSVP.

There remains one technical detail that needs to be taken care of in the definition of BSVP: as given, the definition is not a proper application of TRN since the value of $\mathsf{BSVP}(d, h, m, s)$ is recursively defined in terms of $\mathsf{BSVP}(\blacktriangleright d, h', m, s)$ for more than one value of h'. The situation can easily be remedied, in the following way. First, the bounded "$\bigvee$" over all values of b can be implemented with a subtree whose depth is equal to the number of bits in b (which is a constant), where a single bit is added to b at every level (to get all possible values of b at the bottom) and

$\mathbb{W}$ is used to combine the values at each node in the subtree. Next, the composition of values based on the structure determined by b can similarly be carried out step-by-step using a subtree of constant depth, where at each step, the value of b is used to determine which function ($\circ$ or $\twoheadrightarrow$) to apply to combine the results, and which bits to add to a to select subformulas. Both of these steps only require increasing the length of d by a constant factor, and using powmod^L to determine which level is being evaluated in the subtrees.

Finally, we can define a function $\mathsf{VALUE}(v, x, z)$ that computes the truth-value of the formula encoded by (x, z) under v:

$$\mathsf{VALUE}(v, x, z) = \mathsf{AND}\big(\mathsf{BSVP}\big(2^k \times \mathsf{leaves}(x, z), \varepsilon, \mathsf{mask}(x, z), \mathsf{sentence}(v, \mathsf{pad}(x, z), z)\big)\big),$$

where $\mathsf{pad}(x, z)$ pads the formula encoded by x and z, adding enough copies of "$\top \rightarrow \cdots$" to the left so that it has $2^{\delta+1} - 1$ leaves for some integer $\delta \geq 0$, $\mathsf{mask}(x, z)$, returns the position of the root of the formula encoded by x inside $\mathsf{pad}(x, z)$, using a bitmask (where the root is indicated by its main connective "$\rightarrow$"), $\mathsf{leaves}(x, z)$ returns a string of length 2^δ (for the same δ as above), and k is the fixed number of bits in one block "ba" of the history. All these functions are easily defined in T_1 using $r$powCRN and TRN, as follows.

First, note that a formula with n leaves always contains exactly $4n - 3 = n + 3(n - 1)$ symbols (1 for each leaf, and 3 for each connective: two parentheses and one connective), each one encoded by a block of length $|\mathsf{pow}^L(z)|$. This means that the function

$$\mathsf{numleaves}(x, z) = \mathsf{powdiv}^L\big((x \cdot (3 \times \mathsf{pow}^L(z)))\blacktriangleleft\blacktriangleleft, z\big)$$

returns the number of leaves of the formula encoded by x, in unary. Also, we can check whether a formula has a power of 2 minus 1 leaves or not with the function ispow^L. Hence, if we define $\mathsf{leaves}(x, z) = \mathsf{sevael}(x \cdot (7 \times \mathsf{pow}^L(z)), z)$, where

$$\mathsf{sevael}(x, z) = \begin{cases} 1 & \text{if } x \leq^L 8 \times \mathsf{pow}^L(z), \\ \mathsf{sevael}(x\blacktriangleleft, z) \cdot \mathsf{sevael}(\blacktriangleright x, z) & \text{otherwise,} \end{cases}$$

then the string $x \cdot (7 \times \mathsf{pow}^L(z))$ contains $4n - 3 + 7 = 4(n + 1)$ blocks of bits of length $\mathsf{pow}^L(z)$ so that $\mathsf{leaves}(x, z)$ contains exactly 2^δ bits for any formula x that contains between 2^δ and $2^{\delta+1} - 1$ leaves, inclusive.

Next, we define by TRN a function $\mathsf{padding}(y, w, z)$ that returns a balanced sentence containing exactly $|y|$ leaves, each one having the value $\top$, where the logical depth of each symbol is at least $|w|$ and the length of each symbol's encoding is $|\mathsf{pow}^L(z)|$:

$$\mathsf{padding}(y, w, z) = \begin{cases} y \mathrel{?^{ZL}} \big(\varepsilon, \#_{|\mathsf{pow}^L(z)|}\top\big) & \text{if } y = \varepsilon, 0, 1, \\ \#_{|\mathsf{pow}^L(z)|}(_{|w|} \cdot \mathsf{padding}(y\blacktriangleleft, w1, z) \cdot \#_{|\mathsf{pow}^L(z)|}\rightarrow_{|w|} & \\ \qquad\qquad \cdot\, \mathsf{padding}(\blacktriangleright y, w1, z) \cdot \#_{|\mathsf{pow}^L(z)|})_{|w|} & \text{otherwise.} \end{cases}$$

Now, we can easily define

$$\begin{aligned}
&\mathsf{pad}(x,z) = x \qquad \text{(if } x \text{ has a power of 2 minus 1 leaves)},\\
&\mathsf{pad}(x,z) = \#_{|\mathsf{pow}^L(z)|}(_0 \cdot \mathsf{padding}(\mathsf{numleaves}(x,z) \rhd \mathsf{leaves}(x,z), 1, z)\\
&\qquad\qquad\qquad \cdot \#_{|\mathsf{pow}^L(z)|} \rightarrow_0 \cdot \mathsf{deepen}(x,z) \cdot \#_{|\mathsf{pow}^L(z)|})_0 \qquad \text{(otherwise)},
\end{aligned}$$

where $\mathsf{deepen}(x,z)$ is easily defined by $r$powCRN to add 1 to the logical depth of every symbol in the formula x. Finally, we can define mask by $r$powCRN:

$$\begin{aligned}
&\mathsf{mask}(x,z) = \text{the unique connective} \rightarrow \text{at logical depth 0 inside } \mathsf{pad}(x,z)\\
&\qquad\qquad\qquad \text{(if } x \text{ has a power of 2 minus 1 leaves)},\\
&\mathsf{mask}(x,z) = \text{the second connective} \rightarrow \text{at logical depth 1 inside } \mathsf{pad}(x,z) \qquad \text{(otherwise)}.
\end{aligned}$$

5.3.1 Proof of correctness in T_1

THEOREM 5.3.1 *T_1 proves $\mathsf{VALUE}(v, \#_{|\mathsf{pow}^L(z)|}\top, z) = 1$, $\mathsf{VALUE}(v, \#_{|\mathsf{pow}^L(z)|}\bot, z) = 0$, and for arbitrary formulas M and N,*

$$\mathsf{VALUE}(v, \#_{|\mathsf{pow}^L(z)|}(M \rightarrow N), z) = \mathsf{VALUE}(v, \#_{|\mathsf{pow}^L(z)|}M, z) \rightarrow^B \mathsf{VALUE}(v, \#_{|\mathsf{pow}^L(z)|}N, z)$$

*(*i.e.*, VALUE is intensional).*

PROOF The first two statements follow directly from the definition of the functions involved. The third statement follows from Claim 5.3.1 below. □

CLAIM 5.3.1

1. *If h picks out a supersentence of $(M \rightarrow N)$, possibly scarred at B, in the sentence encoded by s, then $\mathsf{BSVP}(d,h,\mathsf{lca}(m,n),s) = \mathsf{BSVP}(d,h,m,s) \twoheadrightarrow \mathsf{BSVP}(d,h,n,s)$ (where $\mathsf{lca}(m,n)$ is a mask indicating the position of the least common ancestor of the nodes masked by m and n in the sentence s).*

2. *$\mathsf{BSVP}(d,\varepsilon,m,s) = \mathsf{BSVP}(d',\varepsilon,m',s')$ for all values of the parameters that represent the same sentence,* i.e., *given $x = \#_\ell A$, for any s, s' that are supersentences of x, where m, m' and d, d' are defined appropriately.*

PROOF

1. By induction on d. When $d = 1$, only the first four cases of Figure 5.3.3 apply. Suppose we are in case 4; then, $A = (M \rightarrow N)$ and $\mathsf{BSVP}(d,h,\mathsf{lca}(m,n),s) = (c,r,\top)$ (where r

is the value of node R), $\mathsf{BSVP}(d,h,m,s) = (c,\top,\bot)$ (since $M = B$ is the scar), and $\mathsf{BSVP}(d,h,n,s) = (c,r,r)$, so the statement is true. The other three cases are similar.

Now, if h picks out a supersentence of $(M \to N)$, possibly scarred at B, then consider the following cases.

(a) If $\mathsf{BSVP}(d,h,\mathsf{lca}(m,n),s) = \mathsf{BSVP}(\blacktriangleright d, hba, \mathsf{lca}(m,n),s)$, then hba also picks out a supersentence of $(M \to N)$, possibly scarred at B, and the statement is true by the induction hypothesis.

(b) If $U > V = (M \to N)$, then

$$\begin{aligned}&\mathsf{BSVP}(d,h,\mathsf{lca}(m,n),s)\\ &= \mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, \mathsf{lca}(m,n),s) \twoheadrightarrow \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, \mathsf{lca}(m,n),s).\end{aligned}$$

Also,

$$\mathsf{BSVP}(d,h,m,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, m,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, \mathsf{lca}(m,n),s)$$

and

$$\mathsf{BSVP}(d,h,n,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, n,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, \mathsf{lca}(m,n),s)$$

by the induction hypothesis. Hence, the statement is true. (The case when $V > U = (M \to N)$ is similar.)

(c) If $(M \to N) = U > V$, then

$$\begin{aligned}&\mathsf{BSVP}(d,h,\mathsf{lca}(m,n),s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^1}, \mathsf{lca}(m,n),s) \twoheadrightarrow\\ &\big(\mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, \mathsf{lca}(m,n),s) \circ \mathsf{BSVP}(\blacktriangleright d, hba_{V}, \mathsf{lca}(m,n),s)\big).\end{aligned}$$

Also,

$$\mathsf{BSVP}(d,h,m,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^1}, m,s)$$

and

$$\mathsf{BSVP}(d,h,n,s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, n,s) \circ \mathsf{BSVP}(\blacktriangleright d, hba_{V}, n,s),$$

where

$$\begin{aligned}\mathsf{BSVP}(\blacktriangleright d, hba_{V}, \mathsf{lca}(m,n),s) &= \mathsf{BSVP}(\blacktriangleright d, hba_{V}, n,s)\\ &= \mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, n,s) \twoheadrightarrow \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, n,s).\end{aligned}$$

Since $t \twoheadrightarrow (t_1 \circ t_2) = (t \twoheadrightarrow t_1) \circ t_2$ for any truth-triplets t, t_1, and t_2 such that t is unscarred (*i.e.*, $t_\top = t_\bot$), the statement follows immediately by the induction hypothesis. (The case when $(M \to N) = V > U$ is similar.)

(d) If $U > (M \to N) > V$, then assuming $M > V$ (the other cases being similar), we have that

$$\begin{aligned}&\mathsf{BSVP}(d, h, \mathsf{lca}(m,n), s)\\&= \mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, \mathsf{lca}(m,n), s) \circ \mathsf{BSVP}(\blacktriangleright d, hba_V, \mathsf{lca}(m,n), s).\end{aligned}$$

Also,

$$\mathsf{BSVP}(d, h, m, s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, m, s) \circ \mathsf{BSVP}(\blacktriangleright d, hba_V, m, s)$$

and

$$\mathsf{BSVP}(d, h, n, s) = \mathsf{BSVP}(\blacktriangleright d, hba_{U^2}, n, s),$$

where

$$\begin{aligned}\mathsf{BSVP}(\blacktriangleright d, hba_V, \mathsf{lca}(m,n), s) &= \mathsf{BSVP}(\blacktriangleright d, hba_V, m, s)\\&= \mathsf{BSVP}(\blacktriangleright d, hba_{V^1}, m, s) \twoheadrightarrow \mathsf{BSVP}(\blacktriangleright d, hba_{V^2}, m, s).\end{aligned}$$

Since $(t_1 \circ t_2) \twoheadrightarrow t = (t_1 \twoheadrightarrow t) \circ t_2$ for any truth-triplets t, t_1, and t_2 such that t is unscarred, the statement follows immediately by the induction hypothesis. (The case when $U > (M \to N) > V$ is similar.)

(e) The case when $(M \to N) > U, V$ is similar to the last one, and all cases can be suitably simplified when $B > U$ or $B > V$.

2. We will actually prove that for all supersentences s of x, given a mask m for x in s and a suitable value for d, $\mathsf{BSVP}(d, \varepsilon, m, s) = \mathsf{BSVP}(d_x, \varepsilon, m_x, s_x)$, where s_x, m_x, and d_x are the default values for x. This will be proved by induction on the number of leaves of the sentence encoded by x. If x encodes a sentence with only one leaf, then x either encodes "$\top$" or "$\bot$", in which case there is a unique history h that picks out x from any given sentence s containing x. For that history, it is easy to see that $\mathsf{BSVP}(d, \varepsilon, m, s) = \mathsf{BSVP}(1, h, m, s)$ which is equal to the value of x, and the same is true for the default values of s, m, and d.

   Now, suppose that x encodes a sentence $M' = (M \to N)$ with $k > 1$ leaves, and let s be any supersentence of x. Then,

$$\begin{aligned}\mathsf{BSVP}(d, \varepsilon, m', s) &= \mathsf{BSVP}(d, \varepsilon, m, s) \twoheadrightarrow \mathsf{BSVP}(d, \varepsilon, n, s)\\&= \mathsf{BSVP}(d_x, \varepsilon, m_x, s_x) \twoheadrightarrow \mathsf{BSVP}(d_x, \varepsilon, n_x, s_x)\\&= \mathsf{BSVP}(d_x, \varepsilon, m'_x, s_x)\end{aligned}$$

   where the first and last equality are true by Claim 5.3.1(1), and the middle equality is true by the induction hypothesis. □

5.4 The soundness proof

5.4.1 Preliminaries

If we let $\mathsf{TRUE}(v, x, z) = \mathsf{formula}(x, z) \wedge^B \mathsf{VALUE}(v, x, z)$, then we can express the fact that our $\mathcal{F}$-system is sound with the following statement in T_1: $\mathsf{TRUE}(v, F(x, y, z, w), z) = 1$.

We will show that T_1 can prove this statement, by induction on the parameter w (which indicates the number of lines in the proof encoded by x, y). For this, though, we will have to define a form of "strong induction" in T_1. First, we need to define a notion of prefix for strings: "$y \sqsubseteq x$", defined below, returns 1 when y is a prefix of x, 0 otherwise.

$$y \sqsubseteq x = y =^S \mathsf{lc}(x, y).$$

Next, we will formalize the notion of "part-of" quantifiers in T_1. More precisely, we will show how to represent the part-of quantifications $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y]$ and $\bigvee\!\!\!\bigvee_{y \sqsubseteq x} A[y]$ for any fixed formula A. Since we have shown that $=^S$ is equivalent to $=$ in T_1 and that the connectives of T_1 are equivalent to their functional counterparts, we can replace $=$ with $=^S$, $\neg$ with $\neg^B$, etc. inside $A[y]$ to obtain a term $\widehat{A}(y)$ for which we know T_1 can prove $A[y] \leftrightarrow \widehat{A}(y) = 1$, and this for each value of y. Then, if we define

$$\mathsf{cat}_A(yi) = \mathsf{cat}_A(y) \cdot \widehat{A}(yi)$$

by CRN, it is immediately clear that

$$\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y] \leftrightarrow \mathsf{AND}(\mathsf{cat}_A(x)) = 1,$$
$$\bigvee\!\!\!\bigvee_{y \sqsubseteq x} A[y] \leftrightarrow \mathsf{OR}(\mathsf{cat}_A(x)) = 1.$$

Now, suppose that for a particular formula A, we can prove in T_1 that $A[\varepsilon]$ and $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y] \rightarrow A[xi]$. Can we conclude that A is true? An easy NIND on x proves $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y]$ in T_1:

1. $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq \varepsilon} A[y] = A[\varepsilon]$, which we can prove by assumption;

2. assuming that $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y]$, an application of modus ponens gives us $A[xi]$, which implies that $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq xi} A[y]$ holds by properties of AND and the definition of cat_A.

Hence, T_1 proves $\bigwedge\!\!\!\bigwedge_{y \sqsubseteq x} A[y]$, which implies, in particular, that $A[x]$ holds.

5.4.2 The proof

THEOREM 5.4.1 *T_1 proves* $\mathsf{TRUE}(v, F(x, y, z, w), z) = 1$.

We use the "strong induction" described above to prove the theorem. The proof itself will be quite short.

First, $F(x,y,z,\varepsilon) = \#_{|\mathsf{pow}^L(z)|}\top$, and $\mathsf{TRUE}(v, \#_{|\mathsf{pow}^L(z)|}\top, z)$ is obviously equal to 1 by Theorem 5.3.1. From now on, we will implicitly assume that $F(x,y,z,w)$ is not equal to $\#_{|\mathsf{pow}^L(z)|}\top$.

Next, assume that $\bigwedge\!\!\!\bigwedge_{u\sqsubseteq w} \mathsf{TRUE}(v, F(x,y,z,u), z) = 1$, and consider the following cases, based on the value of $y_{|w1|}$, for the value of $\mathsf{TRUE}(v, F(x,y,z,w1), z)$.

- If $y_{|w1|} = \langle 0, 1\rangle$, then $F(x,y,z,w1)$ is the encoding of a formula of the form $(A\to(B\to A))$, in which case Theorem 5.3.1 implies that

  $$\begin{aligned}&\mathsf{TRUE}(v, F(x,y,z,w1), z) = \\ &\quad \mathsf{TRUE}(v, \#_{|\mathsf{pow}^L(z)|}A, z) \to^B \big(\mathsf{TRUE}(v, \#_{|\mathsf{pow}^L(z)|}B, z) \to^B \mathsf{TRUE}(v, \#_{|\mathsf{pow}^L(z)|}A, z)\big),\end{aligned}$$

  which is obviously equal to 1 in T_1, being a simple tautology.

- The other three axioms can easily be dealt with similarly.

- If $y_{|w1|} = \langle k_1, k_2\rangle$, then we know that $F(x,y,z,\widehat{k_2}) = \pi_{k_2}^{|w1|}(x)$ encodes a formula $A_{k_2} = A_{k_1} \to A_{|w1|}$, where A_{k_1} is the formula encoded by $F(x,y,z,\widehat{k_1}) = \pi_{k_1}^{|w1|}(x)$ and $A_{|w1|}$ is the formula encoded by $F(x,y,z,w1)$. But then, by the induction hypothesis, we know that $\mathsf{TRUE}(v, F(x,y,z,\widehat{k_1}), z) = 1$ and

  $$\mathsf{TRUE}(v, F(x,y,z,\widehat{k_2}), z) = \mathsf{TRUE}(v, F(x,y,z,\widehat{k_1}), z) \to^B \mathsf{TRUE}(v, F(x,y,z,w1), z) = 1,$$

  so it immediately follows that $\mathsf{TRUE}(v, F(x,y,z,w1), z) = 1$.

5.5 Simulation results

In this section, we show that $\mathcal{F}$ can p-simulate any proof system S whose soundness can be proved in T_1. A similar result was first proved for PV by Cook [18] (Krajíček gives a more detailed proof in his book [23, Theorem 9.3.17]), but to our knowledge, this is the first theory of *ALOGTIME* reasoning for which such a result is shown.

Intuitively, the proof hinges on the fact that for any formula A, the propositional translations of the T_1 equation "$\mathsf{TRUE}(v, \#_{2^\ell}A, \textsc{l}) = 1$" can be proven equivalent to a substitution instance of A itself, so that if T_1 proves the equation, then A is a tautology with short $\mathcal{F}$-proofs.

More precisely, recall from Chapter 4 that for any equation $t = u$ of T_1, we defined a family of propositional translations $[\![t = u]\!]^{\vec{m},\vec{n}}$ that have polysize $\mathcal{F}$-proofs whenever $t = u$ is a theorem of T_1. Hence, for any formula A and string $\textsc{l}$ such that $|\textsc{l}| = 2^\ell \geq \ell_A + 2$, if T_1 proves

$\mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) = 1$, then there are polysize $\mathcal{F}$-proofs of the corresponding propositional tautologies $[\![\mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) = 1]\!]^k$ (where k is the length of the free variable v), and these tautologies are defined as $\langle \mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) \rangle_1^k \leftrightarrow \top$, which is equivalent to $\langle \mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) \rangle_1^k$ (the term formula describing the first and only bit of the term $\mathsf{TRUE}(v, \#_{2^\ell} A, \text{L})$ as a function of the bits of its free variable v). Also, since T_1 can prove $\mathsf{TRUE}(v, \#_{2^\ell}(A \to B), \text{L}) = \mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) \to^B \mathsf{TRUE}(v, \#_{2^\ell} B, \text{L})$ for any formulas A and B, there are polysize $\mathcal{F}$-proofs that the corresponding propositional translations are equivalent, *i.e.*,

$$\langle \mathsf{TRUE}(v, \#_{2^\ell}(A \to B), \text{L}) \rangle_1^k \leftrightarrow \left(\langle \mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) \rangle_1^k \to \langle \mathsf{TRUE}(v, \#_{2^\ell} B, \text{L}) \rangle_1^k \right),$$

so that if T_1 proves $\mathsf{TRUE}(v, \#_{2^\ell}(A \to B), \text{L}) = 1$, then there are polysize $\mathcal{F}$-proofs of

$$\langle \mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) \rangle_1^k \to \langle \mathsf{TRUE}(v, \#_{2^\ell} B, \text{L}) \rangle_1^k.$$

Applying this reasoning recursively shows that if T_1 proves $\mathsf{TRUE}(v, \#_{2^\ell} A, \text{L}) = 1$, then there are polysize $\mathcal{F}$-proofs (call them Π_k) of $A\big[\langle \mathsf{TRUE}(v, \#_{2^\ell} p_i, \text{L}) \rangle_1^k / p_i\big]$ (*i.e.*, the formula A where each propositional variable p_i has been replaced by the formula $\langle \mathsf{TRUE}(v, \#_{2^\ell} p_i, \text{L}) \rangle_1^k$). Since the subformulas $\langle \mathsf{TRUE}(v, \#_{2^\ell} p_i, \text{L}) \rangle_1^k$ are never "broken up" inside Π_k, we can just substitute p_i for $\langle \mathsf{TRUE}(v, \#_{2^\ell} p_i, \text{L}) \rangle_1^k$ throughout to get polysize $\mathcal{F}$-proofs of A.

Now, since $\mathcal{F}$ can "evaluate" sentences (*i.e.*, given a propositional formula with no variables, $\mathcal{F}$ has polysize proofs that it is equivalent to its truth-value), for any function $f(x_1, \ldots, x_n)$ definable in T_1, and any tuple of strings $a_1 \in \{0,1\}^*, \ldots, a_n \in \{0,1\}^*$, there are polysize $\mathcal{F}$-proofs that $\langle f(\vec{x}) \rangle_i^{\vec{m}} \big[\langle a_j \rangle_{i_j} / \langle x_j \rangle_{i_j}^{m_j} \big] \leftrightarrow \langle f(\vec{a}) \rangle_i$ (where $\langle a_j \rangle_i$ is equal to $\top$ or $\bot$ depending on the value of bit number i of a_j). In particular, if S is a proof system formalizable in T_1 as a function symbol $S(x,z)$, then for any particular S-proof (a,b) of a formula A, there are polysize $\mathcal{F}$-proofs that $S(a,b)$ is equivalent to the encoding of A.

Putting these two facts together, we have that if $S(x,z)$ is a proof system whose soundness can be proved in T_1 (*i.e.*, for which T_1 can prove $\mathsf{TRUE}(v, S(x,z), z) = 1$), then for any particular S-proof (a,b), there are polysize $\mathcal{F}$-proofs of the formula encoded by $S(a,b)$, *i.e.*, $\mathcal{F}$ p-simulates S. Moreover, it appears that the translation from S-proofs to $\mathcal{F}$-proofs can be carried out in NC^1 and thus formalized in T_1, where the simulation proof can also be formalized (although we do not carry this out, we do not expect any technical difficulties to arise in the details of such a formalization).

Remark 5.5.1 Note that Theorems 4.4.1 and 5.4.1 immediately give an alternative proof that $\mathcal{F}$-systems have polysize proofs of their own partial consistency (when suitably expressed), a fact first proved directly by Buss [9]. The partial self-consistency statements obtained through T_1 would be different from the ones Buss considered, but it should be possible to prove that they are equivalent.

Chapter 6

Related Work

In this chapter, we show that Arai's AID is equivalent to QT_1 (a suitably defined quantified version of T_1), and briefly discuss the relationship between T_1 and Clote's ALV or ALV'. We will keep the discussion at a high level, with few technical details.

6.1 T_1 and AID

If we define QT_1 to be a first-order theory whose non-logical symbols are those of T_1 and whose axioms are the universal closures of the axioms of T_1, together with axiom schemes corresponding to NIND and TIND, then we can show that

- QT_1 is a conservative extension of T_1 (the proof is similar to Cook's proof in [17] that $QALV$ is conservative over ALV');
- for every Σ_0^b-formula B in QT_1, there exists a function symbol $\widehat{B}$ in T_1 such that QT_1 proves $B[\vec{x}] \leftrightarrow \widehat{B}(\vec{x}) = 1$ (sharply bounded quantifiers, *e.g.*"$\forall x \leq |t|$", are easy to represent functionally since we already have "part-of" quantifiers from the end of Chapter 5 and $\forall x \leq |t| B[x] \leftrightarrow \bigwedge_{x \sqsubseteq t} B[|x|]$, for example);
- QT_1 proves the scheme of Σ_0^b-LIND (with a straightforward application of NIND).

Next, the primitive functions of AID are all easily defined in T_1 (all treating their inputs "numerically", *i.e.*, ignoring leading zeroes), and their defining axioms can be proven without difficulty. Also, for every inductively defined predicate $A^{\ell,B,\vec{D},I}$ in AID, we can define a $\{0,1\}$-valued function $\widehat{A}$ in T_1 such that the equation $\widehat{A}(\vec{x},p) = 1$ provably satisfies the defining axioms (A.0)–(A.2) of A. This can be done by TRN in a relatively straightforward manner, except for two technicalities that we discuss now.

First, the recursive definition of A in AID involves computing the values of predicates $D_1(\vec{x},p),\ldots,D_m(\vec{x},p)$ at every level of the recursion, even though these computations can only be represented in T_1 by function symbols of rank 1 (*i.e.*, the computations are in NC^1). Since functions of rank 1 are not allowed in the recursive part of a definition by TRN, we need to precompute the values of the $\vec{D}$ predicates for every level and extract the correct values during the recursive definition by TRN. This is accomplished by first computing the concatenation of all the values of p for which the $\vec{D}$ predicates need to be computed, then using CRN to compute the concatenation of the $\vec{D}$ predicates for each value, and finally breaking up this string in the appropriate way during the recursion (so the order in which the values are listed is important and must be chosen carefully).

Second, the depth of the recursion is controlled by the "linear form" $\ell||\vec{x}||$, which is represented in T_1 by a numerical function, *i.e.*, one whose actual string length could be arbitrarily longer than its numerical length because of leading 0's. By simply extending the precomputation carried out above for the $\vec{D}$ predicates so that the values of the B predicate are also computed for each value of p at the "bottom" of the recursion, we can easily define the term $\widehat{A}$ by TRN so that the recursion terminates early once the base cases are reached, so that the exact string length of the parameter controlling the TRN does not matter, as long as it is long enough.

In the other direction, every primitive function of T_1 can easily be defined in AID, using Arai's representation of strings (a string $b_{k-1} \cdots b_0$ is represented by the integer $1b_{k-1} \cdots b_0$), since the primitive "part" function of AID can be used directly to extract arbitrary substrings. From the properties of "part", the defining axioms for the T_1-functions can be proven without difficulty in AID. Also, function symbols defined by ℓCRN or rCRN in T_1 can be defined in AID in a straightforward manner using Comprehension, and the defining axioms for these functions follows directly from the Comprehension axiom in AID. Finally, functions defined by TRN in T_1 can be defined in AID using an inductively defined predicate $A^{\ell,B,\vec{D},I}$ that uses its parameter p to keep track of a *path* through the recursion tree and that computes the appropriate substring of x and the appropriate function of z at each level using the predicates $\vec{D}$. Moreover, the defining axioms of such a function symbol follow directly from the axioms for A in AID.

Now, for every formula B in AID, we let $\widehat{B}$ denote the formula in QT_1 obtained from B by replacing each primitive function symbol by its definition in T_1 and each inductively defined predicate symbol A by its definition $\widehat{A}$ in T_1. Similarly, for every formula B in QT_1, we let $\widetilde{B}$ denote the formula in AID obtained from B by replacing each function symbol of T_1 by its definition in AID.

These translations allow us to show that if a formula B is provable in $AID + \Sigma_0^b$-CA, then QT_1 can prove $\widehat{B}$ (since QT_1 proves Σ_0^b-LIND and Σ_0^b-CA can be defined using CRN and proven using NIND), and that if a formula B is provable in QT_1, then $AID+\Sigma_0^b$-CA can prove $\widetilde{B}$ (since

$AID + \Sigma_0^b$-CA can prove NIND using Comprehension, and TIND in a way similar to Arai's proof of "tree induction").

Hence, AID is equivalent to QT_1, which implies that AID is conservative over T_1. Moreover, since Arai proves in his paper that AID is equivalent to $QALV$ for Σ_1^b-formulas, this also implies that ALV is equivalent to T_1 (modulo the translations from numbers to strings and from strings to numbers given above). Unfortunately, the corresponding result for ALV' and T_1 is not known.

Now, even though AID is conservative over T_1 (through the appropriate translations between strings and numbers), T_1 appears to be more natural and easier to reason with for a variety of reasons.

- T_1 reasons directly with functions in *FALOGTIME*, whereas AID reasons only with predicates (functions have to be defined implicitly).
- T_1's scheme of TRN is simpler than AID's inductive definitions, in the sense that a function defined by TRN carries out only simple computations at each level of the recursion (*i.e.*, the h, h_ℓ, and h_r functions can be computed by constant-depth circuits), unlike the computations requiring logdepth circuits carried out by the "D" predicates at each level of an inductive definition.
- It seems quite tedious to work out precise estimates on the size of $\mathcal{F}$-proofs for the propositional translations of the Σ_1^b-theorems of AID, whereas the corresponding task for T_1 is straightforward (so in a sense, T_1 is "closer" to $\mathcal{F}$-systems than AID).

6.2 T_1 and PV

Based on a similar result of Buss [11], which uses Krajíček, Pudlák & Takeuti's Herbrand-type witnessing theorem [24], Cook [17] has argued that if QPV is conservative over $QALV$, then $P = ALOGTIME$, where QPV is the appropriately defined quantified theory corresponding to PV and $QALV$ is a quantified theory based on Clote's ALV'. A similar result should hold with QT_1 in place of $QALV$, given a suitable interpretation of strings as numbers and numbers as strings.

As for the quantifier-free theories PV and T_1, it is easy to see that if PV is conservative over T_1 (through an appropriate translation between numbers and strings, such as the one used above), then T_1 proves the soundness of $e\mathcal{F}$ (since PV proves the soundness of $e\mathcal{F}$ and $e\mathcal{F}$ can be defined in T_1), so that $\mathcal{F}$ p-simulates $e\mathcal{F}$. Unfortunately, the converse is not known, and this has no known implication for the complexity classes P and NC^1.

Chapter 7

Conclusion

7.1 Summary

As Chapter 2 has shown, L_1 is an elegant and natural recursive characterization of NC^1: simple functions are easy to define and even for more complex functions, the definitions are not unnecessarily complicated. The only exception to that statement might be the "numerical" functions, but even there, the definitions are straightforward and correspond quite closely to the computation of these functions by circuit families. Also, our scheme of TRN seems to capture the computational power of *ALOGTIME* in the most natural way, as evidenced by the short proof that $FALOGTIME \subseteq L_1$. It would be interesting to prove the other direction also (that $L_1 \subseteq FALOGTIME$) by using computations by ATMs as opposed to uniform circuit families, so that both directions of the proof are similar, but time constraints did not allow us to carry out such a proof.

The theory T_1 based on L_1 has the desirable property that its appropriately translated theorems have short $\mathcal{F}$-proofs, and the proof of that fact is quite simple (especially when compared to the corresponding proofs for other theories of *ALOGTIME* reasoning in the literature). In fact, the structure of the $\mathcal{F}$-proofs is straightforward enough that we get precise estimates on their size (as a function of the lengths of the variables). Also, considering the inherent complexity of evaluating Boolean sentences in *ALOGTIME*, the T_1-proof of the soundness of a particular $\mathcal{F}$-system is straightforward, consisting mainly in the formalization of Buss's BSVP algorithm and the proof of its properties. Finally, the fact that $\mathcal{F}$ p-simulates any proof system S whose soundness can be proved in T_1 is also straightforward to prove, and T_1 is the first theory of *ALOGTIME* reasoning for which this result has been shown. All these facts strongly support our claim that T_1 is one of the most natural theories available for *ALOGTIME* reasoning, even though it is based on strings instead of numbers (unlike most of the other theories for polytime or *ALOGTIME* reasoning).

To conclude, it might seem that any algebraic characterization of a complexity class could be used to define a quantifier-free theory like T_1, by simply having function symbols defined recursively and induction rules based on the recursion operators. Although such a theory would undoubtedly reason on functions in the desired complexity class, we would still need to show that the type of reasoning that can be carried out in this theory also falls within the desired complexity class, and there is no clear way of doing this for arbitrary complexity classes. Also, as evidenced by Clote's theories ALV and ALV', it is not an easy task to get a theory that is natural and simple enough to be useful in practice.

7.2 Future work

First, an obvious generalization of L_0 and L_1 suggests itself: for $i \geq 1$, let L_{i+1} be the closure of L_i under COMP, CRN, and $\text{TRN}|_{L_i}^{L_{i+1}}$, defined recursively. A study of these classes (or of a similar extension for the theory T_1) might be interesting. (One fact about L_i which is relatively easy to prove is that it is a subset of the class of functions computable by uniform circuit families of $\mathcal{O}(\log^i)$ depth, but it is unknown if this is a proper containment. It might be interesting to try to prove better results, maybe that the L_i's exactly captures these circuit families, or that the union of the L_i hierarchy defines the class of functions computable by uniform circuit families of polylog depth.)

Next, it would be interesting to compare QT_1 to Takeuti and Clote's TNC^0, maybe to show that the two first-order theories are equivalent. Also, from Arai's results on AID and ALV, we have concluded in Chapter 6 that T_1 and ALV are equivalent, but it is unknown whether or not ALV' is equivalent to ALV (or to T_1).

It would also be interesting to see if "tree recursion" and "tree induction" can be adapted to define a similar quantifier-free theory for uniform TC^0 reasoning, hopefully as natural as T_1 is natural for $ALOGTIME$ reasoning (such a theory would correspond to bounded-depth $\mathcal{F}$-systems with threshold gates in the same way that T_1 corresponds to $\mathcal{F}$-systems). Note that there is already a first-order theory $\bar{R}_2^0$ for TC^0 reasoning, defined by Johannsen [22].

Similarly, there should be a way to extend L_0 and L_1 to capture all of NC (based on Bloch's characterization of NC). This could possibly be used to define a quantifier-free theory for NC reasoning, which might lead to a natural propositional proof system that reasons in NC.

Finally, fully relating conservativity results between logical theories for P and $ALOGTIME$ reasoning to equivalence results between $e\mathcal{F}$ and $\mathcal{F}$ systems to collapse results between P and NC^1 remains a central open problem in this area.

Appendix A

Details of Proofs in the Formal Development of T_1

This appendix contains most of the proofs missing from the formal development of T_1 given in Chapter 3. It is included here mainly for the sake of completeness, so the style will be quite terse. In particular, most proofs that consist only in a straightforward application of NIND will be omitted.

On generalizations of NIND

Claim 3.2.54

1. *(L)* $z \rhd yx = z \rhd y \cdot (z \lhd y) \rhd x$ *(R)* $xy \lhd z = x \lhd (y \rhd z) \cdot y \lhd z$
2. *(L)* $y{\lessdot} \rhd x = \mathsf{lb}(x,y) \cdot y \rhd x$ *(R)* $x \lhd {\gtrdot}y = x \lhd y \cdot \mathsf{rb}(x,y)$
3. *(L)* $y \rhd ((x \lhd y) \rhd x) = \varepsilon$ *(R)* $(x \lhd (y \rhd x)) \lhd y = \varepsilon$
4. *(L)* $\mathsf{lc}(\mathsf{rc}(x,y),y) = \mathsf{rc}(x,y)$ *(R)* $\mathsf{rc}(\mathsf{lc}(x,y),y) = \mathsf{lc}(x,y)$
5. *(L)* ${}^{\backprime}(y \rhd x) = y \rhd x \,?^{ZL} \big(\varepsilon, (x \lhd {\gtrdot}(y \rhd x))'\big)$ *(R)* $(x \lhd y)' = x \lhd y \,?^{ZL} \big(\varepsilon, {}^{\backprime}((x \lhd y){\lessdot} \rhd x)\big)$
6. *(L)* $\mathsf{lc}(x,yi) = \mathsf{lc}(x,y) \cdot \mathsf{lb}(x,yi)$ *(R)* $\mathsf{rc}(x,iy) = \mathsf{rb}(x,iy) \cdot \mathsf{rc}(x,y)$
7. *(L)* $\mathsf{lc}(x,y) \cdot y \rhd x = x$ *(R)* $x = x \lhd y \cdot \mathsf{rc}(x,y)$

Proof

1. (L) By NIND on z, Axioms 4c and 5c, and Claim 3.2.32: $\varepsilon \rhd yx = yx = \varepsilon \rhd y \cdot \varepsilon \rhd x = \varepsilon \rhd y \cdot (\varepsilon \lhd y) \rhd x$, $iz \rhd yx = {\gtrdot}(z \rhd yx) = {\gtrdot}(z \rhd y \cdot (z \lhd y) \rhd x) = z \rhd y \,?^{ZL} \big({\gtrdot}((z \lhd y) \rhd x), {\gtrdot}(z \rhd y) \cdot (z \lhd y) \rhd x\big) = z \rhd y \,?^{ZL} \big(\varepsilon \cdot (i \cdot z \lhd y) \rhd x, iz \rhd y \cdot \varepsilon \rhd x\big) = z \rhd y \,?^{ZL} \big(iz \rhd y \cdot (iz \lhd y) \rhd x, iz \rhd y \cdot (iz \lhd y) \rhd x\big) = iz \rhd y \cdot (iz \lhd y) \rhd x$.

2. (L) By NIND on y: $\varepsilon{\lessdot}{\rhd}x = x = \varepsilon{\cdot}x = \mathsf{lb}(x,\varepsilon){\cdot}\varepsilon{\rhd}x$, $(yi){\lessdot}{\rhd}x = y{\rhd}x = {\backprime}(y{\rhd}x){\cdot}{\gtrdot}(y{\rhd}x) = {\backprime}((yi){\lessdot} \rhd x) \cdot ((yi) \rhd x) = \mathsf{lb}(x, yi) \cdot (yi) \rhd x$.

3. (L) By NIND on y, and preceding claims: $\varepsilon \rhd ((x \lhd \varepsilon) \rhd x) = x \rhd x = \varepsilon$,

$$\begin{aligned} yi \rhd ((x \lhd yi) \rhd x) &= {\gtrdot}\big(y \rhd ((x \lhd y){\lessdot} \rhd x)\big) \\ &= y \rhd {\gtrdot}(\mathsf{lb}(x, x \lhd y) \cdot (x \lhd y) \rhd x) \\ &= y \rhd ((x \lhd y) \rhd x) = \varepsilon \end{aligned}$$

(Note that in this proof, we did not explicitly deal with the cases when $x \lhd y = \varepsilon$ or when $(x \lhd y){\lessdot} \rhd x = \varepsilon$. However, it can easily be seen that both cases make the statement trivially true.)

4. (L) By preceding claims: $\mathsf{lc}(\mathsf{rc}(x,y),y) = ((x \lhd y) \rhd x) \lhd \big(y \rhd ((x \lhd y) \rhd x)\big) = \mathsf{rc}(x,y) \lhd \varepsilon = \mathsf{rc}(x,y)$.

5. (L) By NIND on x and preceding claims: ${\backprime}(y \rhd \varepsilon) = \varepsilon = y \rhd \varepsilon \ ?^{ZL} (\varepsilon, \varepsilon)$, ${\backprime}(y \rhd xi) = y \rhd xi \ ?^{ZL} (\varepsilon, {\backprime}(y \rhd x \cdot i)) = y \rhd xi \ ?^{ZL} \big(\varepsilon, y \rhd x \ ?^{ZL} (i, {\backprime}(y \rhd x))\big) = y \rhd xi \ ?^{ZL} \big(\varepsilon, y \rhd x \ ?^{ZL} \big(i, (x \lhd {\gtrdot}(y \rhd x))'\big)\big) = y \rhd xi \ ?^{ZL} \big(\varepsilon, (xi \lhd (y \rhd x))'\big) = y \rhd xi \ ?^{ZL} \big(\varepsilon, (xi \lhd {\gtrdot}(y \rhd xi))'\big)$.

6. (L) By preceding claims: $\mathsf{lc}(x, yi) = x \lhd (yi \rhd x) = x \lhd {\gtrdot}(y \rhd x) = x \lhd (y \rhd x) \cdot y \rhd x \ ?^{ZL} \big(\varepsilon, (x \lhd {\gtrdot}(y \rhd x))'\big) = x \lhd (y \rhd x) \cdot {\backprime}(y \rhd x) = \mathsf{lc}(x,y) \cdot \mathsf{lb}(x, yi)$. □

On propositional reasoning

THEOREM 3.2.6

1. $\approx^B x = 1 \lor \approx^B x = 0$
2. $\approx^B \approx^B x = \approx^B x$
3. $\neg^B x = 1 \leftrightarrow \neg(\approx^B x = 1)$
4. $x \land^B y = 1 \leftrightarrow (\approx^B x = 1 \land \approx^B y = 1)$
5. $x \lor^B y = 1 \leftrightarrow (\approx^B x = 1 \lor \approx^B y = 1)$
6. $x \rightarrow^B y = 1 \leftrightarrow (\approx^B x = 1 \rightarrow \approx^B y = 1)$
7. $x \leftrightarrow^B y = 1 \leftrightarrow (\approx^B x = 1 \leftrightarrow \approx^B y = 1)$
8. $x \oplus^B y = 1 \leftrightarrow (\approx^B x = 1 \oplus \approx^B y = 1)$

PROOF The first three can be proved by straightforward NIND on x, all the others with a simple and direct application of Derived Rule 3.2.3. □

On “AND” and “OR”

LEMMA 3.2.56

1. *(L)* $x \neq \varepsilon \wedge x \neq 0 \wedge x \neq 1 \rightarrow (\gtrdot x \cdot j)\blacktriangleleft = \gtrdot(x\blacktriangleleft) \cdot \backprime\blacktriangleright x$

   (R) $x \neq \varepsilon \wedge x \neq 0 \wedge x \neq 1 \rightarrow \blacktriangleright(\gtrdot x \cdot j) = \gtrdot\blacktriangleright x \cdot j$

2. *(L)* $\backprime x \wedge^B \mathsf{AND}(\gtrdot x \cdot j) = \mathsf{AND}(x) \wedge^B j$ *(R)* $j \wedge^B \mathsf{AND}(x) = \mathsf{AND}(j \cdot x\lessdot) \wedge^B x'$

3. $\backprime x \wedge^B \mathsf{AND}(\gtrdot x) = \mathsf{AND}(x) = \mathsf{AND}(x\lessdot) \wedge^B x'$ *for* $x \neq \varepsilon, 0, 1$

PROOF

1. (L) By Derived Rule 3.2.1: since $\varepsilon = \varepsilon$, the statement is trivially true for $x = \varepsilon$. Assuming $ix \neq 0 \wedge ix \neq 1$, *i.e.*, $x \neq \varepsilon$,

$$\begin{aligned}(\gtrdot(ix) \cdot j)\blacktriangleleft &= (xj)\blacktriangleleft \\ &= x\ ?^{EL}\ (x\blacktriangleleft, x\blacktriangleleft \cdot \backprime(\blacktriangleright x \cdot j)) \\ &= x\ ?^{EL}\ (x\blacktriangleleft, x\blacktriangleleft \cdot \backprime\blacktriangleright x) \\ &= x\ ?^{EL}\ (x\blacktriangleleft\lessdot \cdot x\blacktriangleleft', \gtrdot(i \cdot x\blacktriangleleft) \cdot \backprime\blacktriangleright x) \\ &= x\ ?^{EL}\ \big(\gtrdot(i \cdot x\blacktriangleleft)\lessdot \cdot \backprime((i \cdot x\blacktriangleleft)' \cdot \blacktriangleright x), \gtrdot(i \cdot x\blacktriangleleft) \cdot \backprime\blacktriangleright x\big) \\ &= \gtrdot((ix)\blacktriangleleft) \cdot \backprime\blacktriangleright(ix).\end{aligned}$$

   (A similar proof shows the same theorem with $i \cdot x\lessdot$ in place of $\gtrdot x \cdot i$.)

2. (L) By the preceding lemma and by TIND on x: $\backprime\varepsilon \wedge^B \mathsf{AND}(\gtrdot\varepsilon \cdot j) = 0 = \mathsf{AND}(\varepsilon) \wedge^B j$, $\backprime i \wedge^B \mathsf{AND}(\gtrdot i \cdot j) = i \wedge^B \mathsf{AND}(j) = \mathsf{AND}(i) \wedge^B j$, and assuming that $\backprime(x\blacktriangleleft) \wedge^B \mathsf{AND}(\gtrdot(x\blacktriangleleft) \cdot j) = \mathsf{AND}(x\blacktriangleleft) \wedge^B j$ and $\backprime(\blacktriangleright x) \wedge^B \mathsf{AND}(\gtrdot(\blacktriangleright x) \cdot j) = \mathsf{AND}(\blacktriangleright x) \wedge^B j$, then, for x such that $\gtrdot x \neq \varepsilon$ (the case when it is equal being trivial),

$$\begin{aligned}\backprime x \wedge^B \mathsf{AND}(\gtrdot x \cdot j) &= \backprime x \wedge^B \mathsf{AND}((\gtrdot x \cdot j)\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright(\gtrdot x \cdot j)) \\ &= \backprime(x\blacktriangleleft) \wedge^B \mathsf{AND}(\gtrdot(x\blacktriangleleft) \cdot \backprime\blacktriangleright x) \wedge^B \mathsf{AND}(\gtrdot\blacktriangleright x \cdot j) \\ &= \mathsf{AND}(x\blacktriangleleft) \wedge^B \backprime\blacktriangleright x \wedge^B \mathsf{AND}(\gtrdot\blacktriangleright x \cdot j) \\ &= \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x) \wedge^B j \\ &= \mathsf{AND}(x) \wedge^B j.\end{aligned}$$

3. By preceding lemmas and by Derived Rule 3.2.5 on x: $\backprime(ij) \wedge^B \mathsf{AND}(\gtrdot(ij)) = i \wedge^B \mathsf{AND}(j) = \mathsf{AND}(ij) = \mathsf{AND}(i) \wedge^B j = \mathsf{AND}((ij)\lessdot) \wedge^B (ij)'$, $\backprime(ikj) \wedge^B \mathsf{AND}(\gtrdot(ikj)) = i \wedge^B \mathsf{AND}(kj) = \mathsf{AND}(ikj) = \mathsf{AND}(ik) \wedge^B j = \mathsf{AND}((ikj)\lessdot) \wedge^B (ikj)'$, and under the induction hypotheses

that ${}^\backprime(x\blacktriangleleft) \wedge^B \mathsf{AND}(\gtrdot(x\blacktriangleleft)) = \mathsf{AND}(x) = \mathsf{AND}((x\blacktriangleleft)\lessdot) \wedge^B (x\blacktriangleleft)'$ and ${}^\backprime(\blacktriangleright x) \wedge^B \mathsf{AND}(\gtrdot(\blacktriangleright x)) = \mathsf{AND}(x) = \mathsf{AND}((\blacktriangleright x)\lessdot) \wedge^B (\blacktriangleright x)'$ for $x\blacktriangleleft >^L 1$, then

$$\begin{aligned}
{}^\backprime x \wedge^B \mathsf{AND}(\gtrdot x) &= {}^\backprime x \wedge^B \mathsf{AND}((\gtrdot x)\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright(\gtrdot x)) \\
&= x \mathbin{?^{EL}} \big({}^\backprime(x\blacktriangleleft) \wedge^B \mathsf{AND}(\gtrdot(x\blacktriangleleft)) \wedge^B \mathsf{AND}(\blacktriangleright x), \\
&\qquad {}^\backprime(x\blacktriangleleft) \wedge^B \mathsf{AND}(\gtrdot(x\blacktriangleleft) \cdot {}^\backprime(\blacktriangleright x)) \wedge^B \mathsf{AND}(\gtrdot(\blacktriangleright x))\big) \\
&= x \mathbin{?^{EL}} \big(\mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x), \mathsf{AND}(x\blacktriangleleft) \wedge^B {}^\backprime(\blacktriangleright x) \wedge^B \mathsf{AND}(\gtrdot(\blacktriangleright x))\big) \\
&= x \mathbin{?^{EL}} \big(\mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x), \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x)\big) \\
&= x \mathbin{?^{EL}} (\mathsf{AND}(x), \mathsf{AND}(x)) \\
&= \mathsf{AND}(x) \\
&= x \mathbin{?^{EL}} (\mathsf{AND}(x), \mathsf{AND}(x)) \\
&= x \mathbin{?^{EL}} \big(\mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x), \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x)\big) \\
&= x \mathbin{?^{EL}} \big(\mathsf{AND}((x\blacktriangleleft)\lessdot) \wedge^B (x\blacktriangleleft)' \wedge^B \mathsf{AND}(\blacktriangleright x), \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright x)\big) \\
&= x \mathbin{?^{EL}} \big(\mathsf{AND}((x\blacktriangleleft)\lessdot) \wedge^B \mathsf{AND}((x\blacktriangleleft)' \cdot (\blacktriangleright x)\lessdot) \wedge^B (\blacktriangleright x)', \\
&\qquad \mathsf{AND}(x\blacktriangleleft) \wedge^B \mathsf{AND}((\blacktriangleright x)\lessdot) \wedge^B (\blacktriangleright x)'\big) \\
&= \mathsf{AND}((x\lessdot)\blacktriangleleft) \wedge^B \mathsf{AND}(\blacktriangleright(x\lessdot)) \wedge^B x' \\
&= \mathsf{AND}(x\lessdot) \wedge^B x'. \qquad \square
\end{aligned}$$

On generalizations of CRN—part I

Theorem 3.2.9 $\quad {}_jx = {}_jy \leftrightarrow x \rhd y = \varepsilon = x \lhd y$

Proof $\quad$ By Derived Rule 3.2.3: ${}_j\varepsilon = {}_jy \leftrightarrow \varepsilon = y \leftrightarrow \varepsilon \rhd y = \varepsilon = \varepsilon \lhd y$, ${}_jx = {}_j\varepsilon \leftrightarrow x = \varepsilon \leftrightarrow x \rhd \varepsilon = \varepsilon = x \lhd \varepsilon$, ${}_j(xi) = {}_j(ky) \leftrightarrow {}_jx \cdot j = j \cdot {}_jy \leftrightarrow {}_jx = {}_jy \leftrightarrow x \rhd y = \varepsilon = x \lhd y \leftrightarrow xi \rhd ky = \varepsilon = xi \lhd ky$. $\square$

Claim 3.2.57 $\quad$ *For $f = rCRN[h]$,*

1. ${}_j(f(x, \vec{y})) = {}_jx$
2. $f(x, \vec{y}) \lhd z = f(x \lhd z, \vec{y})$
3. $\mathsf{lb}(f(x, \vec{y}), z) = x \rhd z \mathbin{?^{ZL}} \big((0 \cdot h(\mathsf{lc}(x, z), \vec{y}))', \varepsilon\big) \quad$ *for $x, z \neq \varepsilon$*
4. $\mathsf{lc}(f(x, \vec{y}), z) = f(\mathsf{lc}(x, z), \vec{y})$

Proof

1. By NIND on x: ${}_j(f(\varepsilon, \vec{y})) = {}_j\varepsilon$, ${}_j(f(xi, \vec{y})) = {}_j\big(f(x, \vec{y}) \cdot (0 \cdot h(xi, \vec{y}))'\big) = {}_jx \cdot j = {}_j(xi)$.

2. By Derived Rule 3.2.3: $f(\varepsilon,\vec{y})\lhd z = \varepsilon\lhd z = f(\varepsilon\lhd z,\vec{y})$, $f(x,\vec{y})\lhd\varepsilon = f(x,\vec{y}) = f(x\lhd\varepsilon,\vec{y})$, $f(xi,\vec{y})\lhd zj = \big(f(x,\vec{y})\cdot(0\cdot h(xi,\vec{y}))'\big)\lessdot\lhd z = f(x,\vec{y})\lhd z = f(x\lhd z,\vec{y}) = f(xi\lhd zj,\vec{y})$.

3. First, a straightforward proof by NIND on x shows that ${}^{\backprime}f(x,\vec{y}) = (0\cdot h({}^{\backprime}x,\vec{y}))'$. Now, by NIND on $z\neq\varepsilon$ and the claims above: $\mathsf{lb}(f(x,\vec{y}),j) = {}^{\backprime}\big(\varepsilon\rhd f(x,\vec{y})\big) = (0\cdot h({}^{\backprime}x,\vec{y}))' = (0\cdot h(\mathsf{lc}(x,j),\vec{y}))'$,

$$\begin{aligned}\mathsf{lb}(f(x,\vec{y}),zi) &= {}^{\backprime}\big(z\rhd f(x,\vec{y})\big)\\ &= z\rhd f(x,\vec{y})\,?^{ZL}\,\Big(\varepsilon,\big(f(x,\vec{y})\lhd\gtrdot(z\rhd f(x,\vec{y}))\big)'\Big)\\ &= z\rhd x\,?^{ZL}\,\big(\varepsilon,\big(f(x,\vec{y})\lhd\gtrdot(z\rhd x)\big)'\big)\\ &= z\rhd x\,?^{ZL}\,\big(\varepsilon,f(x\lhd\gtrdot(z\rhd x),\vec{y})'\big)\\ &= z\rhd x\,?^{ZL}\,\big(\varepsilon,f(x\lhd(z\rhd x)\cdot{}^{\backprime}(z\rhd x),\vec{y})'\big)\\ &= z\rhd x\,?^{ZL}\,\big(\varepsilon,(0\cdot h(x\lhd(z\rhd x)\cdot{}^{\backprime}(z\rhd x),\vec{y}))'\big)\\ &= x\rhd zi\,?^{ZL}\,\big((0\cdot h(\mathsf{lc}(x,z)\cdot\mathsf{lb}(x,zi),\vec{y}))',\varepsilon\big)\\ &= x\rhd zi\,?^{ZL}\,\big((0\cdot h(\mathsf{lc}(x,zi),\vec{y}))',\varepsilon\big).\end{aligned}$$

4. By NIND on z: $\mathsf{lc}(f(x,\vec{y}),\varepsilon) = \varepsilon = f(\mathsf{lc}(x,\varepsilon),\vec{y})$, and

$$\begin{aligned}\mathsf{lc}(f(x,\vec{y}),zi) &= \mathsf{lc}(f(x,\vec{y}),z)\cdot\mathsf{lb}(f(x,\vec{y}),zi)\\ &= f(\mathsf{lc}(x,z),\vec{y})\cdot x\rhd zi\,?^{ZL}\,\big((0\cdot h(\mathsf{lc}(x,zi),\vec{y}))',\varepsilon\big)\\ &= z\rhd x\,?^{ZL}\,\big(f(\mathsf{lc}(x,z),\vec{y})\cdot\varepsilon,f(\mathsf{lc}(x,z),\vec{y})\cdot(0\cdot h(\mathsf{lc}(x,zi),\vec{y}))'\big)\\ &= z\rhd x\,?^{ZL}\,\big(f(\mathsf{lc}(x,zi),\vec{y}),f(\mathsf{lc}(x,zi),\vec{y})\big)\\ &= f(\mathsf{lc}(x,zi),\vec{y}). \qquad\square\end{aligned}$$

Lemma 3.2.58

1. *(L)* $y\rhd x = x\rhd y\,?^{ZL}\,(y\rhd x,\varepsilon)$ *(R)* $x\lhd y = x\rhd y\,?^{ZL}\,(x\lhd y,\varepsilon)$
2. *(L)* $(y\lhd(x\rhd y))\rhd x = y\rhd x$ *(R)* $x\lhd((y\lhd x)\rhd y) = x\lhd y$

Proof

1. (L) Immediate from the fact that $x\rhd y\neq\varepsilon\rightarrow y\rhd x=\varepsilon$.

2. (L) By NIND on y: $(\varepsilon\lhd(x\rhd\varepsilon))\rhd x = (\varepsilon\lhd\varepsilon)\rhd x = \varepsilon\rhd x$,

$$\begin{aligned}(yj\lhd(x\rhd yj))\rhd x &= x\rhd yj\,?^{ZL}\,\big((yj\lhd\varepsilon)\rhd x,(yj\lhd((x\rhd y)\cdot j))\rhd x\big)\\ &= x\rhd yj\,?^{ZL}\,\big(yj\rhd x,(y\lhd(x\rhd y))\rhd x\big)\\ &= x\rhd yj\,?^{ZL}\,(yj\rhd x,y\rhd x) = x\rhd yj\,?^{ZL}\,(yj\rhd x,\varepsilon)\\ &= x\rhd yj\,?^{ZL}\,(yj\rhd x,yj\rhd x) = yj\rhd x. \qquad\square\end{aligned}$$

Claim 3.2.59

1. *(L)* $\mathsf{lp}_j(x, \mathsf{max}^L(x,y)) = \mathsf{lp}_j(x,y)$ *(R)* $\mathsf{rp}_j(x, \mathsf{max}^L(x,y)) = \mathsf{rp}_j(x,y)$
2. *(L)* $\mathsf{lb}\big(\mathsf{lp}_0(xi, \mathsf{max}^L(xi, yj)), \mathsf{max}^L(xi, yj)\big) = i$
   (R) $\mathsf{rb}\big(\mathsf{rp}_0(ix, \mathsf{max}^L(ix, jy)), \mathsf{max}^L(ix, jy)\big) = i$

Proof

1. (L): $\mathsf{lp}_j(x, \mathsf{max}^L(x,y)) = x \rhd y \, ?^{ZL} \big({}_j(x \lhd x) \cdot x, {}_j(y \lhd x) \cdot x\big) = x \rhd y \, ?^{ZL} \big(\varepsilon \cdot x, {}_j(y \lhd x) \cdot x\big) = x \rhd y \, ?^{ZL} \big({}_j(y \lhd x) \cdot x, {}_j(y \lhd x) \cdot x\big) = {}_j(y \lhd x) \cdot x = \mathsf{lp}_j(x,y)$.
2. (L):

$$\begin{aligned}
&\mathsf{lb}\big(\mathsf{lp}_0(xi, \mathsf{max}^L(xi, yj)), \mathsf{max}^L(xi, yj)\big)\\
&= \mathsf{lb}\big(\mathsf{lp}_0(x, \mathsf{max}^L(x,y)) \cdot i, \mathsf{max}^L(x,y) \cdot x \rhd y \, ?^{ZL} \, (i,j)\big)\\
&= {}^{\backprime}\big(\mathsf{max}^L(x,y) \rhd ({}_0(\mathsf{max}^L_x(x,y) \lhd x) \cdot x \cdot i)\big)\\
&= x \rhd y \, ?^{ZL} \, \big({}^{\backprime}\big(x \rhd ({}_0(x \lhd x) \cdot xi)\big), {}^{\backprime}\big(y \rhd ({}_0(y \lhd x) \cdot xi)\big)\big)\\
&= x \rhd y \, ?^{ZL} \, \big({}^{\backprime}(x \rhd (\varepsilon \cdot xi)), {}^{\backprime}\big(y \rhd (x \rhd {}_0 y) \cdot (y \lhd (x \rhd {}_0 y)) \rhd xi\big)\big)\\
&= x \rhd y \, ?^{ZL} \, \big({}^{\backprime} i, {}^{\backprime}(\varepsilon \cdot (y \lhd (x \rhd y)) \lhd x \cdot i)\big)\\
&= x \rhd y \, ?^{ZL} \, \big(i, {}^{\backprime}(y \rhd x \cdot i)\big)\\
&= x \rhd y \, ?^{ZL} \, \big(i, {}^{\backprime}(\varepsilon \cdot i)\big) = x \rhd y \, ?^{ZL} \, (i,i) = i \qquad \square
\end{aligned}$$

Theorem 3.2.10

$$\begin{aligned}
&\ell CRN_m[h](x_1, \ldots, x_m, \vec{y})\\
&\quad = x_1 \cdot \ldots \cdot x_m \, ?^{ZL} \Big(\varepsilon, {}^{\backprime}\big(h\big(\mathsf{lp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)), \ldots, \mathsf{lp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m)), \vec{y}\big) \cdot 0\big)\\
&\qquad\qquad \cdot \ell CRN_m[h]\big({\gtrdot}\mathsf{lp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)), \ldots, {\gtrdot}\mathsf{lp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m)), \vec{y}\big)\Big)\\
&r CRN_m[h](x_1, \ldots, x_m, \vec{y})\\
&\quad = x_1 \cdot \ldots \cdot x_m \, ?^{ZL} \Big(\varepsilon, r CRN_m[h]\big(\mathsf{rp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)){\lessdot}, \ldots, \mathsf{rp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m)){\lessdot}, \vec{y}\big)\\
&\qquad\qquad \cdot \big(0 \cdot h\big(\mathsf{rp}_0(x_1, \mathsf{max}^L_m(\vec{x}_m)), \ldots, \mathsf{rp}_0(x_m, \mathsf{max}^L_m(\vec{x}_m)), \vec{y}\big)\big)'\Big)
\end{aligned}$$

Proof From the fact that $\mathsf{max}^L_m(\vec{x}_m) = \varepsilon \leftrightarrow x_1 \cdot \ldots \cdot x_m = \varepsilon$ (easily proved by Derived Rule 3.2.3), the theorem follows from Claim 3.2.59 by a straightforward application of Derived Rule 3.2.3, generalized to m variables. $\square$

Claim 3.2.60

1. $\mathsf{and}^B_m(x_1i_1,\dots,x_mi_m)=\mathsf{and}^B_m(\vec{x}_m)\cdot\mathsf{and}^B_m(\vec{i}_m)$

2. ${}_jx_1=\cdots={}_jx_m\wedge{}_jy_1=\cdots={}_jy_m\rightarrow\mathsf{and}^B_m(x_1y_1,\dots,x_my_m)=\mathsf{and}^B_m(\vec{x}_m)\cdot\mathsf{and}^B_m(\vec{y}_m)$.

3. $\mathsf{and}^B_m(\vec{x}_m)^{\lessdot}=\mathsf{and}^B_m(x_1{}^{\lessdot},\dots,x_m{}^{\lessdot})$

4. $\mathsf{and}^B_m(\vec{x}_m)\lhd y=\mathsf{and}^B_m(x_1\lhd y,\dots,x_m\lhd y)$

Proof

1. By a straightforward application of Derived Rule 3.2.3, generalized to m variables.
2. Again, by a straightforward application of Derived Rule 3.2.3, generalized to m variables, together with the previous claim.
3. From the first claim, by a straightforward generalized NIND.
4. Directly from the preceding claim, with a straightforward NIND on y. □

Theorem 3.2.11 $\neg^B\mathsf{AND}(x)=\mathsf{OR}(\mathsf{not}^B(x))$ *and* $\neg^B\mathsf{OR}(x)=\mathsf{AND}(\mathsf{not}^B(x))$ *for* $x\neq\varepsilon$

Proof (We prove only the first statement, the second one being almost identical.) By TIND on x: $\neg^B\mathsf{AND}(i)=\neg^Bi=\mathsf{OR}(\neg^Bi)=\mathsf{OR}(\mathsf{not}^B(i))$,

$$\begin{aligned}\neg^B\mathsf{AND}(x)&=\neg^B\big(\mathsf{AND}(x\blacktriangleleft)\wedge^B\mathsf{AND}(\blacktriangleright x)\big)\\&=\neg^B\mathsf{AND}(x\blacktriangleleft)\vee^B\neg^B\mathsf{AND}(\blacktriangleright x)\\&=\mathsf{OR}(\mathsf{not}^B(x\blacktriangleleft))\vee^B\mathsf{OR}(\mathsf{not}^B(\blacktriangleright x))\\&=\mathsf{OR}(\mathsf{not}^B(x)\blacktriangleleft)\vee^B\mathsf{OR}(\blacktriangleright\mathsf{not}^B(x))=\mathsf{OR}(\mathsf{not}^B(x)).\qquad\square\end{aligned}$$

On generalizations of CRN—part II

Claim 3.2.61

1. $\blacktriangleright\mathsf{pow}^L(y)=\mathsf{pow}^L(\blacktriangleright y)=\mathsf{pow}^L(y)\blacktriangleleft$ *(for* $y\neq\varepsilon,0,1$*)*

2. ${}_1\mathsf{pow}^L(y)=\mathsf{pow}^L({}_1y)=\mathsf{pow}^L(y)$

3. $\mathsf{pow}^L(\mathsf{pow}^L(y))=\mathsf{pow}^L(y)$

4. $\mathsf{pow}^L(y)\rhd y=\varepsilon$

5. $\mathsf{ispow}^L(\mathsf{pow}^L(y))=\varepsilon$

PROOF All can be proved by very simple applications of TIND. We give the proof of the third statement as an illustration: $\mathsf{pow}^L(\mathsf{pow}^L(\varepsilon)) = \mathsf{pow}^L(\varepsilon)$, $\mathsf{pow}^L(\mathsf{pow}^L(i)) = \mathsf{pow}^L(1) = \mathsf{pow}^L(i)$, $\mathsf{pow}^L(\mathsf{pow}^L(y)) = \mathsf{pow}^L(\blacktriangleright\mathsf{pow}^L(y)) \cdot \mathsf{pow}^L(\blacktriangleright\mathsf{pow}^L(y)) = \mathsf{pow}^L(\mathsf{pow}^L(\blacktriangleright y)) \cdot \mathsf{pow}^L(\mathsf{pow}^L(\blacktriangleright y)) = \mathsf{pow}^L(\blacktriangleright y) \cdot \mathsf{pow}^L(\blacktriangleright y) = \mathsf{pow}^L(y)$. $\square$

CLAIM 3.2.62

1. $\varepsilon \# y = \varepsilon = x \# \varepsilon$
2. (L) $_1(xi \# y) = {}_1(x \# y) \cdot {}_1y$ (R) $_1(x \# yi) = {}_1(x \# y) \cdot {}_1x$
3. (L) $_1((x \cdot y) \# z) = {}_1(x \# z) \cdot {}_1(y \# z)$ (R) $_1(x \# (y \cdot z)) = {}_1(x \# y) \cdot {}_1(x \# z)$
4. $_1(x \# y) = {}_1(y \# x)$

PROOF Again, all these statements can be proved by very simple applications of NIND or TIND; we give the proof of the third statement (for the left case) as an illustration: $_1((x{\cdot}\varepsilon)\#z) = {}_1(x\#z) = {}_1(x\#z){\cdot}{}_1(\varepsilon\#z)$, $_1((x{\cdot}yi)\#z) = {}_1((x{\cdot}y)\#z){\cdot}{}_1z = {}_1(x\#z){\cdot}{}_1(y\#z){\cdot}{}_1z = {}_1(x\#z){\cdot}{}_1(yi\#z)$. $\square$

CLAIM 3.2.63

1. $\mathsf{powdiv}^L(x, y) = \mathsf{powdiv}^L({}_1x, {}_1y)$
2. $\mathsf{powdiv}^L(x, y) = \mathsf{powdiv}^L(x, \mathsf{pow}^L(y))$
3. $x \rhd \mathsf{pow}^L(y) \neq \varepsilon \to \mathsf{powdiv}^L(x, y) = \varepsilon$
4. $\mathsf{powdiv}^L(\mathsf{pow}^L(y) \# z, y) = y \ ?^{ZL}\ (\varepsilon, {}_1z)$

PROOF

1. By a simple TIND on y.
2. By a simple TIND on y.
3. First, we prove that $x \rhd \mathsf{pow}^L(y) \neq \varepsilon \to x\blacktriangleleft \rhd \mathsf{pow}^L(\blacktriangleright y) \neq \varepsilon$ by proving the contrapositive by TIND on y: $x \rhd \mathsf{pow}^L(\varepsilon) = \varepsilon$ (so the statement is vacuously true), $x\blacktriangleleft \rhd \mathsf{pow}^L(\blacktriangleright 1) = \varepsilon \to x\blacktriangleleft \neq \varepsilon \to x \neq \varepsilon \to x \rhd \mathsf{pow}^L(1) = \varepsilon$, and assuming that $x\blacktriangleleft \rhd \mathsf{pow}^L(\blacktriangleright y) = \varepsilon$, then

$$\begin{aligned} x \rhd y &= (x\blacktriangleleft \cdot \blacktriangleright x) \rhd (\mathsf{pow}^L(y)\blacktriangleleft \cdot \blacktriangleright\mathsf{pow}^L(y)) \\ &= \blacktriangleright x \rhd (x\blacktriangleleft \rhd (\mathsf{pow}^L(\blacktriangleright y) \cdot \mathsf{pow}^L(\blacktriangleright y))) \\ &= \blacktriangleright x \rhd ((x\blacktriangleleft \rhd \mathsf{pow}^L(\blacktriangleright y)) \cdot (x\blacktriangleleft \lhd \mathsf{pow}^L(\blacktriangleright y)) \rhd \mathsf{pow}^L(\blacktriangleright y)) \end{aligned}$$

$$\begin{aligned}
&= \blacktriangleright x \rhd ((x\blacktriangleleft \lhd \mathsf{pow}^L(\blacktriangleright y)) \rhd \mathsf{pow}^L(\blacktriangleright y))\\
&= (x\blacktriangleleft \lhd \mathsf{pow}^L(\blacktriangleright y)) \rhd (\blacktriangleright x \rhd \mathsf{pow}^L(\blacktriangleright y))\\
&= (x\blacktriangleleft \lhd \mathsf{pow}^L(\blacktriangleright y)) \rhd \varepsilon = \varepsilon
\end{aligned}$$

(where we have used the fact that $(z \cdot y) \rhd x = y \rhd (z \rhd x) = z \rhd (y \rhd x)$, which is easy to prove by NIND on z). The result then follows by a simple application of TIND.

4. By TIND on y: $\mathsf{powdiv}^L(\mathsf{pow}^L(\varepsilon) \# z, \varepsilon) = \varepsilon = \varepsilon\ ?^{ZL}\ (\varepsilon, {}_1z)$, $\mathsf{powdiv}^L(\mathsf{pow}^L(i) \# z, i) = \mathsf{powdiv}^L(z, i) = {}_1z = i\,?^{ZL}(\varepsilon, {}_1z)$, $\mathsf{powdiv}^L(\mathsf{pow}^L(y) \# z, y) = \mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z)\blacktriangleleft, \blacktriangleright y) = \mathsf{powdiv}^L(\mathsf{pow}^L(y)\blacktriangleleft \# z, \blacktriangleright y) = \mathsf{powdiv}^L(\mathsf{pow}^L(\blacktriangleright y) \# z, \blacktriangleright y) = {}_1z = y\ ?^{ZL}\ (\varepsilon, {}_1z)$. $\square$

Claim 3.2.65

1. $x \rhd (\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x, y)) = \varepsilon$

2. $y \neq \varepsilon \to \mathsf{powmod}^L(x, y) \rhd \mathsf{pow}^L(y) \neq \varepsilon$

3. $y \neq \varepsilon \to \mathsf{powdiv}^L(x1, y) = \mathsf{powdiv}^L(x, y) \cdot \big((\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y)\ ?^{ZL}\ (1, \varepsilon)\big)$
   $y \neq \varepsilon \to \mathsf{powmod}^L(x1, y) = (\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y)\ ?^{ZL}\ \big(\varepsilon, \mathsf{powmod}^L(x, y) \cdot 1\big)$

4. $\mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) = y\ ?^{ZL}\ (\varepsilon, {}_1z) \cdot \mathsf{powdiv}^L(x, y) \wedge$
   $\mathsf{powmod}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) = \mathsf{powmod}^L(x, y)$

5. $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon \to \mathsf{powdiv}^L(x, y) = \mathsf{powdiv}^L(x \lhd \mathsf{pow}^L(y), y) \cdot 1$
   $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon \to \mathsf{powmod}^L(x, y) = \mathsf{powmod}^L(x \lhd \mathsf{pow}^L(y), y)$

Proof

1. By TIND on y (with $h_\ell = h_r = \blacktriangleleft$): $x \rhd (\mathsf{pow}^L(\varepsilon) \# \mathsf{powdiv}^L(x, \varepsilon)) = x \rhd \varepsilon = \varepsilon$, $x \rhd (\mathsf{pow}^L(i) \# \mathsf{powdiv}^L(x, i)) = x \rhd (1 \# {}_1x) = \varepsilon$,

$$\begin{aligned}
&x \rhd (\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x, y))\\
&\quad= x \rhd (\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\\
&\quad= (x\blacktriangleleft \cdot \blacktriangleright x) \rhd \big((\mathsf{pow}^L(\blacktriangleright y) \cdot \mathsf{pow}^L(\blacktriangleright y)) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y)\big)\\
&\quad= \blacktriangleright x \rhd \Big(\big(x\blacktriangleleft \rhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\big)\cdot\\
&\qquad\qquad \big(x\blacktriangleleft \lhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\big) \rhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\Big)\\
&\quad= (x\blacktriangleleft \lhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))) \rhd \big(\blacktriangleright x \rhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\big)\\
&\quad= (x\blacktriangleleft \lhd (\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))) \rhd \varepsilon = \varepsilon.
\end{aligned}$$

2. By TIND on $y \neq \varepsilon$: $\mathsf{powmod}^L(x,1) \rhd \mathsf{pow}^L(1) = \varepsilon \rhd 1 \neq \varepsilon$,

$$\begin{aligned}
&\mathsf{powmod}^L(x,y) \rhd \mathsf{pow}^L(y)\\
&= \big((\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(x,y)) \rhd {}_1x\big) \rhd \mathsf{pow}^L(y)\\
&= \Big(\big((\mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{pow}^L(\blacktriangleright y)) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y)\big) \rhd ({}_1x\blacktriangleleft \cdot {}_1\blacktriangleright x)\Big)\\
&\qquad \rhd (\mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{pow}^L(\blacktriangleright y))\\
&= \Big(\big((\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\cdot(\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y))\big)\\
&\qquad \rhd ({}_1x\blacktriangleleft \cdot {}_1x\blacktriangleleft \cdot (x \mathbin{?^{EL}} (\varepsilon,1)))\Big) \rhd (\mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{pow}^L(\blacktriangleright y))\\
&= \Big(\big((\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y)) \rhd {}_1x\blacktriangleleft\big)\cdot\\
&\qquad \big((\mathsf{pow}^L(\blacktriangleright y) \# \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y)) \rhd {}_1x\blacktriangleleft \cdot (x \mathbin{?^{EL}} (\varepsilon,1))\big)\Big) \rhd (\mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{pow}^L(\blacktriangleright y))\\
&= \Big(\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot\big(\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot(x \mathbin{?^{EL}} (\varepsilon,1))\big)\Big) \rhd (\mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{pow}^L(\blacktriangleright y))\\
&= \mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y) \rhd \mathsf{pow}^L(\blacktriangleright y)\cdot(\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot(x \mathbin{?^{EL}} (\varepsilon,1))) \rhd \mathsf{pow}^L(\blacktriangleright y) \neq \varepsilon
\end{aligned}$$

(where we have used the fact that $y \lhd x = \varepsilon \wedge w \lhd z = \varepsilon \rightarrow wy \rhd {}_1z_1x = w \rhd {}_1z \cdot y \rhd {}_1x$, which is a direct consequence of Claim 3.2.54 and the fact that $zy \rhd x = y \rhd (z \rhd x)$).

3. We prove the first statement by TIND on $y \neq \varepsilon$: $\mathsf{powdiv}^L(x1,1) = {}_1x1 = {}_1x \cdot (1 \rhd 1 \mathbin{?^{ZL}} (1,\varepsilon)) = \mathsf{powdiv}^L(x,1)\cdot((\mathsf{powmod}^L(x,1)\cdot 1) \rhd 1 \mathbin{?^{ZL}} (1,\varepsilon))$, and before proving the inductive case, we can show by TIND on $y \neq \varepsilon$ that

$$\begin{aligned}
&(\mathsf{powmod}^L(x,y)\cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1,\varepsilon)\\
&= x \mathbin{?^{EL}} \Big(\big((\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y)\cdot \mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y) \rhd \mathsf{pow}^L(\blacktriangleright y)\big) \mathbin{?^{ZL}} (1,\varepsilon),\\
&\qquad \big((\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y)\cdot(\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y)\big) \mathbin{?^{ZL}} (1,\varepsilon)\Big)\\
&= x \mathbin{?^{EL}} \Big(\varepsilon, (\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y) \mathbin{?^{ZL}} (1,\varepsilon)\Big)
\end{aligned}$$

(the proof is similar to that of the preceding claim), so that

$$\begin{aligned}
&\mathsf{powdiv}^L(x1,y)\\
&= \mathsf{powdiv}^L((x1)\blacktriangleleft, \blacktriangleright y)\\
&= x \mathbin{?^{EL}} \big(\mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y), \mathsf{powdiv}^L(x\blacktriangleleft \cdot 1, \blacktriangleright y)\big)\\
&= x \mathbin{?^{EL}} \big(\mathsf{powdiv}^L(x,y), \mathsf{powdiv}^L(x\blacktriangleleft, \blacktriangleright y)\cdot(\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y) \mathbin{?^{ZL}} (1,\varepsilon)\big)\\
&= \mathsf{powdiv}^L(x,y)\cdot\big(x \mathbin{?^{EL}} \big(\varepsilon, (\mathsf{powmod}^L(x\blacktriangleleft, \blacktriangleright y)\cdot 1) \rhd \mathsf{pow}^L(\blacktriangleright y) \mathbin{?^{ZL}} (1,\varepsilon)\big)\big)\\
&= \mathsf{powdiv}^L(x,y)\cdot\big((\mathsf{powmod}^L(x,y)\cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1,\varepsilon)\big).
\end{aligned}$$

As for the second statement, it follows directly from the first by the definition of powmod^L and the fact that $\mathsf{powmod}^L(x,y) \rhd \mathsf{pow}^L(y) \neq \varepsilon \rightarrow \mathsf{pow}^L(y) \rhd (\mathsf{powmod}^L(x,y)\cdot 1) = \varepsilon$.

4. By NIND on x: $\mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot \varepsilon, y) = y \mathbin{?^{ZL}} (\varepsilon, {}_1z) = y \mathbin{?^{ZL}} (\varepsilon, {}_1z) \cdot \mathsf{powdiv}^L(\varepsilon, y) \wedge$ $\mathsf{powmod}^L((\mathsf{pow}^L(y) \# z) \cdot \varepsilon, y) = \varepsilon = \mathsf{powmod}^L(\varepsilon, y)$,

$$\begin{aligned}
&\mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot xi, y)\\
&\quad = \mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) \cdot\\
&\qquad \big((\mathsf{powmod}^L((\mathsf{pow}^L(y) \# z) \cdot x, y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&\quad = (y \mathbin{?^{ZL}} (\varepsilon, {}_1z)) \cdot \mathsf{powdiv}^L(x, y) \cdot \big((\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&\quad = (y \mathbin{?^{ZL}} (\varepsilon, {}_1z)) \cdot \mathsf{powdiv}^L(x1, y),\\
&\mathsf{powmod}^L((\mathsf{pow}^L(y) \# z) \cdot xi, y)\\
&\quad = \big(\mathsf{pow}^L(y) \# \mathsf{powdiv}^L((\mathsf{pow}^L(y) \# z) \cdot xi, y)\big) \rhd {}_1\big((\mathsf{pow}^L(y) \# z) \cdot xi\big)\\
&\quad = \big(\mathsf{pow}^L(y) \# ((y \mathbin{?^{ZL}} (\varepsilon, {}_1z)) \cdot \mathsf{powdiv}^L(xi, y))\big) \rhd {}_1(\mathsf{pow}^L(y) \# z) \cdot {}_1(xi)\\
&\quad = (\mathsf{pow}^L(y) \# (y \mathbin{?^{ZL}} (\varepsilon, {}_1z))) \cdot (\mathsf{pow}^L(y) \# \mathsf{powdiv}^L(xi, y)) \rhd (\mathsf{pow}^L(y) \# {}_1z) \cdot {}_1(xi)\\
&\quad = \mathsf{pow}^L(y) \# \mathsf{powdiv}^L(xi, y) \rhd {}_1(xi)\\
&\quad = \mathsf{powmod}^L(xi, y).
\end{aligned}$$

5. By NIND on x: $y \neq \varepsilon \wedge \varepsilon \rhd \mathsf{pow}^L(y) = \varepsilon$ is false so the statement is vacuously true, and $y \neq \varepsilon \wedge xi \rhd \mathsf{pow}^L(y) = \varepsilon \to (y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon) \vee (y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = 1)$, so we prove the statement by cases.

   First, if $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = \varepsilon$ (which implies that ${}_1(x1 \lhd \mathsf{pow}^L(y)) = {}_1(x \lhd \mathsf{pow}^L(y)) \cdot 1$),

$$\begin{aligned}
&\mathsf{powdiv}^L(x1, y)\\
&\quad = \mathsf{powdiv}^L(x, y) \cdot \big((\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&\quad = \mathsf{powdiv}^L(x \lhd \mathsf{pow}^L(y), y) \cdot 1 \cdot \big((\mathsf{powmod}^L(x \lhd \mathsf{pow}^L(y)) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&\quad = \mathsf{powdiv}^L((x \lhd \mathsf{pow}^L(y)) \cdot 1, y) \cdot 1 = \mathsf{powdiv}^L(x1 \lhd \mathsf{pow}^L(y), y) \cdot 1,\\
&\mathsf{powmod}^L(x1, y)\\
&\quad = (\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (\varepsilon, \mathsf{powmod}^L(x, y) \cdot 1)\\
&\quad = (\mathsf{powmod}^L(x \lhd \mathsf{pow}^L(y), y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (\varepsilon, \mathsf{powmod}^L(x \lhd \mathsf{pow}^L(y), y) \cdot 1)\\
&\quad = \mathsf{powmod}^L((x \lhd \mathsf{pow}^L(y)) \cdot 1, y) = \mathsf{powmod}^L(x1 \lhd \mathsf{pow}^L(y), y).
\end{aligned}$$

   Second, if $y \neq \varepsilon \wedge x \rhd \mathsf{pow}^L(y) = 1$ (which implies that ${}_1(x1 \lhd \mathsf{pow}^L(y)) = \varepsilon$),

$$\begin{aligned}
\mathsf{powdiv}^L(x1, y) &= \mathsf{powdiv}^L(x, y) \cdot \big((\mathsf{powmod}^L(x, y) \cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&= \varepsilon \cdot \big(({}_1x1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (1, \varepsilon)\big)\\
&= 1 = \mathsf{powdiv}^L(\varepsilon, y) \cdot 1\\
&= \mathsf{powdiv}^L(x1 \lhd \mathsf{pow}^L(y), y) \cdot 1,
\end{aligned}$$

$$\begin{aligned}\mathsf{powmod}^L(x1,y) &= (\mathsf{powmod}^L(x,y)\cdot 1) \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (\varepsilon, \mathsf{powmod}^L(x,y)\cdot 1)\\ &= {}_1x1 \rhd \mathsf{pow}^L(y) \mathbin{?^{ZL}} (\varepsilon, {}_1x1)\\ &= \varepsilon = \mathsf{powmod}^L(\varepsilon,y)\\ &= \mathsf{powmod}^L(x1 \lhd \mathsf{pow}^L(y), y). \qquad \square\end{aligned}$$

On "$=^N$" and "$<^N$"

Theorem 3.2.13

1. *$x =^N \varepsilon \leftrightarrow x = {}_0x$*
2. *$x =^N y \leftrightarrow \mathsf{lp}_0(x,y) = \mathsf{lp}_0(y,x)$*

Proof

1. By a straightforward NIND on x.
2. By the preceding property and Derived Rule 3.2.3: $x =^N \varepsilon \leftrightarrow x = {}_0x \leftrightarrow \mathsf{lp}_0(x,\varepsilon) = \mathsf{lp}_0(\varepsilon,x)$ (and similarly for $\varepsilon =^N y$), $xi =^N yj \leftrightarrow x =^N y \wedge^B (i \leftrightarrow^B j) \leftrightarrow \mathsf{lp}_0(x,y) = \mathsf{lp}_0(y,x) \wedge i = j \leftrightarrow \mathsf{lp}_0(x,y)\cdot i = \mathsf{lp}_0(y,x)\cdot j \leftrightarrow \mathsf{lp}_0(xi,yj) = \mathsf{lp}_0(yj,xi)$. $\square$

Claim 3.2.67

1. *$x0 =^N y0 \leftrightarrow x =^N y \leftrightarrow x1 =^N y1$*
2. *$\neg^B(x0 =^N y1)$*
3. *$\neg^B(x1 =^N y0)$*

Proof Directly from Theorem 3.2.13: $xi =^N yj \leftrightarrow \mathsf{lp}_0(xi,yj) = \mathsf{lp}_0(yj,xi) \leftrightarrow \mathsf{lp}_0(x,y)\cdot i = \mathsf{lp}_0(y,x)\cdot j \leftrightarrow \mathsf{lp}_0(x,y) = \mathsf{lp}_0(y,x) \wedge i = j \leftrightarrow x =^N y \wedge i = j$. $\square$

Claim 3.2.68

1. *$x0 <^N y0 = x <^N y \vee^B (x =^N y \wedge^B 0 <^B 0) = x <^N y$*
2. *$x0 <^N y1 = x <^N y \vee^B (x =^N y \wedge^B 0 <^B 1) = x \leq^N y$*
3. *$x1 <^N y0 = x <^N y \vee^B (x =^N y \wedge^B 1 <^B 0) = x <^N y$*
4. *$x1 <^N y1 = x <^N y \vee^B (x =^N y \wedge^B 1 <^B 1) = x <^N y$*

Proof From Theorem 3.2.10, we have that $xi <^N yj = x <^N y \vee^B (x =^N y \wedge^B i <^B j)$, so the theorem follows directly from the preceding claim. $\square$

CLAIM 3.2.69

1. $\neg^B(x <^N \varepsilon)$

2. $\neg^B(x <^N x)$

3. $\neg^B(\varepsilon <^N {}_0x)$

PROOF A straightforward proof by NIND, using the preceding claims. □

LEMMA 3.2.72

1. $x =^N y = 0x =^N y = x =^N 0y = 0x =^N 0y$

2. $x <^N y = 0x <^N 0y$

3. $x <^N y = 0x <^N y = x <^N 0y$

PROOF

1. Direct from the fact that $(\mathsf{lp}_0(0x, y) = \mathsf{lp}_0(x, y)) \ \vee \ (\mathsf{lp}_0(0x, y) = 0 \cdot \mathsf{lp}_0(x, y))$ (which can easily be proved by cases depending on the length of $x \rhd y$), and by Derived Rule 3.2.3.
2. By a simple application of Derived Rule 3.2.3 and the first claim (we show only the inductive step, the base cases being just as simple): $0xi <^N 0yj = 0x <^N 0y \vee^B (0x =^N 0y \wedge^B i <^B j) = x <^N y \vee^B (x =^N y \wedge^B i <^B j) = xi <^N yj$.
3. By the second claim:

$$\begin{aligned} 0x <^N y &= 0x \lhd y \,?^{ZL} \big({}_0(y \lhd 0x) \cdot 0x <^N y, 0x <^N {}_0(0x \lhd y) \cdot y\big) \\ &= x \rhd y \,?^{ZL} \big(0x <^N 0 \cdot {}_0(x \lhd y) \cdot y, {}_0(y \lhd x) \lessdot \cdot 0 \cdot x <^N y\big) \\ &= x \rhd y \,?^{ZL} \big(x <^N {}_0(x \lhd y) \cdot y, {}_0(y \lhd x) \cdot x <^N y\big) \\ &= x <^N y \end{aligned}$$

(and similarly for $x <^N 0y$). □

THEOREM 3.2.15 $x =^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x =^N y \wedge y >^N z \rightarrow x >^N z$

PROOF Follows directly from the (already proven) facts that $x =^N y \leftrightarrow \mathsf{lp}_0(x, \mathsf{max}^L_3(x, y, z)) = \mathsf{lp}_0(y, \mathsf{max}^L_3(x, y, z))$ and $y <^N z \leftrightarrow \mathsf{lp}_0(y, \mathsf{max}^L_3(x, y, z)) <^N \mathsf{lp}_0(z, \mathsf{max}^L_3(x, y, z))$. □

COROLLARY 3.2.73 $x =^N y \wedge y \leq^N z \rightarrow x \leq^N z$ *and* $x =^N y \wedge y \geq^N z \rightarrow x \geq^N z$

PROOF Directly from the theorem. □

THEOREM 3.2.16 $x <^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x >^N y \wedge y >^N z \rightarrow x >^N z$

PROOF By Derived Rule 3.2.3: the base cases for $z = \varepsilon$ and $y = \varepsilon$ are trivially true since $\varepsilon \not<^N x$, while the third base case $\varepsilon <^N y \wedge y <^N z \rightarrow \varepsilon <^N z$ is itself proved by Derived Rule 3.2.3: the two base cases are again trivially true, and $\varepsilon <^N yj \wedge^B yj <^N zk = \big(\varepsilon <^N y \vee^B (\varepsilon =^N y \wedge^B 0 <^B j)\big) \wedge^B \big(y <^N z \vee^B (y =^N z \wedge^B j <^B k)\big) = (\varepsilon <^N y \wedge^B y <^N z) \vee^B (\varepsilon <^N y \wedge^B y =^N z \wedge^B j <^B k) \vee^B (\varepsilon =^N y \wedge^B 0 <^B j \wedge^B y <^N z) \vee^B (\varepsilon =^N y \wedge^B 0 <^B j \wedge^B y =^N z \wedge^B j <^B k) = \varepsilon <^N z \vee^B (\varepsilon =^N z \wedge^B 0 <^B k) = \varepsilon <^N zk$ (the general inductive step is almost identical). $\square$

COROLLARY 3.2.74 $x \leq^N y \wedge y <^N z \rightarrow x <^N z$ *and* $x \geq^N y \wedge y >^N z \rightarrow x >^N z$

PROOF Directly from the theorem. $\square$

COROLLARY 3.2.75 $x \leq^N y \wedge y \leq^N z \rightarrow x \leq^N z$ *and* $x \geq^N y \wedge y \geq^N z \rightarrow x \geq^N z$

PROOF Directly from the theorem. $\square$

On "$|\cdot|$" and "succ^N"

CLAIM 3.2.76

$$\begin{aligned} \mathsf{succ}^N(\varepsilon) &= 1 \\ \mathsf{succ}^N(x0) &= 0x1 \\ \mathsf{succ}^N(x1) &= \mathsf{succ}^N(x) \cdot 0 \end{aligned}$$

PROOF Simple proofs by NIND (proving the relevant properties first for the auxiliary function cuss^N, and then for succ^N). $\square$

CLAIM 3.2.77

$$\begin{aligned} \mathsf{succ}^N(0x) &= 0 \cdot \mathsf{succ}^N(x) \\ \mathsf{succ}^N(1x) &= {}^{\backprime}\mathsf{succ}^N(x) \cdot \neg^B {}^{\backprime}\mathsf{succ}^N(x) \cdot \gtrdot\mathsf{succ}^N(x) \\ {}^{\backprime}\mathsf{succ}^N(x) &= \mathsf{AND}(1x) \\ \mathsf{succ}^N(x) &= \mathsf{AND}(1x)\ ?^B \left(1 \cdot {}_0x, 0 \cdot \gtrdot\mathsf{succ}^N(x)\right) \end{aligned}$$

PROOF All the proofs are simple, but we will illustrate them by proving the second property, by NIND on x: $\mathsf{succ}^N(1\varepsilon) = 10 = 1 \cdot \neg^B 1 \cdot \gtrdot 1 = {}^{\backprime}\mathsf{succ}^N(\varepsilon) \cdot \neg^B {}^{\backprime}\mathsf{succ}^N(\varepsilon) \cdot \gtrdot\mathsf{succ}^N(\varepsilon)$, $\mathsf{succ}^N(1x0) = 01x1 = {}^{\backprime}(0x1) \cdot \neg^B {}^{\backprime}(0x1) \cdot \gtrdot(0x1) = {}^{\backprime}\mathsf{succ}^N(x0) \cdot \neg^B {}^{\backprime}\mathsf{succ}^N(x0) \cdot \gtrdot\mathsf{succ}^N(x0)$, $\mathsf{succ}^N(1x1) = \mathsf{succ}^N(1x) \cdot 0 = {}^{\backprime}\mathsf{succ}^N(x) \cdot \neg^B {}^{\backprime}\mathsf{succ}^N(x) \cdot \gtrdot\mathsf{succ}^N(x) \cdot 0 = {}^{\backprime}(\mathsf{succ}^N(x) \cdot 0) \cdot \neg^B {}^{\backprime}(\mathsf{succ}^N(x) \cdot 0) \cdot \gtrdot(\mathsf{succ}^N(x) \cdot 0) = {}^{\backprime}\mathsf{succ}^N(x1) \cdot \neg^B {}^{\backprime}\mathsf{succ}^N(x1) \cdot \gtrdot\mathsf{succ}^N(x1)$. $\square$

THEOREM 3.2.17

1. $x <^N \mathsf{succ}^N(x)$
2. $x >^N y = x \geq^N \mathsf{succ}^N(y)$

PROOF

1. A simple direct proof by NIND: $\varepsilon <^N \mathsf{succ}^N(\varepsilon) = \varepsilon <^N 1$, $x0 <^N \mathsf{succ}^N(x0) = x0 <^N 0x1 = x \leq^N 0x$, $x1 <^N \mathsf{succ}^N(x1) = x1 <^N \mathsf{succ}^N(x) \cdot 0 = x <^N \mathsf{succ}^N(x)$.

2. By Derived Rule 3.2.3: $\varepsilon >^N y = 0 = \varepsilon \geq^N \mathsf{succ}^N(y)$ (since $\mathsf{succ}^N(y) \neq^N \varepsilon$), $x >^N \varepsilon = x \geq^N \mathsf{succ}^N(\varepsilon)$ (proved by an easy NIND on x),

$$\begin{aligned}
xi >^N y0 &= x >^N y \vee^B (x =^N y \wedge^B i >^B 0)\\
&= x >^N y \vee^B (x =^N y \wedge^B i)\\
&= x >^N 0y \vee^B (x =^N 0y \wedge^B i >^B 1) \vee^B (x =^N 0y \wedge^B i)\\
&= xi >^N 0y1 \vee^B xi =^N 0y1\\
&= xi \geq^N 0y1 = xi \geq^N \mathsf{succ}^N(y0),\\
xi >^N y1 &= x >^N y \vee^B (x =^N y \wedge^B i >^B 1)\\
&= x >^N y\\
&= x \geq^N \mathsf{succ}^N(y)\\
&= x >^N \mathsf{succ}^N(y) \vee^B (x =^N y \wedge^B i \geq^B 0)\\
&= x >^N \mathsf{succ}^N(y) \vee^B (x =^N y \wedge^B i >^B 0) \vee^B (x =^N y \wedge^B i =^B 0)\\
&= xi >^N \mathsf{succ}^N(y) \cdot 0 \vee^B xi =^N \mathsf{succ}^N(y) \cdot 0\\
&= xi \geq^N \mathsf{succ}^N(y) \cdot 0 = xi \geq^N \mathsf{succ}^N(y1). \qquad \square
\end{aligned}$$

CLAIM 3.2.78 $|x| = |_j x|$

PROOF By TIND on x: $|\varepsilon| = \varepsilon = |_j\varepsilon|$, $|i| = 1 = |1| = |_1 i|$, $|x| = x \,?^{EL} \big(|x\blacktriangleleft| \cdot 0, |x\blacktriangleleft| \cdot 1\big) = {}_j x \,?^{EL} \big(|_j x\blacktriangleleft| \cdot 0, |_j x\blacktriangleleft| \cdot 1\big) = |_j x|$. $\square$

THEOREM 3.2.19 $|xi| =^N \mathsf{succ}^N(|x|)$

Proof By TIND on x: $|\varepsilon i| = 1 = \mathsf{succ}^N(|\varepsilon|)$, $|ji| = 10 = \mathsf{succ}^N(1) = \mathsf{succ}^N(|j|)$,

$$\begin{aligned}
|xi| &= xi \mathbin{?^{EL}} \big(|(xi)◀| \cdot 0, |(xi)◀| \cdot 1\big) \\
&= x \mathbin{?^{EL}} \big(|(xi)◀| \cdot 1, |(xi)◀| \cdot 0\big) \\
&= x \mathbin{?^{EL}} \big(|x◀| \cdot 1, |x◀ \cdot \grave{}(▶x \cdot i)| \cdot 0\big) \\
&= x \mathbin{?^{EL}} \big(|x◀| \cdot 1, |x◀ \cdot j| \cdot 0\big) \\
&=^N x \mathbin{?^{EL}} \big(\mathsf{succ}^N(|x◀| \cdot 0), \mathsf{succ}^N(|x◀|) \cdot 0\big) \\
&= x \mathbin{?^{EL}} \big(\mathsf{succ}^N(|x◀| \cdot 0), \mathsf{succ}^N(|x◀| \cdot 1)\big) \\
&= \mathsf{succ}^N(|x|)
\end{aligned}$$

(where the fifth equality, where "$=^N$" is introduced, holds by the induction hypothesis). □

On "masking" functions

Theorem 3.2.20

$$\begin{aligned}
\mathsf{first}_0(0x) &= 1 \cdot {}_0x & \qquad \mathsf{first}_0(1x) &= 0 \cdot \mathsf{first}_0(x) \\
\mathsf{first}_1(0x) &= 0 \cdot \mathsf{first}_1(x) & \qquad \mathsf{first}_1(1x) &= 1 \cdot {}_0x
\end{aligned}$$

Proof By a straightforward NIND. □

Corollary 3.2.81 $\mathsf{first}_0(x) = \mathsf{first}_1(\mathsf{not}^B(x))$

Proof By a straightforward NIND, from the preceding theorem. □

On binary addition

Theorem 3.2.21 $x +^N y = y +^N x$

Proof Direct from the commutativity of the functions involved (*i.e.*, xor^B and and^B). □

Lemma 3.2.82

$$\begin{aligned}
\mathsf{carry}^N(x0, 1) &= \mathsf{carry}^N(x, \varepsilon) \cdot 0 = {}_0x0 \\
\mathsf{carry}^N(x1, 1) &= x \mathbin{?^{ZL}} \big(1, \mathsf{carry}^N(x, 1) \cdot 1\big)
\end{aligned}$$

Proof (We prove only the second statement, the first is a simple application of NIND on x.) By NIND on x: $\mathsf{carry}^N(\varepsilon \cdot 1, 1) = 1 = \varepsilon \mathbin{?^{ZL}} (1, \mathsf{carry}^N(\varepsilon, 1) \cdot 1)$,

$$\begin{aligned}
\mathsf{carry}^N(0x1, 1) &= 0 \cdot \mathsf{carry}^N(x1, 1) \\
&= 0 \cdot x \mathbin{?^{ZL}} \big(1, \mathsf{carry}^N(x, 1) \cdot 1\big) \\
&= x \mathbin{?^{ZL}} \big(0 \cdot 1, 0 \cdot \mathsf{carry}^N(x, 1) \cdot 1\big) \\
&= x \mathbin{?^{ZL}} \big(\mathsf{carry}^N(0, 1) \cdot 1, \mathsf{carry}^N(0x, 1) \cdot 1\big) \\
&= \mathsf{carry}^N(0x, 1) \cdot 1, \\
\mathsf{carry}^N(1x1, 1) &= {}^{\backprime}\mathsf{carry}^N(x1, 1) \cdot \mathsf{carry}^N(x1, 1) \\
&= x \mathbin{?^{ZL}} \big({}^{\backprime}(1) \cdot 1, {}^{\backprime}(\mathsf{carry}^N(x, 1) \cdot 1) \cdot \mathsf{carry}^N(x, 1) \cdot 1\big) \\
&= x \mathbin{?^{ZL}} \big(11, {}^{\backprime}\mathsf{carry}^N(x, 1) \cdot \mathsf{carry}^N(x, 1) \cdot 1\big) \\
&= x \mathbin{?^{ZL}} \big(\mathsf{carry}^N(1, 1) \cdot 1, \mathsf{carry}^N(1x, 1) \cdot 1\big) \\
&= \mathsf{carry}^N(1x, 1) \cdot 1. \qquad \square
\end{aligned}$$

Theorem 3.2.22 $x +^N 1 = x \mathbin{?^{ZL}} \big(0 \cdot \mathsf{succ}^N(x), \mathsf{succ}^N(x)\big) =^N \mathsf{succ}^N(x)$

Proof By NIND on x: $\varepsilon +^N 1 = 01 = \varepsilon \mathbin{?^{ZL}} \big(0 \cdot \mathsf{succ}^N(\varepsilon), \mathsf{succ}^N(\varepsilon)\big)$,

$$\begin{aligned}
x0 +^N 1 &= \mathsf{xor}_3^B(\mathsf{carry}^N(x0, 1) \cdot 0, x0, 1) \\
&= \mathsf{xor}_3^B(0_0x0, 0x0, 0_0x1) \\
&= 0x1 \\
&= \mathsf{succ}^N(x0), \\
x1 +^N 1 &= \mathsf{xor}_3^B(\mathsf{carry}^N(x1, 1) \cdot 0, x1, 1) \\
&= \mathsf{xor}_3^B(\mathsf{carry}^N(x, 1) \cdot 10, 0x1, 0_0x1) \\
&= \mathsf{xor}_3^B(\mathsf{carry}^N(x, 1) \cdot 1, 0x, {}_0x0) \cdot 0 \\
&= \mathsf{xor}_3^B(\mathsf{carry}^N(x, 1) \cdot 0, 0x, {}_0x1) \cdot 0 \\
&= \mathsf{xor}_3^B(\mathsf{carry}^N(x, 1) \cdot 0, x, 1) \cdot 0 \\
&= (x +^N 1) \cdot 0 \\
&=^N \mathsf{succ}^N(x) \cdot 0 \\
&= \mathsf{succ}^N(x1). \qquad \square
\end{aligned}$$

CLAIM 3.2.83 *For* ${}_jx = {}_jy$,

$$\begin{aligned}
\mathsf{carry}^N(x,\varepsilon) &= 0 \cdot {}_0x \\
\mathsf{carry}^N(x,x) &= x0 \\
\mathsf{carry}^N(0x,0y) &= 0 \cdot \mathsf{carry}^N(x,y) \\
\mathsf{carry}^N(1x,0y) &= {\backprime}\mathsf{carry}^N(x,y) \cdot \mathsf{carry}^N(x,y) \\
\mathsf{carry}^N(1x,1y) &= 1 \cdot \mathsf{carry}^N(x,y)
\end{aligned}$$

PROOF All these properties can be proved with a simple application of Derived Rule 3.2.6, or directly from the definition of carry^N. The last three depend on the following facts.

$$\begin{aligned}
\mathsf{maskbit}\big(\mathsf{and}_2^B(0x,0y),\mathsf{first}_0(\mathsf{xor}_2^B(0x,0y))\big) &= \mathsf{maskbit}\big(0\cdot\mathsf{and}_2^B(x,y),\mathsf{first}_0(0\cdot\mathsf{xor}_2^B(x,y))\big) \\
&= \mathsf{maskbit}\big(0\cdot\mathsf{and}_2^B(x,y), 1\cdot{}_0\mathsf{xor}_2^B(x,y)\big) \\
&= 0 \\
\mathsf{maskbit}\big(\mathsf{and}_2^B(1x,0y),\mathsf{first}_0(\mathsf{xor}_2^B(1x,0y))\big) &= \mathsf{maskbit}\big(0\cdot\mathsf{and}_2^B(x,y),\mathsf{first}_0(1\cdot\mathsf{xor}_2^B(x,y))\big) \\
&= \mathsf{maskbit}\big(0\cdot\mathsf{and}_2^B(x,y), 0\cdot\mathsf{first}_0(\mathsf{xor}_2^B(x,y))\big) \\
&= \mathsf{maskbit}\big(\mathsf{and}_2^B(x,y),\mathsf{first}_0(\mathsf{xor}_2^B(x,y))\big) \\
\mathsf{maskbit}\big(\mathsf{and}_2^B(1x,1y),\mathsf{first}_0(\mathsf{xor}_2^B(1x,1y))\big) &= \mathsf{maskbit}\big(1\cdot\mathsf{and}_2^B(x,y),\mathsf{first}_0(0\cdot\mathsf{xor}_2^B(x,y))\big) \\
&= \mathsf{maskbit}\big(1\cdot\mathsf{and}_2^B(x,y), 1\cdot{}_0\mathsf{xor}_2^B(x,y)\big) \\
&= 1 \qquad \square
\end{aligned}$$

CLAIM 3.2.84 *For* ${}_jx = {}_jy$,

$$\begin{aligned}
x +^N \varepsilon &= 0x \\
x +^N x &= x0 \\
0x +^N 0y &= 0 \cdot (x +^N y) \\
1x +^N 0y &= {\backprime}(x +^N y) \cdot \neg^{B}{\backprime}(x +^N y) \cdot {\gtrdot}(x +^N y) \\
1x +^N 1y &= 1 \cdot (x +^N y)
\end{aligned}$$

PROOF Directly from the corresponding properties for carry^N, where we have used the fact that ${\backprime}(x +^N y) = {\backprime}\mathsf{carry}^N(x,y)$. Note that the second property implies that ${\gtrdot}\mathsf{succ}^N(x +^N x) = x1$.
$\square$

Claim 3.2.85

$$\begin{aligned} x0 +^N y0 &= (x +^N y) \cdot 0 \\ x1 +^N y0 &= (x +^N y) \cdot 1 \\ x1 +^N y1 &= \gtrdot\mathsf{succ}^N(x +^N y) \cdot 0 \end{aligned}$$

Proof The first two properties follow directly from the Claim above by Derived Rule 3.2.6. We prove the third property because it is more involved. First, note that $\neg^B\mathsf{AND}(x+^N y)$ can be proved directly from Claim 3.2.84 by Derived Rule 3.2.6. This implies that $\backprime\mathsf{succ}^N(x +^N y) = 0$, which in turn implies that $\mathsf{succ}^N(1 \cdot (x +^N y)) = 01 \cdot \gtrdot\mathsf{succ}^N(x +^N y)$. Now, we prove the third property by Derived Rule 3.2.6: $1 +^N 1 = 10 = \gtrdot(01) \cdot 0 = \gtrdot\mathsf{succ}^N(0) \cdot 0 = \gtrdot\mathsf{succ}^N(\varepsilon +^N \varepsilon) \cdot 0$,

$$\begin{aligned}
&0x1 +^N 0y1 \\
&\quad = 0 \cdot (x1 +^N y1) \\
&\quad = 0 \cdot \gtrdot\mathsf{succ}^N(x +^N y) \cdot 0 \\
&\quad = \mathsf{succ}^N(x +^N y) \cdot 0 \\
&\quad = \gtrdot(0 \cdot \mathsf{succ}^N(x +^N y)) \cdot 0 \\
&\quad = \gtrdot\mathsf{succ}^N(0 \cdot (x +^N y)) \cdot 0 \\
&\quad = \gtrdot\mathsf{succ}^N(0x +^N 0y) \cdot 0, \\
&1x1 +^N 1y1 \\
&\quad = 1 \cdot (x1 +^N y1) \\
&\quad = 1 \cdot \gtrdot\mathsf{succ}^N(x +^N y) \cdot 0 \\
&\quad = \gtrdot(01 \cdot \gtrdot\mathsf{succ}^N(x +^N y)) \cdot 0 \\
&\quad = \gtrdot\mathsf{succ}^N(1 \cdot (x +^N y)) \cdot 0 \\
&\quad = \gtrdot\mathsf{succ}^N(1x +^N 1y) \cdot 0, \\
&1x1 +^N 0y1 \\
&\quad = \backprime(x1 +^N y1) \cdot \neg^B\backprime(x1 +^N y1) \cdot \gtrdot(x1 +^N y1) \\
&\quad = \backprime(\gtrdot\mathsf{succ}^N(x +^N y) \cdot 0) \cdot \neg^B\backprime(\gtrdot\mathsf{succ}^N(x +^N y) \cdot 0) \cdot \gtrdot(\gtrdot\mathsf{succ}^N(x +^N y) \cdot 0) \\
&\quad = \backprime\gtrdot\mathsf{succ}^N(x +^N y) \cdot \neg^B\backprime\gtrdot\mathsf{succ}^N(x +^N y) \cdot \gtrdot\gtrdot\mathsf{succ}^N(x +^N y) \cdot 0 \\
&\quad = \backprime(x +^N y) \;?^{ZL} \Big(\backprime\gtrdot\mathsf{succ}^N(1\gtrdot(x +^N y)) \cdot \neg^B\backprime\gtrdot\mathsf{succ}^N(1\gtrdot(x +^N y)) \cdot \gtrdot\gtrdot\mathsf{succ}^N(1\gtrdot(x +^N y)) \cdot 0, \\
&\qquad\qquad \backprime\gtrdot\mathsf{succ}^N(0\gtrdot(x +^N y)) \cdot \neg^B\backprime\gtrdot\mathsf{succ}^N(0\gtrdot(x +^N y)) \cdot \gtrdot\gtrdot\mathsf{succ}^N(0\gtrdot(x +^N y)) \cdot 0\Big) \\
&\quad = \backprime(x +^N y) \;?^{ZL} \Big(\neg^B\backprime\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot \backprime\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot \gtrdot\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot 0, \\
&\qquad\qquad \backprime\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot \neg^B\backprime\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot \gtrdot\mathsf{succ}^N(\gtrdot(x +^N y)) \cdot 0\Big)
\end{aligned}$$

$$
\begin{aligned}
&= \backprime(x +^N y) \mathbin{?^{ZL}} \Big(\neg^B \mathsf{AND}(1 \gtrdot (x +^N y)) \cdot \mathsf{succ}^N(\gtrdot(x +^N y)) \cdot 0, \mathsf{succ}^N(1 \gtrdot (x +^N y)) \cdot 0\Big) \\
&= \backprime(x +^N y) \mathbin{?^{ZL}} \Big(1 \cdot \mathsf{succ}^N(\gtrdot(x +^N y)) \cdot 0, \mathsf{succ}^N(1 \gtrdot (x +^N y)) \cdot 0\Big) \\
&= \backprime(x +^N y) \mathbin{?^{ZL}} \Big(\gtrdot(01 \cdot \mathsf{succ}^N(\gtrdot(x +^N y))) \cdot 0, \gtrdot(0 \cdot \mathsf{succ}^N(1 \gtrdot (x +^N y))) \cdot 0\Big) \\
&= \backprime(x +^N y) \mathbin{?^{ZL}} \Big(\gtrdot \mathsf{succ}^N(10 \gtrdot (x +^N y)) \cdot 0, \gtrdot \mathsf{succ}^N(01 \gtrdot (x +^N y)) \cdot 0\Big) \\
&= \gtrdot \mathsf{succ}^N\big(\backprime(x +^N y) \cdot \neg^B \backprime(x +^N y) \cdot \gtrdot(x +^N y)\big) \cdot 0 \\
&= \gtrdot \mathsf{succ}^N(1x +^N 0y) \cdot 0. \qquad \square
\end{aligned}
$$

THEOREM 3.2.23

1. $x +^N \mathsf{succ}^N(y) =^N \mathsf{succ}^N(x +^N y)$
2. $x +^N (y +^N z) =^N (x +^N y) +^N z$
3. $y <^N z \leftrightarrow x +^N y <^N x +^N z$
4. $x =^N y \wedge z =^N w \rightarrow x +^N z =^N y +^N w$
5. $x =^N y \wedge z <^N w \rightarrow x +^N z <^N y +^N w$
6. $x <^N y \wedge z <^N w \rightarrow x +^N z <^N y +^N w$

PROOF

1. By Derived Rule 3.2.3: $\varepsilon +^N \mathsf{succ}^N(y) =^N \mathsf{succ}^N(y) =^N \mathsf{succ}^N(\varepsilon +^N y)$, $x +^N \mathsf{succ}^N(\varepsilon) = x +^N 1 =^N \mathsf{succ}^N(x) =^N \mathsf{succ}^N(x +^N \varepsilon)$,

$$
\begin{aligned}
x0 +^N \mathsf{succ}^N(y0) &=^N x0 +^N y1 \\
&=^N (x +^N y) \cdot 1 \\
&=^N \mathsf{succ}^N((x +^N y) \cdot 0) \\
&=^N \mathsf{succ}^N(x0 +^N y0), \\
x0 +^N \mathsf{succ}^N(y1) &=^N x0 +^N \mathsf{succ}^N(y) \cdot 0 \\
&=^N (x +^N \mathsf{succ}^N(y)) \cdot 0 \\
&=^N \mathsf{succ}^N(x +^N y) \cdot 0 \\
&=^N \mathsf{succ}^N((x +^N y) \cdot 1) \\
&=^N \mathsf{succ}^N(x0 +^N y1), \\
x1 +^N \mathsf{succ}^N(y0) &=^N x1 +^N y1 \\
&=^N \mathsf{succ}^N(x +^N y) \cdot 0 \\
&=^N \mathsf{succ}^N((x +^N y) \cdot 1) \\
&=^N \mathsf{succ}^N(x1 +^N y0),
\end{aligned}
$$

$$\begin{aligned}
x1 +^N \mathsf{succ}^N(y1) &=^N x1 +^N \mathsf{succ}^N(y) \cdot 0 \\
&=^N (x +^N \mathsf{succ}^N(y)) \cdot 1 \\
&=^N \mathsf{succ}^N(x +^N y) \cdot 1 \\
&=^N \mathsf{succ}^N(\mathsf{succ}^N(x +^N y) \cdot 0) \\
&=^N \mathsf{succ}^N(x1 +^N y1).
\end{aligned}$$

2. By Derived Rule 3.2.3, generalized to three variables: $\varepsilon +^N (y +^N z) =^N y +^N z =^N (\varepsilon +^N y) +^N z$ (and similarly for x, ε, z and x, y, ε), $x0 +^N (y0 +^N z0) =^N x0 +^N (y +^N z)0 =^N (x +^N (y +^N z))0 =^N ((x +^N y) +^N z)0 =^N (x +^N y)0 +^N z0 =^N (x0 +^N y0) +^N z0$, $x1 +^N (y0 +^N z0) =^N x1 +^N (y +^N z)0 =^N (x +^N (y +^N z))1 =^N ((x +^N y) +^N z)1 =^N (x +^N y)1 +^N z0 =^N (x1 +^N y0) +^N z0$ (and similarly for $x0, y1, z0$ and $x0, y0, z1$), $x0 +^N (y1 +^N z1) =^N x0 +^N \mathsf{succ}^N(y +^N z)0 =^N (x +^N \mathsf{succ}^N(y +^N z))0 =^N \mathsf{succ}^N(x +^N (y +^N z))0 =^N \mathsf{succ}^N((x +^N y) +^N z)0 =^N (x +^N y)1 +^N z1 =^N (x0 +^N y1) +^N z1$ (and similarly for $x1, y0, z1$ and $x1, y1, z0$), $x1 +^N (y1 +^N z1) =^N x1 +^N \mathsf{succ}^N(y +^N z)0 =^N (x +^N \mathsf{succ}^N(y +^N z))1 =^N \mathsf{succ}^N(x +^N (y +^N z))1 =^N \mathsf{succ}^N((x +^N y) +^N z)1 =^N (\mathsf{succ}^N(x +^N y) +^N z)1 =^N \mathsf{succ}^N(x +^N y)0 +^N z1 =^N (x1 +^N y1) +^N z1$.

3–6. Similar to the last case, straightforward, case-by-case proofs by Derived Rule 3.2.3. □

On iterated sums

Claim 3.2.86

$$\begin{gathered}
\mathsf{CScar}_3(x0, y0, z0) = \mathsf{CScar}_3(x, y, z) \cdot 0 \\
\mathsf{CScar}_3(x1, y0, z0) = \mathsf{CScar}_3(x0, y1, z0) = \mathsf{CScar}_3(x1, y0, z1) = \mathsf{CScar}_3(x, y, z) \cdot 0 \\
\mathsf{CScar}_3(x0, y1, z1) = \mathsf{CScar}_3(x1, y0, z1) = \mathsf{CScar}_3(x1, y1, z0) = \mathsf{CScar}_3(x, y, z) \cdot 1 \\
\mathsf{CScar}_3(x1, y1, z1) = \mathsf{CScar}_3(x, y, z) \cdot 1 \\
{}_0\mathsf{CScar}_3(x, y, z) \cdot 0 = {}_0\mathsf{CSadd}_3(x, y, z) = 0 \cdot {}_0\mathsf{max}^L_3(x, y, z) = 0 \cdot \mathsf{max}^L_3({}_0x, {}_0y, {}_0z) \\
0 \cdot \mathsf{max}^L_3({}_0x, {}_0y, {}_0z) = \mathsf{CScar}_3({}_0x, {}_0y, {}_0z) \cdot 0 = \mathsf{CSadd}_3({}_0x, {}_0y, {}_0z) \\
{}_0\mathsf{CScar}(x, y, z, w) = {}_0\mathsf{CSadd}(x, y, z, w) = 00 \cdot {}_0\mathsf{max}^L_4(x, y, z, w) = 00 \cdot \mathsf{max}^L_4({}_0x, {}_0y, {}_0z, {}_0w) \\
00 \cdot \mathsf{max}^L_4({}_0x, {}_0y, {}_0z, {}_0w) = \mathsf{CScar}({}_0x, {}_0y, {}_0z, {}_0w) = \mathsf{CSadd}({}_0x, {}_0y, {}_0z, {}_0w)
\end{gathered}$$

Proof All can be proved with a very simple application of Derived Rule 3.2.6, generalized to three variables, directly from the definitions of the functions involved. □

Lemma 3.2.87

$$\begin{aligned}
&(\mathsf{CScar}_3(\mathsf{succ}^N(x), y, z) \cdot 0) +^N \mathsf{CSadd}_3(\mathsf{succ}^N(x), y, z) \\
&\qquad =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(x, y, z) \cdot 0) +^N \mathsf{CSadd}_3(x, y, z)\big)
\end{aligned}$$

Proof The proof is a straightforward, if tedious, application of Derived Rule 3.2.3 (generalized to three variables). First, the base cases for ε, y, z and x, ε, z and x, y, ε are proved (each one with another application of Derived Rule 3.2.3), and then the eight cases from $x0, y0, z0$ to $x1, y1, z1$ are proved from the assumption that the lemma holds for x, y, z. We do not show the full proof here as it is not particularly instructive; instead, we give parts of the proof for two illustrative cases. First, in the proof of the base case $(\mathsf{CScar}_3(\mathsf{succ}^N(\varepsilon), y, z) \cdot 0) +^N \mathsf{CSadd}_3(\mathsf{succ}^N(\varepsilon), y, z) =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(\varepsilon, y, z) \cdot 0) +^N \mathsf{CSadd}_3(\varepsilon, y, z)\big)$ by Derived Rule 3.2.3, we show the case for $y1, z1$.

$$\begin{aligned}
&(\mathsf{CScar}_3(\mathsf{succ}^N(\varepsilon), y1, z1) \cdot 0) +^N \mathsf{CSadd}_3(\mathsf{succ}^N(\varepsilon), y1, z1)\\
&\ =^N (\mathsf{CScar}_3(1, y1, z1) \cdot 0) +^N \mathsf{CSadd}_3(1, y1, z1)\\
&\ =^N (\mathsf{CScar}_3(\varepsilon, y, z) \cdot 10) +^N (\mathsf{CSadd}_3(\varepsilon, y, z) \cdot 1)\\
&\ =^N \big((\mathsf{CScar}_3(\varepsilon, y, z) \cdot 1) +^N \mathsf{CSadd}_3(\varepsilon, y, z)\big) \cdot 1\\
&\ =^N \mathsf{succ}^N\big(\big((\mathsf{CScar}_3(\varepsilon, y, z) \cdot 1) +^N \mathsf{CSadd}_3(\varepsilon, y, z)\big) \cdot 0\big)\\
&\ =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(\varepsilon, y1, z1) \cdot 0) +^N \mathsf{CSadd}_3(\varepsilon, y1, z1)\big)
\end{aligned}$$

Second, in the inductive step, we show the case for $x0, y1, z0$.

$$\begin{aligned}
&(\mathsf{CScar}_3(\mathsf{succ}^N(x0), y1, z0) \cdot 0) +^N \mathsf{CSadd}_3(\mathsf{succ}^N(x0), y1, z0)\\
&\ =^N (\mathsf{CScar}_3(x1, y1, z0) \cdot 0) +^N \mathsf{CSadd}_3(x1, y1, z0)\\
&\ =^N (\mathsf{CScar}_3(x, y, z) \cdot 10) +^N (\mathsf{CSadd}_3(x, y, z) \cdot 0)\\
&\ =^N \big(\mathsf{succ}^N(\mathsf{CScar}_3(x, y, z) \cdot 0) +^N \mathsf{CSadd}_3(x, y, z)\big) \cdot 0\\
&\ =^N \mathsf{succ}^N\big(\big((\mathsf{CScar}_3(x, y, z) \cdot 0) +^N \mathsf{CSadd}_3(x, y, z)\big) \cdot 1\big)\\
&\ =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(x, y, z) \cdot 00) +^N (\mathsf{CSadd}_3(x, y, z) \cdot 1)\big)\\
&\ =^N \mathsf{succ}^N\big((\mathsf{CScar}_3(x0, y1, z0) \cdot 0) +^N \mathsf{CSadd}_3(x0, y1, z0)\big) \qquad \square
\end{aligned}$$

Theorem 3.2.24 $\mathsf{CScar}(x, y, z, w) +^N \mathsf{CSadd}(x, y, z, w) =^N x +^N y +^N z +^N w$

Proof As for the preceding lemma, the proof is a straightforward, if tedious, application of Derived Rule 3.2.3 (generalized to four variables). First, the base cases for ε, y, z, w and x, ε, z, w and x, y, ε, w and x, y, z, ε are proved (each one with another application of Derived Rule 3.2.3), and then the sixteen cases from $x0, y0, z0, w0$ to $x1, y1, z1, w1$ are proved from the assumption that the lemma holds for x, y, z, w. We do not show the full proof here as it is not particularly instructive; instead, we give parts of the proof for one illustrative case. In the inductive step, we show the case for $x1, y1, z1, w1$.

$$\begin{aligned}
&\mathsf{CScar}(x1,y1,z1,w1) +^N \mathsf{CSadd}(x1,y1,z1,w1)\\
&\quad =^N \mathsf{CScar}_3\big(\mathsf{CScar}_3(x,y,z)\cdot 10, \mathsf{CSadd}_3(x,y,z)\cdot 1, w1\big)\cdot 0 +^N\\
&\qquad \mathsf{CSadd}_3\big(\mathsf{CScar}_3(x,y,z)\cdot 10, \mathsf{CSadd}_3(x,y,z)\cdot 1, w1\big)\\
&\quad =^N \mathsf{CScar}_3\big(\mathsf{succ}^N(\mathsf{CScar}_3(x,y,z)\cdot 0), \mathsf{CSadd}_3(x,y,z), w\big)\cdot 10 +^N\\
&\qquad \mathsf{CSadd}_3\big(\mathsf{succ}^N(\mathsf{CScar}_3(x,y,z)\cdot 0), \mathsf{CSadd}_3(x,y,z), w\big)\cdot 0\\
&\quad =^N \mathsf{succ}^N\Big(\mathsf{CScar}_3\big(\mathsf{succ}^N(\mathsf{CScar}_3(x,y,z)\cdot 0), \mathsf{CSadd}_3(x,y,z), w\big)\cdot 0 +^N\\
&\qquad\qquad \mathsf{CSadd}_3\big(\mathsf{succ}^N(\mathsf{CScar}_3(x,y,z)\cdot 0), \mathsf{CSadd}_3(x,y,z), w\big)\Big)\cdot 0\\
&\quad =^N \mathsf{succ}^N\big(\mathsf{succ}^N\big(\mathsf{CScar}(x,y,z,w) +^N \mathsf{CSadd}(x,y,z,w)\big)\big)\cdot 0\\
&\quad =^N \mathsf{succ}^N\big(\mathsf{succ}^N\big(x +^N y +^N z +^N w\big)\big)\cdot 0\\
&\quad =^N \mathsf{succ}^N(x +^N y)\cdot 0 +^N \mathsf{succ}^N(z +^N w)\cdot 0\\
&\quad =^N x1 +^N y1 +^N z1 +^N w1 \qquad \Box
\end{aligned}$$

Claim 3.2.88

1. $\mathsf{CARADD}({}_0x) =^N 0$

2. $\mathsf{sum}({}_0x) = \mathsf{CAR}({}_0x) +^N \mathsf{ADD}({}_0x) =^N 0 +^N 0 =^N 0$

Proof

1. By TIND on x: $\mathsf{CARADD}({}_0\varepsilon) = \varepsilon =^N 0$, $\mathsf{CARADD}({}_0 i) = 00 =^N 0$,

$$\begin{aligned}
\mathsf{CARADD}({}_0x) &= \mathsf{CScar}(\mathsf{CAR}({}_0x\blacktriangleleft), \mathsf{ADD}({}_0x\blacktriangleleft), \mathsf{CAR}(\blacktriangleright{}_0x), \mathsf{ADD}(\blacktriangleright{}_0x))\cdot\\
&\quad \mathsf{CSadd}(\mathsf{CAR}({}_0x\blacktriangleleft), \mathsf{ADD}({}_0x\blacktriangleleft), \mathsf{CAR}(\blacktriangleright{}_0x), \mathsf{ADD}(\blacktriangleright{}_0x))\\
&=^N \mathsf{CScar}(0,0,0,0)\cdot \mathsf{CSadd}(0,0,0,0)\\
&=^N 0\cdot 0 =^N 0.
\end{aligned}$$

2. Direct corollary of the first claim. $\Box$

Theorem 3.2.25 $\mathsf{sum}(x) =^N \mathsf{sum}(x\blacktriangleleft) +^N \mathsf{sum}(\blacktriangleright x)$

Proof

$$\begin{aligned}\mathsf{sum}(x) &= \mathsf{CScar}(\mathsf{CAR}(x◀), \mathsf{ADD}(x◀), \mathsf{CAR}(▶x), \mathsf{ADD}(▶x)) +^{N} \\ &\quad \mathsf{CSadd}(\mathsf{CAR}(x◀), \mathsf{ADD}(x◀), \mathsf{CAR}(▶x), \mathsf{ADD}(▶x)) \\ &=^{N} \mathsf{CAR}(x◀) +^{N} \mathsf{ADD}(x◀) +^{N} \mathsf{CAR}(▶x) +^{N} \mathsf{ADD}(▶x) \\ &= \mathsf{sum}(x◀) +^{N} \mathsf{sum}(▶x) \qquad \square\end{aligned}$$

From this theorem, it is possible to prove that $\mathsf{sum}(xy) =^{N} \mathsf{sum}(x) +^{N} \mathsf{sum}(y)$ with a sequence of lemmas and theorems similar to the ones used to show that $\mathsf{AND}(xy) = \mathsf{AND}(x) \wedge^{B} \mathsf{AND}(y)$. In particular, we have that $\mathsf{sum}(x0) =^{N} \mathsf{sum}(x) +^{N} 0 =^{N} \mathsf{sum}(x)$ and $\mathsf{sum}(x1) =^{N} \mathsf{sum}(x) +^{N} 1 =^{N} \mathsf{succ}^{N}(\mathsf{sum}(x))$.

Theorem 3.2.26 $\quad \mathsf{sum}(x) \leq^{N} \mathsf{sum}({}_1x) =^{N} |x|$

Proof By TIND on x: $\mathsf{sum}(\varepsilon) = \mathsf{sum}({}_1\varepsilon) = 0 =^{N} |\varepsilon|$, $\mathsf{sum}(i) =^{N} i \leq^{N} 1 =^{N} \mathsf{sum}(1) =^{N} |1|$, $\mathsf{sum}(x) =^{N} \mathsf{sum}(x◀) +^{N} \mathsf{sum}(▶x) \leq^{N} \mathsf{sum}({}_1x◀) +^{N} \mathsf{sum}(▶{}_1x) =^{N} \mathsf{sum}({}_1x) =^{N} \mathsf{sum}({}_1x◀) +^{N} \mathsf{sum}(▶{}_1x) =^{N} |{}_1x◀| +^{N} |▶{}_1x| =^{N} |x|$. To complete the inductive case for $\mathsf{sum}({}_1x) = |x|$, we need to prove that $|x| =^{N} |x◀| +^{N} |▶x|$ by TIND on x: the base cases are trivial, and

$$\begin{aligned}|x◀| +^{N} |▶x| &=^{N} x \,?^{EL} \big(|{}_1x◀| +^{N} |{}_1x◀|, |{}_1x◀| +^{N} |{}_1x◀ \cdot 1|\big) \\ &=^{N} x \,?^{EL} \big(|{}_1x◀| \cdot 0, |x◀| +^{N} \mathsf{succ}^{N}(|x◀|)\big) \\ &=^{N} x \,?^{EL} \big(|x◀| \cdot 0, \mathsf{succ}^{N}(|x◀| +^{N} |x◀|)\big) \\ &=^{N} x \,?^{EL} \big(|x◀| \cdot 0, |x◀| \cdot 1\big) \\ &=^{N} |x|. \qquad \square\end{aligned}$$

Lemmas for the proof of PHP

Lemma A.1 $\quad \mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(x), y)\big) \rightarrow^{B} \mathsf{sum}(x) \leq^{N} \mathsf{sum}(y)$

Proof By Derived Rule 3.2.3:

$$\begin{aligned}&\mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(\varepsilon), y)\big) \rightarrow^{B} \mathsf{sum}(\varepsilon) \leq^{N} \mathsf{sum}(y) = \mathsf{AND}(y) \rightarrow^{B} \varepsilon \leq^{N} \mathsf{sum}(y) = 1, \\ &\mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(x), \varepsilon)\big) \rightarrow^{B} \mathsf{sum}(x) \leq^{N} \mathsf{sum}(\varepsilon) \\ &\quad = \mathsf{AND}(\mathsf{not}^{B}(x)) \rightarrow^{B} \mathsf{sum}(x) =^{N} \varepsilon = \neg^{B}\mathsf{OR}(x) \rightarrow^{B} x =^{S} {}_0x = 1, \\ &\mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(xi), yj)\big) \rightarrow^{B} \mathsf{sum}(xi) \leq^{N} \mathsf{sum}(yj) \\ &\quad = \mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(x), y)\big) \wedge^{B} (\neg^{B} i \vee^{B} j) \rightarrow^{B} \mathsf{sum}(x) +^{N} i \leq^{N} \mathsf{sum}(y) +^{N} j \\ &\quad = \mathsf{AND}\big(\mathsf{or}_2^{B}(\mathsf{not}^{B}(x), y)\big) \wedge^{B} (\neg^{B} i \vee^{B} j) \rightarrow^{B} \mathsf{sum}(x) \leq^{N} \mathsf{sum}(y) \wedge^{B} \neg^{B}(i \wedge^{B} \neg^{B} j) = 1. \qquad \square\end{aligned}$$

LEMMA A.2 $\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(x), y)\big) \wedge^B \mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(x), y)\big) \rightarrow \mathsf{sum}(x) <^N \mathsf{sum}(y)$

PROOF By Derived Rule 3.2.3:

$$\begin{aligned}
&\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(\varepsilon), y)\big) \wedge^B \mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(\varepsilon), y)\big) \rightarrow^B \mathsf{sum}(\varepsilon) <^N \mathsf{sum}(y)\\
&\quad = \mathsf{AND}(y) \wedge^B \mathsf{OR}({}_0y) \rightarrow^B \varepsilon <^N \mathsf{sum}(y)\\
&\quad = 0 \rightarrow^B \varepsilon <^N \mathsf{sum}(y) = 1,\\
&\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(x), \varepsilon)\big) \wedge^B \mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(x), \varepsilon)\big) \rightarrow^B \mathsf{sum}(x) <^N \mathsf{sum}(\varepsilon)\\
&\quad = \mathsf{AND}(\mathsf{not}^B(x)) \wedge^B \mathsf{OR}({}_0x) \rightarrow^B \mathsf{sum}(x) <^N \varepsilon\\
&\quad = 0 \rightarrow^B 0 = 1,\\
&\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(xi), yj)\big) \wedge^B \mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(xi), yj)\big) \rightarrow^B \mathsf{sum}(xi) <^N \mathsf{sum}(yj)\\
&\quad = \big(\mathsf{AND}\big(\mathsf{or}_2^B(\mathsf{not}^B(x), y)\big) \wedge^B (\neg^B i \wedge^B j)\big) \wedge^B \big(\mathsf{OR}\big(\mathsf{and}_2^B(\mathsf{not}^B(x), y)\big) \vee^B (\neg^B i \wedge^B j)\big) \rightarrow^B\\
&\qquad \mathsf{sum}(x) +^N i <^N \mathsf{sum}(y) +^N j
\end{aligned}$$

(and a simple check of all four cases for the possible values of i and j shows that the property holds in each one). □

LEMMA A.3 $\mathsf{maskbit}(x, \mathsf{first}_1(x)) = \mathsf{OR}(x)$

PROOF By NIND on x: $\mathsf{maskbit}(\varepsilon, \mathsf{first}_1(\varepsilon)) = \mathsf{markbit}(\varepsilon, \varepsilon) = \mathsf{OR}(\mathsf{and}_2^B(\varepsilon, \varepsilon)) = \mathsf{OR}(\varepsilon)$, and $\mathsf{maskbit}(xi, \mathsf{first}_1(xi)) = \mathsf{maskbit}(x, \mathsf{first}_1(x)) \vee^B (i \wedge^B \mathsf{OR}(x) \,?^B (0, i)) = \mathsf{OR}(x) \vee^B \mathsf{OR}(x) \,?^B (0, i) = \mathsf{OR}(x) \,?^B (1, i) = \mathsf{OR}(x) \vee^B i = \mathsf{OR}(xi)$. □

LEMMA A.4
(L) $\approx^B\mathsf{lb}(x, y) = \mathsf{maskbit}\big(x, (1 \cdot {}_0(y \rhd x1))\lessdot\big)$
(R) $\approx^B\mathsf{rb}(x, y) = \mathsf{maskbit}\big(x, (1 \cdot {}_0y)\lessdot\big)$

PROOF (L) By Derived Rule 3.2.3:

$$\begin{aligned}
&\mathsf{maskbit}\big(\varepsilon, (1 \cdot {}_0(y \rhd 1))\lessdot\big)\\
&\quad = y \,?^{ZL} \big(\mathsf{maskbit}(\varepsilon, 1), \mathsf{maskbit}(\varepsilon, \varepsilon)\big)\\
&\quad = y \,?^{ZL} (0, 0) = 0 = \approx^B\varepsilon = \approx^B\mathsf{lb}(\varepsilon, y),\\
&\mathsf{maskbit}\big(x, (1 \cdot {}_0(\varepsilon \rhd x1))\lessdot\big)\\
&\quad = \mathsf{maskbit}(x, 1 \cdot {}_0x)\\
&\quad = \mathsf{OR}(\mathsf{and}_2^B(0x, 1 \cdot {}_0x))\\
&\quad = (0 \wedge^B 1) \vee^B \mathsf{OR}(\mathsf{and}_2^B(x, {}_0x))\\
&\quad = 0 \vee^B \mathsf{OR}({}_0x)\\
&\quad = 0 = \approx^B\varepsilon = \approx^B\mathsf{lb}(x, \varepsilon),
\end{aligned}$$

$$
\begin{aligned}
&\mathsf{maskbit}\big(ix, (1 \cdot {}_0(yj \rhd ix1))\lessdot\big)\\
&\quad = \mathsf{maskbit}\big(ix, (1 \cdot {}_0(y \rhd x1))\lessdot\big)\\
&\quad = y \mathbin{?^{ZL}} \Big(\mathsf{maskbit}(ix, 1 \cdot {}_0 x),\\
&\qquad\qquad \mathsf{maskbit}\big(ix, {}_0\big(ix \lhd (1 \cdot {}_0(y \rhd x1))\lessdot\big) \cdot (1 \cdot {}_0(y \rhd x1))\lessdot\big)\Big)\\
&\quad = y \mathbin{?^{ZL}} \Big((i \wedge^B 1) \vee^B \mathsf{maskbit}(x, {}_0 x),\\
&\qquad\qquad \mathsf{maskbit}\big(ix, {}_0(ix \lhd (y \rhd x1)) \cdot (1 \cdot {}_0(y \rhd x1))\lessdot\big)\Big)\\
&\quad = y \mathbin{?^{ZL}} \Big(i \vee^B 0, \mathsf{maskbit}\big(ix, 0 \cdot {}_0(x \lhd (y \rhd x1)) \cdot (1 \cdot {}_0(y \rhd x1))\lessdot\big)\Big)\\
&\quad = y \mathbin{?^{ZL}} \Big(i, (i \wedge^B 0) \vee^B \mathsf{maskbit}\big(x, {}_0(x \lhd (y \rhd x1)) \cdot (1 \cdot {}_0(y \rhd x1))\lessdot\big)\Big)\\
&\quad = y \mathbin{?^{ZL}} \Big(i, \mathsf{maskbit}\big(x, (1 \cdot {}_0(y \rhd x1))\lessdot\big)\Big)\\
&\quad = y \mathbin{?^{ZL}} (i, \approx^B \mathsf{lb}(x, y))\\
&\quad = \approx^B \mathsf{lb}(ix, yj). \qquad \square
\end{aligned}
$$

Lemma A.5 $\neg^B \mathsf{OR}(\mathsf{delfirst}_1(x)) \wedge^B \mathsf{lb}(x, y) \to^B \neg^B \mathsf{OR}(\mathsf{lc}(x, y\lessdot))$

Proof By NIND on x: $\neg^B \mathsf{OR}(\mathsf{delfirst}_1(\varepsilon)) \wedge^B \mathsf{lb}(\varepsilon, y) \to^B \neg^B \mathsf{OR}(\mathsf{lc}(\varepsilon, y\lessdot)) = \neg^B \mathsf{OR}(\varepsilon) \wedge^B \varepsilon \to^B \neg^B \mathsf{OR}(\varepsilon) = 0 \to^B 0 = 1$, If $y = \varepsilon$ or $y >^L xi$, then $\mathsf{lb}(xi, y) = \varepsilon$, which makes the antecedent of $\to^B$ false and the entire statement trivially true. Hence, we prove the inductive step under the implicit assumption that $y \neq \varepsilon$ and $y \leq^L xi$.

$$
\begin{aligned}
&\neg^B \mathsf{OR}(\mathsf{delfirst}_1(xi)) \wedge^B \mathsf{lb}(xi, y) \to^B \neg^B \mathsf{OR}(\mathsf{lc}(xi, y\lessdot))\\
&\quad = y =^L xi \mathbin{?^{ZL}} \Big(\neg^B(i \wedge^B \neg^B(\mathsf{OR}(x) \mathbin{?^B} (0, i))) \wedge^B \neg^B \mathsf{OR}(\mathsf{delfirst}_1(x)) \wedge^B i \to^B \neg^B \mathsf{OR}(\mathsf{lc}(xi, (xi)\lessdot)),\\
&\qquad\qquad \neg^B(i \wedge^B \neg^B(\mathsf{OR}(x) \mathbin{?^B} (0, i))) \wedge^B \neg^B \mathsf{OR}(\mathsf{delfirst}_1(x)) \wedge^B \mathsf{lb}(x, y) \to^B \neg^B \mathsf{OR}(\mathsf{lc}(xi, y\lessdot))\Big)\\
&\quad = y =^L xi \mathbin{?^{ZL}} \Big(\neg^B(i \wedge^B \mathsf{OR}(x) \mathbin{?^B} (1, \neg^B i)) \wedge^B i \wedge^B \neg^B \mathsf{OR}(\mathsf{delfirst}_1(x)) \to^B \neg^B \mathsf{OR}(\mathsf{lc}(xi, x)), 1\Big)\\
&\quad = y =^L xi \mathbin{?^{ZL}} \Big(\neg^B(\mathsf{OR}(x) \mathbin{?^B} (i, 0)) \wedge^B \mathsf{OR}(x) \mathbin{?^B} (i, i) \wedge^B \neg^B \mathsf{OR}(\mathsf{delfirst}_1(x)) \to^B \neg^B \mathsf{OR}(x), 1\Big)\\
&\quad = y =^L xi \mathbin{?^{ZL}} \big(\mathsf{OR}(x) \mathbin{?^B} (0, i) \to^B \mathsf{OR}(x) \mathbin{?^B} (0, 1), 1\big)\\
&\quad = y =^L xi \mathbin{?^{ZL}} (1, 1) = 1 \qquad \square
\end{aligned}
$$

Bibliography

[1] Bill Allen. Arithmetizing uniform NC. *Annals of Pure and Applied Logic*, 53:1–50, 1991.

[2] Toshiyasu Arai. A bounded arithmetic *AID* for Frege system. Technical Report FI–CXT1998–003, The Fields Institute, April 1998.

[3] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Science*, 38:150–164, 1989.

[4] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[5] Stephen Bloch. Functional characterizations of uniform log-depth and polylog-depth circuit families. In *Proceedings of IEEE 7th Annual Structure in Complexity Theory Conference*, pages 193–206, Boston, Massachusetts, June 1992.

[6] Stephen Bloch. Function-algebraic characterizations of log and polylog parallel time. *Computational Complexity*, 4:175–205, 1994.

[7] Samuel R. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory.* Bibliopolis, Naples, 1986.

[8] Samuel R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52(4):916–927, December 1987.

[9] Samuel R. Buss. Propositional consistency proofs. *Annals of Pure and Applied Logic*, 52:3–29, 1991.

[10] Samuel R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

[11] Samuel R. Buss. Relating the bounded arithmetic and polynomial time hierarchies. Manuscript, November 1994.

[12] Peter Clote. Sequential, machine-independent characterizations of the parallel complexity classes ALOGTIME, AC^k, NC^k, and NC. In S. R. Buss and P. Scott, editors, *Feasible Mathematics*, pages 49–69. Birkhäuser, 1989.

[13] Peter Clote. ALOGTIME and a conjecture of S. A. Cook. *Annals of Mathematics and Artificial Intelligence*, 6:57–106, 1992. Extended abstract in: Proceedings of IEEE Symposium on Logic in Computer Science, Philadelphia, June 1990.

[14] Peter Clote. On polynomial size Frege proofs of certain combinatorial principles. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 162–184. Oxford University Press, 1993.

[15] Peter Clote and Gaisi Takeuti. First order bounded arithmetic and small Boolean circuit complexity classes. In Peter Clote and Jeffrey Remmel, editors, *Feasible Mathematics II*, volume 13 of *Progress in Computer Science and Applied Logic*, pages 154–218, Boston, 1995. Birkhäuser.

[16] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the 1964 International Congress for Logic Methodology and the Philosophy of Science*, pages 24–30, Amsterdam, 1964. North Holland.

[17] Stephen Cook. Relating the provable collapse of P to NC^1 and the power of logical theories. In Paul Beame and Samuel Buss, editors, *Proof Complexity and Feasible Arithmetics: DIMACS Workshop, April 21–24, 1996*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 73–91. American Mathematical Society, 1998.

[18] Stephen A. Cook. Feasible constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing*, pages 83–97, May 1975.

[19] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979.

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

[21] Russell Impagliazzo, Toniann Pitassi, and Alasdair Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Proceedings of IEEE 9th Annual Symposium on Logic in Computer Science*, pages 220–228, 1994.

[22] Jan Johannsen. A bounded arithmetic theory for constant depth threshold circuits. In Petr Hájek, editor, *GÖDEL '96*, volume 6 of *Springer Lecture Notes in Logic*, pages 224–234, 1996.

[23] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1995.

[24] Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.

[25] Daniel Leivant. A foundational delineation of computational feasibility. In *Proceedings of IEEE 6th Annual Symposium on Logic in Computer Science*, 1991.

[26] Daniel Leivant. Peano theories and their computable functions. Manuscript (extended abstract), December 1992.

[27] Alexander A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. In Peter Clote and Jeffrey Remmel, editors, *Feasible Mathematics II*, volume 13 of *Progress in Computer Science and Applied Logic*, pages 344–386, Boston, 1995. Birkhäuser.

[28] Walter L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, June 1981.

[29] Gaisi Takeuti. Frege proof system and TNC^0. Manuscript, 1994.