Propositional Branching Program Proofs and Logics for L and NL

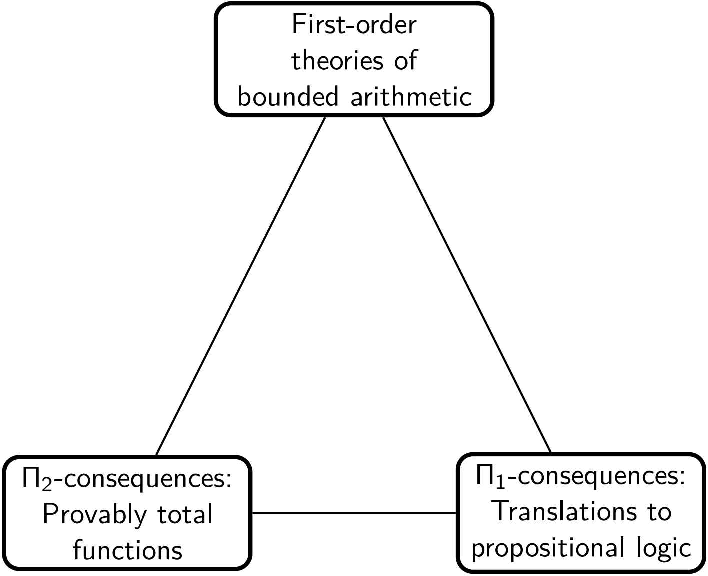
Sam Buss
U.C. San Diego

Logic Seminar
Prague IM-CAS via Zoom
December 14, 2020

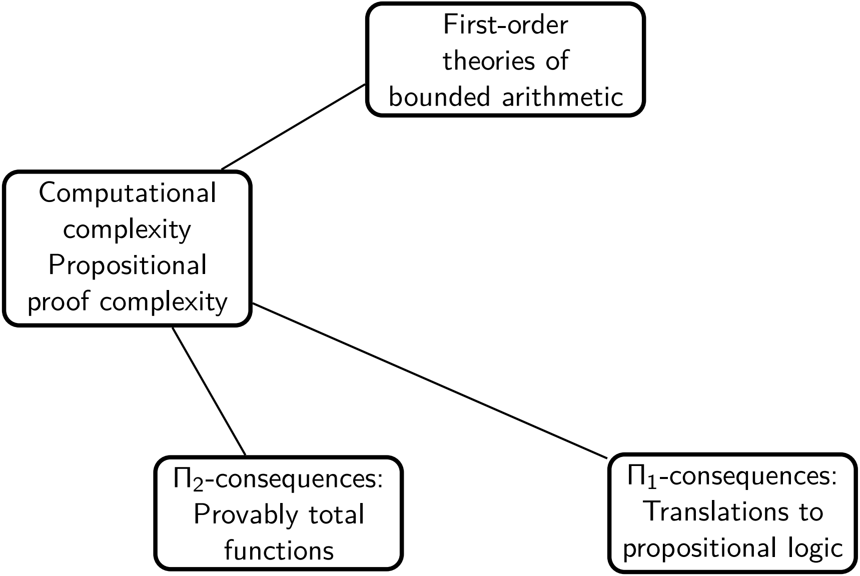
(joint work with Anupam Das and Alexander Knop)

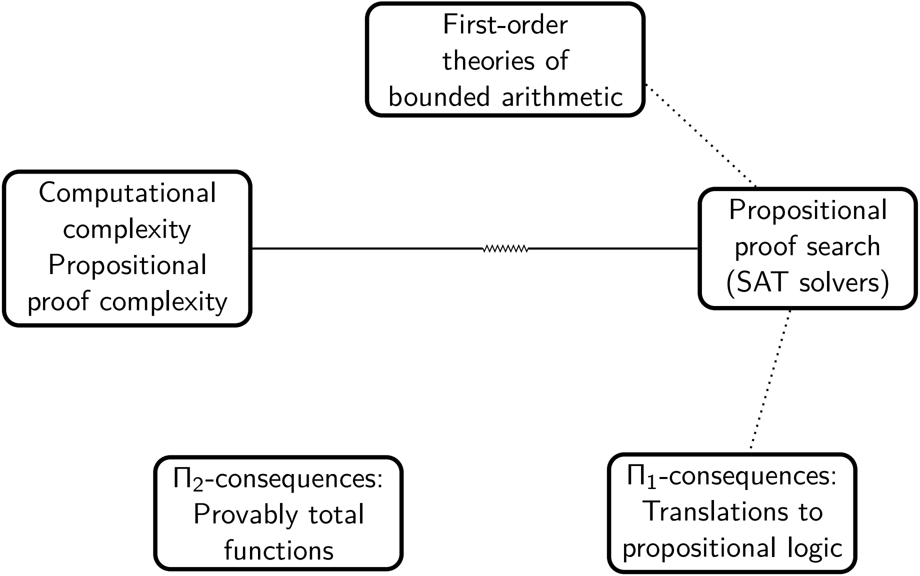
This talk

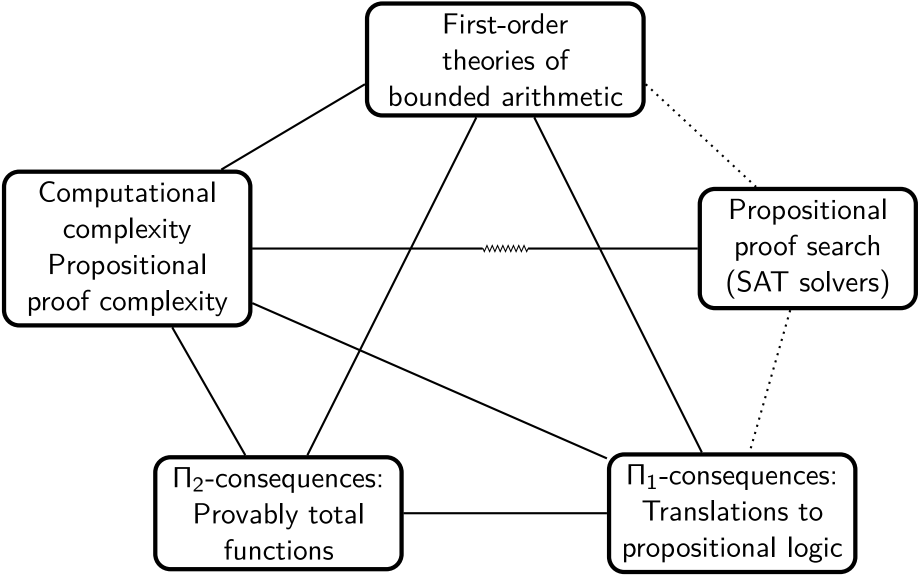
- Propositional and second-order systems for logspace and non-deterministic log space.
- Motivation is for use for propositional translations from bounded arithmetic.
- Main portion of the talk will describe different propositional proof systems, including new systems that can work with formulas expressing (non-uniform) L and NL properties.

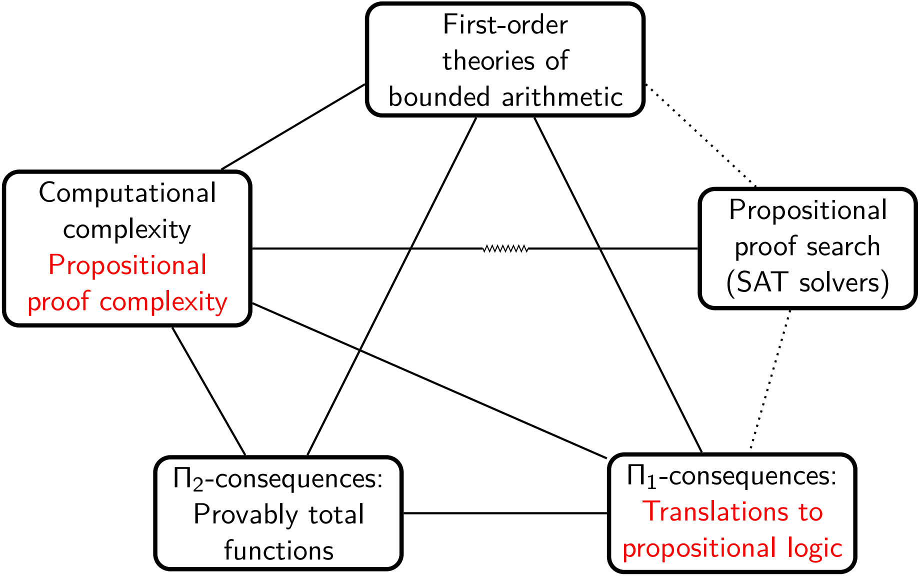


First-order theories of bounded arithmetic
Π_2-consequences: Provably total functions
Π_1-consequences: Translations to propositional logic


First-order theories of bounded arithmetic
Computational complexity Propositional proof complexity
Π_2-consequences: Provably total functions
Π_1-consequences: Translations to propositional logic


First-order theories of bounded arithmetic
Computational complexity Propositional proof complexity
Propositional proof search (SAT solvers)
Π_2-consequences: Provably total functions
Π_1-consequences: Translations to propositional logic


First-order theories of bounded arithmetic
Computational complexity Propositional proof complexity
Propositional proof search (SAT solvers)
Π2-consequences: Provably total functions
Π1-consequences: Translations to propositional logic


First-order theories of bounded arithmetic
Computational complexity
Propositional proof complexity
Propositional proof search (SAT solvers)
Π_2-consequences: Provably total functions
Π_1-consequences: Translations to propositional logic

Propositional proof systems (general)

- A proof system is always defined relative to some language of formulas. Formulas will use variables and connectives from
  - Input variables: $p_1, p_2, \ldots$, which appear in proved formulas.
  - Other free variables: $a, b, \ldots$ and bound variables $x, y, \ldots$ (in quantified propositional logics)
  - Extension variables: $e_1, e_2, \ldots$.
  - Negation ($\neg A$ or $\overline{p}$)
  - Disjunction ($\vee$), Conjunction ($\wedge$)
  - Decision (a.k.a "Case" or "Select"):

$$(ApB) \quad \text{means} \quad \text{"If } p \text{ then } B \text{ else } A\text{"}$$

- Lines in a proof will be ***sequents*** of (multisets of) formulas

$$A_1, \ldots, A_k \longrightarrow B_1, \ldots, B_\ell$$

  meaning $\bigwedge_i A_i \to \bigvee_j B_j$.

- Proofs may be allowed to be ***dag-like*** or required to be ***tree-like***.

Proof systems in this talk use the sequent calculus, with initial axioms including $A \to A$ for A atomic.
They all allow structural rules:

$$\textit{weak-l}\,\frac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \textit{weak-r}\,\frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A}$$

$$\textit{contract-l}\,\frac{A, A, \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \textit{contract-r}\,\frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A}$$

$$\textit{Cut}\,\frac{\Gamma \longrightarrow \Delta, A \qquad A, \Gamma \longrightarrow , \Delta}{\Gamma \longrightarrow \Delta}$$

Extended Frege proofs [Cook, Reckhow'75, '76, '79]

- Extended Frege proofs allow connectives $\neg$, $\vee$, $\wedge$.
- Allows extension axioms (initial sequents)

$$e_i \leftrightarrow A_i, \qquad i = 1, \ldots, \ell$$

where e_i does not appear in A_j for $j \geq i$.[1]

- The sequent calculus formulation eLK has rules of inferences:

$$\wedge\text{-}l\ \frac{A, B, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta} \qquad \wedge\text{-}r\ \frac{\Gamma \longrightarrow \Delta, A \qquad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}$$

$$\vee\text{-}l\ \frac{A, \Gamma \longrightarrow \Delta \qquad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta} \qquad \vee\text{-}r\ \frac{\Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A \vee B}$$

$$\neg\text{-}l\ \frac{\Gamma \longrightarrow \Delta, A}{\neg A, \Gamma \longrightarrow \Delta} \qquad \neg\text{-}r\ \frac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \neg A}$$

[1] $e_i \leftrightarrow A_i$ denotes the two sequents $e_i \longrightarrow A_i$ and $A_i \longrightarrow e_i$.

Properties of eLK proofs:

- $\mathrm{Tree\text{-}eLK}$ p-simulates eLK. [Krajíček]
- Best lower bounds known for eLK proofs for sequents of length n are $\Omega(n^2)$. [c.f. B'95] Thus, it is open whether eLK has polynomial size proof for all tautologies.
- Polynomial-size formulas with extension variables effectively represent polynomal size (Boolean) circuits. Thus eLK-proofs are able to reason about (non-uniform) polynomial-time properties.
- Correspondingly, theorems of the equational theory PV and $\forall\Pi_1^{\mathrm{b}}$-theorems of the first-order theory S_2^1 have propositional translations to eLK-formulas which have polynomial size eLK proofs. [Cook'75, ...]

The Circuit-Frege proof system (CF) is a variant of the extended Frege proof system in which circuits in are used (in sequents) instead of formulas. [Jerábek'04].

CF is defined similarly to eLK, but:

- CF uses circuits instead of formulas. Circuits are represented straightforwardly with labelled acyclic directed graphs.
- Extension variables are not permitted.
- The similarity inferences are allowed:

$$\frac{A, \Gamma \longrightarrow \Delta}{A', \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta, A}{\Gamma \longrightarrow \Delta, A'}$$

  where A and A' are *similar* as circuits.
- "Similar" circuits are ones that can equated by identifying equivalent gates. Similarity in is $\mathrm{coNL} \subseteq \mathrm{P}$.
  (Assuming gate inputs are ordered?)

Hence: Recognizing valid eLK or valid CF proofs is in P.

Frege proofs

Frege proofs are defined like extended Frege proofs, but disallowing the use of extension variables.

- LK is the sequent calculus formulation of Frege proofs.
- LK is defined exactly like eLK but disallowing the extension rule initial sequents.

Recognizing valid LK proofs can be done in alternating logarithmic time (Alogtime), i.e., in NC^1 or with polynomial size formulas.

Properties of LK proofs:

- $\mathrm{Tree}\text{-}\mathrm{LK}$ p-simulates LK. [Krajíček]
- Best lower bounds known for LK proofs for sequents of length n are $\Omega(n^2)$. [Buss'95] Thus, it is open whether LK has polynomial size proof for all tautologies.
- Also open: does LK p-simulate eLK?
- The Boolean formula value problem is complete for Alogtime. Thus LK-proofs are able to reason about NC^1 (nonuniform Alogtime) properties.
- Correspondingly, $\forall\Sigma_0^B$-theorems of the second-order theory VNC^1 have propositional translations to LK-formulas which have polynomial size LK proofs. [Cook-Morioka'05, Cook-Nguyen'10]

Constant depth Frege proofs

For integers d, the depth d LK-proof system, denoted d-LK, is the system LK modified to

(a) allow negations to apply only to variables, and
(b) require all formulas appearing in sequents in the proof have depth d.

Here,

- Constant depth Frege proofs use connectives $\land$ and $\lor$, and negation only on variables, $\overline{p}_i$.
- The ***depth*** of an LK-formula is the number of alternating levels of $\land$'s and $\lor$'s.
  E.g., a conjunction of disjunction of literals has depth 2.
- d-LK-proofs use sequents of depth d formulas, in essence, are disjunctions of depth d formulas.

Constant depth Frege proofs were first used for propositional translations of bounded arithmetic proofs by [Paris-Wilkie'85], for translation of the pigeonhole principle.

Properties of constant depth Frege proofs.

- d-LK is p-equivalent to Tree-$(d{+}1)$-LK, for $d \geq 0$ — for sequents of depth d formulas. [Krajíček, Razborov; see Beckmann-B]
- Proving a sequent of depth d formulas is equivalent to refuting a set of depth d-formulas.
- With aggressive encoding of the syntax of proofs, sequents and formulas, the validity of a d-LK proof can be verified in co-nondeterministic logarithmic time. (The same holds for all of our systems, c.f. [Beckmann-B'17])
  This is a uniform version of depth $1\frac{1}{2}$ formulas.

Half-integer depths, or Σ depth [Krajíček'94]

Let d be an integer.

Intuition: A depth $(d+\frac{1}{2})$ formula is a depth $d+1$ formula, but with the restriction that the fanins of gates at depth $d+1$ are logarithmically bounded by the size S of the formula.

More formally: A depth $(d+\frac{1}{2})$ formula A is a depth $d+1$ formula. Its ***size*** is the maximum of the number of symbols in A and of 2^f for f the largest number of literals in any conjunction or disjunction at depth $(d+1)$ in A.

Propositional translations of the bounded arithmetic theory T_2^{d+1} or S_2^{d+2} naturally yield Tree-$(d+\frac{1}{2})$-LK proofs, or $(d-\frac{1}{2})$-LK proofs. (For $d \geq 0$.)
$\frac{1}{2}$-LK proofs are more commonly known as Res(log) proofs.

Stronger propositional theories

A number of stronger propositional theories have been proposed:

- Quantified propositional logic. [Dowd'78] [Krajíček-Pudlák'90]
- Implicit proof systems [Krajíček'04]
- Q-EFF [Goldberg-Papadimitriou'17]

However, this talk will consider instead weaker systems.

Part II: Proof systems for L and NL

A first proposal for logspace (L) was suggested by [Cook, unp.'01]. That system was based on Liar-Prover games [Pudlák-B'94]. In Cook's system, the Liar-Prover game was run on (dag-like) branching programs.

The theory GL^* for logspace [Perron'05]

GL^* is an extension of LK to quantified propositional logic, allowing only Σ-$\mathrm{CNF}(2)$ formulas as cut formulas.

Def'n $\mathrm{SAT}(2)$ is the set of instances of SAT in which no variable appears more than twice.
Thm. $\mathrm{SAT}(2)$ is logspace complete [Johannsen'04]

Def'n A pre-Σ-$\mathrm{CNF}(2)$ formula A is a purely existential, prenex, quantified propositional formula, such the

- The quantifier free part of A is a conjunction of clauses;
- If a bound variable occurs in two clauses with the same polarity, then the clauses clash on a free variable.

Defn. The Σ-$\mathrm{CNF}(2)$ formulas are the formulas that can be obtained by substituting quantifier-free formulas into a pre-Σ-$\mathrm{CNF}(2)$ formula.

GL^* inference rules

- GL^* proofs are tree-like.
- Quantifier inference rules: (b is an eigenvariable)

$$\exists\text{-}l\,\frac{A(b),\Gamma \longrightarrow \Delta}{\exists x A(x),\Gamma \longrightarrow \Delta} \qquad \exists\text{-}r\,\frac{\Gamma \longrightarrow \Delta, A(B)}{\Gamma \longrightarrow \Delta, \exists x A(x)}$$

$$\forall\text{-}l\,\frac{A(B),\Gamma \longrightarrow \Delta}{\forall x A(x),\Gamma \longrightarrow \Delta} \qquad \forall\text{-}r\,\frac{\Gamma \longrightarrow \Delta, A(b)}{\Gamma \longrightarrow \Delta, \forall x A(x)}$$

- Cuts are permitted only on (a) quantifier free formulas, and (b) $\Sigma\text{-}\mathrm{CNF}(2)$ formulas which do not contain any variable used as an eigenvariable.

(Without the restriction of (b), the system would simulate G_1^*.)

[Perron'05]
Gives a faithful propositional translation from VL to GL^*.

The theory GNL^* for nondeterministic logspace [Perron'09]

Def'n A $\Sigma\mathrm{Krom}$ formula A is a purely existential, prenex, quantified propositional formula, such the

- The quantifier free part of A is a conjunction of disjunctions C_i;
- Each C_i is a disjunction of at most two bound literals and of a clause involving only free variables.

Thm The set of true $\Sigma\mathrm{Krom}$ formulas is NL-complete. [Grädel'92]

Defn. GNL^* proofs are quantified LK proofs such that

- The proof is tree-like
- Cuts are permitted only on (a) quantifier free formulas, and (b) $\Sigma\mathrm{Krom}$ formulas which do not contain any variable used as an eigenvariable.

Thm GNL^* provides a faithful propositional translation of VNL. [Perron'09] (See also [Cook-Kolokolova'04])

New propositional theories for L and NL [B-Das-Knop'20]

We formulate new propositional proof systems corresponding to L and NL, based directly on branching programs. These are closer to Cook's suggestion of Liar-Prover games.

- The propositional proof systems **LDT** and **LNDT** use
  - Decision trees (DT formulas), or
  - Non-deterministic decision trees (NDT formulas).
- The propositional proof systems **eLDT** and **eLNDT** use
  - Branching programs (eDT formulas), or
  - Non-deterministic branching programs (eNDT formulas).

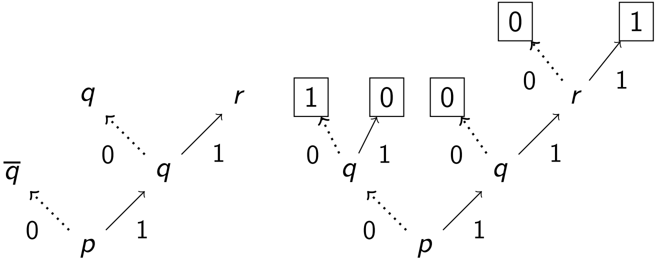
Notation: "DT" means "Decision Tree". An "eDT" ("extension Decision Tree") allows also extension variables, which converts the decision tree into a decision DAG, i.e., into a Branching Program (BP). "N" means "non-deterministic" ($\vee$ gates).

DT formulas (Decision trees)

Defn. DT-formulas are inductively defined by:

- **Atomic DT formulas:** p, $\overline{p}$, optionally 0 and 1.
- **Decision (or case/if-then-else) connective:** (ApB). Meaning "if p then B else A" or "case(p,B,A)".

Example:



These represent the equivalent formulas

$$\overline{q}\, p\, (q\, q\, r), \text{ and } (1q0)\, p\, (0\, q\, (0r1)).$$

A sequent calculus LDT for DT formulas:

Defn. LDT proofs use sequents of DT formulas.
Allowed inferences are:

- Initial sequents: (No inference rules for negation.)

  $p_i \longrightarrow p_i$ and $p_i, \overline{p}_i \longrightarrow$ and $\overline{p}_i \longrightarrow \overline{p}_i$ and $\longrightarrow p_i, \overline{p}_i$
- Structural inferences, cut rule, and
- Decision connective rules:

$$\textit{dec-l:}\ \frac{A, \Gamma \longrightarrow \Delta, p \qquad p, B, \Gamma \longrightarrow \Delta}{(ApB), \Gamma \longrightarrow \Delta}$$

$$\textit{dec-r:}\ \frac{\Gamma \longrightarrow \Delta, A, p \qquad p, \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, (ApB)}$$

eDT - Extension DT formulas; and proof system eLDT

Extension DT (eDT) formulas. Defined the same as DT formulas, but allowing an extension variable e_i as an atomic formula. Extension variables e can be used — unnegated — as atomic formulas, but cannot be used as decision literals:

- **Atomic eDT formulas:** p, $\overline{p}$, e.
- **Decision (or case/if-then-else) connective:** (ApB). Meaning "if p then B else A" or "case(p,B,A)".

Defining equations for extension variables have the form $\{e_i \leftrightarrow A_i\}_i$ where A_i is an eDT formula and e_i does not appear in A_j for $j \geq i$.

eDT formulas, together with their defining equations, express exactly (deterministic) branching programs.

Remark: Whenever working with a set of eDT formulas or a eLNDT proof, there is an implicit set of common extension variables with defining equations.

Defn. The proof system eLDT uses sequents of eDT formulas and has the initial sequents and inference rules of LDT plus the initial axioms

$$e_i \longrightarrow e_i \qquad \text{and} \qquad e_i \longrightarrow A_i \qquad \text{and} \qquad A_i \longrightarrow e_i.$$

Remark #2: For eDT's it is not important that extension variables cannot be negated, since it is easy to form the negation of a DT or an eDT formula.

Remark #3: It ***is*** important that extension variables cannot be used as decision variables. Otherwise, we could form $e_1 \wedge e_2$ as the formula $(e_1\, e_1\, e_2)$. With this construction, we could express any eLK formula.

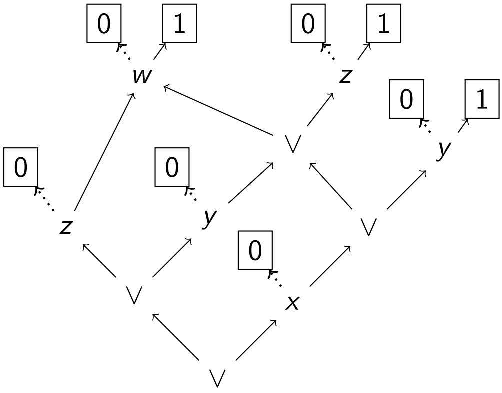
NDT formulas (Nondeterministic decision trees)

Defn. The NDT formulas are inductively defined like DT formulas but allowing also disjunction $\vee$ as a connective:

- **Atomic NDT formulas:** p, $\overline{p}$.
- **Decision (or case/if-then-else) connective:** (ApB). Meaning "if p then B else A" or "case(p,B,A)".
- **Disjunction ($\vee$) connective:** $(A \vee B)$.

NDT formulas are non-deterministic decision trees.

Example (Branching program as NDT/eNDT formula):



NDT formula: $((zzw)\vee(yy(w\vee z)))\vee(xx((w\vee z)\vee y))$.

eNDT formula: $((zzw)\vee(yye_1))\vee(xx(e_1\vee y))$ with the sole extension axiom $e_1 \leftrightarrow (w\vee z)$.

eNDT formulas and eLNDT formulas

Defn. eNDT formulas are defined by adding extension variables to the definition used for NDT formulas. Inductively, eNDT formulas are defined as:

- **Atomic NDT formulas:** p, $\overline{p}$, e.
- **Decision (or case/if-then-else) connective:** (ApB). Meaning “if p then B else A” or “case(p,B,A)”.
- **Disjunction ($\vee$) connective:** $(A \vee B)$.

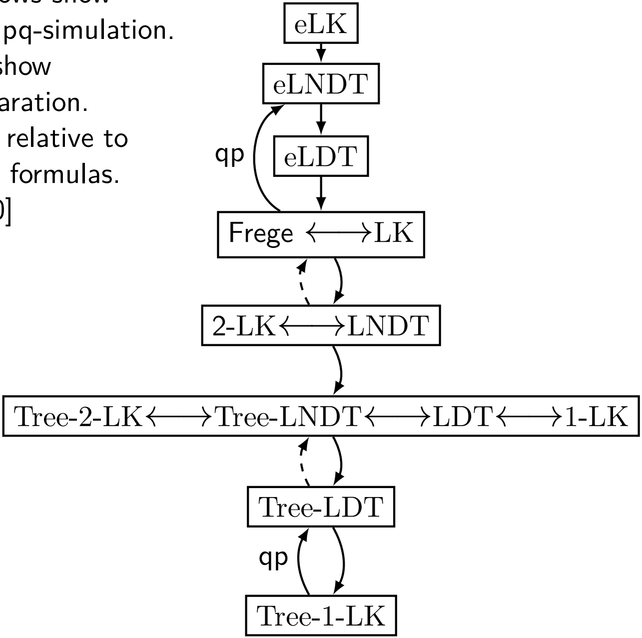
Extension axioms $\{e_i \leftrightarrow A_i\}_i$ are as before, now allowing A_i to be an eNDT formula.

LNDT and eLNDT proof systems

Defn. The LNDT proof system uses sequents of NDT formulas. It has the initial sequents of LDT and allows the structural, decision and $\vee$ inferences rules.

Defn. The proof system eLNDT uses sequents of eNDT formulas. It has the initial sequents of LDT and the initial sequents from extension axioms. It allows the structural, decision and $\vee$ inferences rules.

Thm. Solid arrows show p-simulation or pq-simulation. Dotted arrows show exponential separation. Simulations are relative to sequents of DT formulas. [B-Das-Knop'20]



Theorem ([BDK] - work in progress)

- *The* $\forall\Sigma_0^B$*-consequences of* VL *have natural propositional translations which have polynomial size* eLDT*-proofs.*
- VL *can prove the consistency of* eLDT *proofs.*
- *Any propositional proof system which is* VL*-provably sound is p-simulated by* eLDT*.*

Theorem ([BDK] - work in progress)

- *The* $\forall\Sigma_0^B$*-consequences of* VNL *have natural propositional translations which have polynomial size* eLNDT*-proofs.*
- VNL *can prove the consistency of* eLNDT *proofs.*
- *Any propositional proof system which is* VNL*-provably sound is p-simulated by* eLDT*.*

This includes (re)proving the Immerman-Szelepcsényi theorem that $\mathrm{NL} = \mathrm{coNL}$ in VNL. C.f. [Cook-Kolokolova'04, Perron'09].

Thank you for the virtual invitation to Prague!