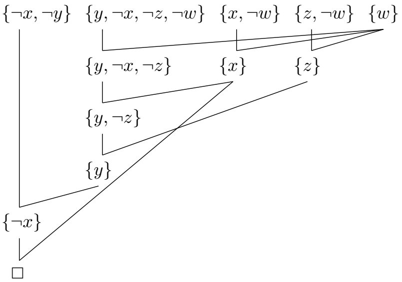
Reversible Pebble Games and the Relation Between Tree-Like and General Resolution Space

Florian Wörz
Universität Ulm
florian.woerz@uni-ulm.de

25. Jahrestagung der GI-Fachgruppe „Logik in der Informatik“ in Jena
22nd October, 2019

Joint work with **Jacobo Torán**, Universität Ulm

Just to Check We Are on the Same Page...



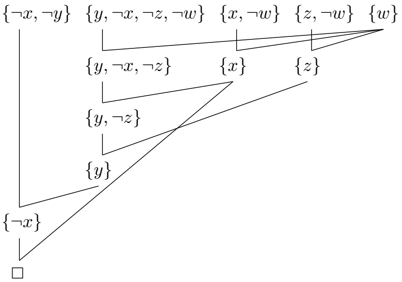
- Resolution: most studied proof system for refuting CNF formulas
- only one derivation rule:

$$\frac{B \lor x \quad C \lor \overline{x}}{B \lor C}$$

- Length of π = # of clauses in π
- Clause Space of $\pi = \max$ # of clauses in memory simultaneously during π

- Tree-Res, if refutation DAG is a tree ($\rightarrow$ maybe need to rederive clauses)

Just to Check We Are on the Same Page...



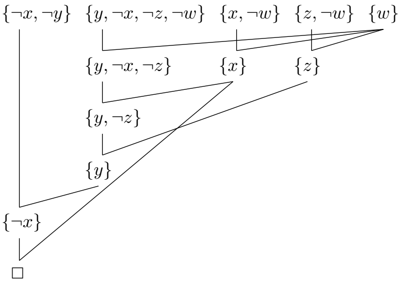
- Resolution: most studied proof system for refuting CNF formulas
- only one derivation rule:

$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

- Length of $\pi = \#$ of clauses in π
- Clause Space of $\pi = \max \#$ of clauses in memory simultaneously during π

- Tree-Res, if refutation DAG is a tree ($\rightarrow$ maybe need to rederive clauses)

Just to Check We Are on the Same Page...



- Resolution: most studied proof system for refuting CNF formulas
- only one derivation rule:

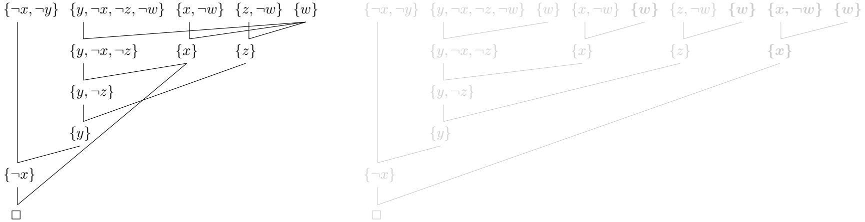
$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

- Length of $\pi = \#$ of clauses in π
- Clause Space of $\pi = \max \#$ of clauses in memory simultaneously during π

- Tree-Res, if refutation DAG is a tree ($\rightarrow$ maybe need to rederive clauses)

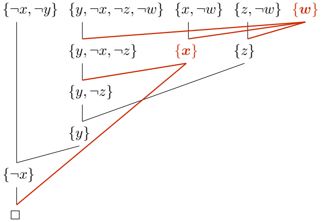
General vs. Tree-like Resolution Refutations

If a clause is needed more than once in a refutation, it has to be rederived each time.



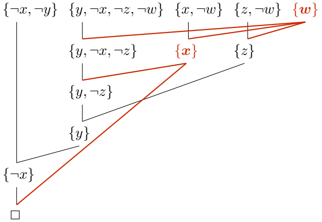
General vs. Tree-like Resolution Refutations

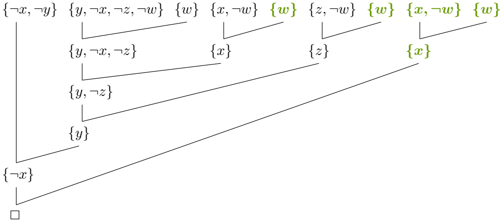
If a clause is needed more than once in a refutation, it has to be rederived each time.



General vs. Tree-like Resolution Refutations

If a clause is needed more than once in a refutation, it has to be rederived each time.





Complexity Measures for Resolution

For a complexity measure μ and a formula F

$$\mu(F \vdash \Box) := \min_{\pi : F \vdash \Box} \mu(\pi).$$

Prefix "Tree-" before a complexity measure indicates tree-like resolution.

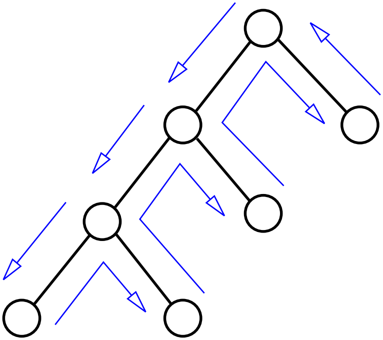
Complexity Measures for Resolution

For a complexity measure μ and a formula F

$$\mu(F \vdash \Box) := \min_{\pi : F \vdash \Box} \mu(\pi).$$

Prefix "Tree-" before a complexity measure indicates tree-like resolution.

Why Care About these Measures for Resolution?

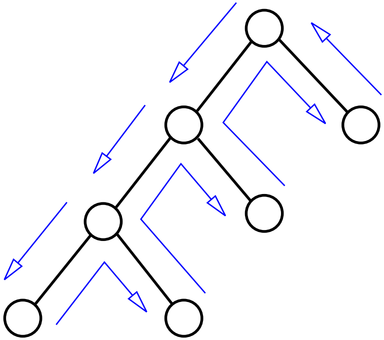


- After more than 50 years, DPLL is still the basis of most modern SAT Solvers (Chaff, zChaff, GRASP, MiniSAT, . . .).
- Tree-like resolultion and DPLL are p-equivalent proof systems.
- Experimental results (and even theoretical arguments): tree-space measures for resolution correlate well with the hardness of solving formulas with SAT solvers in practice.

[Järvisalo, Matsliah, Nordström, Živný '12: Relating Proof Complexity Measures and Practical Hardness of SAT]

[Ansótegui, Bonet, Levy, Manyà '08: Measuring the Hardness of SAT Instances]

Why Care About these Measures for Resolution?

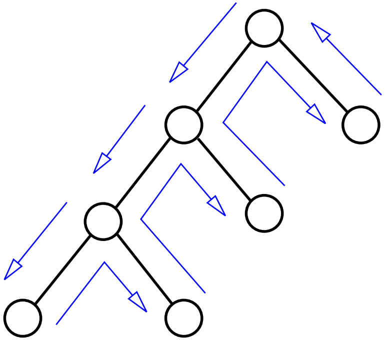


- After more than 50 years, DPLL is still the basis of most modern SAT Solvers (Chaff, zChaff, GRASP, MiniSAT, . . .).
- Tree-like resolultion and DPLL are p-equivalent proof systems.
- Experimental results (and even theoretical arguments): tree-space measures for resolution correlate well with the hardness of solving formulas with SAT solvers in practice.

[Järvisalo, Matsliah, Nordström, Živný '12: Relating Proof Complexity Measures and Practical Hardness of SAT]

[Ansótegui, Bonet, Levy, Manyà '08: Measuring the Hardness of SAT Instances]

Why Care About these Measures for Resolution?



- After more than 50 years, DPLL is still the basis of most modern SAT Solvers (Chaff, zChaff, GRASP, MiniSAT, . . .).
- Tree-like resolultion and DPLL are p-equivalent proof systems.
- Experimental results (and even theoretical arguments): tree-space measures for resolution correlate well with the hardness of solving formulas with SAT solvers in practice.

[Järvisalo, Matsliah, Nordström, Živný '12: Relating Proof Complexity Measures and Practical Hardness of SAT]

[Ansótegui, Bonet, Levy, Manyà '08: Measuring the Hardness of SAT Instances]

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know an almost optimal separation between general and tree-like resolution w. r. t. length:

$\exists$ a family $(F_n)_{n \in \mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\left(\Omega(\frac{L}{\log L})\right)$.

Matching upper bound of $\exp\left(\mathrm{O}\left(\frac{L \log \log L}{\log L}\right)\right)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

¿What about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know an almost optimal separation between general and tree-like resolution w. r. t. length:

$\exists$ a family $(F_n)_{n \in \mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\left(\Omega(\frac{L}{\log L})\right)$.

Matching upper bound of $\exp\left(\mathrm{O}\left(\frac{L \log \log L}{\log L}\right)\right)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

¿What about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know an almost optimal separation between general and tree-like resolution w. r. t. length:

$\exists$ a family $(F_n)_{n\in\mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\left(\Omega(\frac{L}{\log L})\right)$.

Matching upper bound of $\exp\left(\mathrm{O}\left(\frac{L \log\log L}{\log L}\right)\right)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

¿What about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know an almost optimal separation between general and tree-like resolution w. r. t. length:

$\exists$ a family $(F_n)_{n \in \mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\left(\Omega(\frac{L}{\log L})\right)$.

Matching upper bound of $\exp\left(\mathrm{O}\left(\frac{L \log \log L}{\log L}\right)\right)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

¿What about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know an almost optimal separation between general and tree-like resolution w. r. t. length:
$\exists$ a family $(F_n)_{n\in\mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\big(\Omega(\frac{L}{\log L})\big)$.

Matching upper bound of $\exp\big(\mathrm{O}\big(\frac{L\log\log L}{\log L}\big)\big)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

¿What about space?

Our Results / Outline of This Talk

In this talk we will:

I. a) characterize $\mathrm{Tree\text{-}CS}$ for special formulas defined over a DAG G in terms of a pebble game played on G.
   b) deduce the best known separation between $\mathrm{Tree\text{-}CS}$ and CS with this new characterization.

II. show that this is almost optimal by proving an upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS (for general formulas).

Our Results / Outline of This Talk

In this talk we will:

I.
  a) characterize $\mathrm{Tree\text{-}CS}$ for special formulas defined over a DAG G in terms of a pebble game played on G.
  b) deduce the best known separation between $\mathrm{Tree\text{-}CS}$ and CS with this new characterization.

II. show that this is almost optimal by proving an upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS (for general formulas).

Our Results / Outline of This Talk

In this talk we will:

I. a) characterize $\mathrm{Tree\text{-}CS}$ for special formulas defined over a DAG G in terms of a pebble game played on G.
   b) deduce the best known separation between $\mathrm{Tree\text{-}CS}$ and CS with this new characterization.

II. show that this is almost optimal by proving an upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS (for general formulas).

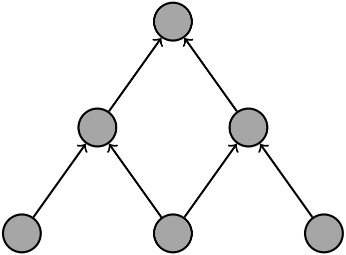
Part I

Separations for Pebbling Formulas

Pebble Games
(games played on graphs)

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

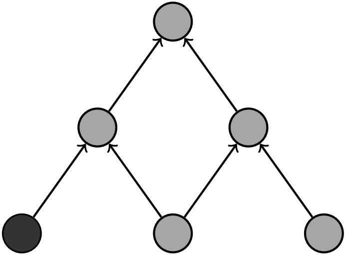


max # of pebbles
used at any point:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

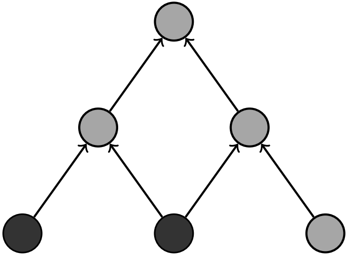


max # of pebbles
used at any point:
1

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

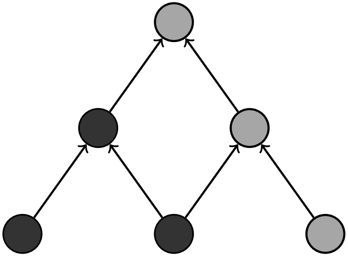


max # of pebbles
used at any point:
II

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

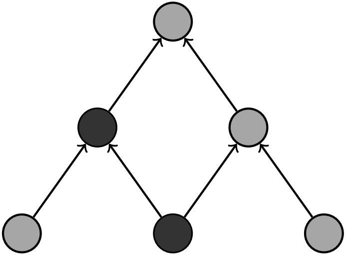


max # of pebbles
used at any point:
III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

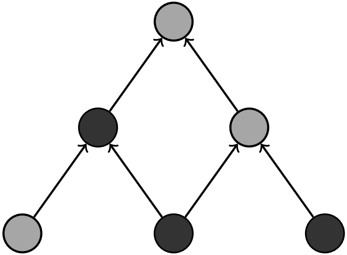


max # of pebbles used at any point: III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

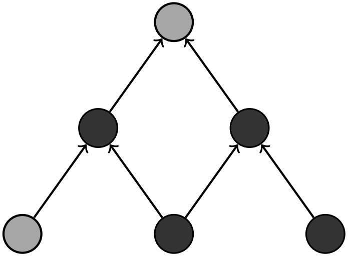


max # of pebbles used at any point:
III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

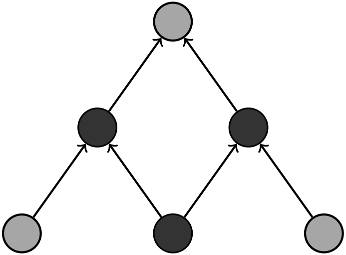


max # of pebbles used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

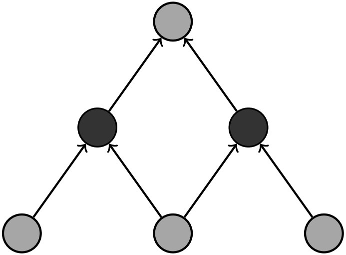


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.



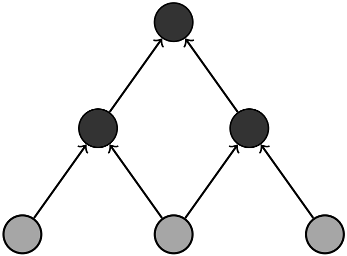
max # of pebbles used at any point:

IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

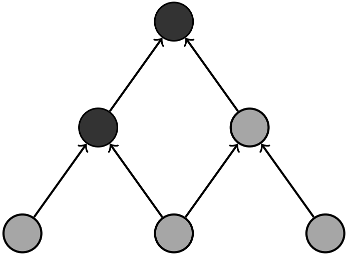


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

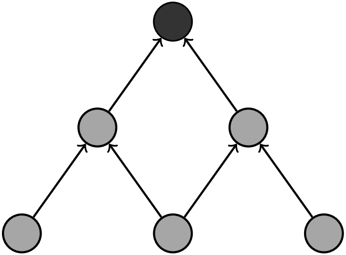


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

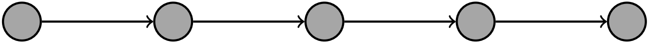


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point:

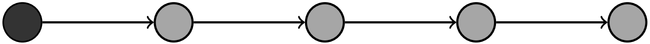


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: max # of pebbles used at any point: 1

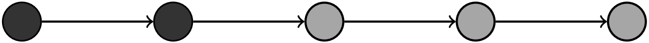


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: II

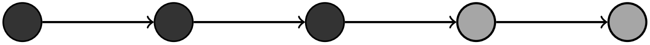


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

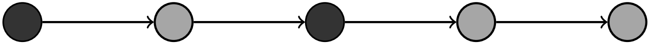


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

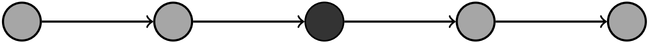


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

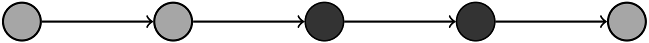


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

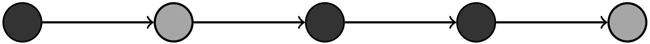


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

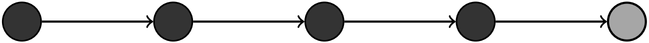


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

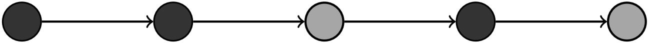


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

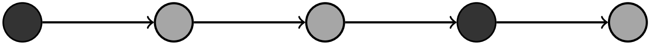


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

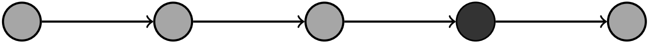


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: max # of pebbles used at any point: IIII

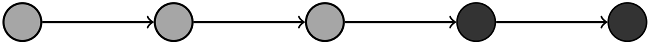


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

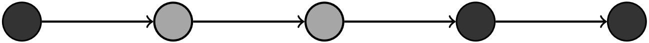


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

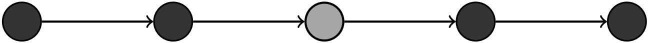


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

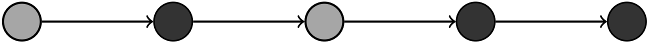


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

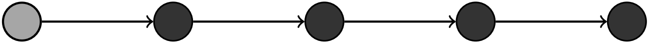


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

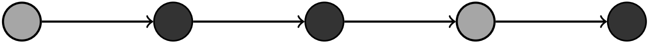


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

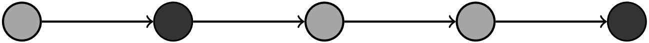


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

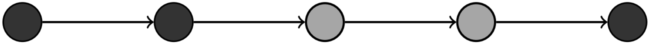


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

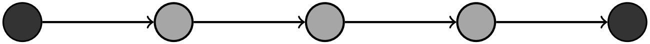


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

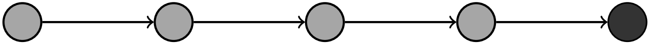


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII



Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

Complexity Measures for the Pebble Games

$$\text{Black}(G) := \min_{\text{black pebblings } \mathcal{P}} \Big(\max \; \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

$$\text{Rev}(G) := \min_{\text{rev. pebblings } \mathcal{P}} \Big(\max \; \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

Why even care about these pebbling prices?

Plethora of connections to resolution i. a.:

$\mathrm{CS}(\pi) = \text{Black}(G_\pi)$ $\forall \pi : F \vdash \Box$ [Esteban, Torán '01: Space bounds for resolution].

Complexity Measures for the Pebble Games

$$\mathsf{Black}(G) := \min_{\text{black pebblings } \mathcal{P}} \left(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \right)$$

$$\mathsf{Rev}(G) := \min_{\text{rev. pebblings } \mathcal{P}} \left(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \right)$$

Why even care about these pebbling prices?

Plethora of connections to resolution i. a.:

$\mathrm{CS}(\pi) = \mathsf{Black}(G_\pi) \ \forall \pi : F \vdash \Box$ [Esteban, Torán '01: Space bounds for resolution].

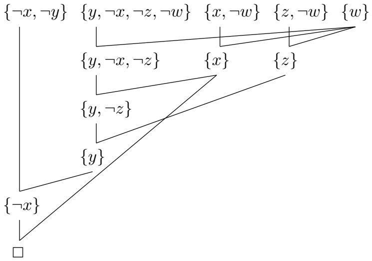
Complexity Measures for the Pebble Games

$$\mathsf{Black}(G) := \min_{\text{black pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

$$\mathsf{Rev}(G) := \min_{\text{rev. pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

Why even care about these pebbling prices?

Plethora of connections to resolution i. a.:
$\mathrm{CS}(\pi) = \mathsf{Black}(G_\pi) \ \forall \pi : F \vdash \Box$ [Esteban, Torán '01: Space bounds for resolution].



Pebbling Formulas (formulas over DAGs)

Pebbling Formula

Clauses of Peb_G:

u

v

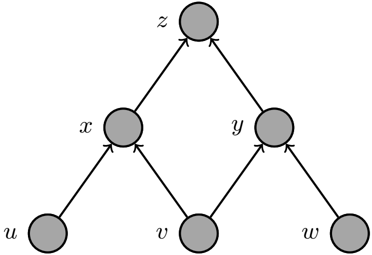
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

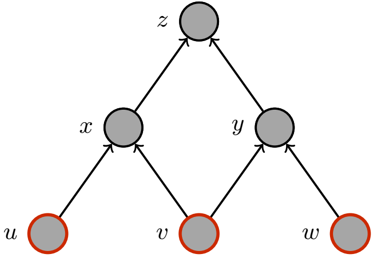
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

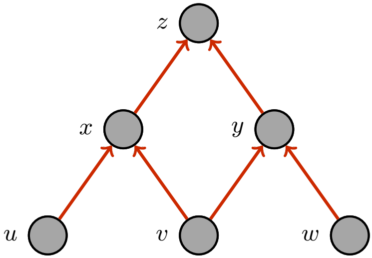
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

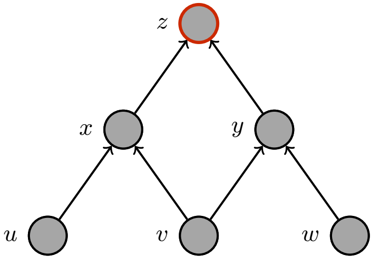
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

XORification $\oplus_2$

Make formulas slightly harder to refute

- For a technical reason we need the XORification of our pebbling formulas.
- (XORification being a common technique used in proof complexity).

- **Simple Idea:** Substitute each variable x with $x_1 \oplus x_2$ and expand result into CNF.

XORification $\oplus_2$

Make formulas slightly harder to refute

- For a technical reason we need the XORification of our pebbling formulas.
- (XORification being a common technique used in proof complexity).

- **Simple Idea:** Substitute each variable x with $x_1 \oplus x_2$ and expand result into CNF.

Reversible Pebbling meets Tree-CS in the Special Case of Pebbling Formulas

Reversible Pebbling meets Tree-CS

Theorem

For all DAGs G with a unique sink:

$$\mathsf{Rev}(G) + 2 \leq \text{Tree-CS}\big(\text{Peb}_G[\oplus_2] \vdash \Box\big) \leq 2 \cdot \mathsf{Rev}(G) + 2.$$

Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n

- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

[Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

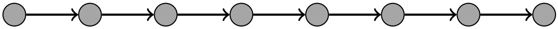
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\text{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\text{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- Black$(P_n) = \mathrm{O}(1)$ $\forall n \in \mathbb{N}$
- Rev$(P_n) = \Theta(\log n)$ $\forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

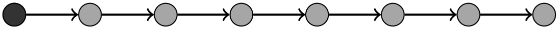
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\text{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\text{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- Black$(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- Rev$(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

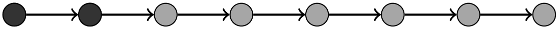
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

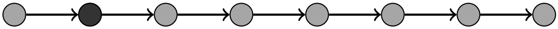
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

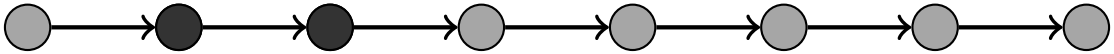
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\text{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\text{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- Black$(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- Rev$(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

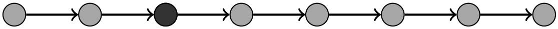
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

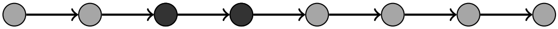
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

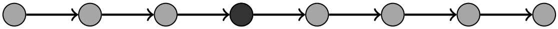
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

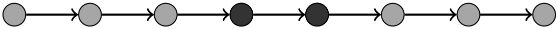
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

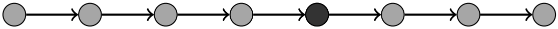
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

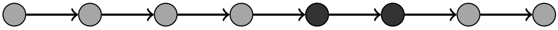
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

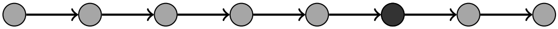
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\text{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\text{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- Black$(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- Rev$(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

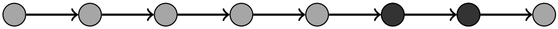
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \square) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \square\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

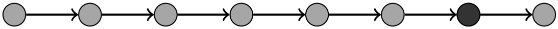
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \square) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \square\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

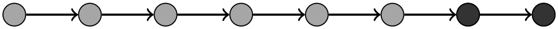
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\text{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\text{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- Black$(P_n) = \mathrm{O}(1)$ $\forall n \in \mathbb{N}$
- Rev$(P_n) = \Theta(\log n)$ $\forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

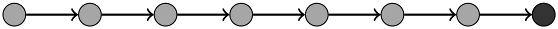
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

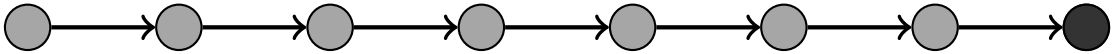
Obtaining Space-Separations with Pebble games (1/3)

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\text{Tree-CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

Example: Path graphs P_n of length n

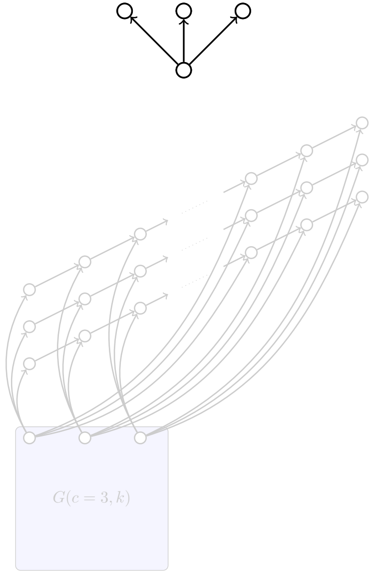


- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$ $\quad\exists$ Results for non-const. space?
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

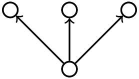
Obtaining Space-Separations with Pebble games (2/3)

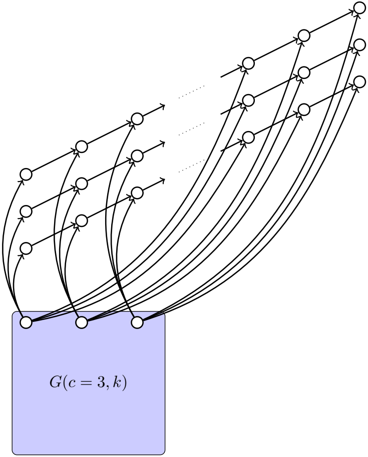
Non-constant black pebbling number and Black-Rev-separation:



Obtaining Space-Separations with Pebble games (2/3)

Non-constant black pebbling number and Black-Rev-separation:





Obtaining Space-Separations with Pebble games (3/3)

Conclusion: The best known separation

For any "slowly enough" growing space function $s(n)$ there is a family of pebbling formulas $\big(\mathrm{Peb}_{G_n}[\oplus_2]\big)_{n=1}^{\infty}$ with $\Theta(n)$ variables such that

- $\mathrm{CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \mathrm{O}\big(s(n)\big)$
- $\mathrm{Tree\text{-}CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \Omega\big(s(n)\log n\big)$.

¿Can we do any better?

Obtaining Space-Separations with Pebble games (3/3)

Conclusion: The best known separation

For any "slowly enough" growing space function $s(n)$ there is a family of pebbling formulas $\left(\mathrm{Peb}_{G_n}[\oplus_2]\right)_{n=1}^{\infty}$ with $\Theta(n)$ variables such that

- $\mathrm{CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \mathrm{O}\big(s(n)\big)$
- $\mathrm{Tree\text{-}CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \Omega\big(s(n)\log n\big)$.

¿Can we do any better?

Part II

Upper Bounds for Tree-CS for General Formulas

An upper bound for Tree-CS

How large can the gap between CS and Tree-CS grow?

Theorem

For any unsatisfiable formula F it holds

$$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2.$$

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

We will now prove this theorem… after introducing yet another two games.

An upper bound for $\mathrm{Tree\text{-}CS}$

How large can the gap between CS and $\mathrm{Tree\text{-}CS}$ grow?

Theorem

For any unsatisfiable formula F it holds

$$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2.$$

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

We will now prove this theorem... after introducing yet another two games.

An upper bound for $\mathrm{Tree\text{-}CS}$

How large can the gap between CS and $\mathrm{Tree\text{-}CS}$ grow?

Theorem

For any unsatisfiable formula F it holds

$$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2.$$

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

We will now prove this theorem... after introducing yet another two games.

A combinatorial characterization of Tree-CS (by a game played on formulas)

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over) • Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose")
• Plugs answer of Delayer in / chooses value for $*$	

The Prover-Delayer Game
A Combinatorial Characterisation for Tree-CS

Definition (Game value of the Prover-Delayer game)

Let F be an unsatisfiable CNF formula.
$\text{PD}(F) := \max$ pts. of Delayer on F against optimal strategy of Prover.

Theorem ([Esteban, Torán '03: A combinatorial char. of treelike res. space])

Let F be an unsatisfiable CNF formula. Then

$$\text{Tree-CS}(F \vdash \square) = \text{PD}(F) + 2.$$

The Prover-Delayer Game
A Combinatorial Characterisation for Tree-CS

Definition (Game value of the Prover-Delayer game)

Let F be an unsatisfiable CNF formula.
PD$(F) := \max$ pts. of Delayer on F against optimal strategy of Prover.

Theorem ([Esteban, Torán '03: A combinatorial char. of treelike res. space])

Let F be an unsatisfiable CNF formula. Then

$$\text{Tree-CS}(F \vdash \Box) = \mathsf{PD}(F) + 2.$$

The equivalence of Rev and R-Mc

Rev(G) is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

Pebbler	Colourer
• Places pebble on sink	
	• Colours it with red $\widehat{=} 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=} 0$ or blue $\widehat{=} 1$

Rev(G) is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

Pebbler	Colourer
• Places pebble on sink	
	• Colours it with red $\widehat{=} 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=} 0$ or blue $\widehat{=} 1$

Rev(G) is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

Pebbler	Colourer
• Places pebble on sink	
	• Colours it with red $\widehat{=}\, 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=}\, 0$ or blue $\widehat{=}\, 1$

$\mathsf{Rev}(G)$ is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

Pebbler	**Colourer**
• Places pebble on sink	
	• Colours it with red $\widehat{=} 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=} 0$ or blue $\widehat{=} 1$

$\text{Rev}(G)$ is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

Pebbler	**Colourer**
• Places pebble on sink	
	• Colours it with red $\widehat{=}\, 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=}\, 0$ or blue $\widehat{=}\, 1$

Rev(G) is hard to compute

Raz–McKenzie Game to the help

Given: A single sink DAG G

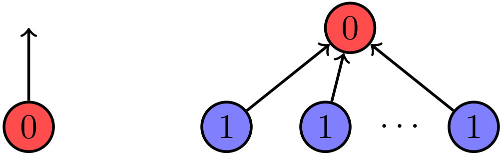
Two players take rounds... until Game Over...

Pebbler	**Colourer**
• Places pebble on sink	
	• Colours it with red $\widehat{=} 0$
• Chooses empty vertex	
	• Colours it red $\widehat{=} 0$ or blue $\widehat{=} 1$

Rev(G) is hard to compute

Raz–McKenzie Game to the help

Two players take rounds... until Game Over..., i. e., when we have:



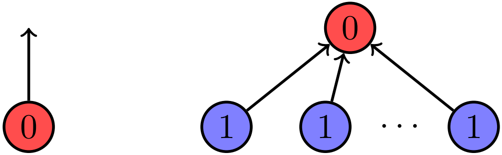
Either a red source **or** red vertex with all predecessors blue.

R-Mc(G) := smallest r s. th. Pebbler wins in $\leq r$ rounds
regardless of how Colourer plays

$\mathrm{Rev}(G)$ is hard to compute

Raz–McKenzie Game to the help

Two players take rounds... until Game Over..., i. e., when we have:



Either a red source **or** red vertex with all predecessors blue.

R-Mc(G) := smallest r s. th. Pebbler wins in $\leq r$ rounds
regardless of how Colourer plays

$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

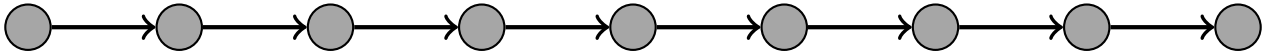
Rev(G) = R-Mc(G)

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall Rev(P_n) = R-Mc$(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



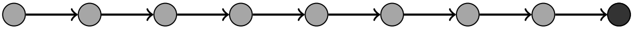
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



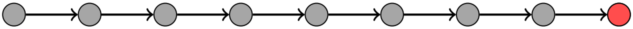
Rev(G) = R-Mc(G)

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

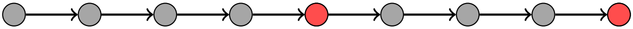
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



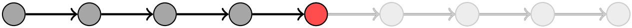
Rev(G) = R-Mc(G)

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



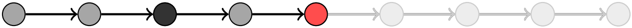
Rev(G) = R-Mc(G)

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



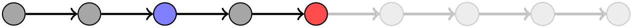
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



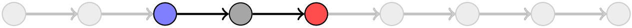
Rev(G) = R-Mc(G)

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



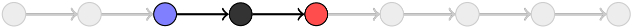
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



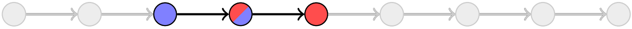
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



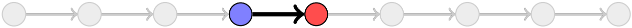
$\text{Rev}(G) = \text{R-Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



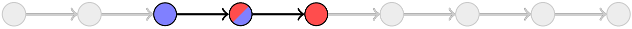
$\text{Rev}(G) = \text{R-Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\text{Rev}(G) = \text{R-Mc}(G)$$

Example: Recall $\text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: Recall $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

The Actual Proof

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C{\restriction}_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game

$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.
Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C{\restriction}_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses C $\longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.
Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C{\restriction}_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.
Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C{\restriction_\alpha}$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! □

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \square) \leq \min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) + 2$

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\square$

An upper bound for Tree-CS

Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

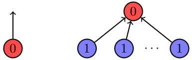
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$



An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \le \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

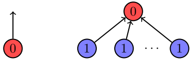
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! □



An upper bound for Tree-CS

Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

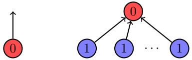
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! □



An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18: On space and depth in resolution] introduced amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \left(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\right)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04: Time and Space Complexity of Reversible Pebbling] $\mathrm{Rev}(G_\pi) + 2 \leq \min_{\mathcal{P}} \left(\mathrm{space}(\mathcal{P}) \cdot \log \mathrm{time}(\mathcal{P})\right) + 2$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathrm{space}(\mathcal{P})$ and length $\mathrm{time}(\mathcal{P})$. □

An upper bound for Tree-CS in terms of CS^*

[Razborov '18: On space and depth in resolution] introduced amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\text{Tree-CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04: Time and Space Complexity of Reversible Pebbling] $\mathrm{Rev}(G_\pi) + 2 \leq \min_{\mathcal{P}} \big(\mathrm{space}(\mathcal{P}) \cdot \log \mathrm{time}(\mathcal{P})\big) + 2$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathrm{space}(\mathcal{P})$ and length $\mathrm{time}(\mathcal{P})$. □

An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18: On space and depth in resolution] introduced amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04: Time and Space Complexity of Reversible Pebbling] $\mathsf{Rev}(G_\pi) + 2 \leq \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big) + 2$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathsf{space}(\mathcal{P})$ and length $\mathsf{time}(\mathcal{P})$. □

An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18: On space and depth in resolution] introduced amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04: Time and Space Complexity of Reversible Pebbling] $\mathsf{Rev}(G_\pi) + 2 \leq \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big) + 2$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathsf{space}(\mathcal{P})$ and length $\mathsf{time}(\mathcal{P})$. □

Take-Home Message

$\mathrm{Tree\text{-}CS}$ and CS are fundamentally different measures

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F

Take-Home Message

$\mathrm{Tree\text{-}CS}$ *and* CS *are fundamentally different measures*

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F (*)

(*) Some open questions hidden here. We've solved these for Tseitin formulas.

Take-Home Message

$\mathrm{Tree\text{-}CS}$ and CS are fundamentally different measures

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F (*)

(*) Some open questions hidden here. We've solved these for Tseitin formulas.

Thank you for your attention!