Playing Structure Rewriting Games with Formulas on States

Łukasz Kaiser

Ongoing work with help from D. Fischer, T. Ganzow, E. Abraham, U. Loup, Ł. Stafiniak

Mathematische Grundlagen der Informatik
RWTH Aachen

AlgoSyn

Aachen, 2009

Structure Rewriting Rules

Relational Structures and Embeddings

$$\sigma \ : \quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}}) \quad \to \quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \ldots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

Structure Rewriting Rules

Relational Structures and Embeddings

$$\sigma \;:\quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \dots, R_k^{\mathfrak{A}}) \quad\to\quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \dots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \dots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \dots, \sigma(a_{r_i}))$

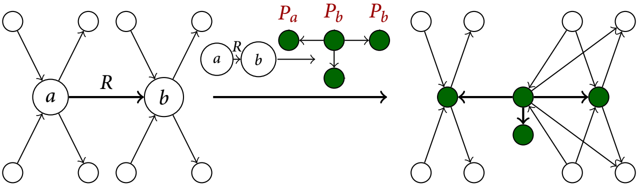
Rewriting Definition

$\mathfrak{B} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$ iff $B = (A \setminus \sigma(L)) \dot{\cup} R$ and,
for $M = \{(r, a) \mid a = \sigma(l), r \in P_l^{\mathfrak{R}} \text{ for some } l \in L\} \cup \{(a, a) \mid a \in A\}$,

$$(b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{B}} \;\Leftrightarrow\; (b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{R}} \textbf{ or } (b_1 M \times \dots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \emptyset.$$

(in the second case at least one $b_j \notin \mathfrak{A}$)

Rewriting Example



Structure Rewriting Games

Game arena is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Structure Rewriting Games

Game arena is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

Structure Rewriting Games

Game arena is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

Winning conditions:

- L_μ (or temporal) formula ψ with **MSO sentences** for predicates, or
- MSO formula φ to be evaluated on the **limit** of the play
  Limit of $\mathfrak{A}_0\mathfrak{A}_1\mathfrak{A}_2 \ldots = \left(\bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i,\ \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} R^{\mathfrak{A}_i}\right)$
- **Reach φ**: Player 1 **wins** if the play reaches $\mathfrak{A}$ s.t. $\mathfrak{A} \models \varphi$

Structure Rewriting Games

Game arena is a **directed graph** with:

- vertices partitioned into positions of **Player 0** and **Player 1**
- edges **labelled by rewriting rules**

Two interpretations of $\mathfrak{L} \to \mathfrak{R}$:

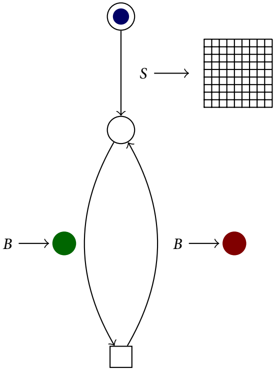
- **Existential**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$, the player chooses the embedding σ
- **Universal**: $\mathfrak{A}_{\text{next}} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}]$, **all** occurrences of $\mathfrak{L}$ are rewritten to $\mathfrak{R}$

Winning conditions:

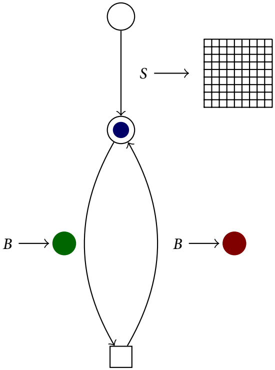
- L_μ (or temporal) formula ψ with **MSO sentences** for predicates, or
- MSO formula φ to be evaluated on the **limit** of the play
  Limit of $\mathfrak{A}_0\mathfrak{A}_1\mathfrak{A}_2 \ldots \;=\; \left(\bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} A_i,\ \bigcup_{n\in\mathbb{N}} \bigcap_{i\geq n} R^{\mathfrak{A}_i}\right)$
- **Reach φ**: Player 1 **wins** if the play reaches $\mathfrak{A}$ s.t. $\mathfrak{A} \models \varphi$

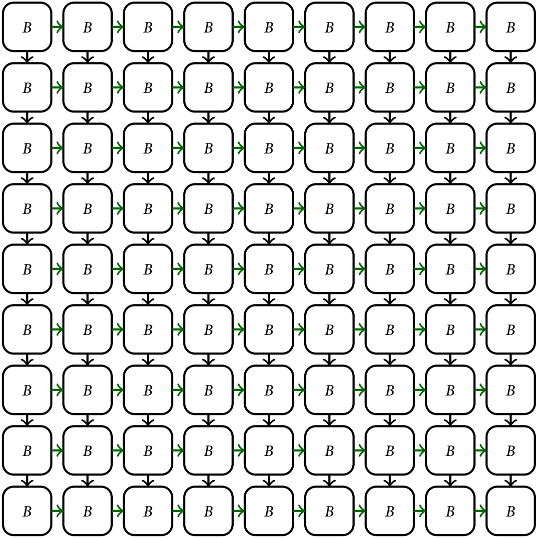
Motivation: many questions are **naturally defined as such games**:
constraint satisfaction, model checking, graph measures, games people play

Example Game: Gomoku (Connect–5)

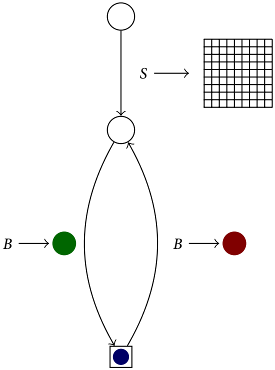


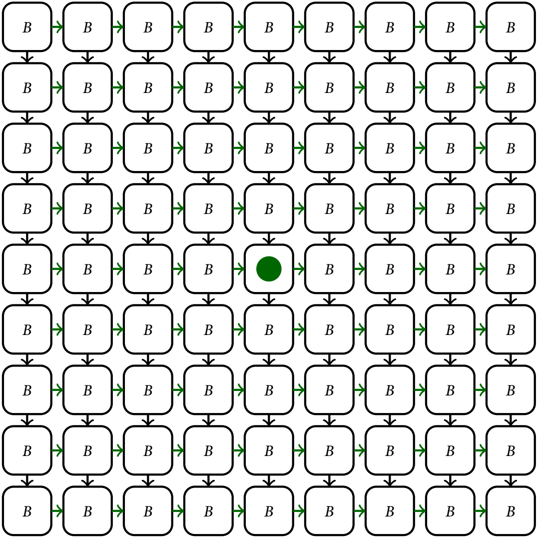
Example Game: Gomoku (Connect–5)



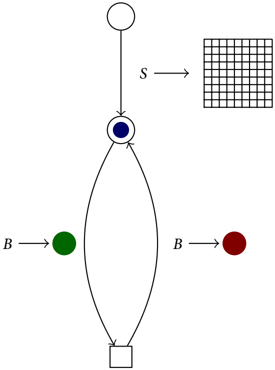


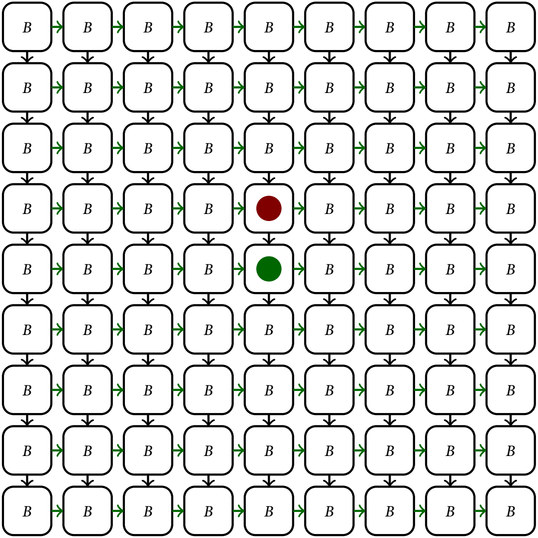
Example Game: Gomoku (Connect–5)



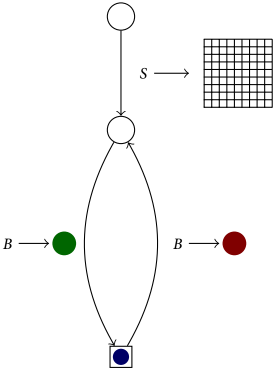


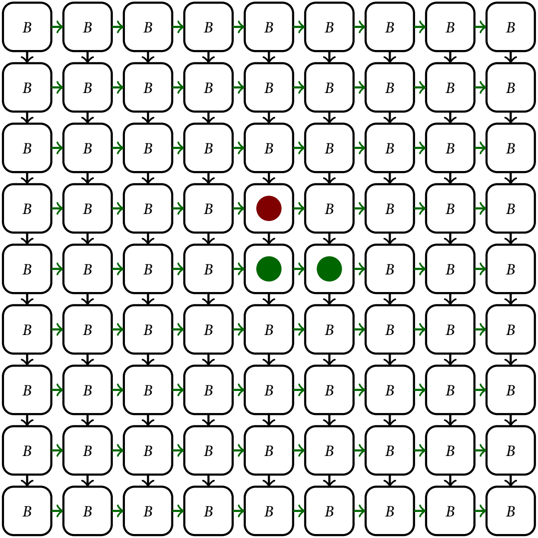
Example Game: Gomoku (Connect–5)



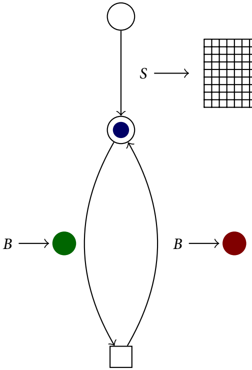


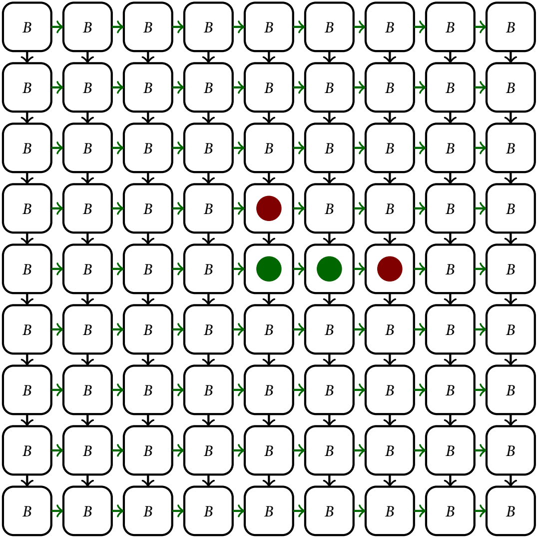
Example Game: Gomoku (Connect–5)





Example Game: Gomoku (Connect–5)





$$\exists x_1 \dots x_5 \Big(\bigwedge_{1 \le i \le 5} G(x_i) \wedge \Big(\bigwedge_{1 \le i \le 5} R(x_i, x_{i+1}) \vee \bigwedge_{1 \le i \le 5} C(x_i, x_{i+1}) \vee \bigwedge_{1 \le i \le 5} \exists y (R(x_i, y) \wedge C(y, x_{i+1})) \vee \bigwedge_{1 \le i \le 5} \exists y (R(x_i, y) \wedge C(x_{i+1}, y)) \Big) \Big)$$

Simple Structure Rewriting

Separated Structures: no element is in two non-terminal relations **(Courcelle, Engelfriet, Rozenberg, 1991)**

Separated: $\circ \xrightarrow{R} \circ \xrightarrow{a} \circ \xrightarrow{R} \circ$

Not Separated: $\circ \xrightarrow{\quad R \quad} \circ \xrightarrow{\quad R \quad} \circ$

Simple Structure Rewriting

Separated Structures: no element is in two non-terminal relations
(Courcelle, Engelfriet, Rozenberg, 1991)

Separated: $\circ \xrightarrow{R} \circ \xrightarrow{a} \circ \xrightarrow{R} \circ$

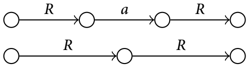
Not Separated: $\circ \xrightarrow{\quad R \quad} \circ \xrightarrow{\quad R \quad} \circ$

Simple Rule $\mathfrak{L} \to \mathfrak{R}$: $\mathfrak{R}$ is **separated** and $\mathfrak{L}$ is a **single tuple in relation**

Simple Structure Rewriting

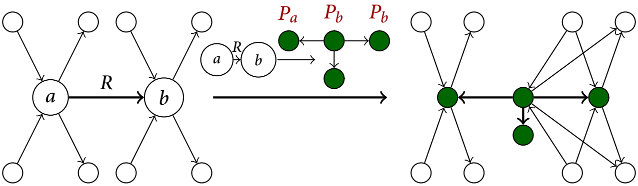
Separated Structures: no element is in two non-terminal relations
(Courcelle, Engelfriet, Rozenberg, 1991)

Separated:
Not Separated:



Simple Rule $\mathfrak{L} \to \mathfrak{R}$: $\mathfrak{R}$ is **separated** and $\mathfrak{L}$ is a **single tuple in relation**

Example



Previous Result

Logics

- $L_\mu[\mathrm{MSO}]$**:** Temporal properties expressed in L_μ (subsumes LTL) with properties of structures (states) expressed in MSO
- $\lim \mathrm{MSO}$**:** Property of the limit structure expressed in MSO

Theorem

- *Let R be a **finite** set of **(universal) simple structure rewriting rules,***
- *and φ be an $L_\mu[\mathrm{MSO}]$ or $\lim \mathrm{MSO}$ formula.*

*Then the set $\{\pi \in R^\omega : (\lim)S(\pi) \models \varphi\}$ is **ω-regular**.*

Corollary

Establishing the winner of (universal) finite simple rewriting games is decidable.

Implementing the Result

Using Tree Automata

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game
- **Result:** **memory overflow** on 2×2 grid
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Implementing the Result

Using Tree Automata

- **Tool**: **MONA**
  - Developed at **BRICS** since **1996** by Nils Klarlund and Anders Møller
  - Symbolic representation with **BDDs**
  - **Minimisation** at each step
- **Example:** a simple **tic-tac-toe** game
- **Result:** **memory overflow** on 2×2 grid
- **Problems** due to **inefficient coding**
  - **Bounded clique-width graphs** not good for MONA
  - Only **universal interpretation** decidable, must encode games

Present Approach

- Use **simulation** to detect **promising moves**
- Construct a **good (not necessarily optimal)** strategy
- **Perspective:** prove that the strategy is **winning**

Simulation-Based Game Playing

Game Playing Methods

- **General pattern**: **Minimax**
- ***α*-*β* pruning** and other optimisations
- Multiple special **heuristics**, **opening tables**
- **Common pattern**: need **position evaluation function**

Simulation-Based Game Playing

Game Playing Methods

- **General pattern**: **Minimax**
- *α-β* **pruning** and other optimisations
- Multiple special **heuristics**, **opening tables**
- **Common pattern**: need **position evaluation function**

Monte Carlo Evaluation Function

How to determine the **value of a position** v in a **general** game?

- **both players** play from v **randomly** a (large) number of times
- return the **ratio of wins** of the player

Simulation-Based Game Playing

Game Playing Methods

- **General pattern**: **Minimax**
- *α-β* **pruning** and other optimisations
- Multiple special **heuristics**, **opening tables**
- **Common pattern**: need **position evaluation function**

Monte Carlo Evaluation Function

How to determine the **value of a position** v in a **general** game?

- **both players** play from v **randomly** a (large) number of times
- return the **ratio of wins** of the player

Immediate Problems:

- Makes **trivially stupid** moves
- Very **flat lookahead**

Upper Confidence Bounds for Trees

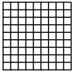
Build a tree during Monte-Carlo

- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success

Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

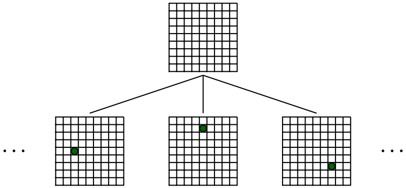
- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

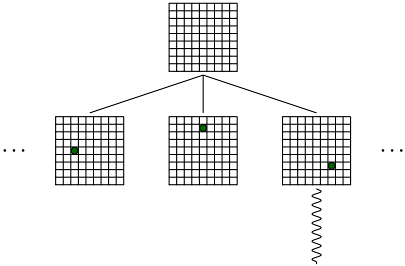
- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



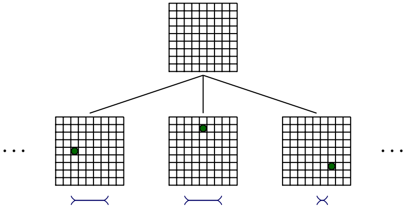
Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success

Pick Max Upper Confidence

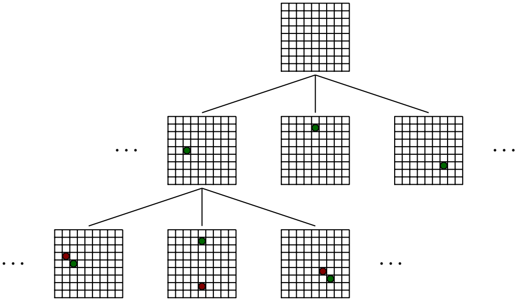
$$C \cdot \Delta \cdot \sqrt{\frac{\ln(n(v)+1)}{n(w)+1}}$$



Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

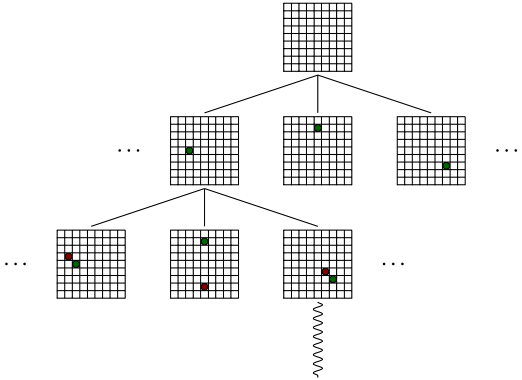
- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

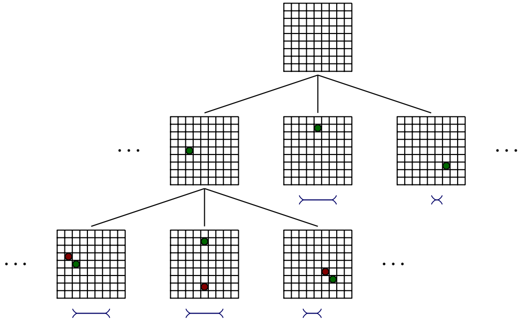
- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



Upper Confidence Bounds for Trees

Build a tree during Monte-Carlo

- **Idea:** memorise first Monte-Carlo moves, **Minimax** there
- **History:** encouraged by **MoGo** success



UCT for Structure Rewriting Games

Problems

- Random player is **stupid**
- Large number of **formula evaluations** (slow)

UCT for Structure Rewriting Games

Problems

- Random player is **stupid**
- Large number of **formula evaluations** (slow)

Improvements

- **Hints** for random player using **formulas**
- Makes it **even slower**: **improve solver**

UCT for Structure Rewriting Games

Problems

- Random player is **stupid**
- Large number of **formula evaluations** (slow)

Improvements

- **Hints** for random player using **formulas**
- Makes it **even slower**: **improve solver**

Results of Hints

- **Breakthrough**: **beat if possible** ca. 70% improvement
- **Gomoku**: **play near your stone** ca. 80% improvement

How to Make Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

How to Make Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

MSO is compositional:

$$\mathrm{Th}^k(\mathfrak{A} \oplus^{\text{connect}} \mathfrak{B}) = \mathrm{Th}^k(\mathfrak{A}) \oplus^{\text{connect}} \mathrm{Th}^k(\mathfrak{B})$$

Using this requires **multiple CNF-DNF conversions**

How to Make Solver Faster?

Solver Requirements

(1) **Obvious**: evaluate formulas fast

(2) **Repetition**: the same formula on **many structures**

(3) **Composition**: structures change **only slightly**

MSO is compositional:

$$\mathrm{Th}^k(\mathfrak{A} \oplus^{\text{connect}} \mathfrak{B}) = \mathrm{Th}^k(\mathfrak{A}) \oplus^{\text{connect}} \mathrm{Th}^k(\mathfrak{B})$$

Using this requires **multiple CNF-DNF conversions**

Current Solver Architecture

- **FO assignments**: represented directly
- **MSO assignments**: semi-symbolically

$$(1 \in X \wedge 2 \in X \wedge 3 \notin X) \vee (1 \notin X)$$

- Operations on MSO assignments: **use SAT solver**, **CNF-DNF again**
- Are **BDDs** better? **Unclear**

Outlook

Learning formulas

- Many **states collected during play**
- Formula should **separate high-confidence** good from bad states
- Preliminary tests: **good** but **parameter dependent**
- **Perspective**: describe **full winning region**

Outlook

Learning formulas

- Many **states collected during play**
- Formula should **separate high-confidence** good from bad states
- Preliminary tests: **good** but **parameter dependent**
- **Perspective**: describe **full winning region**

Extensions

- **Already supported**: **preconditions and postconditions**
- **Types of structures** (based on bounded clique-width)
- **Continuous dynamics** can be added
  - defined e.g. using $\mathbb{R}$**-structures and differential equations**
  - simple **quantitative logics** can be used

Outlook

Learning formulas

- Many **states collected during play**
- Formula should **separate high-confidence** good from bad states
- Preliminary tests: **good** but **parameter dependent**
- **Perspective**: describe **full winning region**

Extensions

- **Already supported**: **preconditions and postconditions**
- **Types of structures** (based on bounded clique-width)
- **Continuous dynamics** can be added
  - defined e.g. using **$\mathbb{R}$-structures and differential equations**
  - simple **quantitative logics** can be used

Thank You