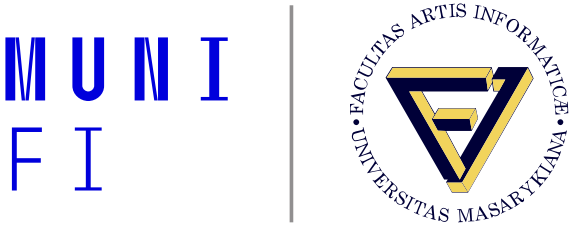



# A Rust Library For Manipulating Sentential Decision Diagrams

**Bc. Josef Podaný, advised by RNDr. Samuel Pastva, Ph.D.**

Faculty of Informatics, Masaryk University

February 3, 2025

# Motivation

- What is knowledge compilation?
  - *Knowledge compilation* is the process of translating a *knowledge base* into a *target compilation language* [3]
  - Target compilation language should be *tractable* and *succinct*
- Why is it important?
  - Model-based diagnosis & model-checking
  - Explainability in AI & neurosymbolic reasoning
  - Medical diagnostic systems

# Why Sentential Decision Diagrams?

- Tractable and succinct representation
- Strict superset of Ordered Decision Binary Diagrams (OBBDs)
- SDDs are exponentially more succinct than OBDDs [7]

# Vtrees

- A *vtree* for a set of variables **Z** is a full, rooted binary tree whose leaves are in one-to-one correspondence with the variables in **Z** [6]
- A vtree dissects a total variable order $\pi$ if and only if left-to-right traversal of the vtree visits the variables in the same order as $\pi$

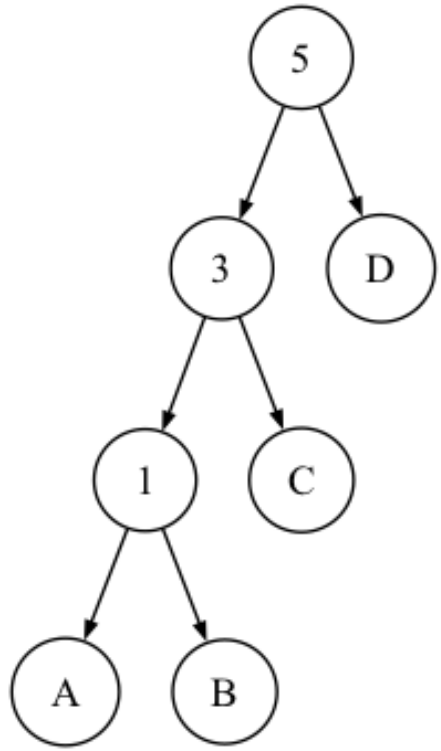



Figure 1: A chosen vtree dissecting order $(A, B, C, D)$.

# X-Partitions

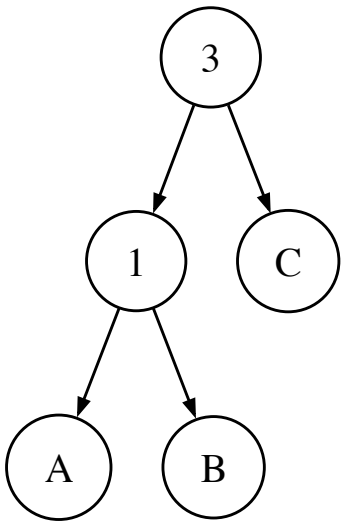
An $\boldsymbol{X}$-partition [2] is a structured way to decompose a Boolean function $f$ with respect to a given variable set $\boldsymbol{X}$. It is a set of pairs $(p_i, s_i)$ such that $f = \bigvee_i (p_i \wedge s_i)$ where

- Primes form a partition
- Each prime is consistent
- Every pair of distinct primes is mutually exclusive (strong determinism)
- The disjunction of all primes is valid

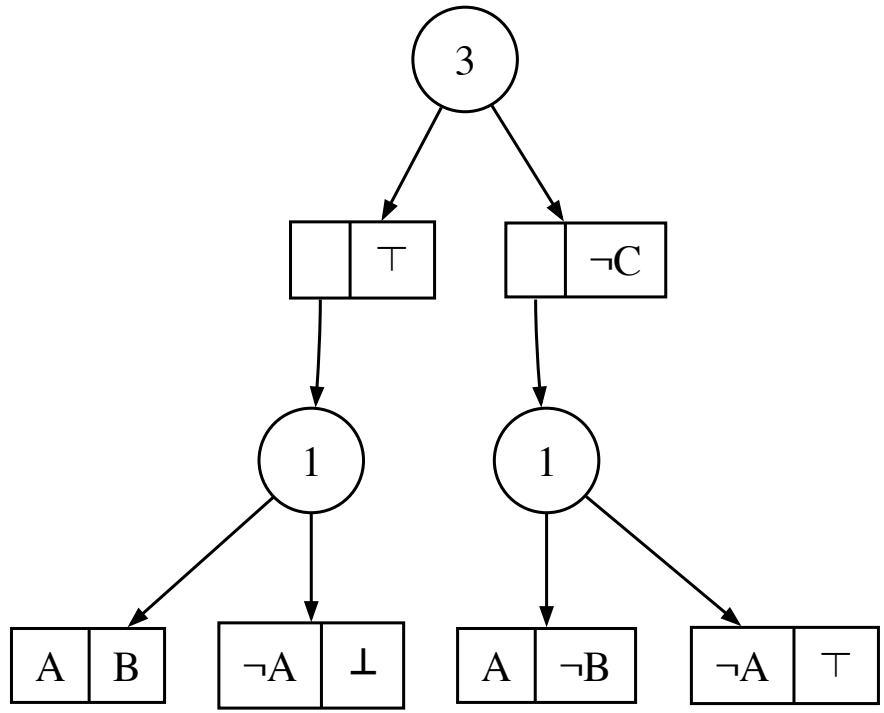
# Sentential Decision Diagrams

- SDD is a rooted, directed acyclic graph characterized by *vtrees* representing a Boolean function [2]
- Nodes of the graph are either terminals or ***X***-*partitions* branching on formulas
- Can be efficiently queried and combined (when *not canonical*)

# Example I: SDD Normalized for Left-Linear Vtree



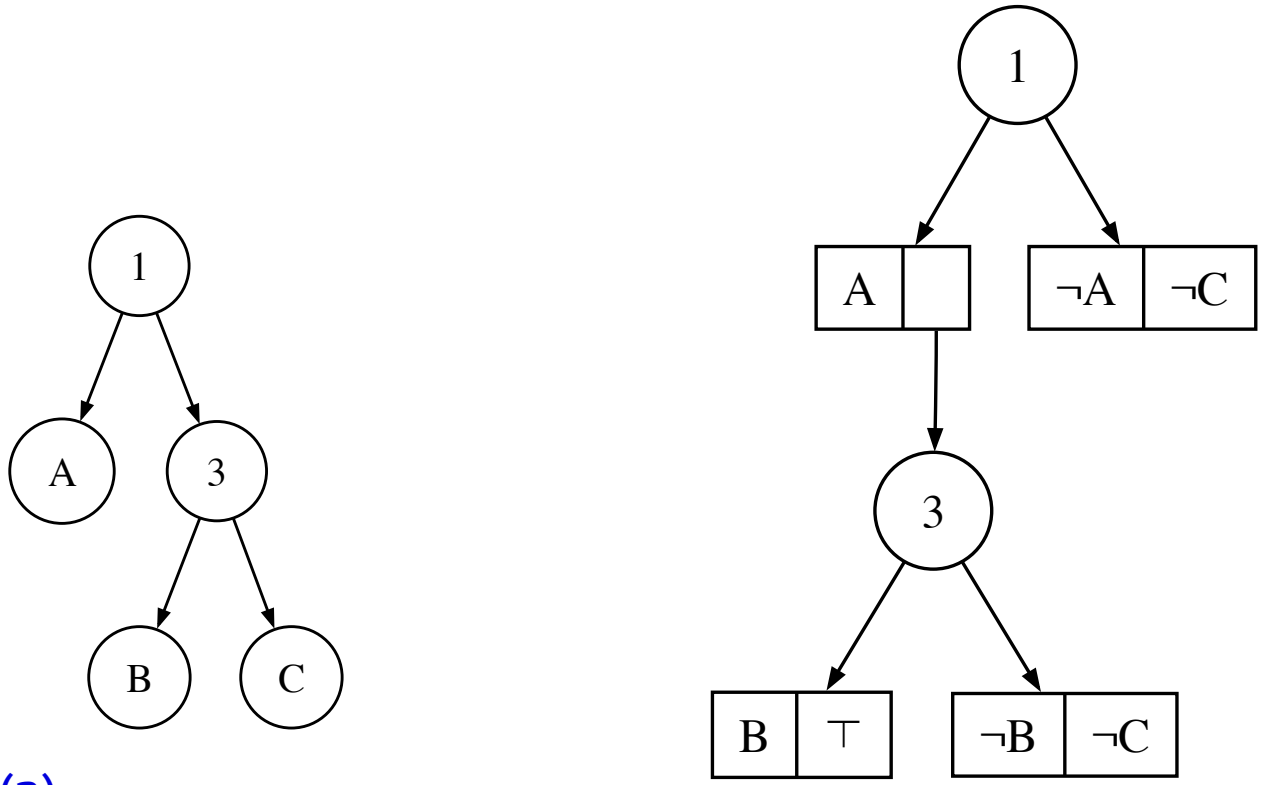

(a) Left-linear vtree




(b) SDD representing function $f$

Figure 2: SDD representing $f = (A \wedge B) \vee \neg C$ normalized for left-linear vtree.

# Example II: SDD Normalized for Right-Linear Vtree




(a) Right-linear vtree

(b) SDD representing function $f$

Figure 3: SDD representing $f = (A \wedge B) \vee \neg C$ normalized for right-linear vtree.

# Compilation & Minimization Methods

SDDs can be compiled either in a *bottom-up* or *top-down* [5] manner.

- Bottom-up: incremental building of the knowledge base
- Top-down: facts are compiled at once

SDDs can be minimized either *statically* or *dynamically* [1].

- Static minimization: construct appropriate vtree before compilation
- Dynamic minimization: adjust vtree as needed during compilation

# sddrs: Bottom-up SDD Compiler

- A bottom-up SDD compiler with dynamic minimization written in Rust[1]
- Available as a library[2] or an executable
- Supports
  - Tractable model counting, model enumeration, equivalence checking, and validity and consistency checking
  - Incremental compilation of facts into canonical SDDs
  - Dynamic minimization (via fragments)
  - Garbage collection

[1] `https://github.com/jsfpdn/sdd-rs/`
[2] `https://crates.io/crates/sddrs`

# Compiler Architecture

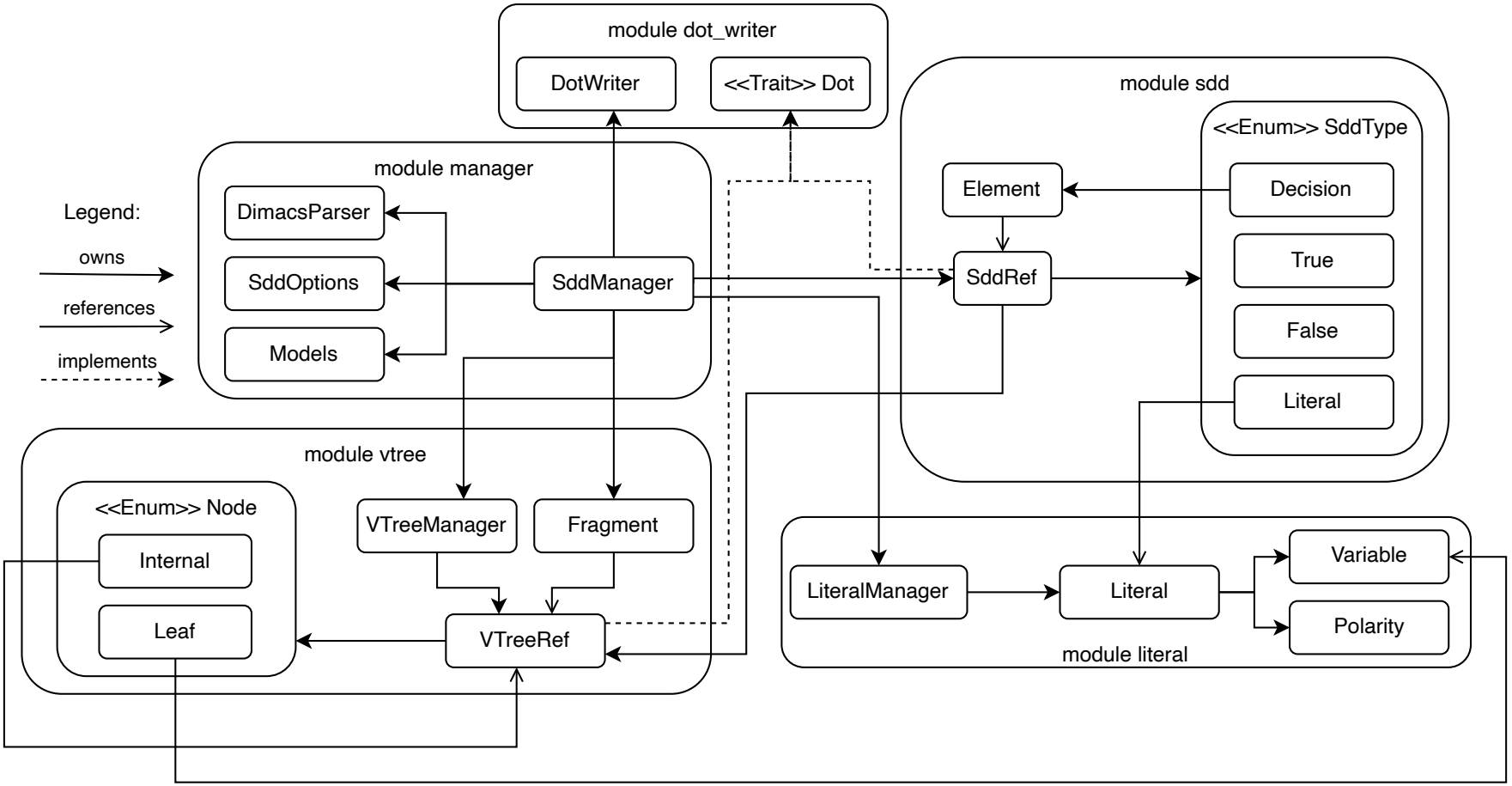



Figure 4: High-level decomposition of the compiler into individual modules. The most important structures and their relationships are presented for each module.

# Dynamic Minimization via Fragments

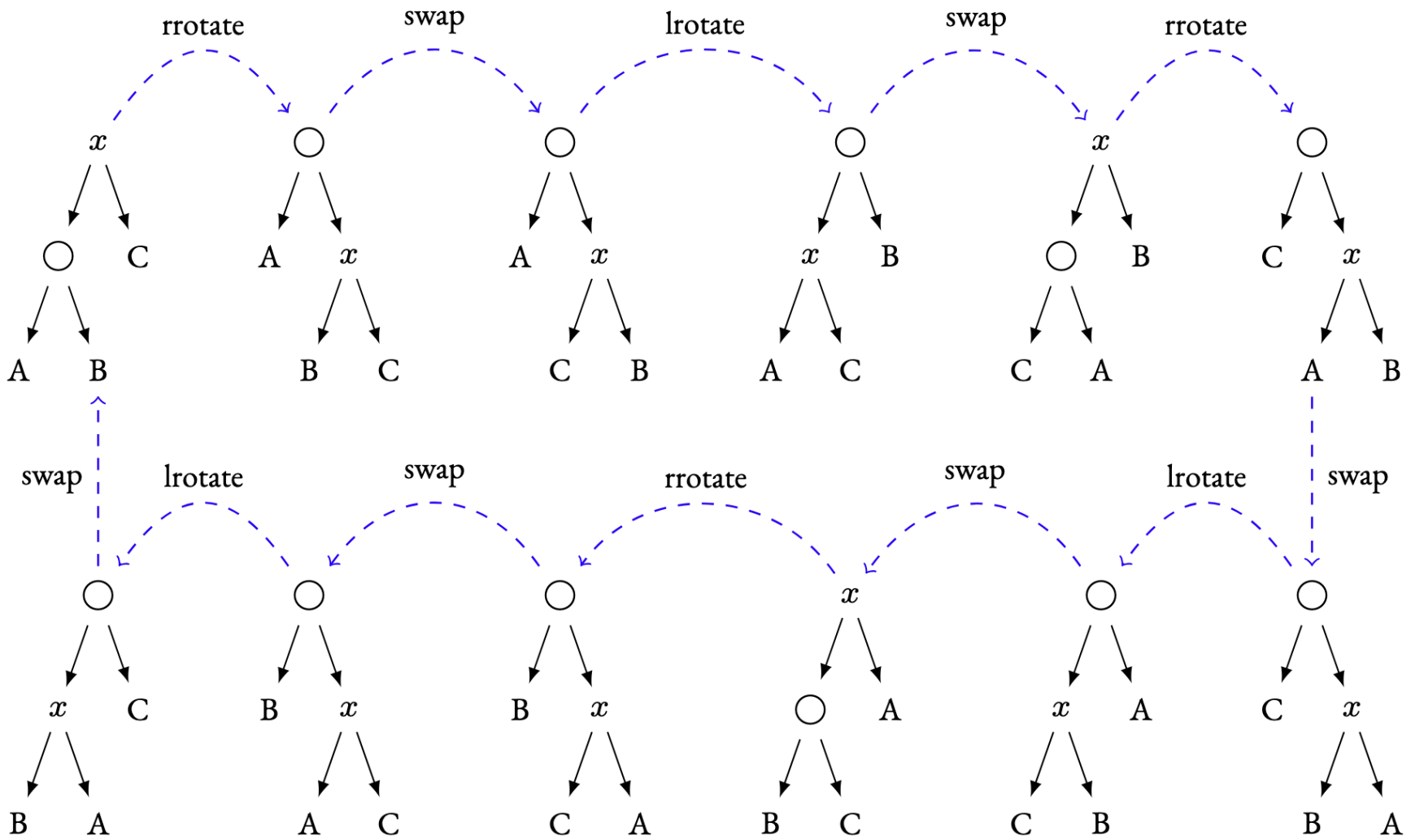



Figure 5: Exploration of all fragment states using swap and rotate operations.

# Overview of Benchmarks

- Compiled datasets from the SATLIB benchmark suite [4]
- Measured compilation time and size of compiled SDDs
- Benchmarked various configurations of sdd-package, rsdd, and sddrs

# Results

| configuration | dataset name | | | | | | |
|---|---|---|---|---|---|---|---|
| | uf20 | uf50 | uf75 | gcp | plan_a | plan_m | pigeon |
| **sddrs-static-gc-rl** | 0.073 | ✗ | ✗ | 72.140 | 0.149 | ✗ | 4.260 |
| rsdd-comp-linear | 0.002 | 2.030 | ✗ | 0.510 | 0.002 | 0.052 | 0.009 |
| sdd-package-rl | 0.038 | 12.404 | 2291.530 | 0.494 | 0.169 | 3.032 | 0.264 |

Table 1: SDD compilation times in seconds.

| configuration | dataset name | | | | | | |
|---|---|---|---|---|---|---|---|
| | uf20 | uf50 | uf75 | gcp | plan_a | plan_m | pigeon |
| **sddrs-static-gc-rl** | 288 | ✗ | ✗ | 69732 | 94 | ✗ | 0 |
| rsdd-comp-linear | 104 | 156 | ✗ | 5674 | 94 | 384 | 0 |
| sdd-package-rl | 182 | 187 | 1744 | 2196 | 136 | 582 | 0 |

Table 2: Sizes of compiled SDDs (sums of sizes of X-partitions).

# Conclusion

- Developed a bottom-up compiler for Sentential Decision Diagrams, featuring dynamic minimization and garbage collection
- Designed an extensible API to support user-defined heuristics for both dynamic minimization and garbage collection
- Demonstrated the compiler's competitive performance in specific configurations on smaller datasets

# Bibliography I

[1] Arthur Choi and Adnan Darwiche. “Dynamic Minimization of Sentential Decision Diagrams”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 27.1 (June 2013), pp. 187–194.

[2] Adnan Darwiche. “SDD: A new canonical representation of propositional knowledge bases”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[3] Adnan Darwiche and Pierre Marquis. “A knowledge compilation map”. In: *Journal of Artificial Intelligence Research* 17 (2002), pp. 229–264.

[4] Holger H Hoos and Thomas Stützle. “SATLIB: An online resource for research on SAT”. In: *Sat* 2000 (2000), pp. 283–292.

# Bibliography II

[5] Umut Oztok and Adnan Darwiche. “A top-down compiler for sentential decision diagrams”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[6] Knot Pipatsrisawat and Adnan Darwiche. “New Compilation Languages Based on Structured Decomposability.”. In: *AAAI*. Vol. 8. 2008, pp. 517–522.

[7] Yexiang Xue, Arthur Choi, and Adnan Darwiche. “Basing decisions on sentences in decision diagrams”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 1. 2012, pp. 842–849.

# Library Usage

```
let options = options::SddOptions::builder()
    .vtree_strategy(options::VTreeStrategy::RightLinear)
    .garbage_collection(options::GarbageCollection::Automatic)
    .variables(["A".to_string(), "B".to_string(), "C".to_string()])
    .build();
let manager = SddManager::new(&options);

let a = manager.literal("A", Polarity::Positive).unwrap();
let b = manager.literal("B", Polarity::Positive).unwrap();
let n_c = manager.literal("C", Polarity::Negative).unwrap();

let a_and_b_or_nc = manager.disjoin(&manager.conjoin(&a, &b), &n_c);

let model_count = manager.model_count(&a_and_b_or_nc);
let models = manager.model_enumeration(&a_and_b_or_nc);
```

# Binary Usage

```
sddrsc \
    --dimacs-path ./datasets/easy.cnf \
    --vtree right-linear \
    --minimize-after-k-clauses 2 \
    --collect-garbage  \
    --count-models \
    --enumerate-models \
    --print-statistics \
    --sdd.dot

4
  1  2  3  4
 ------------
  0  1  1  1
  1  0  0  1
  1  1  0  1
  1  1  1  1
compilation time: 148.00us
model count time: 20.00us
sdd size        : 10
all sdds        : 22
```