Mechanically-Verified Validation of Satisfiability Solvers

Nathan Wetzler

The University of Texas at Austin

Dissertation Proposal
October 18, 2013

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Motivation

Satisfiability solvers are used in amazing ways...

- Hardware verification: Centaur x86 verification
- Combinatorial problems:
  - van der Waerden numbers
    [Dransfield, Marek, and Truszczynski, 2004]
  - Gardens of Eden in Conway's Game of Life
    [Hartman, Heule, Kwekkeboom, and Noels, 2013; Kouril and Paul, 2008]
- Unsatisfiability is often more important

Motivation

Satisfiability solvers are used in amazing ways...

- Hardware verification: Centaur x86 verification
- Combinatorial problems:
  - van der Waerden numbers
    [Dransfield, Marek, and Truszczynski, 2004]
  - Gardens of Eden in Conway's Game of Life
    [Hartman, Heule, Kwekkeboom, and Noels, 2013; Kouril and Paul, 2008]
- Unsatisfiability is often more important

..., but satisfiability solvers have errors.

- Documented bugs in SAT, SMT, and QBF solvers
  [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Competition winners have contradictory results
  (HWMCC winners from 2011 and 2012)
- Implementation errors often imply conceptual errors

Solutions

Verify SAT solvers

- Requires verification of all crucial search techniques
- Delicate balance between efficiency and ease of verification
- Verification proofs are specific to each solver
- New developments in solving require additional proof effort

Solutions

~~Verify SAT solvers~~

- ~~Requires verification of all crucial search techniques~~
- ~~Delicate balance between efficiency and ease of verification~~
- ~~Verification proofs are specific to each solver~~
- ~~New developments in solving require additional proof effort~~

Solutions

~~Verify SAT solvers~~

- ~~Requires verification of all crucial search techniques~~
- ~~Delicate balance between efficiency and ease of verification~~
- ~~Verification proofs are specific to each solver~~
- ~~New developments in solving require additional proof effort~~

Validate SAT solver output

- Emit “proof” of unsatisfiability from SAT solver
- A single proof checker can validate results from many state-of-the-art solvers
- Proof checker uses limited number of techniques and can be mechanically verified

Proposal

For my dissertation, I will develop a fast mechanically-verified satisfiability proof checker using ACL2.

Proposal

For my dissertation, I will develop a fast mechanically-verified satisfiability proof checker using ACL2.

This project has three tasks:

1. Design a suitable proof format,
2. Implement an efficient proof checker for the format, and
3. Demonstrate a proof of correctness for the proof checker.

Proof Properties

Easy to Emit

Proof Properties

Easy to Emit

Compact

Proof Properties

Easy to Emit

Compact

Checked Efficiently

Proof Properties

Easy to Emit

Compact

Checked Efficiently

Verified Checker

Proof Properties

Easy to Emit

Compact

Checked Efficiently

Verified Checker

Expressive

Proof Properties

🟦 Easy to Emit

🟩 Compact

🟥 Checked Efficiently

🟪 Verified Checker

🟨 Expressive

🟥 Resolution Proofs

[Zhang and Malik, 2003]
[Van Gelder, 2008]
[Biere, 2008]

🟥🟪 with Verified Checker

[Weber, 2006, and Amjad, 2009] (Isabelle/HOL)
[Darbari et al., 2010] (Coq)
[Armand et al., 2011] (Coq)

🟦🟩 Clausal (RUP) Proofs

[Goldberg and Novikov, 2003]
[Van Gelder, 2008]

Proof Properties

- 🟦 Easy to Emit
- 🟩 Compact
- 🟥 Checked Efficiently
- 🟪 Verified Checker
- 🟨 Expressive

🟥 Resolution Proofs
[Zhang and Malik, 2003]
[Van Gelder, 2008]
[Biere, 2008]

🟥🟪 with Verified Checker
[Weber, 2006, and Amjad, 2009] (Isabelle/HOL)
[Darbari et al., 2010] (Coq)
[Armand et al., 2011] (Coq)

🟦🟩 Clausal (RUP) Proofs
[Goldberg and Novikov, 2003]
[Van Gelder, 2008]

🟦🟩🟥 DRUP (DRUP-Trim)
[Heule, Hunt, Jr., and **Wetzler**, STVR 201X]
[Heule, Hunt, Jr., and **Wetzler**, FMCAD 2013]

🟦🟩🟨 RAT Proofs
[Heule, Hunt, Jr., and **Wetzler**, CADE 2013]

🟦🟩🟪🟨 with Verified Checker
[**Wetzler**, Heule, and Hunt, Jr., ITP 2013] (ACL2)

Proof Properties

🟦 Easy to Emit

🟩 Compact

🟥 Checked Efficiently

🟪 Verified Checker

🟨 Expressive

🟥 Resolution Proofs
[Zhang and Malik, 2003]
[Van Gelder, 2008]
[Biere, 2008]

🟥🟪 with Verified Checker
[Weber, 2006, and Amjad, 2009] (Isabelle/HOL)
[Darbari et al., 2010] (Coq)
[Armand et al., 2011] (Coq)

🟦🟩 Clausal (RUP) Proofs
[Goldberg and Novikov, 2003]
[Van Gelder, 2008]

🟦🟩🟥 DRUP (DRUP-Trim)
[Heule, Hunt, Jr., and **Wetzler**, STVR 201X]
[Heule, Hunt, Jr., and **Wetzler**, FMCAD 2013]

🟦🟩🟨 RAT Proofs
[Heule, Hunt, Jr., and **Wetzler**, CADE 2013]

🟦🟩🟪🟨 with Verified Checker
[**Wetzler**, Heule, and Hunt, Jr., ITP 2013] (ACL2)

🟦🟩🟧🟪🟨 Proposed Work

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Satisfiability

Is there an assignment of values to variables such that a given Boolean formula evaluates to TRUE?

🟦 $(x_1 \vee x_2 \vee \neg x_3) \wedge$

🟦 $(\neg x_1 \vee \neg x_2 \vee x_3) \wedge$

🟦 $(x_2 \vee x_3 \vee \neg x_4) \wedge$

🟦 $(\neg x_2 \vee \neg x_3 \vee x_4) \wedge$

🟦 $(x_1 \vee x_3 \vee x_4) \wedge$

🟦 $(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$

🟦 $(x_1 \vee \neg x_2 \vee \neg x_4) \wedge$

🟦 $(\neg x_1 \vee x_2 \vee x_4)$

Satisfiability

Is there an assignment of values to variables such that a given Boolean formula evaluates to TRUE?

Checking a solution is easy.

Determining unsatisfiability is more difficult.

🟦 $(x_1 \lor x_2 \lor \neg x_3) \land$

🟦 $(\neg x_1 \lor \neg x_2 \lor x_3) \land$

🟦 $(x_2 \lor x_3 \lor \neg x_4) \land$

🟦 $(\neg x_2 \lor \neg x_3 \lor x_4) \land$

🟦 $(x_1 \lor x_3 \lor x_4) \land$

🟦 $(\neg x_1 \lor \neg x_3 \lor \neg x_4) \land$

🟦 $(x_1 \lor \neg x_2 \lor \neg x_4) \land$

🟦 $(\neg x_1 \lor x_2 \lor x_4)$

Satisfiability

Is there an assignment of values to variables such that a given Boolean formula evaluates to TRUE?

Checking a solution is easy.

Determining unsatisfiability is more difficult.

Formulas are in conjunctive-normal form (CNF).

🟦 $(x_1 \lor x_2 \lor \neg x_3) \land$

🟦 $(\neg x_1 \lor \neg x_2 \lor x_3) \land$

🟦 $(x_2 \lor x_3 \lor \neg x_4) \land$

🟦 $(\neg x_2 \lor \neg x_3 \lor x_4) \land$

🟦 $(x_1 \lor x_3 \lor x_4) \land$

🟦 $(\neg x_1 \lor \neg x_3 \lor \neg x_4) \land$

🟦 $(x_1 \lor \neg x_2 \lor \neg x_4) \land$

🟦 $(\neg x_1 \lor x_2 \lor x_4)$

Satisfiability

Is there an assignment of values to variables such that a given Boolean formula evaluates to TRUE?

Checking a solution is easy.

Determining unsatisfiability is more difficult.

Formulas are in conjunctive-normal form (CNF).

```
   CNF
🟦  1  2 -3 0
🟦 -1 -2  3 0
🟦  2  3 -4 0
🟦 -2 -3  4 0
🟦  1  3  4 0
🟦 -1 -3 -4 0
🟦  1 -2 -4 0
🟦 -1  3  4 0
```

Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

Proofs and Refutations

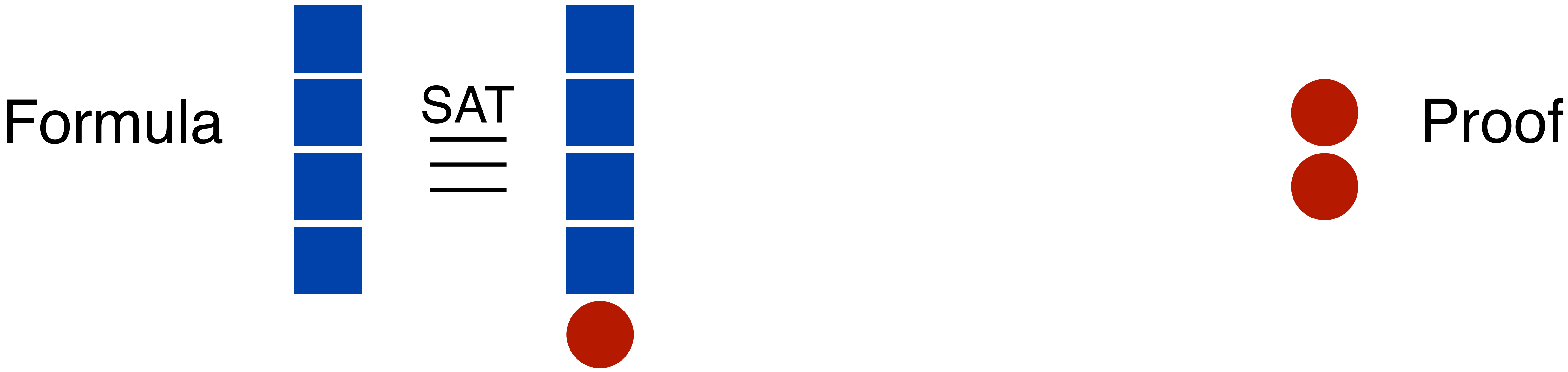
A clause `C` is **redundant** with respect to a formula `F` if `C` conjoined with `F` is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to `F`.

A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.

Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

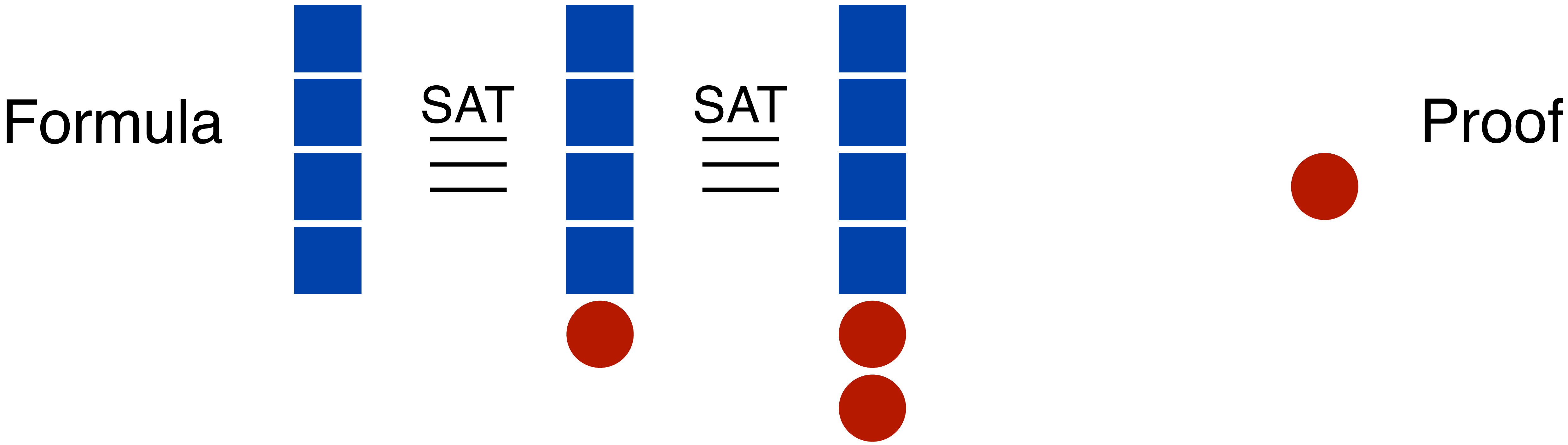
A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.



Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

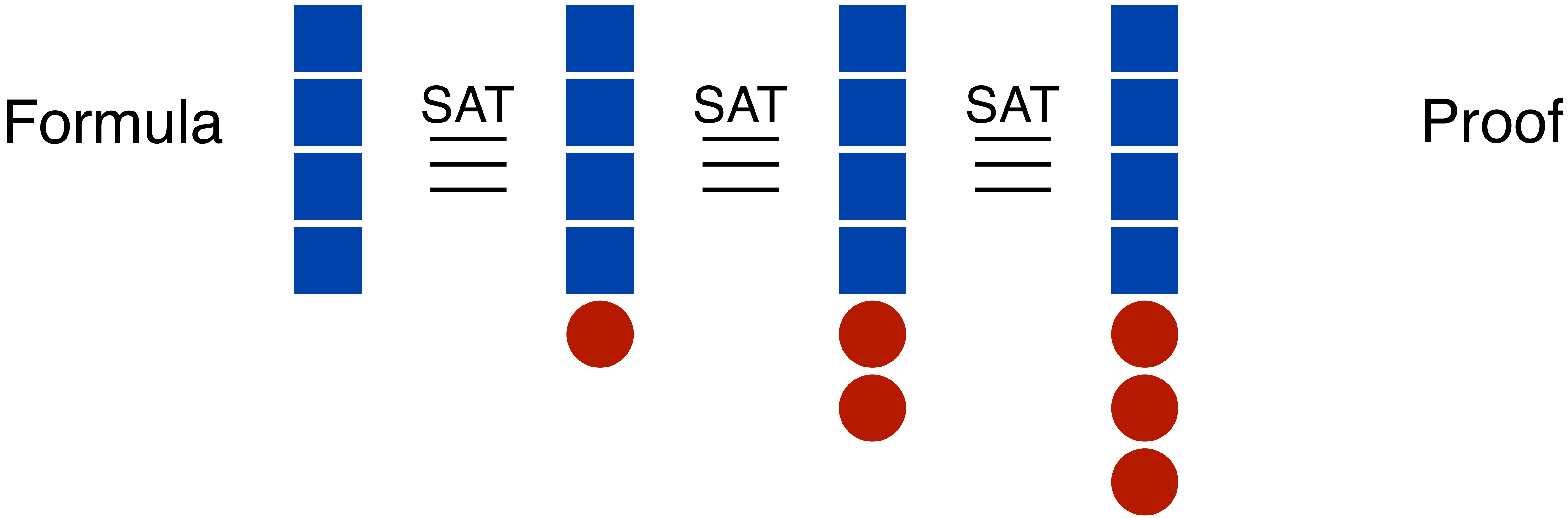
A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.



Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.

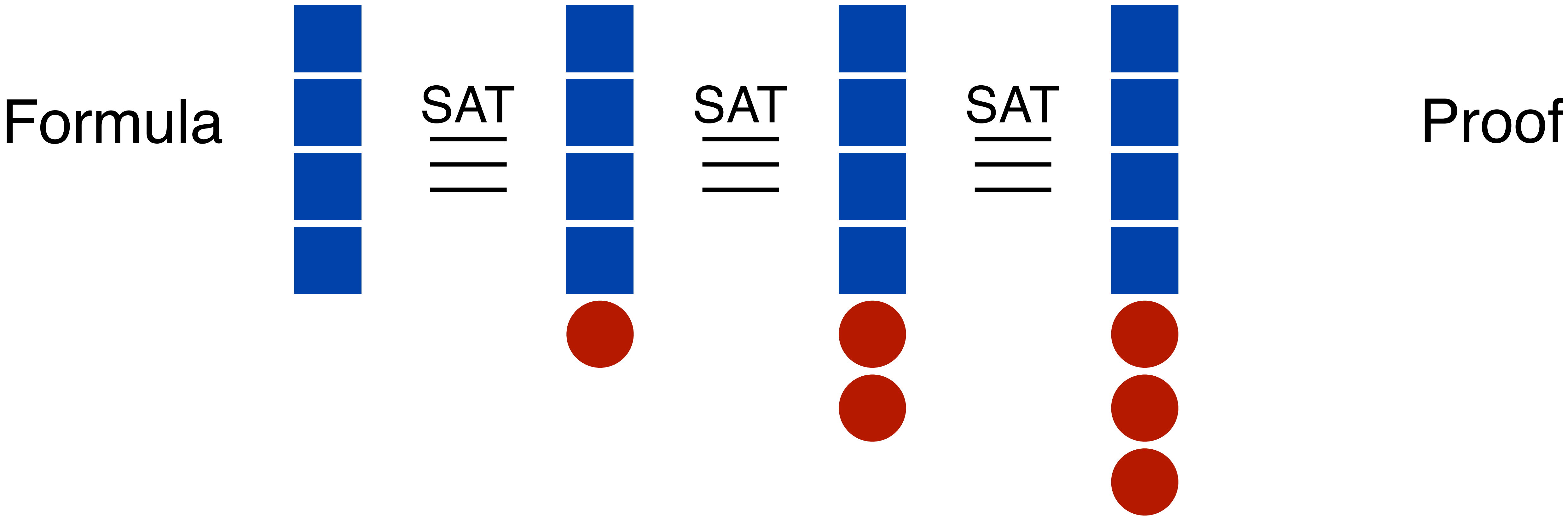


Formula $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ Proof

Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.



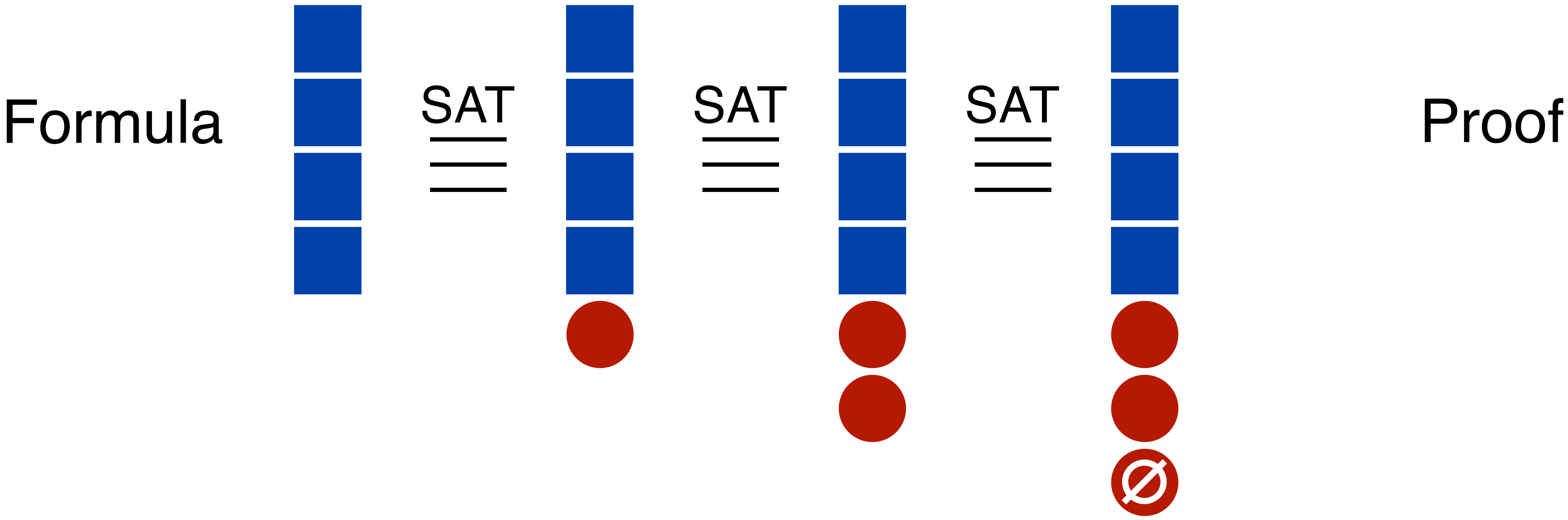
Formula $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ Proof

A **refutation** is a proof trace that contains the (unsatisfiable) empty clause, ∅.

Proofs and Refutations

A clause C is **redundant** with respect to a formula F if C conjoined with F is satisfiability equivalent, $\overset{\text{SAT}}{\equiv}$, to F.

A **proof trace** is a sequence of clauses that are redundant with respect to a evolving formula.



Formula $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ $\overset{\text{SAT}}{\equiv}$ Proof

A **refutation** is a proof trace that contains the (unsatisfiable) empty clause, ∅.

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Task 1: Designing a Proof Format

Existing proof formats are insufficient.

- Resolution proofs are large and hard to emit
- Clausal (RUP) proofs are inefficient, but are compact and easy to emit

Task 1: Designing a Proof Format

Existing proof formats are insufficient.

- Resolution proofs are large and hard to emit
- Clausal (RUP) proofs are inefficient, but are compact and easy to emit

Use clausal proofs as a foundation with two extensions:

- Add deletion information
- Extend equivalence from logical to satisfiability

Extension 1: Deletion Information

Proofs can be extended with clause deletion information.

- Solvers remove clauses during search
- Remove unnecessary clauses during validation
- Emit learning and deletion information
- New format called DRUP (**D**eletion **RUP**)

Extension 1: Deletion Information

Proofs can be extended with clause deletion information.

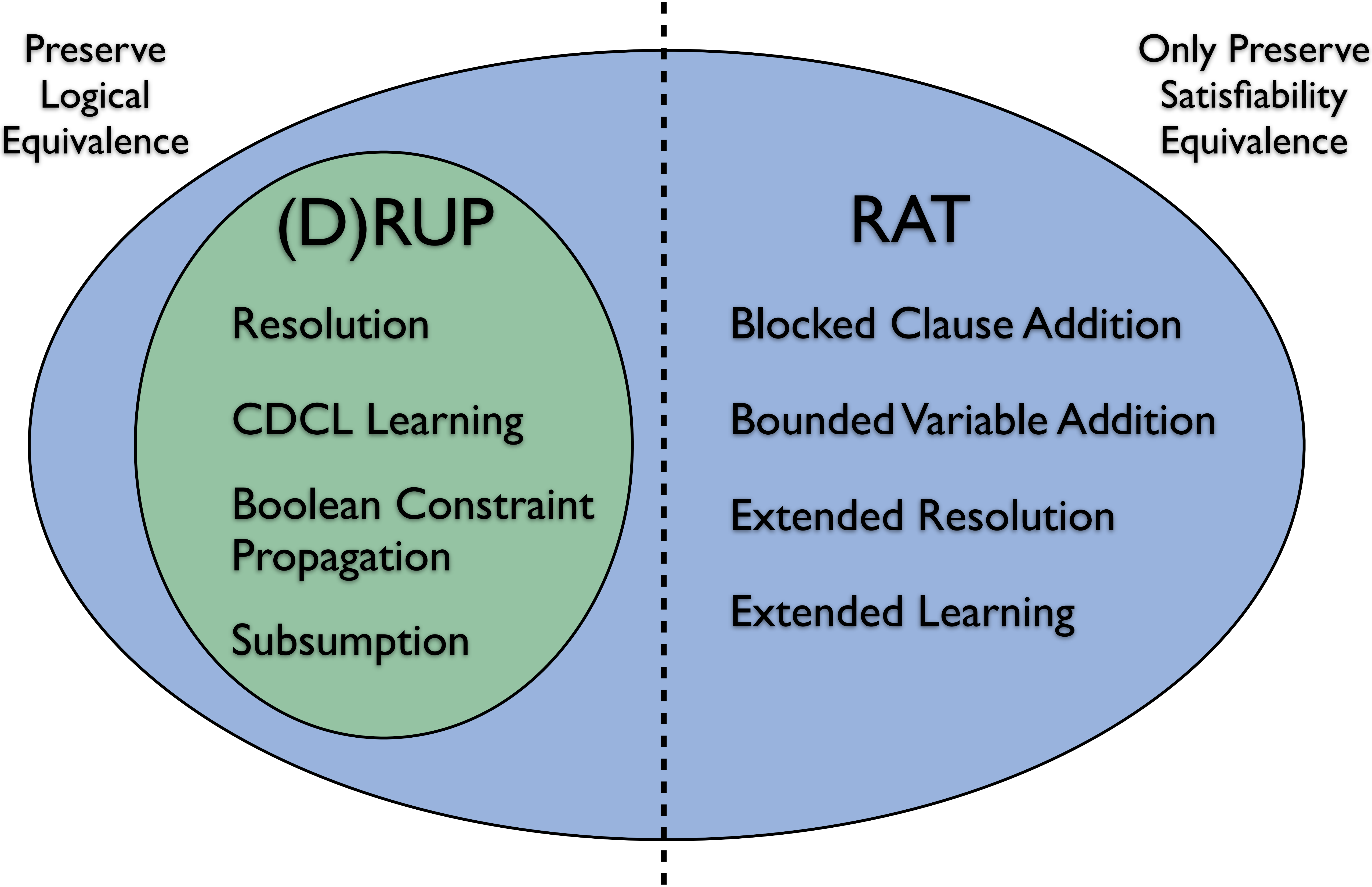
- Solvers remove clauses during search
- Remove unnecessary clauses during validation
- Emit learning and deletion information
- New format called DRUP (**D**eletion **RUP**)

```
CNF  🟦   1  2 -3 0
     🟦  -1 -2  3 0
     🟦   2  3 -4 0
     🟦  -2 -3  4 0
     🟦   1  3  4 0
     🟦  -1 -3 -4 0
     🟦   1 -2 -4 0
     🟦  -1  2  4 0
```

```
RUP  🔴  -1  2 0
     🔴  -1    0
     🔴   2    0
     ∅         0
```

```
DRUP  🔴    -1  2    0
      ⬛  d -1  2  4 0
      🔴    -1       0
      ⬛  d -1 -2  3 0
      ⬛  d -1 -3 -4 0
      ⚫  d -1  2    0
      🔴     2       0
      ⬛  d  1  2 -3 0
      ⬛  d  2  3 -4 0
      ∅              0
```

Extension 2: Expressiveness



Preserve Logical Equivalence

Only Preserve Satisfiability Equivalence

(D)RUP

- Resolution
- CDCL Learning
- Boolean Constraint Propagation
- Subsumption

RAT

- Blocked Clause Addition
- Bounded Variable Addition
- Extended Resolution
- Extended Learning

DRAT Format

The DRUP and RAT proof formats can be combined.

- How will the two formats interact?
- With what frequency are RAT clauses produced?
- Will the addition of RAT clauses lead to more deletions?

CNF

```
🟦   1  2 -3 0
🟦  -1 -2  3 0
🟦   2  3 -4 0
🟦  -2 -3  4 0
🟦   1  3  4 0
🟦  -1 -3 -4 0
🟦   1 -2 -4 0
🟦  -1  2  4 0
```

DRUP

```
🔴    -1  2    0
■   d -1  2  4 0
🔴    -1       0
■   d -1 -2  3 0
■   d -1 -3 -4 0
●   d -1  2    0
🔴        2    0
■   d  1  2 -3 0
■   d  2  3 -4 0
∅              0
```

DRAT

```
🔴    -1       0
■   d -1  2  4 0
■   d -1 -2  3 0
■   d -1 -3 -4 0
🔴     2       0
■   d  1  2 -3 0
■   d  2  3 -4 0
∅              0
```

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Task 2: Developing an Efficient Checker

Efficiency is necessary on industrial-scale problems.

- Validate proofs in a time similar to solving
- Performance without full range of solver techniques

Task 2: Developing an Efficient Checker

Efficiency is necessary on industrial-scale problems.

- Validate proofs in a time similar to solving
- Performance without full range of solver techniques

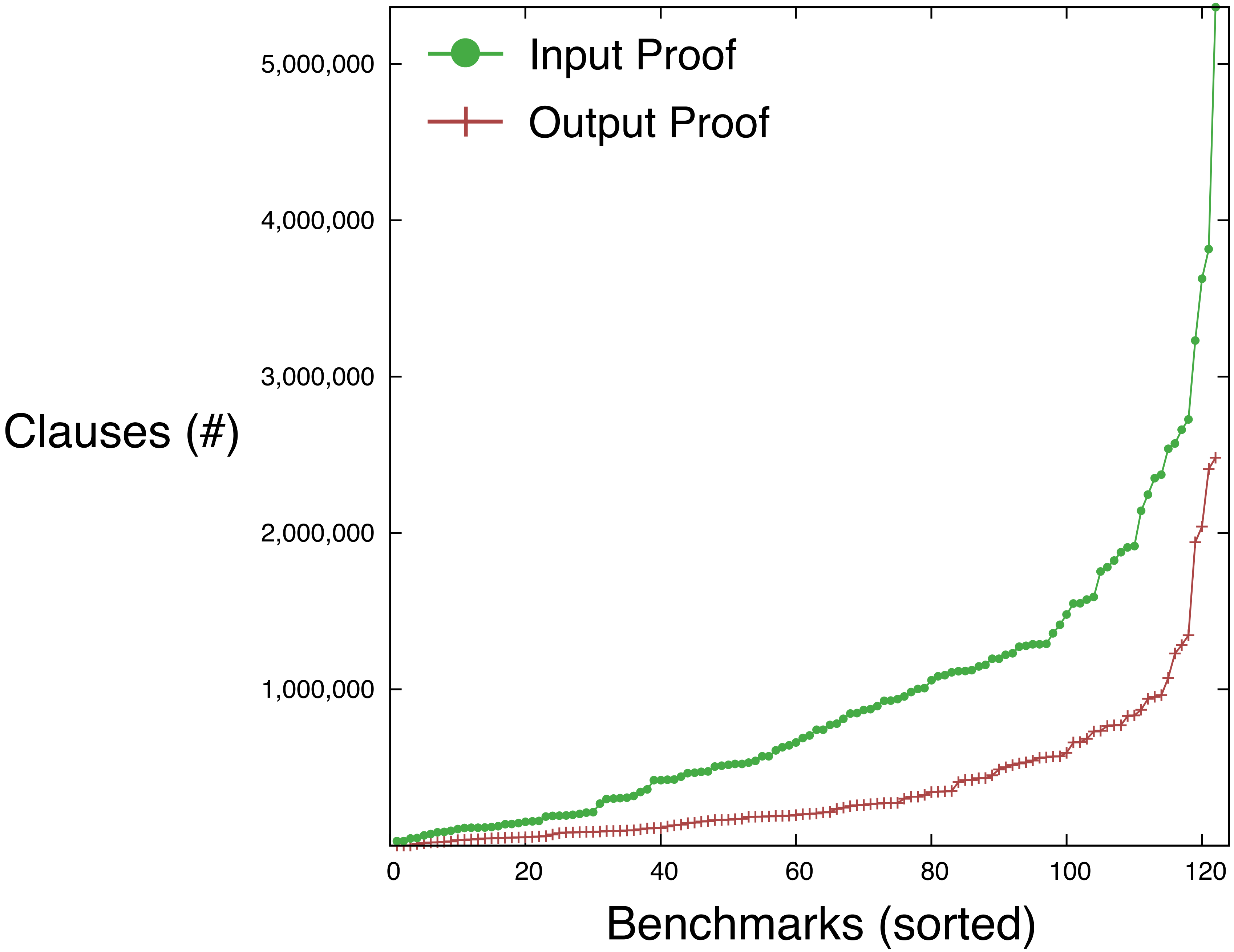
Several techniques to gain performance.

- Proofs can be trimmed before validated
- Efficient Boolean constraint propagation
- Constant-time, indexed memory access and update

Proof Trimming

Proofs often contain clauses that are unnecessary.

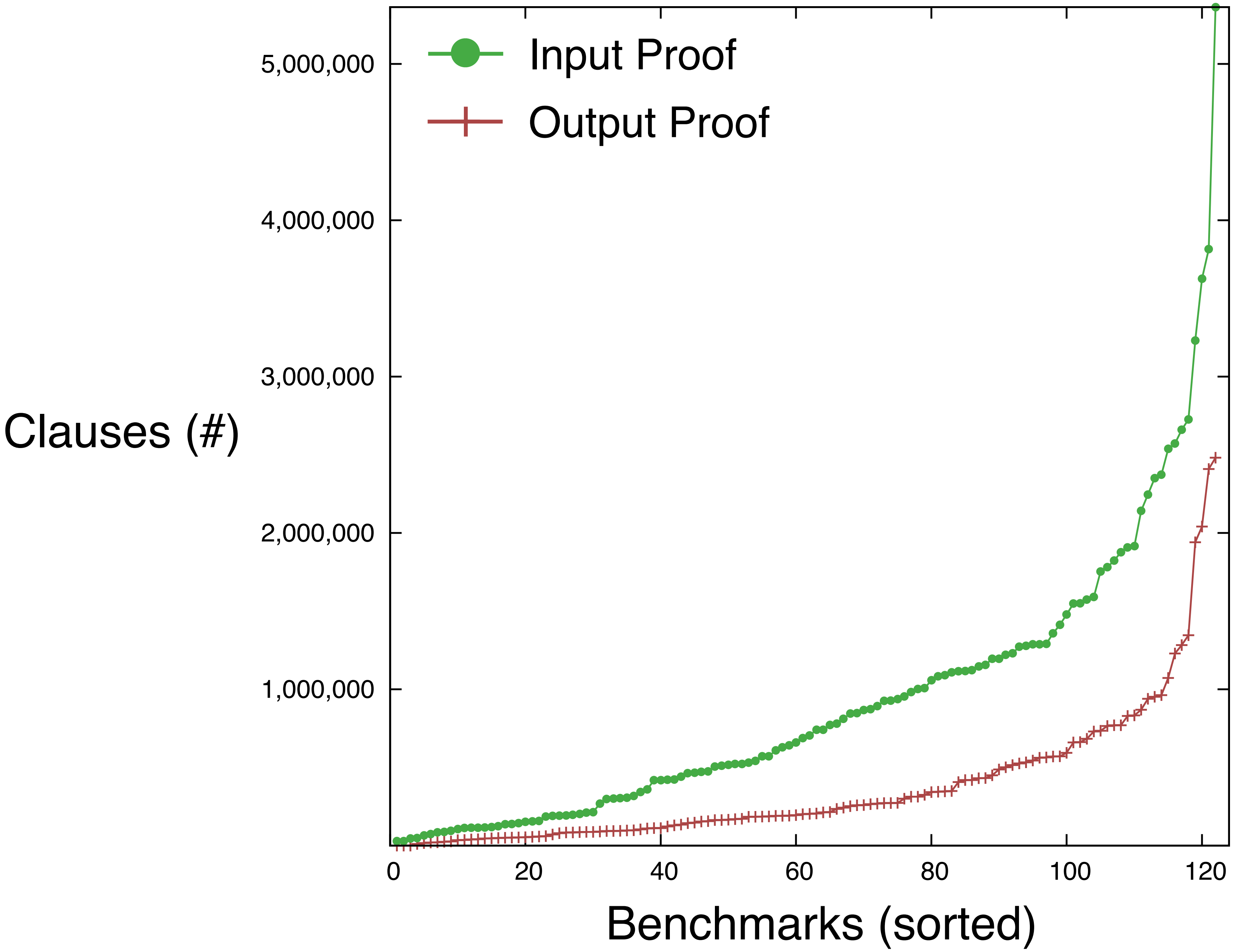
Our DRUP-trim tool trims (and checks) proofs.



Proof Trimming

Proofs often contain clauses that are unnecessary.

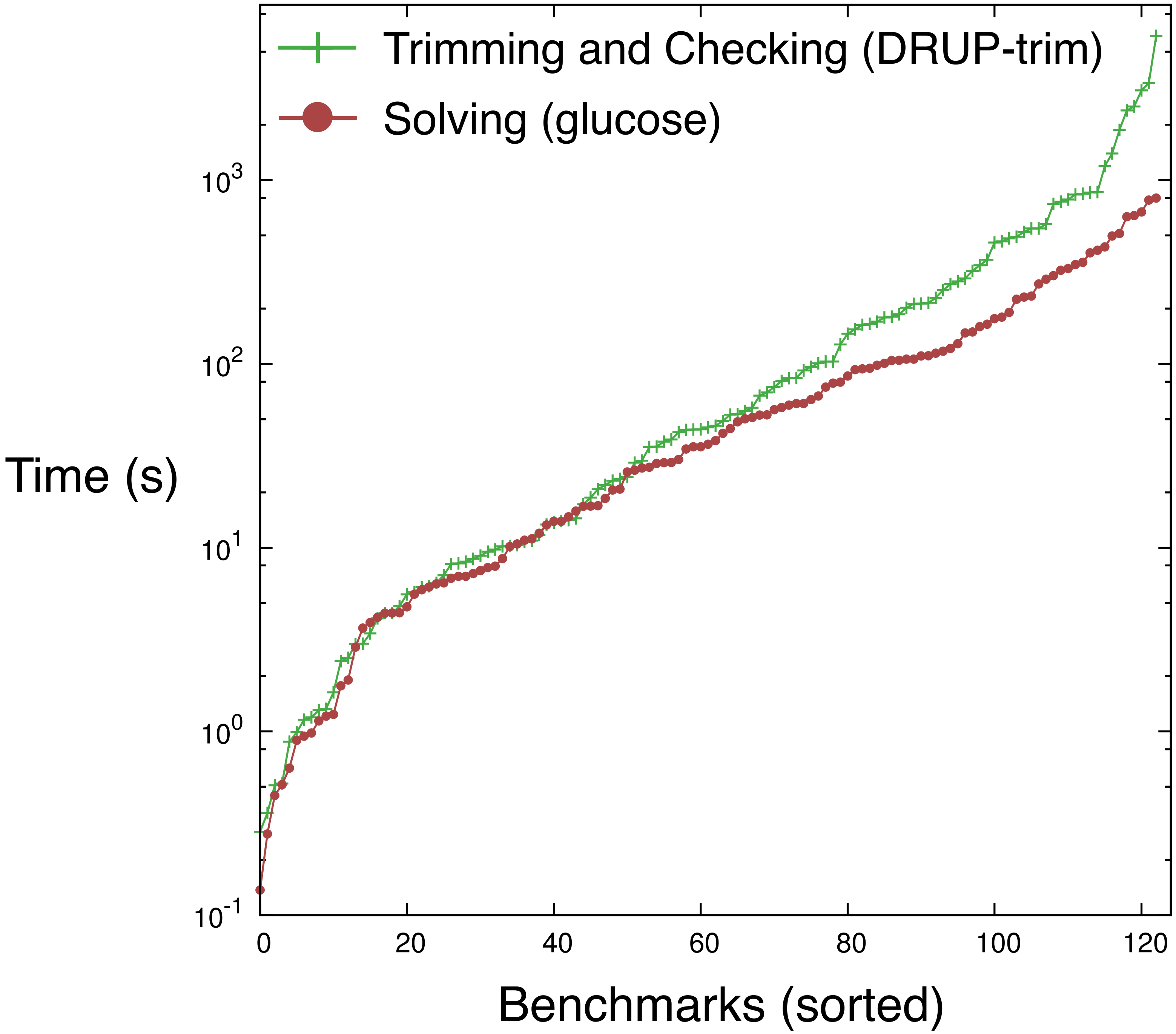
Our DRUP-trim tool trims (and checks) proofs.



Adds optimal deletion information.

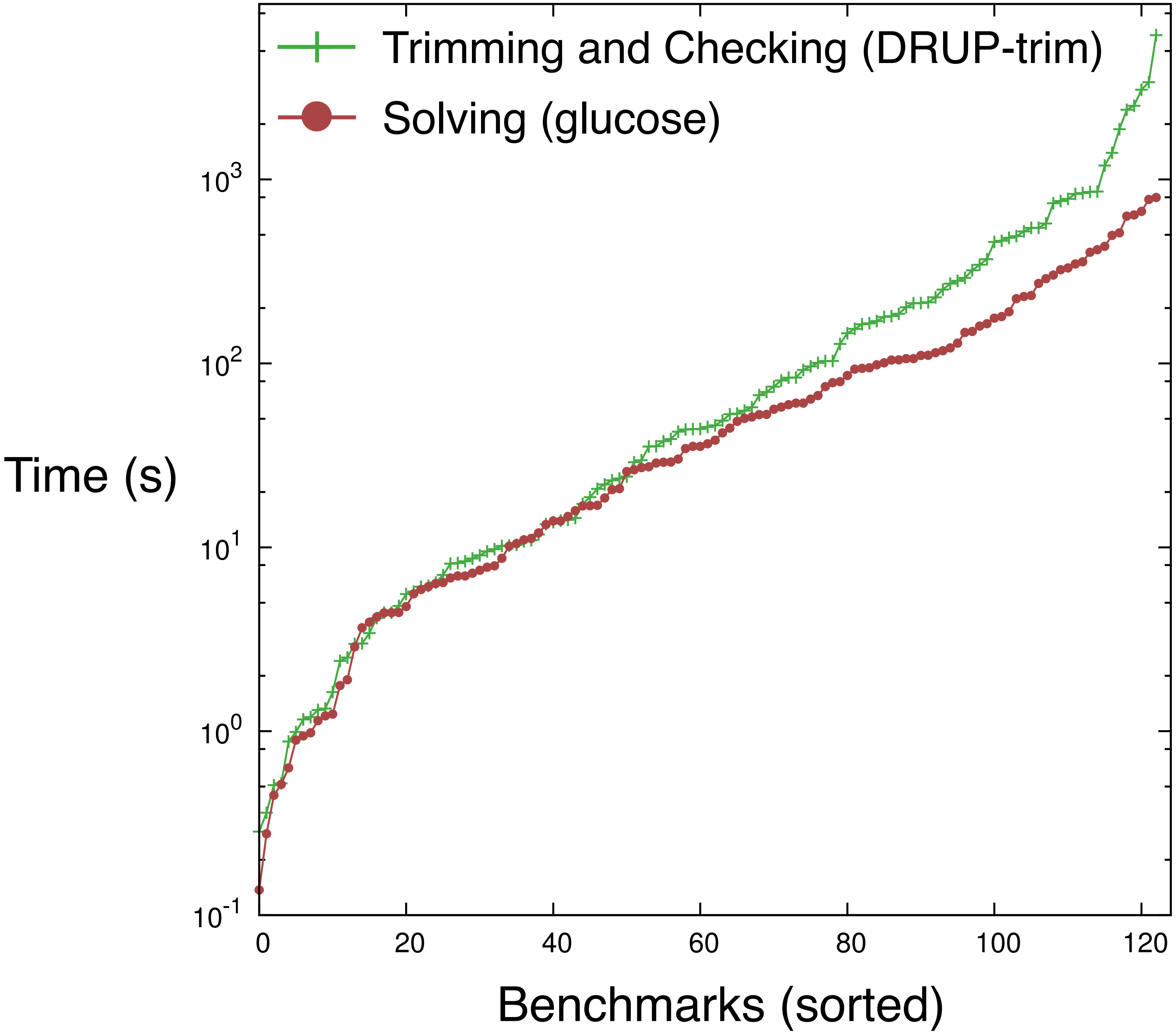
Fast Proof Checking

DRUP-trim is able to closely match solving time.



Fast Proof Checking

DRUP-trim is able to closely match solving time.



Used to check unsatisfiability results from SAT Competition 2013.

Fast Boolean Constraint Propagation

Clausal proof checkers spend around 95% of their time performing Boolean Constraint Propagation.

- Core technique in solvers
- Often implemented using a watched-literal data structure

Watched-Literal Invariant:

All clauses are satisfied or contain at least two unassigned literals.

This is just one of many implementation techniques that must be verified.

Efficiency of ACL2 Code

Typical ACL2 list-only data structures are not efficient.

- Access and update are linear time operations

Instead, one can:

- Mimic array-like structures using `STOBJs`
- Disassemble key functions to compare compiled code to a highly optimized version

We have implemented a basic RUP proof checker in ACL2 that achieves roughly 60% of a similar proof checker written in C.

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Task 3: Proving Correctness

Interactive theorem provers assist with verification.

- ACL2 combines a programming language, first-order logic, and theorem prover
- Proof checker is modeled in ACL2
- Specification for termination and soundness (but not completeness) are formalized
- Efficient execution by way of Common LISP compilers

Task 3: Proving Correctness

Interactive theorem provers assist with verification.

- ACL2 combines a programming language, first-order logic, and theorem prover
- Proof checker is modeled in ACL2
- Specification for termination and soundness (but not completeness) are formalized
- Efficient execution by way of Common LISP compilers

Incremental approach to proof process:

- Prove correctness of proof checkers for different formats
- Refine code to resemble C-equivalent

Verified Proof Checkers

Verified SAT solvers and proof checkers using ACL2.

- Verified RUP proof checker
- Verified IORUP (deletion information) proof checker
- Verified RAT proof checker

Verified Proof Checkers

Verified SAT solvers and proof checkers using ACL2.

- Verified RUP proof checker
- Verified IORUP (deletion information) proof checker
- Verified RAT proof checker

```
(defthm main-theorem
  (implies (and (formulap f)                  ; Given formula AND
                (refutationp r f))            ; refutation
           (not (exists-solution f))))        ; Then formula is unsatisfiable
```

Verified Proof Checkers

Verified SAT solvers and proof checkers using ACL2.

- Verified RUP proof checker
- Verified IORUP (deletion information) proof checker
- Verified RAT proof checker

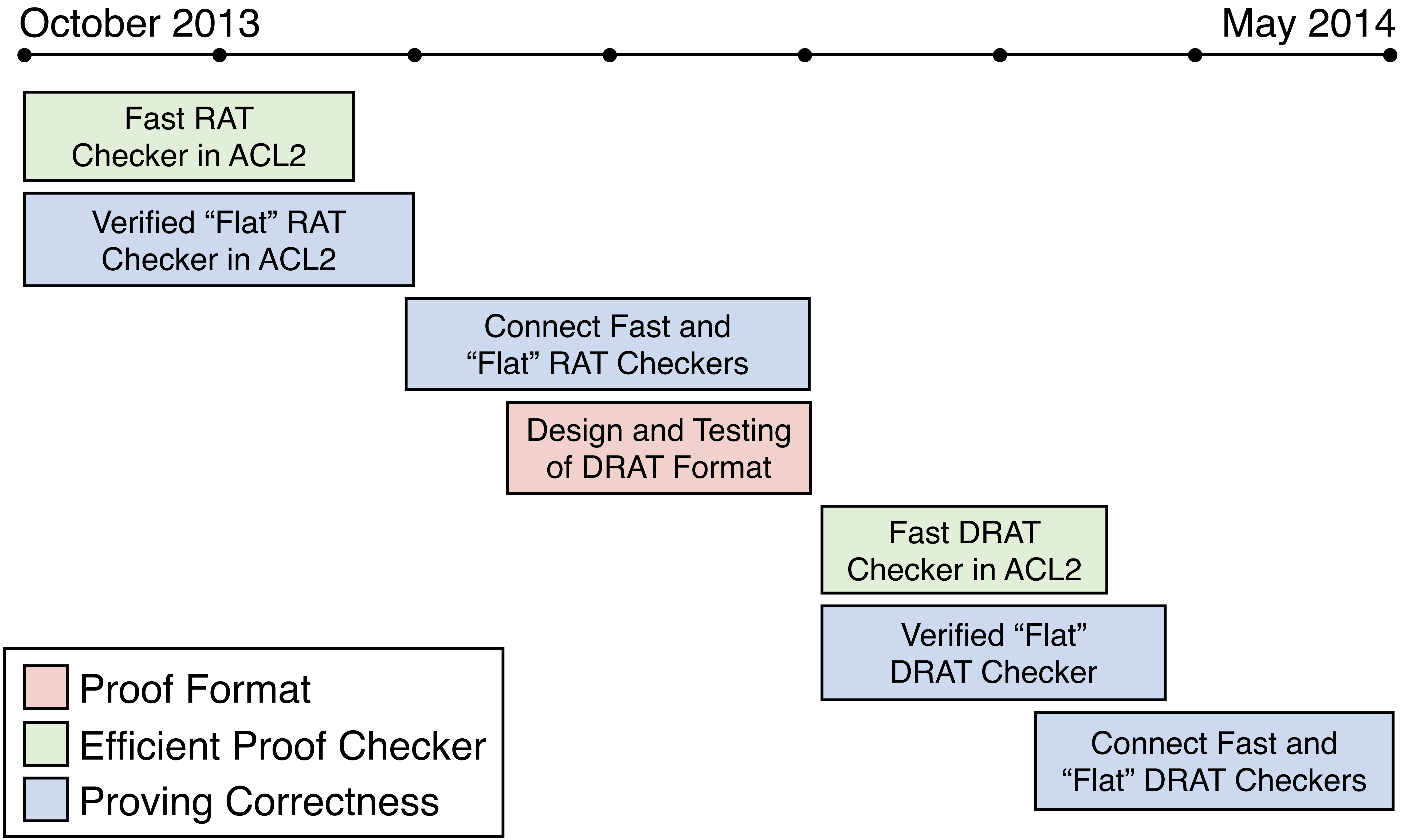
```
(defthm main-theorem
  (implies (and (formulap f)                    ; Given formula AND
                (refutationp r f))              ; refutation
           (not (exists-solution f))))          ; Then formula is unsatisfiable
```

Litany of transformations and refinements eventually resulting in code that corresponds to our C code.

Outline

- Motivation and Proposal
- Satisfiability and Proofs
- Task 1: Designing a Proof Format
- Task 2: Developing an Efficient Checker
- Task 3: Proving Correctness
- Timeline and Conclusion

Timeline



October 2013

May 2014

Fast RAT Checker in ACL2

Verified “Flat” RAT Checker in ACL2

Connect Fast and “Flat” RAT Checkers

Design and Testing of DRAT Format

Fast DRAT Checker in ACL2

Verified “Flat” DRAT Checker

Connect Fast and “Flat” DRAT Checkers

🟥 Proof Format

🟩 Efficient Proof Checker

🟦 Proving Correctness

Conclusion

This project has three components:

- Design a suitable proof format,
- Implement an efficient proof checker for the format, and
- Demonstrate a proof of correctness for the proof checker.

Easy to Emit

Compact

Checked Efficiently

Verified Checker

Expressive

Our goal is to increase confidence in **all** satisfiability solvers by efficiently checking proofs with a mechanically-verified proof checker.

Recent Work

Bridging the Gap Between Easy Generation and Efficient Verification of Unsatisfiability Proofs

Marijn J.H. Heule, Warren A. Hunt, Jr., and Nathan Wetzler

Accepted: Software Testing, Verification, and Reliability (STVR 201X)

Verifying Refutations with Extended Resolution

Marijn J.H. Heule, Warren A. Hunt, Jr., and Nathan Wetzler

Published: Conference on Automated Deduction (CADE 2013)

Mechanical Verification of SAT Refutations with Extended Resolution

Nathan Wetzler, Marijn J.H. Heule, and Warren A. Hunt, Jr.

Published: Interactive Theorem Proving (ITP 2013)

Trimming while Checking Clausal Proofs

Marijn J.H. Heule, Warren A. Hunt, Jr., and Nathan Wetzler

Published: Formal Methods in Computer-Aided Design (FMCAD 2013)

Thank you for your attention! Questions?

Redundancy

- Two formulas `F1` and `F2` are **logically equivalent** if they have the same set of satisfying assignments.

$$\texttt{F1} \equiv \texttt{F2}$$

Redundancy

- Two formulas F1 and F2 are **logically equivalent** if they have the same set of satisfying assignments.

$$F1 \equiv F2$$

- Two formulas F1 and F2 are **satisfiability equivalent** if they are both satisfiable or both unsatisfiable.

$$F1 \overset{\text{SAT}}{\equiv} F2$$

Redundancy

- Two formulas `F1` and `F2` are **logically equivalent** if they have the same set of satisfying assignments.

$$\mathtt{F1} \equiv \mathtt{F2}$$

- Two formulas `F1` and `F2` are **satisfiability equivalent** if they are both satisfiable or both unsatisfiable.

$$\mathtt{F1} \overset{\text{SAT}}{\equiv} \mathtt{F2}$$

- A clause `C` is **redundant** with respect to a formula `F` if `C` conjoined with `F` is satisfiability equivalent to `F`.

$$\mathtt{(and\ F\ C)} \overset{\text{SAT}}{\equiv} \mathtt{F}$$

Resolution Proof

1	1	2	-3	
2	-1	-2	3	
3	2	3	-4	
4	-2	-3	4	
5	1	3	4	
6	-1	-3	-4	
7	1	-2	-4	
8	-1	2	4	
9	-1	2		3 6 8
10	-1			4 6 2 9
11	2			3 5 1 10
∅				4 5 7 10 11

9

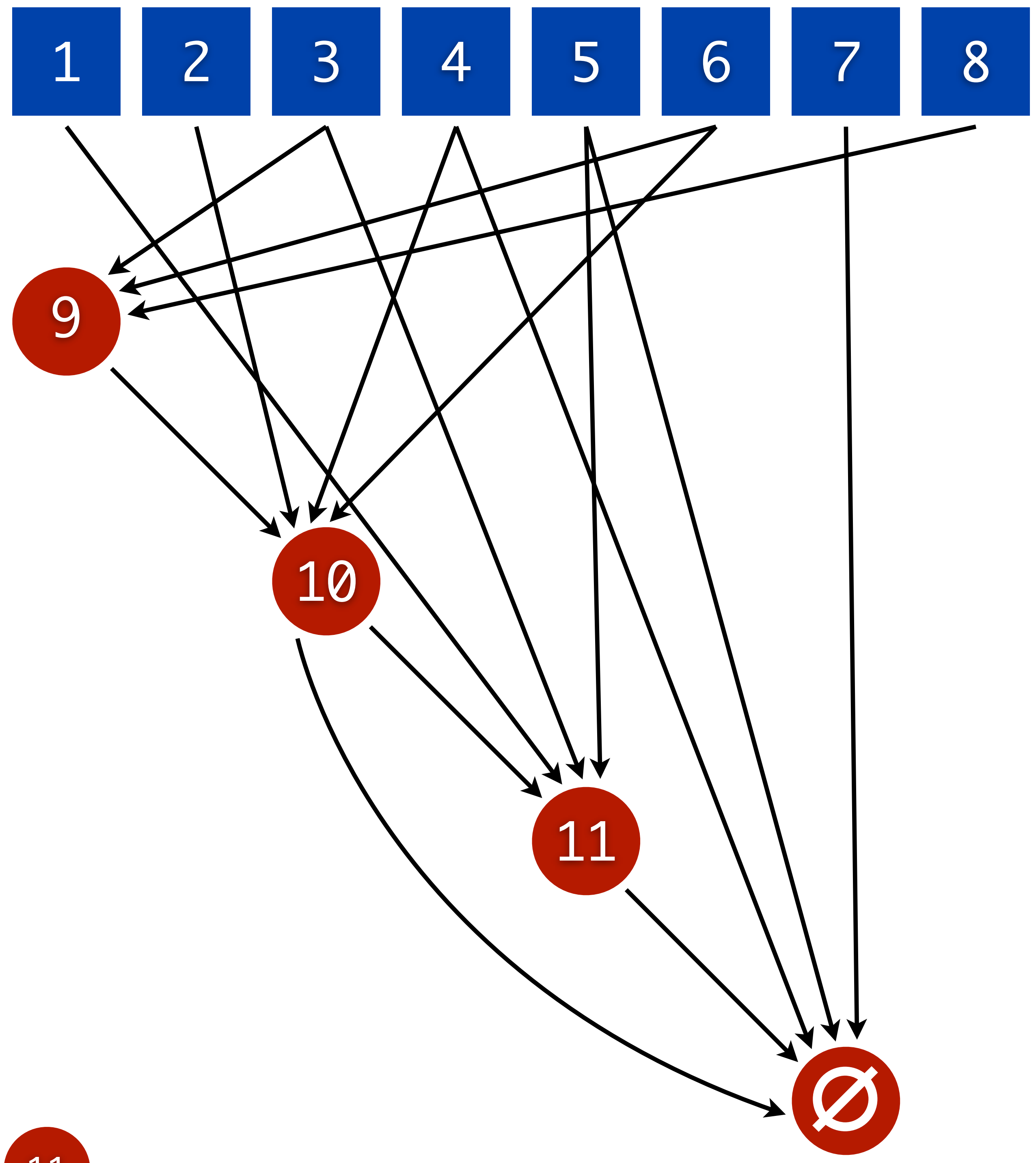
10

11

∅

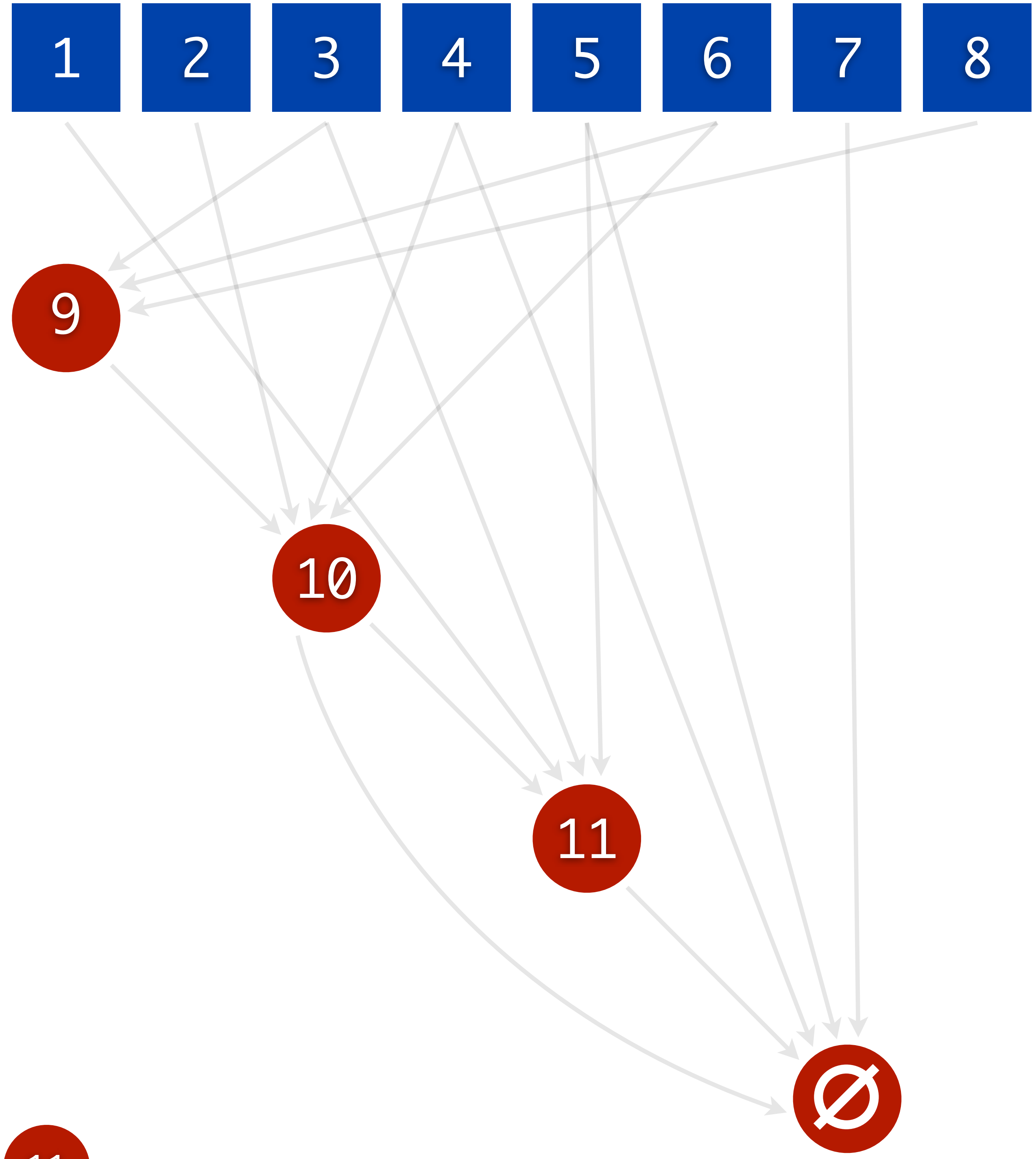
Resolution Proof

1	1	2	-3					
2	-1	-2	3					
3	2	3	-4					
4	-2	-3	4					
5	1	3	4					
6	-1	-3	-4					
7	1	-2	-4					
8	-1	2	4					
9	-1	2		3	6	8		
10	-1			4	6	2	9	
11	2			3	5	1	10	
∅				4	5	7	10	11



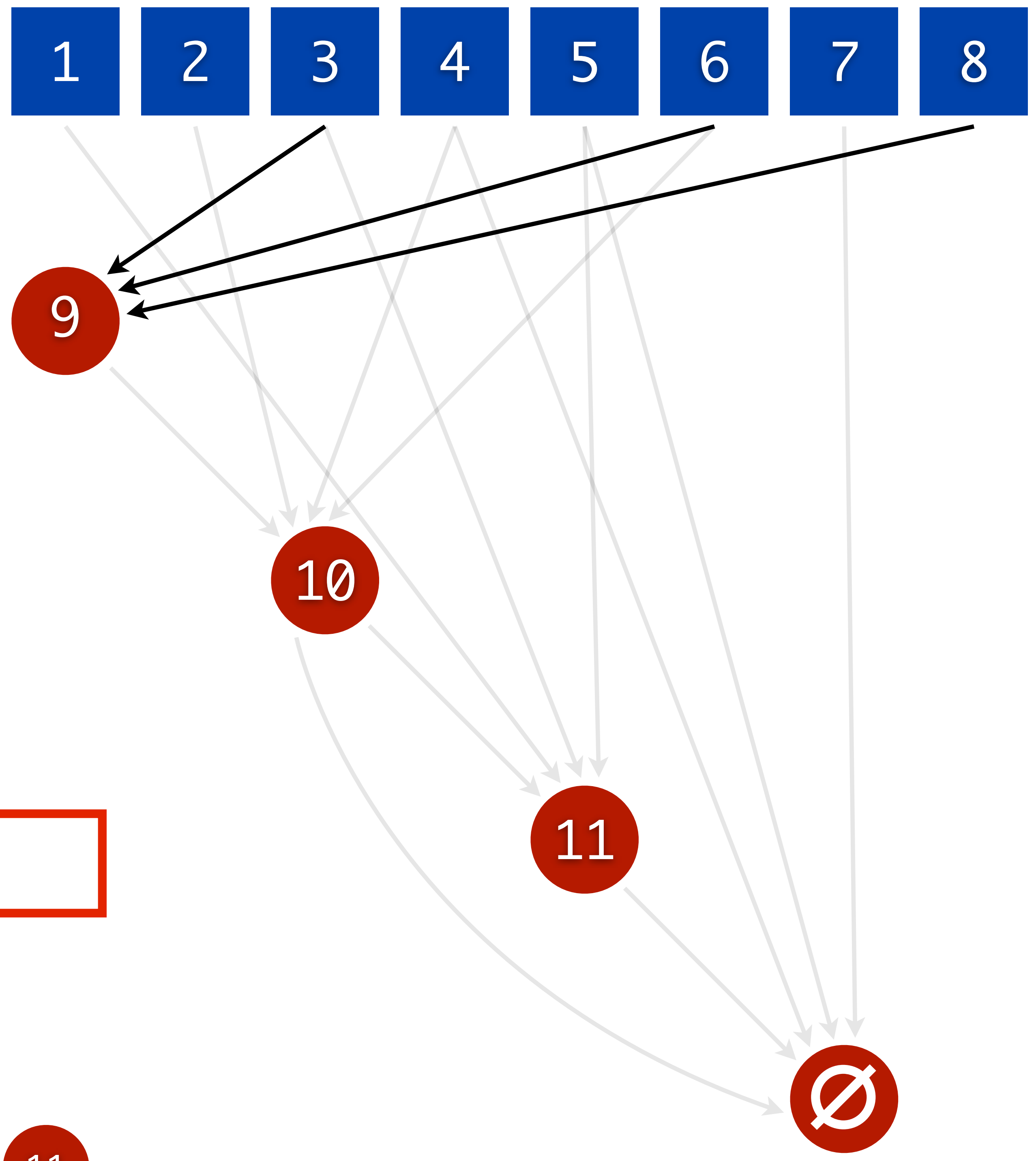
Resolution Proof

1	1 2 -3	
2	-1 -2 3	
3	2 3 -4	
4	-2 -3 4	
5	1 3 4	
6	-1 -3 -4	
7	1 -2 -4	
8	-1 2 4	
9	-1 2	3 6 8
10	-1	4 6 2 9
11	2	3 5 1 10
∅		4 5 7 10 11



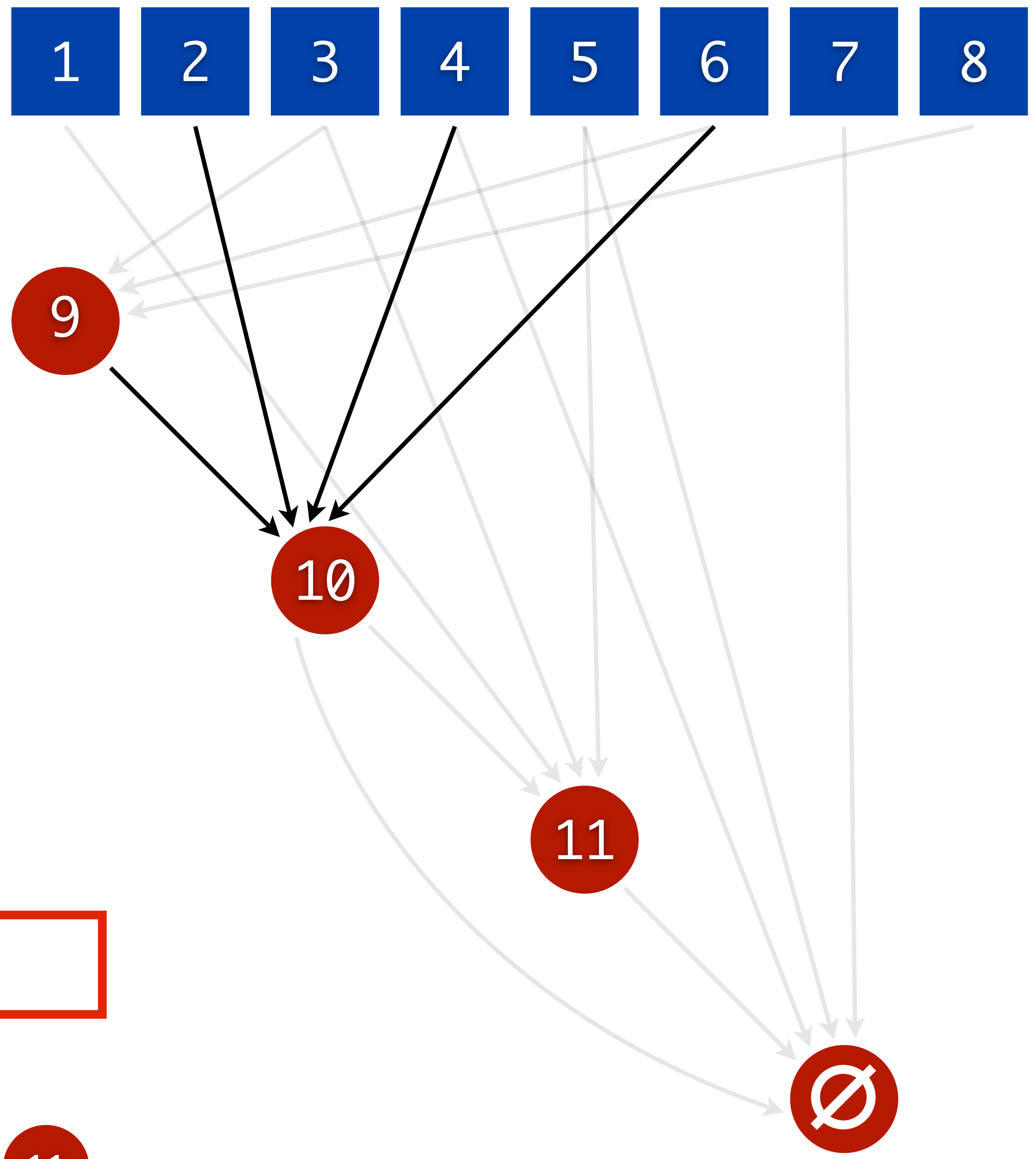
Resolution Proof

1	1	2	-3					
2	-1	-2	3					
3	2	3	-4					
4	-2	-3	4					
5	1	3	4					
6	-1	-3	-4					
7	1	-2	-4					
8	-1	2	4					
9	-1	2		3	6	8		
10	-1			4	6	2	9	
11	2			3	5	1	10	
∅				4	5	7	10	11



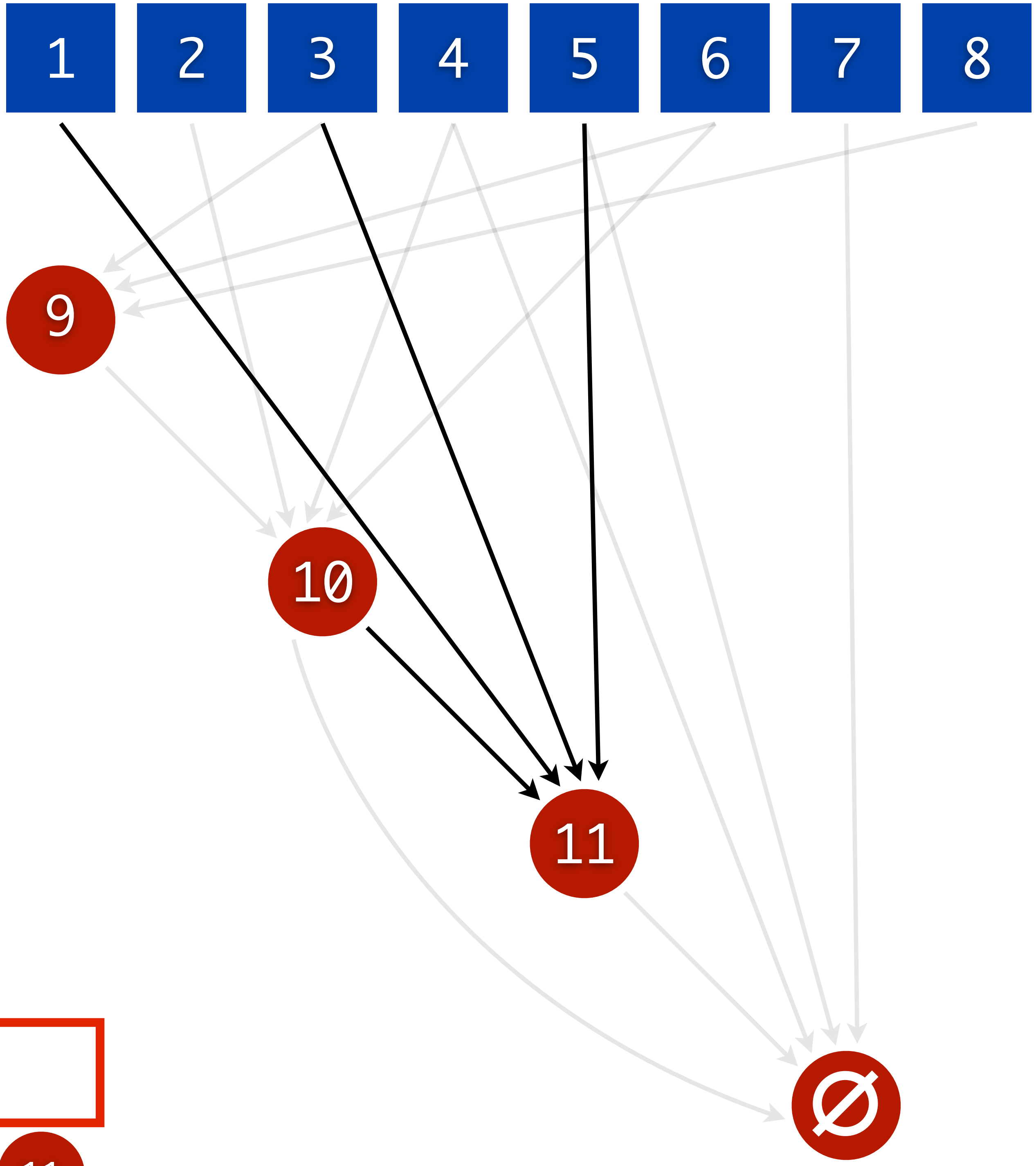
Resolution Proof

1	1 2 -3	
2	-1 -2 3	
3	2 3 -4	
4	-2 -3 4	
5	1 3 4	
6	-1 -3 -4	
7	1 -2 -4	
8	-1 2 4	
9	-1 2	3 6 8
10	-1	4 6 2 9
11	2	3 5 1 10
∅		4 5 7 10 11



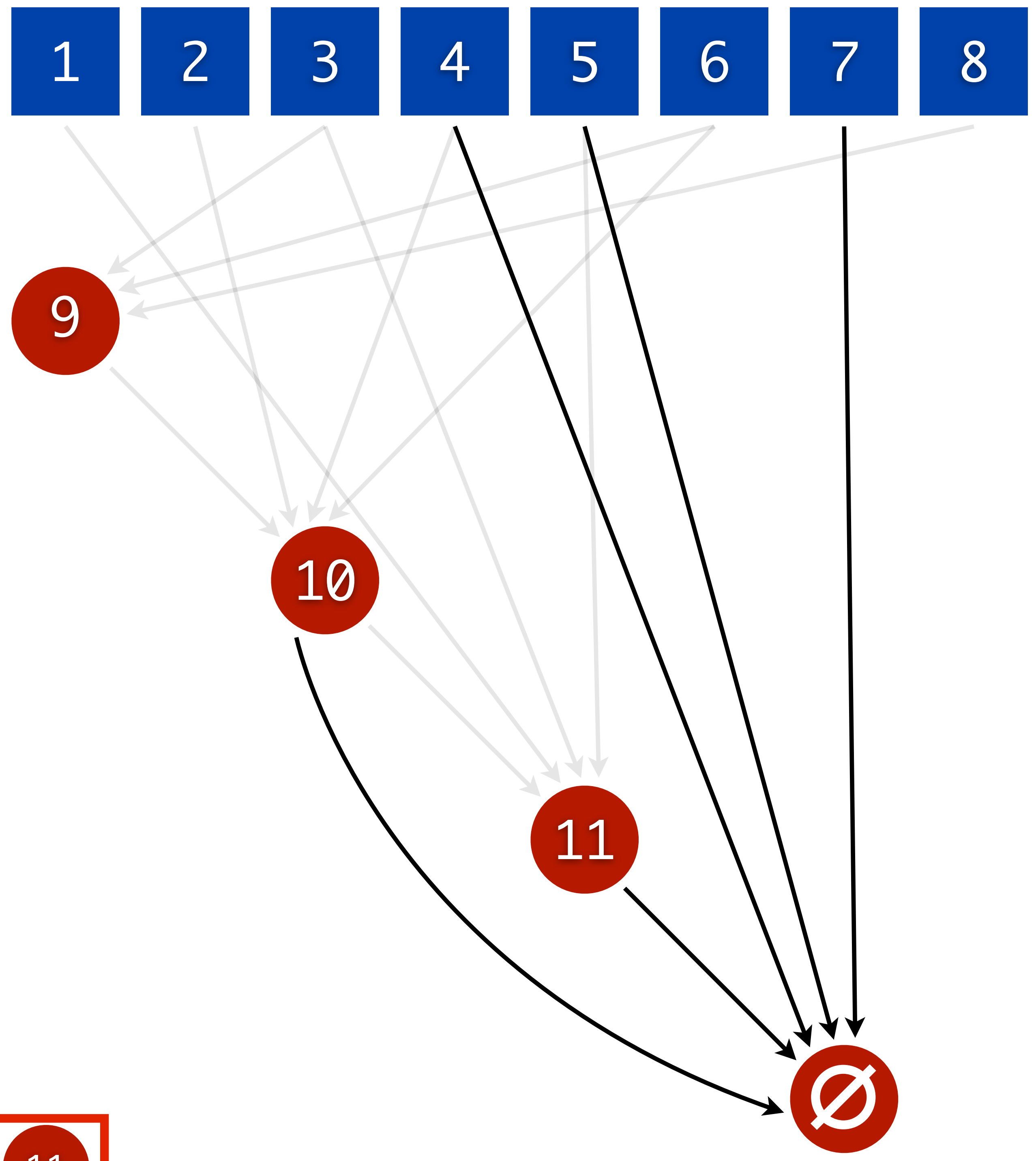
Resolution Proof

1	1	2	-3					
2	-1	-2	3					
3	2	3	-4					
4	-2	-3	4					
5	1	3	4					
6	-1	-3	-4					
7	1	-2	-4					
8	-1	2	4					
9	-1	2		3	6	8		
10	-1			4	6	2	9	
11		2		3	5	1	10	
∅				4	5	7	10	11



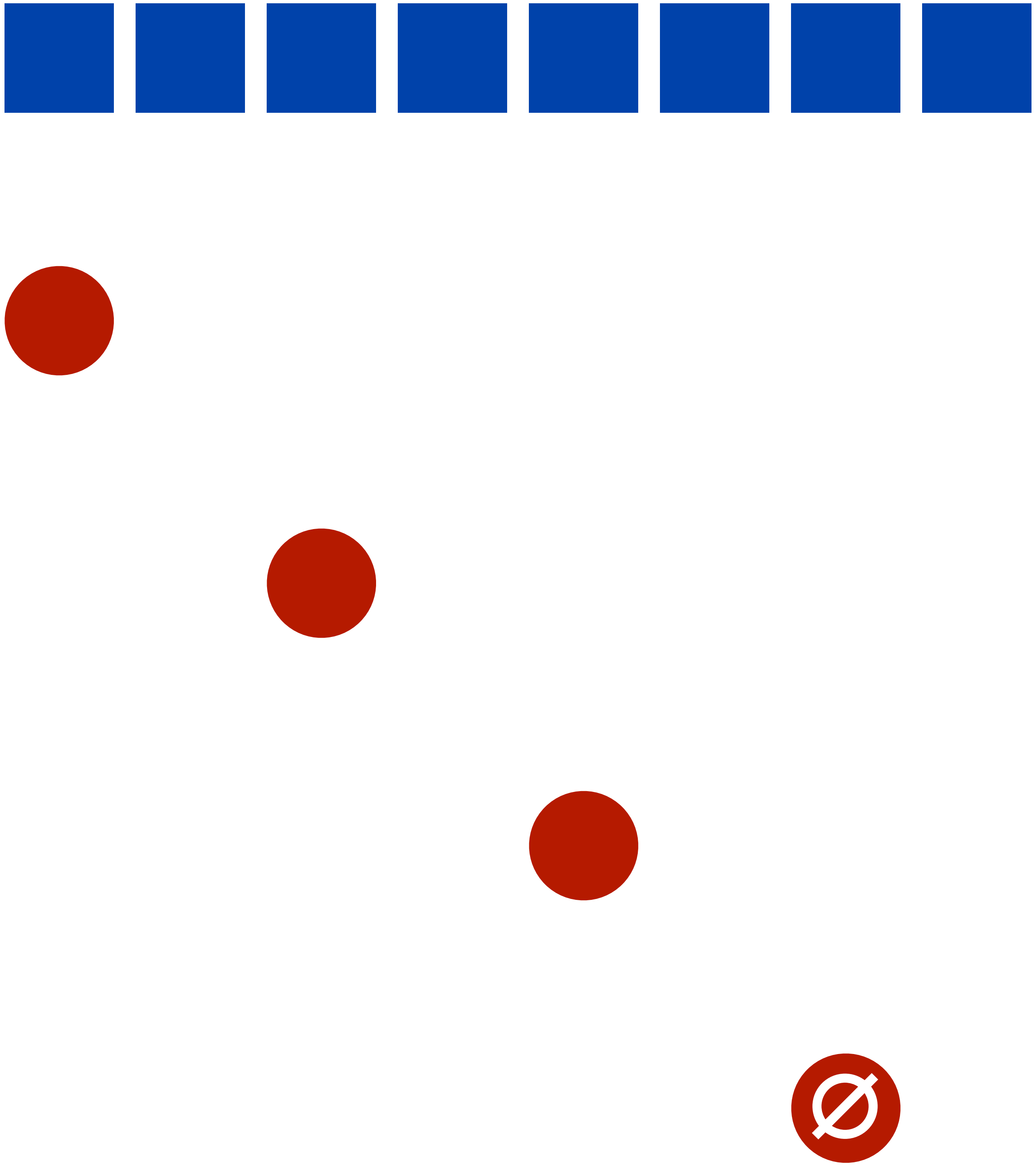
Resolution Proof

1	1	2	-3					
2	-1	-2	3					
3	2	3	-4					
4	-2	-3	4					
5	1	3	4					
6	-1	-3	-4					
7	1	-2	-4					
8	-1	2	4					
9	-1	2		3	6	8		
10	-1			4	6	2	9	
11	2			3	5	1	10	
∅				4	5	7	10	11



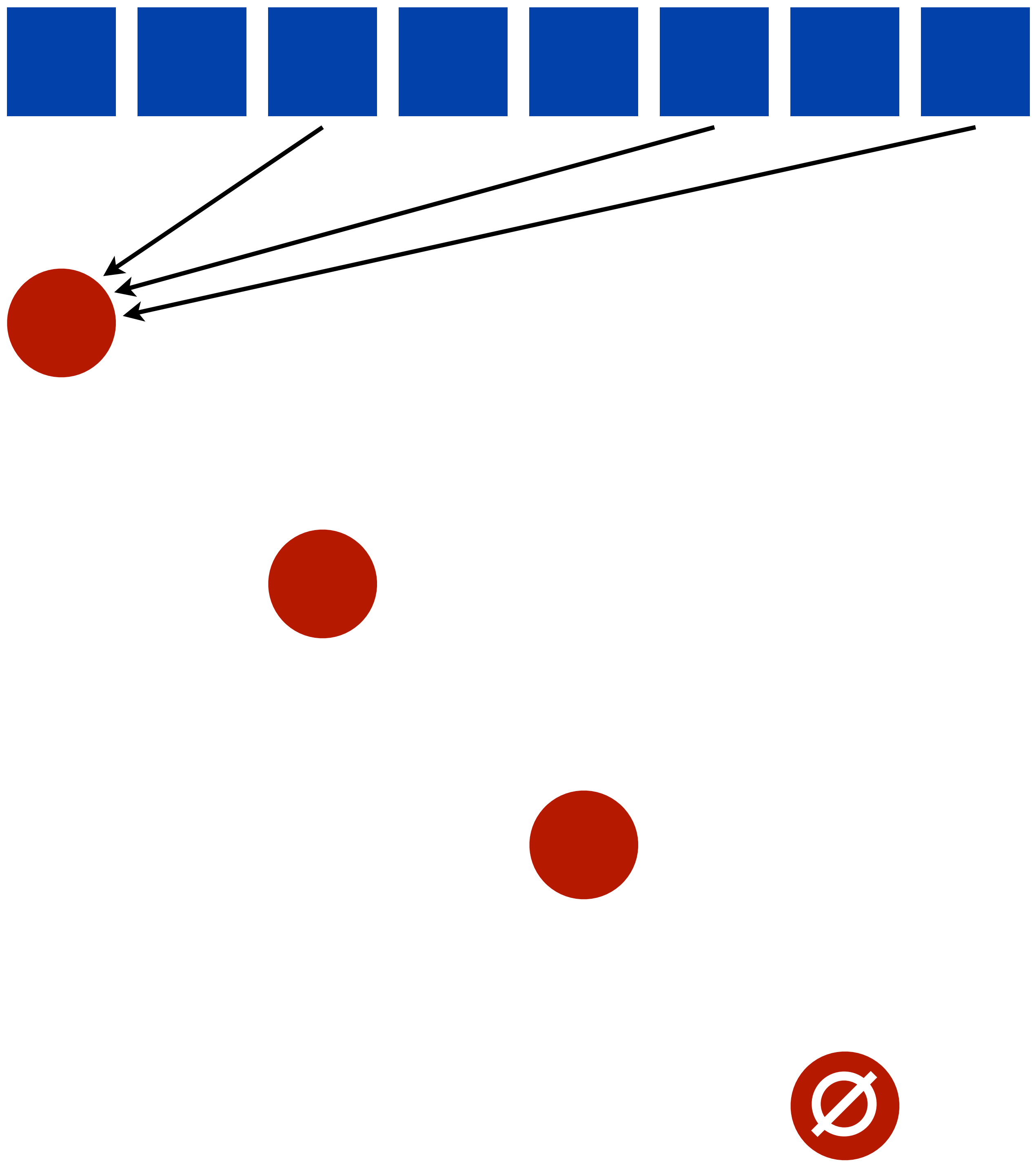
RUP Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4
● -1 2
● -1
● 2
∅



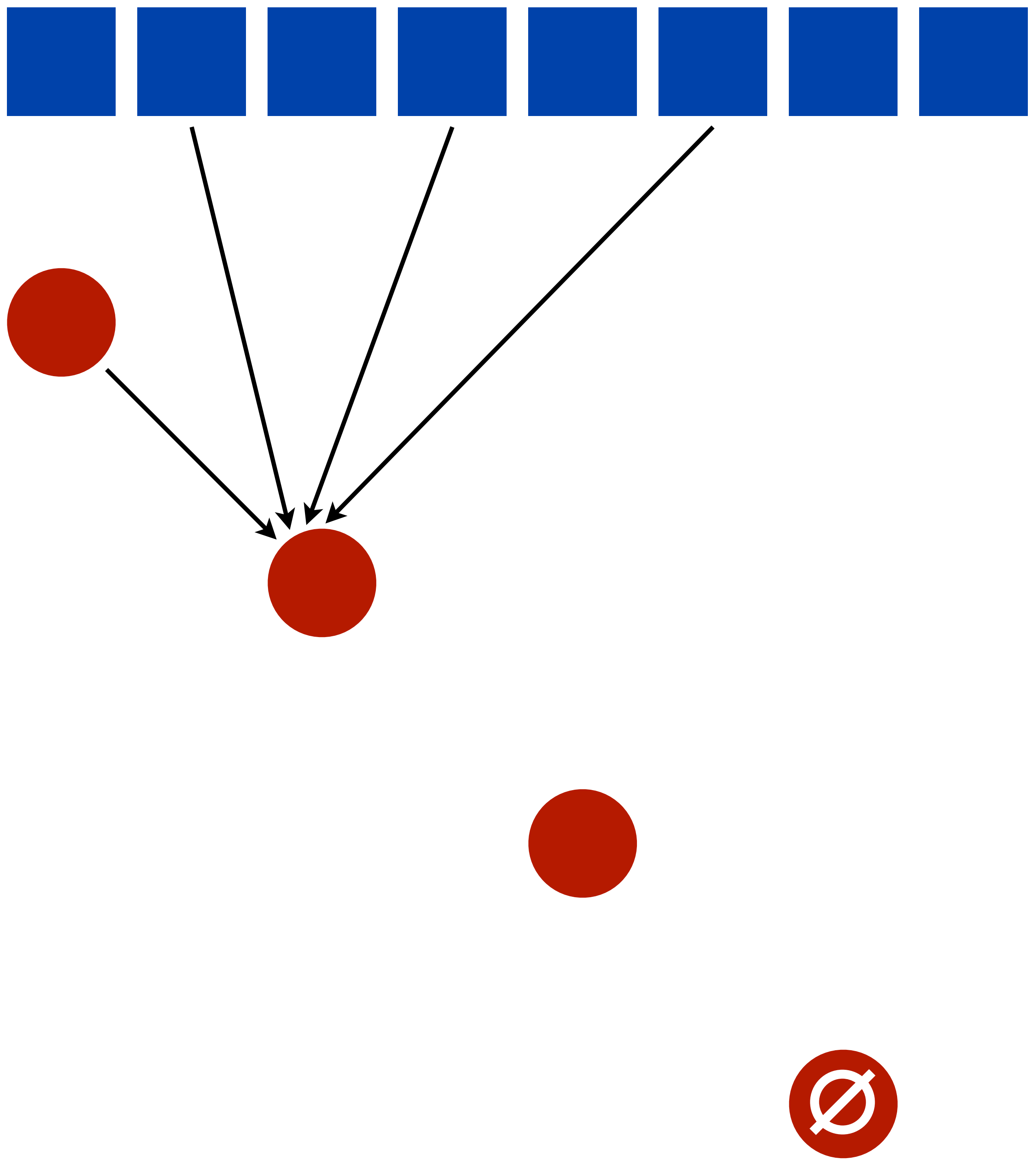
RUP Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4
● $\boxed{-1\ 2}$
● -1
● 2
∅



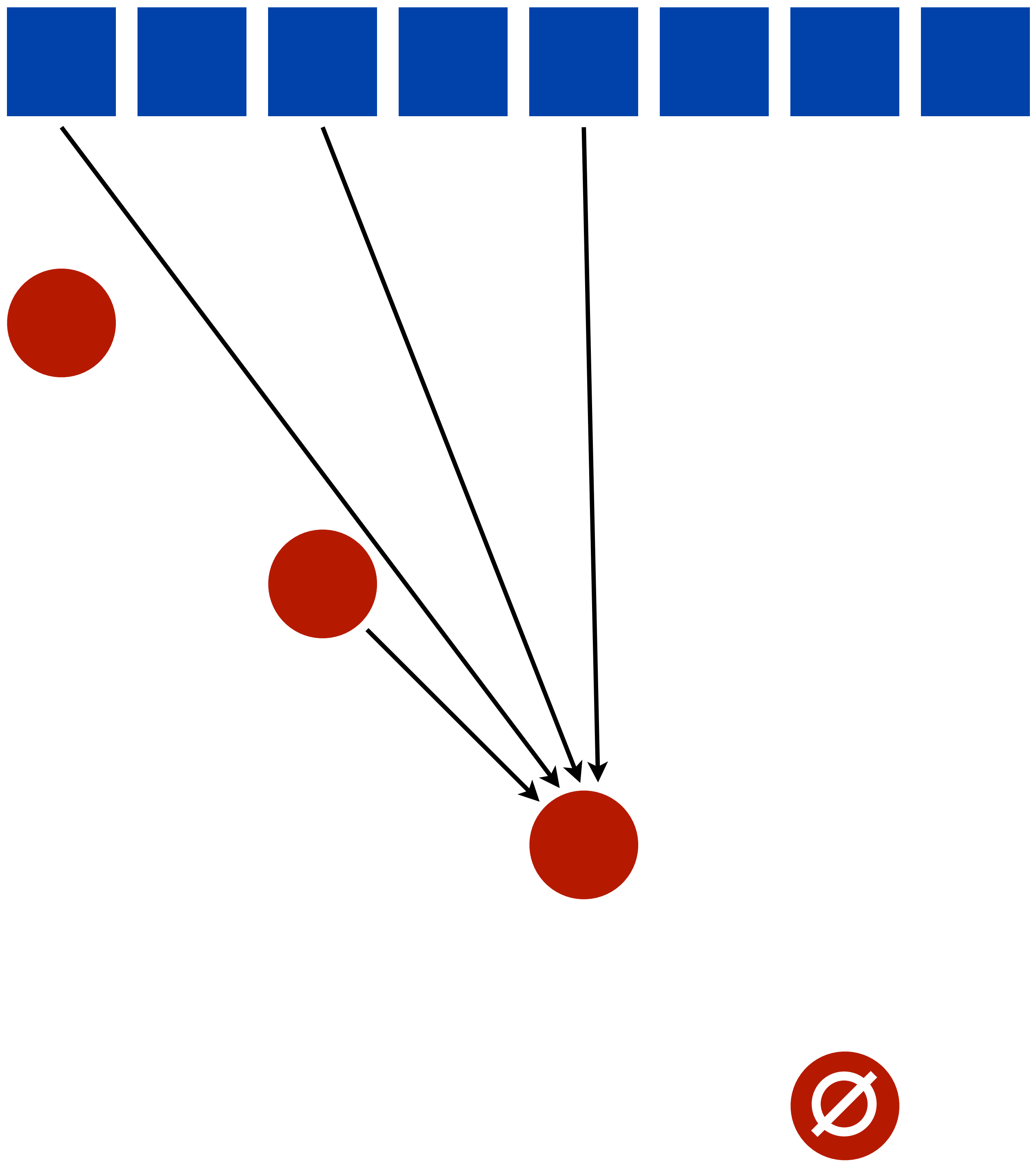
RUP Proof

```
■   1  2 -3
■  -1 -2  3
■   2  3 -4
■  -2 -3  4
■   1  3  4
■  -1 -3 -4
■   1 -2 -4
■  -1  2  4

●  -1  2
●  -1
●   2
∅
```



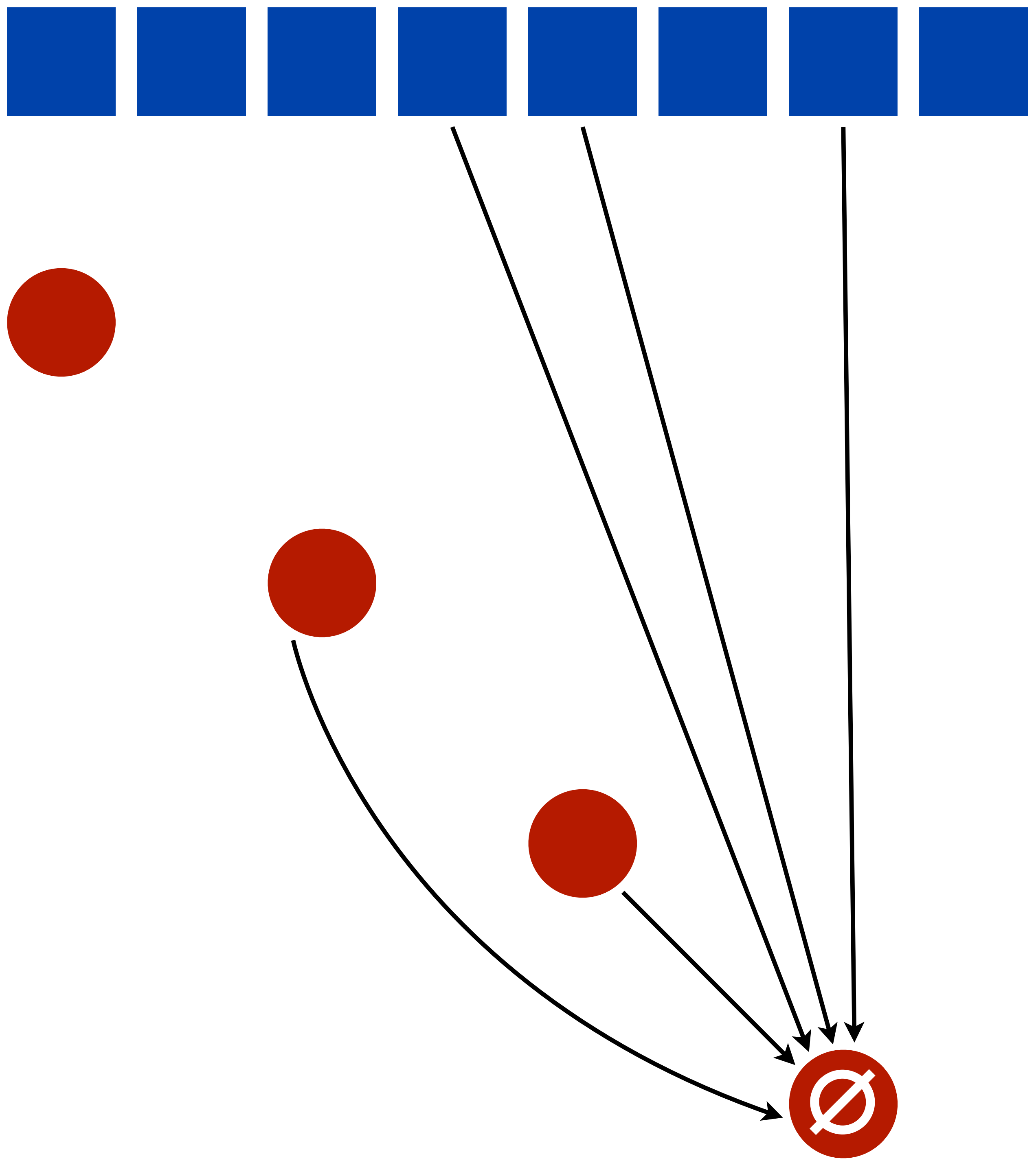
RUP Proof

```
■   1  2 -3
■  -1 -2  3
■   2  3 -4
■  -2 -3  4
■   1  3  4
■  -1 -3 -4
■   1 -2 -4
■  -1  2  4

●  -1  2
●  -1
●   2
∅
```



RUP Proof

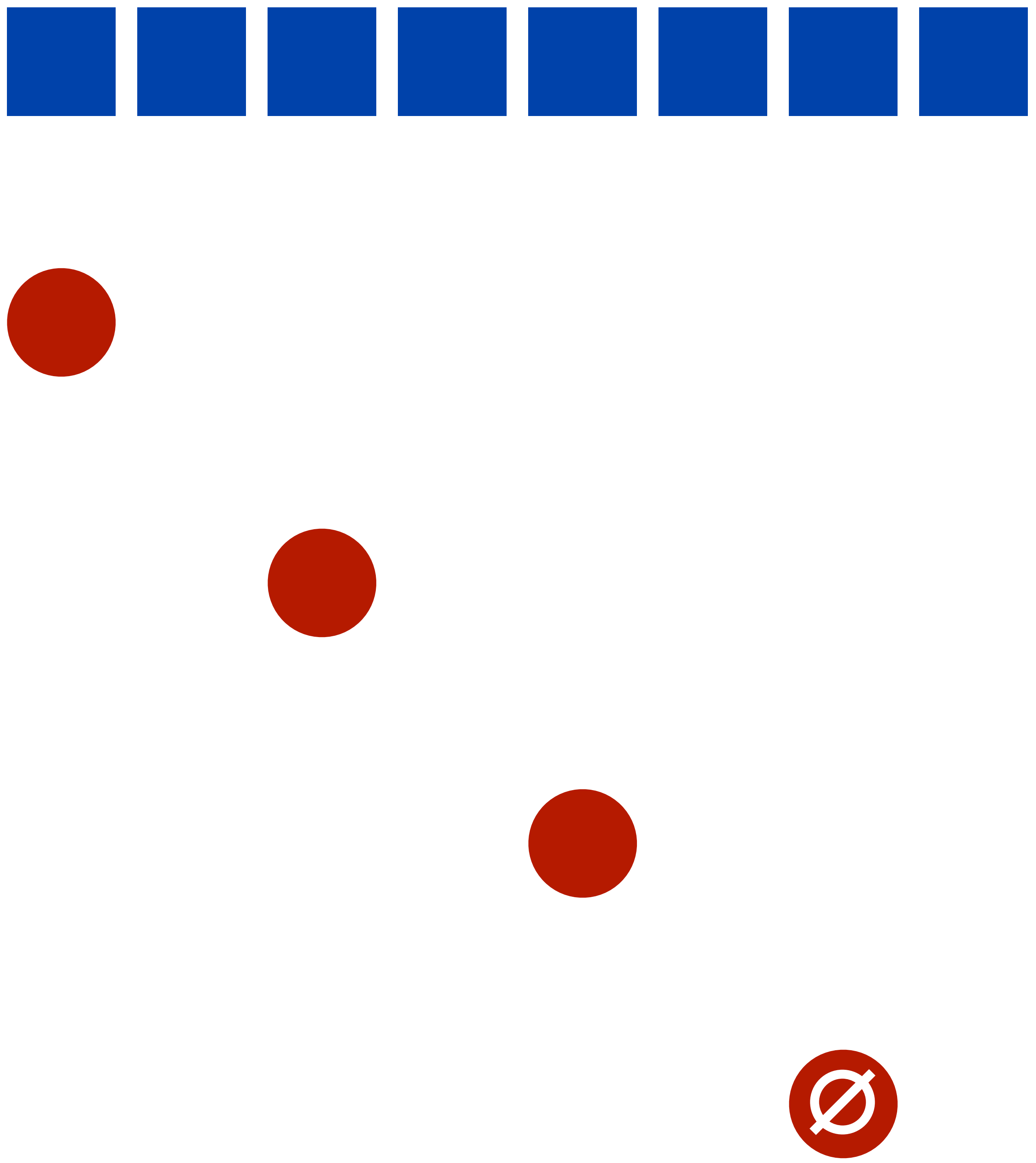
■	1	2	-3
■	-1	-2	3
■	2	3	-4
■	-2	-3	4
■	1	3	4
■	-1	-3	-4
■	1	-2	-4
■	-1	2	4
●	-1	2	
●	-1		
●	2		
∅	☐		



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

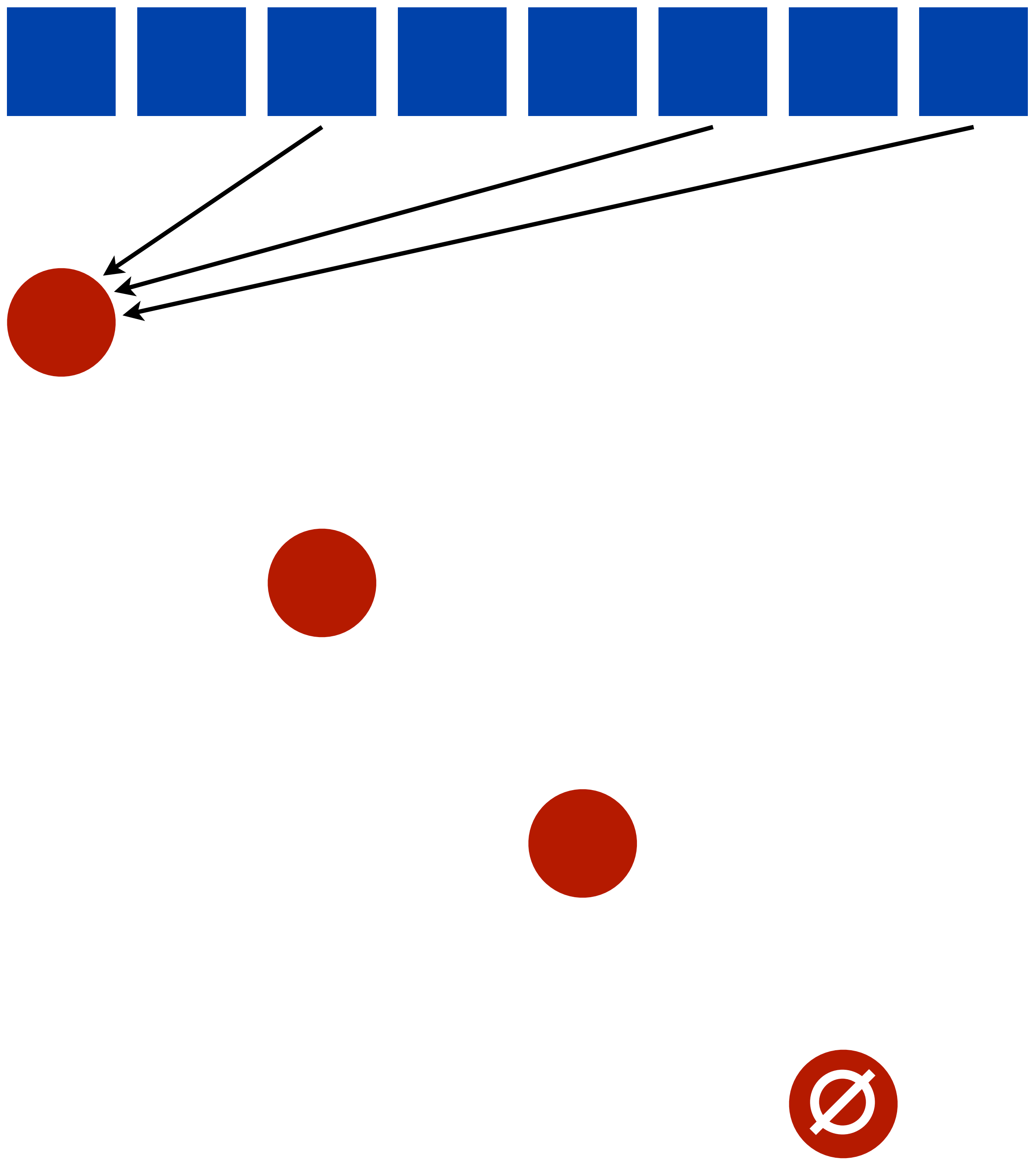
🔴 -1 2
◼ d -1 2 4
🔴 -1
◼ d -1 -2 3
◼ d -1 -3 -4
● d -1 2
🔴 2
◼ d 1 2 -3
◼ d 2 3 -4
∅



DRUP Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4

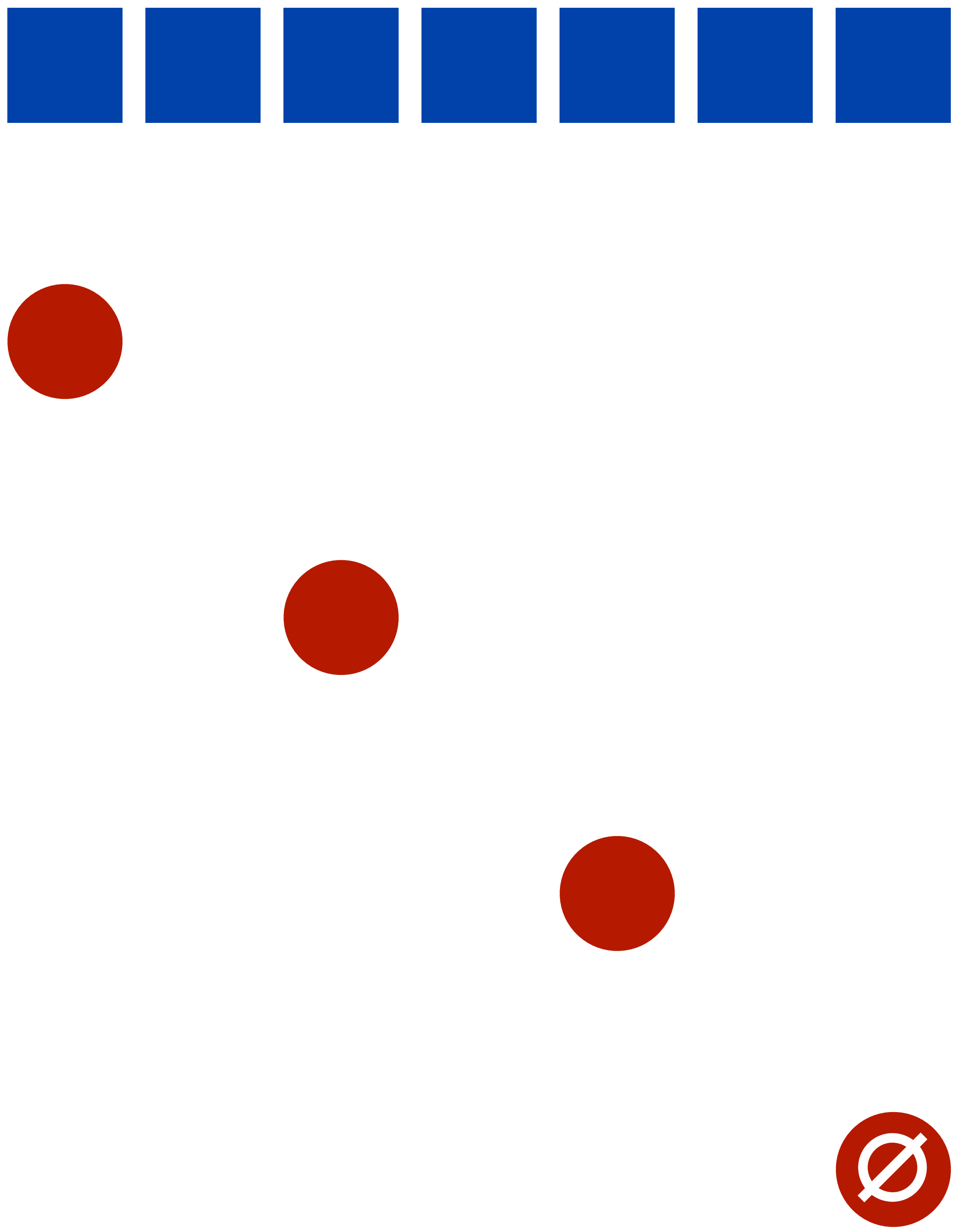
● -1 2
■ d -1 2 4
● -1
■ d -1 -2 3
■ d -1 -3 -4
● d -1 2
● 2
■ d 1 2 -3
■ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

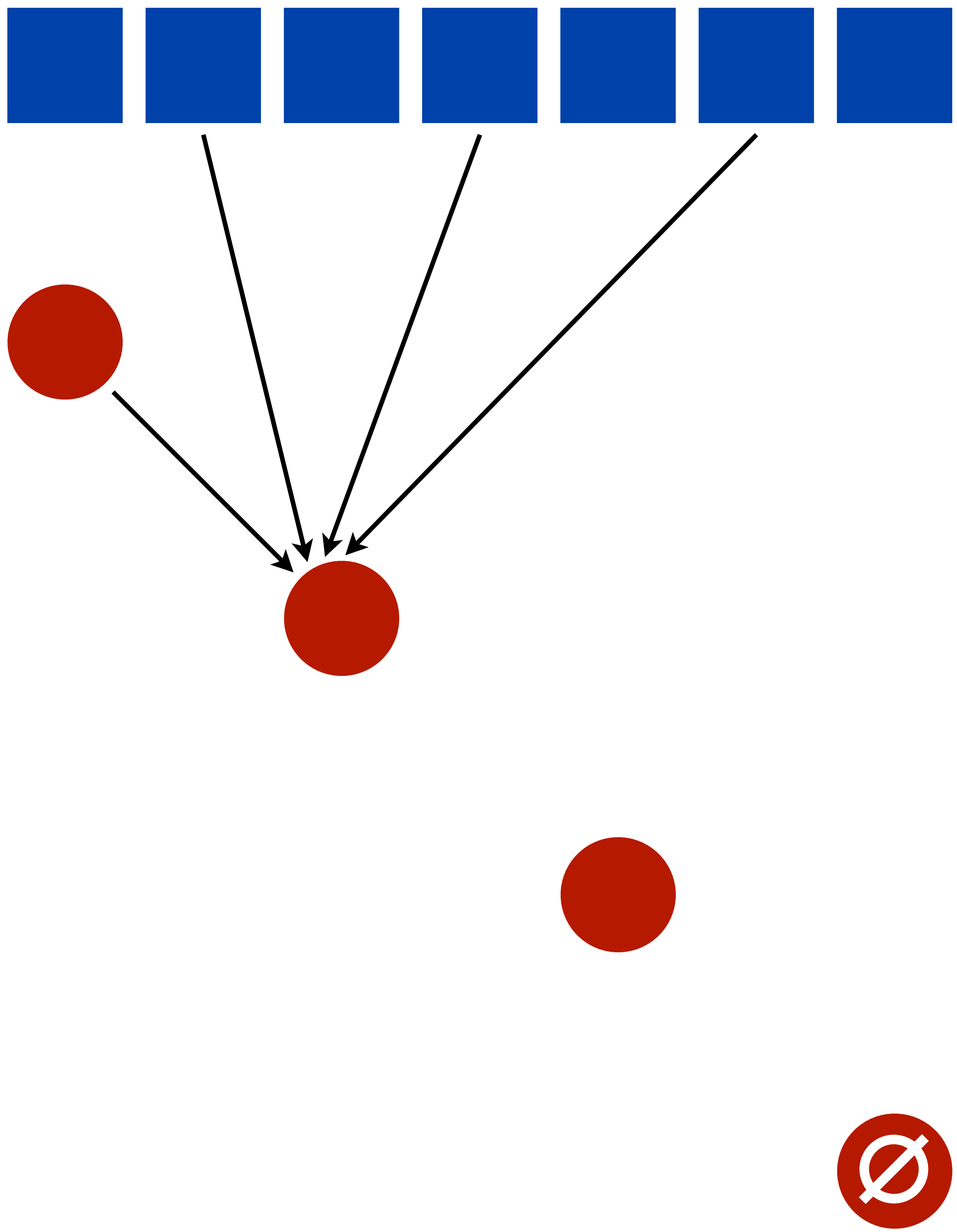
🔴 -1 2
⬛ d -1 2 4
🔴 -1
⬛ d -1 -2 3
⬛ d -1 -3 -4
⚫ d -1 2
🔴 2
⬛ d 1 2 -3
⬛ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

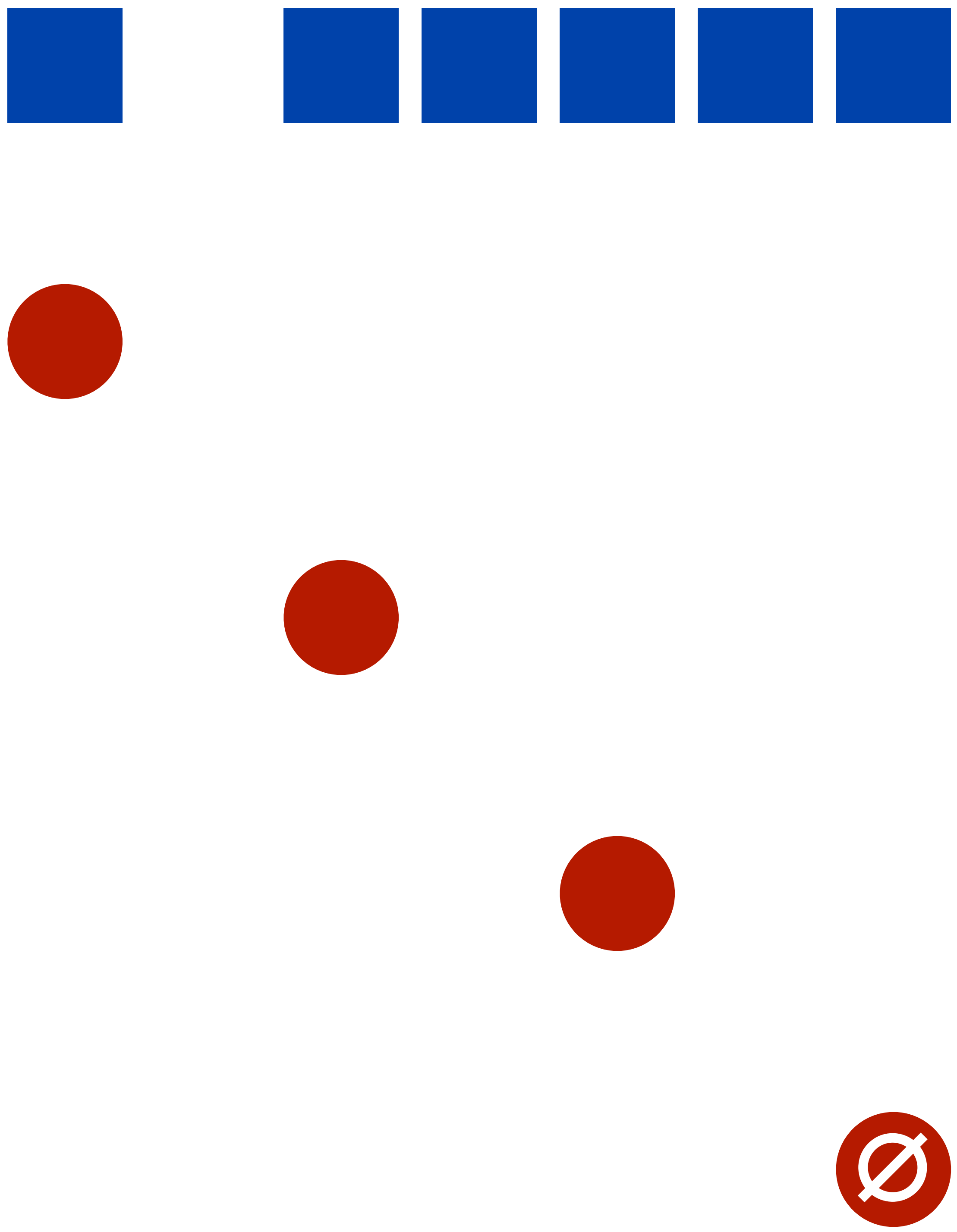
🔴 -1 2
⬛ d -1 2 4
🔴 -1
⬛ d -1 -2 3
⬛ d -1 -3 -4
● d -1 2
🔴 2
⬛ d 1 2 -3
⬛ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

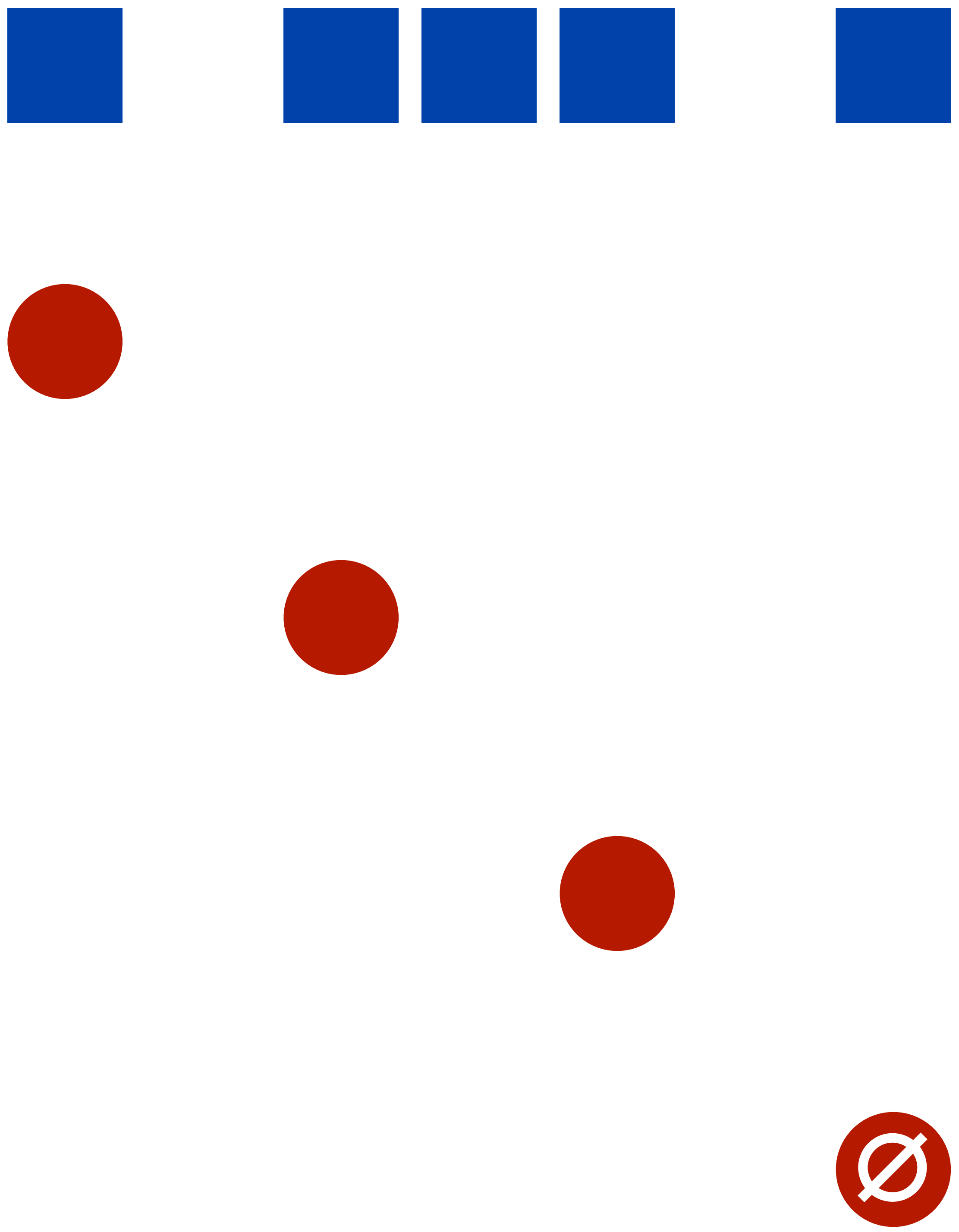
🔴 -1 2
◼ d -1 2 4
🔴 -1
$\boxed{\text{◼ d -1 -2 3}}$
◼ d -1 -3 -4
● d -1 2
🔴 2
◼ d 1 2 -3
◼ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

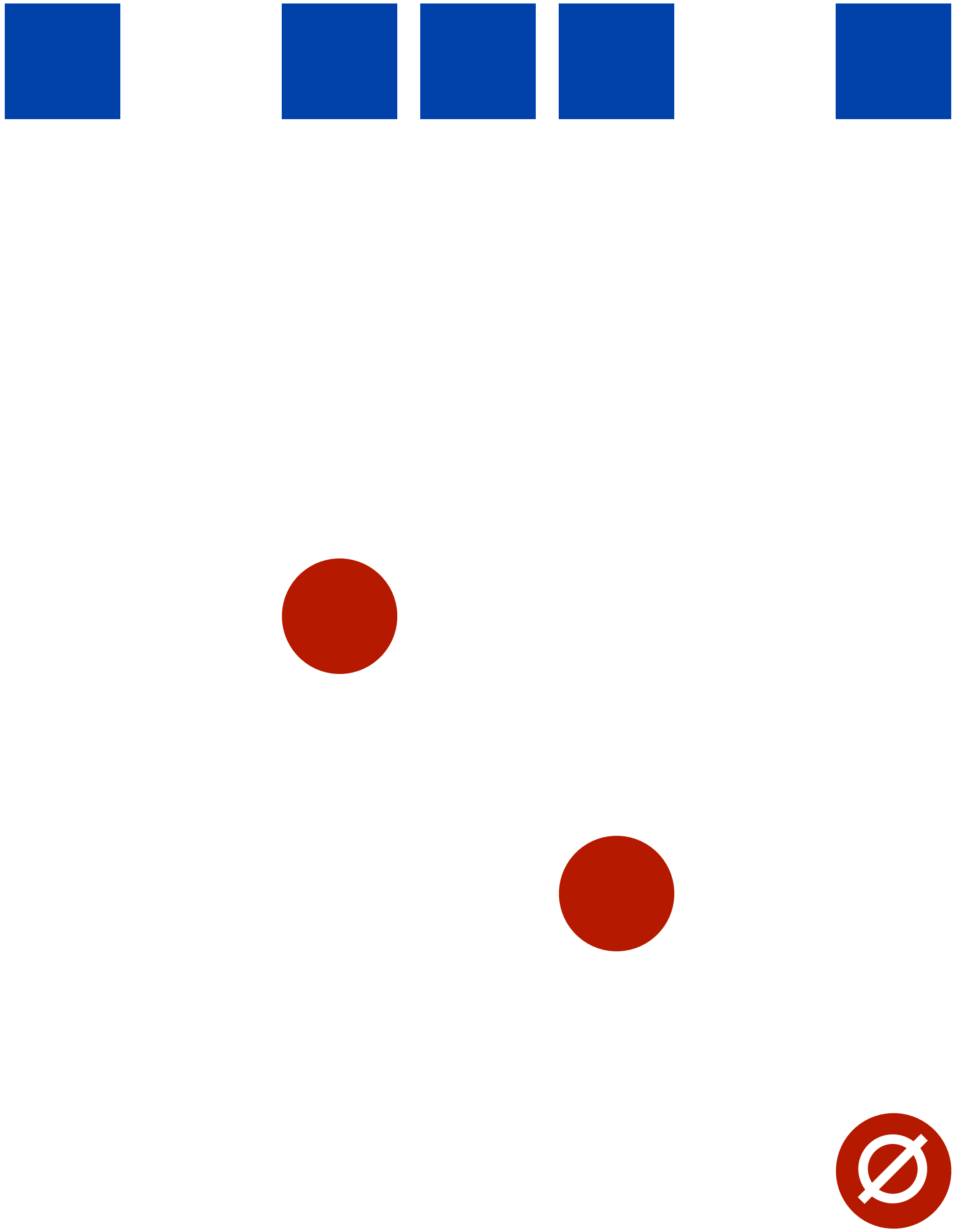
🔴 -1 2
■ d -1 2 4
🔴 -1
■ d -1 -2 3
■ d -1 -3 -4
● d -1 2
🔴 2
■ d 1 2 -3
■ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

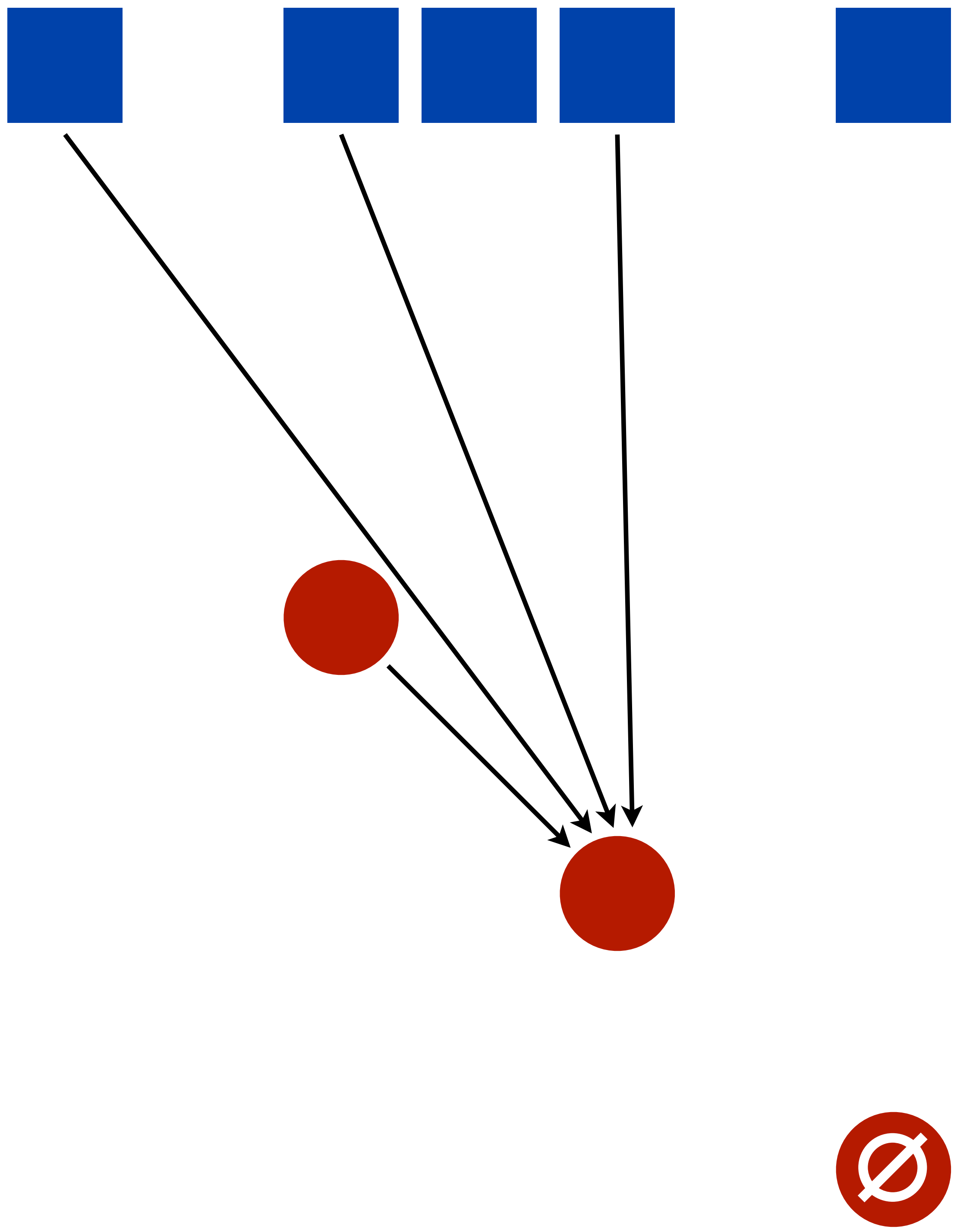
🔴 -1 2
◼ d -1 2 4
🔴 -1
◼ d -1 -2 3
◼ d -1 -3 -4
● d -1 2
🔴 2
◼ d 1 2 -3
◼ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

🔴 -1 2
⬛ d -1 2 4
🔴 -1
⬛ d -1 -2 3
⬛ d -1 -3 -4
⚫ d -1 2
🔴 2
⬛ d 1 2 -3
⬛ d 2 3 -4
∅



DRUP Proof

🟦 1 2 -3

🟦 -1 -2 3

🟦 2 3 -4

🟦 -2 -3 4

🟦 1 3 4

🟦 -1 -3 -4

🟦 1 -2 -4

🟦 -1 2 4

🔴 -1 2

■ d -1 2 4

🔴 -1

■ d -1 -2 3

■ d -1 -3 -4

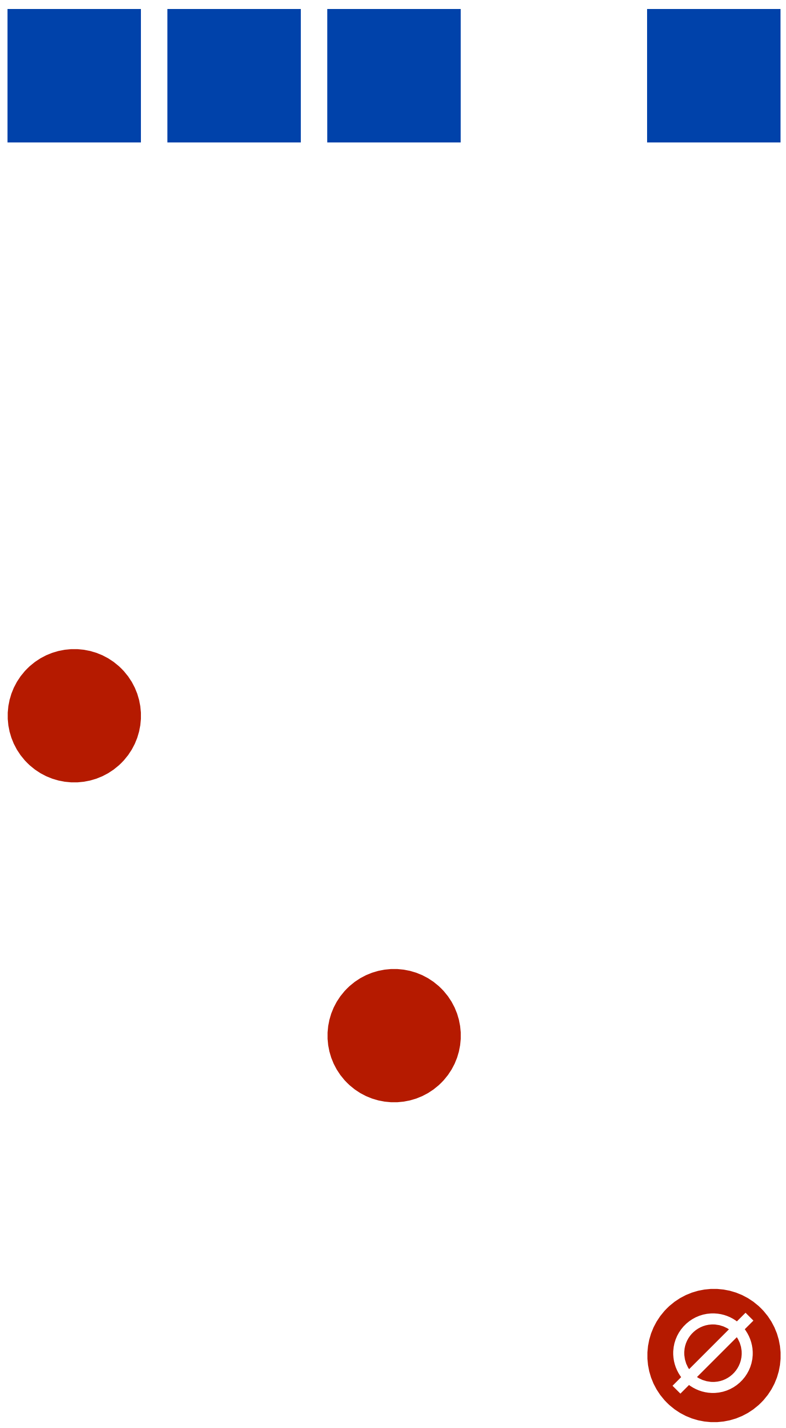
● d -1 2

🔴 2

■ d 1 2 -3

■ d 2 3 -4

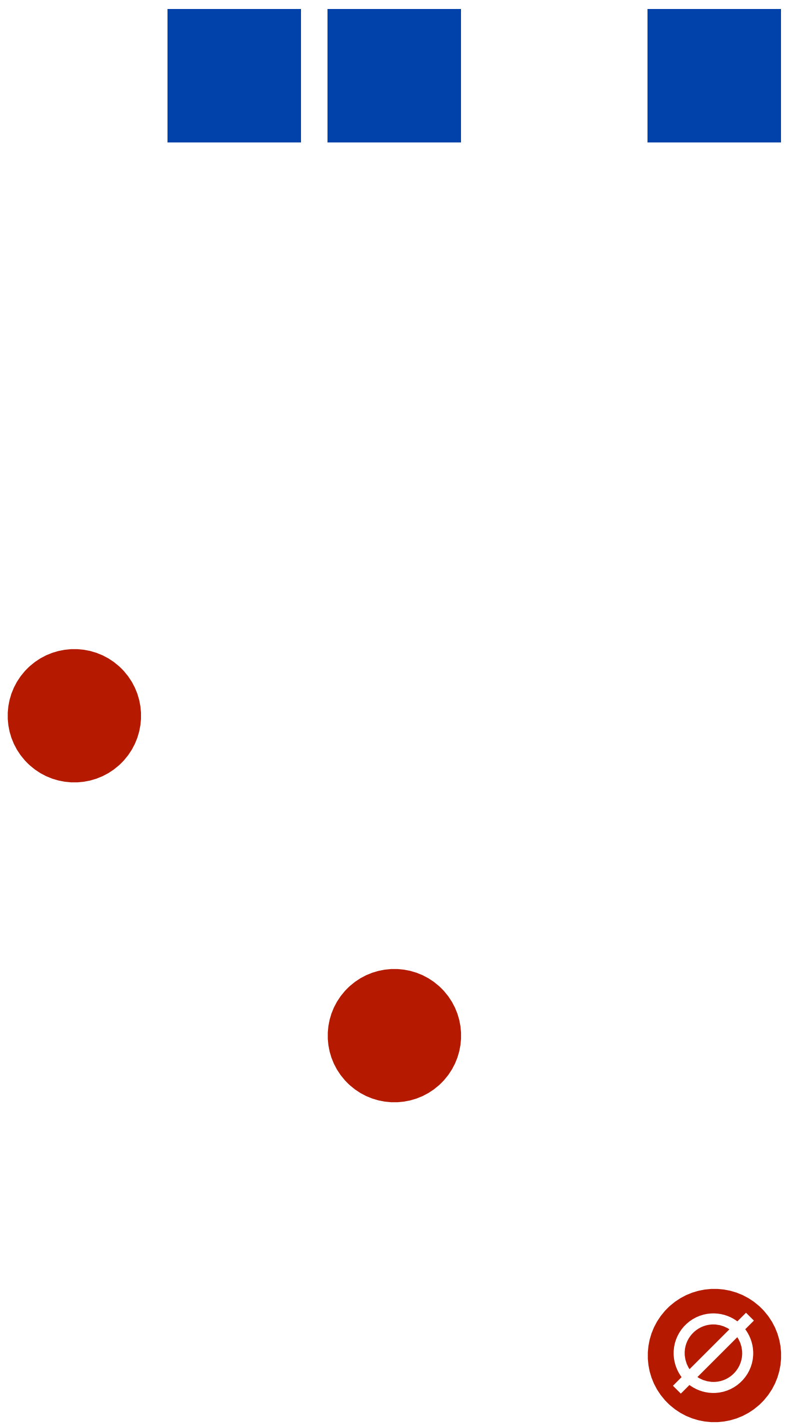
∅



DRUP Proof

🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

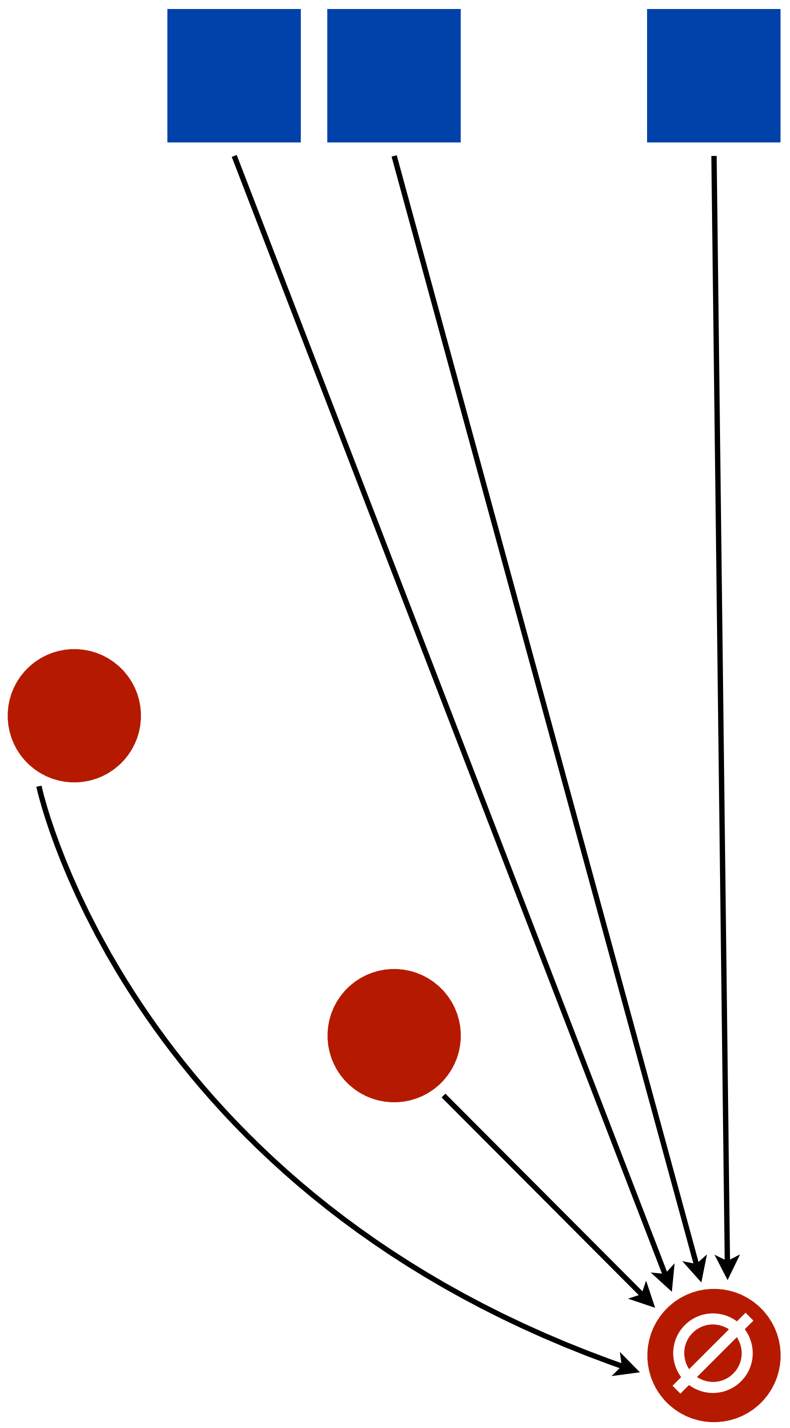
🔴 -1 2
■ d -1 2 4
🔴 -1
■ d -1 -2 3
■ d -1 -3 -4
● d -1 2
🔴 2
■ d 1 2 -3
■ d 2 3 -4
∅



DRUP Proof

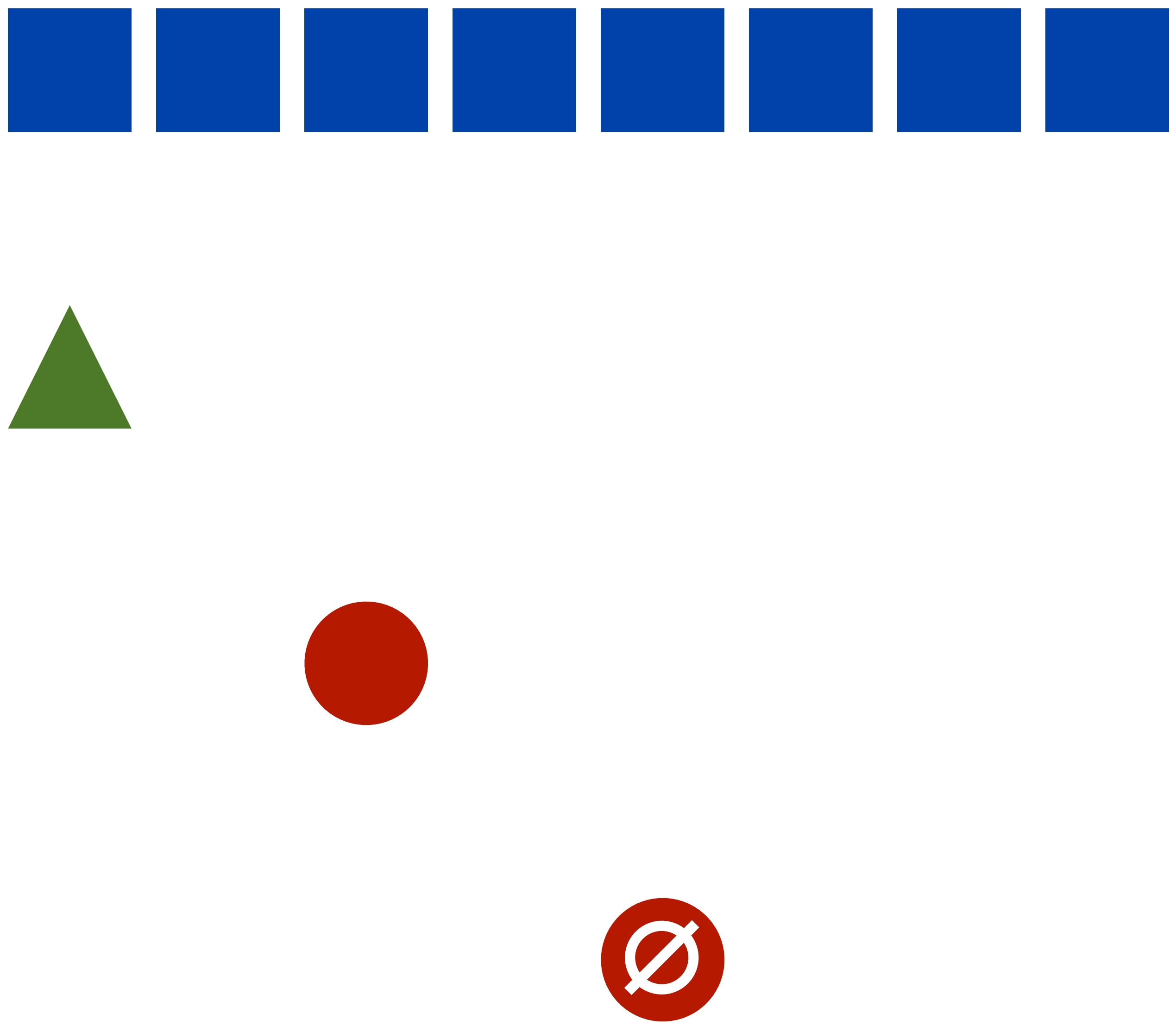
🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4

🔴 -1 2
⬛ d -1 2 4
🔴 -1
⬛ d -1 -2 3
⬛ d -1 -3 -4
⚫ d -1 2
🔴 2
⬛ d 1 2 -3
⬛ d 2 3 -4
∅



RAT Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4
▲ -1
● 2
∅



RAT Proof

■ 1 2 -3

■ -1 -2 3

■ 2 3 -4

■ -2 -3 4

■ 1 3 4

■ -1 -3 -4

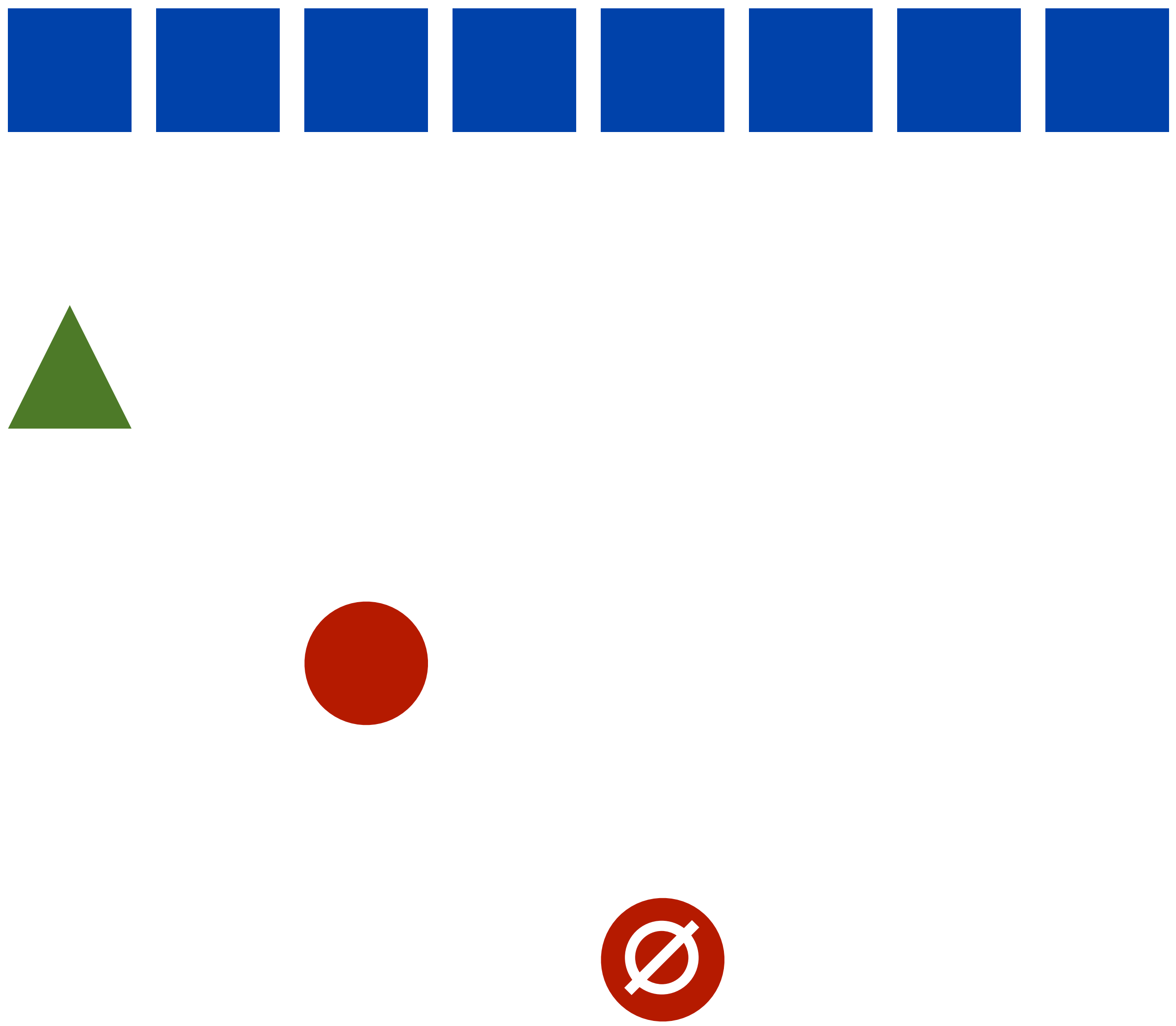
■ 1 -2 -4

■ -1 2 4

▲ -1

● 2

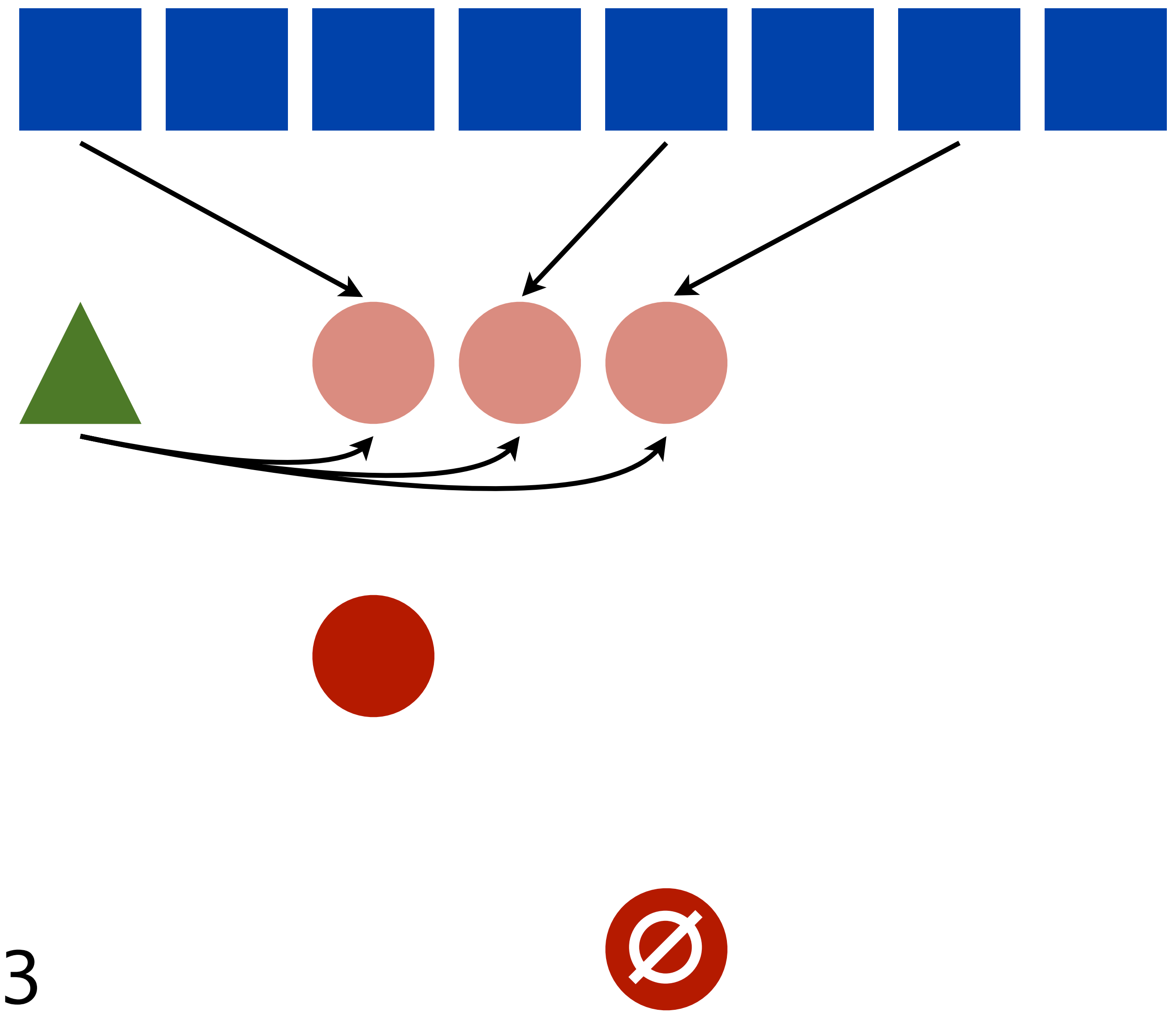
∅



RAT Proof

■	1	2	-3
■	-1	-2	3
■	2	3	-4
■	-2	-3	4
■	1	3	4
■	-1	-3	-4
■	1	-2	-4
■	-1	2	4
▲	-1		
●	2		
∅			

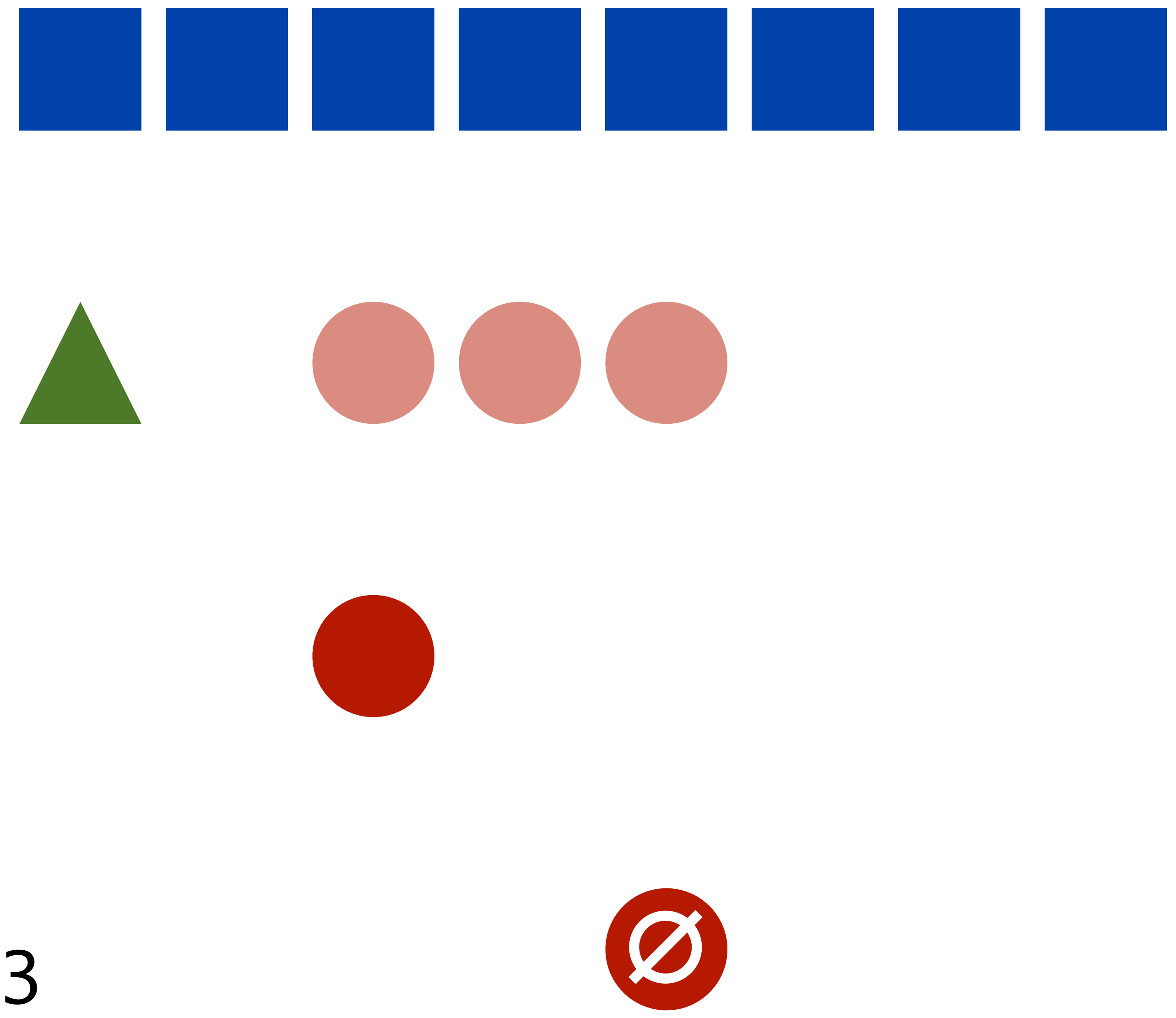
●	2	-3
●	3	4
●	-2	-4



RAT Proof

■	1 2 -3
■	-1 -2 3
■	2 3 -4
■	-2 -3 4
■	1 3 4
■	-1 -3 -4
■	1 -2 -4
■	-1 2 4
▲	-1
●	2
∅	

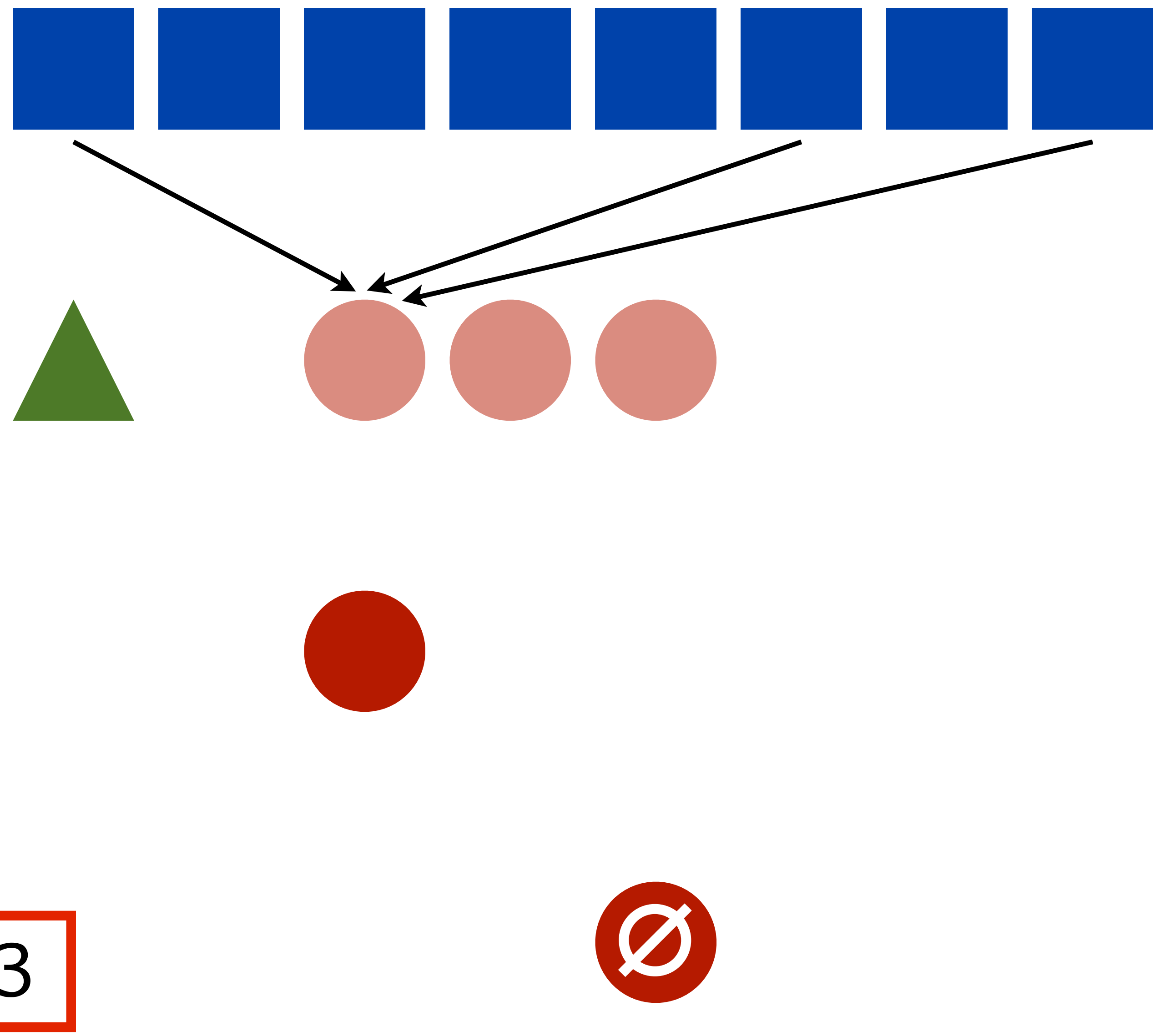
●	2 -3
●	3 4
●	-2 -4



RAT Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4
▲ -1
● 2
∅

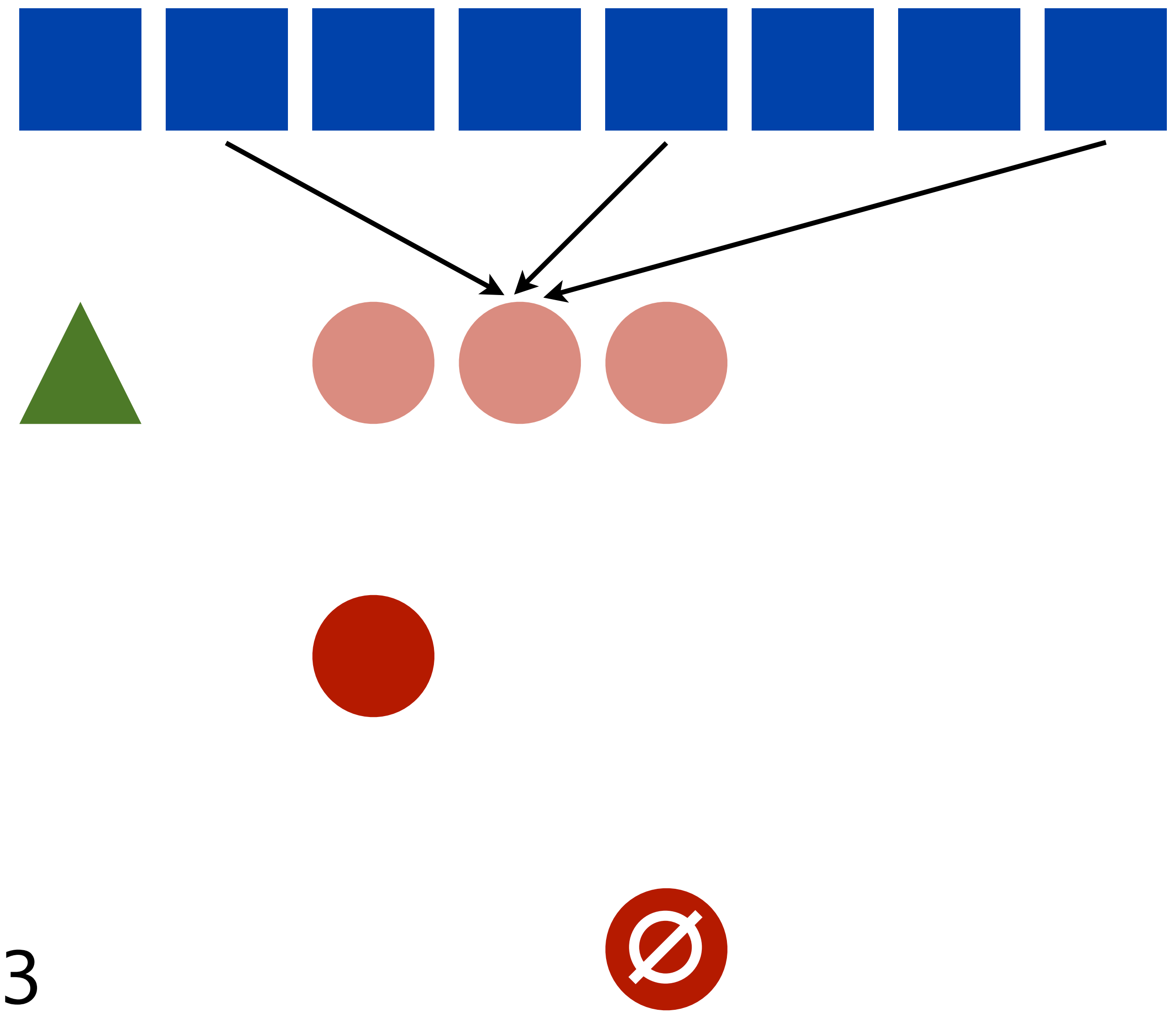
● 2 -3
● 3 4
● -2 -4



RAT Proof

■ 1 2 -3
■ -1 -2 3
■ 2 3 -4
■ -2 -3 4
■ 1 3 4
■ -1 -3 -4
■ 1 -2 -4
■ -1 2 4
▲ -1
● 2
∅

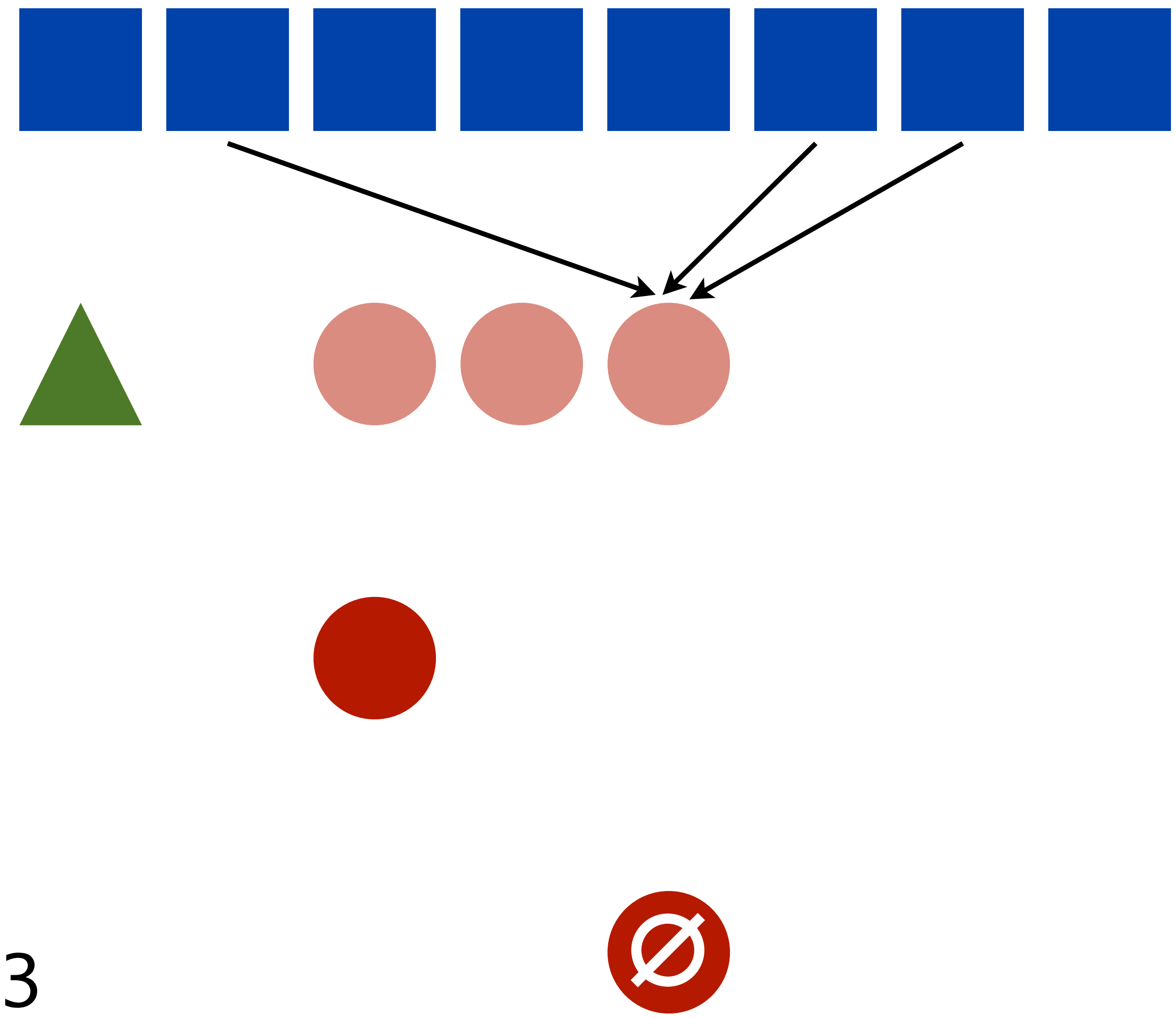
● 2 -3
● 3 4
● -2 -4



RAT Proof

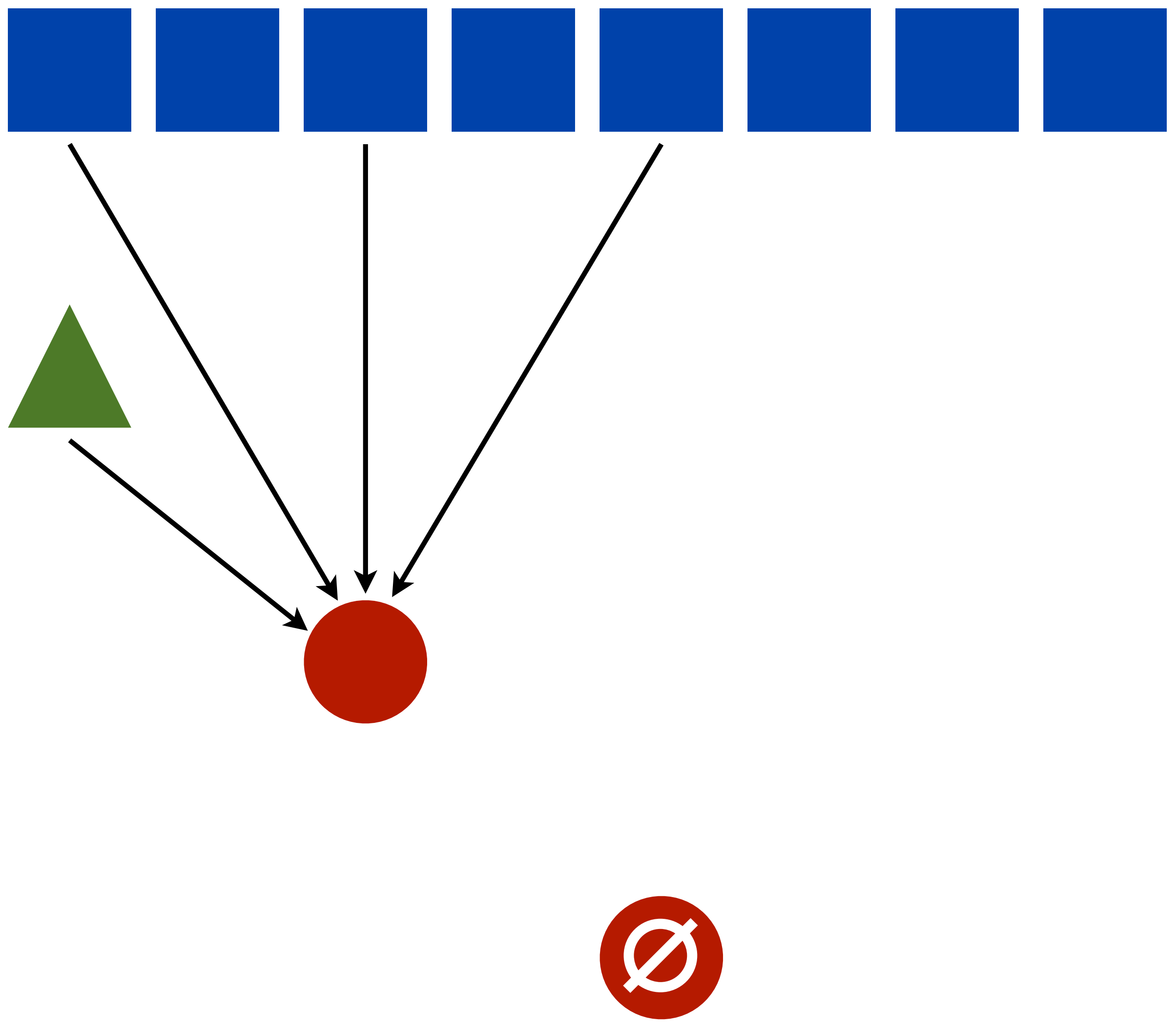
🟦 1 2 -3
🟦 -1 -2 3
🟦 2 3 -4
🟦 -2 -3 4
🟦 1 3 4
🟦 -1 -3 -4
🟦 1 -2 -4
🟦 -1 2 4
▲ -1
● 2
∅

● 2 -3
● 3 4
● -2 -4



RAT Proof

■	1	2	-3
■	-1	-2	3
■	2	3	-4
■	-2	-3	4
■	1	3	4
■	-1	-3	-4
■	1	-2	-4
■	-1	2	4
▲	-1		
●	2		
∅			



RAT Proof

🟦 1 2 -3

🟦 -1 -2 3

🟦 2 3 -4

🟦 -2 -3 4

🟦 1 3 4

🟦 -1 -3 -4

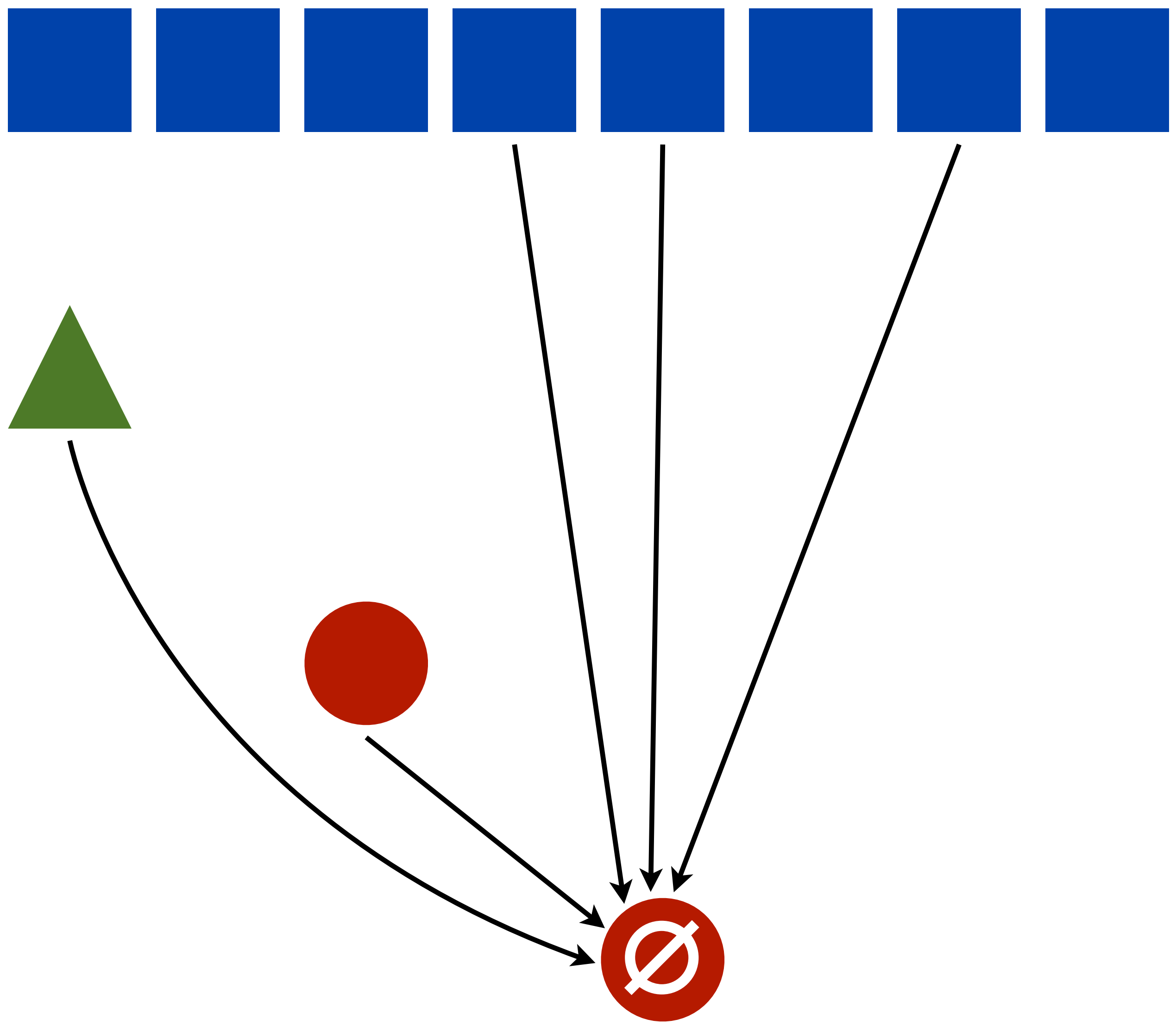
🟦 1 -2 -4

🟦 -1 2 4

▲ -1

● 2

∅ ☐



Incremental Approach

Efficient code can be difficult to verify.

- STOBJs provide array-like memory, but require complex invariants
- Abstract STOBJs simplify these invariants by maintaining an equivalence
- Currently developing a “cons-less” model that does not use STOBJs, but organizes data structures in a similar way.
- Refinements
- Litany of transformations eventually resulting in array-like code

Proof Properties

🟦 Easy to Emit

🟩 Compact

🟥 Checked Efficiently

🟪 Verified Checker

🟨 Expressive

🟥 Resolution Proofs

🟥🟪 with Verified Checker

🟦🟩 Clausal (RUP) Proofs

🟦🟩🟥 DRUP (DRUP-Trim)

🟦🟩🟨 RAT Proofs

🟦🟩🟪🟨 with Verified Checker

🟦🟩🟥🟪🟨 Proposed Work