Trimming while Checking Clausal Proofs

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

The University of Texas at Austin

Formal Methods in Computer-Aided Design (FMCAD)
Portland, Oregon
October 23, 2013

Outline

- Motivation and Contributions
- Resolution versus Clausal Proofs
- Checking Clausal Proofs Efficiently
- Experimental Evaluation
- Conclusion

Motivation

Motivation

SAT solvers are used in many tools and applications.

- Counter-examples (satisfiable) using symbolic simulation;
- Equivalence-checking (unsatisfiable) using miters;
- Small explanations (unsatisfiable core) for diagnosis;
- Small (trimmed) proofs to validate with a verified checker.

Motivation

SAT solvers are used in many tools and applications.

- Counter-examples (satisfiable) using symbolic simulation;
- Equivalence-checking (unsatisfiable) using miters;
- Small explanations (unsatisfiable core) for diagnosis;
- Small (trimmed) proofs to validate with a verified checker.

However,

- Documented bugs in SAT, SMT, and QBF solvers [Brummayer and Biere, 2009; Brummayer et al., 2010];
- Solvers that emit additional information use lots of memory.

Motivation

SAT solvers are used in many tools and applications.

- Counter-examples (satisfiable) using symbolic simulation;
- Equivalence-checking (unsatisfiable) using miters;
- Small explanations (unsatisfiable core) for diagnosis;
- Small (trimmed) proofs to validate with a verified checker.

However,

- Documented bugs in SAT, SMT, and QBF solvers [Brummayer and Biere, 2009; Brummayer et al., 2010];
- Solvers that emit additional information use lots of memory.

We developed a tool that can efficiently validate the results of SAT solvers and produce trimmed formulas and trimmed proofs

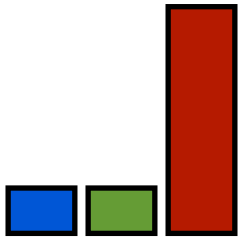
Contributions and Related Work

Contributions and Related Work

Easy to Emit

Compact

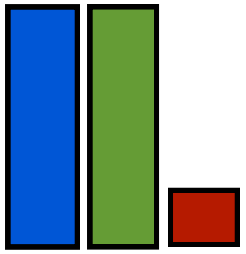
Checked Efficiently



Resolution Proofs

Zhang and Malik, 2003

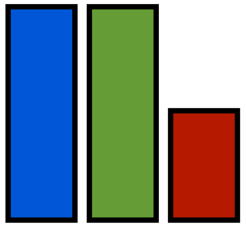
Van Gelder, 2008; Biere, 2008



Clausal Proofs

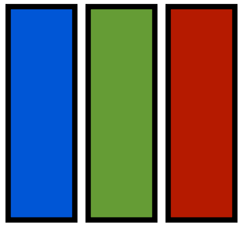
Goldberg and Novikov, 2003

Van Gelder, 2008



Clausal proofs + clause deletion

Heule, Hunt, Jr., and Wetzler [STVR 201X]



A fast clausal proof checker, called DRUP-trim

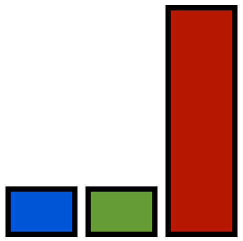
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]

Contributions and Related Work

🟦 Easy to Emit

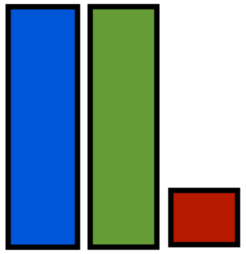
🟩 Compact

🟥 Checked Efficiently



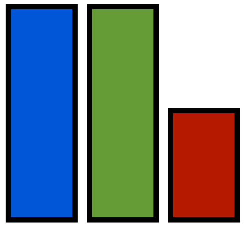
Resolution Proofs

Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



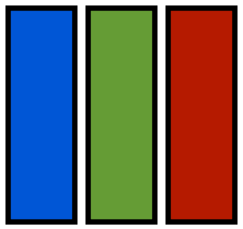
Clausal Proofs

Goldberg and Novikov, 2003
Van Gelder, 2008



Clausal proofs + clause deletion

Heule, Hunt, Jr., and Wetzler [STVR 201X]



A fast clausal proof checker, called DRUP-trim

Heule, Hunt, Jr., and Wetzler [FMCAD 2013]

All approaches can be used for applications such as minimal unsatisfiable core extraction, computing interpolants, reduce proofs

Satisfiability and Resolution

Satisfiability and Resolution

Given a Boolean formula F, is there an assignment to variables in F such that the formula evaluates to TRUE?

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

Satisfiability and Resolution

Given a Boolean formula *F*, is there an assignment to variables in *F* such that the formula evaluates to TRUE?

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

Checking a solution, such as assignment $\bar{a}\ \bar{b}\ c$, is easy.

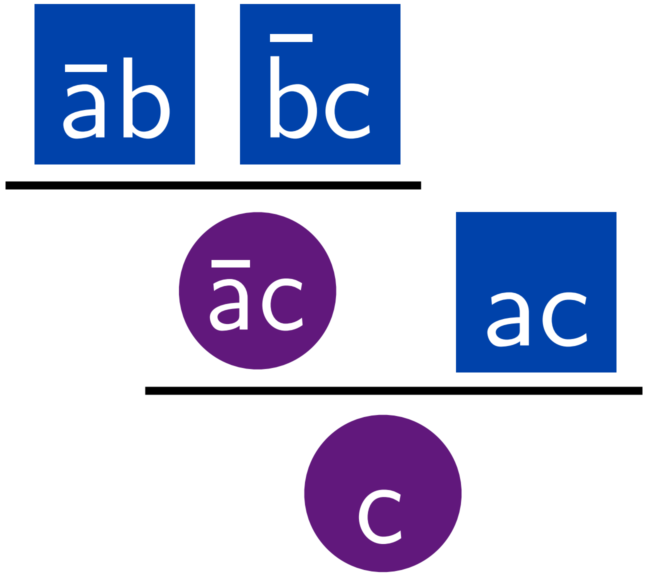
Satisfiability and Resolution

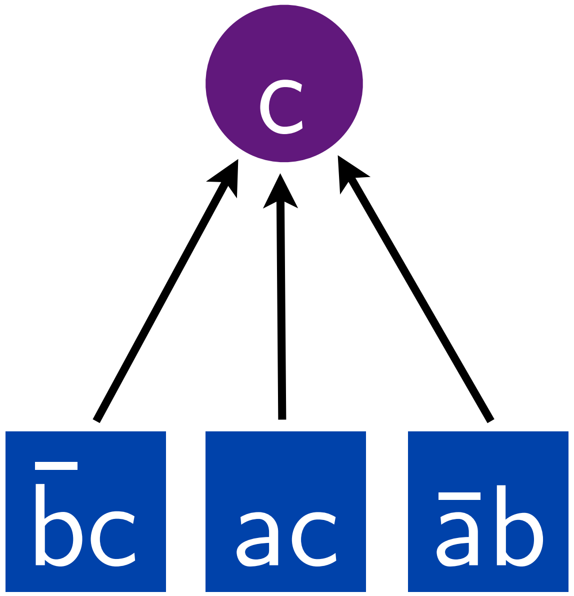
Given a Boolean formula *F*, is there an assignment to variables in *F* such that the formula evaluates to TRUE?

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

Checking a solution, such as assignment $\bar{a}\ \bar{b}\ c$, is easy.

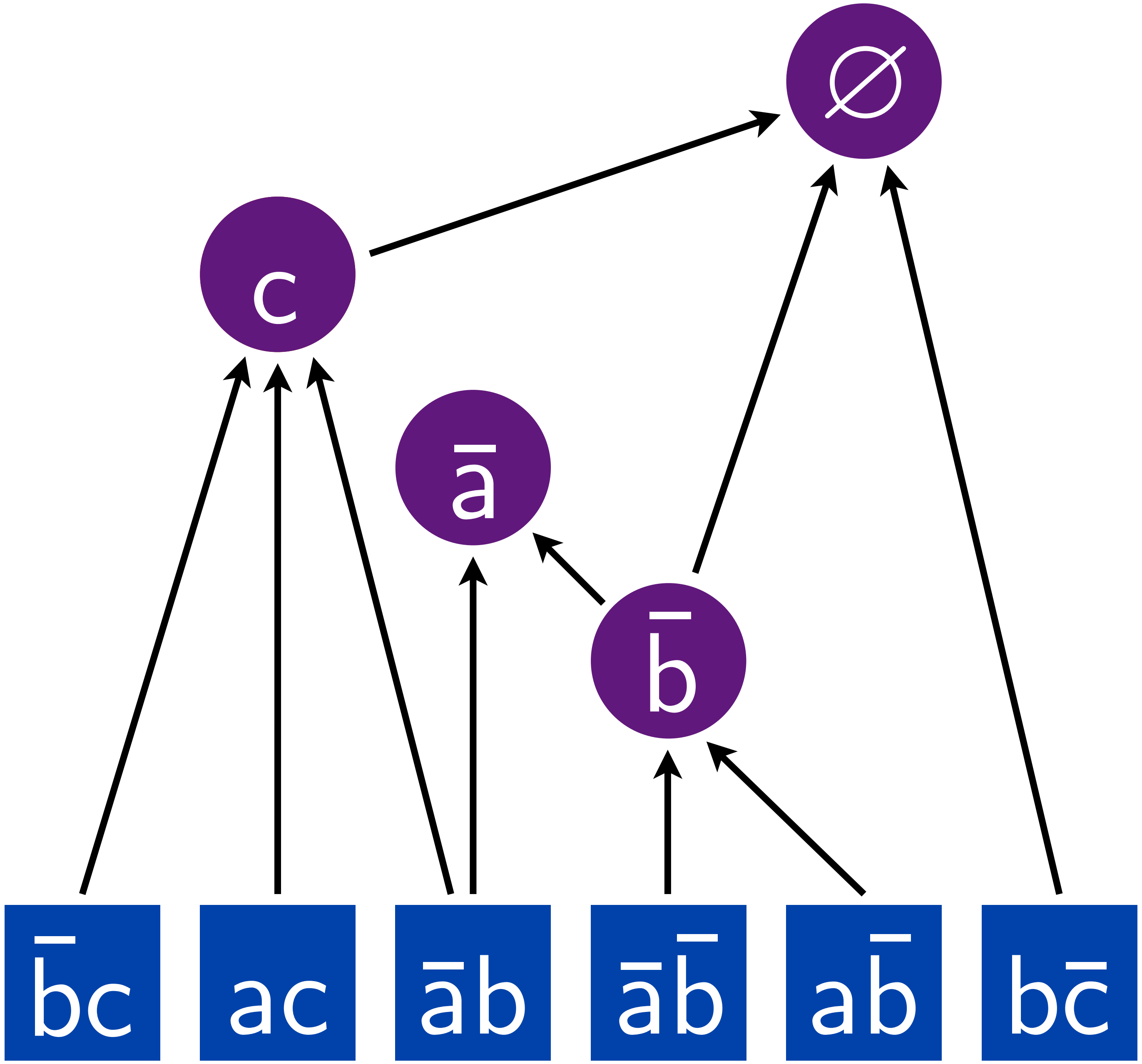
Unsatisfiability proofs use lemmas (resolvents):





Resolution Graph / Proof and Core

Resolution Graph / Proof and Core

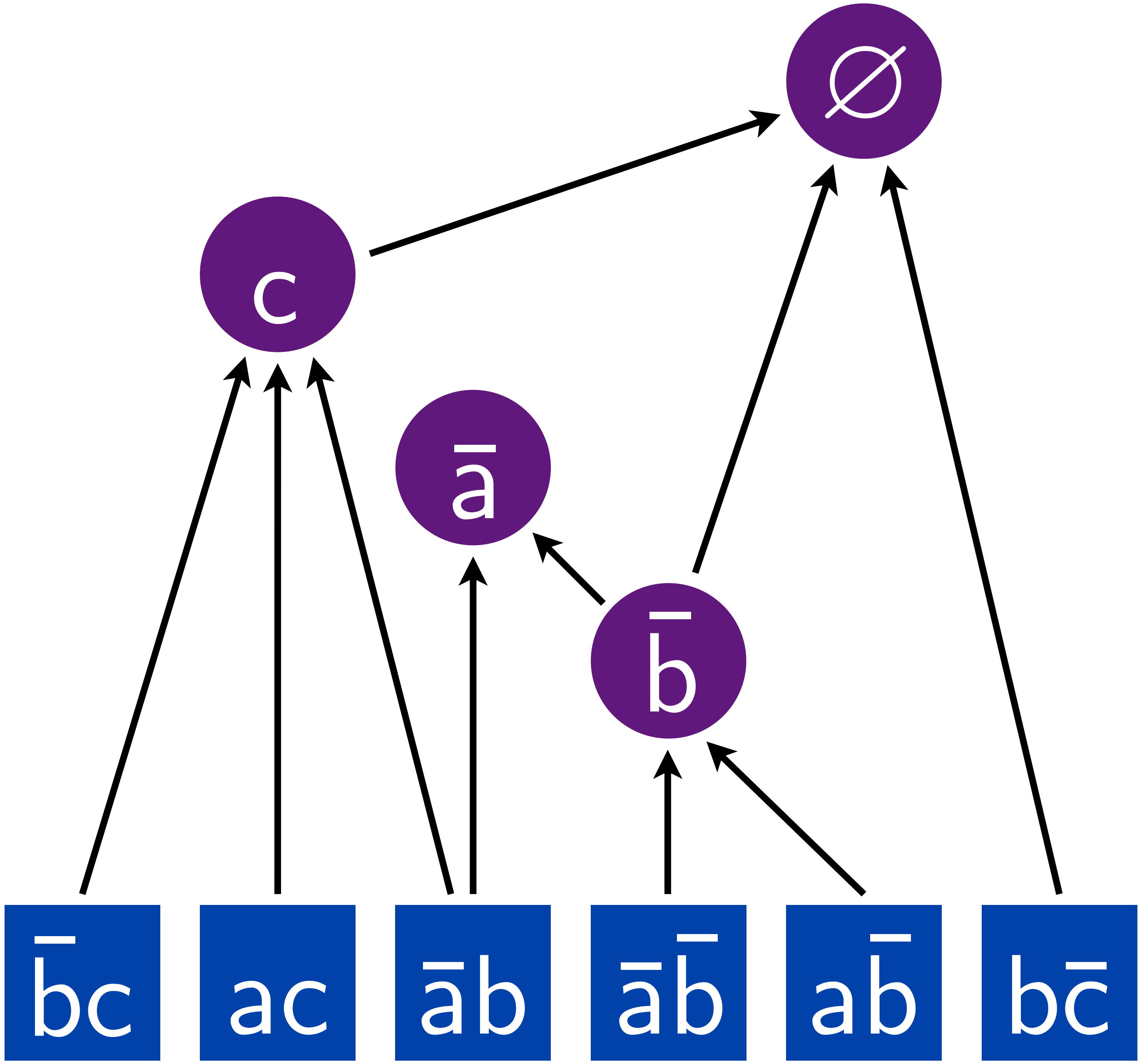


resolution graph

Resolution Graph / Proof and Core

$\bar{b}c$	
ac	
$\bar{a}b$	
$\bar{a}\bar{b}$	
$a\bar{b}$	
$b\bar{c}$	
$\bar{b}$	$\bar{a}\bar{b}$ $a\bar{b}$
$\bar{a}$	$\bar{a}b$ $\bar{b}$
c	ac $\bar{a}b$ $\bar{b}c$
$\emptyset$	$b\bar{c}$ $\bar{b}$ c

resolution proof

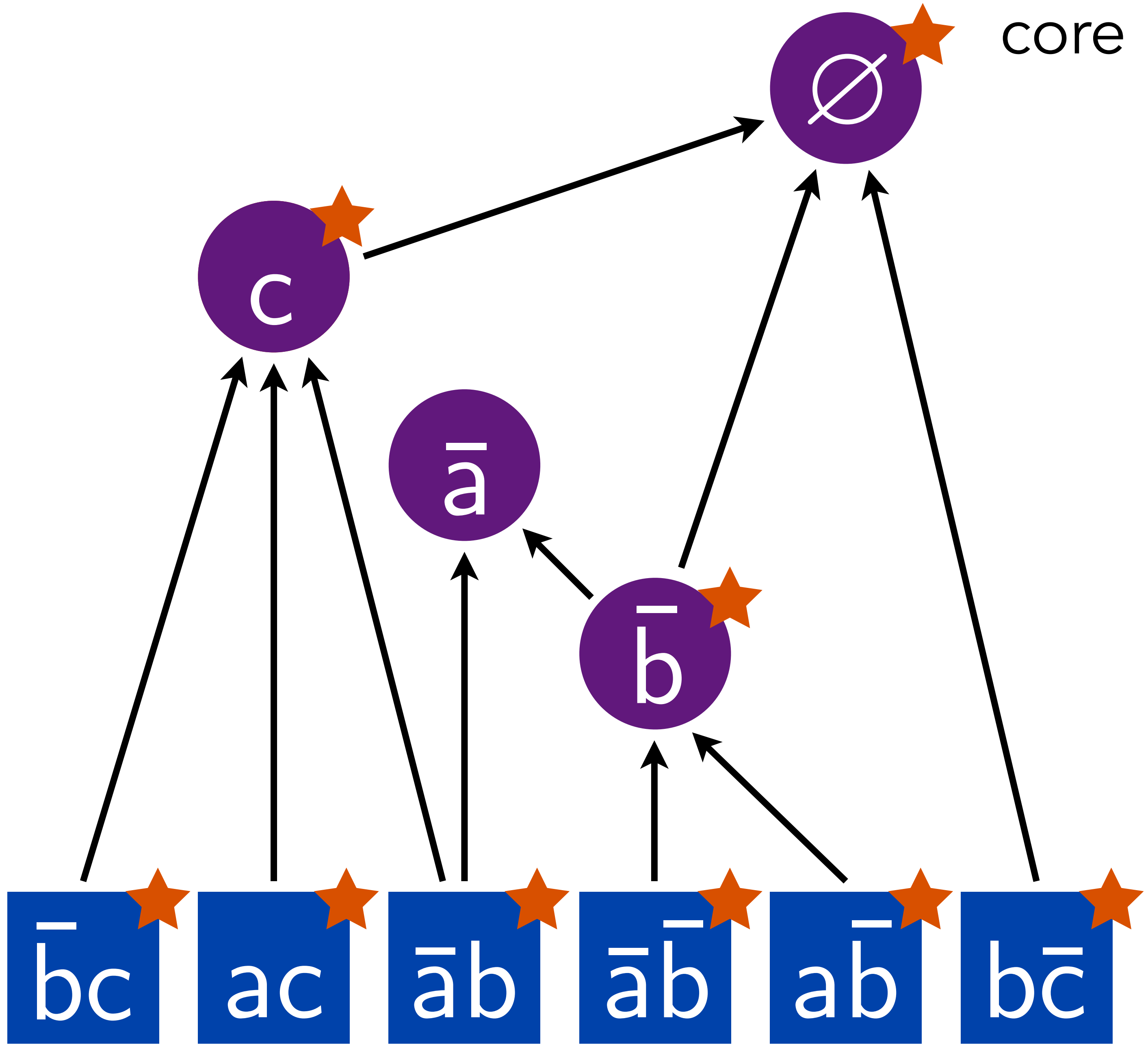


resolution graph

Resolution Graph / Proof and Core

$\bar{b}c$	
ac	
$\bar{a}b$	
$\bar{a}\bar{b}$	
$a\bar{b}$	
$b\bar{c}$	
$\bar{b}$	$\bar{a}\bar{b}$ $a\bar{b}$
$\bar{a}$	$\bar{a}b$ $\bar{b}$
c	ac $\bar{a}b$ $\bar{b}c$
$\emptyset$	$b\bar{c}$ $\bar{b}$ c

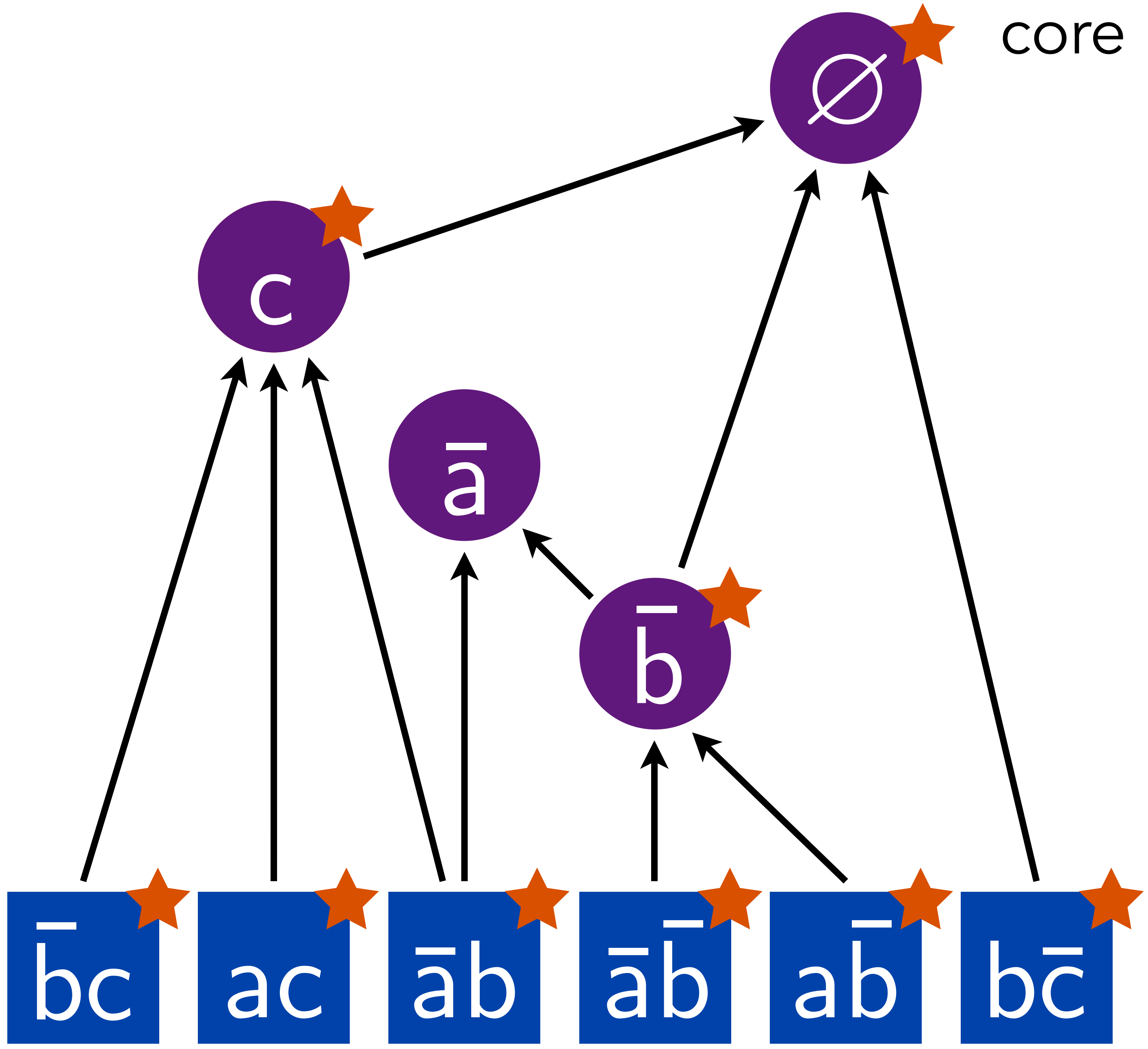
resolution proof



resolution graph

Resolution Graph / Proof and Core

$\bar{b}c$			
ac			
$\bar{a}b$			
$\bar{a}\bar{b}$			
$a\bar{b}$			
$b\bar{c}$			
$\bar{b}$	$\bar{a}\bar{b}$	$a\bar{b}$	
$\bar{a}$	$\bar{a}b$	$\bar{b}$	
c	ac	$\bar{a}b$	$\bar{b}c$
$\emptyset$	$b\bar{c}$	$\bar{b}$	c



resolution proofs are HUGE

Checking Lemmas by Unit Propagation

Checking Lemmas by Unit Propagation

A clause is **unit** with respect to an assignment if all literals in the clause are falsified except for one literal, which is unassigned.

Unit propagation:

- If a unit clause is found, extend the assignment and repeat.
- Else, return the assignment.

Checking Lemmas by Unit Propagation

A clause is **unit** with respect to an assignment if all literals in the clause are falsified except for one literal, which is unassigned.

Unit propagation:

- If a unit clause is found, extend the assignment and repeat.
- Else, return the assignment.

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

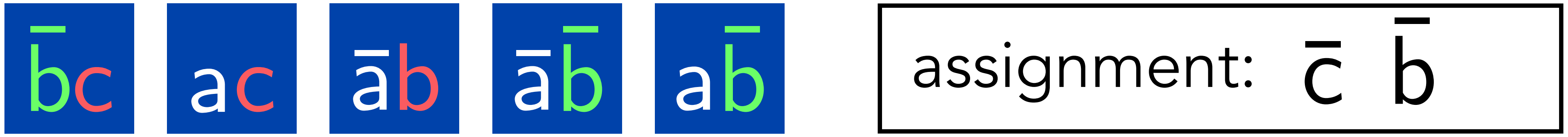
assignment: $\bar{c}$

Checking Lemmas by Unit Propagation

A clause is **unit** with respect to an assignment if all literals in the clause are falsified except for one literal, which is unassigned.

Unit propagation:

- If a unit clause is found, extend the assignment and repeat.
- Else, return the assignment.



$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

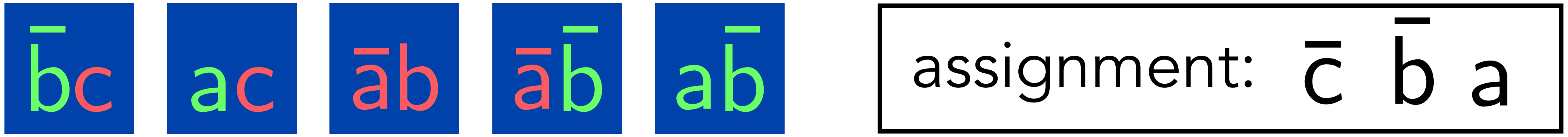
assignment: $\bar{c}$ $\bar{b}$

Checking Lemmas by Unit Propagation

A clause is **unit** with respect to an assignment if all literals in the clause are falsified except for one literal, which is unassigned.

Unit propagation:

- If a unit clause is found, extend the assignment and repeat.
- Else, return the assignment.



$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

assignment: $\bar{c}$ $\bar{b}$ a

Checking Lemmas by Unit Propagation

A clause is **unit** with respect to an assignment if all literals in the clause are falsified except for one literal, which is unassigned.

Unit propagation:

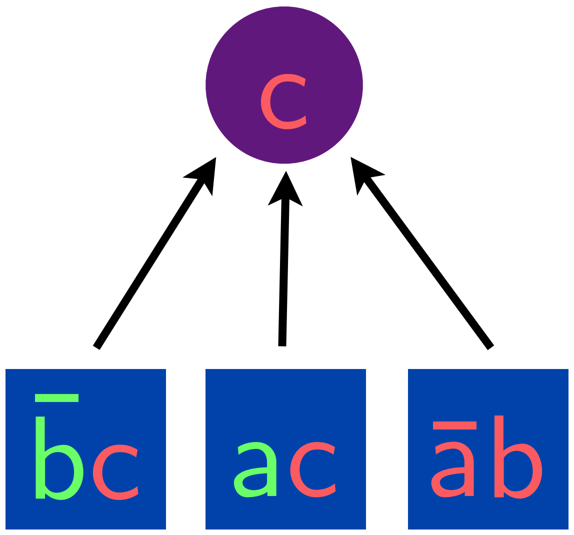
- If a unit clause is found, extend the assignment and repeat.
- Else, return the assignment.

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$

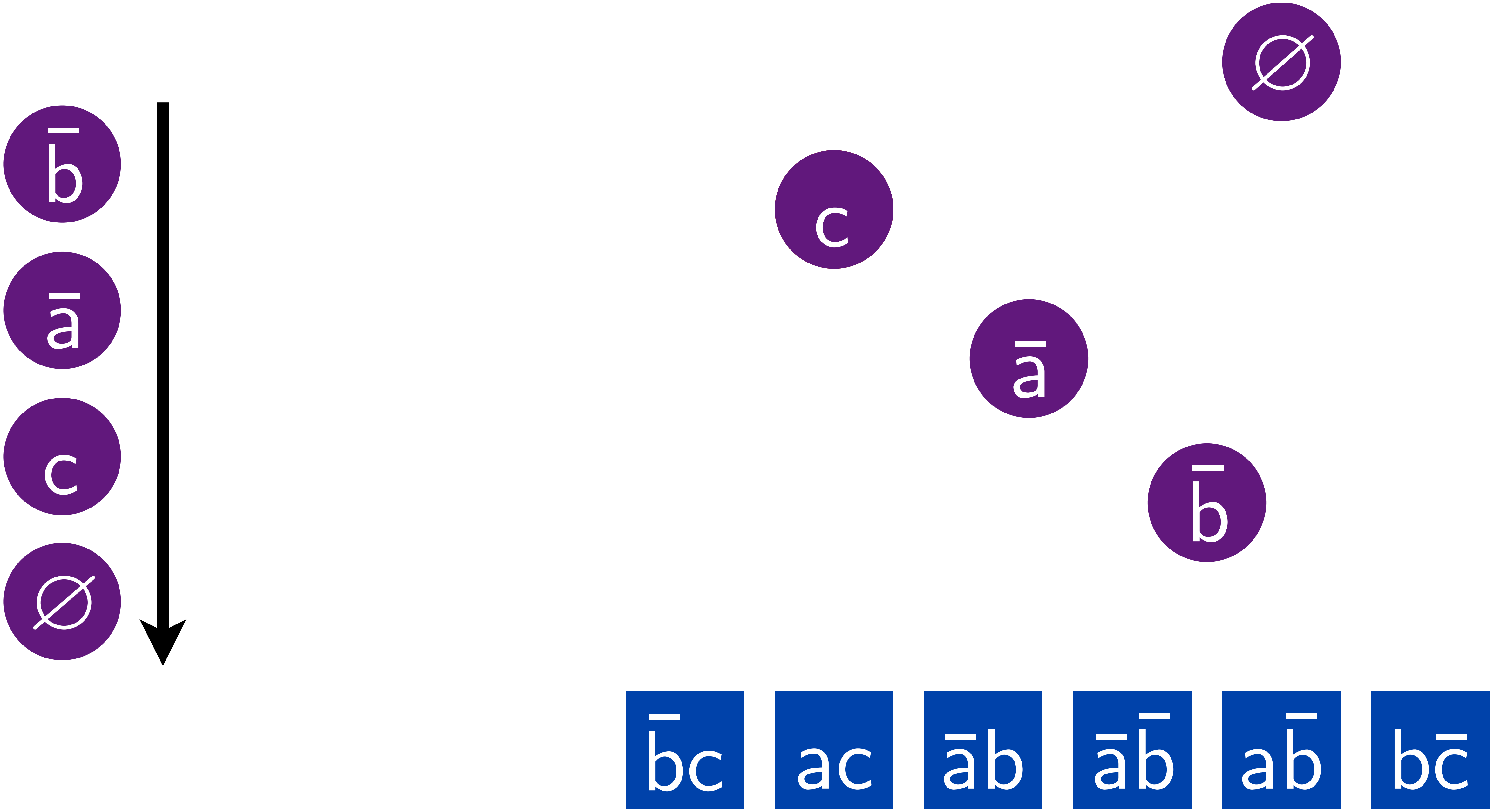
assignment: $\bar{c}$ $\bar{b}$ a

Reverse Unit Propagation (RUP) of a lemma:

- Assign all literals in the lemma to false and apply unit propagation
- If another clause / lemma becomes falsified, then the lemma is valid



Clausal Proof: Check using Unit Propagation



$\bar{b}$ $\bar{a}$ c $\emptyset$

$\emptyset$

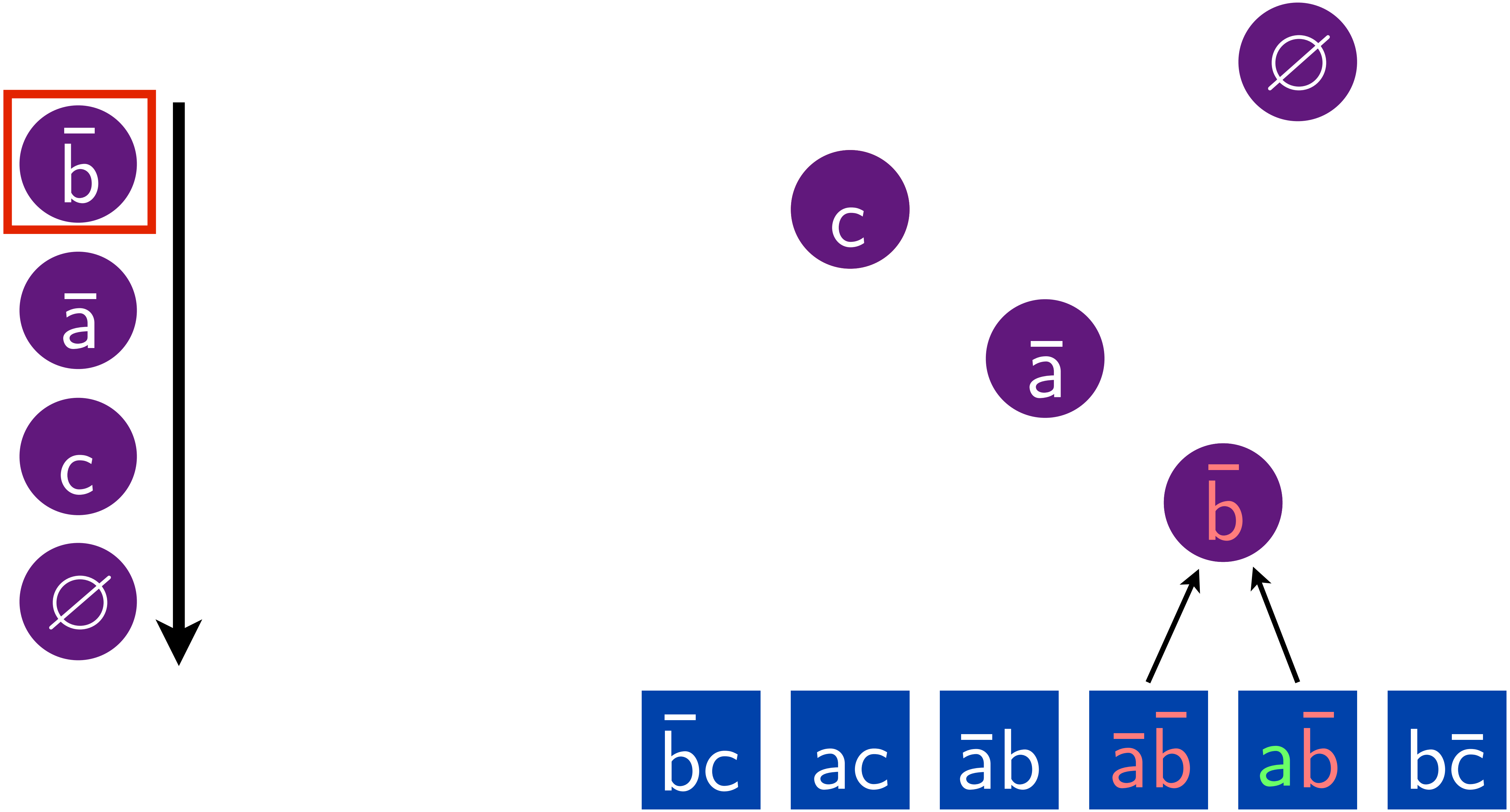
c

$\bar{a}$

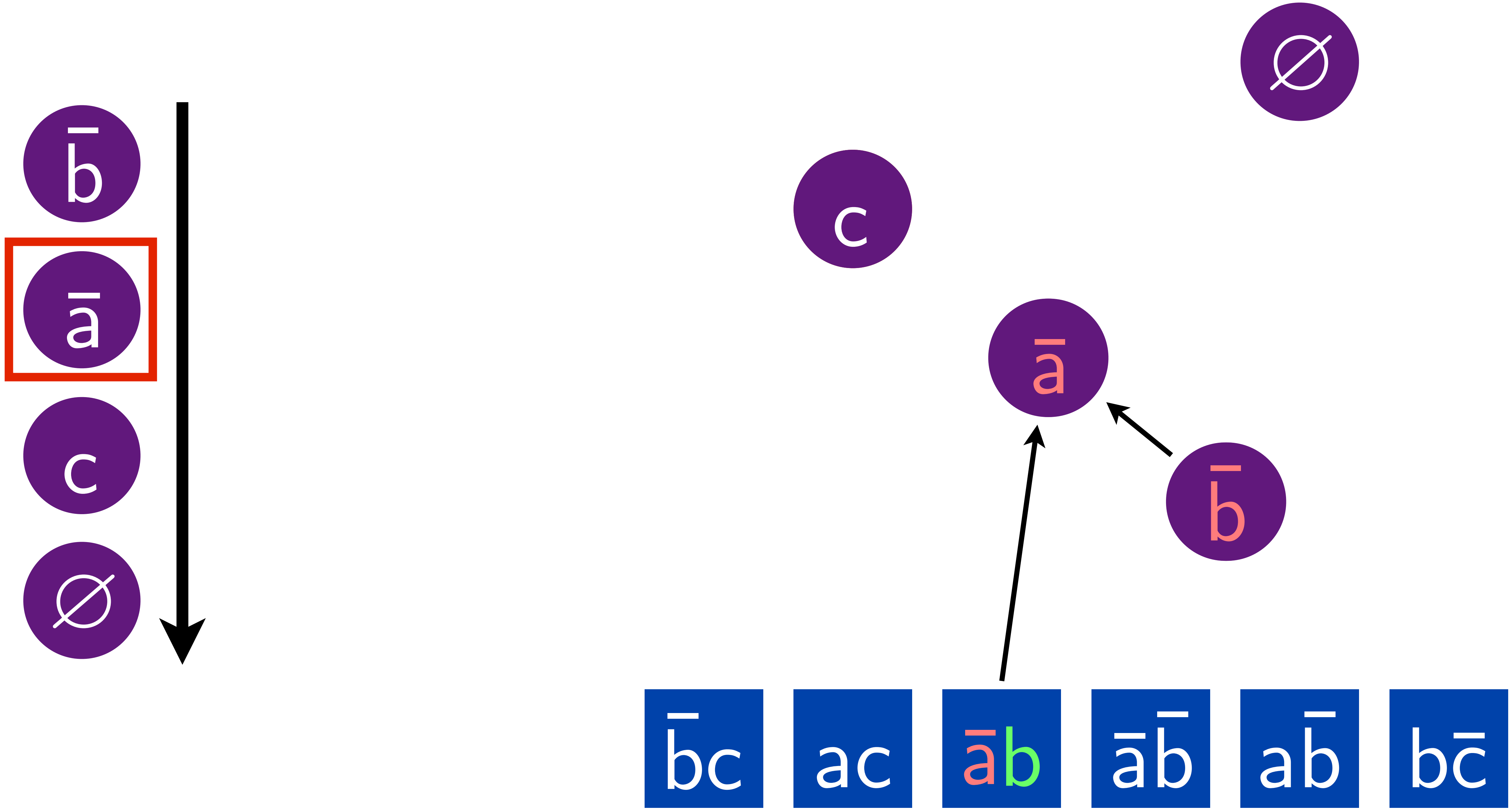
$\bar{b}$

$\bar{b}c$ ac $\bar{a}b$ $\bar{a}\bar{b}$ $a\bar{b}$ $b\bar{c}$

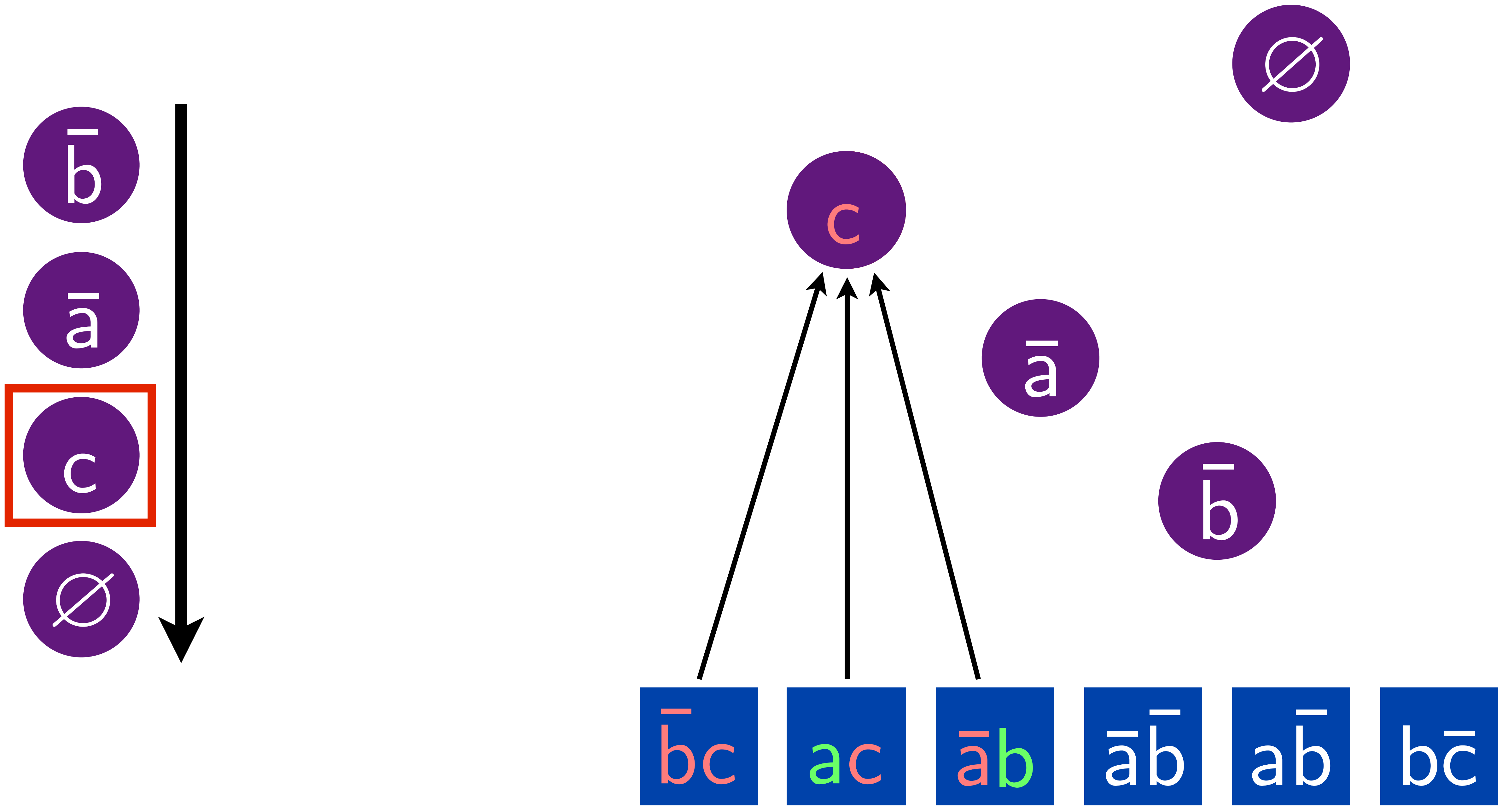
Clausal Proof: Check using Unit Propagation



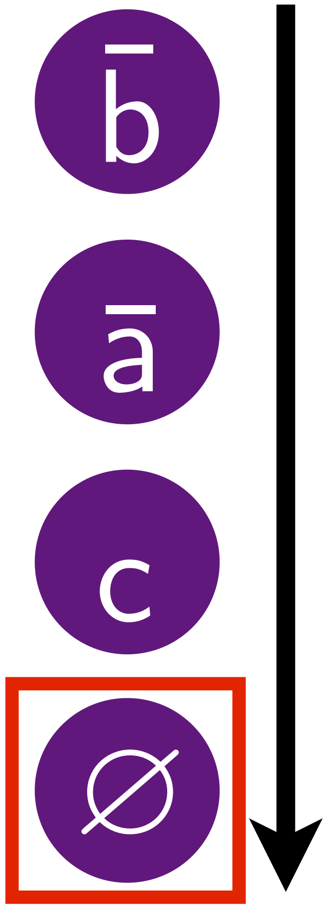
Clausal Proof: Check using Unit Propagation

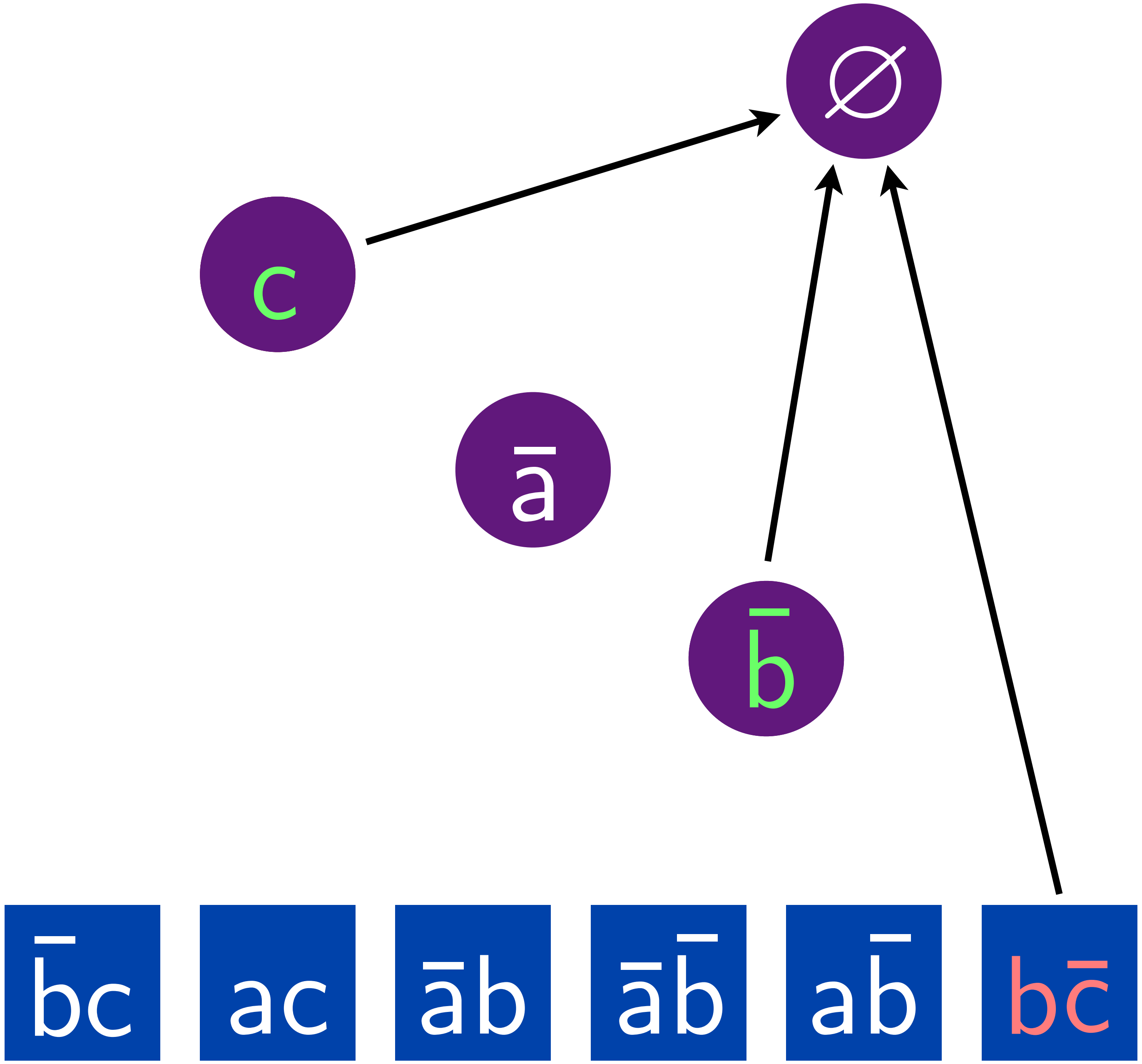


Clausal Proof: Check using Unit Propagation

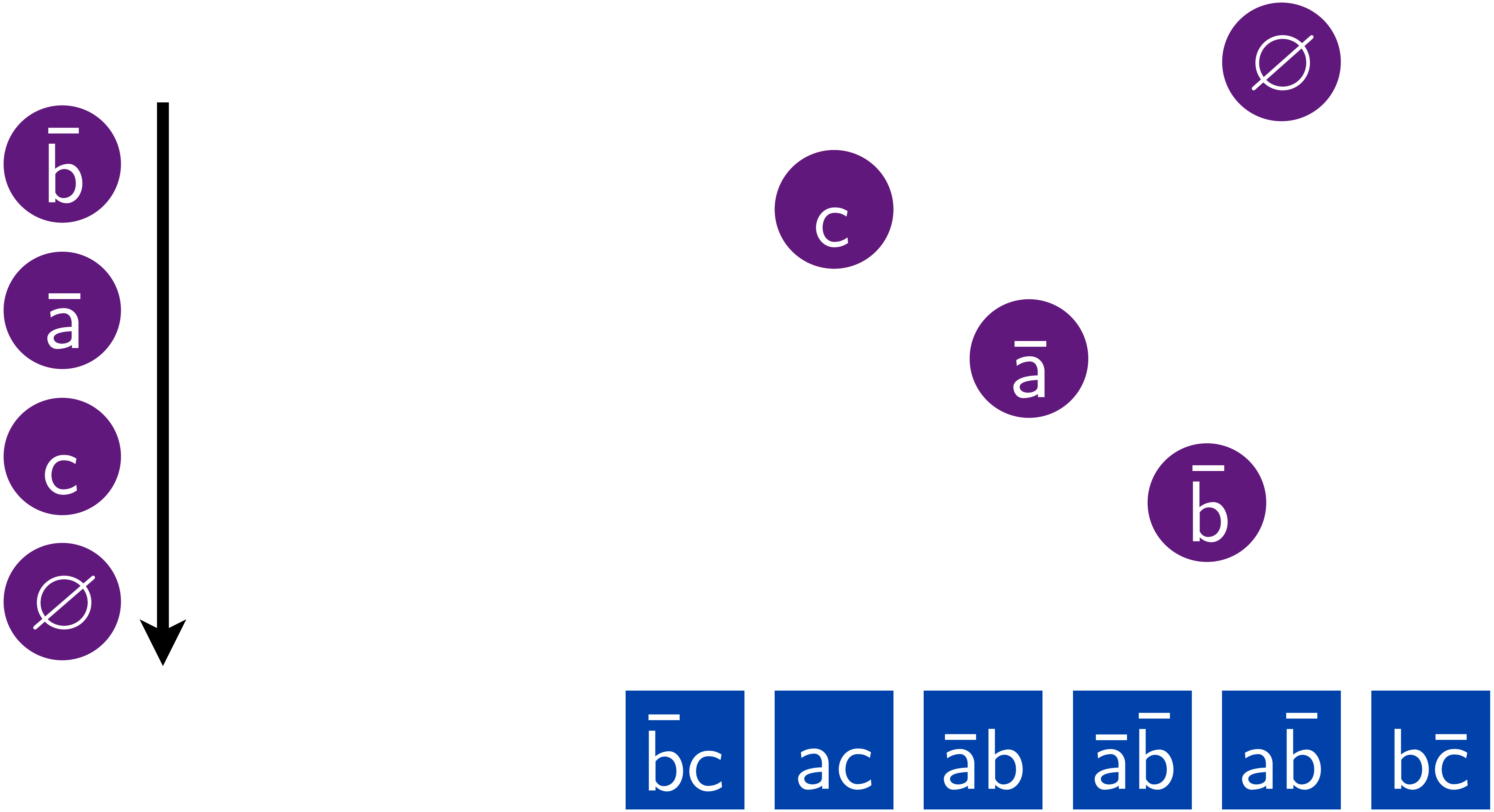


Clausal Proof: Check using Unit Propagation





Clausal Proof: Check using Unit Propagation



clausal proofs are expensive to validate

Improvement I: Backwards Checking

Goldberg and Novikov proposed checking the refutation backwards [DATE 2003]:

- start by validating the empty clause;

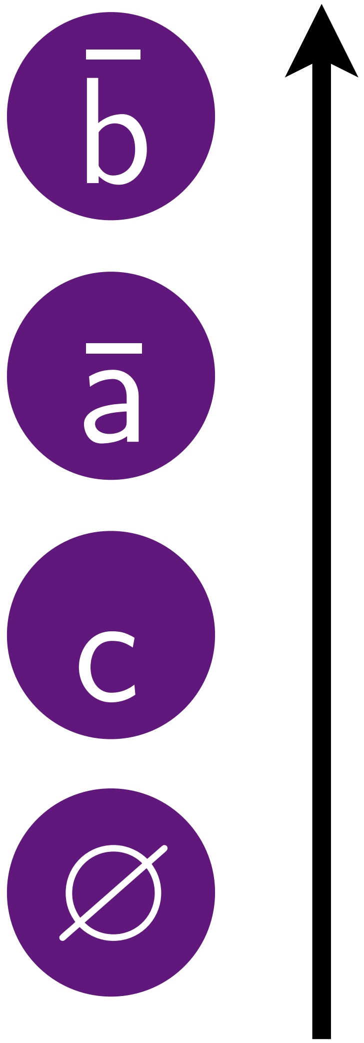
- mark all lemmas using conflict analysis;

- only validate marked lemmas.

Advantage: validate fewer lemmas.

Disadvantage: more complex.

We provide a fast open source implementation of this procedure.



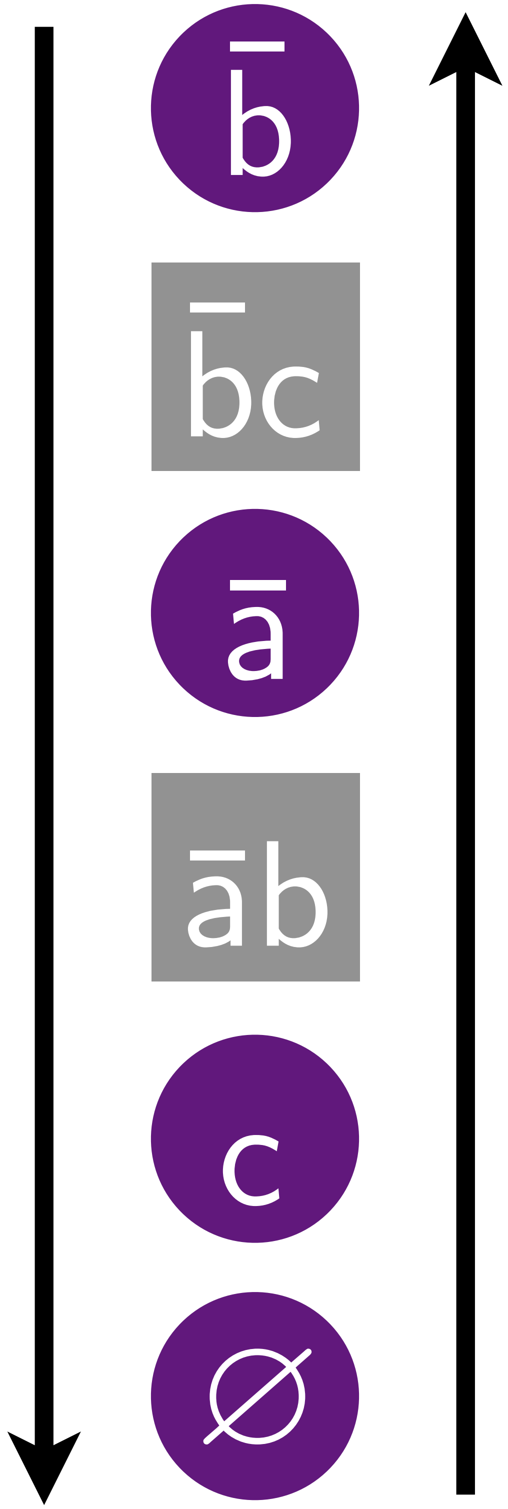
Improvement II: Clause Deletion

We proposed to extend clausal proofs with deletion information [STVR 201X]:

- clause deletion is crucial for efficient solving;
- emit learning and deletion information;
- proof size might double;
- checking speed can be reduced significantly.

Clause deletion can be combined with backwards checking [FMCAD 2013]:

- ignore deleted clauses earlier in the proof;
- optimize clause deletion for trimmed proofs.



Improvement III: Core-first Unit Propagation

We propose a new unit propagation variant:

1) propagate using clauses already in the core;

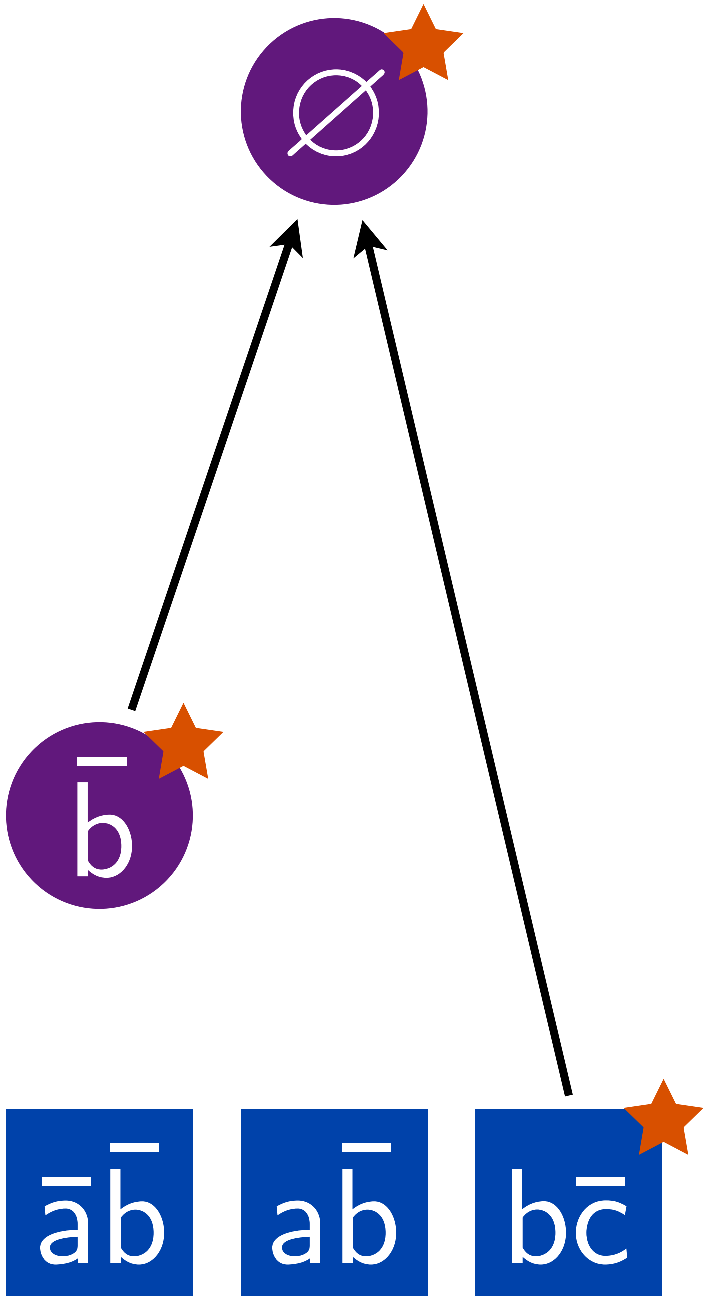
2) examine non-core clauses only at fixpoint;

3) if a non-core unit clause is found, goto 1);

4) otherwise terminate.

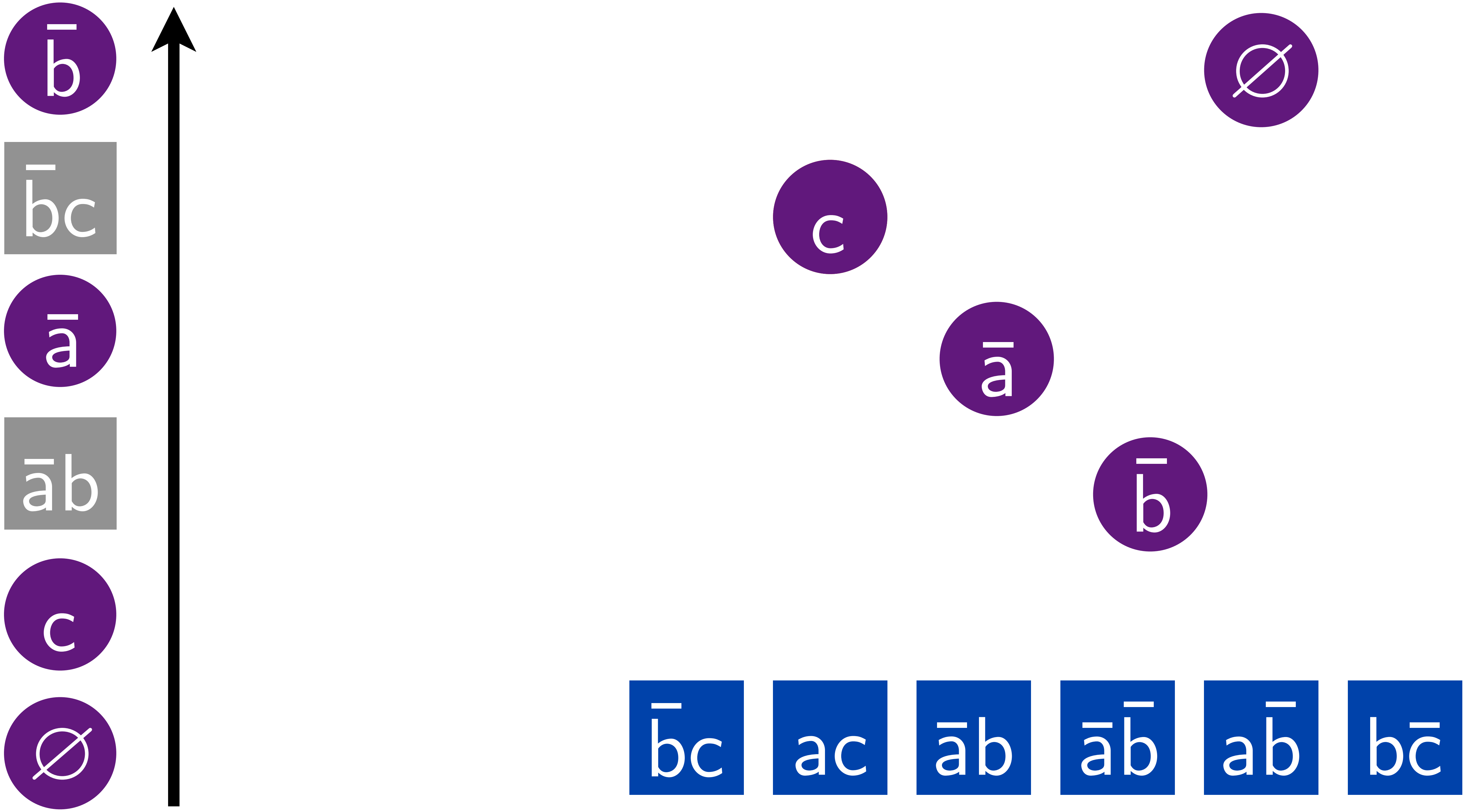
Our variant, called *Core-first Unit Propagation*, can reduce checking costs considerably.

Fast propagation in a checker is different than fast propagation in a SAT solver.

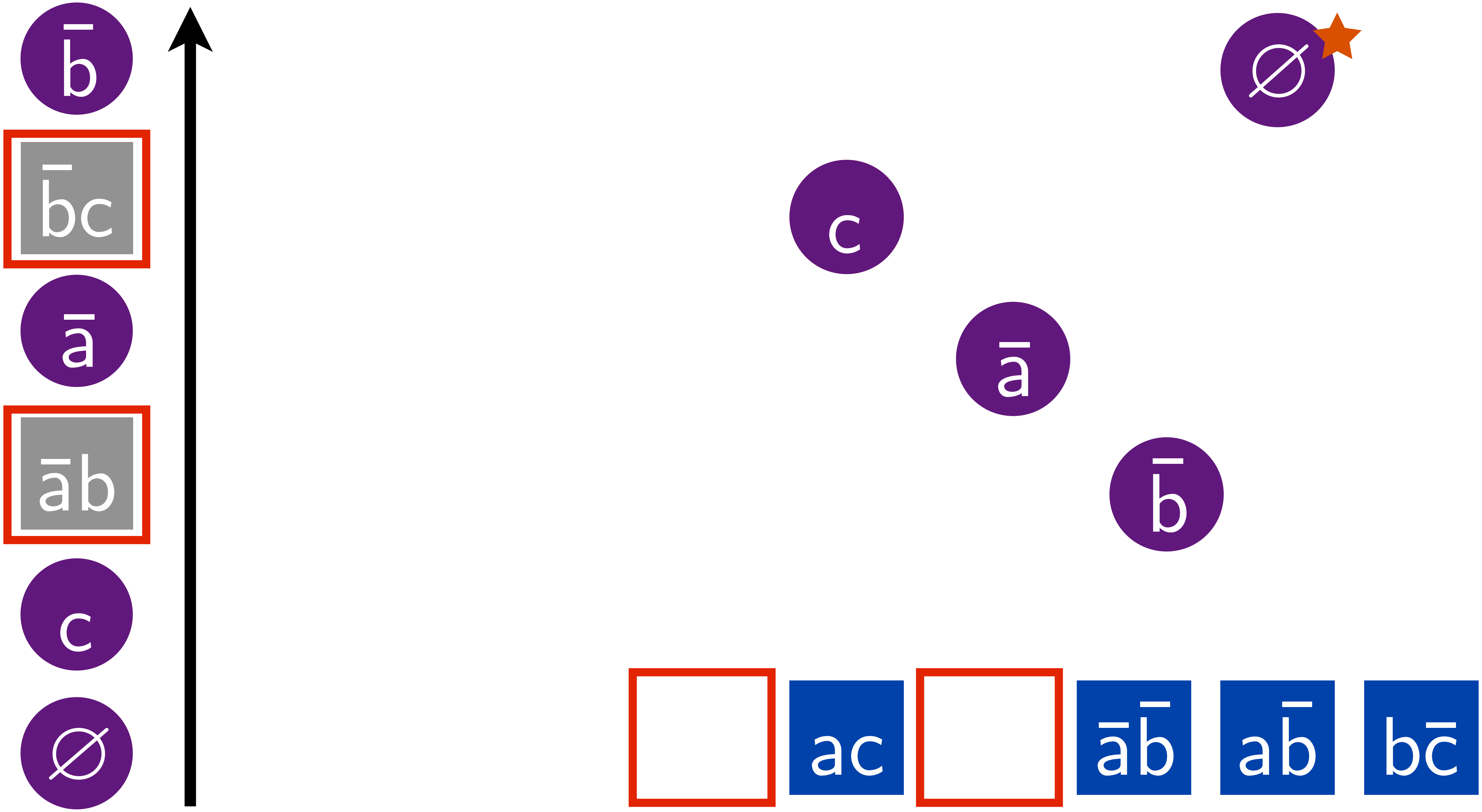


Also, the resulting core and proof are smaller

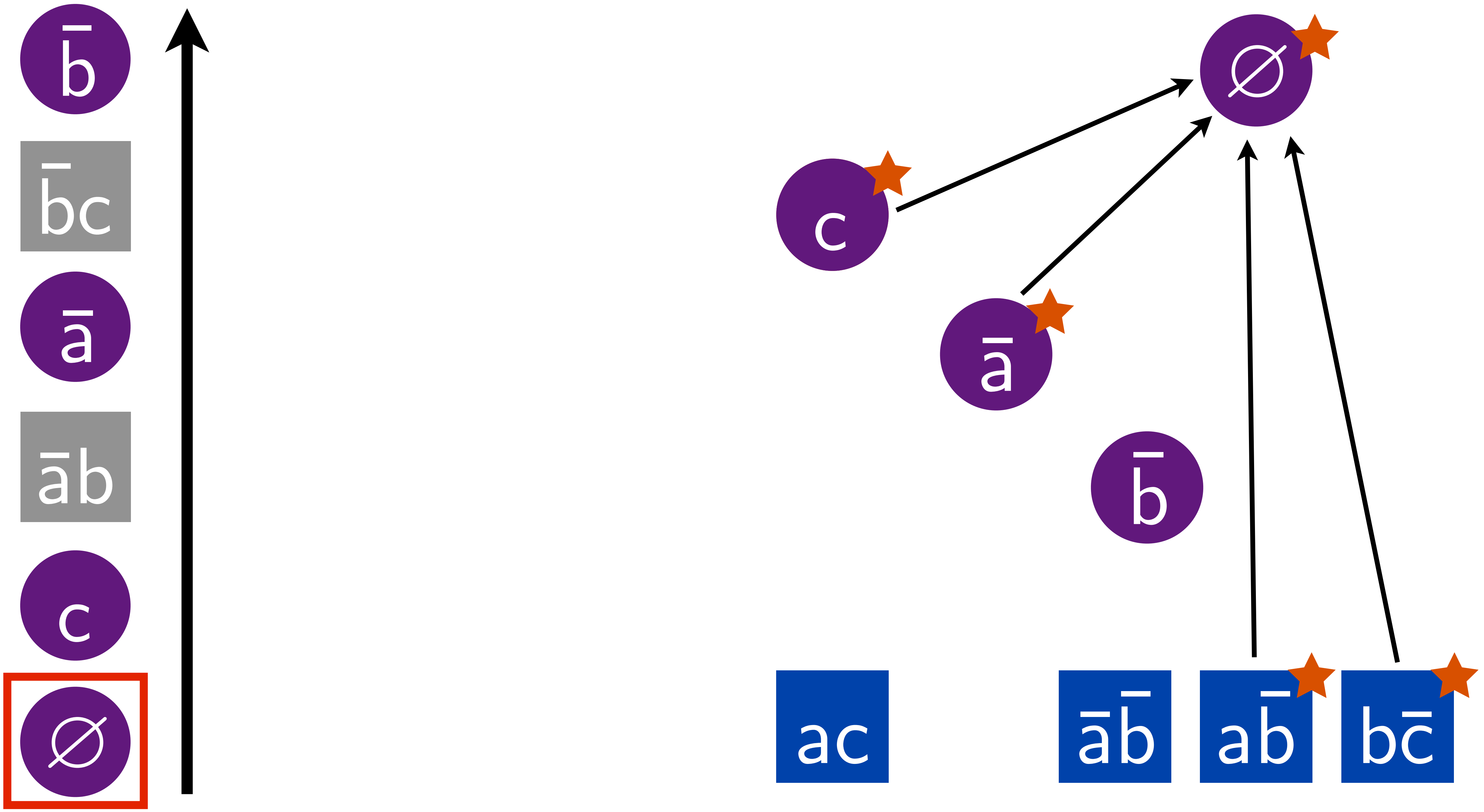
Checking: Backwards + Core-first + Deletion



Checking: Backwards + Core-first + Deletion

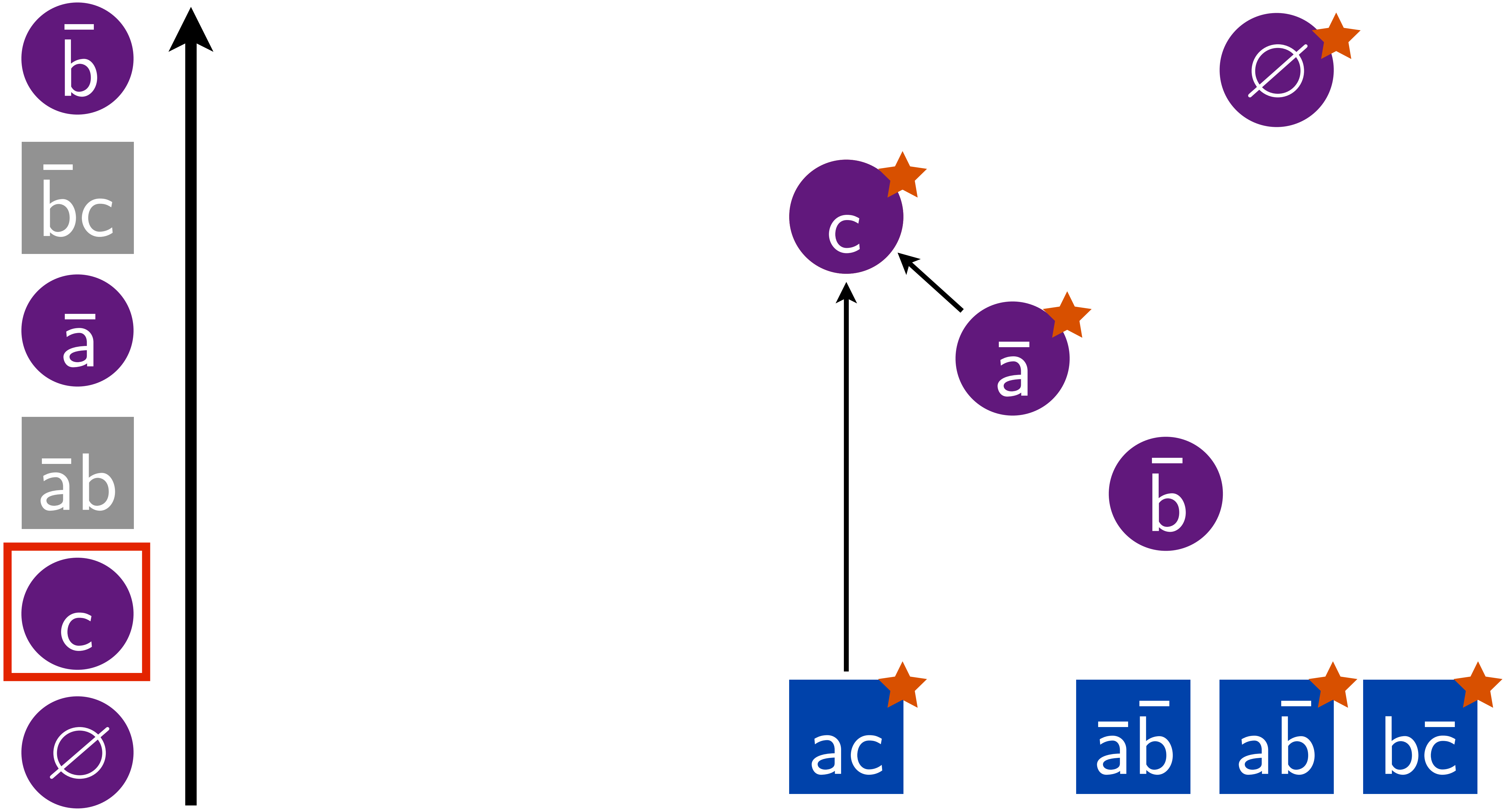


Checking: Backwards + Core-first + Deletion

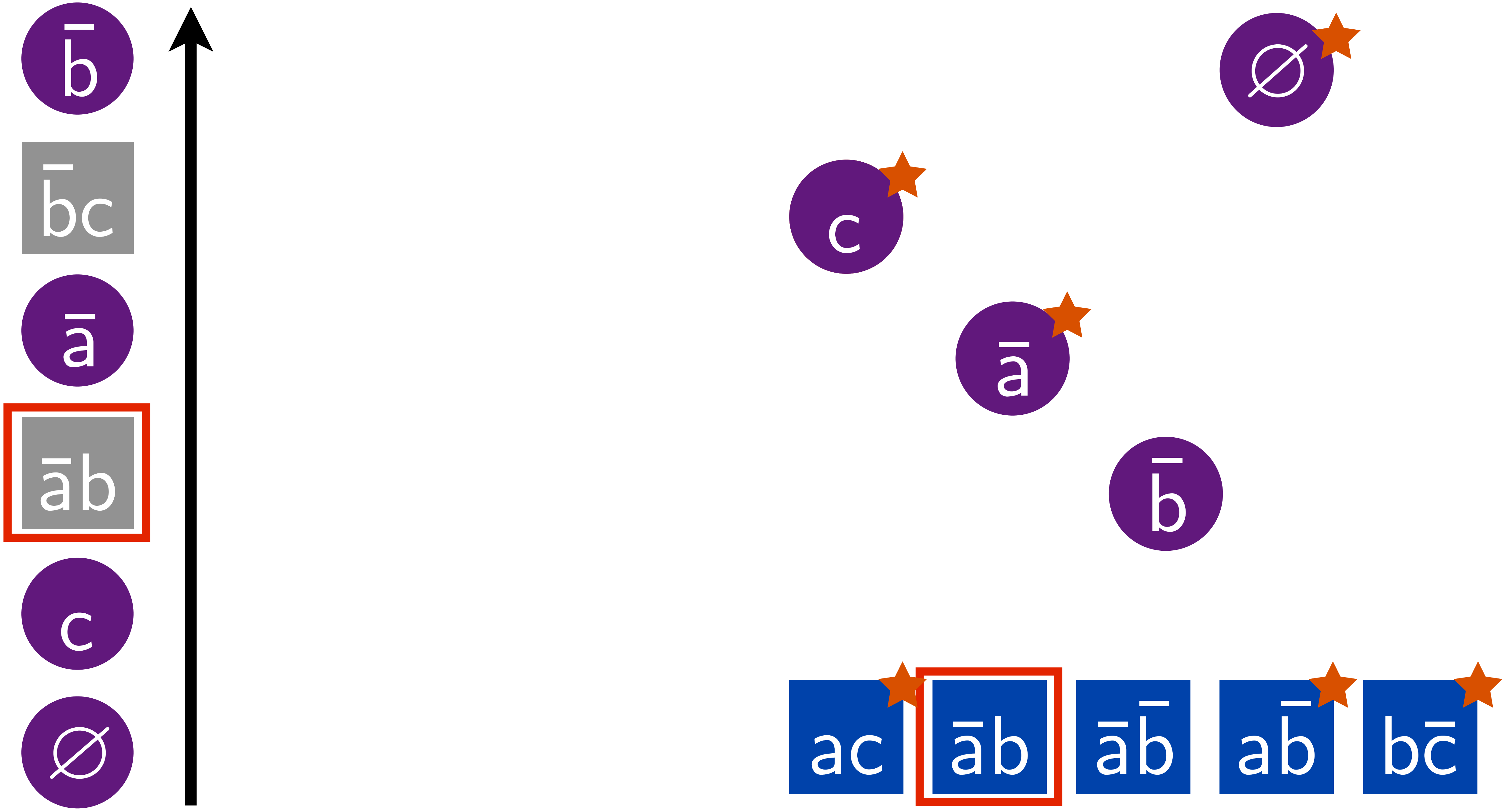


Wednesday, October 23, 13

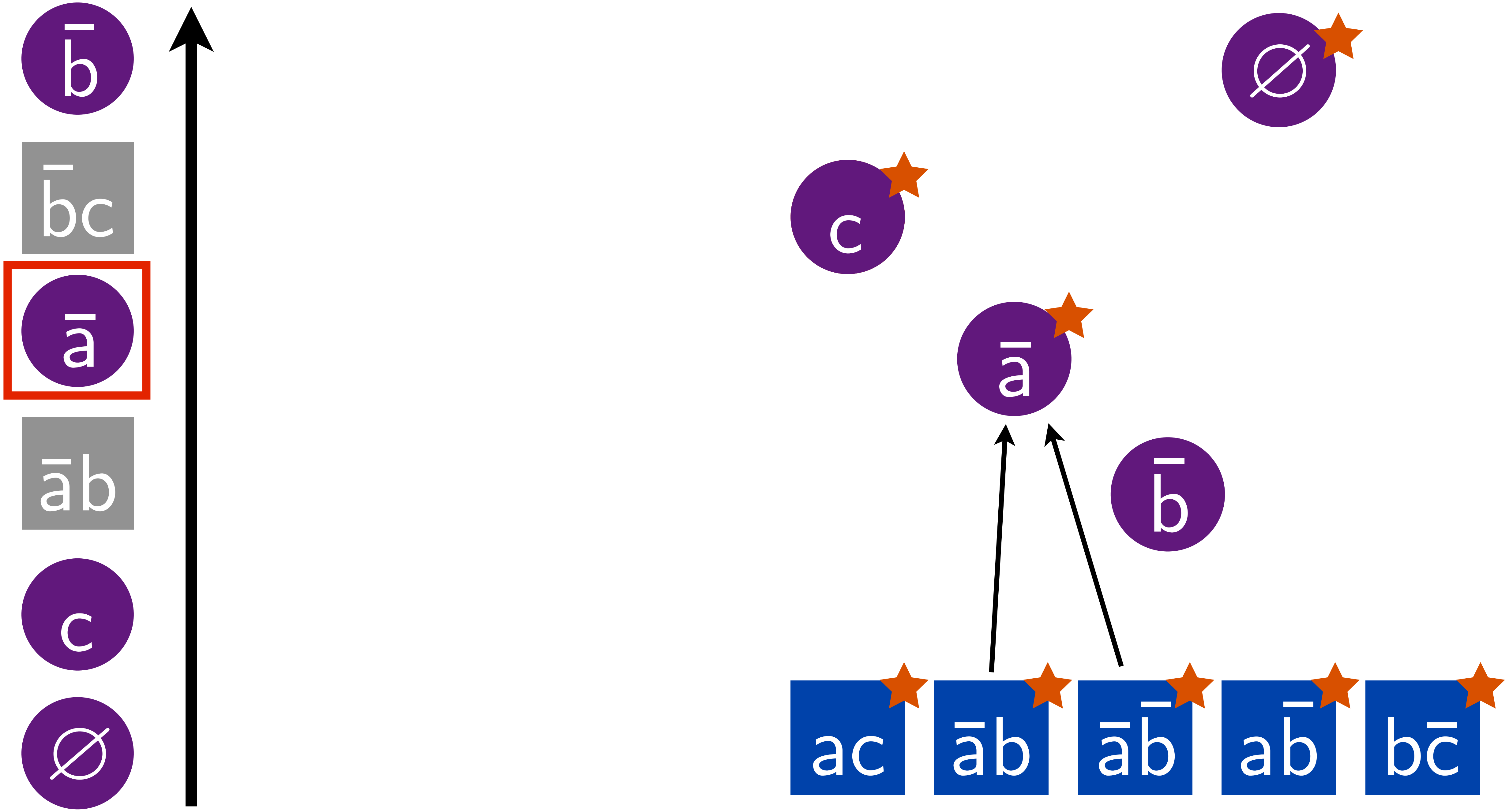
Checking: Backwards + Core-first + Deletion



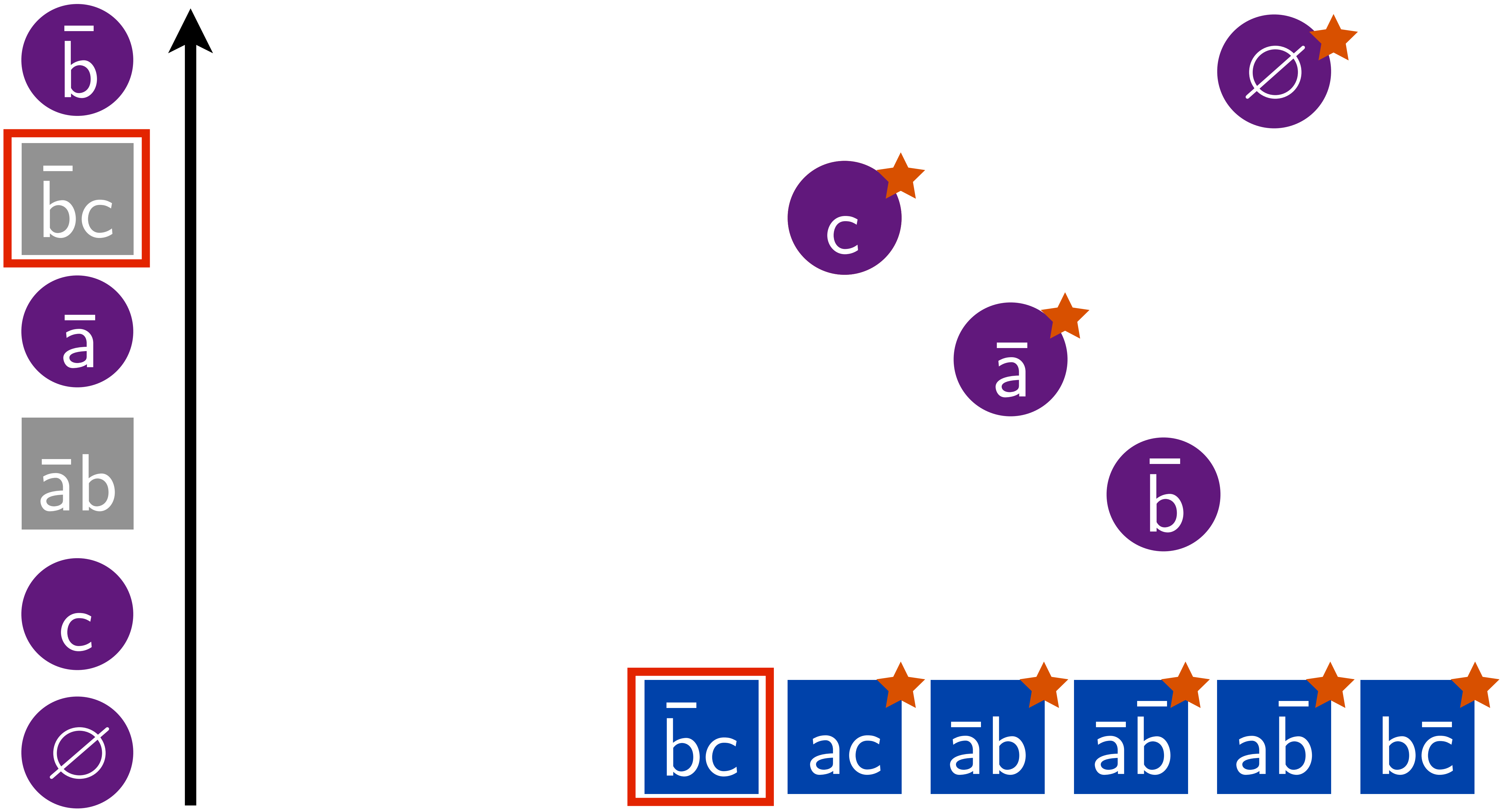
Checking: Backwards + Core-first + Deletion



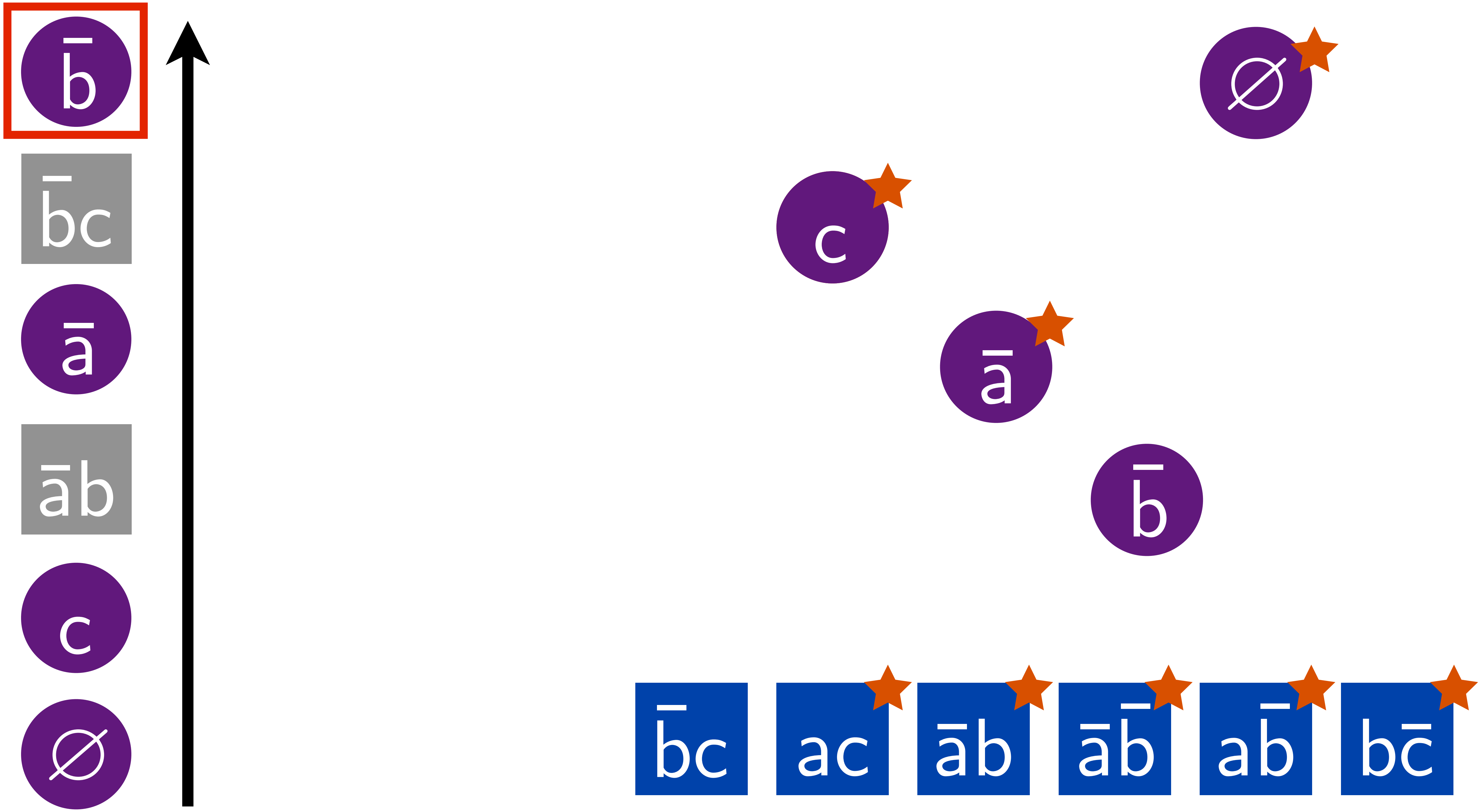
Checking: Backwards + Core-first + Deletion



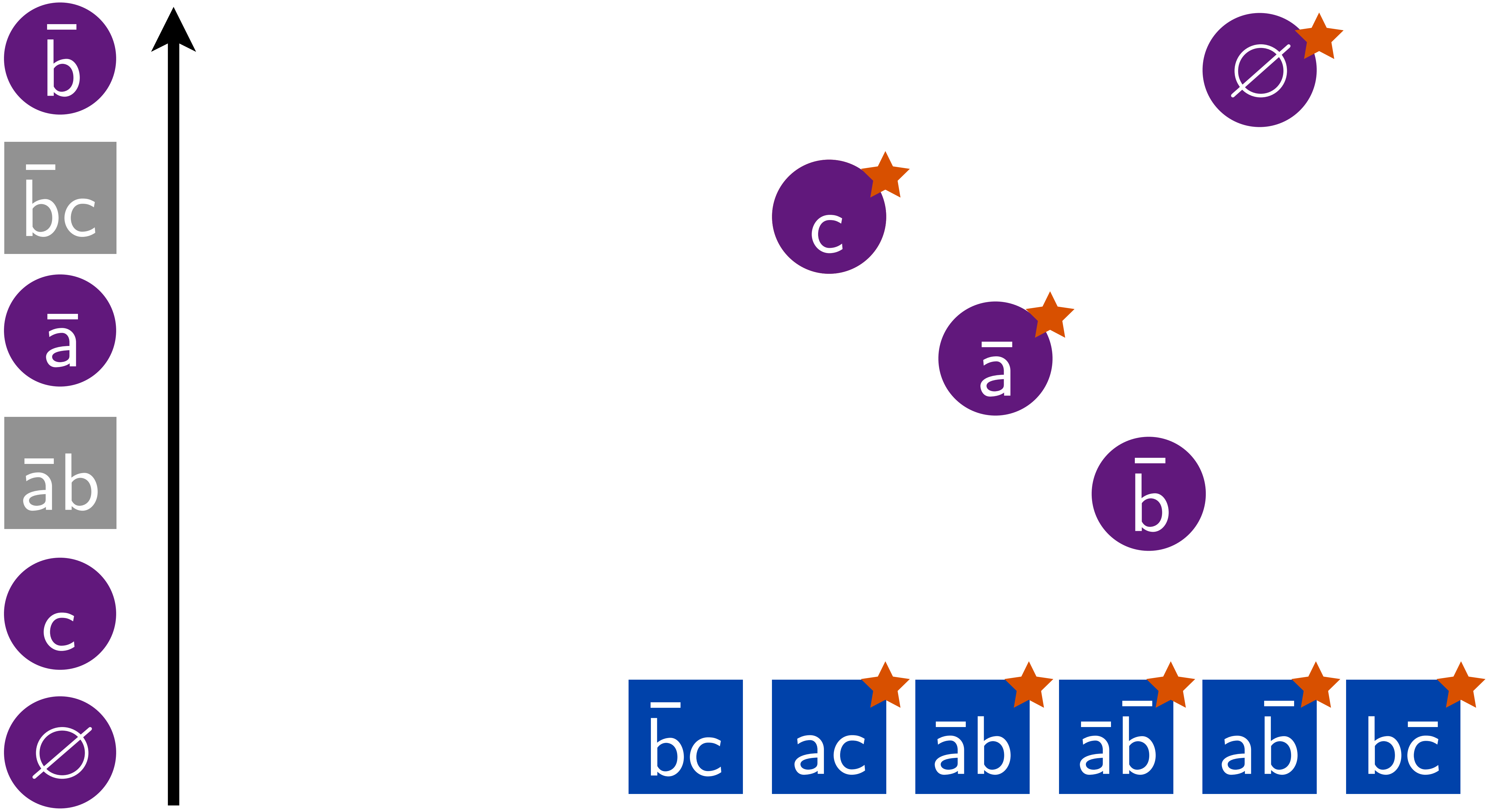
Checking: Backwards + Core-first + Deletion



Checking: Backwards + Core-first + Deletion



Checking: Backwards + Core-first + Deletion



Core-first unit propagation results in smaller cores and proofs

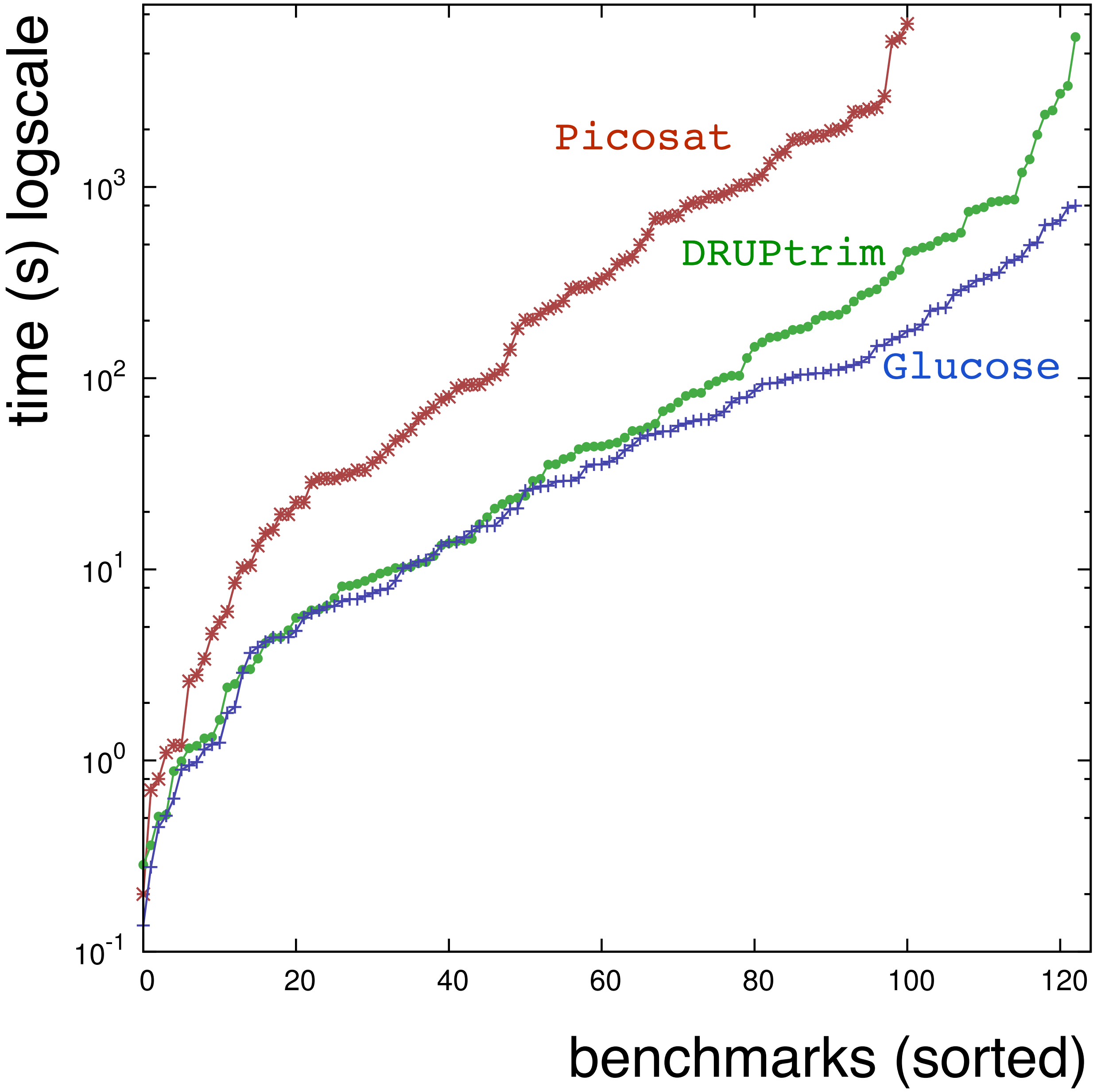
Experimental Evaluation

We implemented DRUP logging into Glucose using only 40 LoC.

Glucose (DRUP) solves about twice as many benchmarks as compared to Picosat (resolution).

Resolution proof logging increased memory usage up to a factor 100.

DRUPtrim validated clausal proofs in a time similar to the solving time.



DRUPtrim in SAT Competition 2013

Unsatisfiable tracks required certificates. Allowed formats:

- TraceCheck (resolution);
- DRUP, Delete Reverse Unit Propagation.

Timeout : 5,000 seconds for solving and 20,000 seconds for checking

Submissions with proof logging:

- 11 application solvers (9 DRUP, 2 RUP);
- 9 hard-combinatorial solvers (7 DRUP, 2 RUP);
- Most submissions were certified unsatisfiable versions of top-tier solvers.

Statistics:

- 98% of DRUP proofs of top-tier solvers were checked within the time limit;
- Checking most RUP proofs (i.e., no clause deletion) results in a timeout.

Conclusion

Our `DRUPtrim` tool:

- makes it feasible to check the results of state-of-the-art solvers efficiently (demonstrated at SAT Competition 2013);
- validates learning, preprocessing, and inprocessing techniques;
- and produces trimmed proofs and trimmed formulas.

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

Discussion: should UNSAT proof logging be mandatory for tools participating in competitive events (e.g., SAT Competition, HWMCC)?

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

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

Thank you for your attention! **Questions?**

Resolution Graphs: Arcs vs Vertices vs Literals

Resolution graphs are huge

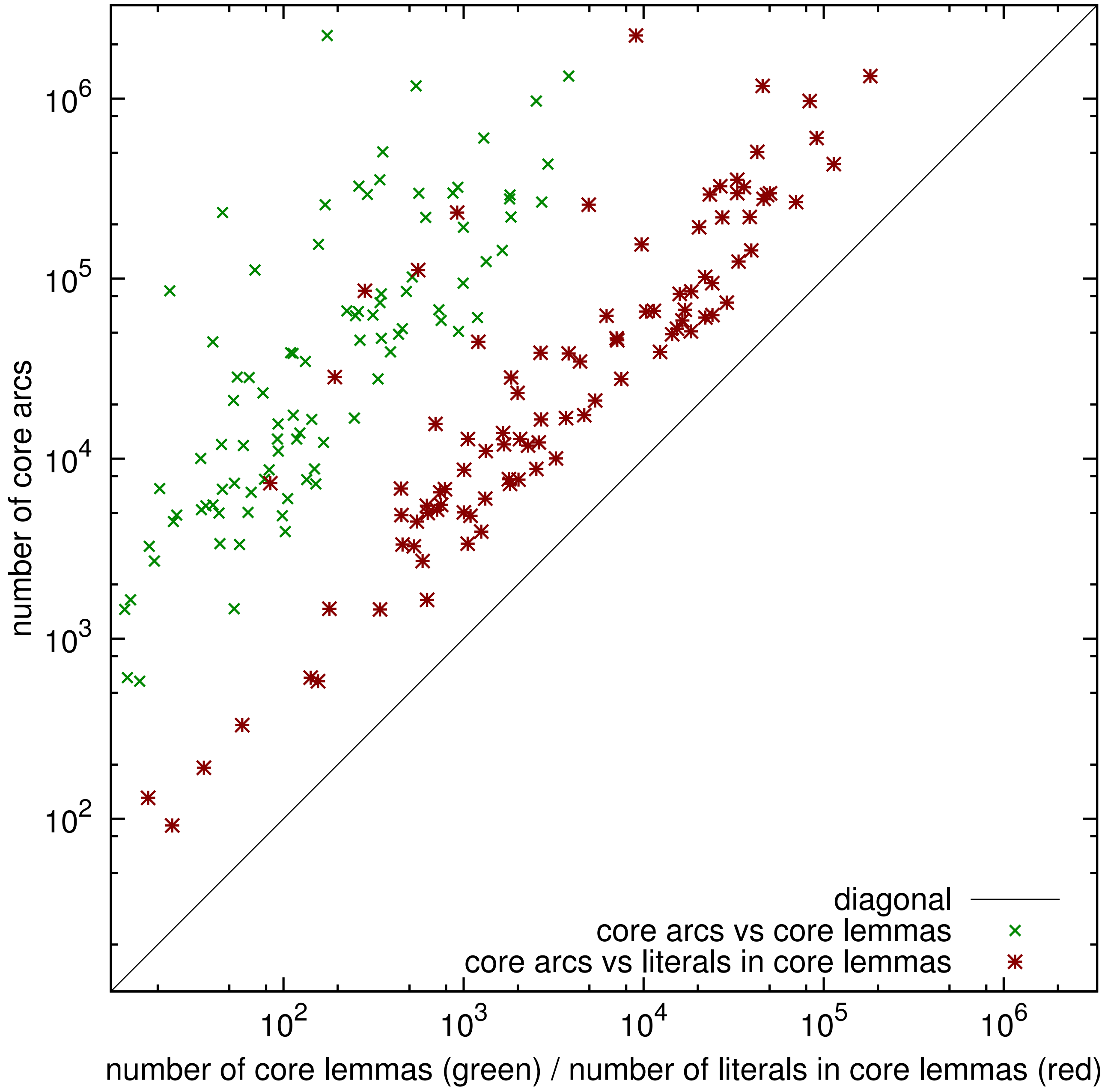
- plot obtained using `picosat`

Lemmas require ~400 resolution steps (arcs in graph)

- due to clause minimization

Lemmas have ~40 literals

Resolution proofs are at least 10x larger than clausal proofs, up to 100x memory footprint !



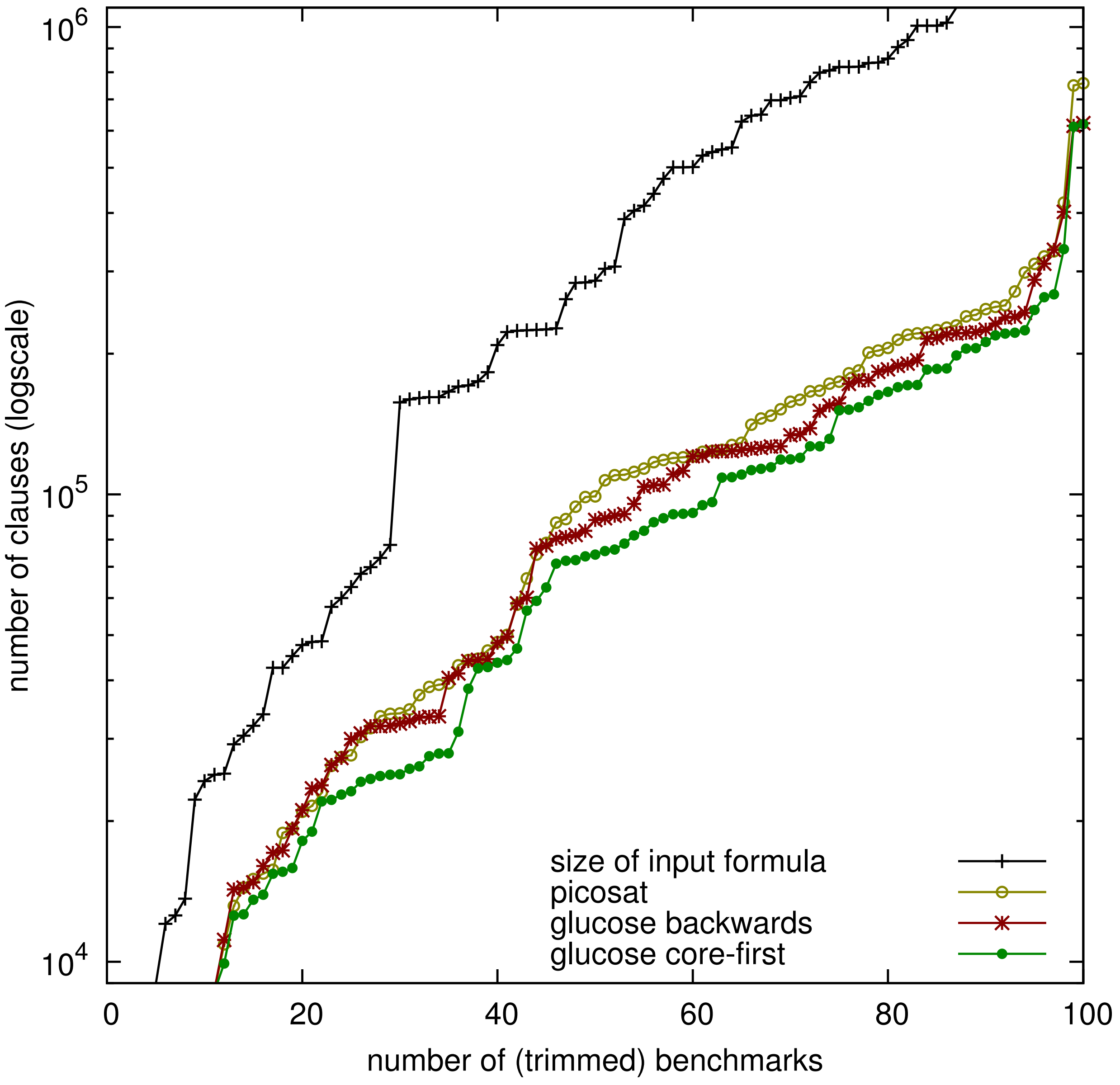
Results: Trimming

The number of core clauses using

- `Picosat` (resolution + no preprocessing)
- `Glucose` (backwards + preprocessing)
- `Glucose` (core-first + preprocessing)

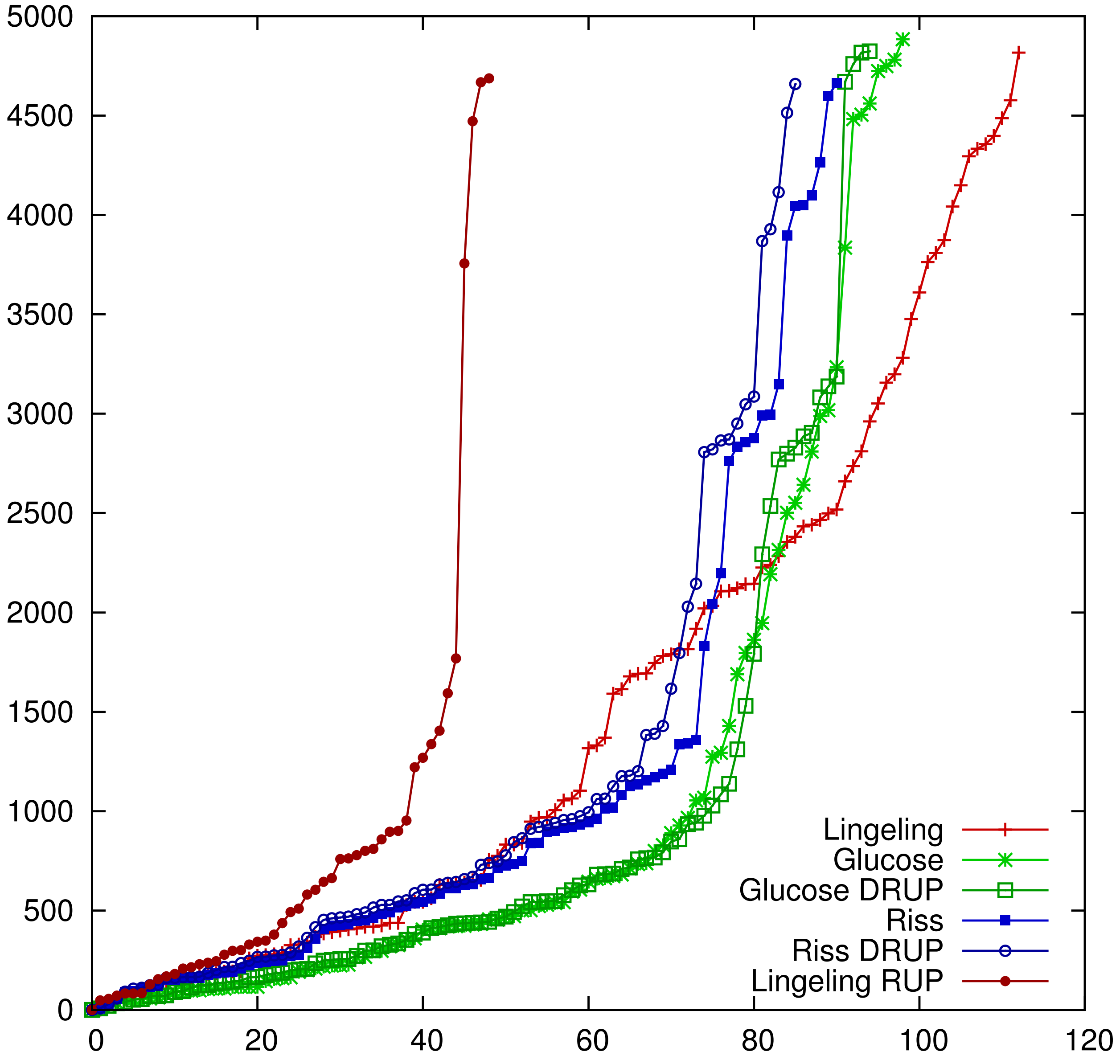
Checking clausal proofs results in smaller trimmed formulas

The core-first unit propagation technique further trims the formula

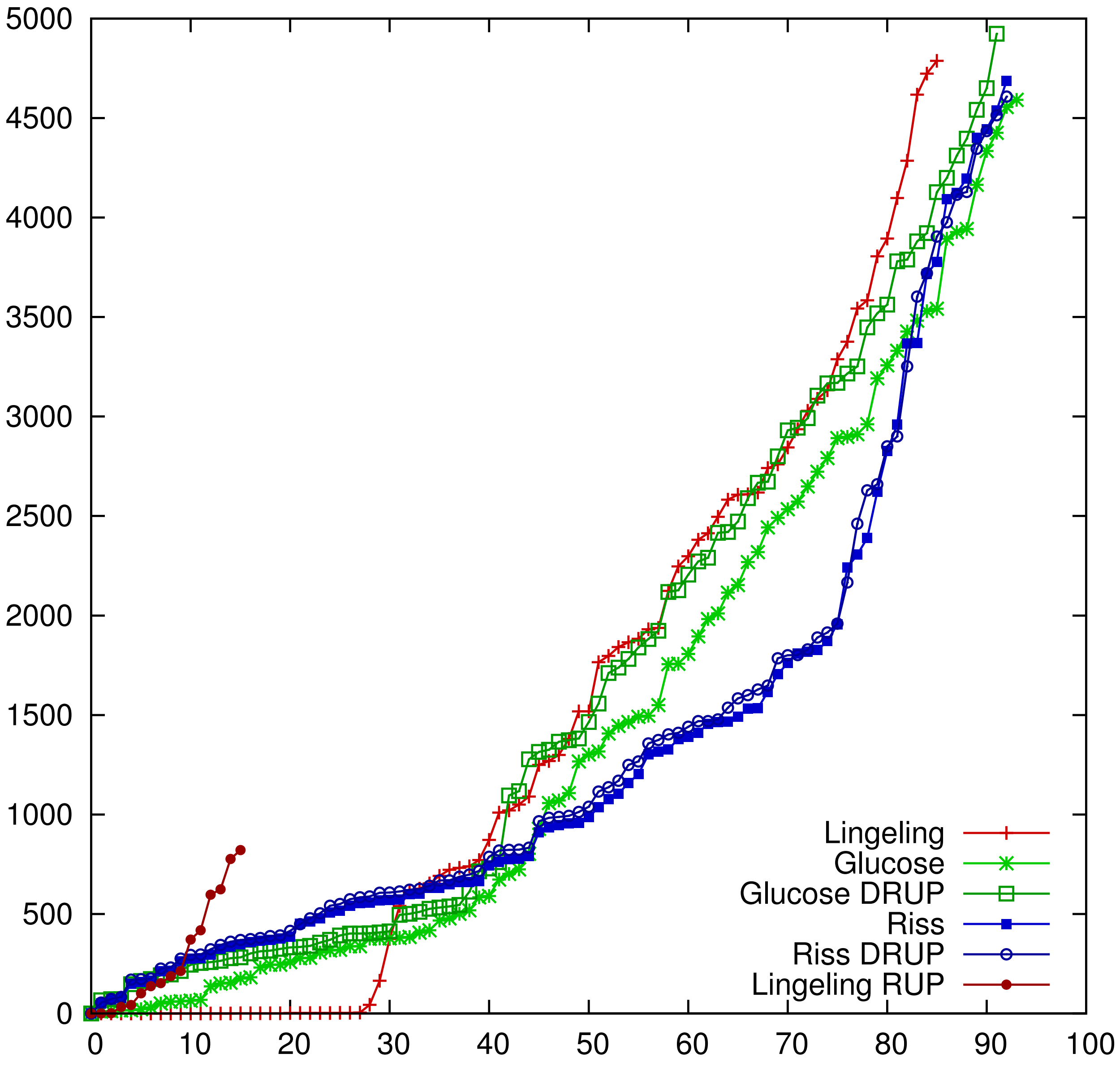


Proofs Plots for SAT Competition 2013

Application benchmarks



Hard-combinatorial benchmarks



NB: a solved benchmark was only counted if the output was verified

UNSAT Results of SAT Competition 2013

DRUP proofs can be checked in a time similar to the solving time

Lack of deletion information made checking much more costly

Enabling DRUP support has a small effect on the solving time

Big performance differences were due to bugs

- some "features" were turned off by enabling DRUP support
- two buggy solvers were not disqualified, due to "correct results"

Our DRUP-trim tool made it feasible to validate all DRUP results