Numeric Bounds Analysis with Conflict-Driven Learning

Vijay D'Silva, Leopold Haller,
Daniel Kroening, Michael Tautschnig

UNIVERSITY OF OXFORD

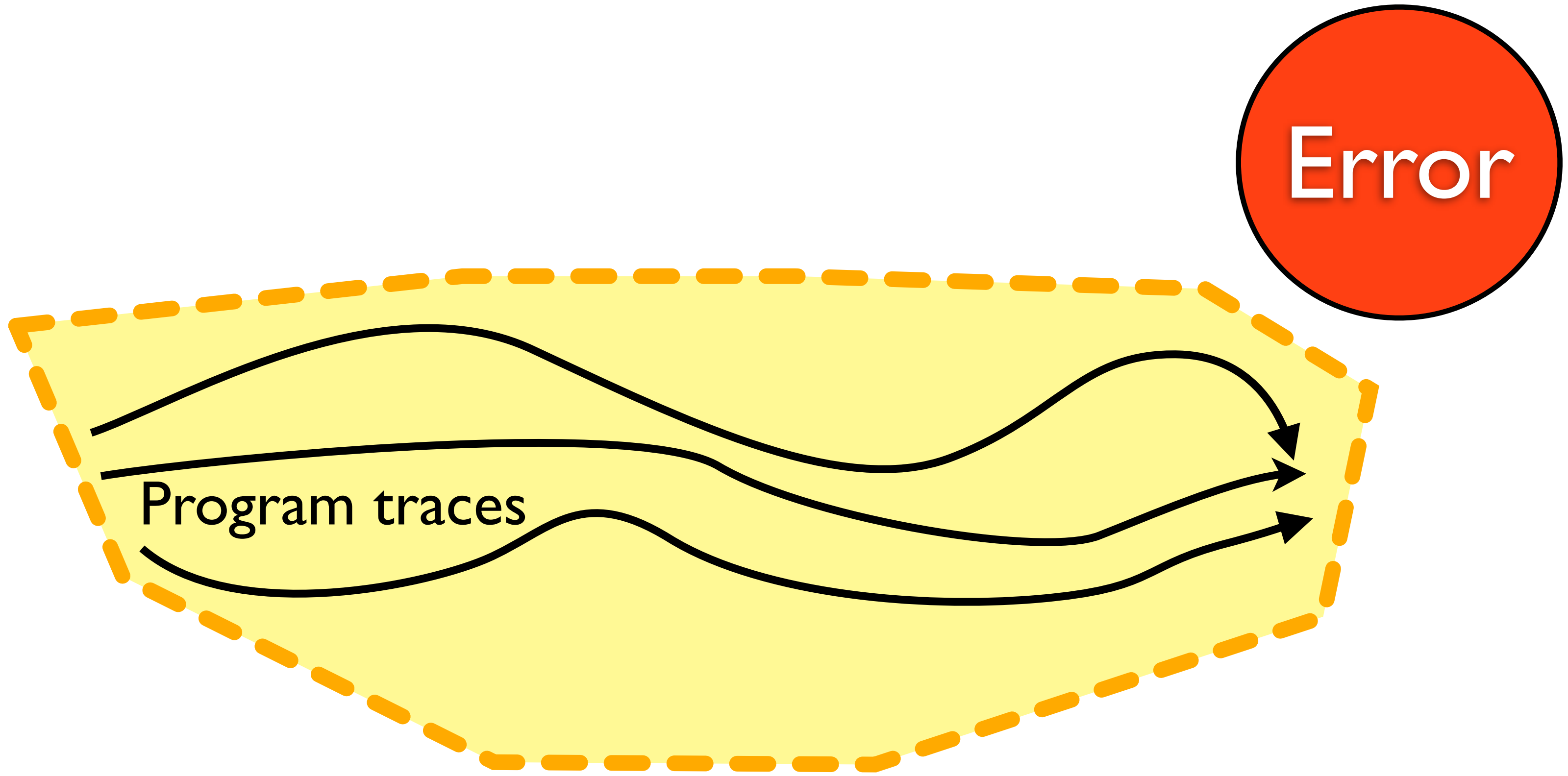
TACAS 2012

PRECISION VS EFFICIENCY

Static Analysis vs Decision Procedures

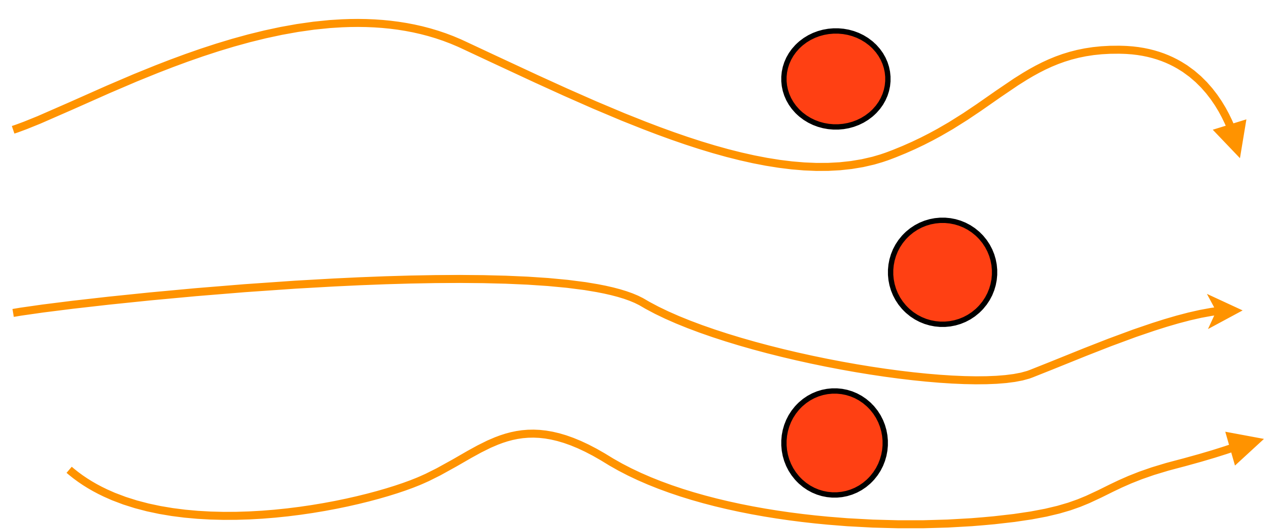
Static Analysis

Static analyses aggressively over-approximate disjunction for efficiency.



Decision Procedures

Modern SAT solvers precisely reason about disjunction.



Static Analysis, BMC, and the Problem of Disjunction

```
float x;
if(x >= 0 && x <= 10)
{
  for(int i = 0; i < 5; ++i)
    x*=x;

  assert(x >= 0 && x <= 1e32);
}
```

Interval Analysis

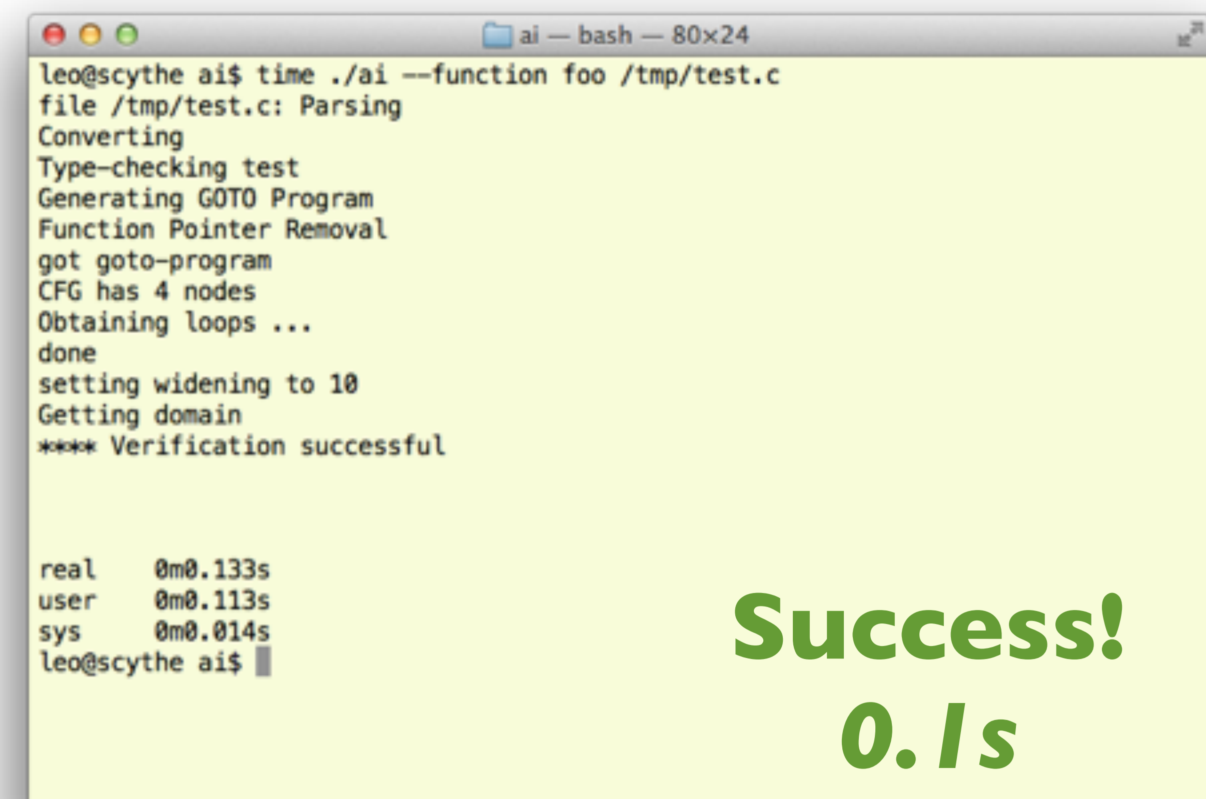
Bounded Model Checking

Static Analysis, BMC, and the Problem of Disjunction

```c
float x;
if(x >= 0 && x <= 10)
{
  for(int i = 0; i < 5; ++i)
    x*=x;

  assert(x >= 0 && x <= 1e32);
}
```

Interval Analysis



Bounded Model Checking

Static Analysis, BMC, and the Problem of Disjunction

```c
float x;
if(x >= 0 && x <= 10)
{
  for(int i = 0; i < 5; ++i)
    x*=x;

  assert(x >= 0 && x <= 1e32);
}
```

Interval Analysis

```
ai — bash — 80×24
leo@scythe ai$ time ./ai --function foo /tmp/test.c
file /tmp/test.c: Parsing
Converting
Type-checking test
Generating GOTO Program
Function Pointer Removal
got goto-program
CFG has 4 nodes
Obtaining loops ...
done
setting widening to 10
Getting domain
**** Verification successful


real    0m0.133s
user    0m0.113s
sys     0m0.014s
leo@scythe ai$
```

Success!
0.1s

Bounded Model Checking

```
cbmc — bash — 80×24
file /tmp/test.c: Parsing
Converting
Type-checking test
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 20 assignments
simple slicing removed 2 assignments
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Solving with MiniSAT2 with simplifier
132504 variables, 576600 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 293.838s
VERIFICATION SUCCESSFUL

real    4m54.088s
user    4m53.011s
sys     0m0.745s
leo@scythe cbmc$
```

Success!
290s

Static Analysis, BMC, and the Problem of Disjunction

```
float a,x;

if(x < 0)
  a = 1;
else
  a = -1;

assert(a != 0);
```

Interval Analysis

Bounded Model Checking

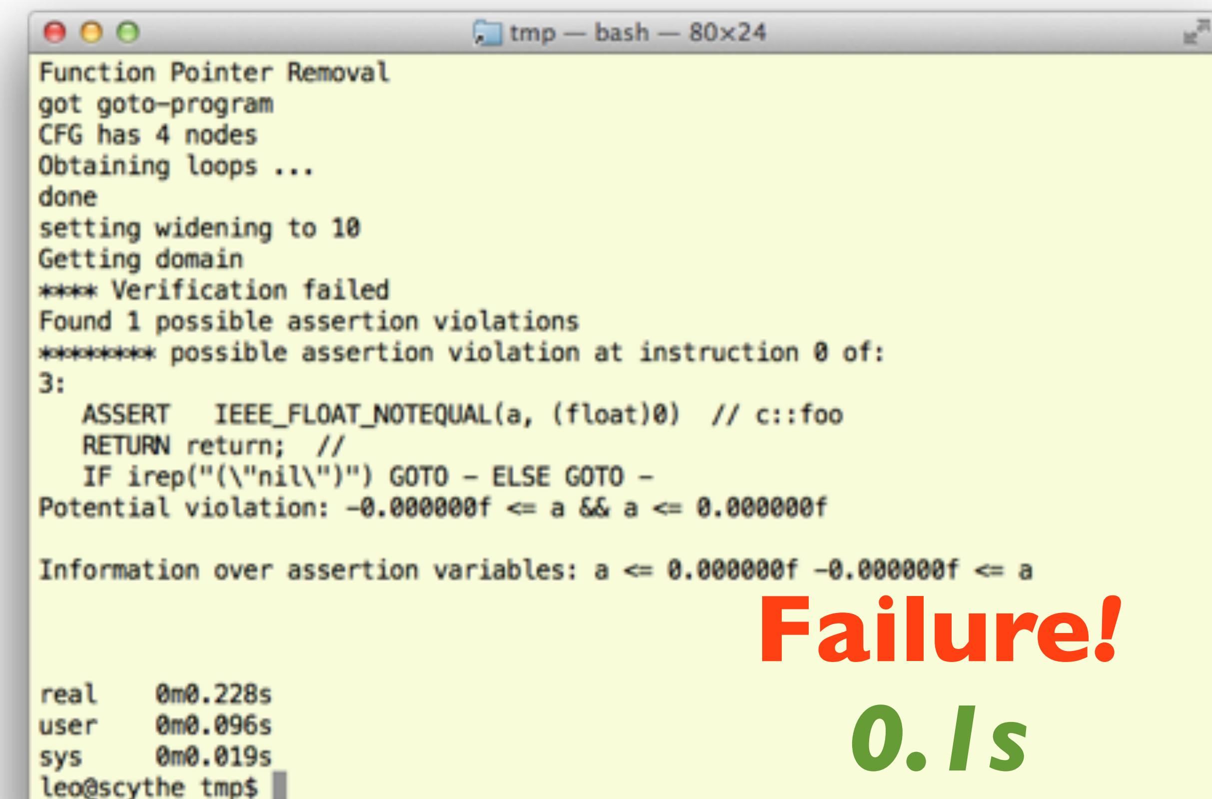
Static Analysis, BMC, and the Problem of Disjunction

```
float a,x;

if(x < 0)
  a = 1;
else
  a = -1;
assert(a != 0);
```

$\langle x : \top, a : [-1.0, 1.0] \rangle \longleftarrow$ `assert(a != 0);`

Interval Analysis



Failure!

0.1s

Bounded Model Checking

Static Analysis, BMC, and the Problem of Disjunction

```c
float a,x;

if(x < 0)
  a = 1;
else
  a = -1;
assert(a != 0);
```

$\langle x : \top, a : [-1.0, 1.0] \rangle \longleftarrow$ `assert(a != 0);`

Interval Analysis

```
tmp — bash — 80×24
Function Pointer Removal
got goto-program
CFG has 4 nodes
Obtaining loops ...
done
setting widening to 10
Getting domain
**** Verification failed
Found 1 possible assertion violations
******** possible assertion violation at instruction 0 of:
3:
   ASSERT   IEEE_FLOAT_NOTEQUAL(a, (float)0)  // c::foo
   RETURN return;  //
   IF irep("(\"nil\")") GOTO - ELSE GOTO -
Potential violation: -0.000000f <= a && a <= 0.000000f

Information over assertion variables: a <= 0.000000f -0.000000f <= a


real    0m0.228s
user    0m0.096s
sys     0m0.019s
leo@scythe tmp$
```

Failure!

0.1s

Bounded Model Checking

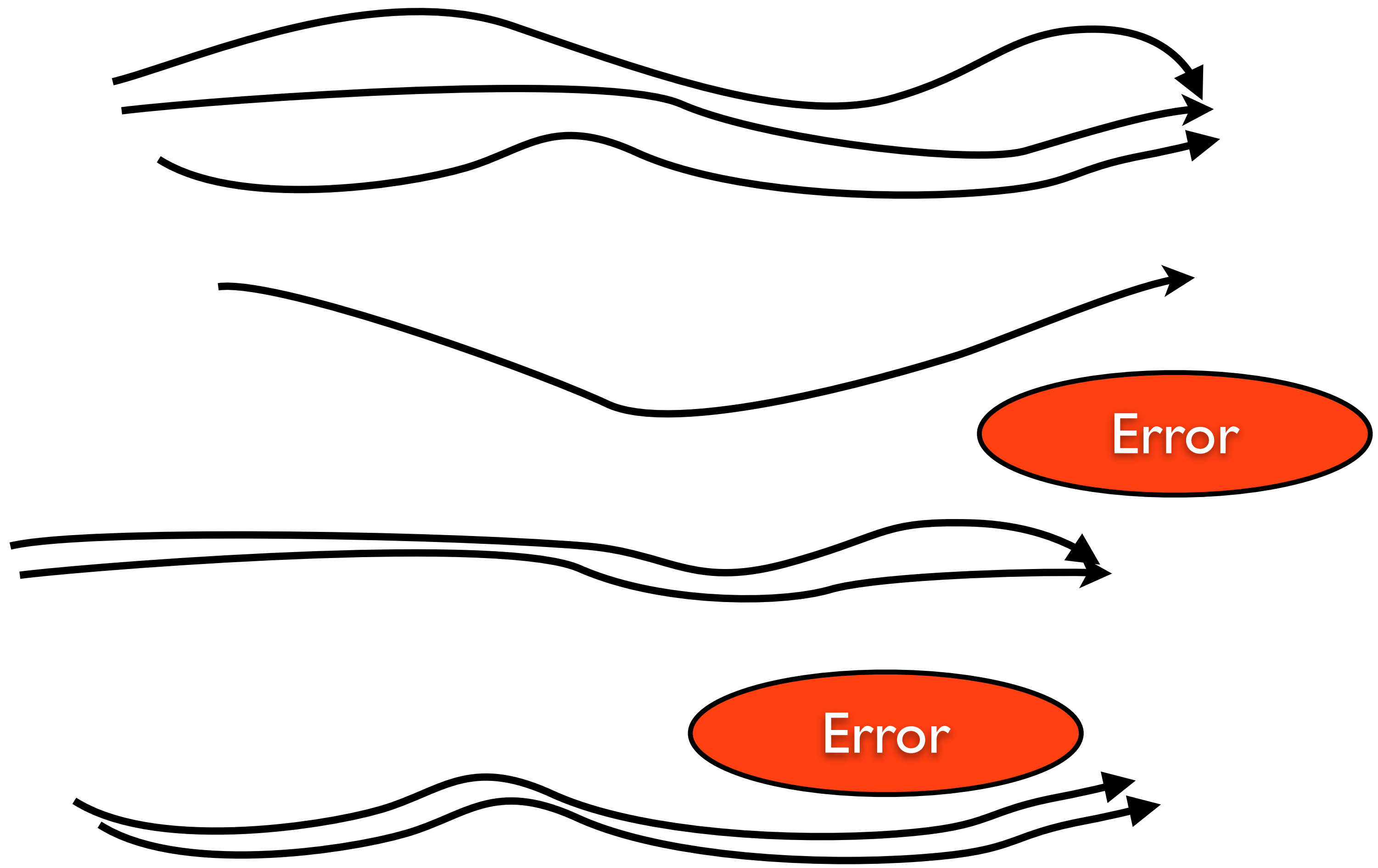
```
tmp — bash — 80×24
file test.c: Parsing
Converting
Type-checking test
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 17 assignments
simple slicing removed 0 assignments
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Solving with MiniSAT2 with simplifier
139 variables, 304 clauses
empty clause: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.004s
VERIFICATION SUCCESSFUL

real    0m0.117s
user    0m0.099s
sys     0m0.014s
leo@scythe tmp$
```

Success!

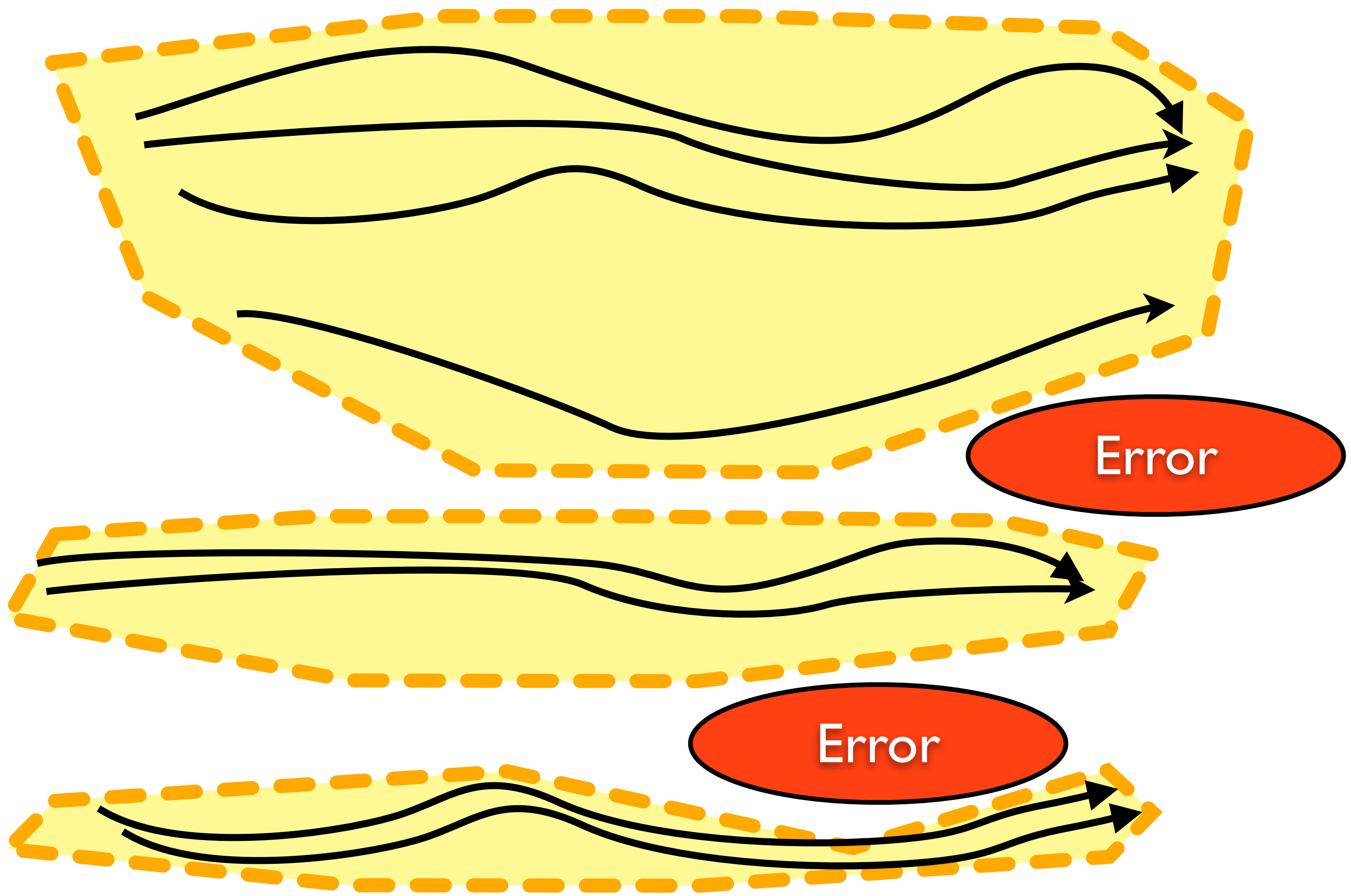
0.1s

Static Analysis or Bit-Blasting?



Standard static analysis fails, but we could do better than bit-blasting?

Static Analysis or Bit-Blasting?



Standard static analysis fails, but we could do better than bit-blasting?

Idea: Partition the traces so that we can prove correctness for each partition.
Question: Where does the partition come from?

To be efficient, we want partitions that are **just precise enough**

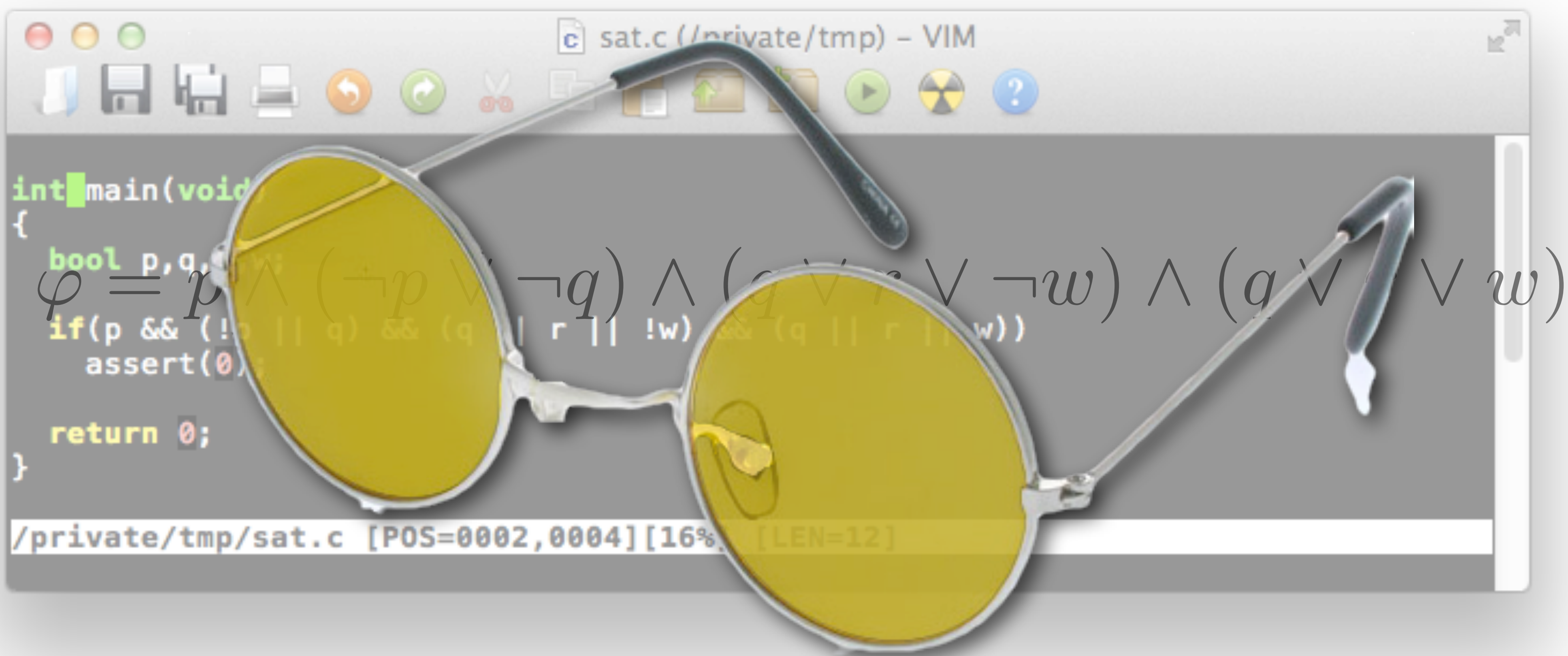
Our Contribution

- Conflict Driven Fixed Point Learning (CDFL)
  - Intelligent, property-driven refinement for abstract analyses
  - Distinct from and orthogonal to CEGAR
- Instantiation of CDFL(Interval)
  - Significantly faster than modern SAT solvers on FP programs
  - Better precision than straightforward abstract analysis

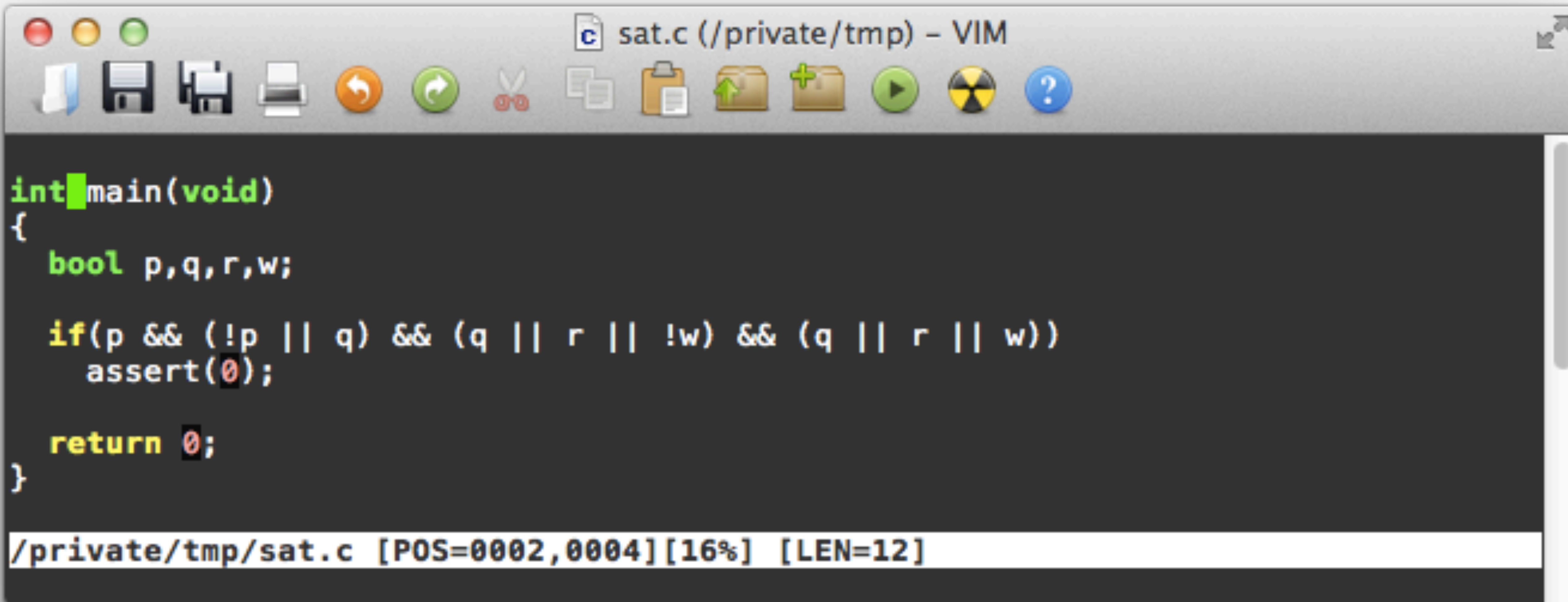
WHAT WOULD
A SAT SOLVER DO?

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

Imagine no assignments,
it's easy if you try



Imagine only Booleans,
I wonder if you can


sat.c (/private/tmp) - VIM
int main(void)
{
  bool p,q,r,w;

  if(p && (!p || q) && (q || r || !w) && (q || r || w))
    assert(0);

  return 0;
}
/private/tmp/sat.c [POS=0002,0004][16%] [LEN=12]

SAT Solvers Operate over Abstract Domains

SAT Solvers Operate over Abstract Domains

Partial assignment

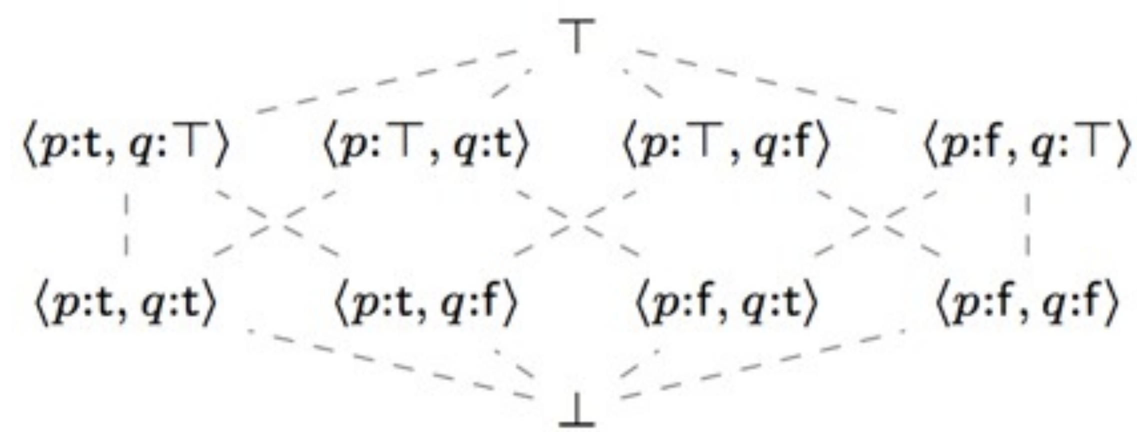
$$Prop \rightarrow \{\mathsf{t}, \mathsf{f}, ?\}$$

SAT Solvers Operate over Abstract Domains

Partial assignment

$$Prop \to \{\mathsf{t}, \mathsf{f}, ?\}$$

Boolean Constants Domain



Deduction

Deduction

SAT

```
bool p,q;

if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

Deduction

SAT

```
bool p,q;
```
$\longrightarrow \langle p : \top, q : \top \rangle$

```
if(p)
```
$\longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

```
  if(!p || !q)
```
$\longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

```
    if( ... )
      assert(0);
```

Deduction

SAT

```
bool p,q;
if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

$\texttt{bool p,q;} \longrightarrow \langle p : \top, q : \top \rangle$

$\texttt{if(p)} \longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

$\texttt{if(!p || !q)} \longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

$p : \mathsf{t} \longrightarrow q : \mathsf{f}$

Deduction

SAT

```
bool p,q;
if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

bool p,q; $\longrightarrow \langle p : \top, q : \top \rangle$

if(p) $\longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

if(!p || !q) $\longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

$p : \mathsf{t} \longrightarrow q : \mathsf{f}$

Intervals

```
if(x <= 10.0)
{
  l1: y = x * 2;
  l2:
}
```

Deduction

SAT

```
bool p,q;
if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

$\texttt{bool p,q;} \longrightarrow \langle p : \top, q : \top \rangle$

$\texttt{if(p)} \longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

$\texttt{if(!p || !q)} \longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

$p : \mathsf{t} \longrightarrow q : \mathsf{f}$

Intervals

```
if(x <= 10.0)
{
  l1: y = x * 2;
  l2:
}
```

$\texttt{l1:} \longrightarrow \langle x : [-\infty, 10.0], y : \top \rangle$

$\texttt{l2:} \longrightarrow \langle x : [-\infty, 10.0], y : [-\infty, 20] \rangle$

Deduction

SAT

```
bool p,q;
if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

$\texttt{bool p,q;} \longrightarrow \langle p : \top, q : \top \rangle$

$\texttt{if(p)} \longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

$\texttt{if(!p || !q)} \longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

$p : \mathsf{t} \longrightarrow q : \mathsf{f}$

Intervals

```
if(x <= 10.0)
{
  l1: y = x * 2;
  l2:
}
```

$\texttt{l1:} \longrightarrow \langle x : [-\infty, 10.0], y : \top \rangle$

$\texttt{l2:} \longrightarrow \langle x : [-\infty, 10.0], y : [-\infty, 20] \rangle$

$l_1 : \langle x : [\infty, 10.0] \rangle \longrightarrow l_2 : \langle x : [\infty, 10.0] \rangle$

$l_1 : \langle x : [\infty, 10.0] \rangle \longrightarrow l_2 : \langle y : [\infty, 20.0] \rangle$

Deduction

SAT

```
bool p,q;
if(p)
  if(!p || !q)
    if( ... )
      assert(0);
```

`bool p,q;` $\longrightarrow \langle p : \top, q : \top \rangle$

`if(p)` $\longrightarrow \langle p : \mathsf{t}, q : \top \rangle$

`if(!p || !q)` $\longrightarrow \langle p : \mathsf{t}, q : \mathsf{f} \rangle$

$p : \mathsf{t} \longrightarrow q : \mathsf{f}$

Intervals

```
if(x <= 10.0)
{
  l1: y = x * 2;
  l2:
}
```

$\longrightarrow \langle x : [-\infty, 10.0], y : \top \rangle$

$\longrightarrow \langle x : [-\infty, 10.0], y : [-\infty, 20] \rangle$

$l_1 : \langle x : [\infty, 10.0] \rangle \longrightarrow l_2 : \langle x : [\infty, 10.0] \rangle$

$l_1 : \langle x : [\infty, 10.0] \rangle \longrightarrow l_2 : \langle y : [\infty, 20.0] \rangle$

CDFL

Apply abstract strongest post-condition

$$post^A : A \to A$$

Deduction over loops

Intervals

```
x = 0;
l1:
while(x<10)
  x = x + 1;
l2:
```

Deduction over loops

Intervals

```
x = 0;
l1:
while(x<10)
  x = x + 1;
l2:
```

l1: $\longrightarrow \langle x : [0.0, 0.0] \rangle$

l2: $\longrightarrow \langle x : [10.0, 10.0] \rangle$

Deduction over loops

Intervals

```
x = 0;
l1:
while(x<10)
  x = x + 1;
l2:
```

$\longrightarrow \langle x : [0.0, 0.0] \rangle$

$\longrightarrow \langle x : [10.0, 10.0] \rangle$

$$l_1 : \langle x : [0.0, 0.0] \rangle \longrightarrow l_2 : \langle x : [10.0, 10.0] \rangle$$

SAT

Decisions

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

SAT

Decisions

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

bool p,q; $\longrightarrow \langle p : \top, q : \top \rangle$

[...]; $\longrightarrow \langle p : \top, q : \top \rangle$

No information gained!

Decisions

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

bool p,q; $\longrightarrow \langle p : \top, q : \top \rangle$

[...]; $\longrightarrow \langle p : \top, q : \top \rangle$

No information gained!

Case 1
q : f

SAT

Decisions

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

bool p,q; $\longrightarrow \langle p : \top, q : \top \rangle$

[...]; $\longrightarrow \langle p : \top, q : \top \rangle$

No information gained!

Case 1

$q : \mathsf{f}$

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

$\langle p : \top, q : \mathsf{f} \rangle \longleftarrow$ bool p,q;

$\langle p : \mathsf{t}, q : \mathsf{f} \rangle \longleftarrow$ if(p || q)

$\bot \longleftarrow$ if(!p || q)

SAT

Decisions

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

`bool p,q;` $\longrightarrow \langle p : \top, q : \top \rangle$

`[...];` $\longrightarrow \langle p : \top, q : \top \rangle$

No information gained!

Case 1
$q : \mathsf{f}$

Case 2
$q : \mathsf{t}$

Case 1:

```
bool p,q;
if(p || q)
  if(!p || q)
    [...];
```

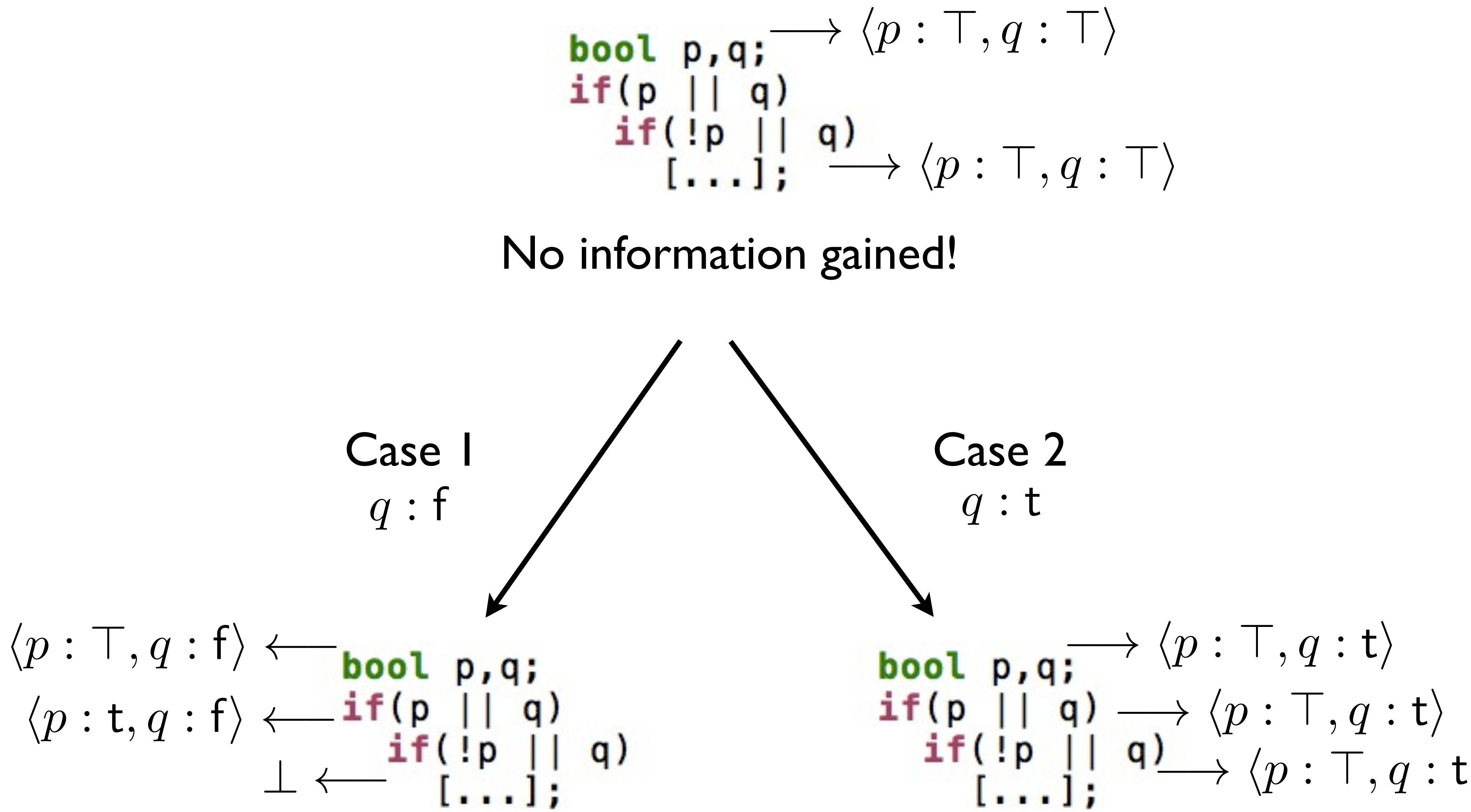
$\langle p : \top, q : \mathsf{f} \rangle \longleftarrow$ `bool p,q;`

$\langle p : \mathsf{t}, q : \mathsf{f} \rangle \longleftarrow$ `if(p || q)`

$\bot \longleftarrow$ `[...];`

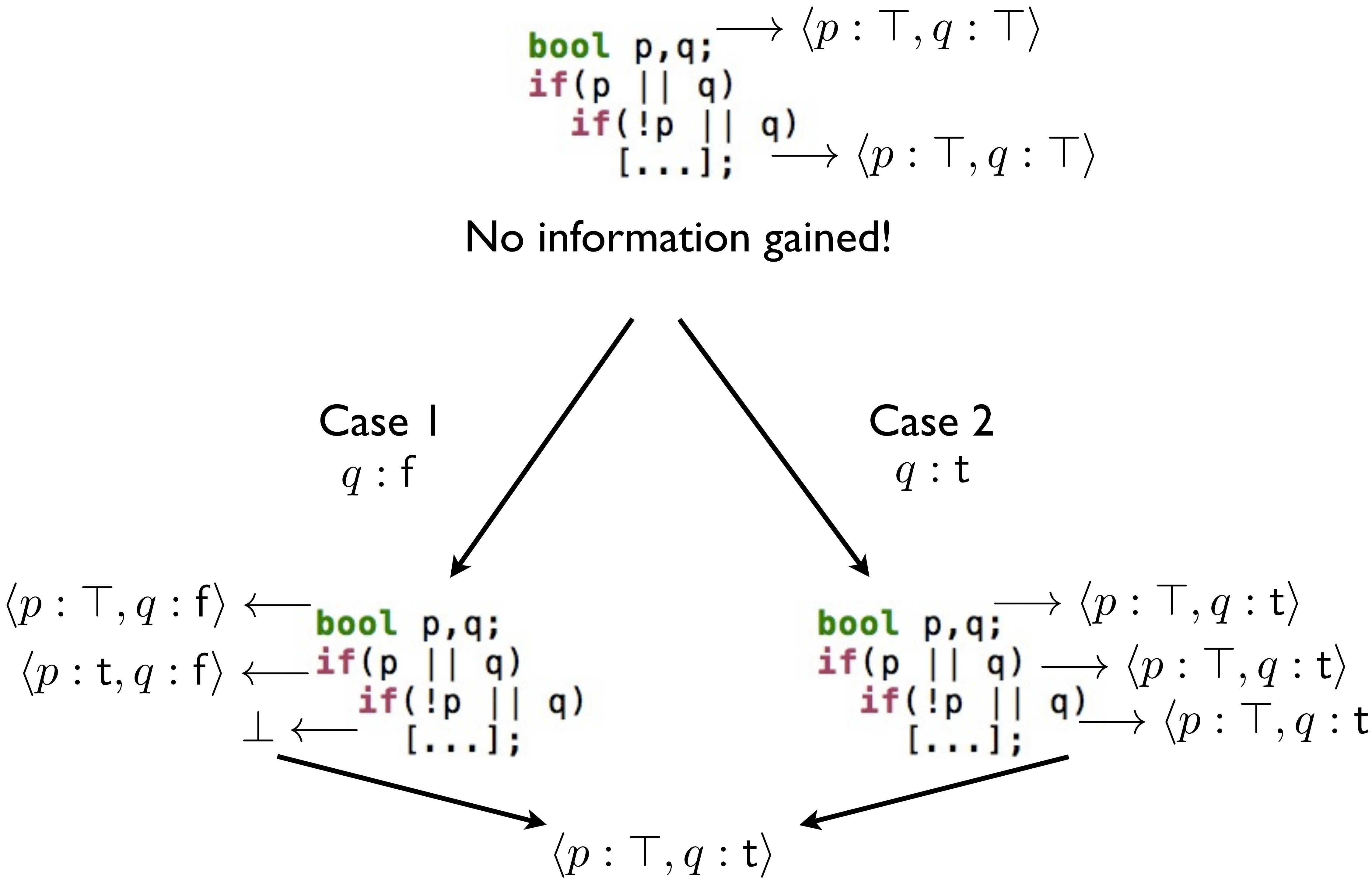
SAT

Decisions



SAT

Decisions



Intervals

Decisions

```
float a;
if(x < 0)
  a = 1;
else
  a = -1;

assert(a!=0);
```

$$\langle x : \top, a : [-1.0, 1.0] \rangle$$

possibly unsafe

Intervals

Decisions

```
float a;
if(x < 0)
  a = 1;
else
  a = -1;

assert(a!=0);
```

$$\langle x : \top, a : [-1.0, 1.0] \rangle$$

possibly unsafe

Case 1

$x < 0$

Intervals

Decisions

```
float a;
if(x < 0)
  a = 1;
else
  a = -1;

assert(a!=0);
```

$$\langle x : \top, a : [-1.0, 1.0]\rangle$$

possibly unsafe

Case 1

$x < 0$

$$\langle x : [\infty, -0.], a : [1.0, 1.0]\rangle$$

Safe

Intervals

Decisions

```
float a;
if(x < 0)
  a = 1;
else
  a = -1;

assert(a!=0);
```

$$\langle x : \top, a : [-1.0, 1.0] \rangle$$

possibly unsafe

Case 1

$x < 0$

Case 2

$x \geq 0$

$$\langle x : [\infty, -0.], a : [1.0, 1.0] \rangle$$

Safe

Intervals

Decisions

```
float a;
if(x < 0)
  a = 1;
else
  a = -1;

assert(a!=0);
```

$$\langle x : \top, a : [-1.0, 1.0] \rangle$$

possibly unsafe

Case 1

$x < 0$

$$\langle x : [\infty, -0.], a : [1.0, 1.0] \rangle$$

Safe

Case 2

$x \geq 0$

$$\langle x : [\infty, -0.], a : [-1.0, -1.0] \rangle$$

Safe

Decisions

There is a common pattern!

CDFL

Decisions

There is a common pattern!

SAT Solvers:

$$\underbrace{\langle p : \mathrm{t}, q : \mathrm{t} \rangle}_{} = \underbrace{\langle p : \mathrm{t} \rangle}_{\langle p:\mathrm{f} \rangle} \sqcap \underbrace{\langle q : \mathrm{t} \rangle}_{\langle q:\mathrm{f} \rangle}$$

no precise complement
as a partial assignment

precise complements

CDFL

Decisions

There is a common pattern!

SAT Solvers:

$$\underbrace{\langle p : \mathsf{t}, q : \mathsf{t}\rangle}_{\text{no precise complement as a partial assignment}} = \underbrace{\langle p : \mathsf{t}\rangle}_{\langle p:\mathsf{f}\rangle} \sqcap \underbrace{\langle q : \mathsf{t}\rangle}_{\langle q:\mathsf{f}\rangle}$$

no precise complement
as a partial assignment

precise complements

Interval Analysis:

$$\underbrace{\langle x : [0, 10], y : [3, \infty]\rangle}_{} = \underbrace{\langle x : [0, \infty]\rangle}_{\langle x:[-\infty,-0.]\rangle} \sqcap \underbrace{\langle x : [-\infty, 10]\rangle}_{\langle x:[10.0,\infty]\rangle} \sqcap \underbrace{\langle y : [3, \infty]\rangle}_{\langle x:[-\infty,2.999]\rangle}$$

no precise complement
as an interval

precise complements

CDFL

Decisions

To instantiate CDFL, we need that:

Lattice elements are decomposable into
meets of precisely complementable elements

$$\forall a \in A.\ a = a_1 \sqcap \ldots \sqcap a_k \text{ s.t. all } a_i \text{ can be precisely complemented}$$

Learning

$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$

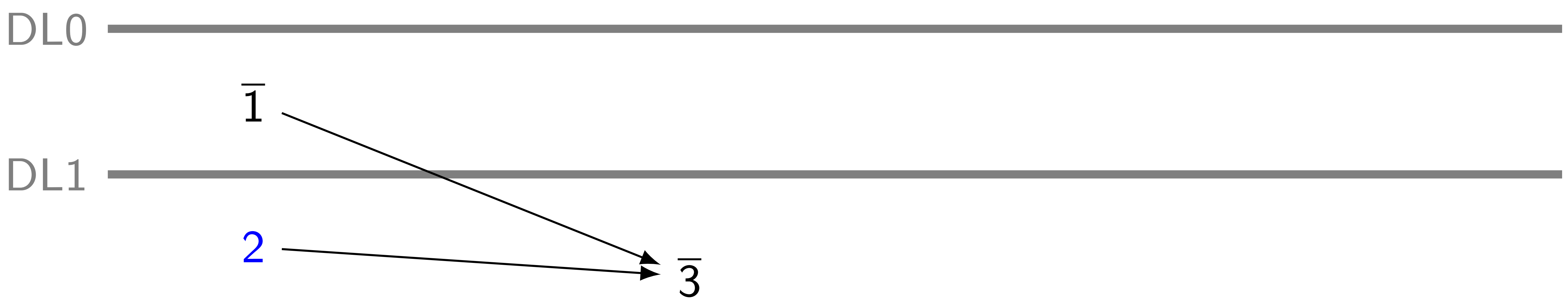
DL0

$\overline{1}$

SAT

Learning

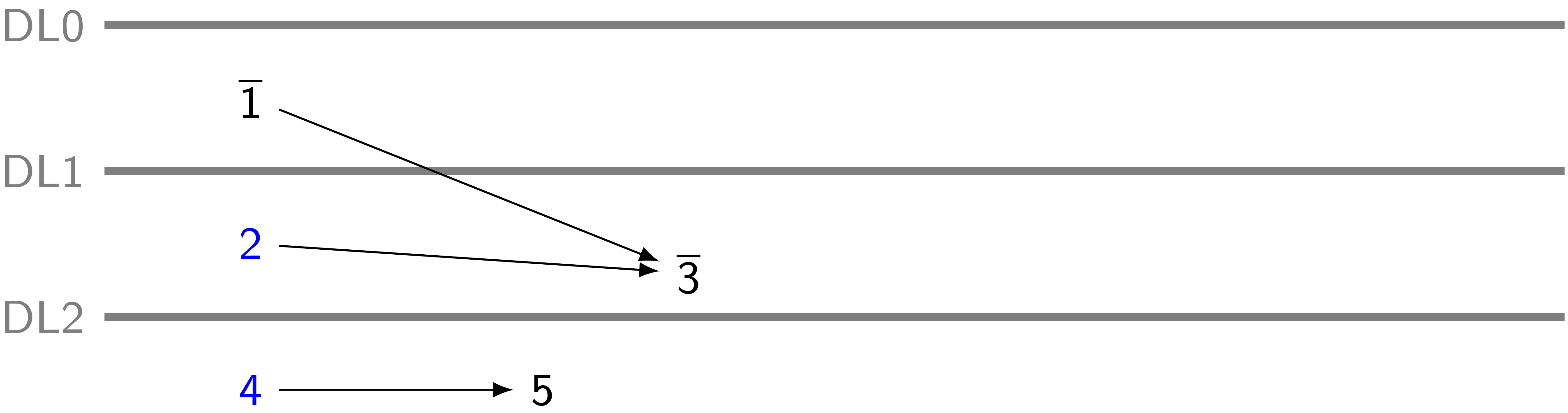
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



SAT

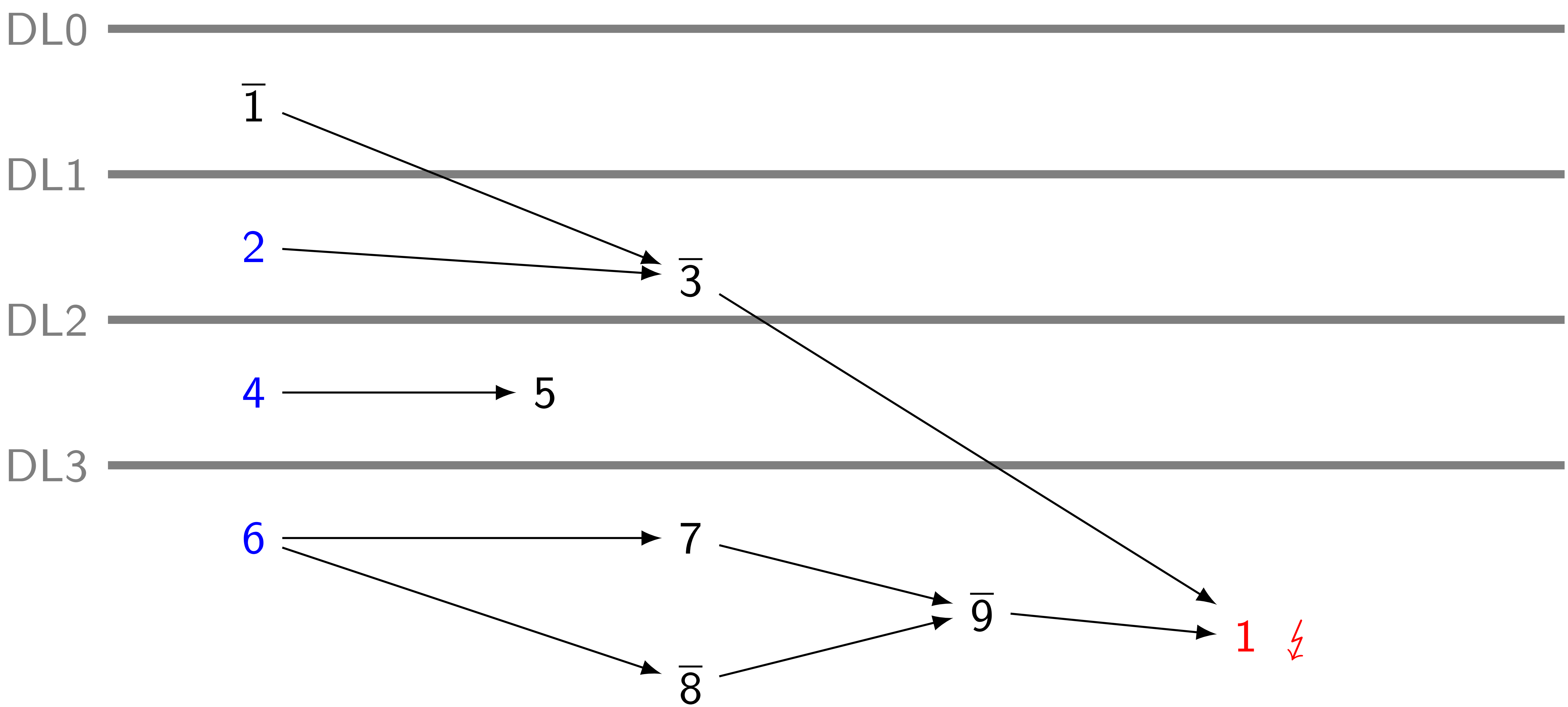
Learning

$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



Learning

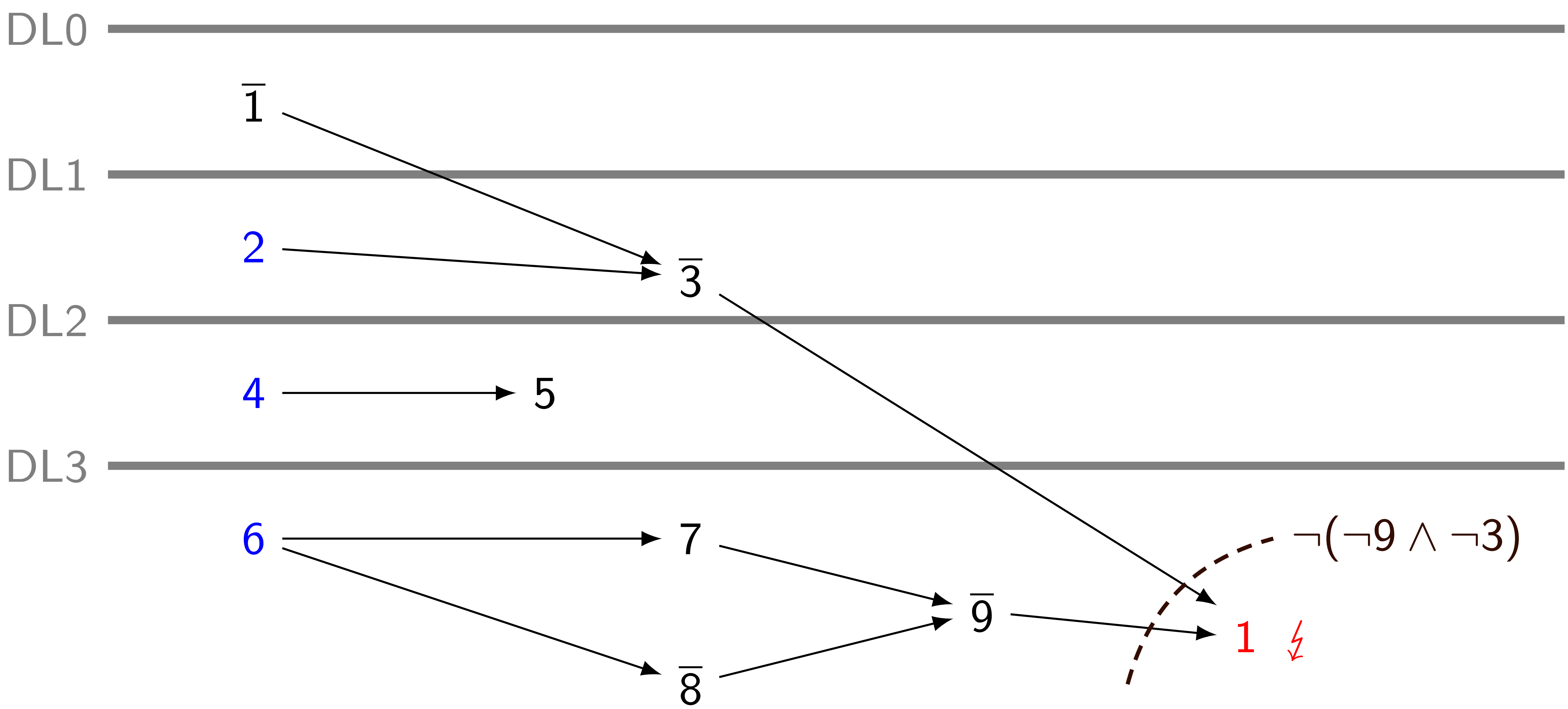
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



SAT

Learning

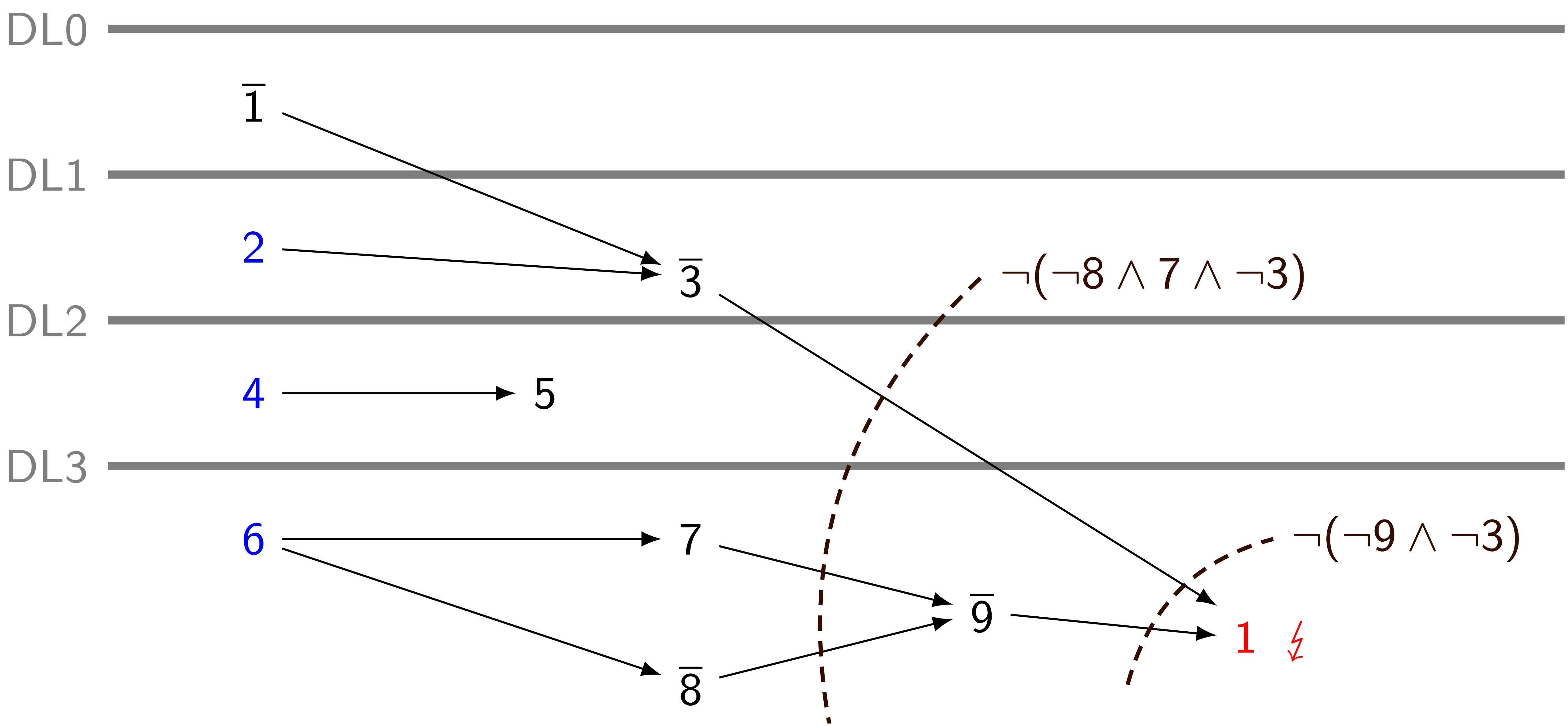
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



SAT

Learning

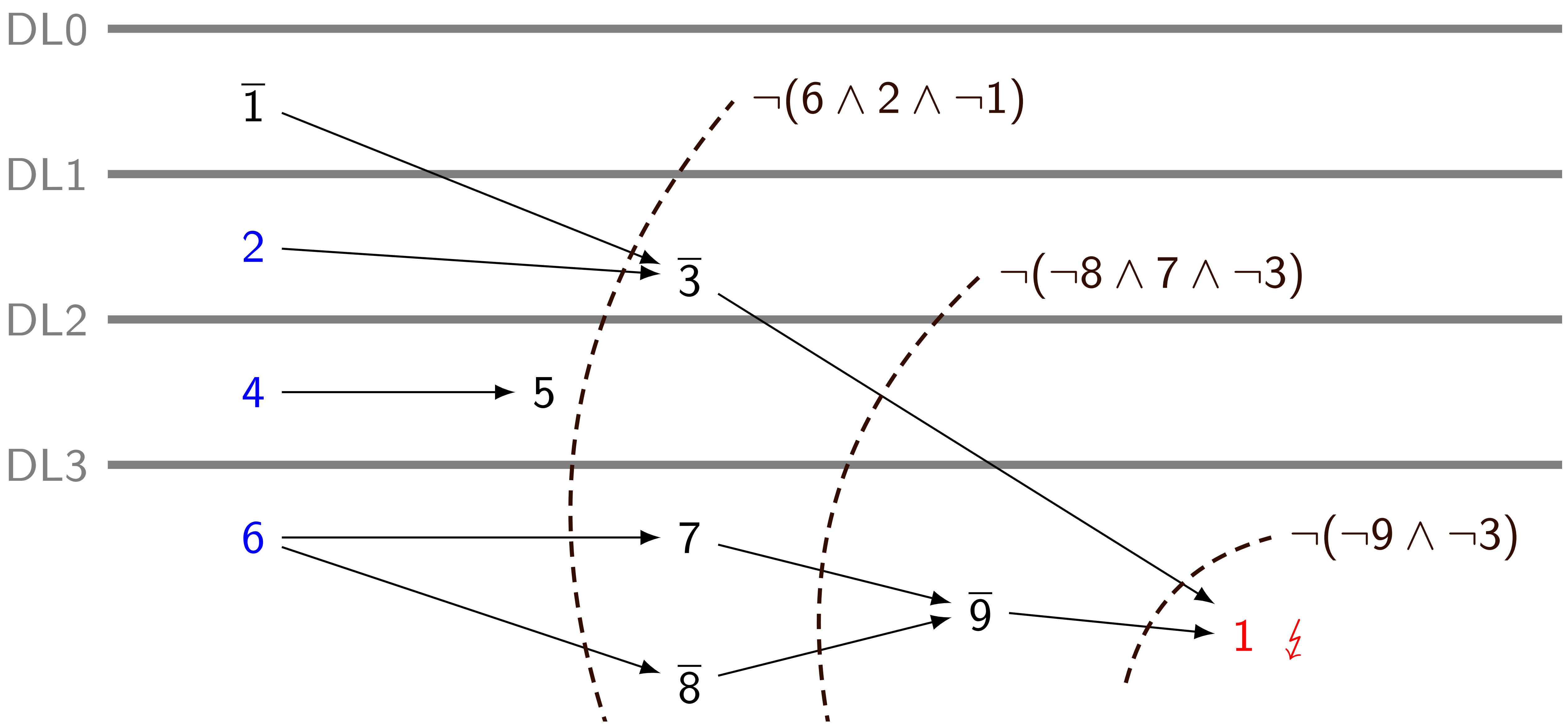
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



SAT

Learning

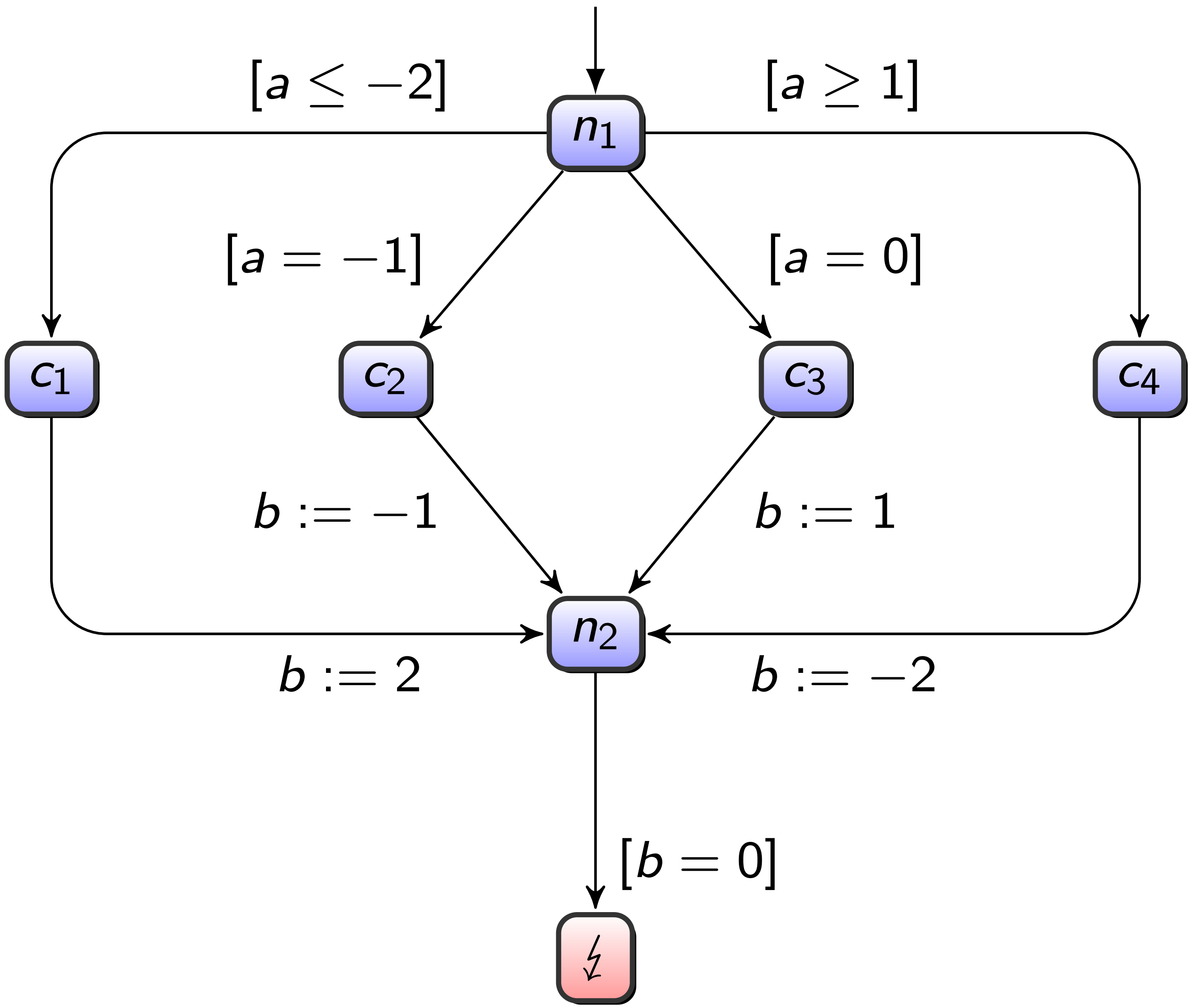
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



Cuts = Heuristic underapproximation of the weakest pre-condition

Intervals

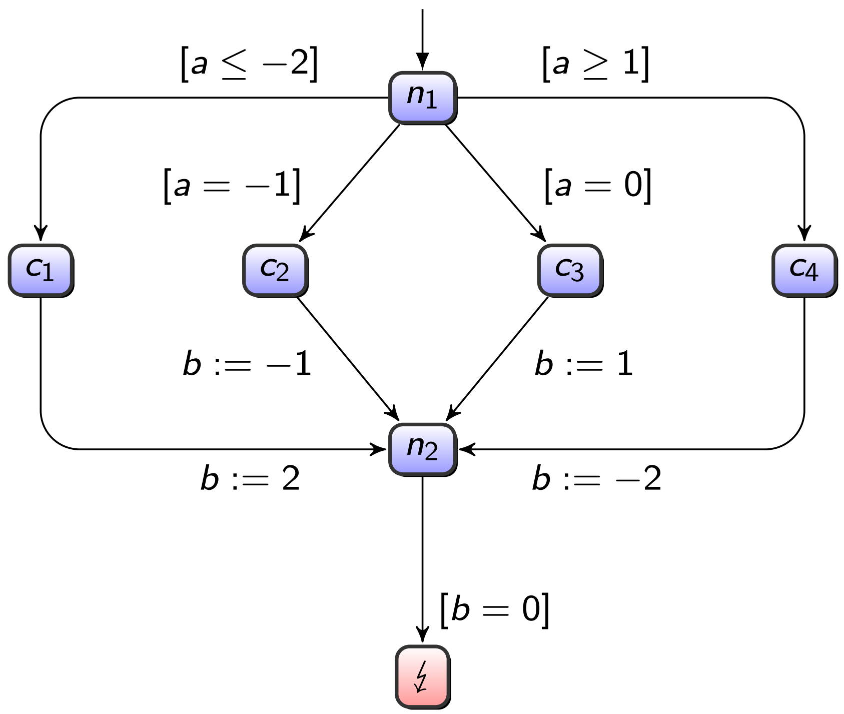
Learning and Conflict Graphs



Intervals

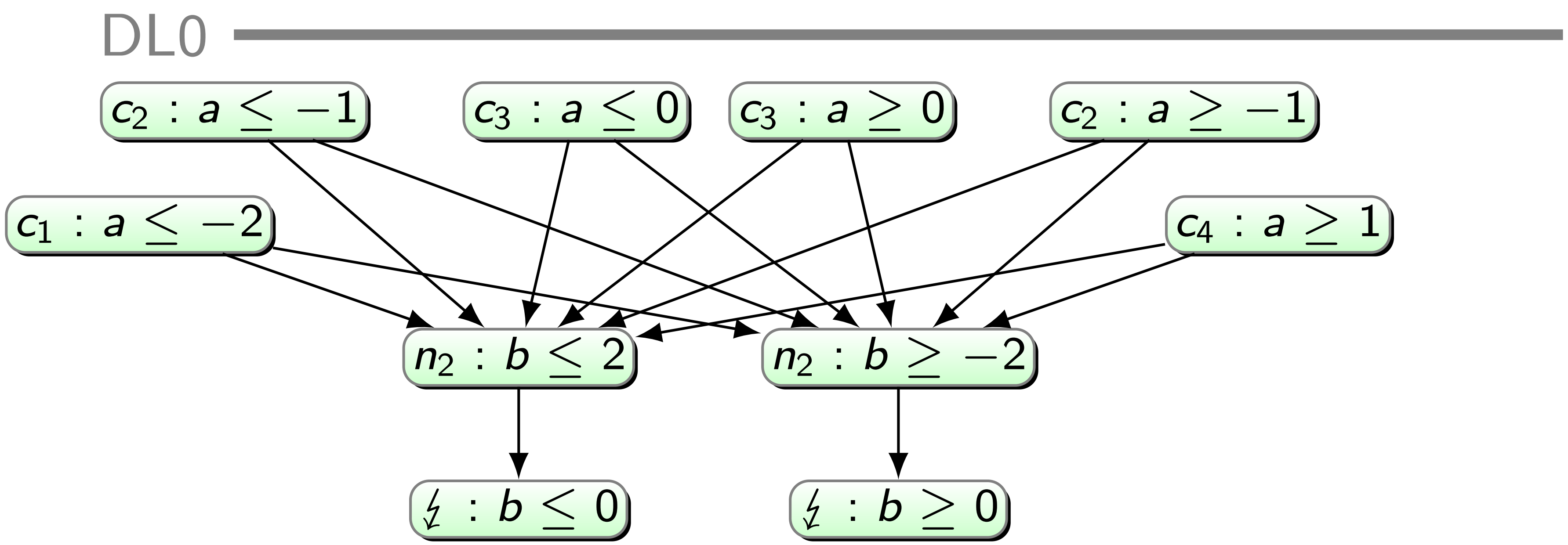
Learning and Conflict Graphs

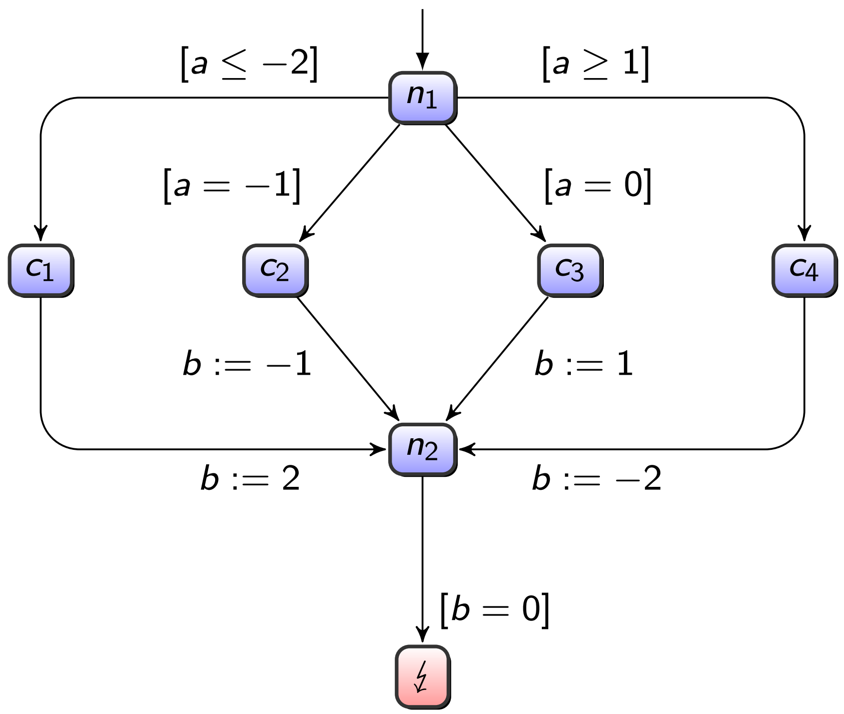
DL0



Intervals

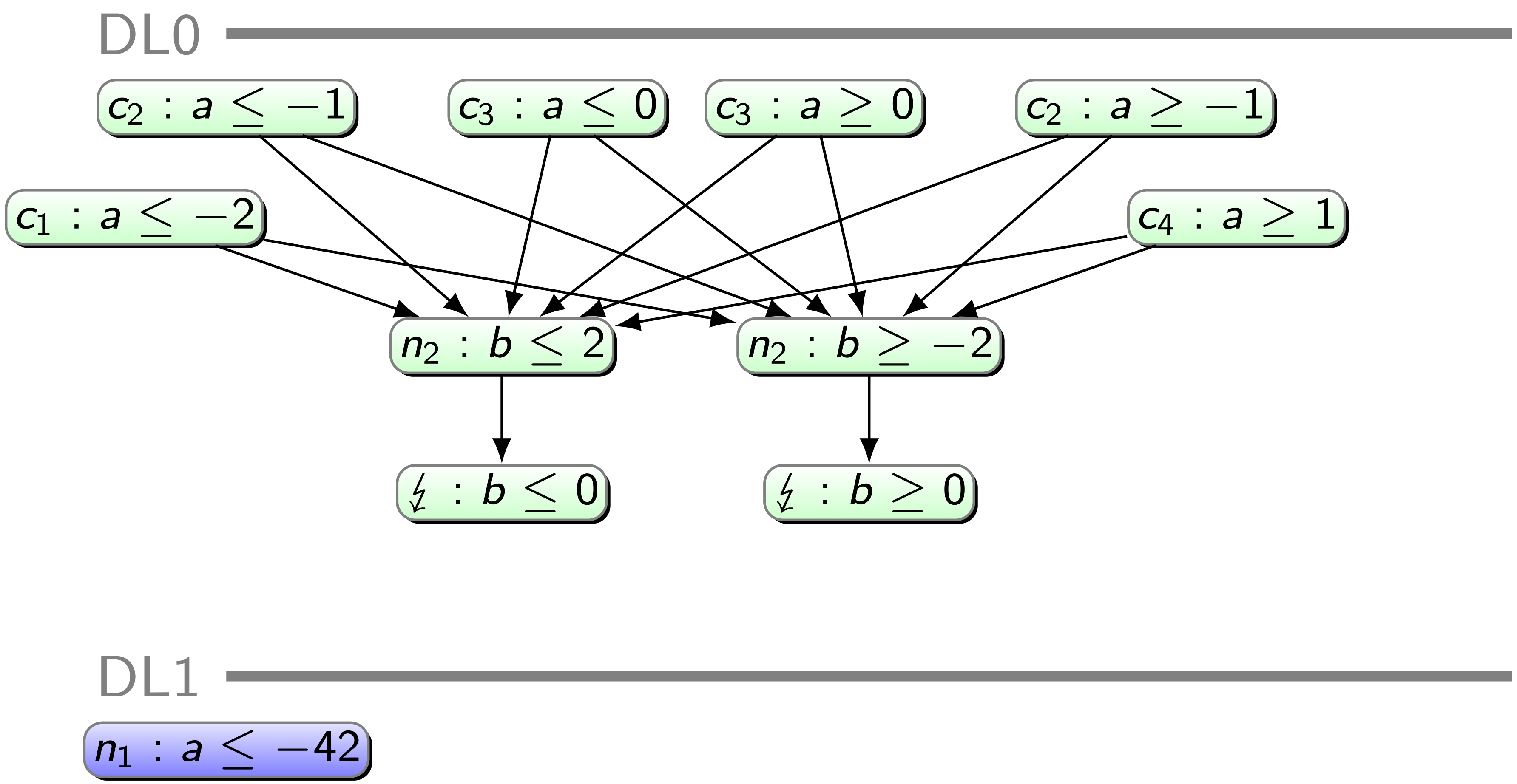
Learning and Conflict Graphs

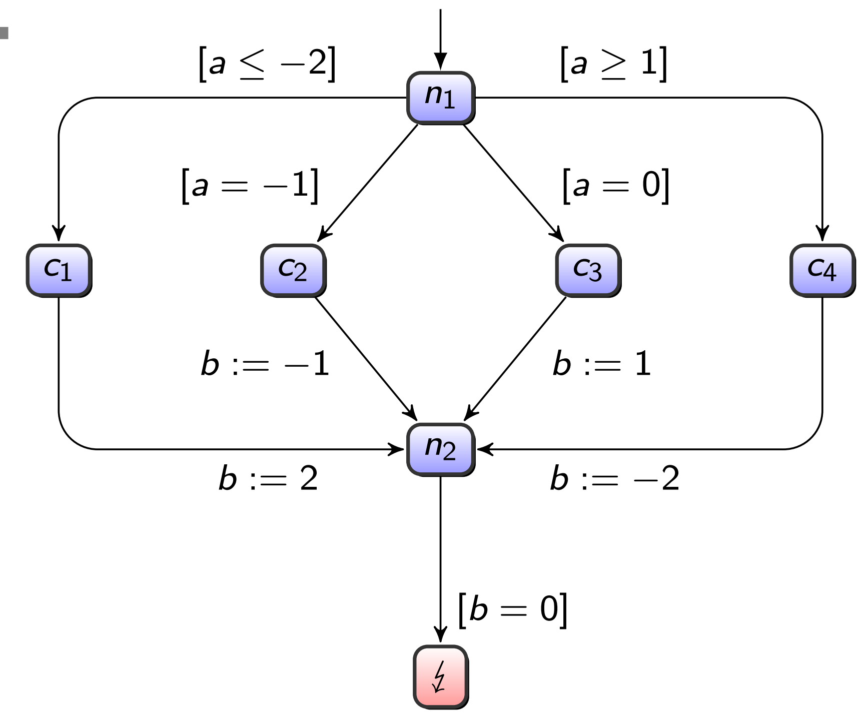




Intervals

Learning and Conflict Graphs

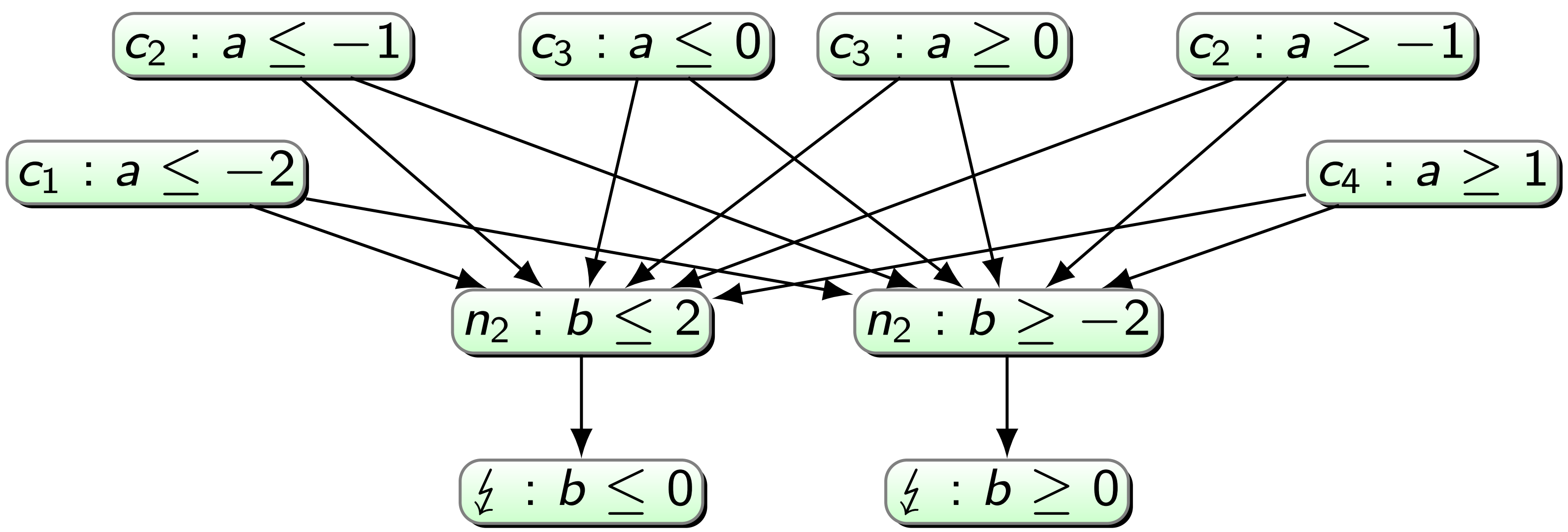


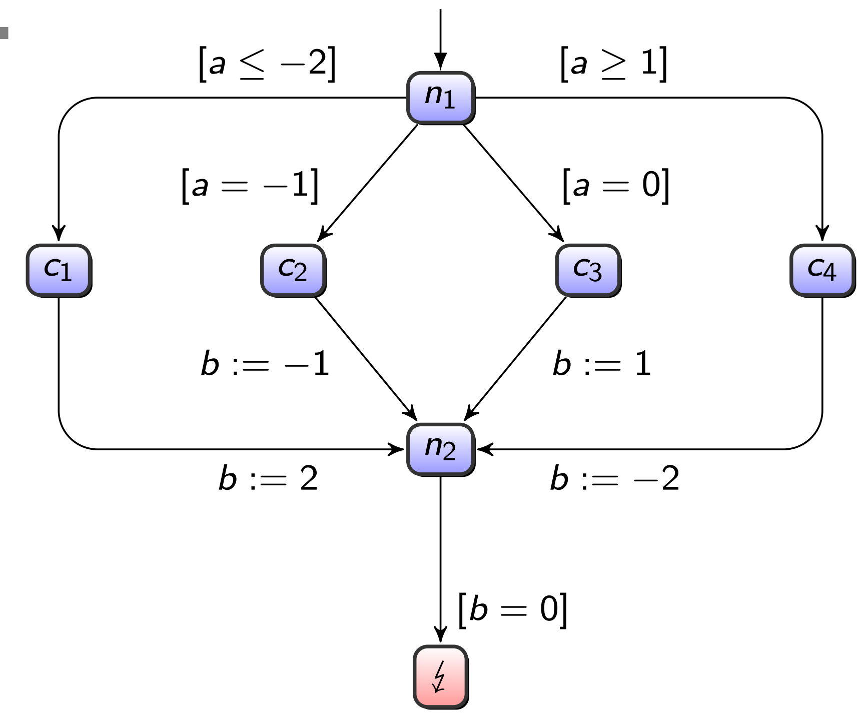


Intervals

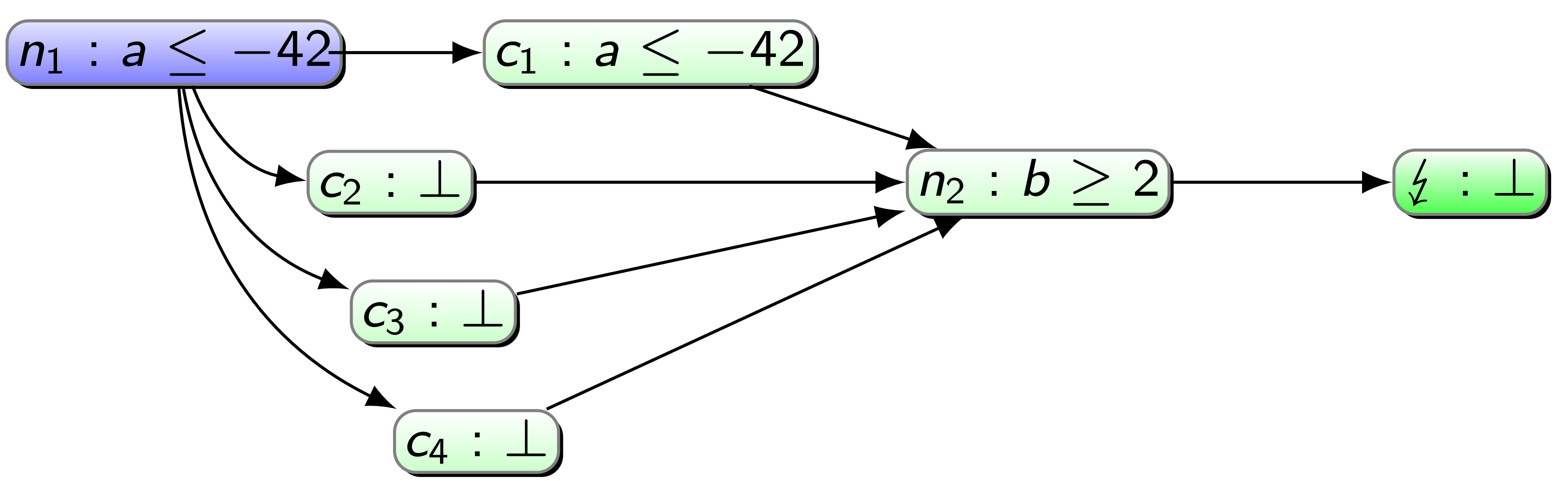
Learning and Conflict Graphs

DL0





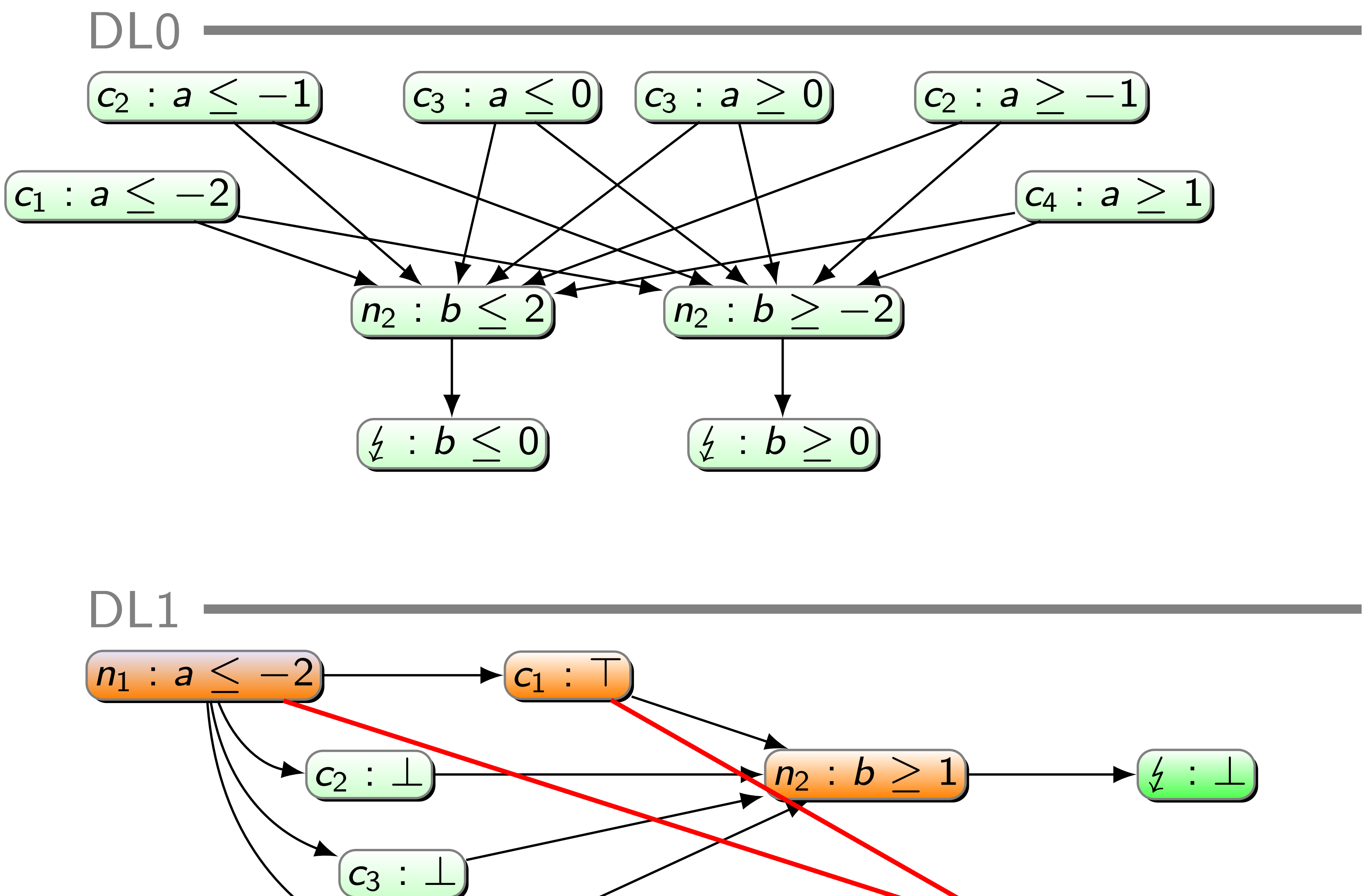
DL1

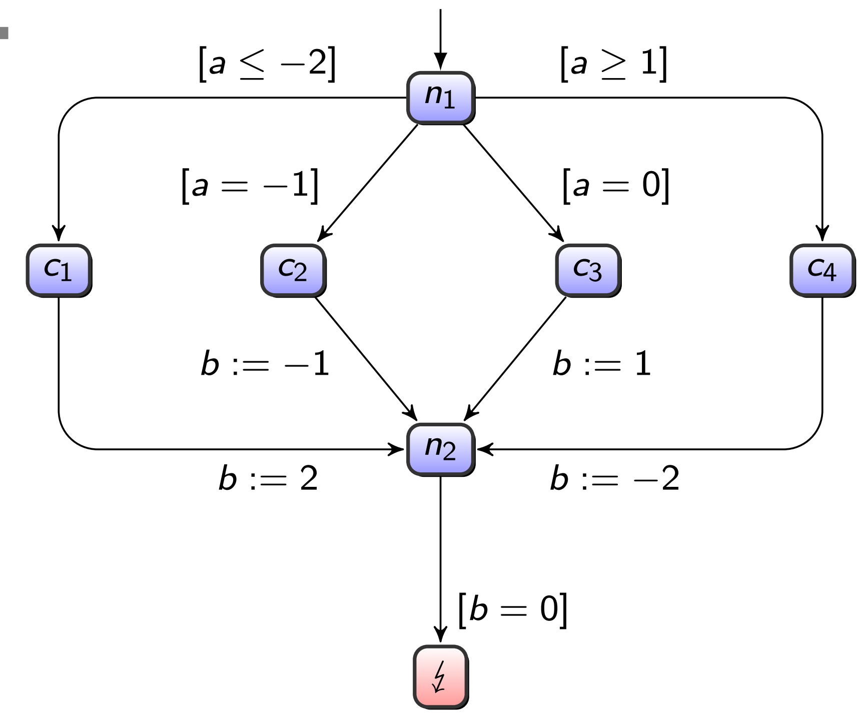


SAFE

Intervals

Learning and Conflict Graphs



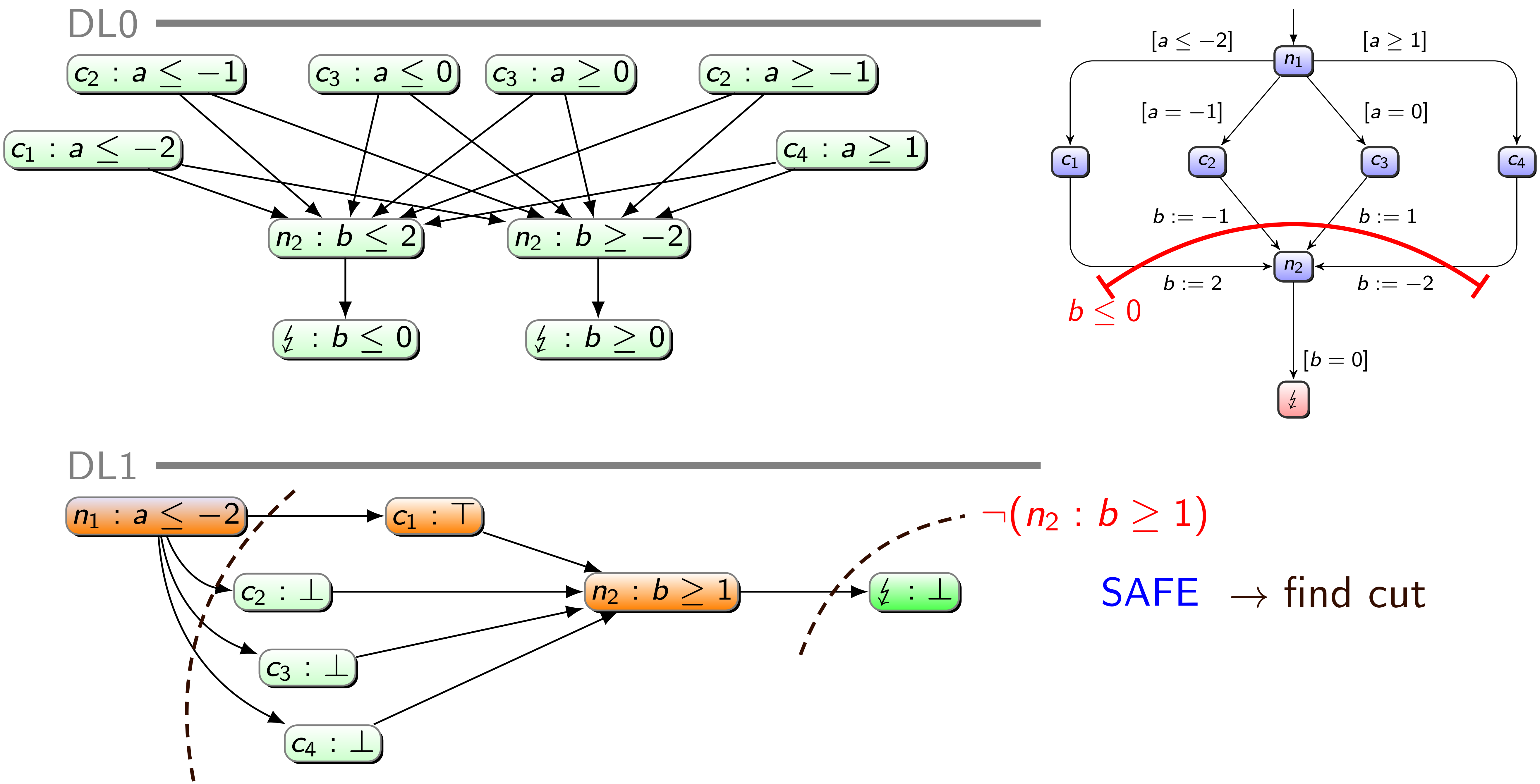


SAFE $\rightarrow$ Generalise!

Heuristic underapproximation of the weakest pre-condition

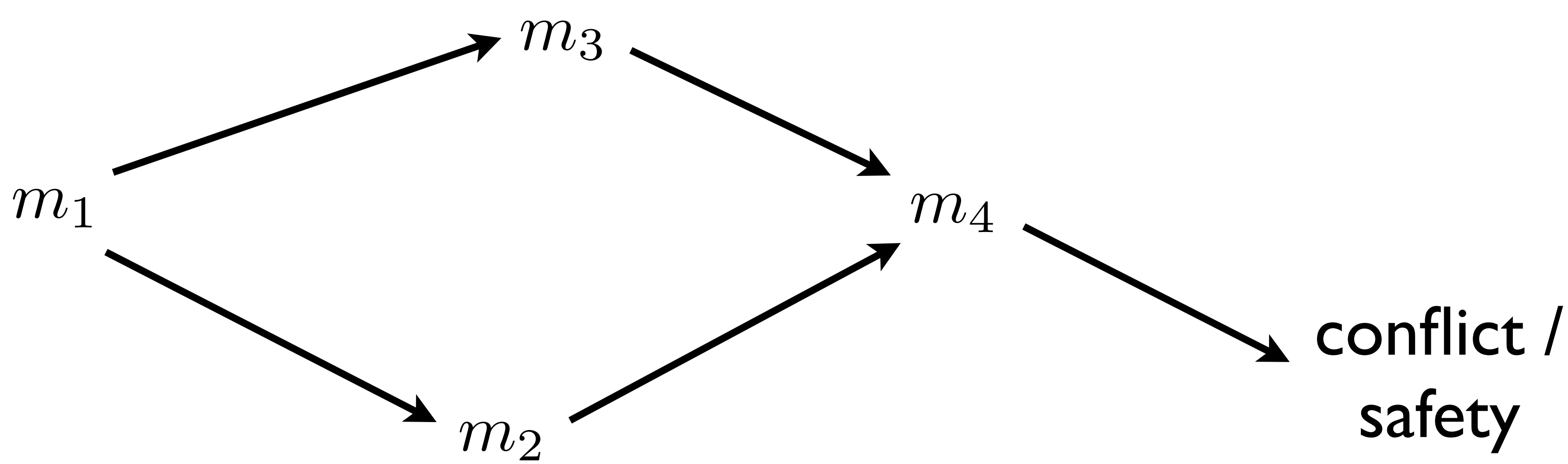
Intervals

Learning and Conflict Graphs



CDFL

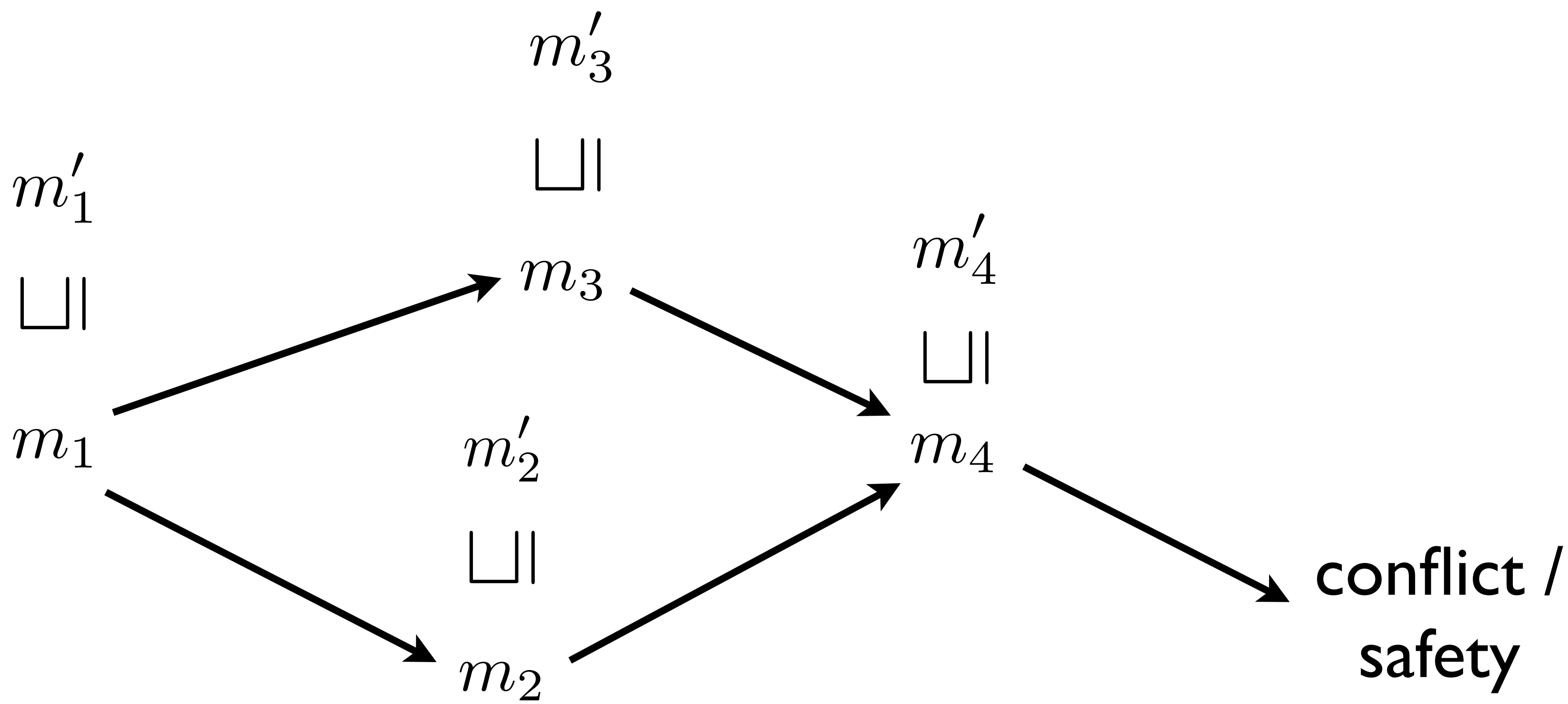
Learning and Conflict Graphs



1. Build an implication graph over complementable elements

CDFL

Learning and Conflict Graphs

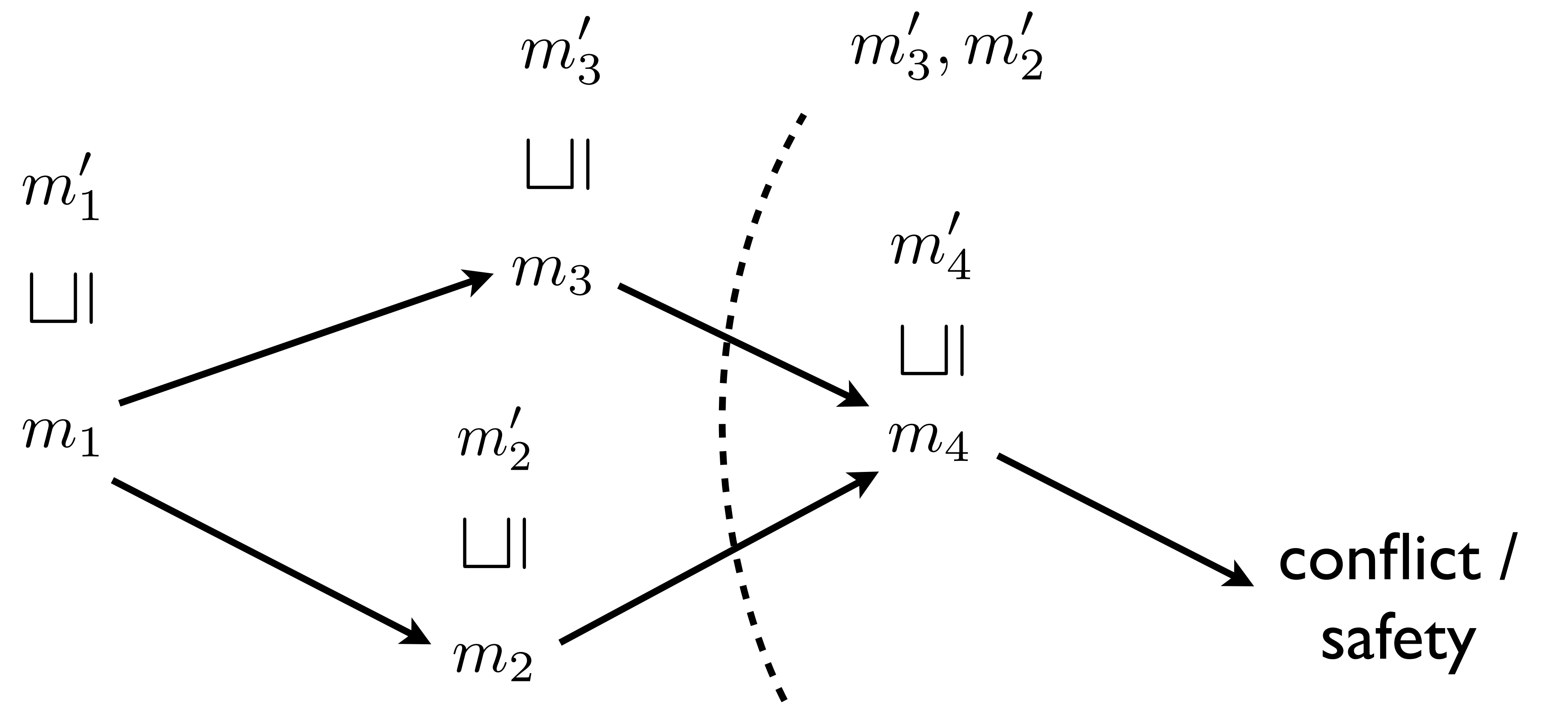


1. Build an implication graph over complementable elements

2. On conflict, generalise the implication graph using under-approximate weakest pre-condition.

CDFL

Learning and Conflict Graphs



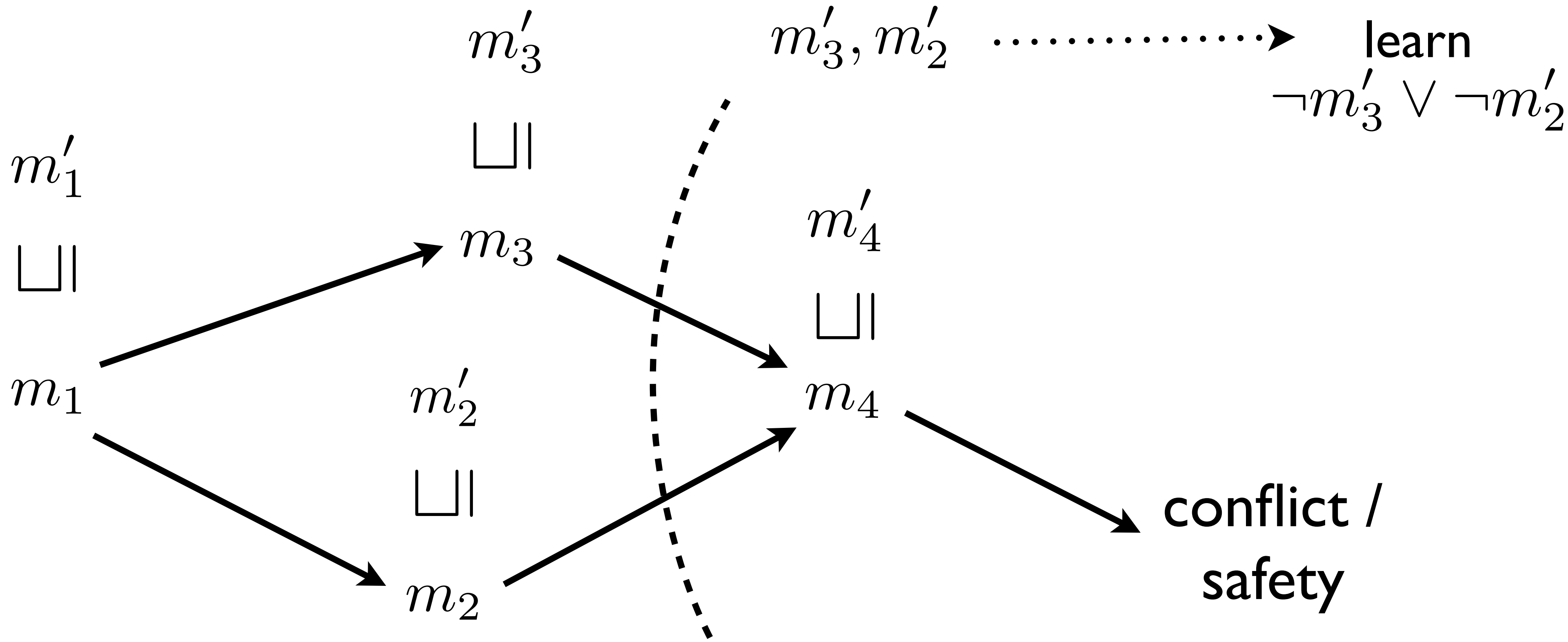
1. Build an implication graph over complementable elements

2. On conflict, generalise the implication graph using under-approximate weakest pre-condition.

3. Cut the implication graph to obtain conflict reason

CDFL

Learning and Conflict Graphs



1. Build an implication graph over complementable elements

2. On conflict, generalise the implication graph using under-approximate weakest pre-condition.

3. Cut the implication graph to obtain conflict reason

4. Negate and add as clause

IMPLEMENTATION AND EXPERIMENTS

Experiments

- Implementation of CDFL(Intervals) applied to floating point programs.

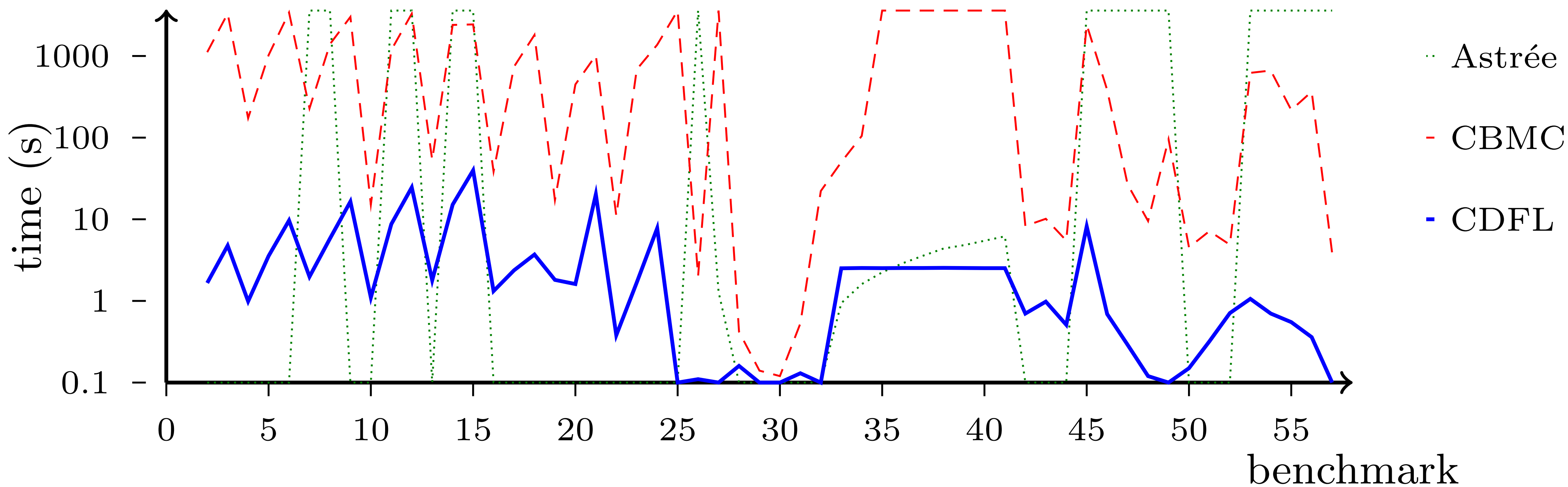
- Analysis is sound and complete in the absence of loops.

- Compared to Astrée and CBMC + state of the art decision procedure, on small, non-linear programs.

Experiments

	safe	bug	unknown / timeout
Astrée	17	0	40
CDFL	**33**	**24**	**0**
CBMC	25	23	9

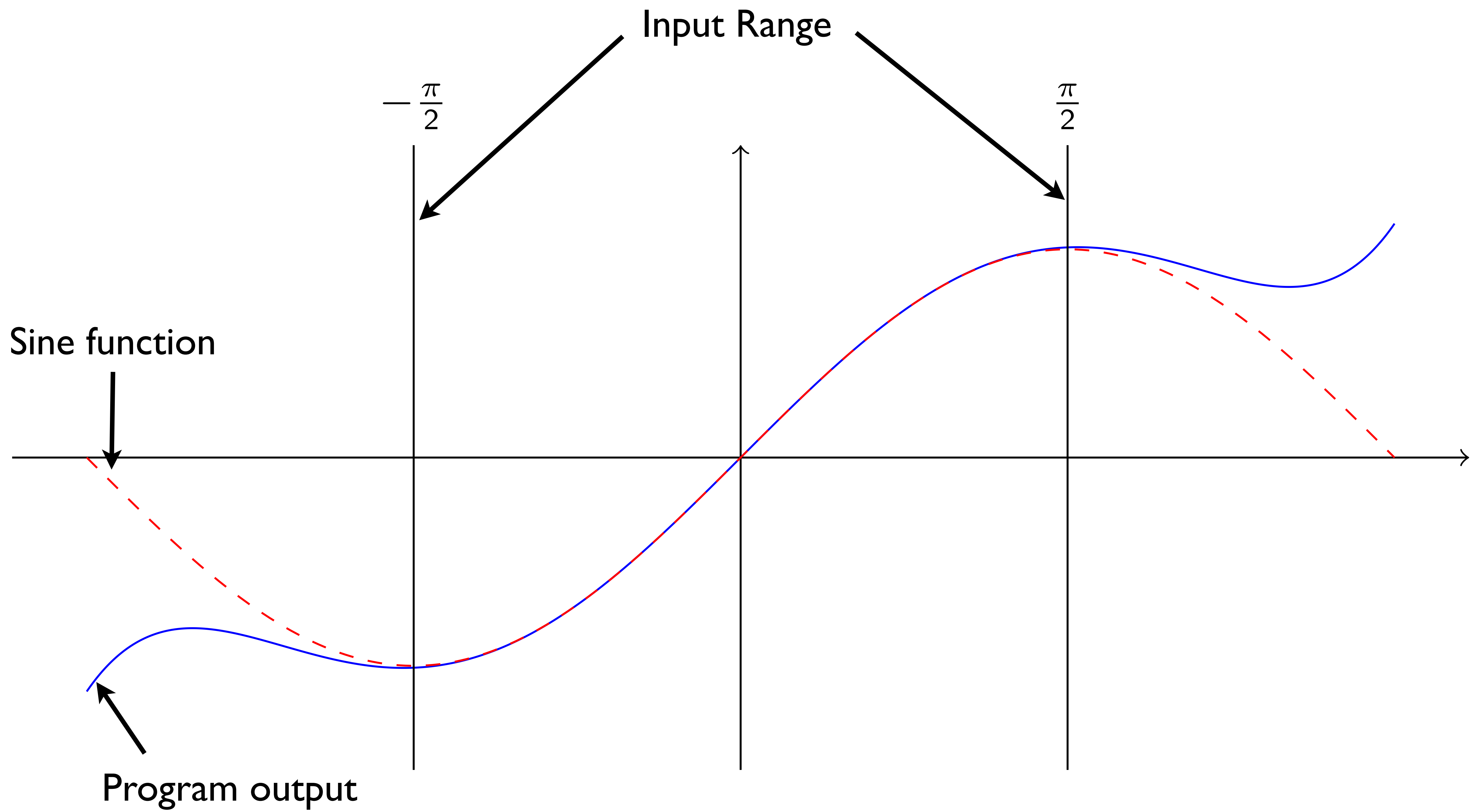
57 small, non-linear FP programs w. bounded loops



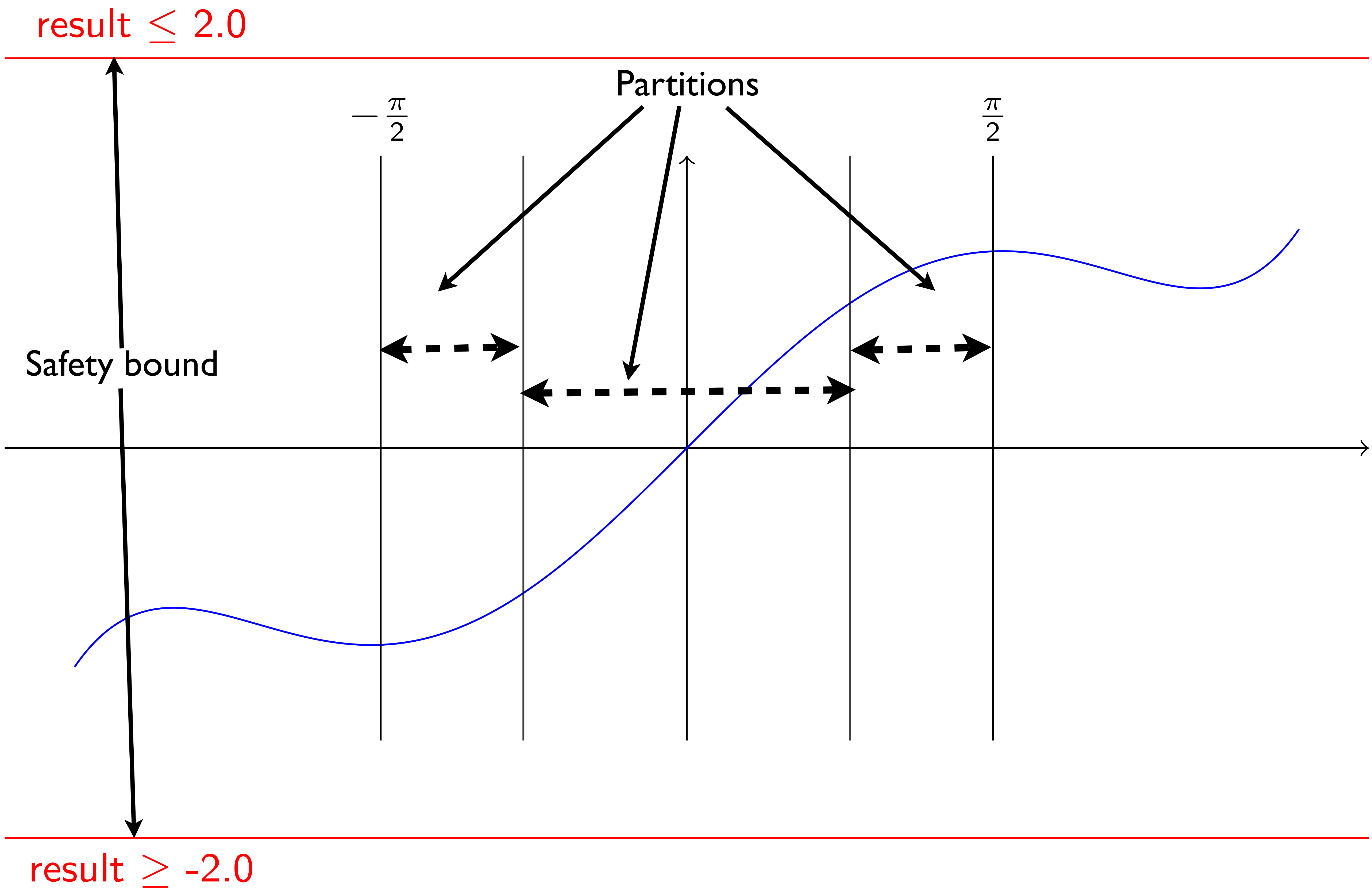
(Astree "spurious" false alarms treated as timeouts)

CDFL on average 260x faster than propositional SAT

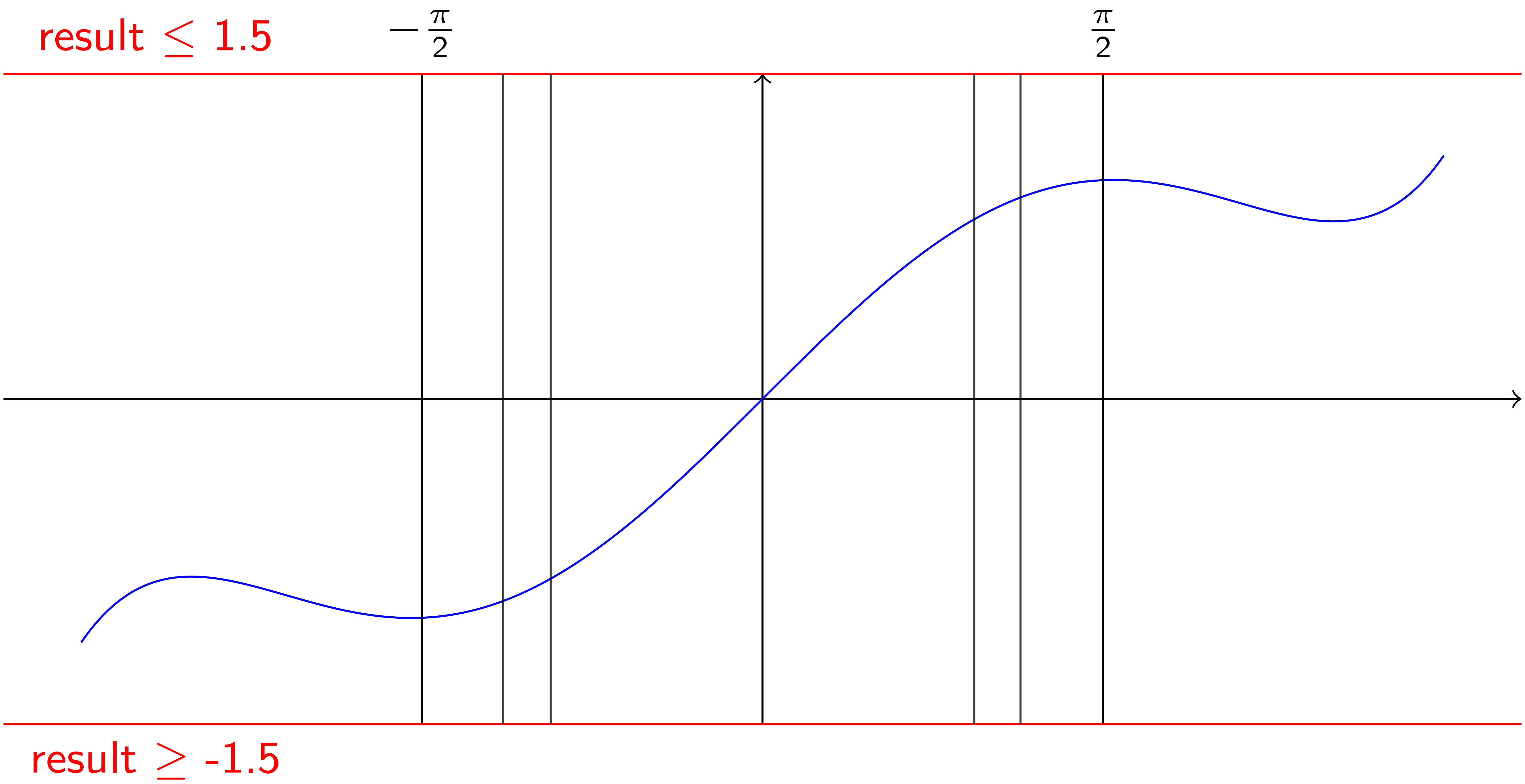
Approximating a Sine Function



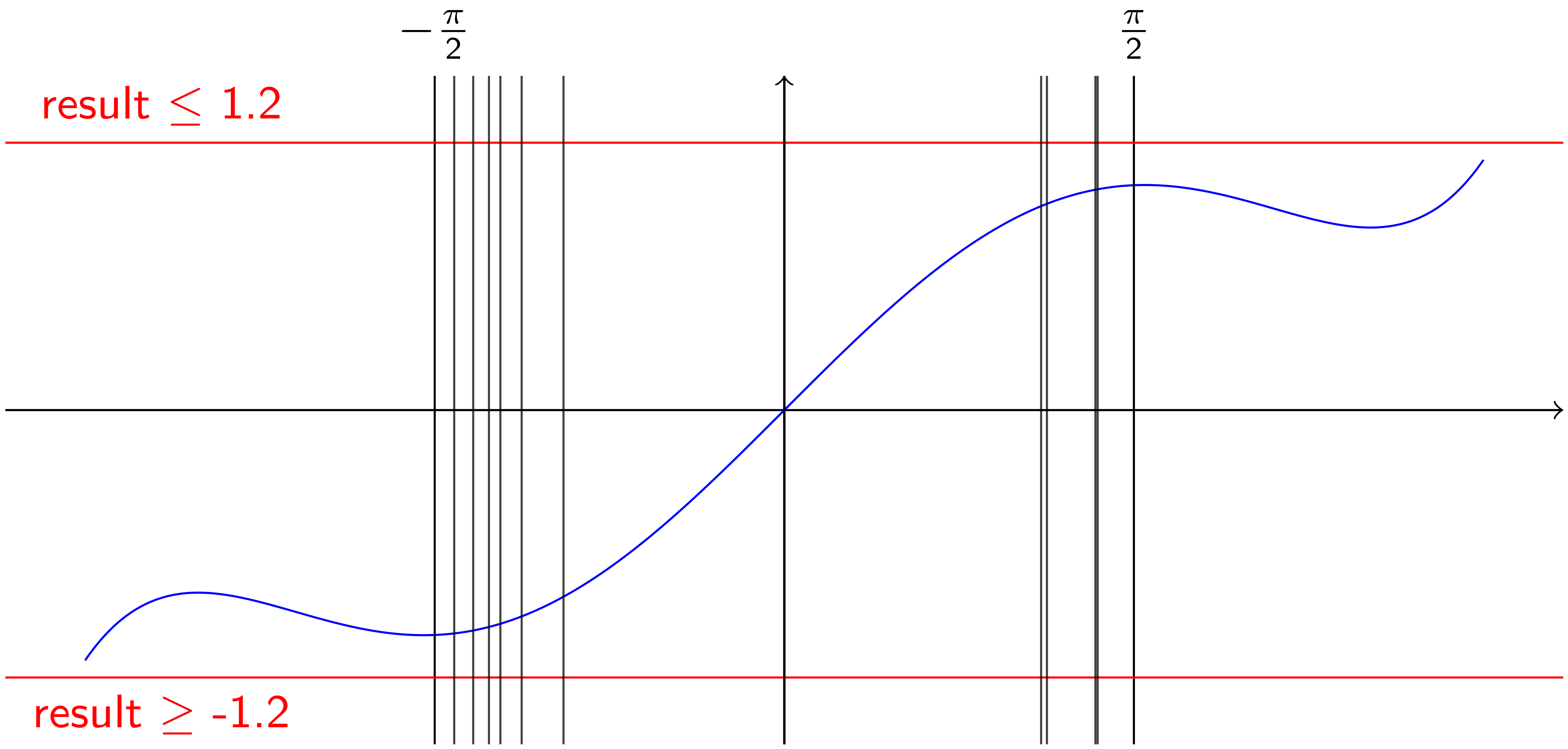
Number of partitions vs. tightness of bound



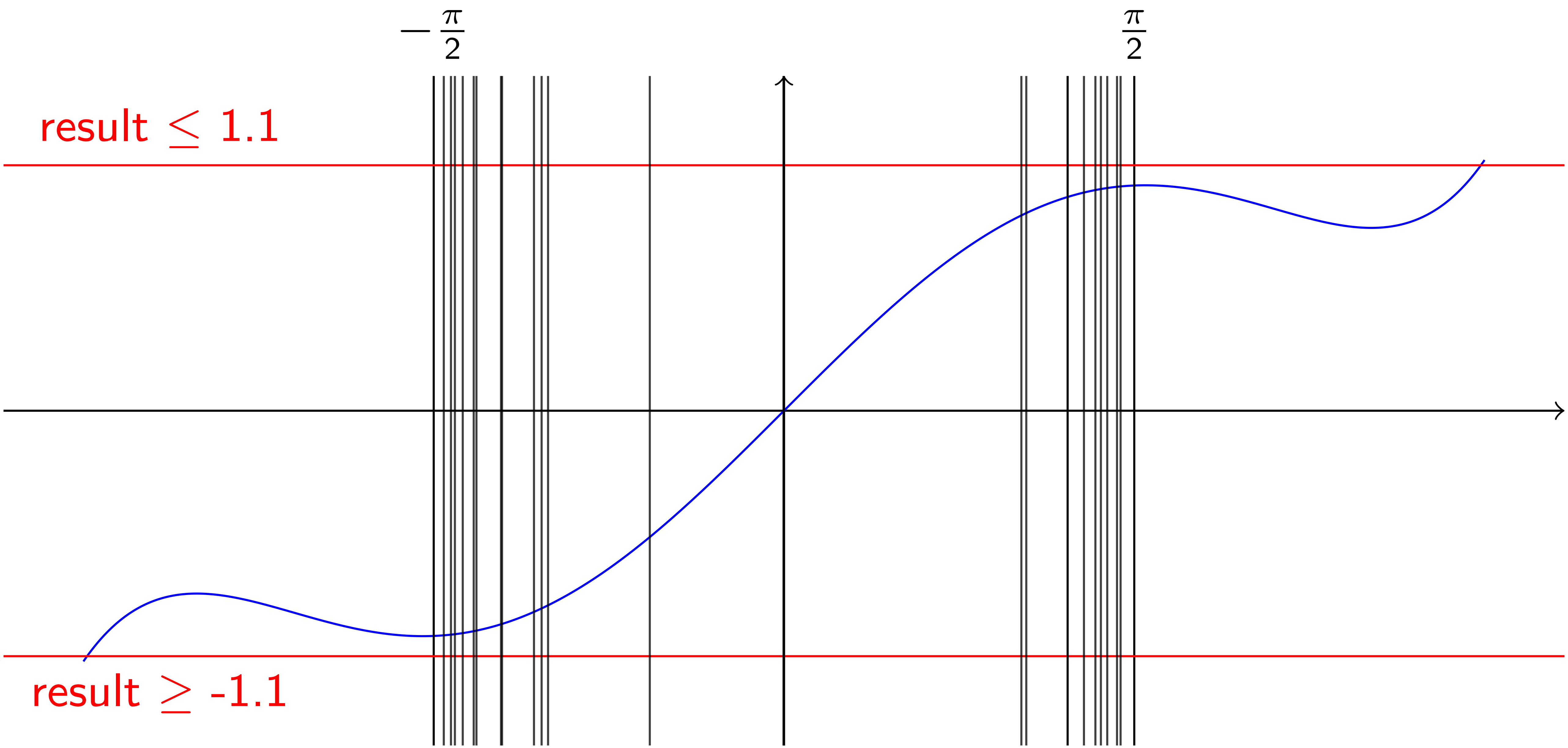
Number of partitions vs. tightness of bound



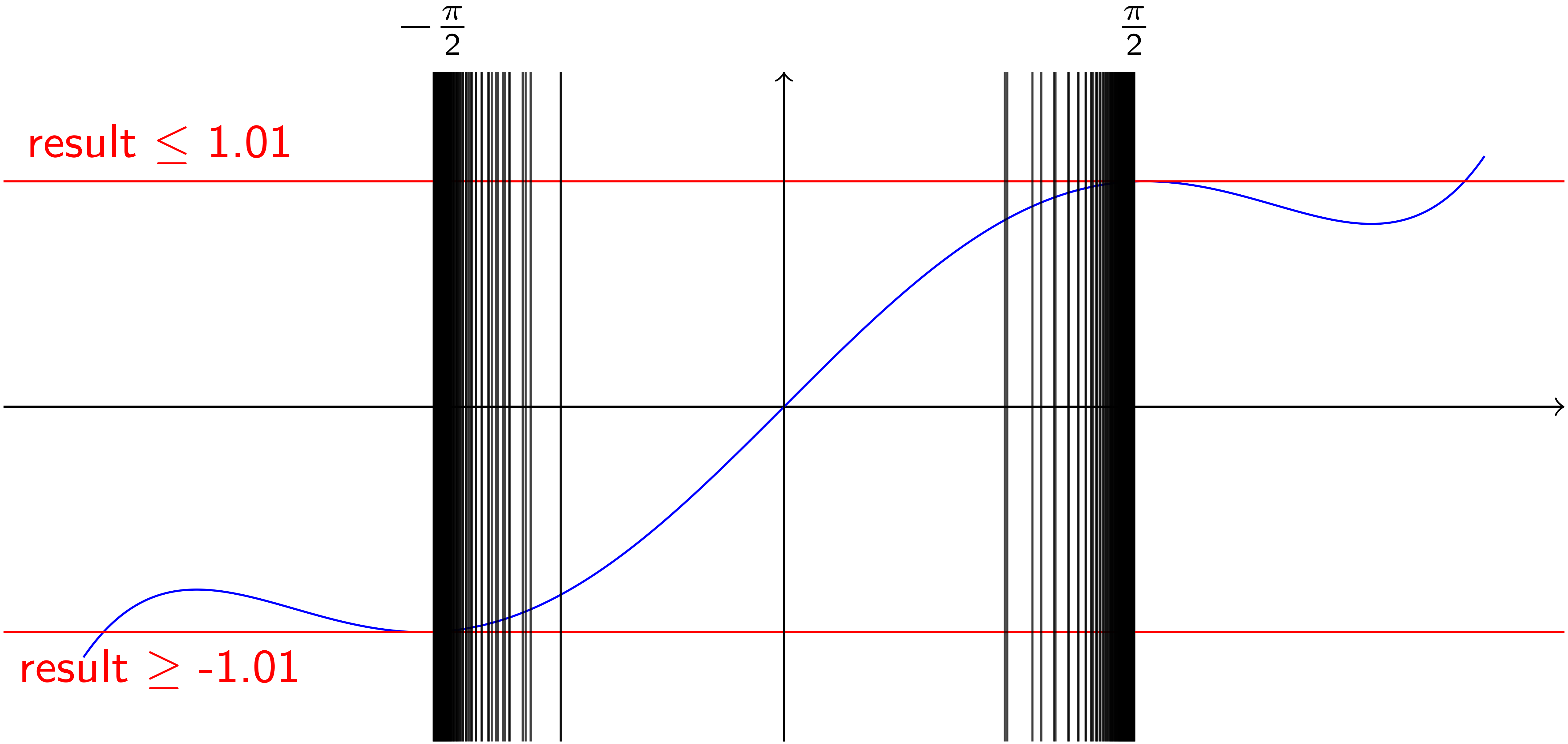
Number of partitions vs. tightness of bound



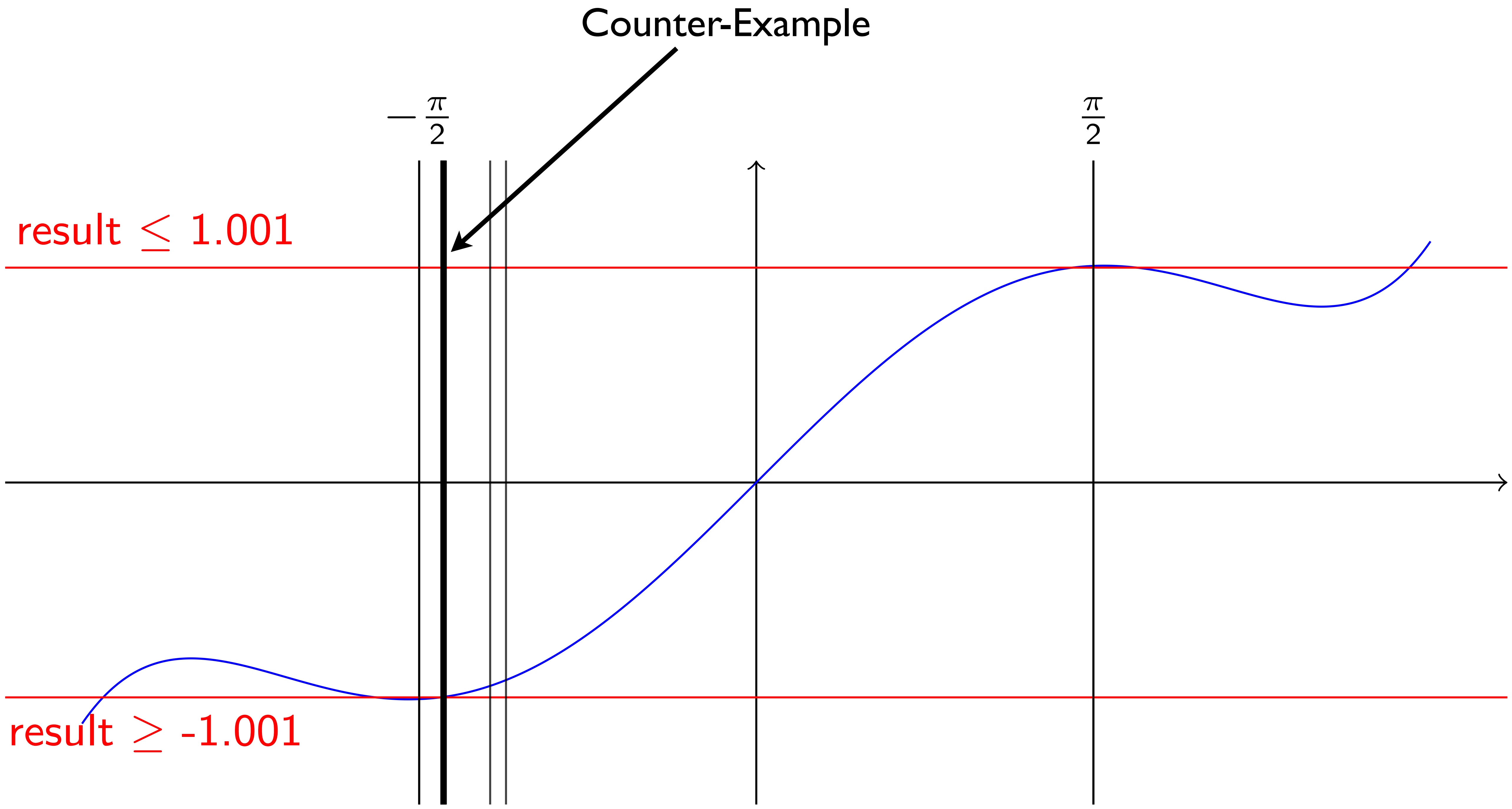
Number of partitions vs. tightness of bound



Number of partitions vs. tightness of bound



Number of partitions vs. tightness of bound

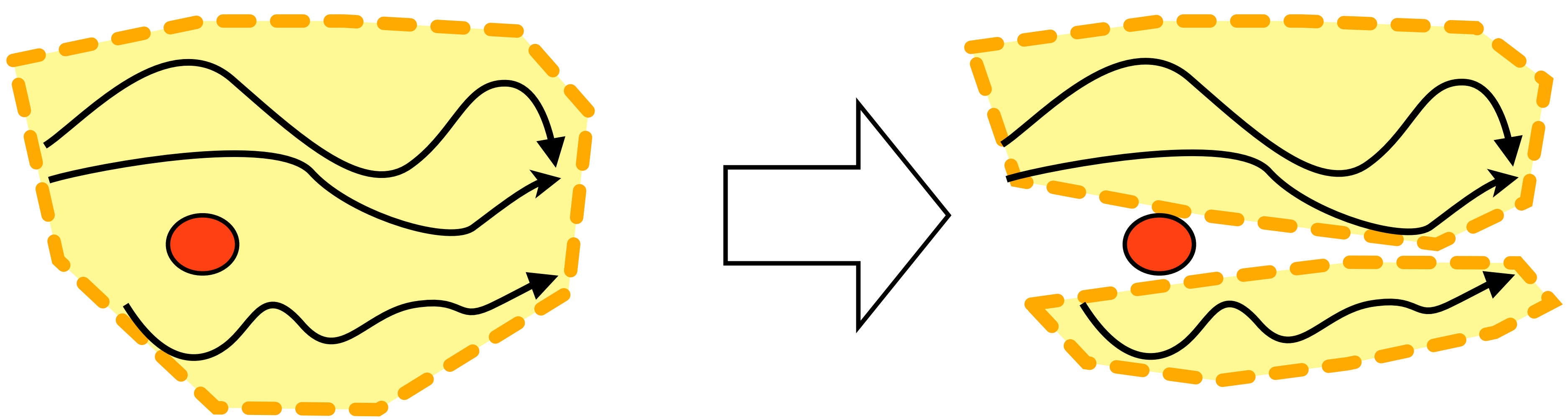


Precise results using a strict abstraction!

Orders of magnitude faster than propositional SAT

Conclusion

- CDFL lifts architecture of a modern SAT solver to abstract domains.
- Property dependent analysis: Analysis is *just precise enough*.
- CDFL(Intervals) significantly outperforms classical CDCL on natural domain problems and is significantly more precise than standard analysis.
- You can probably apply this to your static analysis problem



Thanks for your attention!

Backup Slides

Is it a variant of CEGAR?

	Abstract Domain	Analysis
CEGAR		
CDFL		

Is it a variant of CEGAR?

	Abstract Domain	Analysis
CEGAR	Refined	Fixed
CDFL		

Is it a variant of CEGAR?

	Abstract Domain	Analysis
CEGAR	Refined	Fixed
CDFL	Fixed	Refined

Is it a variant of CEGAR?

	Abstract Domain	Analysis
CEGAR	Refined	Fixed
CDFL	Fixed	Refined

CEGAR finds an abstraction that allows proving a property.

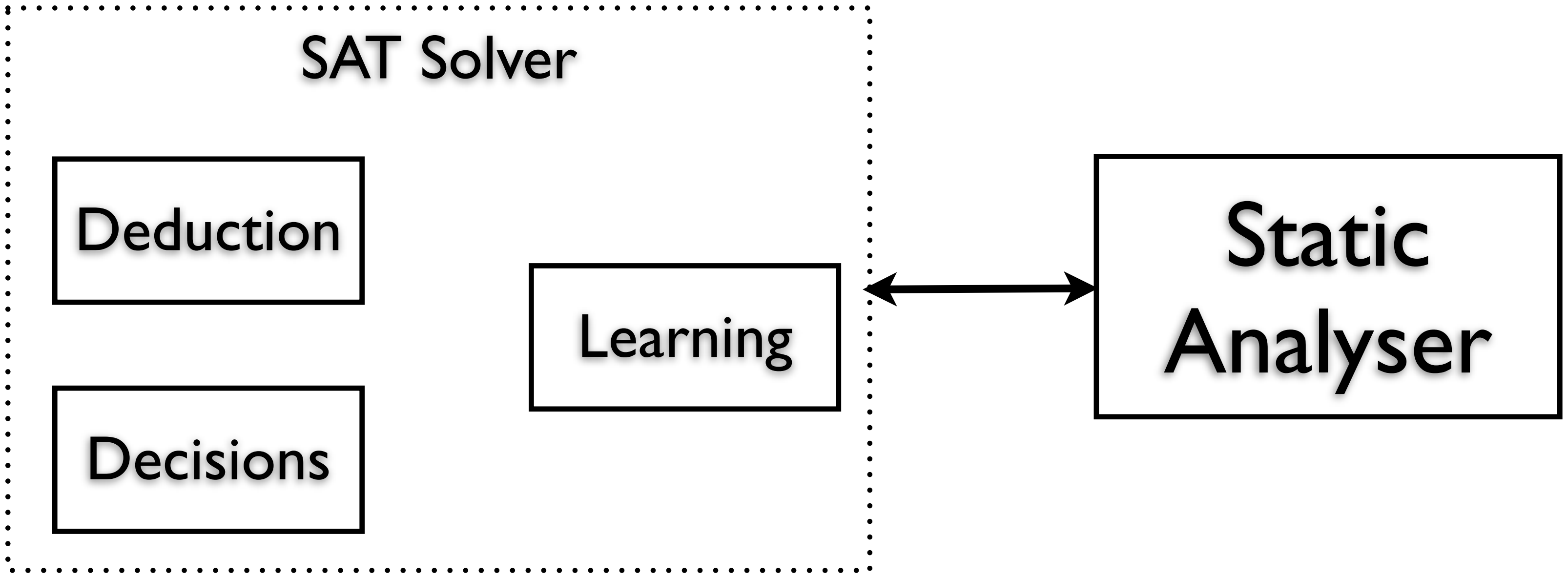
CDFL finds a way to efficiently reason within a fixed abstraction

Orthogonal!

Shallow vs Deep Integration

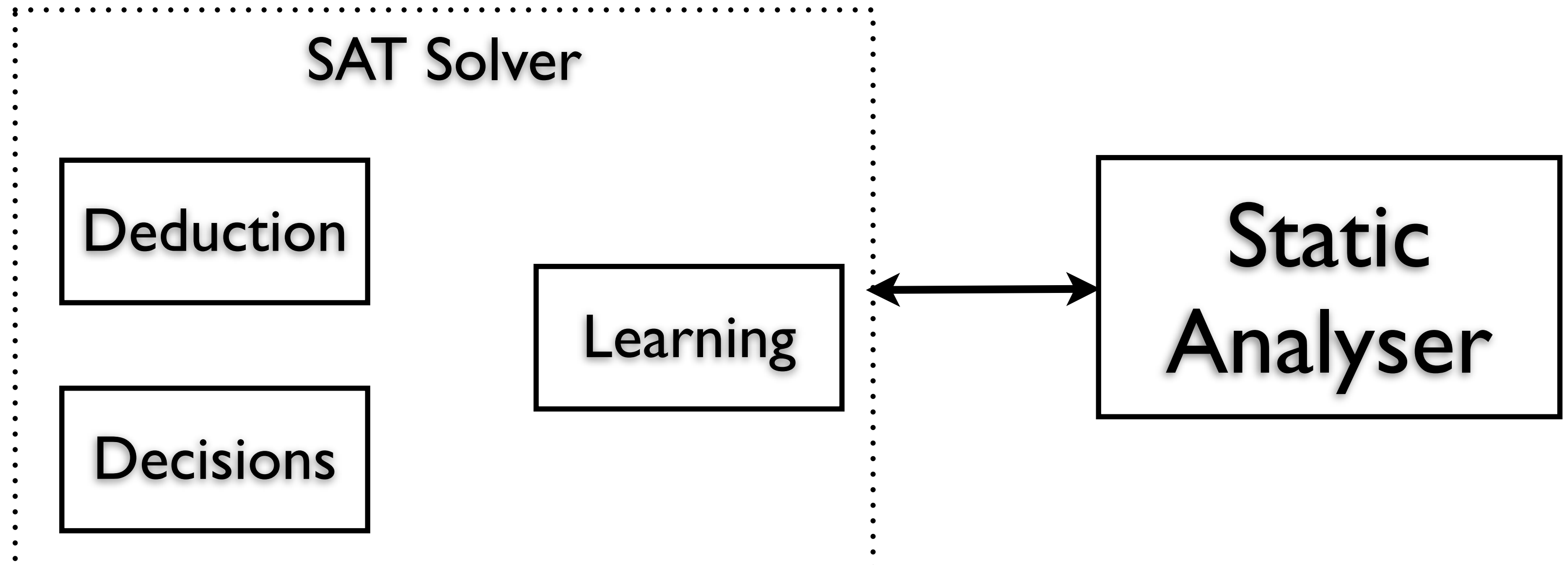
Shallow vs Deep Integration

Shallow Integration
(e.g., SMPP by Harris et al.
at POPL2010)

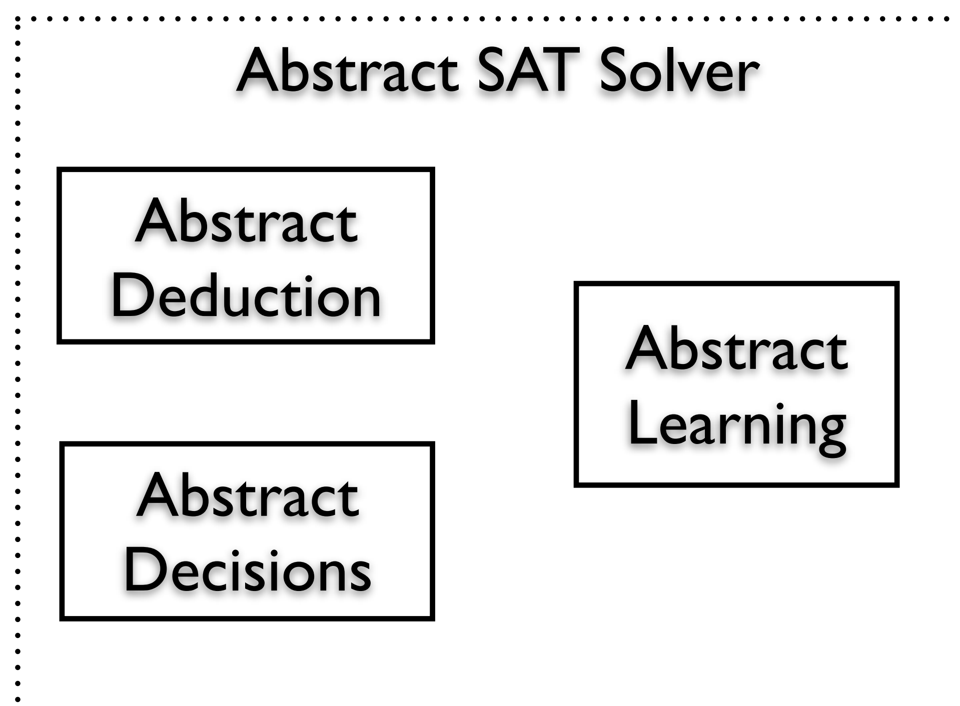


Shallow vs Deep Integration

Shallow Integration
(e.g., SMPP by Harris et al. at POPL2010)



CDFL is deep integration

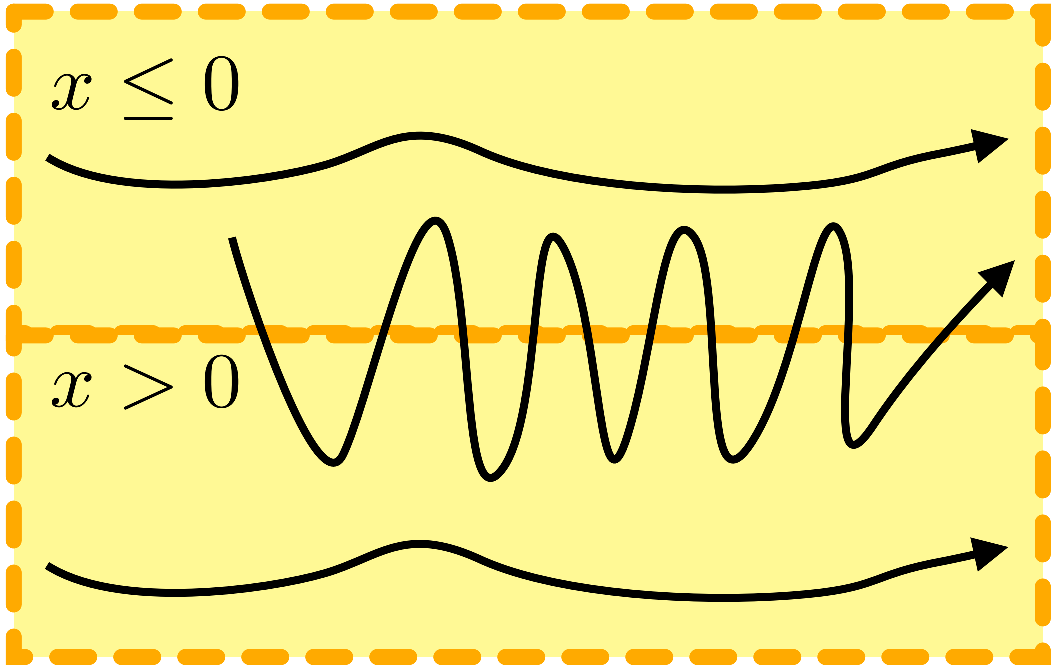


Decisions Inside Loops

```
x = -1;

while(*)
  x = -x;

assert(x != 0);
```



not precisely complementable

Solution:

Use richer abstraction, e.g.,
$\{\text{evenloop}, \text{oddloop}\} \longrightarrow Interval$

```
x = -1;

if(*)
{
  while(*)
  { x = -x; x = -x;}
} else
{
  x = -x;
  while(*)
  { x = -x; x = -x;}
}
```