SAT Modulo Monotonic Theories

Sam Bayless*, Noah Bayless†, Holger H. Hoos*, Alan J. Hu*

*University of British Columbia
†Point Grey Secondary School
Procedural Content Generation
MonoSAT Applications

Circuit Layout

Data Center Allocation

- Switch
- Switch
- Switch
MonoSAT Applications

Finite State Machine Synthesis

CTL Controller Synthesis
MonoSAT

MonoSAT is a SAT Modulo Theory Solver for:

- Graph Predicates:
  - Reachability
  - Shortest paths
  - Maximum $s-t$ flow
  - Minimum Spanning Tree
  - Acyclicity

Collision Detection for Convex Hulls
Finite State Machine String Acceptance
L-Systems, Boolean Geometry, CTL checking (soon)

These are all monotonic theories.

Sam Bayless (UBC)
MonoSAT

MonoSAT is a SAT Modulo Theory Solver for:

- **Graph Predicates:**
  - Reachability
  - Shortest paths
  - Maximum $s - t$ flow
  - Minimum Spanning Tree
  - Acyclicity

- Collision Detection for Convex Hulls
- Finite State Machine String Acceptance
- L-Systems, Boolean Geometry, CTL checking (soon)

These are all *monotonic theories*
A function $p$ is a *Boolean monotonic predicate* iff:

1. $p$ returns a Boolean
2. All arguments are of $p$ are Boolean
3. $p(\ldots, F, \ldots) \implies p(\ldots, T, \ldots)$
Boolean Monotonic Theories

A function $p$ is a *Boolean monotonic predicate* iff:

1. $p$ returns a Boolean
2. All arguments are of type $p$ are Boolean
3. $p(\ldots, F, \ldots) \implies p(\ldots, T, \ldots)$

Definition (Boolean Monotonic Theory)

A theory $T$ with signature $\Sigma$ is Boolean monotonic if and only if:

1. The only sort in $\Sigma$ is Boolean;
2. all predicates in $\Sigma$ are monotonic; and
3. all functions in $\Sigma$ are monotonic.
Graph Constraints in SMMT

A formula over Booleans, edges, and monotonic predicates:

\[(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})\]

And 1 or more symbolic graphs:
Graph Constraints in SMMT

‘Reachability’ is Boolean monotonic:

\[ \text{reaches}_{1,3} \iff \text{False} \]
\[ \text{reaches}_{1,4} \iff \text{False} \]
Graph Constraints in SMMT

‘Reachability’ is Boolean monotonic:

\[ \text{reaches}_{1,3} \rightarrow \text{False} \]
\[ \text{reaches}_{1,4} \rightarrow \text{False} \]
Graph Constraints in SMMT

‘Reachability’ is Boolean monotonic:

\[ \text{reaches}_{1,3} \mapsto \text{True} \]

\[ \text{reaches}_{1,4} \mapsto \text{False} \]
Graph Constraints in SMMT

‘Reachability’ is Boolean monotonic:

```
1 \rightarrow 2
2 \rightarrow 3
3 \rightarrow 1
3 \rightarrow 4
4 \rightarrow 2
```

\[ \text{reaches}_{1,3} \leftrightarrow \text{True} \]
\[ \text{reaches}_{1,4} \leftrightarrow \text{False} \]
‘Reachability’ is Boolean monotonic:

\[
\text{reaches}_{1,3} \rightarrow \text{True}
\]

\[
\text{reaches}_{1,4} \rightarrow \text{True}
\]
Theory Propagation in SMMT

Formula:

\[(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})\]

Assignment:

Underapproximation

Overapproximation

\[\text{reaches}_{1,3}\]

\[\text{reaches}_{1,4}\]
Theory Propagation in SMMT

Formula:

\[(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})\]

Assignment: \(\neg b\)

![Diagram showing underapproximation and overapproximation with nodes 1, 2, 3, 4 and edges a, c, d.]

underapproximation

overapproximation

\(\text{reaches}_{1,3}\)

\(\text{reaches}_{1,4}\)
Theory Propagation in SMMT

Formula:

\[(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})\]

Assignment: \(\neg b, a\)

Underapproximation

Overapproximation

\(\text{reaches}_{1,3}\)

\(\text{reaches}_{1,4}\)
Theory Propagation in SMMT

Formula:

$$(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})$$

Assignment: $\neg b, a, c$

Underapproximation

Overapproximation

$\text{reaches}_{1,3} \rightarrow \text{True}$

$\text{reaches}_{1,4}$
Theory Propagation in SMMT

Formula:

\[(a \lor \neg b) \land (b \lor c) \land (\neg c \lor \neg d) \land (\text{reaches}_{1,3} \lor \text{reaches}_{1,4})\]

Assignment: \(\neg b, a, c, \neg d\)

Underapproximation

Overapproximation

\[\text{reaches}_{1,3} \mapsto \text{True}\]

\[\text{reaches}_{1,4} \mapsto \text{False}\]
Theory propagation in SMMT has useful properties:

- Easy to implement.
- Can use off-the-shelf algorithms.
- Improved worst-case clause learning.
MonoSAT Applications: Shortest Path Constraints
MonoSAT Applications: Maximum Flow Constraints

Maximum Flow Runtime (s)

- 8x8 16Flow
- 16x16 8Flow
- 16x16 16Flow
- 16x16 24Flow
- 24x24 16Flow
- 32x32 16Flow

- MonoSAT
- clasp
- MiniSat
## MonoSAT Applications: Convex Hull Containment

<table>
<thead>
<tr>
<th>Art Gallery Synthesis</th>
<th>MonoSAT</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 points, 3 hulls, ≤3 cameras</td>
<td>2s</td>
<td>7s</td>
</tr>
<tr>
<td>20 points, 4 hulls, ≤4 cameras</td>
<td>36s</td>
<td>433s</td>
</tr>
<tr>
<td>30 points, 5 hulls, ≤5 cameras</td>
<td>187s</td>
<td>&gt; 3600s</td>
</tr>
<tr>
<td>40 points, 6 hulls, ≤6 cameras</td>
<td>645s</td>
<td>&gt; 3600s</td>
</tr>
<tr>
<td>50 points, 7 hulls, ≤7 cameras</td>
<td>3531s</td>
<td>&gt; 3600s</td>
</tr>
</tbody>
</table>

![Diagram of convex hull containment](image)
Monotonic theories have many applications.
Building SMT solvers for them is easy.
MonoSAT supports many graph properties (and more!), and it is free & open-source:
  ▶ New! Bit vector support,
  ▶ New! Python support.

Website: cs.ubc.ca/labs/isd/Projects/monosat

GitHub: github.com/sambayless/monosat
from monosat import *

a = Var()
b = Var()
c = Or(a, Not(b))

Assert(c)

result = Solve()
```python
from monosat import *
g = Graph()
e1 = g.addEdge(1, 2)
e2 = g.addEdge(2, 3)
e3 = g.addEdge(1, 3)

Assert(Not(And(e1, e3)))

Assert(g.reaches(1, 3))

result = Solve()
```
from monosat import *
g= Graph()
e1 = g.addEdge(1,2)
e2 = g.addEdge(2,3)
e3 = g.addEdge(1,3)

Assert(Or(g.reaches(1,3),
        g.distance_leq(1,3,2)))

result = Solve()
from monosat import *
g = Graph()
bv1 = BitVector(4)
bv2 = BitVector(4)
bv3 = BitVector(4)
e1 = g.addEdge(1, 2, bv1)
e2 = g.addEdge(2, 3, bv2)

Assert(g.distance_leq(1, 3, bv3))
Assert(Not(g.distance_lt(1, 3, bv3)))

Assert((bv1 + bv3) == 9)

result = Solve()