A Tree-based Modular SMT Solver

Georg Schadler
Georg Hofferek
Outline

• Motivation
• Proof structure, requirements & properties
• Implementation & example
• Theory checks & interpolation
• Outlook & conclusion
Motivation
Motivation

- Synthesize **multiple Boolean control signals** e.g. for a pipelined processor.

- Specification given as a **quantified first-order formula**.

- Uninterpreted functions to abstract specification
Abstract Quantified Formula

\[ \Psi = \forall \text{inputs, states} . \exists \text{controls signals} . \forall \text{auxvars} . \Phi \]

- stuff that choice of control signals depends on
- correctness axiom
- stuff that control signals do not depend on
- unknown $\rightarrow$ to be synthesized

Unknown $\rightarrow$ To be synthesized
Formula Expansion

\[ \forall \vec{a} \exists c_0, c_1 \forall \vec{b} . \Phi (\vec{a}, \vec{b}, c_0, c_1) = \top \]

• Expansion of \( \exists \)
• Renaming of \( \vec{b} \)
• Negation

\[ \neg \Phi (\vec{a}, \vec{b}_{00}, 0,0) \land \Phi (\vec{a}, \vec{b}_{10}, 1,0) \land \Phi (\vec{a}, \vec{b}_{01}, 0,1) \land \Phi (\vec{a}, \vec{b}_{11}, 1,1) = \bot \]

“Partitions”: \( \phi_{00}, \phi_{01}, \phi_{10}, \phi_{11} \)
Motivation Recap

1. Construct unsatisfiable SMT formula from specification and compute proof
2. Craig interpolation to compute multiple coordinated interpolants.
3. Interpolants implement the Boolean control signals.

More Information:
Hofferek et al., FMCAD 2013
Hofferek & Bloem, MEMOCODE 2011
Refutation Proof
• Proof requires two properties:
  
  • **Local-first**
    Local literals are resolved before global literals
  
  • **Colorable**
    No literals or leaves with symbols from two partitions
Proof Requirements

\[ \neg \varphi (\vec{a}, \vec{b}_0, 0,0) \land \neg \varphi (\vec{a}, \vec{b}_1, 1,0) \land \neg \varphi (\vec{a}, \vec{b}_2, 0,1) \land \neg \varphi (\vec{a}, \vec{b}_3, 1,1) \]

Local Literals 0

Local Literals 1

Local Literals 2

Local Literals 3

Global Literals

\[ I_0 \downarrow I_1 \]
Proof Requirements

\[ \neg \varphi (\vec{a}, \vec{b}_0, 0, 0) \land \neg \varphi (\vec{a}, \vec{b}_1, 1, 0) \land \neg \varphi (\vec{a}, \vec{b}_2, 0, 1) \land \neg \varphi (\vec{a}, \vec{b}_3, 1, 1) \]

Certificates (computed with interpolation) implement control signals

Cf. Pudlak’s Interpolation Procedure (JSL’97)
Colorability

Partitions \approx \text{Colors}:
\neg \Phi_{00}(\tilde{a}, \tilde{b}_{00}) \land \neg \Phi_{10}(\tilde{a}, \tilde{b}_{10}) \land \neg \Phi_{01}(\tilde{a}, \tilde{b}_{01}) \land \neg \Phi_{11}(\tilde{a}, \tilde{b}_{11})

Local Symbols: \tilde{b}_{00}, \tilde{b}_{10}, \tilde{b}_{01}, \tilde{b}_{11} \text{ (colored)}
Global Symbols: \hat{a} \text{ (colorless)}

Colorable: (x = y), (u = v), (w = z)
Non-colorable: (x = u)
Colorability

\[
\Phi_{00} \quad \Phi_{01} \quad \Phi_{10} \quad \Phi_{11}
\]

\[
\overline{x}_0' \quad \overline{x}_0' \quad \overline{x}_1' \quad \overline{x}_1'
\]

\[
\overline{\mathcal{X}}
\]

No literals or leaves with symbols from two partitions
Implementation
Implementation

Control signals can depend on inputs that are independent from each other

\[ \forall \tilde{a} \ \exists \tilde{c} \ \forall \tilde{a}' \ \exists \tilde{c}' \ \forall \tilde{a}'' \ \exists \tilde{c}'' \ldots \Phi \]

- 1 level per \( \forall \ \exists \) - alternation
- \( 2^{\mid \tilde{a} \mid} \) nodes per level
Implementation

Input Partitions $\phi_i$
Implementation

Extension of modular SAT
Bayles et al., (FMCAD 2013)

Input Partitions $\phi_i$

$\phi_0$  $\phi_1$  $\phi_2$  $\phi_3$
Example
Example

\[ a \lor b \]
\[ \neg c \]
\[ a \lor c \]
\[ e \]
\[ \neg a \lor \neg d \]
\[ \neg a \lor \neg d \]
\[ \neg d \]
Example

Theory is abstracted as propositional logic

Abstracted Theory Literals
Example

\[
a \lor b \\
\neg c
\]

\[
a \lor c \\
e
\]

\[
\neg a \lor \neg d
\]

\[
\neg a \lor \neg d \\
\neg d
\]
Example

Assign **Global** Symbols to **lowest common ancestor** for **Local-First** property

```
\( a \lor b \)
\( \neg c \)

\( a \lor c \)
\( e \)

\( \neg a \lor \neg d \)

\( \neg a \lor \neg d \)
\( \neg d \)
```
Example

Assign **Global** Symbols to **lowest common ancestor** for **Local-First** property.
Assign **Global** Symbols to **lowest common ancestor** for **Local-First** property.
Example

Assign **Global** Symbols
to **lowest common ancestor**
for **Local-First** property

\[
\begin{align*}
& \text{SOLVER} \\
& a \\
& \text{SOLVER} \\
& \text{SOLVER} \\
& a \lor b \\
& \neg c
\end{align*}
\]

\[
\begin{align*}
& \text{SOLVER} \\
& a \lor c \\
& \text{SOLVER} \\
& \neg a \lor \neg d \\
& e
\end{align*}
\]

\[
\begin{align*}
& \text{SOLVER} \\
& a \\
& \text{SOLVER} \\
& \text{SOLVER} \\
& \neg a \lor f \\
& \neg d
\end{align*}
\]
Example

Assign **Global** Symbols to **lowest common ancestor** for **Local-First** property
Example

\[ a \lor b \]

\[ \neg c \]

\[ a \lor c \]

\[ e \]

\[ \neg a \lor \neg d \]

\[ \neg f \]

\[ \neg d \]
Example

Every node can only decide their „own“ symbols

A Tree-based Modular SMT Solver
Georg Schadler
Example

Every node can only decide their „own“ symbols
Example

Every node can only decide their „own“ symbols
Example

Every node can only decide their „own“ symbols
Every node can only decide their „own“ symbols
Example

Every node can only decide their „own“ symbols
Example

Every node can only decide their „own“ symbols

\[ a \lor b \lor \neg c \lor a \lor \neg a \lor \neg d \lor \neg a \lor f \lor \neg d \]
Example

\[ a \lor b \]
\[ \neg c \]
\[ \text{UNSAT} \]

\[ a \lor c \]
\[ \text{SAT} \]

\[ a \lor \neg d \]

\[ a \lor \neg f \]
\[ \neg d \]
Example
Example

A Tree-based Modular SMT Solver

Georg Schadler
Example

```
\begin{align*}
\text{SOLVER} & \quad a \\
\text{SOLVER} & \quad a \lor b \\
\text{SOLVER} & \quad \neg c \\
\text{SOLVER} & \quad a \lor c \\
\text{SOLVER} & \quad e \\
\text{SOLVER} & \quad \neg a \lor \neg d \\
\text{SOLVER} & \quad \neg a \lor f \\
\text{SOLVER} & \quad \neg d
\end{align*}
```
Example

\[ a \lor b \]
\[ \neg c \]
Example
Example

\[ \neg a \lor \neg d \]

\[ \neg a \lor \neg d \]

\[ a \lor c \]

\[ a \lor b \]

\[ a \lor b \]

\[ c \]

\[ c \]

\[ e \]

\[ e \]
Example

\[ a \lor b \]
\[ \neg c \]
\[ a \lor c \]
\[ \neg a \lor \neg d \]
\[ \neg a \lor f \]
\[ \neg d \]
Example
Example

A Tree-based Modular SMT Solver

Georg Schadler
Example

\[
\begin{align*}
\text{SAT} & \quad (a \lor b) \land \neg c \\
\text{SAT} & \quad (a \lor c) \land e \\
\text{SAT} & \quad (a \lor \neg d) \\
\text{SAT} & \quad (a \lor f) \land \neg d
\end{align*}
\]
Example

Theory check of conjunction

\[ a, b, c \land a, c, e \]

Theory consistent?

\[ a, b, c \land a, c, e \]

\[ \neg a \lor \neg d \]

\[ a \lor c \]

\[ a \lor b \]

\[ \neg c \]

\[ \neg a \lor \neg f \]

\[ \neg d \]

\[ a \]

\[ c \]

\[ e \]

\[ d \]
Theory Check

\[ x = a \land a = z \]

\[ (x = b) \land (b \neq z) \]
Theory Check

\((x = a) \land (a = z) \land (x = b) \land (b \neq z)\)

\(x = a \land a = z\)

\((x = b) \land (b \neq z)\)

A Tree-based Modular SMT Solver

Georg Schadler
Theory Check

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]
Theory Check

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]

Theory Inconsistent!

Because \[(x = z) \land (x \neq z)\]

Possible Solution: Blocking clause \[(a \neq b)\]

BUT: Blocking clause \[(a \neq b)\] is not colorable!
Craig Interpolation

$$CNF(\Phi) = C_1 \land C_2 \land C_3 \land \cdots \land C_{n-1} \land C_n = \bot$$

**Interpolant $I$:**

- $A \rightarrow I$

- $I \rightarrow \neg B$, in other words: $I \land B = \bot$

- $V(I) \subseteq V(A) \cap V(B)$

**Interpolant contains only global symbols.**
Interpolation

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]

Interpolate

SOLVER

SOLVER

SOLVER
Interpolation

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]

Interpolant \(I_p\) : \(x = z\)
Interpolation

\((x = a) \land (a = z) \land (x = b) \land (b \neq z)\)

\(I_p: x = z\)
Interpolation

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]

\[I_p: \ x = z\]
Interpolation

\[(x = a) \land (a = z) \land (x = b) \land (b \neq z)\]

\[I_p: \ x = z\]

No literals from different partitions in blocking clauses

→ Proof always colorable
Proof production

Root node resolves only over global literals

Premises of proof in root node are proofs of child nodes
Current State & Outlook

- Prototype implemented („Proof of concept“) with MiniSat + MathSAT

- Relatively good runtime but much optimisation potential...

- Currently implementing proof production.
Conclusion

• Modular SMT Solving
  • Colorable and local-first proof directly from SMT solver.
  • Possible for all theories with interpolants in same theory.
• Craig Interpolation
  • Produces colorable blocking clauses
  • Multiple coordinated interpolants from just one proof
• Therefore the world is now a slightly better place 😊
Thank You!

Questions?
A Tree-based Modular SMT Solver

Georg Schadler

Institute for Applied Information Processing and Communications

Secure & Correct Systems

Thank You!

Questions?
Appendix
Specification
Specification

Correctness: First-Order Logic Formula $\Phi$

Important Building Blocks:

- Array Variables
  - Addressable Memories
- Uninterpreted Functions & Predicates
  - Combinational Circuits
- Domain Variables
  - Single Element Storage
  - Primary Inputs/Outputs
Certificate via Interpolation
Certificate via Interpolation

\[ \Psi = \forall \text{mem, reg, pipelinestate}. \]
\[ \exists \text{stall, forward}. \]
\[ \forall \text{mem}', \text{reg}', \text{pipelinestate}'. \Phi \]

- **stall, forward**: Boolean control signals
- **mem, reg, pipelinestate**: Uninterpreted domain

Compute **Certificates**:

\[ (\text{stall, forward}) = f(\text{mem, reg, pipelinestate}) \]
Certificate via Interpolation

- $\Psi = \forall \hat{a}. \ \exists \hat{c}. \ \forall \hat{b}. \ \Phi(\hat{a}, \hat{b}, \hat{c})$
  - $\Psi$ is valid

- Function $\hat{c} = \sigma(\hat{a})$

- Such that: $\Phi(\hat{a}, \hat{b}, \sigma(\hat{a}))$ is valid
Certificate via Interpolation

\[ \neg \Phi(\tilde{a}, 0, \tilde{b}_0) \land \neg \Phi(\tilde{a}, 1, \tilde{b}_1) = \bot \]

- \( A \)
  - 0 not allowed
- \( B \)
  - 1 not allowed

**Interpolant** \( I(\tilde{a}) \):

- \( \neg \Phi(\tilde{a}, 0, \tilde{b}_0) \rightarrow I \)
  - \( I \) is 1, whenever 0 not allowed
- \( I \rightarrow \Phi(\tilde{a}, 1, \tilde{b}_1) \)
  - Whenever \( I \) is 1, 1 is allowed

Boolean Case: see Jiang et al., ICCAD’09
Sample Application
A Processor

Tough:
64-bit datapath
very complex arithmetic logic unit

How do I pipeline that?
A Pipelined Processor

IF → DE → EX → MEM → WB

That’s trivial!
A Pipelined Processor

Instructions:
- \( r_1 := \text{mem}[1] \)
- \( r_2 := r_1 + r_2 \)

\[ \begin{align*}
  r_1 &= 1 \\
  r_2 &= 2
\end{align*} \]

\[ \begin{align*}
  \text{mem}[1] &= 15 \\
  r_1 &= \text{mem}[1] \\
  r_2 &= r_1 + r_2 \\
  r_1 &= \text{mem}[1] \\
  r_2 &= \text{mem}[1] \\
  r_1 &= 15 \\
  \text{mem}[1] &= 15 \\
  r_2 &= r_1 + r_2 \\
  r_1 &= \text{mem}[1] \\
  r_2 &= 15 + 2 \\
  r_2 &= 17 \\
  r_1 &= \text{mem}[1] \\
  r_2 &= 17
\end{align*} \]
A Pipelined Processor

- Hard to test
- Hard to implement
- Easy to specify → Burch-Dill paradigm

Not so trivial!
Craig Interpolation
Craig Interpolation

\[ \text{CNF}(\Phi) = C_1 \land C_2 \land C_3 \land \cdots \land C_{n-1} \land C_n = \bot \]

Interpolant \( I \):

- \( A \to I \)
- \( I \to \neg B \), in other words: \( I \land B = \bot \)
- \( V(I) \subseteq V(A) \cap V(B) \)