On Craig Interpolation in SMT

Philipp Rümmer

University of Regensburg
Uppsala University

2024-04-22
CIBD Workshop, Amsterdam

Outline

- Craig interpolation in verification
- Summary of some interpolation results for theories
- SMT solvers supporting Craig interpolation
- Beyond binary interpolation

Motivation: inference of invariants

Generic verification problem ("safety")

$$\{ \texttt{pre} \} \texttt{ while (*) Body } \{ \texttt{post} \}$$

Standard approach: loop rule using invariant

$$\frac{\texttt{pre} \Rightarrow \phi \qquad \{ \phi \} \texttt{ Body } \{ \phi \} \qquad \phi \Rightarrow \texttt{post}}{\{ \texttt{pre} \} \texttt{ while (*) Body } \{ \texttt{post} \}}$$

How to compute ϕ automatically?

From intermediate assertions to invariants

$\{\texttt{pre}\}$ Body; Body $\{\texttt{post}\}$?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_1

↓

$\{\texttt{pre}\}$ Body $\{\psi_1\}$ $\{\psi_1\}$ Body $\{\texttt{post}\}$

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre}\}$ Body; Body $\{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_1

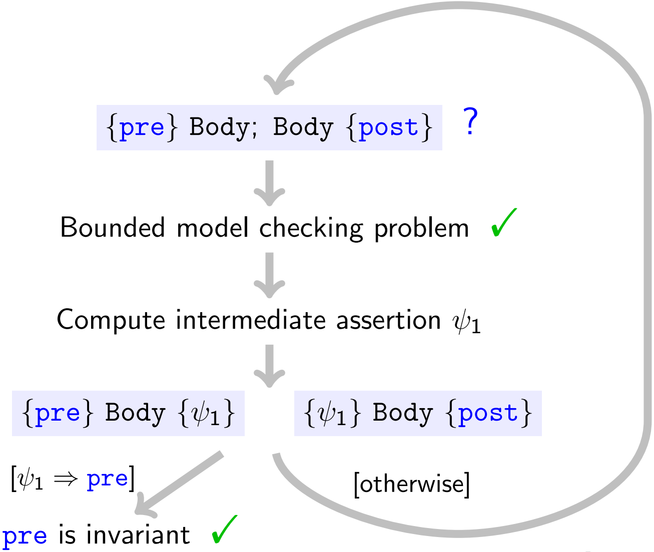
$\{\texttt{pre}\}$ Body $\{\psi_1\}$ $\{\psi_1\}$ Body $\{\texttt{post}\}$

$[\psi_1 \Rightarrow \texttt{pre}]$

pre is invariant ✓

[McMillan, 2003]

From intermediate assertions to invariants



[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \vee \psi_1\}\ \texttt{Body};\ \texttt{Body}\ \{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_2

$\{\texttt{pre} \vee \psi_1\}\ \texttt{Body}\ \{\psi_2\}$ $\{\psi_2\}\ \texttt{Body}\ \{\texttt{post}\}$

$[\psi_1 \Rightarrow \texttt{pre}]$ [otherwise]

$\texttt{pre}$ is invariant ✓

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \lor \psi_1\}$ Body; Body $\{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_2

$\{\texttt{pre} \lor \psi_1\}$ Body $\{\psi_2\}$ $\{\psi_2\}$ Body $\{\texttt{post}\}$

$[\psi_2 \Rightarrow \texttt{pre} \lor \psi_1]$

$\texttt{pre} \lor \psi_1$ is invariant ✓

[otherwise]

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \vee \psi_1\}$ Body; Body $\{\texttt{post}\}$?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_2

↓

$\{\texttt{pre} \vee \psi_1\}$ Body $\{\psi_2\}$ $\{\psi_2\}$ Body $\{\texttt{post}\}$

$[\psi_2 \Rightarrow \texttt{pre} \vee \psi_1]$ ...

$\texttt{pre} \vee \psi_1$ is invariant ✓

[McMillan, 2003]

How to compute intermediate assertions?

VC generation

{ pre }	pre (s_0)
Body;	$\to$ Body (s_0, s_1)
Body	$\to$ Body (s_1, s_2)
{ post }	$\to$ post (s_2)

How to compute intermediate assertions?

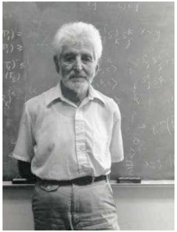
VC generation

{ pre }	$\texttt{pre}(s_0)$
Body;	$\to \texttt{Body}(s_0, s_1)$
Body	$\to \texttt{Body}(s_1, s_2)$
{ post }	$\to \texttt{post}(s_2)$

Theorem (Craig, 1957)

Suppose $A \to C$ is a valid implication. A formula I is called a Craig interpolant if

- *$A \to I$ and $I \to C$ are valid,*
- *every non-logical symbol of I occurs in both A and C.*



How to compute intermediate assertions?

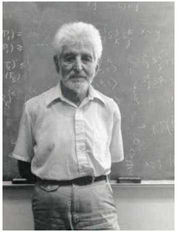
VC generation

{ `pre` }	`pre` (s_0)	$A(s_0, s_1)$
`Body;`	$\rightarrow$ `Body` (s_0, s_1)	$\downarrow$
....	....	$I(s_1)$
`Body`	$\rightarrow$ `Body` (s_1, s_2)	$\downarrow$
{ `post` }	$\rightarrow$ `post` (s_2)	$C(s_1, s_2)$

Theorem (Craig, 1957)

Suppose $A \rightarrow C$ is a valid implication. A formula I is called a Craig interpolant if

- *$A \rightarrow I$ and $I \rightarrow C$ are valid,*
- *every non-logical symbol of I occurs in both A and C.*



Abstraction with interpolants

{pre} Body; Body {post} ?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_1

↓

...

Abstraction with interpolants

{pre} Body; Body {post} ?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_1

↓

...

Interpolant extracted from proof
⇒
Abstraction from unnecessary details

Theories

- Following [McMillan 2003], several solvers and theorem provers add interpolation support:
  - SAT solvers
  - Foci $\rightarrow$ iZ3 $\rightarrow$ Z3
  - MathSAT
  - CLPprover
  - CSIsat
  - OpenSMT
  - Princess
  - SMTInterpol
  - Vampire
  - AXDInterpolator
  - *etc.*
- "Race" to find interpolation procedures for relevant theories.

Standard SMT theories

- EUF
- Arrays
- LRA
- LIA
- NRA
- NIA
- Bit-vectors
- Floats
- ADTs
- Strings
- *(+ combinations)*

Towards Satisfiability Modulo Theories paradigm (SMT)

- Satisfiability Modulo Theories (SMT) solvers are today the standard backends in verification
- Maintained solvers supporting Craig interpolation:

Solver	...
MathSAT5	
OpenSMT2	
Princess	
SMTInterpol	
cvc5	
Vampire	
Z3	

- *(any tools missing?)*

Reverse interpolants

- It is common in verification to use the following variant of interpolation:

Definition

Suppose $A \wedge B$ is unsatisfiable. A *reverse interpolant* is a formula I such that

- $A \rightarrow I$ and $B \rightarrow \neg I$ are valid,
- every non-logical symbol of I occurs in both A and B.

Lemma

In classical logic, reverse interpolants and ordinary interpolants are interchangeable:

$$I \text{ is reverse interpolant for } A \wedge B \quad \Longleftrightarrow \quad I \text{ is interpolant for } A \rightarrow \neg B$$

Interpolation in theories

Theorem (Kovacs, Voronkov, 2009)

Suppose T is a theory and $A \wedge B$ a T-unsatisfiable conjunction in first-order logic:

$$A \wedge B \models_T \mathit{false}$$

Then there is a formula I such that:

- $A \models_T I$
- $B \models \neg I$
- *every non-logical symbol ...*

- *Problem:* even if $A \wedge B$ is quantifier-free, the I might contain quantifiers.
- Often a problem in verification.

Plain quantifier-free theory interpolation

Definition (Bruttomesso, Ghilardi, Ranise, 2014)

A theory T admits *plain quantifier-free interpolation* if for every quantifier-free T-unsatisfiable conjunction $A \wedge B$ (with arbitrary free variables, but otherwise only containing T-symbols) there is a quantifier-free formula I with:

- $A \models_T I$
- $B \models_T \neg I$
- every variable in I occurs in both A and B.

General quantifier-free theory interpolation

Definition (Bruttomesso, Ghilardi, Ranise, 2014)

A theory T admits *general quantifier-free interpolation* if for every closed quantifier-free T-unsatisfiable conjunction $A \wedge B$ (with symbols from T, but also including other functions or predicates) there is a quantifier-free (reverse) interpolant I.

- Plain and general QFI can be characterized in terms of (sub-)amalgamation.
- The second property is equivalent to the notion of *equality interpolation*, and important for theory combination.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI										
gen. QFI										

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓		✓							
gen. QFI	✓		✓							

- Kenneth L. McMillan: An interpolating theorem prover. Theor. Comput. Sci. 345(1): 101-121 (2005)

Interpolating LRA

LRA proof rules

$$\frac{s \geq 0 \qquad t \geq 0}{\alpha s + \beta t \geq 0} \quad (\text{for } \alpha, \beta \geq 0) \qquad\qquad \frac{\alpha \geq 0}{\square} \quad (\text{for } \alpha < 0)$$

Interpolating LRA (2)

Interpolating LRA proof rules

- Annotate every inequality with a *partial interpolant:*

$$\frac{s \geq 0 \text{ is a formula from } A}{s \geq 0 \; [s]} \qquad \frac{s \geq 0 \text{ is a formula from } B}{s \geq 0 \; [0]}$$

- Propagate those partial interpolants:

$$\frac{s \geq 0 \; [s'] \qquad t \geq 0 \; [t']}{\alpha s + \beta t \geq 0 \; [\alpha s' + \beta t']} \quad (\text{for } \alpha, \beta \geq 0) \qquad \frac{\alpha \geq 0 \; [s']}{\Box \; [s' \geq 0]} \quad (\text{for } \alpha < 0)$$

- The partial interpolant annotating $\Box$ is an interpolant for $A \wedge B$.

Interpolating LRA (2)

Interpolating LRA proof rules

- Annotate every inequality with a *partial interpolant:*

$$\frac{s \geq 0 \text{ is a formula from } A}{s \geq 0 \; [s]} \qquad \frac{s \geq 0 \text{ is a formula from } B}{s \geq 0 \; [0]}$$

- Propagate those partial interpolants:

$$\frac{s \geq 0 \; [s'] \qquad t \geq 0 \; [t']}{\alpha s + \beta t \geq 0 \; [\alpha s' + \beta t']} \quad (\text{for } \alpha, \beta \geq 0) \qquad \frac{\alpha \geq 0 \; [s']}{\Box \; [s' \geq 0]} \quad (\text{for } \alpha < 0)$$

- The partial interpolant annotating $\Box$ is an interpolant for $A \wedge B$.

- Similar rules can be defined for EUF.

Interpolation paradigms

1. Proof-based
    1.1 Bottom-up: propagate partial interpolants ("resolution-style")

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
2. Graph-based: summarize edges in an e-graph

- Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstic, Cesare Tinelli: Ground interpolation for the theory of equality. Log. Methods Comput. Sci. 8(1) (2012)

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
2. Graph-based: summarize edges in an e-graph
3. Quantifier elimination

- Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstic, Cesare Tinelli: Ground interpolation for the theory of equality. Log. Methods Comput. Sci. 8(1) (2012)

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓		✓							
gen. QFI	✓		✓							

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓		✓	✓[1]	✓		✓	✓	✓	
gen. QFI	✓		✓							

- Every theory that admits quantifier elimination also has plain quantifier-free interpolation.

[1] Needs a divisibility operator $|$.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓		✓	✓[1]	✓		✓	✓	✓	
gen. QFI	✓		✓							

- Every theory that admits quantifier elimination also has plain quantifier-free interpolation.
- Interpolants computed using quantifier elimination tend to be less useful in verification: no “abstraction from unnecessary details”

[1] Needs a divisibility operator $|$.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓		✓	✓	✓	
gen. QFI	✓	✓[2]	✓							

- Every theory that admits quantifier elimination also has plain quantifier-free interpolation.
- Interpolants computed using quantifier elimination tend to be less useful in verification: no “abstraction from unnecessary details”

[1] Needs a divisibility operator $|$.
[2] Needs a *diff* function, see Silvio’s talk.

Proof-based LIA Interpolation

LIA proof rules

LRA proof rules + some combination of:

- Branch & bound:

$$\frac{x = \alpha}{x \leq \lfloor \alpha \rfloor \quad x \geq \lceil \alpha \rceil}$$

- Cuts:

$$\frac{\sum_i \alpha_i x_i + \beta \geq 0}{\sum_i \frac{\alpha_i}{\gamma} x_i + \lfloor \frac{\beta}{\gamma} \rfloor \geq 0} \quad (\gamma > 0 \text{ divides all } \alpha_i)$$

- Strengthening (e.g., Omega test):

$$\frac{t \geq n}{t = n \quad t \geq n+1}$$

Proof-based LIA Interpolation

LIA proof rules

LRA proof rules + some combination of:

- Branch & bound:

$$\frac{x = \alpha}{x \leq \lfloor \alpha \rfloor \quad x \geq \lceil \alpha \rceil}$$

- Cuts:

$$\frac{\sum_i \alpha_i x_i + \beta \geq 0}{\sum_i \frac{\alpha_i}{\gamma} x_i + \lfloor \frac{\beta}{\gamma} \rfloor \geq 0} \quad (\gamma > 0 \text{ divides all } \alpha_i)$$

- Strengthening (e.g., Omega test):

$$\frac{t \geq n}{t = n \quad t \geq n + 1}$$

- Splitting requires a further paradigm in interpolation . . .

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
   1.2 **Top-down: recursive computation of interpolants**
2. Graph-based: summarize edges in an e-graph
3. Quantifier elimination

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
   1.2 **Top-down: recursive computation of interpolants**
2. Graph-based: summarize edges in an e-graph
3. Quantifier elimination

Computation of interpolants with splitting

$$\frac{A_1 \vee A_2, B \;\triangleright\; I_1 \vee I_2}{A_1, B \;\triangleright\; I_1 \qquad A_2, B \;\triangleright\; I_2} \qquad\qquad \frac{A, B_1 \vee B_2 \;\triangleright\; I_1 \wedge I_2}{A, B_1 \;\triangleright\; I_1 \qquad A, B_2 \;\triangleright\; I_2}$$

Proof-based LIA Interpolation

LIA proof rules

- Branch & bound:

$$\frac{x = \alpha}{x \leq \lfloor \alpha \rfloor \quad x \geq \lceil \alpha \rceil}$$

- Cuts:

$$\frac{\sum_i \alpha_i x_i + \beta \geq 0}{\sum_i \frac{\alpha_i}{\gamma} x_i + \lfloor \frac{\beta}{\gamma} \rfloor \geq 0} \quad (\gamma > 0 \text{ divides all } \alpha_i)$$

- Strengthening (e.g., Omega test):

$$\frac{t \geq n}{t = n \quad t \geq n + 1}$$

Proof-based LIA Interpolation

LIA proof rules

- Branch & bound: ✓

$$\frac{x = \alpha}{x \leq \lfloor \alpha \rfloor \quad x \geq \lceil \alpha \rceil}$$

- Cuts:

$$\frac{\sum_i \alpha_i x_i + \beta \geq 0}{\sum_i \frac{\alpha_i}{\gamma} x_i + \lfloor \frac{\beta}{\gamma} \rfloor \geq 0} \quad (\gamma > 0 \text{ divides all } \alpha_i)$$

- Strengthening (e.g., Omega test):

$$\frac{t \geq n}{t = n \quad t \geq n + 1}$$

Proof-based LIA Interpolation

LIA proof rules

- Branch & bound: ✓

$$\frac{x = \alpha}{x \leq \lfloor \alpha \rfloor \quad x \geq \lceil \alpha \rceil}$$

- Cuts: ✓

$$\frac{\sum_i \alpha_i x_i + \beta \geq 0}{\sum_i \frac{\alpha_i}{\gamma} x_i + \lfloor \frac{\beta}{\gamma} \rfloor \geq 0} \quad (\gamma > 0 \text{ divides all } \alpha_i)$$

- Strengthening (e.g., Omega test): ✓

$$\frac{t \geq n}{t = n \quad t \geq n + 1}$$

- For poly-size interpolants: either integer division $\div$ or bounded quantifiers needed.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓		✓	✓	✓	
gen. QFI	✓	✓[2]	✓							

- Alberto Griggio, Thi Thieu Hoa Le, Roberto Sebastiani: Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic. TACAS 2011: 143-157
- Angelo Brillout, Daniel Kroening, PR, Thomas Wahl: An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. J. Autom. Reason. 47(4): 341-367 (2011)

[1] Needs ~~a divisibility operator $\mid$~~ integer division $\div$ or bounded quantifiers.
[2] Needs a *diff* function.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓		✓	✓	✓	
gen. QFI	✓	✓[2]	✓	✓[1]						

- Alberto Griggio, Thi Thieu Hoa Le, Roberto Sebastiani: Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic. TACAS 2011: 143-157
- Angelo Brillout, Daniel Kroening, PR, Thomas Wahl: An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. J. Autom. Reason. 47(4): 341-367 (2011)
- Angelo Brillout, Daniel Kroening, PR, Thomas Wahl: Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic. VMCAI 2011: 88-102

[1] Needs ~~a divisibility operator $|$~~ integer division $\div$ or bounded quantifiers.
[2] Needs a *diff* function.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓	✗[3]	✓	✓	✓	
gen. QFI	✓	✓[2]	✓	✓[1]						

- Alberto Griggio, Thi Thieu Hoa Le, Roberto Sebastiani: Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic. TACAS 2011: 143-157
- Angelo Brillout, Daniel Kroening, PR, Thomas Wahl: An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. J. Autom. Reason. 47(4): 341-367 (2011)
- Angelo Brillout, Daniel Kroening, PR, Thomas Wahl: Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic. VMCAI 2011: 88-102
- Peter Backeman, PR, Aleksandar Zeljic: Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. FMCAD 2018: 1-10

[1] Needs ~~a divisibility operator $\mid$~~ integer division $\div$ or bounded quantifiers.
[2] Needs a *diff* function.
[3] Integer polynomials.

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
   1.2 Top-down: recursive computation of interpolants
2. Graph-based: summarize edges in an e-graph
3. Quantifier elimination
4. **Reduction-based: by mapping interpolation problem to another theory**

- Deepak Kapur, Rupak Majumdar, Calogero G. Zarba: Interpolation for data structures. SIGSOFT FSE 2006: 105-116

Fixed-length bit-vectors

- Formalization of machine arithmetic, very widely used in verification
- Domains $x \in \mathbb{B}^n$, often for $n = 32$ or $n = 64$
- Different classes of operations:
  - Arithmetic: $\text{bvadd}, \text{bvmul}, \dots$
  - Sequence: $\text{concat}, \text{extract}, \text{shift}, \dots$
  - Bit-wise: $\text{bvand}, \text{bvor}, \dots$
- Though finite, often resulting in very hard constraints

Bit-vector interpolation by reduction

Approaches

- Approach 1: reduction to propositional logic $\rightarrow$ “bit-blasting”
- Approach 2: reduction to LIA/NIA
- Approach 3: lazy reduction to LIA/NIA

Bit-vector interpolation by reduction

Approaches

- Approach 1: reduction to propositional logic $\rightarrow$ "bit-blasting"
  - Good for formulas with many bit-wise operations
  - Low-level propositional interpolants, less useful for software verification
- Approach 2: reduction to LIA/NIA
- Approach 3: lazy reduction to LIA/NIA

Bit-vector interpolation by reduction

Approaches

- Approach 1: reduction to propositional logic $\rightarrow$ "bit-blasting"
  - Good for formulas with many bit-wise operations
  - Low-level propositional interpolants, less useful for software verification
- Approach 2: reduction to LIA/NIA
  - Good for formulas with mostly linear, arithmetic operations
  - Due to overflows, often leads to hard LIA formulas and convoluted interpolants
- Approach 3: lazy reduction to LIA/NIA

- A. Griggio, "Effective word-level interpolation for software verification," FMCAD 2011

Bit-vector interpolation by reduction

Approaches

- Approach 1: reduction to propositional logic $\rightarrow$ "bit-blasting"
  - Good for formulas with many bit-wise operations
  - Low-level propositional interpolants, less useful for software verification
- Approach 2: reduction to LIA/NIA
  - Good for formulas with mostly linear, arithmetic operations
  - Due to overflows, often leads to hard LIA formulas and convoluted interpolants
- Approach 3: lazy reduction to LIA/NIA
  - Good for formulas with mostly arithmetic operations; much "nicer" interpolants
  - Still difficult to support bit-wise operations efficiently

- A. Griggio, "Effective word-level interpolation for software verification," FMCAD 2011
- Peter Backeman, PR, Aleksandar Zeljic: Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. FMCAD 2018: 1-10

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓	✗[3]	✓	✓	✓	
gen. QFI	✓	✓[2]	✓	✓[1]						

[1] Needs integer division $\div$ or bounded quantifiers.
[2] Needs a *diff* function.
[3] Integer polynomials.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓	✗[3]	✓	✓	✓	
gen. QFI	✓	✓[2]	✓	✓[1]					✓	

- Hossein Hojjat, PR: Deciding and Interpolating Algebraic Data Types by Reduction. SYNASC 2017: 145-152

[1]Needs integer division $\div$ or bounded quantifiers.
[2]Needs a *diff* function.
[3]Integer polynomials.

The Big Picture

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT	Strings
plain QFI	✓	✓[2]	✓	✓[1]	✓	✗[3]	✓	✓	✓	
gen. QFI	✓	✓[2]	✓	✓[1]			✓	✓	✓	

- Hossein Hojjat, PR: Deciding and Interpolating Algebraic Data Types by Reduction. SYNASC 2017: 145-152

[1]Needs integer division $\div$ or bounded quantifiers.
[2]Needs a *diff* function.
[3]Integer polynomials.

Interpolation paradigms

1. Proof-based
   1.1 Bottom-up: propagate partial interpolants ("resolution-style")
   1.2 Top-down: recursive computation of interpolants
2. Graph-based: summarize edges in an e-graph
3. Quantifier elimination
4. Reduction-based: by mapping interpolation problem to another theory
5. **Constraint-based: systematic search for interpolants in some language**
   - Syntax-guided synthesis
   - Linear arithmetic constraint solving

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	**Arrays**	**LRA**	**LIA**	**NRA**	**NIA**	**BV**	**Floats**	**ADT**
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5									
OpenSMT2									
Princess									
SMTInterpol									
cvc5									
Vampire									
Z3									

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	**Arrays**	**LRA**	**LIA**	**NRA**	**NIA**	**BV**	**Floats**	**ADT**
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2									
Princess									
SMTInterpol									
cvc5									
Vampire									
Z3									

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess									
SMTInterpol									
cvc5									
Vampire									
Z3									

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess	✓	✓		✓		✓	✓		✓
SMTInterpol									
cvc5									
Vampire									
Z3									

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess	✓	✓		✓		✓	✓		✓
SMTInterpol	✓	✓	✓	✓					
cvc5									
Vampire									
Z3									

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess	✓	✓		✓		✓	✓		✓
SMTInterpol	✓	✓	✓	✓					
cvc5	✓[1]	✓	✓	✓	✓	✓	✓	✓	✓
Vampire									
Z3									

[1]Via syntax-guided synthesis.

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess	✓	✓		✓		✓	✓		✓
SMTInterpol	✓	✓	✓	✓					
cvc5	✓[1]	✓	✓	✓	✓	✓	✓	✓	✓
Vampire	✓	✓[2]	✓	✓					
Z3									

[1] Via syntax-guided synthesis.

[2] Focussing on first-order interpolants.

Interpolation support in SMT solvers *(apologies for errors!)*

	EUF	Arrays	LRA	LIA	NRA	NIA	BV	Floats	ADT
plain QFI	✓	✓	✓	✓	✓	✗	✓	✓	✓
gen. QFI	✓	✓	✓	✓			✓	✓	✓
MathSAT5	✓	?	✓	✓			✓	?	
OpenSMT2	✓		✓	✓					
Princess	✓	✓		✓		✓	✓		✓
SMTInterpol	✓	✓	✓	✓					
cvc5	✓[1]	✓	✓	✓	✓	✓	✓	✓	✓
Vampire	✓	✓[2]	✓	✓					
Z3		✓[3]	✓	✓	✓	✓	✓		✓

[1] Via syntax-guided synthesis.
[2] Focussing on first-order interpolants.
[3] Via its constrained Horn clause engine.

Beyond binary interpolation

Extended versions of interpolation

- Sequence interpolants
- Tree interpolants
- Disjunctive interpolants
- DAG interpolants

- All those notions can be reduced to binary/standard interpolation.
- But they are quite widely used: most solvers support sequence and/or tree interpolants.

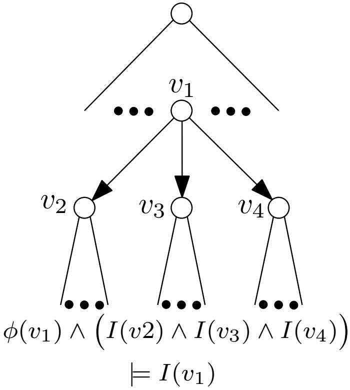
Example: tree interpolation

Tree interpolant

Suppose $T = (V, E)$ is a finite directed tree, and $\phi : V \to For$ a labeling function such that $\bigwedge_{v \in V} \phi(v)$ is unsatisfiable.

$I : V \to For$ is a *tree interpolant* if

- $I(root) = false$
- For all $v \in V$:
  $\phi(v) \wedge \bigwedge_{(v,w) \in E} I(w) \models I(v)$
- Non-logical symbols in $I(v)$ occur both in the sub-tree underneath v and in the rest of the tree.



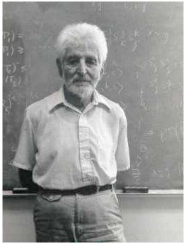
Craig interpolation as recursion-free Horn solving

Observation

- Let A, B be formulas with common variables $\bar{x}$.
- Then:

$$I(\bar{x}) \text{ is a reverse interpolant of } A \wedge B$$
$$\Leftrightarrow$$
$$\text{Formulas } A \rightarrow I(\bar{x}) \text{ and } B \wedge I(\bar{x}) \rightarrow \mathit{false} \text{ are valid}$$

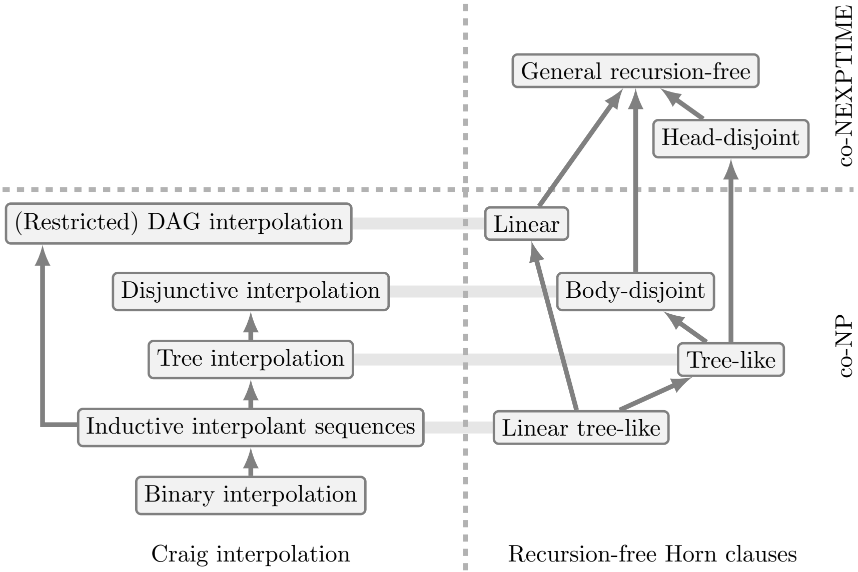
- $A \rightarrow I(\bar{x})$, $B \wedge I(\bar{x}) \rightarrow \mathit{false}$ can be seen as *constrained Horn clauses* over a relation symbol I.
- Correspondence between (extended) Craig interpolants and solution sets of recursion-free Horn clauses



William Craig

Alfred Horn

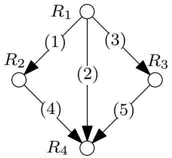
Taxonomy of Recursion-free Horn Clauses & Interpolation



Recursion-Free Horn Clause Fragments

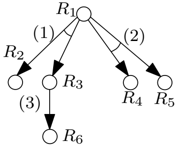
Linear: the body of each clause contains at most one relation symbol.

$$
\begin{aligned}
&1)\, C_1 \wedge R_2(\bar{x}) \rightarrow R_1(\bar{x})\\
&2)\, C_2 \wedge R_4(\bar{x}) \rightarrow R_1(\bar{x})\\
&3)\, C_3 \wedge R_3(\bar{x}) \rightarrow R_1(\bar{x})\\
&4)\, C_4 \wedge R_4(\bar{x}) \rightarrow R_2(\bar{x})\\
&5)\, C_5 \wedge R_4(\bar{x}) \rightarrow R_3(\bar{x})
\end{aligned}
$$



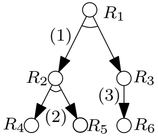
Body-disjoint: each relation symbol occurs at most once in body of a clause.

$$
\begin{aligned}
&1)\, C_1 \wedge R_2(\bar{x}) \wedge R_3(\bar{x}) \rightarrow R_1(\bar{x})\\
&2)\, C_2 \wedge R_4(\bar{x}) \wedge R_5(\bar{x}) \rightarrow R_1(\bar{x})\\
&3)\, C_3 \wedge R_6(\bar{x}) \rightarrow R_3(\bar{x})
\end{aligned}
$$



Tree-like: body-disjoint & head-disjoint: each relation symbol occurs at most once in head of a clause.

$$
\begin{aligned}
&1)\, C_1 \wedge R_2(\bar{x}) \wedge R_3(\bar{x}) \rightarrow R_1(\bar{x})\\
&2)\, C_2 \wedge R_4(\bar{x}) \wedge R_5(\bar{x}) \rightarrow R_2(\bar{x})\\
&3)\, C_3 \wedge R_6(\bar{x}) \rightarrow R_3(\bar{x})
\end{aligned}
$$



Horn solving in verification

- Constrained Horn clauses are considered a "unifying framework" in software model checking
- Horn solvers often internally use Craig interpolation
- Vice versa, Horn solvers are able to compute Craig interpolants

- PR, Hossein Hojjat, Viktor Kuncak: The Relationship between Craig Interpolation and Recursion-Free Horn Clauses. CoRR abs/1302.4187 (2013)

Conclusions

- Consider the talk as the starting point of a systematic survey
- Several dimensions remain to be explored:
  - Support for theory combination
  - Interpolation vs. uniform interpolation
  - Support for quantifiers
  - Complexity
- Comments, questions?

Challenges

- Interpolation for some of the theories:
  - Bit-vectors
  - Floating-point numbers
  - Strings, sequences

- What is a good interpolant? How to search for interpolants?