Spatial Interpolants

Aws Albargouthi Josh Berdine Byron Cook Zachary Kincaid

August 2014
Dagstuhl
Seminar 14351: Decision Procedures and Abstract Interpretation

Problem

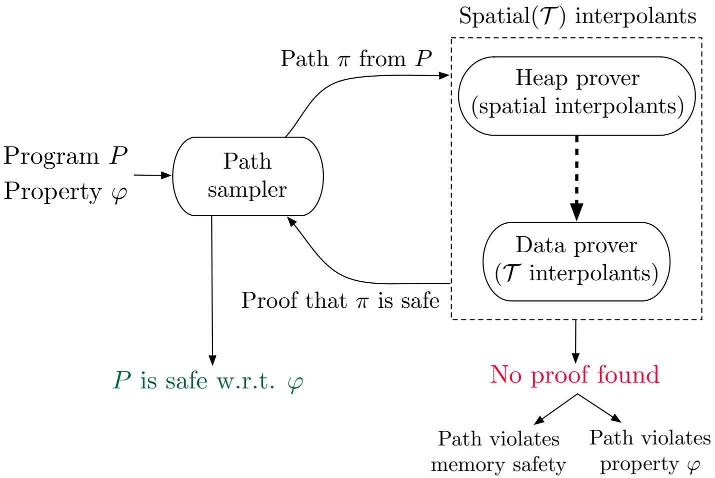
Combined heap and data reasoning for automatic verification

Examples:

- scalar constraints on heap-resident data
- traversing linked structures by size
- storing array indices in linked data-structures
- manual reference counting

```
1: int i = nondet();
   node* x = null;
2: while (i != 0)
     node* tmp = malloc(node);
     tmp->N = x;
     tmp->D = i;
     x = tmp;
     i--;
3: while (x != null)
4:   assert(x->D >= 0);
     x = x->N;
```

SplInter from 10,000 feet



- No heap: specializes to Impact (McMillan's *lazy abstraction with interpolants*)
- No data: specializes to new path-based separation logic analysis

Motivation for Path Sampling

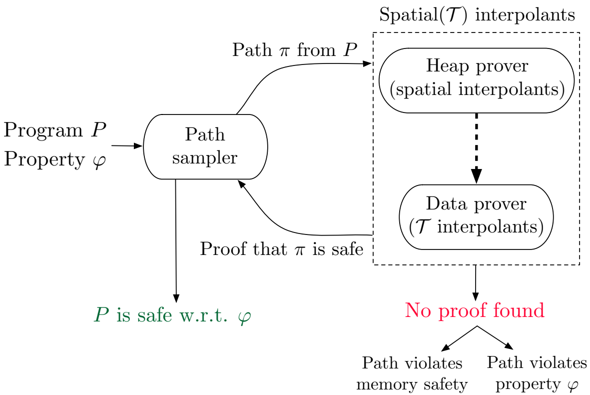
Path sampling enables

- *Path-based refinement*
  - *progress* guarantee by tightly correlating program exploration with refinement
  - *precision* guarantee by avoiding lossy join and widening operations
  - produces *counter-examples* for violated properties
  - no false alarms (diverges instead, as usual)
- *Property-direction*
  - don't try to compute strongest invariant possible
  - compute one *just strong enough* to prove property holds
  - key enabler for scalable precise reasoning in "rich" program logics

Main impediment

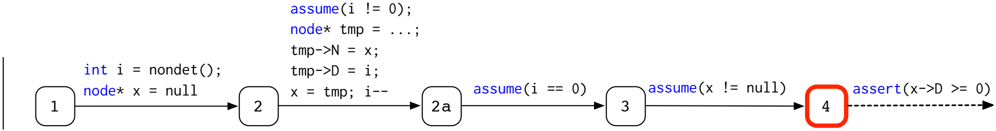
- *(infinitely-) many* paths may be analyzed before finding proof

Path Sampling



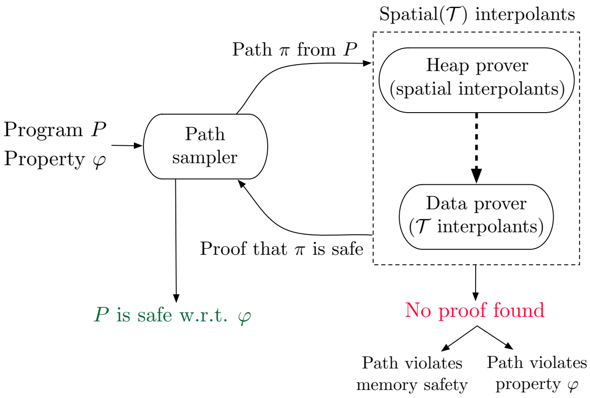
- Follows IMPACT
- Optimizations exist, *but basically*:
  - Maintain set of paths and their proofs
  - At each step, choose an arbitrary path
    - finite path through control-flow graph
    - from program entry to an assertion
    - not already proved

Path Sampling: Example




```
1: int i = nondet();
   node* x = null;
2: while (i != 0)
     node* tmp = malloc(node);
     tmp->N = x;
     tmp->D = i;
     x = tmp;
     i--;
3: while (x != null)
4:   assert(x->D >= 0);
     x = x->N;
```

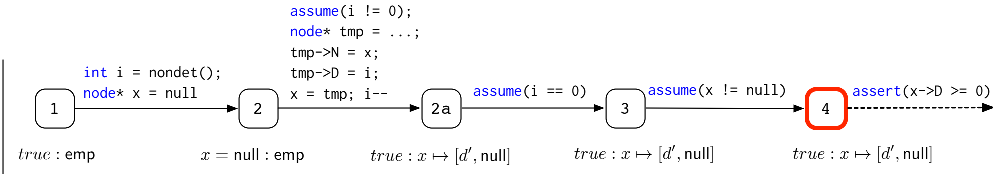
Spatial Interpolation



- Construct Hoare-style memory safety proof for path
- Call annotations *spatial path interpolants*
  - logical strength *between* strongest postconditions and weakest preconditions
  - *do not* impose other conditions of Craig interpolants
- Two-phase computation
  1. symbolically execute path *forward* to compute strongest data-free postconditions
  2. relax proof via *backward* under-approximation of weakest preconditions
     - heuristic
     - guided by strongest postconditions along path

Strongest Postconditions: Example

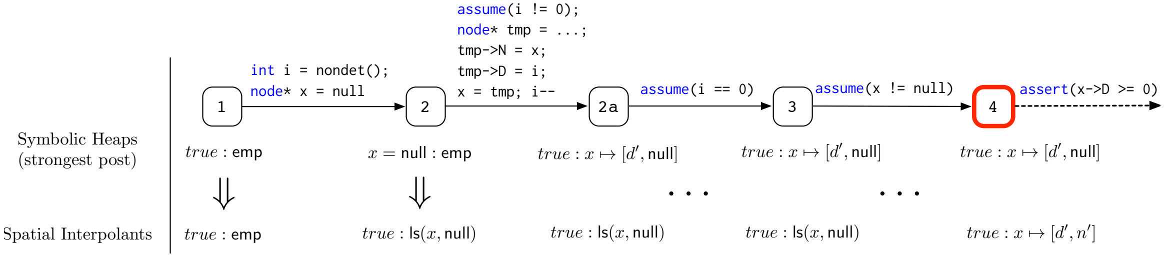
Symbolic Heaps (strongest post)



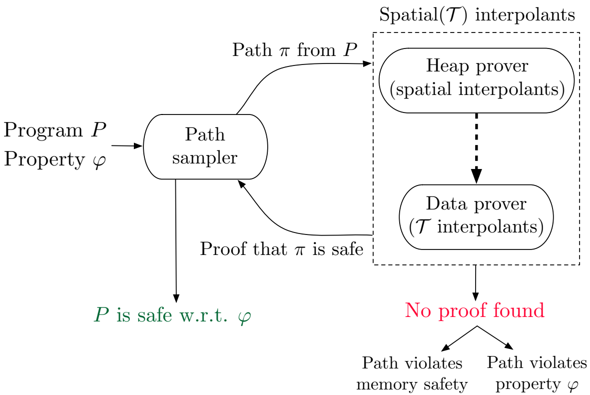
$$\mathsf{exec}(\texttt{x->N}_i \coloneqq \texttt{E},\ (\exists X\,.\,\Pi : \Sigma * z \mapsto [\vec{d}, \vec{n}])) \ = \ (\exists X\,.\,\Pi : \Sigma * x \mapsto [\vec{d}, \vec{n}[E/n_i]])$$

$$\text{where } i \leqslant |\vec{n}| \text{ and } \Pi : \Sigma * z \mapsto [\vec{d}, \vec{n}] \vdash x = z$$

Spatial Interpolation: Example

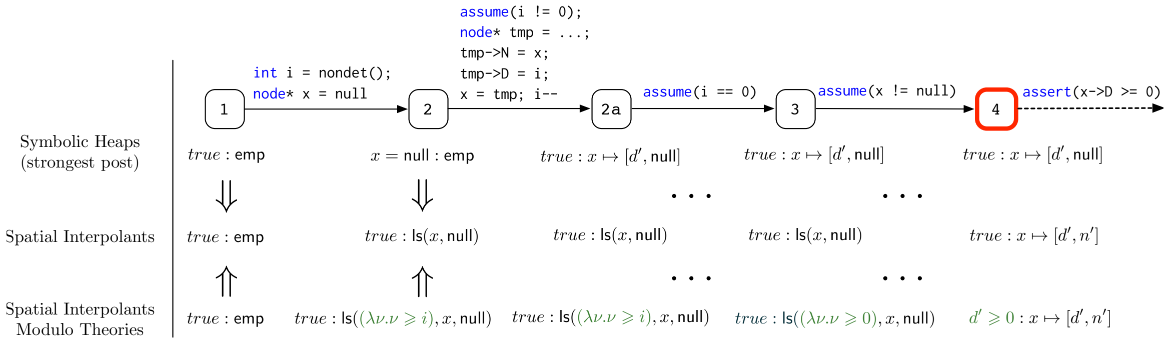


Spatial Interpolation Modulo Theories



- Strengthen memory safety proof of path
  - add data constraints
  - prove path satisfies safety property
- Generate system of Horn clause constraints
  - encode data manipulation along path, and its memory safety proof
  - solve using existing techniques
  - solution determines *refinement* (strengthening) of memory safety proof

Spatial Interpolation Modulo Theories: Example



Spatial Interpolants

- Bounded from *below* by strongest memory safety proof
- Bounded from *above* (implicitly) by weakest memory safety proof
- Without *upper* bound
  - Interpolant/invariant computable using forward transformer and widening
  - Risks widening too aggressively
    - so analyses widen conservatively at the price of computing unnecessarily strong proofs
  - Upper bound captures information needed to prove future execution
- Without *lower* bound
  - Interpolant/invariant computable using backward transformer (and lower widening)
  - Backward transformers in shape analysis explode
    - due to issues such as not knowing the aliasing relationship in the pre-state
  - Lower bound captures such information, containing the explosion
- Price of both bounds is operating over *full paths* from entry to error
- Heuristics for weakening at each point along the path have information about
  - *one* execution's *past and future* when analyzing full paths
  - *many past* executions in a forwards iterative analysis via join or widening

Bounded Abduction

Definition (Bounded abduction)

A solution to the *bounded abduction problem* $L \vdash (\exists X.\, M * [\,]) \vdash R$

is a formula A such that $L \models (\exists X.\, M * A) \models R$.

Compared to bi-abduction

- *Bounded abduction* solution: 1 formula constrained from above and below
- *Bi-abduction* solution: 2 formulas, one constrained from above and one from below
- Bounded abduction: fixed lower and upper bounds give considerable guidance to solvers
- Bi-abduction: bounds are part of the solution

Solving Bounded Abduction

$$L \vdash (\exists X.\, M * [\,]) \vdash R$$

Sound but incomplete algorithm

1. Find a *coloring* of L
   - each heaplet in L is either red or blue
   - red heaplets satisfy M, blue heaplets are left over
   - computed by recursion on proof of $L \vdash (\exists X.\, M * \mathsf{true})$
2. Find a *colored strengthening* $\Pi : [M']^{\mathrm{r}} * [A]^{\mathrm{b}}$ of R
   - entails R
   - is colored such that
     - red heaplets correspond to red heaplets of L
     - blue heaplets correspond to blue heaplets of L
   - computed by recursion on proof of $L \vdash R$ using coloring of L
3. Check $\Pi' : M * A \models R$, where Π' is the strongest pure formula implied by L
   - necessary because M may be weaker than M'
   - if entailment check fails, then algorithm fails
   - if entailment check succeeds, then $\Pi'' : A$ is a solution
     - Π'' is all equalities and disequalities used in proof of $\Pi' : M * A \models R$

Bounded Abduction: Example

Example

$$\underbrace{x \mapsto [a, y] * y \mapsto [b, \mathsf{null}]}_{L} \vdash \mathsf{ls}(x, y) * [\,] \vdash \underbrace{(\exists z.\ x \mapsto [a, z] * \mathsf{ls}(y, \mathsf{null}))}_{R}$$

1. Color L: $[x \mapsto [a, y]]^{\mathsf{r}} * [y \mapsto [b, \mathsf{null}]]^{\mathsf{b}}$ using proof of $L \vdash \mathsf{ls}(x, y) * \mathsf{true}$
2. Color R: $(\exists z.\ [x \mapsto [a, z]]^{\mathsf{r}} * [\mathsf{ls}(y, \mathsf{null})]^{\mathsf{b}})$ using proof of $L \vdash R$
3. Prove

   $$\underbrace{x \neq \mathsf{null} \land y \neq \mathsf{null} \land x \neq y}_{\text{strongest pure consequence of } L} : \mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{null}) \models R$$

   This proof succeeds, and uses pure assertion $x \neq y$.
4. Return solution $x \neq y : \mathsf{ls}(y, \mathsf{null})$

Computing Spatial Interpolants

Given command c and Sep formulas S and I' such that $\mathsf{exec}(\mathsf{c}, S) \vdash I'$
Compute a Sep formula $\mathsf{itp}(S, \mathsf{c}, I')$ such that $S \models I$ and $\{I\}\ \mathsf{c}\ \{I'\}$ is valid

$$\mathsf{itp}(S, \texttt{x->N}_i \coloneqq \texttt{E}, I') = (\exists \vec{a}, \vec{z}.\ A * x \mapsto [\vec{a}, \vec{z}])$$

where A satisfies

$$\mathsf{exec}(\mathsf{c}, S) \vdash (\exists \vec{a}, \vec{z}.\ x \mapsto [\vec{a}, \vec{z}[E/z_i]] * [A]) \vdash I'$$

Example

Suppose $S = t \mapsto [4, y, \mathsf{null}] * x \mapsto [2, \mathsf{null}, \mathsf{null}]$ $\quad \mathsf{c} = \texttt{t->N}_0 \coloneqq \texttt{x}$ $\quad I' = \mathsf{bt}(t)$

Compute $$\mathsf{exec}(\mathsf{c}, S) = t \mapsto [4, x, \mathsf{null}] * x \mapsto [2, \mathsf{null}, \mathsf{null}]$$

Solve $$\mathsf{exec}(\mathsf{c}, S) \vdash (\exists a, z_1.\ t \mapsto [a, x, z_1] * [\ ]) \vdash I'$$

One solution is $\mathsf{bt}(x) * \mathsf{bt}(z_1)$, yielding

$$\mathsf{itp}(S, \mathsf{c}, I') = (\exists a, z_0, z_1.\ t \mapsto [a, z_0, z_1] * \mathsf{bt}(z_1) * \mathsf{bt}(x))$$

Spatial Interpolation Modulo Theories

Given proof ζ of $\{\mathit{true} : \mathsf{emp}\}\ \pi\ \{\mathit{true} : \mathsf{true}\}$, and a postcondition ϕ
Transform ζ into proof of $\{\mathit{true} : \mathsf{emp}\}\ \pi\ \{\phi : \mathsf{true}\}$

1. Traverse ζ and build
   - *refined* proof ζ' where refinements may contain 2nd-order variables
   - constraint system $\mathcal{C}$ which encodes logical dependencies between 2nd-order variables
2. Solve $\mathcal{C}$
   - for an assignment of data formulas to 2nd-order variables that satisfies all constraints
3. If successful, instantiate 2nd-order variables in ζ'
   - yields valid proof of $\{\mathit{true} : \mathsf{emp}\}\ \pi\ \{\phi : \mathsf{true}\}$

Sound and Complete (per path, when heap-feasible)

Spatial Interpolation Modulo Theories: Example

Refined memory safety proof ζ'	Constraint system $\mathcal{C}$	Solution σ
$\{R_0(i) : \mathsf{true}\}$	$R_0(i') \leftarrow \mathit{true}$	$R_0(i) : \mathit{true}$
`i = nondet(); x = null`	$R_1(i') \leftarrow R_0(i)$	$R_1(i) : \mathit{true}$
$\{R_1(i) : \mathsf{ls}((\lambda a.R_{\mathsf{ls}1}(\nu, i)), x, \mathsf{null})\}$	$R_2(i') \leftarrow R_1(i) \wedge i \neq 0 \wedge i' = i + 1$	$R_2(i) : \mathit{true}$
`assume(i != 0); ...; i--;`	$R_3(i) \leftarrow R_2(i) \wedge i = 0$	$R_3(i) : \mathit{true}$
$\{R_2(i) : \mathsf{ls}((\lambda a.R_{\mathsf{ls}2}(\nu, i)), x, \mathsf{null})\}$	$R_4(i, d') \leftarrow R_3(i) \wedge R_{\mathsf{ls}3}(d', i)$	$R_4(i, d') : d' \geqslant 0$
`assume(i == 0)`	$R_{\mathsf{ls}2}(\nu, i') \leftarrow R_1(i) \wedge R_{\mathsf{ls}1}(\nu, i) \wedge i \neq 0 \wedge i' = i+1$	$R_{\mathsf{ls}1}(\nu, i) : \nu \geqslant i$
$\{R_3(i) : \mathsf{ls}((\lambda a.R_{\mathsf{ls}3}(\nu, i)), x, \mathsf{null})\}$	$R_{\mathsf{ls}2}(\nu, i') \leftarrow R_1(i) \wedge \nu = i \wedge i \neq 0 \wedge i' = i + 1$	$R_{\mathsf{ls}2}(\nu, i) : \nu \geqslant i$
`assume(x != null)`	$R_{\mathsf{ls}3}(\nu, i) \leftarrow R_2(i) \wedge R_{\mathsf{ls}2}(\nu, i) \wedge i = 0$	$R_{\mathsf{ls}3}(\nu, i) : \nu \geqslant 0$
$\{(\exists d', y.\ R_4(i, d') : x \mapsto [d', y])\}$	$d' \geqslant 0 \leftarrow R_4(i, d')$	

Symbolic Heaps (strongest post)	$\mathit{true} : \mathsf{emp}$	$x = \mathsf{null} : \mathsf{emp}$	$\mathit{true} : x \mapsto [d', \mathsf{null}]$	$\mathit{true} : x \mapsto [d', \mathsf{null}]$	$\mathit{true} : x \mapsto [d', \mathsf{null}]$
	$\Downarrow$	$\Downarrow$	$\cdots$	$\cdots$	
Spatial Interpolants	$\mathit{true} : \mathsf{emp}$	$\mathit{true} : \mathsf{ls}(x, \mathsf{null})$	$\mathit{true} : \mathsf{ls}(x, \mathsf{null})$	$\mathit{true} : \mathsf{ls}(x, \mathsf{null})$	$\mathit{true} : x \mapsto [d', n']$
	$\Uparrow$	$\Uparrow$	$\cdots$	$\cdots$	
Spatial Interpolants Modulo Theories	$\mathit{true} : \mathsf{emp}$	$\mathit{true} : \mathsf{ls}((\lambda\nu.\nu \geqslant i), x, \mathsf{null})$	$\mathit{true} : \mathsf{ls}((\lambda\nu.\nu \geqslant i), x, \mathsf{null})$	$\mathit{true} : \mathsf{ls}((\lambda\nu.\nu \geqslant 0), x, \mathsf{null})$	$d' \geqslant 0 : x \mapsto [d', n']$

Conclusions & Challenges

- SplInter is IMHO an important step in precise and generic automatic heap/data analyses
- Novel heap analysis, that specializes to a leading technique for numerical and control-sensitive property verification

- Not the last word on interface between spatial interpolation and bounded abduction
- Unclear if the spatial then data phasing can be relaxed
- Want better understanding of currently enumerative heuristic for spatial interpolation of assumptions
- Want better under-approximation of classical conjunction in separation logic
  - or generalize everything to handle it natively
- Want to revise "real" separation logic provers to generate data constraints