SLEdge: Bounded Model Checking in Separation Logic

Josh Berdine, Facebook

with contributions by
Timotej Kapus, Imperial College London
Benno Stein, University of Colorado Boulder
Scott Owens, University of Kent

Automated Deduction for Separation Logic (ADSL) 2020
New Orleans

Accurate symbolic execution for C++ via LLVM

- First focus: “memory safety” bugs
  - object lifetime / use-after-free, in various manifestations
- It is ok for a pointer to outlive the object to which it points,
- But it is not ok to dereference it
  - Must hold of all code, no matter what idioms it follows
- Goal is to refute safety properties
- Proving properties is currently a non-goal
  - general-purpose automatic heap abstractions are extremely hard
  - most practical value is in actionable bug reports
  - there are few bugs that cannot be found with a small number of objects

Accurate symbolic execution for C++ via LLVM

Why/how accurate?

- Important to have reasonable understanding of source code
- Need byte-sizes of memory accesses
- Need aliasing between different names of same field (the first member, inline structs, union members, etc.)
- Need relationship between bytewise (e.g. memcpy) and structured (e.g. field access) operations
- Aliasing at byte granularity

Accurate symbolic execution for C++ via *LLVM*

Why LLVM?

- C++: what does it even mean?
- LLVM: much smaller language
  - with semantics closer to level needed by analyzer
- Leave understanding the dark corners of C++ to clang
- Reuse bitcode already produced by standard builds
- Analyze (bit)code close to what runs in production
  - don't have to trust much of clang
  - not analyzing binary: still must trust some of llvm / linker / etc.

Accurate *symbolic* execution for C++ via LLVM

Why symbolic?

- Key for good coverage without good test suite / input generator
- Analysis "executes" all code branches
- Breadth-first exploration, with iterative deepening
- Locality-of-reference a la in hardware but in symbolic representation
- Join execution paths eagerly: DAGify the execution tree
- Enable later adaptation of Infers techniques for scalability
  - Summarization: intermediate results caching
  - Compositionality: "start anywhere" analysis

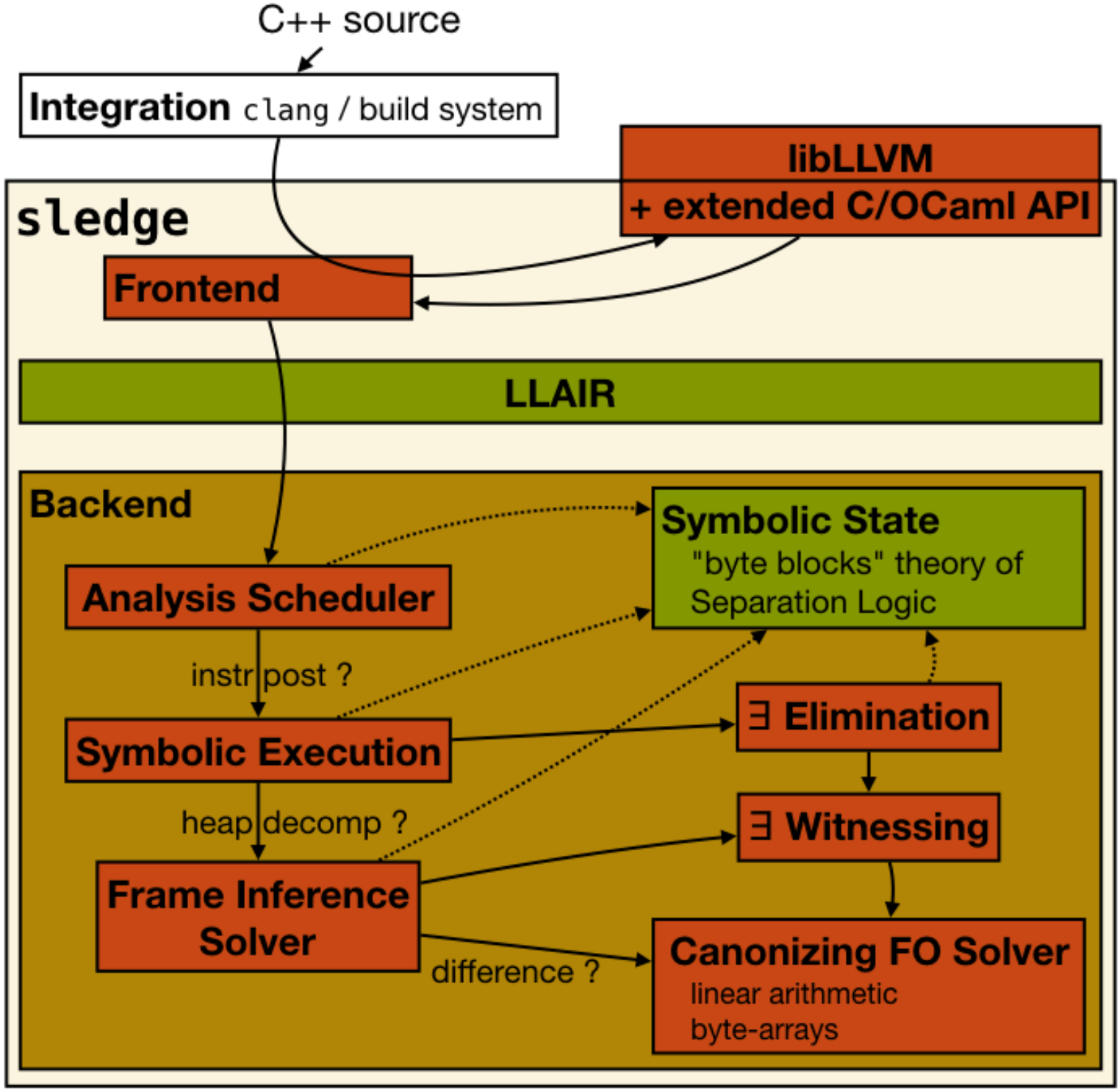
Accurate symbolic execution for *C++* via LLVM

Why C++?

- There is a *lot* of it
- There is a lot of it that matters
- Language is unsafe...
- But does have a justifiable reason to exist (other than legacy)
- Not so far from "generic unsafe code" via LLVM

Current limitations / sacrifices for initial iterations

- Summarization: currently intraprocedural, "as if inlined"
- Compositionality / (bi)abduction: no "start anywhere" capability, doesn't scale as $O(|\text{diff}|)$
- Dynamically-sized arrays imprecisely handled
- Over-approximate: maybe better to prove existence of bugs rather than fail to prove their absence
- C++ exceptions: frontend translation needs to be extended
- ...


C++ source
Integration clang / build system
libLLVM
+ extended C/OCaml API
sledge
Frontend
LLAIR
Backend
Symbolic State
"byte blocks" theory of
Separation Logic
Analysis Scheduler
instr post ?
Symbolic Execution
∃ Elimination
heap decomp ?
∃ Witnessing
Frame Inference
Solver
difference ?
Canonizing FO Solver
linear arithmetic
byte-arrays

LLAIR (Low-Level Analysis Internal Representation)

LLAIR vs. LLVM: different design constraints from different usage

- LLVM: compiler internal representation (IR) aimed at code transformation, generation, etc.
- LLAIR: IR tailored for static analysis using a low-level model of memory

Llair: SSA

Local variables

- LLVM: SSA
  - useful when performing code transformations, especially directed by static analysis results
- LLAIR: parallel-assignment (+ trampolines)
  - *static* single assignment does not buy much as there will be multiple *dynamic* assignments in any case, due to e.g. loops
  - traditional ϕ-nodes need some execution history to interpret: poor fit for analysis operating on representations of sets of states at program points
  - parallel-assignment hard for a compiler, easy for an analyzer
  - SSA essentially in continuation-passing style
  - Hoare logic (with triples) in direct style
  - mismatch: no SSA scope for variables in postconditions representing return values

LLAIR: ALU

Arithmetic and logic operations

- LLVM: linear 3-address code

```
%r1 = add %r2, %r3
%r5 = add %r1, %r4
%r6 = add %r5, %r7
```

- LLAIR: compound expressions over registers

```
%r6 = ((%r2 + %r3) + %r4) + %r7
```

Bounded vs mathematical integers

- LLVM: types have bitwidths, operations have signedness
- LLAIR: backend operates over mathematical integers, frontend adds conversions

LLAIR: Formalization

Formalization in HOL4 by Scott Owens

- semantics of LLVM
- semantics of LLAIR
- model of the translation implemented by the frontend
- correctness proofs
  - (full disclosure: mechanically verified with some unproven assumptions, some simplifications of LLVM e.g. pointers are just integers)
- aim: ensure that bogus/missed bugs are due to analysis trade-offs, not correctly analyzing incorrect code

Analysis Scheduler

- interprets control flow
  - lifts semantics of instructions to whole programs
  - handles scoping and parameter passing
- overall objective: find violations of safety properties using short executions before exploring longer executions
- which of the possible control-flow paths to explore next: iterative deepening breadth-first algorithm
- maintains a "depth map" associating retreating control-flow edges with the number of times they have been crossed
- schedules using a priority queue
  - order retreating edges by depth
  - otherwise use a topological order

Analysis Scheduler: Join

- joins different executions to the same control point as eagerly as possible
  - DAGify the tree of executions
- joining depth maps turns out to be tricky
  - pointwise max: unsound due to potentially missing a short counter-example execution
  - pointwise min: unfair due to potentially exploring long executions before a shorter counter-example execution
  - so need to be a bit less eager

Symbolic State

- Symbolic representation of sets of states
- Assertions of a "byte blocks" theory of separation logic

$$
\begin{array}{rcll}
\mathcal{X} \ni x & ::= & \mathtt{x} \mid \cdots & \text{variables} \\
\mathcal{N} \ni \eta & ::= & 0 \mid 1 \mid \cdots & \text{numerals} \\
\mathcal{C} \ni f & ::= & \eta \mid \cdots & \text{constants} \\
A, B, N, L, E & ::= & x \mid f(E^*) \mid \forall x{:}\tau.\ B \mid \exists x{:}\tau.\ B & \text{terms} \\
Q & ::= & B \mid Q \vee Q \mid \exists x{:}\tau.\ Q & \text{formulas} \\
& & \mid \mathsf{emp} \mid Q * Q \mid L \xmapsto{[L;N)} A &
\end{array}
$$

First-Order Base

- assume some sorts
  - bool: classical Booleans
  - siz: object sizes
  - loc: memory locations
  - byt: bytes
- assume function symbols for standard operations

  $\simeq$, true, $\neg$, $\wedge$, $\vee$, $<$, $+$, $-$, $\eta\times$, 0,...
- some sorts $\mathcal{W}$ are "scalars"
  - contains at least siz and loc
  - sizeof : $\mathcal{W} \to \mathcal{N}$ size of scalar sort's representation

Interpretation:

- $\mathbb{U}_\sigma$ subset of universe interpreting sort σ
- $[\![f : \left(\bigtimes_{i=1}^{n} \sigma_i\right) \to \sigma]\!] \in \left(\bigtimes_{i=1}^{n} \mathbb{U}_{\sigma_i}\right) \to \mathbb{U}_\sigma$
  interpretation of each constant $f : \left(\bigtimes_{i=1}^{n} \sigma_i\right) \to \sigma$

Byte-Arrays

Simplified peek ahead:

$$\{L \mapsto \_\} \qquad \{L \mapsto \alpha\}$$

$$\texttt{store}\eta\ L\ E \qquad x \coloneqq \texttt{load}\eta\ L$$

$$\{L \mapsto \texttt{cast}^{\text{typeof } E}_{\text{byt}[\eta]}\, E\} \qquad \{L \mapsto \alpha * (x \simeq \texttt{cast}^{\text{byt}[\eta]}_{\text{typeof } x}\, \alpha)\}$$

Known size: $\text{byt}[\eta]$ for $\eta \in \mathcal{N}$

- $\texttt{cast}^{\omega}_{\text{byt}[\eta]}\, E : \omega \to \text{byt}[\eta]$
  reinterpret scalar ω as aggregate $\text{byt}[\eta]$ for $\eta = \text{sizeof}\, \omega$
- $\texttt{cast}^{\text{byt}[\eta]}_{\omega}\, A : \text{byt}[\eta] \to \omega$
  reinterpret aggregate $\text{byt}[\eta]$ as scalar ω for $\eta = \text{sizeof}\, \omega$
- $\texttt{cast}^{\text{byt}[\eta]}_{\omega}(\texttt{cast}^{\omega}_{\text{byt}[\eta]}\, E) \simeq E$ and vice versa

Byte-Arrays

Simplified peek ahead:

$$\{L \mapsto \_\}\,\texttt{memset}\ L\ E\ N\,\{L \mapsto E^N\}$$

N.b.: size of access to memory not always known

Unknown size: byt[·]
Size-tagged: byt[]

- $\langle N, A \rangle$: siz × byt[·] → byt[] tag a byte-array with its size
- E^N : byt × siz → byt[] iterated byte concatenation
- $A^\frown A$: byt[] × byt[] → byt[] concatenation
- $A[N;N)$: byt[] × siz × siz → byt[] extraction

Allocations

Simplified peek ahead:

$$\{\mathsf{emp}\} \qquad\qquad \{L \simeq 0 \vee L \overset{[L;N)}{\longmapsto} \langle N, \alpha \rangle\}$$
$$x \coloneqq \mathtt{alloc}\ N \qquad\qquad \mathtt{free}\ L$$
$$\{x \simeq 0 \vee \exists \alpha.\, x \overset{[x;N)}{\longmapsto} \langle N, \alpha \rangle\} \qquad\qquad \{\mathsf{emp}\}$$

N.b.: need to free the whole allocation

$L \overset{[B;M)}{\longmapsto} A$ keeps track of the enclosing allocation $[B;M)$ of the claimed chunk of memory

Block $[b;m)$

- m contiguous locations starting from b
- is an element (b, m) of $\mathbb{U}_{\mathsf{loc}} \times \mathbb{U}_{\mathsf{siz}}$
- often punned with interval $\{\, l \mid b \leqslant l \wedge l < b + m \,\}$

Heaps

Heaps:

$$\mathbb{H} = \left\{ h = (h_\alpha, h_\mathsf{b}) \in \begin{array}{r} \bigcup_{L \subset_\mathsf{fin} \mathbb{U}_\mathsf{loc}} \\ (L \to \mathbb{U}_\mathsf{byt}) \\ \times\, (\mathbb{U}_\mathsf{loc} \times \mathbb{U}_\mathsf{siz}) \end{array} \;\middle|\; \begin{array}{l} h_\mathsf{b} \text{ is compatible} \\ h_\mathsf{b} \text{ is inhabited over } L \\ L \text{ is enclosed in } h_\mathsf{b} \end{array} \right\}$$

Heap combination:

$$\_ \uplus \_ \;:\; \mathbb{H} \times \mathbb{H} \to \mathbb{H}_\bot$$

$$h \uplus h' = \begin{cases} \bot & \text{if } \operatorname{dom} h_\alpha \cap \operatorname{dom} h'_\alpha \neq \emptyset \text{ or } h_\mathsf{b} \frown h'_\mathsf{b} \\ (h_\alpha \cup h'_\alpha, h_\mathsf{b} \cup h'_\mathsf{b}) & \text{otherwise} \end{cases}$$

Semantics:

$$s, h \vDash L \xmapsto{[B;M)} A \text{ iff let } l, b, m, \alpha = [\![L]\!]_\mathbb{E}\, s,\ [\![B]\!]_\mathbb{E}\, s,\ [\![M]\!]_\mathbb{E}\, s,\ [\![A]\!]_\mathbb{E}\, s$$
$$[l;|\alpha|) \subseteq [b;m) \text{ and } (|\alpha| \simeq 0 \text{ implies } m \simeq 0) \text{ and}$$
$$h = \left(\left[[l;|\alpha|) \to \alpha \right], \{\, [b;m) \,\} \right)$$

Example (non-)Entailments

$$l \overset{[b;m)}{\longmapsto} \langle n, \alpha_0 \rangle * l{+}n \overset{[b;m)}{\longmapsto} \langle o, \alpha_1 \rangle \quad \vDash \quad l \overset{[b;m)}{\longmapsto} \langle n, \alpha_0 \rangle^\frown \langle o, \alpha_1 \rangle$$

$$l \overset{[b;m)}{\longmapsto} \langle n, \alpha_0 \rangle^\frown \langle o, \alpha_1 \rangle * n > 0 \quad \vDash \quad l \overset{[b;m)}{\longmapsto} \langle n, \alpha_0 \rangle * l{+}n \overset{[b;m)}{\longmapsto} \langle o, \alpha_1 \rangle$$

$$l \overset{[b;m)}{\longmapsto} \langle 0, \alpha_0 \rangle^\frown \langle n, \alpha_1 \rangle \quad \nvDash \quad l \overset{[b;m)}{\longmapsto} \langle 0, \alpha_0 \rangle * l \overset{[b;m)}{\longmapsto} \langle n, \alpha_1 \rangle$$

$$l \overset{[l;0)}{\longmapsto} 0^0 * l \overset{[l;0)}{\longmapsto} 0^0 \quad \vDash \quad \mathsf{false}$$

$$l \overset{[l;0)}{\longmapsto} 0^0 \quad \nvDash \quad \mathsf{emp} \qquad\qquad \mathsf{emp} \quad \nvDash \quad l \overset{[l;0)}{\longmapsto} 0^0$$

$$l \overset{[b;m)}{\longmapsto} \langle 0, \alpha \rangle \quad \vDash \quad l \simeq b * m \simeq 0 * l \overset{[b;m)}{\longmapsto} \langle 0, \alpha \rangle$$

$$l \overset{[b;0)}{\longmapsto} \langle n, \alpha \rangle \quad \vDash \quad l \simeq b * n \simeq 0 * l \overset{[b;0)}{\longmapsto} \langle n, \alpha \rangle$$

Symbolic Execution

Symbolic execution (simple):

$$\frac{\vec{u},\vec{x}' \mid P' \vDash \exists\vec{g}.\, F * R \qquad \overline{\vec{v} \mid \{F\}\ I\ \{Q\}}}{\vec{u} \mid \{\exists\vec{x}.\, P\}\ I\ \{\exists\vec{x}',\vec{g}.\, Q * R\}} \quad \begin{array}{l} \_' \text{ freshens } \vec{x} \text{ wrt } \vec{u},\vec{v} \\ \vec{g} = \vec{v} \setminus \vec{u} \end{array}$$

Derivable using frame, consequence, substitution, quantification

Excision judgment $\forall\vec{u}.\, C \ast M \mathrel{\backslash\!\!-} \exists\vec{x}.\, S \ast R$

- *valid* iff $C * M \vDash \exists\vec{x}.\, C * S * R$ valid
- *strong* only if every model of R can be extended with a model of $C * S$ to obtain a model of $C * M$
- *query* $\forall\vec{u}.\, M \mathrel{\backslash\!\!-} \exists\vec{x}.\, S$ yields R s.t. $\forall\vec{u}.\, \mathrm{emp} \ast M \mathrel{\backslash\!\!-} \exists\vec{x}.\, S \ast R$

Solver for excision queries yields proof procedure for symbolic execution

Symbolic Execution with Modified Variables

Actual small axioms:

$$\theta.\{\mathsf{emp}\} \quad \vec{x} \coloneqq \vec{E} \quad \{\ast\, x_i \simeq E_i\theta\}$$

$$\theta.\{L \overset{[b;m)}{\longmapsto} \langle \eta, \alpha\rangle\} \quad x \coloneqq \mathtt{load}_\eta\ L \quad \{(L \overset{[b;m)}{\longmapsto} \langle \eta, \alpha\rangle)\theta * x \simeq (\mathtt{cast}^{\mathsf{byt}[\eta]}_{\mathsf{typeof}\ x}\ \alpha)\theta\}$$

θ records renaming of modified variables:

$$\theta.\{P\}\ I\ \{Q\} \quad \text{iff} \quad \left\{\left(\ast_{o \in \mathrm{dom}\,\theta}\ o \simeq o\theta\right) * P\right\}\ I\ \left\{Q\right\}$$

Symbolic execution:

$$\frac{\forall \vec{u}, \vec{x}'.\, \mathsf{emp} \mathrel{\not\ast} P' \mathrel{\not\vdash} \exists \vec{g}.\, F \mathrel{\not\ast} R \qquad \overline{\vec{v} \mid \theta.\{F\}\ I\ \{Q\}}}{\vec{u} \mid \{\exists \vec{x}.\, P\}\ I\ \{\exists \vec{x}', \vec{g}.\, Q * \exists \vec{m}.\, R\theta\}}$$

θ freshens $(\mathrm{mod}\, I \cap (\mathrm{fv}\, F \cup \mathrm{fv}\, I))$ wrt $\vec{u}, \mathrm{fv}\, F$ $\quad \vec{g} = \vec{v} \setminus \vec{u}$

$\_'$ freshens $\vec{x}$ wrt $\vec{u}, \vec{v}$ $\quad \vec{m} = \mathrm{mod}\, I \setminus \mathrm{dom}\,\theta$

Frame Inference Solver

Primitive rules:

$$\frac{\forall \vec{u}, \vec{x}.\ S \vdash \mathsf{emp} \qquad \llcorner \vdash \exists \vec{x}.\ S}{\forall \vec{u}.\ C \mathrel{\ast\!\!\!\vdash} M \mathrel{\not\vdash} \exists \vec{x}.\ S \mathrel{\not\ast\!\!\!\vdash} M * S} \qquad \frac{\llcorner \vdash \mathsf{false}}{\forall \vec{u}.\ C \mathrel{\ast\!\!\!\vdash} M \mathrel{\not\vdash} \exists \vec{x}.\ S \mathrel{\not\ast\!\!\!\vdash} \mathsf{false}}$$

$$\frac{\forall \vec{u}.\ C * H \mathrel{\ast\!\!\!\vdash} M \mathrel{\not\vdash} \exists \vec{x}.\ S \mathrel{\not\ast\!\!\!\vdash} R}{\forall \vec{u}.\ C \mathrel{\ast\!\!\!\vdash} H * M \mathrel{\not\vdash} \exists \vec{x}.\ H * S \mathrel{\not\ast\!\!\!\vdash} R}$$

Pure consequences, in rule with conclusion $\forall \vec{u}.\ C \mathrel{\ast\!\!\!\vdash} M \mathrel{\not\vdash} \exists \vec{x}.\ S \mathrel{\not\ast\!\!\!\vdash} M$:

- $\llcorner = \forall \vec{u}.\ \lceil C * M \rceil$
- $\llcorner\!\!\lrcorner = \forall \vec{u}.\ \lceil C * M \rceil \wedge \lceil S \rceil$

where first-order term $\lceil Q \rceil : \mathsf{bool}$:

- *over-approximates* Q: $Q \vDash \lceil Q \rceil$
- better *equisatisfiable*: $[\![\lceil Q \rceil]\!]_{\mathrm{Q}} = \emptyset$ iff $[\![Q]\!]_{\mathrm{Q}} = \emptyset$
- optimal *exact*: $[\![\lceil Q \rceil]\!]_{\mathrm{E}} \sim \{\, s \mid s, h \vDash Q \,\}$ for some bijection $\sim$

Frame Inference Solver: $\mapsto$

Omitted:

- treatment of disjunction
- algorithm to choose left and right subformulas to try to match

Focus:

- theory-specific reasoning rules

Simple case:

$$\frac{\begin{array}{cc} & \forall \vec{u}.\, C * k \xmapsto{[b;m)} \langle o, \alpha \rangle \mathrel{⫽} M \\ \sqsubset \vdash k \simeq l * o \simeq n & \models \exists \vec{x}.\, b \simeq b' * m \simeq m' * \alpha \simeq \alpha' * S \mathrel{⫽} R \end{array}}{\forall \vec{u}.\, C \mathrel{⫽} k \xmapsto{[b;m)} \langle o, \alpha \rangle * M \models \exists \vec{x}.\, l \xmapsto{[b';m')} \langle n, \alpha' \rangle * S \mathrel{⫽} R}$$

Involves only equality reasoning and "subformula shuffling"

Frame Inference Solver: $\mapsto$

$$\frac{\begin{array}{l}\lfloor\!\rfloor \vdash l < k * k{+}o \simeq l{+}n \\ \quad \forall \vec{u}.\, C * k \xmapsto{[b;m)} \langle o, \alpha\rangle \nVdash M \\ \quad \vdash \exists \vec{x}, \alpha_0'.\, \langle k{-}l, \alpha_0'\rangle^\frown \langle o, \alpha\rangle \simeq \langle n, \alpha'\rangle * \\ \qquad l \xmapsto{[b';m')} \langle k{-}l, \alpha_0'\rangle * b \simeq b' * m \simeq m' * S \nVdash R\end{array}}{\forall \vec{u}.\, C \nVdash k \xmapsto{[b;m)} \langle o, \alpha\rangle * M \vdash \exists \vec{x}.\, l \xmapsto{[b';m')} \langle n, \alpha'\rangle * S \nVdash \exists \alpha_0'.\, R}$$

$$\frac{\begin{array}{l}\lfloor\!\rfloor \vdash k < l * k{+}o \simeq l{+}n \\ \quad \forall \vec{u}, \alpha_0, \alpha_1.\, C * l \xmapsto{[b;m)} \langle n, \alpha_1\rangle \\ \quad \nVdash k \xmapsto{[b;m)} \langle l{-}k, \alpha_0\rangle * \langle o, \alpha\rangle \simeq \langle l{-}k, \alpha_0\rangle^\frown \langle n, \alpha_1\rangle * M \\ \quad \vdash \exists \vec{x}.\, b \simeq b' * m \simeq m' * \alpha_1 \simeq \alpha' * S \nVdash R\end{array}}{\forall \vec{u}.\, C \nVdash k \xmapsto{[b;m)} \langle o, \alpha\rangle * M \vdash \exists \vec{x}.\, l \xmapsto{[b';m')} \langle n, \alpha'\rangle * S \nVdash \exists \alpha_0, \alpha_1.\, R}$$

Frame Inference Solver $\leftrightarrow$ First-Order Solver

First-order reasoning needed for frame inference:

- $\pi \vdash \_ \simeq \_$: equality queries (integers, byte-arrays, ...)
- $\pi \vdash \_ < \_$: inequality queries (integers)
- $\pi \vdash \mathsf{false}$: FO inconsistency checking
- $\pi \vdash \exists \vec{x}.\, S$: FO entailment

Canonizing First-Order Theory Solver: Interface

"Shostak-style" first-order theory solver:

- σ: "solution set" of equations maintained by solver
- $\texttt{assert}\,(a \simeq b)\,\sigma$: conjoin $a \simeq b$ to σ, yield $\bot$ if unsat or σ' ow
- $a\sigma$: canonical form of term a under equations σ

Answer queries:

- $P \vdash \mathsf{false}$: $\texttt{assert}\,P\,\emptyset = \bot$
- $P \vdash a \simeq b$: $\text{let } \sigma = \texttt{assert}\,P\,\emptyset \text{ in } a\sigma = b\sigma$
- $P \vdash a < b$: $\text{let } \sigma = \texttt{assert}\,P\,\emptyset \text{ in } (b-a)\sigma \text{ is a positive integer}$
- $P \vdash b$: $\text{let } \sigma = \texttt{assert}\,P\,\emptyset \text{ in } b\sigma = \mathsf{true}$

May need to handle Boolean combinations in P: either in frame inference solver or directly on σ (perhaps inefficiently).

Canonizing First-Order Theory Solver: Canonizer

η is a *canonizer* if:

- $\vDash a \simeq b$ iff $\eta a = \eta b$
- $fv(\eta a) \subseteq \mathrm{fv}(a)$
- $\eta b = b$ for each subterm b of ηa

For linear arithmetic, rewriting into ordered sum of monomials is a canonizer.
E.g.:

$$\eta\,(x + z + w - y - w + z) = x - y + 2z$$

Canonizing First-Order Theory Solver: Solver

`solve` is a *solver* if:

- `solve` $(a \simeq b) = \bot$ iff $a \simeq b$ is unsat
- `solve` $(a \simeq b) = \sigma$ for σ a solution substitution/set s.t.:
  - $\operatorname{dom} \sigma \subseteq \operatorname{fv}(a \simeq b)$
  - σ is idempotent
  - σ and $a \simeq b$ are logically equivalent

For linear arithmetic, solving for one variable gives a solver.
E.g.:

$$\texttt{solve}\ (2x - 3y + 2z = 5y) = [x \mapsto 4y - z]$$

Byte-Array Canonizer

Rewrite system generated by:

$$\langle n, \alpha \rangle \Rightarrow \alpha \text{ when } n \equiv |\alpha|$$

$$\_[\_;0) \Rightarrow \langle\rangle$$

$$\alpha[m;k)[o;l) \Rightarrow \alpha[m+o;l) \text{ when } k \geqslant o+l$$

$$e^n[o;l) \Rightarrow e^l \text{ when } n \geqslant o+l$$

$$\langle n, \alpha \rangle [0;n) \Rightarrow \langle n, \alpha \rangle$$

$$(\alpha_0 \cdots \alpha_i \cdots \alpha_j \cdots)[|\alpha_0 \cdots \alpha_{i-1}|; |\alpha_i \cdots \alpha_j|) \Rightarrow \alpha_i \cdots \alpha_j$$

$$\alpha^\frown(\beta^\frown\gamma) \Rightarrow \alpha^\frown\beta^\frown\gamma$$

$$\langle n, \alpha \rangle [o;k)^\frown \langle n, \alpha \rangle [o+k;l) \Rightarrow \langle n, \alpha \rangle [o;k+l) \text{ when } n \geqslant o+k+l$$

$$e^{m\frown}e^n \Rightarrow e^{m+n}$$

Byte-Array Solver

$$\langle 0, \alpha \rangle \simeq \beta \Rightarrow \alpha \simeq \langle\rangle \wedge \beta \simeq \langle\rangle \qquad x \simeq \langle n, \alpha \rangle \Rightarrow x \simeq \alpha$$

$$\langle n, \alpha \rangle \simeq \langle m, \beta \rangle \Rightarrow n \simeq m \wedge \alpha \simeq \beta \qquad \langle n, \alpha \rangle \simeq \beta \Rightarrow n \simeq |\beta| \wedge \alpha \simeq \beta$$

$$x \simeq \alpha[o;l) \Rightarrow x \mapsto \alpha[o;l) \text{ when } x \notin \text{fv}\, \alpha[o;l)$$

$$x \simeq \alpha[o;l) \Rightarrow \alpha[o;l) \mapsto \langle l, x \rangle \text{ when } x \in \text{fv}\, \alpha[o;l)$$

$$x \simeq \alpha_0 \cdots \alpha_n \Rightarrow x \mapsto \alpha_0 \cdots \alpha_n \text{ when } x \notin \text{fv}\, \alpha_0 \cdots \alpha_n$$

$$x \simeq \alpha_0 \cdots \alpha_n \Rightarrow \langle |\alpha_0 \cdots \alpha_n|, x \rangle \simeq \alpha_0 \cdots \alpha_n \text{ when } x \in \text{fv}\, \alpha_0 \cdots \alpha_n$$

$$\alpha_0 \cdots \alpha_i \cdots \alpha_n \simeq \beta \Rightarrow |\alpha_0 \cdots \alpha_n| \simeq |\beta| \wedge \cdots \wedge \alpha_i \simeq \beta\,[|\alpha_0 \cdots \alpha_{i-1}|;\, |\alpha_i|) \wedge \cdots$$

$$\alpha[o;l) \simeq \beta \Rightarrow l \simeq |\beta| \wedge \alpha \simeq \langle |\alpha|, \chi \rangle [0;o) \frown \beta \frown \langle |\alpha|, \chi \rangle [o+l;|\alpha|-o-l) \quad \text{where } \chi \text{ fresh}$$

Witnessing Existentials

Recall:

$$\frac{\forall \vec{u}, \vec{x}.\ S \vdash \mathsf{emp} \qquad \leftthreetimes \vdash \exists \vec{x}.\ S}{\forall \vec{u}.\ C \mathrel{-\!\!*} M \mathrel{\reflectbox{$\vdash$}} \exists \vec{x}.\ S \mathrel{\reflectbox{$-\!\!*$}} M * S}$$

Use FO solver's solution set for $\leftthreetimes$ to compute σ

- if $X\sigma = U$ then $\mathrm{fv}\, X \cap \vec{x} \neq \emptyset$ and $\mathrm{fv}\, U \cap \vec{x} = \emptyset$
- enumerating congruence classes
- solving polynomial equations for existential variables
- solving for byte-arrays by concatenating extractions:

$$\alpha \simeq \langle m, x \rangle[0;n) \ \wedge\ \beta \simeq \langle m, x \rangle[n;m-n) \quad \models \quad x \simeq \alpha \frown \beta$$

$$\frac{\forall \vec{u}, \vec{k}.\ C \mathrel{-\!\!*} M * \sigma_m \mathrel{\reflectbox{$\vdash$}} \exists \vec{x} \setminus \vec{k}.\ \sigma_s * S\sigma \mathrel{\reflectbox{$-\!\!*$}} R}{\forall \vec{u}.\ C \mathrel{-\!\!*} M \mathrel{\reflectbox{$\vdash$}} \exists \vec{x}.\ S \mathrel{\reflectbox{$-\!\!*$}} R}$$

$$\leftthreetimes \models \sigma \qquad\qquad \vec{k} = \vec{x} \setminus \mathrm{fv}\, S\sigma$$

$\sigma_m \wedge \sigma_s = \sigma$ where σ_m is (maximal) s.t. $(C * M) \models \exists \vec{k}.\ \sigma_m$

Eliminating Existentials

- Performed regularly during symbolic execution / analysis
- Rewrites formulas $\exists \vec{x}.\, S$ to $\exists \vec{x} \setminus \vec{k}.\, \sigma_s * S\sigma$ similarly to solver
- Operates directly on NNF
- Existential quantification used for scope management
  - prevent constraints on overwritten values of variables from interfering with constraints on their current values (Floyd's assignment axiom and cousins)
  - logical analogue of shadowing variables when pushing a stack frame
  - out-scoping local variables when popping stack frames
  - procedure parameter passing
- "Garbage collects" constraints about execution history

Parting Thoughts

- LLVM makes C++ approachable
- Fuzzer harnesses lower overhead of applying tools to existing code
- Analyzing bitcode requires more arithmetic support than might otherwise be strictly necessary
- Something like the theory for byte-arrays to manipulate memory contents is needed, seems to work, simpler would be better
- Interaction between Shostak-style first-order solvers and separation logic proof search works nicely
- Redundant quantifier elimination is mandatory