Here be Wyverns!
Verifying LLVM-IR with llStar

Jules Villard

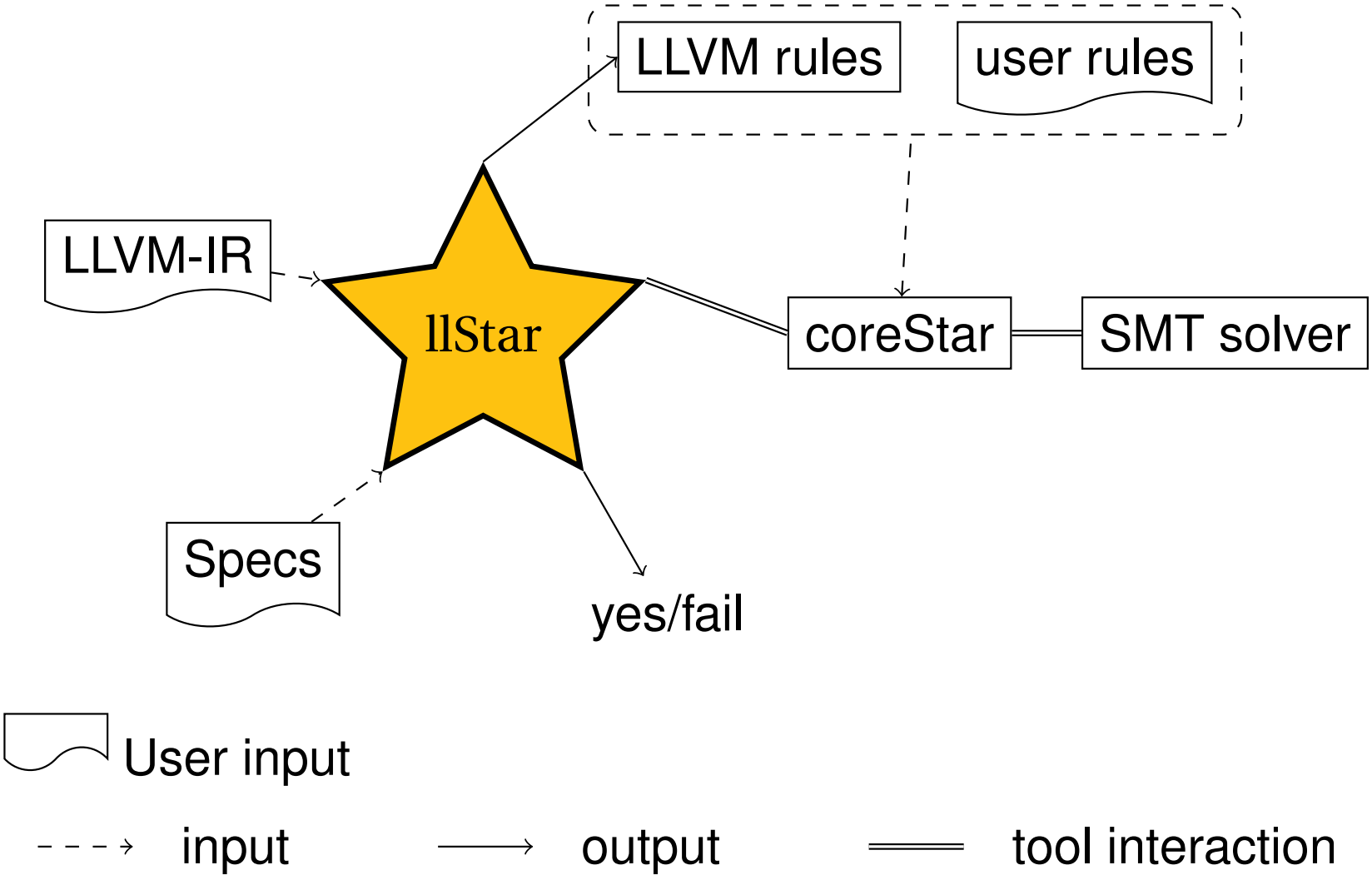
Imperial College London

http://llstar.pauvre.org/

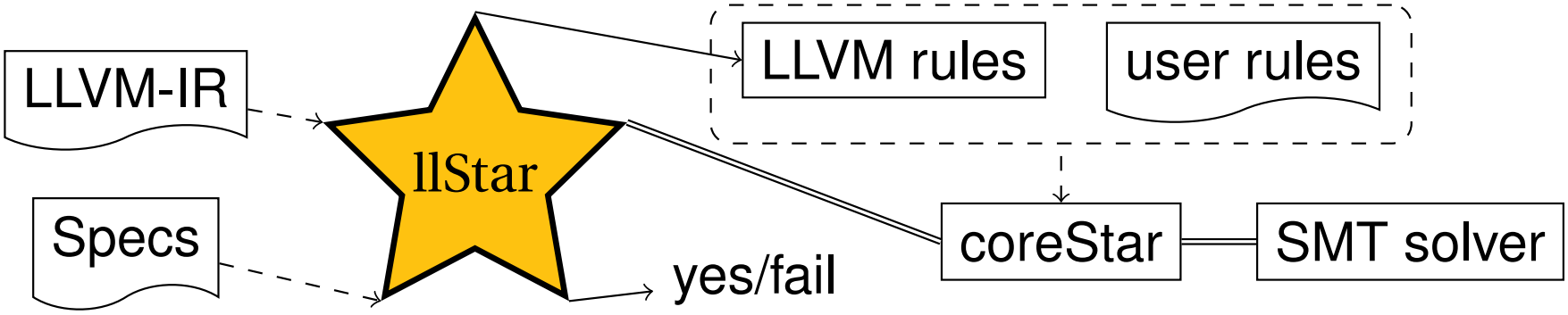
LLVM-IR

- LLVM is a compiler framework (front-ends, optimisations, JIT, assembler, . . .)
- Common intermediate language: LLVM-IR
- Front-ends to C, C++, Fortran, . . .

Overview of llStar

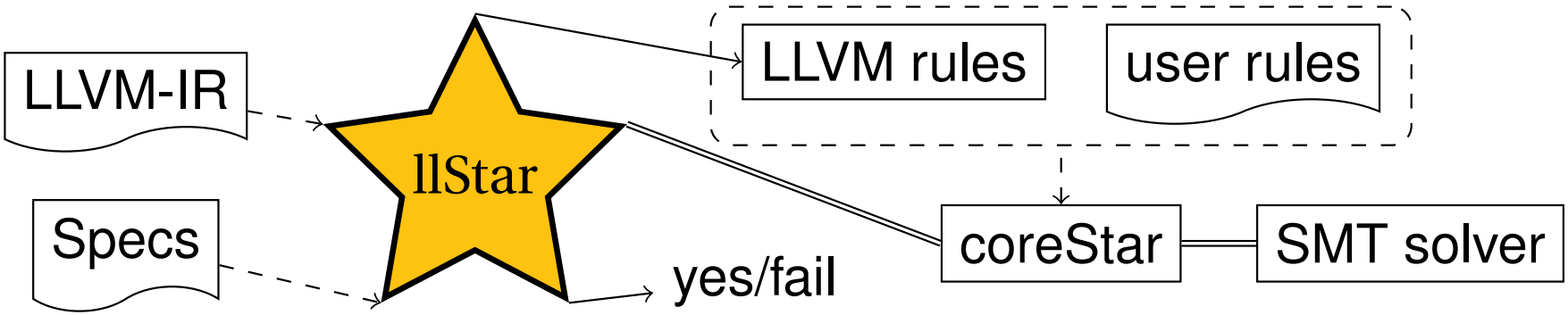


Architecture



LLVM-IR The LLVM module to analyse (LLVM 3.4)

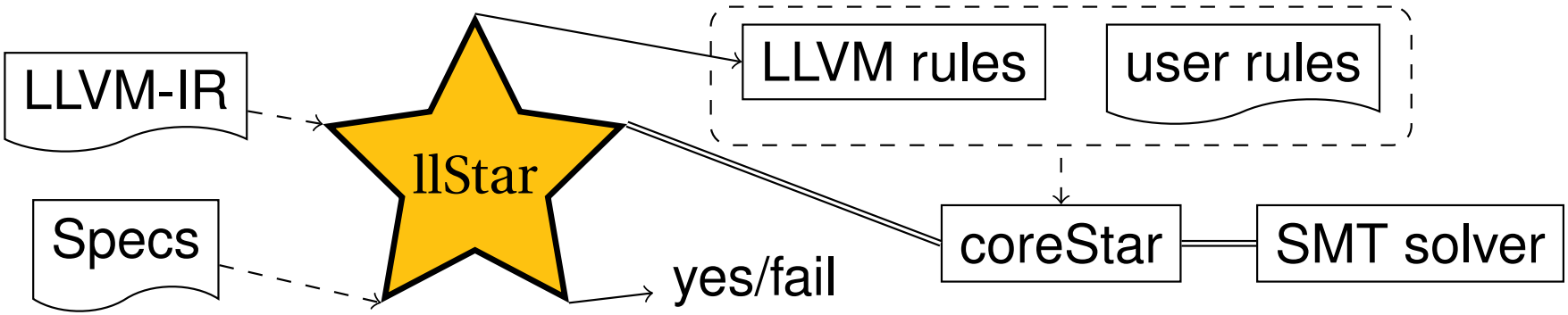
Architecture



LLVM-IR The LLVM module to analyse (LLVM 3.4)

Specs Separation logic pre/post-conditions for functions

Architecture

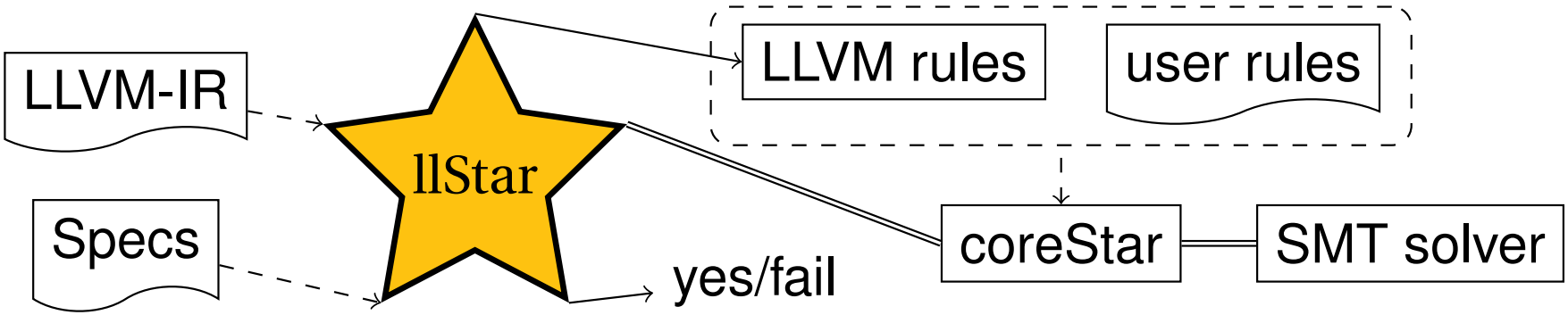


LLVM-IR — The LLVM module to analyse (LLVM 3.4)

Specs — Separation logic pre/post-conditions for functions

coreStar — Symbolic execution and frame inference engine for separation logic.

Architecture



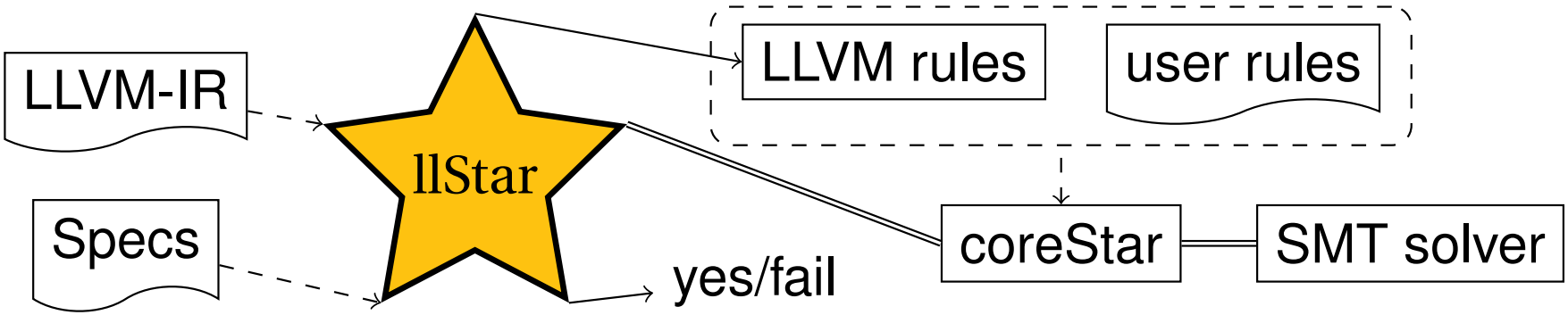
LLVM-IR — The LLVM module to analyse (LLVM 3.4)

Specs — Separation logic pre/post-conditions for functions

coreStar — Symbolic execution and frame inference engine for separation logic.

LLVM rules — Reasoning rules for pointers, structures, etc.

Architecture



LLVM-IR — The LLVM module to analyse (LLVM 3.4)

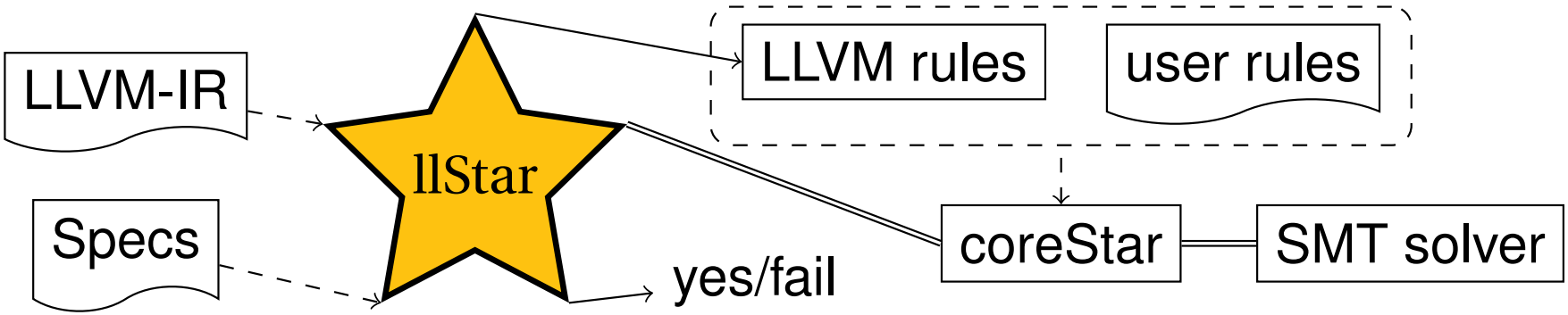
Specs — Separation logic pre/post-conditions for functions

coreStar — Symbolic execution and frame inference engine for separation logic.

LLVM rules — Reasoning rules for pointers, structures, etc.

user rules — Reasoning rules for inductive data structures.

Architecture



LLVM-IR — The LLVM module to analyse (LLVM 3.4)

Specs — Separation logic pre/post-conditions for functions

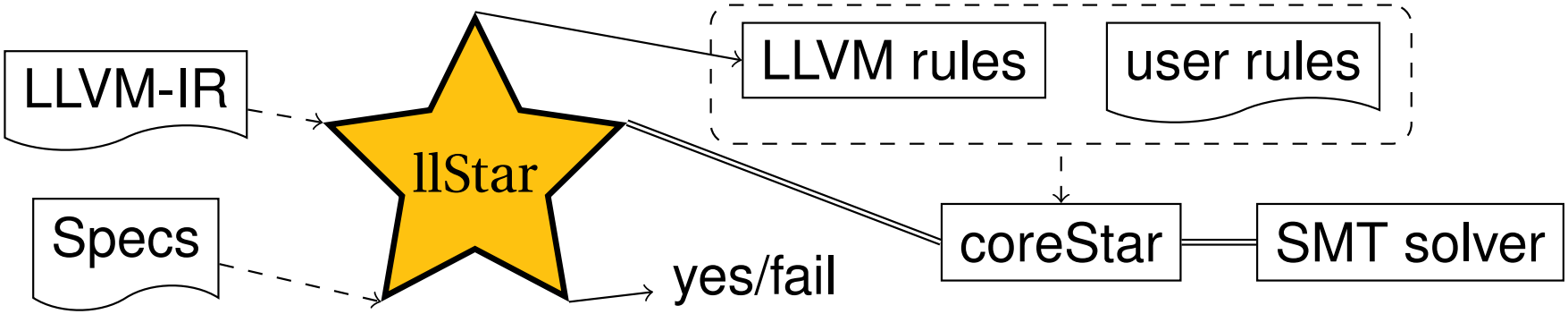
coreStar — Symbolic execution and frame inference engine for separation logic.

LLVM rules — Reasoning rules for pointers, structures, etc.

user rules — Reasoning rules for inductive data structures.

SMT solver — Z3 checks formulas over **pure** values (bit-vectors, structs, arrays, floats)

Architecture



LLVM-IR	The LLVM module to analyse (LLVM 3.4)
Specs	Separation logic pre/post-conditions for functions
coreStar	Symbolic execution and frame inference engine for separation logic.
LLVM rules	Reasoning rules for pointers, structures, etc.
user rules	Reasoning rules for inductive data structures.
SMT solver	Z3 checks formulas over **pure** values (bit-vectors, structs, arrays, floats)
yes/fail	proved specs + memory safety / failed proof

Symbolic states

llStar and coreStar use **separation logic** formulas to describe the state of the heap and variables during program execution.

$A ::= e_1 = e_2 \mid e_1 \neq e_2 \mid e_1 >_{sbv} e_2 \mid \ldots$	pure predicates
$\mid$ emp	empty heap
$\mid e_p \overset{t}{\mapsto} e_v$	points-to (with type)
$\mid$ malloced(e_p, e_s)	malloc block (with size)
$\mid \ldots$	more spatial predicates
$\mid A_1 * A_2 \mid A_1 \vee A_2$	logical connectives

Symbolic states

llStar and coreStar use **separation logic** formulas to describe the state of the heap and variables during program execution.

$$
\begin{array}{lll}
A ::= e_1 = e_2 \mid e_1 \neq e_2 \mid e_1 >_{sbv} e_2 \mid \ldots & & \text{pure predicates} \\
\mid \text{emp} & & \text{empty heap} \\
\mid e_p \overset{t}{\mapsto} e_v & & \text{points-to (with type)} \\
\mid \text{malloced}(e_p, e_s) & & \text{malloc block (with size)} \\
\mid \ldots & & \text{more spatial predicates} \\
\mid A_1 * A_2 \mid A_1 \vee A_2 & & \text{logical connectives}
\end{array}
$$

- Pure predicates: values only, no heap

Symbolic states

llStar and coreStar use **separation logic** formulas to describe the state of the heap and variables during program execution.

$A ::= e_1 = e_2 \mid e_1 \neq e_2 \mid e_1 >_{sbv} e_2 \mid \ldots$	pure predicates
$\mid \mathsf{emp}$	empty heap
$\mid e_p \overset{t}{\mapsto} e_v$	points-to (with type)
$\mid \mathsf{malloced}(e_p, e_s)$	malloc block (with size)
$\mid \ldots$	more spatial predicates
$\mid A_1 * A_2 \mid A_1 \vee A_2$	logical connectives

- Pure predicates: values only, no heap
- Classical conjunction $\wedge$ and SL conjunction $*$ coincide on pure facts:

$$x = \texttt{bv\_const(32, 4)} * y = x \Leftrightarrow x = \texttt{bv\_const(32, 4)} \wedge y = x$$

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson
- **New!** Extended to reason about LLVM values as translated by llStar:

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson
- **New!** Extended to reason about LLVM values as translated by llStar:
  - LLVM integers and operations (`add`/`mul`/`shl`/...)
    $\rightsquigarrow$ bitvectors and bitvector arithmetic

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson
- **New!** Extended to reason about LLVM values as translated by llStar:
  - LLVM integers and operations (`add`/`mul`/`shl`/...) $\rightsquigarrow$ bitvectors and bitvector arithmetic
  - LLVM structs $\rightsquigarrow$ records

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson
- **New!** Extended to reason about LLVM values as translated by llStar:
  - LLVM integers and operations (`add`/`mul`/`shl`/...) $\rightsquigarrow$ bitvectors and bitvector arithmetic
  - LLVM structs $\rightsquigarrow$ records
  - LLVM arrays $\rightsquigarrow$ arrays

- Splinter tool from jStar [Distefano, Parkinson'08] (SL for Java)
- Focuses on Java-independent reasoning
- Originally developed by Botincan, Distefano, Dodds, Grigore, Naudziuniene, Parkinson
- **New!** Extended to reason about LLVM values as translated by llStar:
  - LLVM integers and operations (`add`/`mul`/`shl`/...) $\rightsquigarrow$ bitvectors and bitvector arithmetic
  - LLVM structs $\rightsquigarrow$ records
  - LLVM arrays $\rightsquigarrow$ arrays
- Reasoning about values delegated to Z3

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, `...`, @`parameterN:` and $`ret`)

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, `...`, @`parameterN:` and $`ret`)

llStar translates an LLVM module to a coreStar CFG:

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, `...`, @`parameterN:` and $`ret`)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1, v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, `...`, @`parameterN:` and $`ret`)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1\texttt{, } v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

$$\texttt{unreachable} \rightsquigarrow \{\mathit{false}\}\{\mathit{false}\}$$

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, `...`, @`parameterN:` and $`ret`)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1, v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

$$\texttt{unreachable} \rightsquigarrow \{\mathit{false}\}\{\mathit{false}\}$$

$$\texttt{x = load } t* \ v \rightsquigarrow \texttt{x} = \{v \overset{t}{\mapsto} e\}\{\$\texttt{ret} = e * v \overset{t}{\mapsto} e\}$$

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @`parameter0:`, ..., @`parameterN:` and $`ret`)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1, v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

$$\texttt{unreachable} \rightsquigarrow \{\mathit{false}\}\{\mathit{false}\}$$

$$\texttt{x = load } t* \ v \rightsquigarrow \texttt{x} = \{v \overset{t}{\mapsto} e\}\{\$\texttt{ret} = e * v \overset{t}{\mapsto} e\}$$

$$\texttt{store } t* \ v_1 \ v_2 \rightsquigarrow \{v_1 \overset{t}{\mapsto} -\}\{v_1 \overset{t}{\mapsto} v_2\}$$

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @parameter0:, ..., @parameterN: and $ret)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1, v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

$$\texttt{unreachable} \rightsquigarrow \{\mathit{false}\}\{\mathit{false}\}$$

$$\texttt{x = load } t* \ v \rightsquigarrow \texttt{x} = \{v \overset{t}{\mapsto} e\}\{\$\texttt{ret} = e * v \overset{t}{\mapsto} e\}$$

$$\texttt{store } t* \ v_1 \ v_2 \rightsquigarrow \{v_1 \overset{t}{\mapsto} -\}\{v_1 \overset{t}{\mapsto} v_2\}$$

$$\texttt{x = alloca}(t) \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} \overset{t}{\mapsto} x'\}$$

From LLVM-IR to coreStar

coreStar input language

CI	$::= \texttt{label}(l)$	block label
	$\mid \texttt{goto}(l_1, \ldots, l_n)$	non-deterministic jump
	$\mid \texttt{x} = \{A\}\{B\}(e_0, \ldots, e_N)$	Hoare triple

(A, B separation logic formulas with free variables @parameter0:, ..., @parameterN: and $ret)

llStar translates an LLVM module to a coreStar CFG:

$$\texttt{x = add } t\ v_1, \ v_2 \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} = \mathit{bvadd}(v_1, v_2)\}$$

$$\texttt{unreachable} \rightsquigarrow \{\mathit{false}\}\{\mathit{false}\}$$

$$\texttt{x = load } t* \ v \rightsquigarrow \texttt{x} = \{v \overset{t}{\mapsto} e\}\{\$\texttt{ret} = e * v \overset{t}{\mapsto} e\}$$

$$\texttt{store } t* \ v_1 \ v_2 \rightsquigarrow \{v_1 \overset{t}{\mapsto} -\}\{v_1 \overset{t}{\mapsto} v_2\}$$

$$\texttt{x = alloca}(t) \rightsquigarrow \texttt{x} = \{\text{emp}\}\{\$\texttt{ret} \overset{t}{\mapsto} x'\}$$

$$\texttt{x = call } t\ f(v_1, \ \ldots, v_n) \rightsquigarrow \texttt{x} = \{A\}\{B\}(v_1, \ldots, v_n)$$

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$
- Current symbolic state: C

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$
- Current symbolic state: C
- $\Rightarrow$ **Frame problem**: is there F such that:

$$C \vdash A[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F$$

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$
- Current symbolic state: C
- $\Rightarrow$ **Frame problem**: is there F such that:

$$C \vdash A[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F$$

- If F exists, then the resulting state is (x' fresh):

$$(B[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F)[\mathtt{x} \leftarrow x'][\$\mathtt{ret} \leftarrow \mathtt{x}]$$

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$
- Current symbolic state: C
- $\Rightarrow$ **Frame problem**: is there F such that:

$$C \vdash A[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F$$

- If F exists, then the resulting state is (x' fresh):

$$(B[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F)[\mathtt{x} \leftarrow x'][\$\mathtt{ret} \leftarrow \mathtt{x}]$$

- Problem: coreStar doesn't know how to reason about **our spatial predicates**!

coreStar: frame inference

- To execute: $\mathtt{x} = \{A\}\{B\}(e_1, \ldots, e_n)$
- Current symbolic state: C
- $\Rightarrow$ **Frame problem**: is there F such that:

$$C \vdash A[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F$$

- If F exists, then the resulting state is (x' fresh):

$$(B[\mathtt{p}_1, \ldots, \mathtt{p}_n \leftarrow e_1, \ldots, e_n] * F)[\mathtt{x} \leftarrow x'][\$\mathtt{ret} \leftarrow \mathtt{x}]$$

- Problem: coreStar doesn't know how to reason about **our spatial predicates**!
- Solution: coreStar expects **rewrite rules** to help solve frame problems

llStar rules for simple pointers

- Rules transform the current goal $A \vdash B$ to hypotheses $A_1 \vdash B_1, \ldots, A_n \vdash B_n$.

llStar rules for simple pointers

- Rules transform the current goal $A \vdash B$ to hypotheses $A_1 \vdash B_1, \ldots, A_n \vdash B_n$.
- Implicitly rewrites $F * A \vdash B * F'$ to $F * A_1 \vdash B_1 * F', \ldots, F * A_n \vdash B_n * F'$.

llStar rules for simple pointers

- Rules transform the current goal $A \vdash B$ to hypotheses $A_1 \vdash B_1, \ldots, A_n \vdash B_n$.
- Implicitly rewrites $F * A \vdash B * F'$ to $F * A_1 \vdash B_1 * F', \ldots, F * A_n \vdash B_n * F'$.
- Rules for pointer predicate:

$$\frac{}{\texttt{null} \overset{\texttt{?t}}{\longmapsto} \texttt{?v} \vdash}$$

$$\frac{\vdash \texttt{?v} = \texttt{?w}}{\texttt{?x} \overset{\texttt{?t}}{\longmapsto} \texttt{?v} \vdash \texttt{?x} \overset{\texttt{?t}}{\longmapsto} \texttt{?w}}$$

llStar rules for simple pointers

- Rules transform the current goal $A \vdash B$ to hypotheses $A_1 \vdash B_1, \ldots, A_n \vdash B_n$.
- Implicitly rewrites $F * A \vdash B * F'$ to $F * A_1 \vdash B_1 * F', \ldots, F * A_n \vdash B_n * F'$.
- Rules for pointer predicate:

$$\frac{}{\texttt{null} \overset{\texttt{?t}}{\longmapsto} \texttt{?v} \vdash} \qquad \frac{\vdash \texttt{?v} = \texttt{?w}}{\texttt{?x} \overset{\texttt{?t}}{\longmapsto} \texttt{?v} \vdash \texttt{?x} \overset{\texttt{?t}}{\longmapsto} \texttt{?w}}$$

- Pattern variables of the form `?x` can be used to match any expression.

llStar generated rules for structs

llStar automatically generates rules for structs. For instance, for `%s = type { i32, %s* }`

$$\mathit{sizeof}(\texttt{\%s}) = 16$$

llStar generated rules for structs

llStar automatically generates rules for structs. For instance, for `%s = type { i32, %s* }`

$$\mathit{sizeof}(\texttt{\%s}) = 16$$

$$\frac{\texttt{?x} \xrightarrow{\texttt{i32}} \mathit{field}^{0}_{\texttt{\%s}}(\texttt{?v}) * \texttt{?x} + 8 \xrightarrow{\texttt{\%s*}} \mathit{field}^{1}_{\texttt{\%s}}(\texttt{?v}) \vdash \texttt{?x} \xrightarrow{\texttt{i32}} \texttt{?w}}{\texttt{?x} \xrightarrow{\texttt{\%s}} \texttt{?v} \vdash \texttt{?x} \xrightarrow{\texttt{i32}} \texttt{?w}}$$

llStar generated rules for structs

llStar automatically generates rules for structs. For instance, for `%s = type { i32, %s* }`

$$\mathit{sizeof}(\texttt{\%s}) = 16$$

$$\frac{\texttt{?x} \xmapsto{\texttt{i32}} \mathit{field}^{0}_{\texttt{\%s}}(\texttt{?v}) * \texttt{?x} + 8 \xmapsto{\texttt{\%s*}} \mathit{field}^{1}_{\texttt{\%s}}(\texttt{?v}) \vdash \texttt{?x} \xmapsto{\texttt{i32}} \texttt{?w}}{\texttt{?x} \xmapsto{\texttt{\%s}} \texttt{?v} \vdash \texttt{?x} \xmapsto{\texttt{i32}} \texttt{?w}}$$

$$\frac{\texttt{?x} \xmapsto{\texttt{i32}} \mathit{field}^{0}_{\texttt{\%s}}(\texttt{?v}) * \texttt{?x} + 8 \xmapsto{\texttt{\%s*}} \mathit{field}^{1}_{\texttt{\%s}}(\texttt{?v}) \vdash \texttt{?x} + 8 \xmapsto{\texttt{\%s*}} \texttt{?w}}{\texttt{?x} \xmapsto{\texttt{\%s}} \texttt{?v} \vdash \texttt{?x} + 8 \xmapsto{\texttt{\%s*}} \texttt{?w}}$$

Translating `getelementptr`

- The LLVM instruction does pointer arithmetic (no heap access).

$$\texttt{x = getelementptr } t* \ v, \ t_1 \ v_1, \ \ldots, \ t_n \ v_n$$
$$\rightsquigarrow \texttt{x} = \{\mathsf{emp}\}\{\$\texttt{ret} = \mathit{eltptr}(v, t*, [v_1; \ldots; v_n])\}$$

Translating `getelementptr`

- The LLVM instruction does pointer arithmetic (no heap access).

$$\texttt{x = getelementptr } t\texttt{*} \ v\texttt{,} \ t_1 \ v_1\texttt{,} \ \ldots\texttt{,} \ t_n \ v_n$$
$$\rightsquigarrow \texttt{x} = \{\mathsf{emp}\}\{\$\texttt{ret} = \mathit{eltptr}(v, t\texttt{*}, [v_1; \ldots; v_n])\}$$

- Reasoning rules (`%s = type { i32, %s* }`):

$$\mathit{eltptr}(\texttt{?x}, \texttt{?t}, []) = \texttt{?x}$$

Translating `getelementptr`

- The LLVM instruction does pointer arithmetic (no heap access).

$$\texttt{x = getelementptr } t* \ v, \ t_1 \ v_1, \ \ldots, \ t_n \ v_n$$
$$\rightsquigarrow \texttt{x} = \{\mathsf{emp}\}\{\$\texttt{ret} = \mathit{eltptr}(v, t*, [v_1; \ldots; v_n])\}$$

- Reasoning rules (`%s = type { i32, %s* }`):

$$\mathit{eltptr}(\texttt{?x}, \texttt{?t}, []) = \texttt{?x}$$

$$\mathit{eltptr}(\texttt{?x}, \texttt{\%s}, 0 :: \texttt{?j}) = \mathit{eltptr}(\texttt{?x}, \texttt{i32}, \texttt{?j})$$

Translating `getelementptr`

- The LLVM instruction does pointer arithmetic (no heap access).

$$\texttt{x = getelementptr } t\texttt{*} \ v\texttt{,} \ t_1 \ v_1\texttt{,} \ \ldots\texttt{,} \ t_n \ v_n$$
$$\rightsquigarrow \texttt{x} = \{\mathsf{emp}\}\{\texttt{\$ret} = \mathit{eltptr}(v, t\texttt{*}, [v_1; \ldots; v_n])\}$$

- Reasoning rules (`%s = type { i32, %s* }`):

$$\mathit{eltptr}(\texttt{?x}, \texttt{?t}, []) = \texttt{?x}$$

$$\mathit{eltptr}(\texttt{?x}, \texttt{\%s}, 0 :: \texttt{?j}) = \mathit{eltptr}(\texttt{?x}, \texttt{i32}, \texttt{?j})$$

$$\mathit{eltptr}(\texttt{?x}, \texttt{\%s}, 1 :: \texttt{?j}) = \mathit{eltptr}(\texttt{?x} + 8, \texttt{\%s*}, \texttt{?j})$$

Extending llStar

- llStar does symbolic execution for LLVM-IR

Extending llStar

- llStar does symbolic execution for LLVM-IR
- Based on a flexible reasoning engine: coreStar

Extending llStar

- llStar does symbolic execution for LLVM-IR
- Based on a flexible reasoning engine: coreStar
- llStar generates reasoning rules for LLVM datatypes

Extending llStar

- llStar does symbolic execution for LLVM-IR
- Based on a flexible reasoning engine: coreStar
- llStar generates reasoning rules for LLVM datatypes
- Easily extensible with more data structures

Extending llStar

- llStar does symbolic execution for LLVM-IR
- Based on a flexible reasoning engine: coreStar
- llStar generates reasoning rules for LLVM datatypes
- Easily extensible with more data structures
- . . . Just add more reasoning rules!

Conclusion

Summary

- llStar can **verify** small/medium sequential LLVM-IR programs, and reason about their low-level features

Summary

- llStar can **verify** small/medium sequential LLVM-IR programs, and reason about their low-level features
- **Compositional reasoning** using separation logic: per-function, bottom-up verification, no `main()` needed, etc.

Summary

- llStar can **verify** small/medium sequential LLVM-IR programs, and reason about their low-level features
- **Compositional reasoning** using separation logic: per-function, bottom-up verification, no `main()` needed, etc.
- **Extensible** reasoning engine enables custom theories to be quickly added and tested on real code

Summary

- llStar can **verify** small/medium sequential LLVM-IR programs, and reason about their low-level features
- **Compositional reasoning** using separation logic: per-function, bottom-up verification, no `main()` needed, etc.
- **Extensible** reasoning engine enables custom theories to be quickly added and tested on real code
- Cool(?) Web interface

Summary

- llStar can **verify** small/medium sequential LLVM-IR programs, and reason about their low-level features
- **Compositional reasoning** using separation logic: per-function, bottom-up verification, no `main()` needed, etc.
- **Extensible** reasoning engine enables custom theories to be quickly added and tested on real code
- Cool(?) Web interface
- Extended the coreStar tool in the process

Future work

- Concurrency

Future work

- Concurrency
- Nicer syntax for formulas

Future work

- Concurrency
- Nicer syntax for formulas
- Better error reports

Future work

- Concurrency
- Nicer syntax for formulas
- Better error reports
- Make specs of LLVM-IR commands and rule generation also parameterisable

Future work

- Concurrency
- Nicer syntax for formulas
- Better error reports
- Make specs of LLVM-IR commands and rule generation also parameterisable
- Inference of pre/post-conditions using biabduction
  $\Rightarrow$ prove large programs

Future work

- Concurrency
- Nicer syntax for formulas
- Better error reports
- Make specs of LLVM-IR commands and rule generation also parameterisable
- Inference of pre/post-conditions using biabduction
  $\Rightarrow$ prove large programs
- Connect to mechanised semantics (*e.g.* Vellvm)

Future work

- Concurrency
- Nicer syntax for formulas
- Better error reports
- Make specs of LLVM-IR commands and rule generation also parameterisable
- Inference of pre/post-conditions using biabduction
  $\Rightarrow$ prove large programs
- Connect to mechanised semantics (*e.g.* Vellvm)
- Ongoing collaboration to rewrite coreStar entirely with Grigore (Oxford), Peterson (MSR), Tzevelekos (Queen Mary), Gorogiannis (Middlesex)

Here be Wyverns!
Verifying LLVM-IR with llStar

Jules Villard

Imperial College London

http://llstar.pauvre.org/