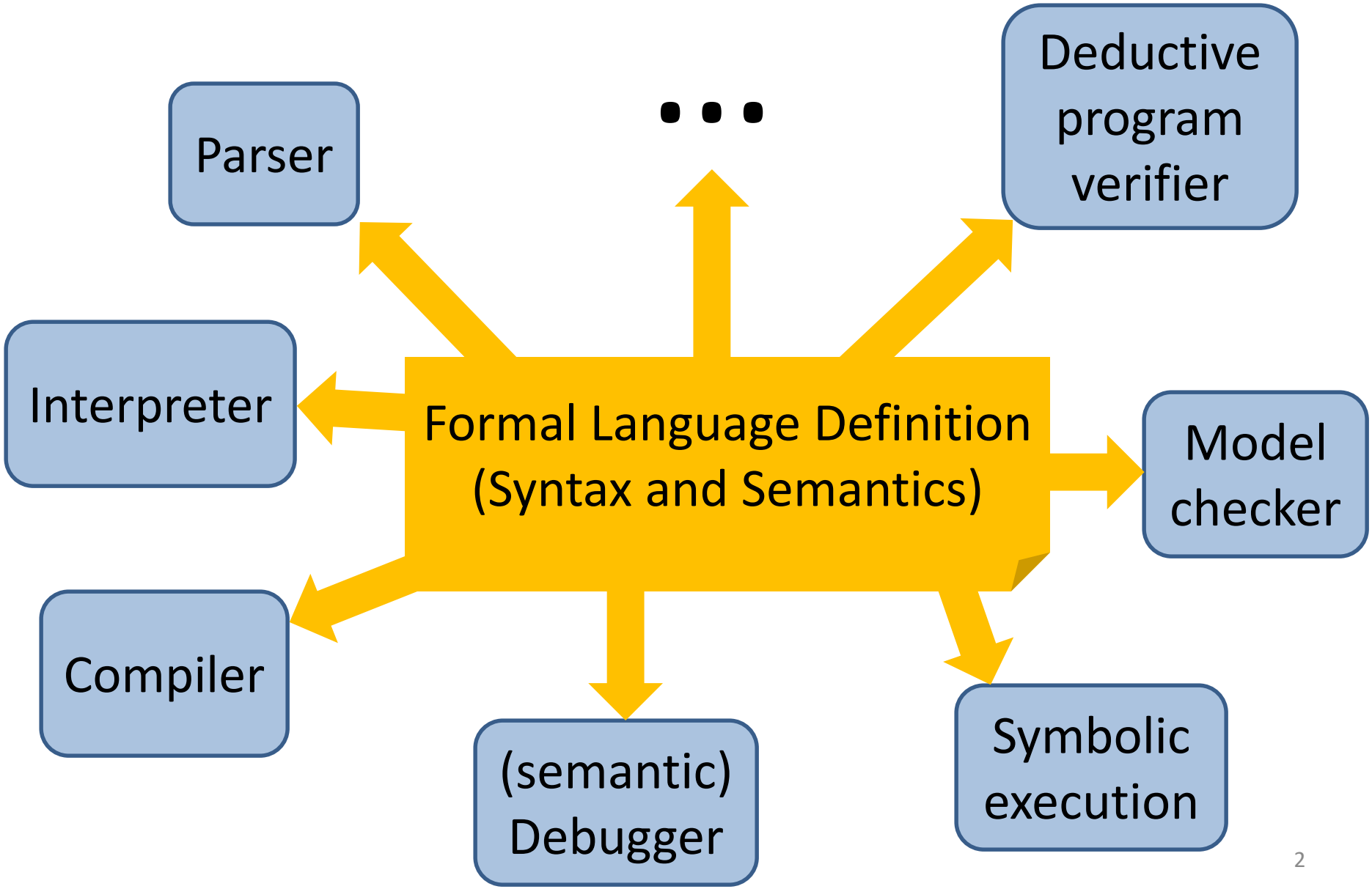
Matching Logic

Grigore Rosu
University of Illinois at Urbana-Champaign, USA

Traian Florin Serbanuta
University of Bucharest, Romania

8 July 2017

Ideal Language Framework Vision



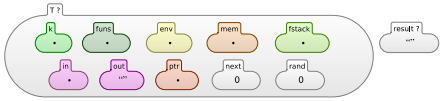
Our Attempt: the K Framework
http://kframework.org

- We tried various semantic styles
  - Small-step and big-step SOS; Evaluation contexts; Chemical abstract machine; Continuation-based style; Denotational; Rewriting logic; …
- But each of the above had limitations
  - Especially related to modularity, notation, verification
- K framework initially *engineered*: keep advantages and avoid limitations of various semantic styles
  - Then theory came

Complete K Definition of KernelC

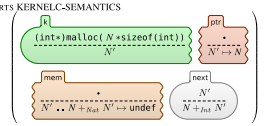
```
MODULE KERNELC-SYNTAX
  IMPORTS K-LATEX+PL-ID+PL-INT
  SYNTAX  Exp ::= Exp + Exp [strict]
                | DeclId
                | Id
                | Int
                | Exp - Exp [strict]
                | Exp ++
                | Exp == Exp [strict]
                | Exp != Exp [strict]
                | Exp <= Exp [strict]
                | Exp < Exp [strict]
                | Exp % Exp [strict]
                | ! Exp
                | Exp && Exp
                | Exp ? Exp : Exp
                | Exp || Exp
                | printf("%d;", Exp ) [strict]
                | scanf("%d",& Exp )
                | scanf("%d", Exp ) [strict]
                | NULL
                | PointerId
                | (int*)malloc( Exp *sizeof(int)) [strict]
                | free( Exp ) [strict]
                | * Exp [strict]
                | Exp [ Exp ]
                | Exp = Exp [strict(2)]
                | Id ( List{Exp} ) [strict(2)]
                | Id ( )
                | random()
                | srandom( Exp ) [strict]
  SYNTAX  Stmt ::= Exp ; [strict]
                | {}
                | { StmtList }
                | if( Exp ) Stmt
                | if( Exp ) Stmt else Stmt [strict(1)]
                | while( Exp ) Stmt
                | return Exp ; [strict]
                | DeclId List{DeclId} { StmtList }
                | #include< StmtList >
  SYNTAX  StmtList ::= StmtList StmtList
                | Stmt
  SYNTAX  Pgm ::= StmtList
  SYNTAX  Id ::= main
  SYNTAX  PointerId ::= * PointerId [ditto]
                | Id
  SYNTAX  DeclId ::= int Exp
                | void PointerId
  SYNTAX  StmtList ::= stdio.h
                | stdlib.h
  SYNTAX  List{Bottom} ::= List{Bottom} , List{Bottom} [assoc hybrid id: ( ) strict]
                | ( )
                | Bottom
  SYNTAX  List{PointerId} ::= List{PointerId} , List{PointerId} [ditto]
                | List{Bottom}
                | PointerId
  SYNTAX  List{DeclId} ::= List{DeclId} , List{DeclId} [ditto]
                | DeclId
                | List{Bottom}
  SYNTAX  List{Exp} ::= List{Exp} , List{Exp} [ditto]
                | Exp
                | List{DeclId}
                | List{PointerId}
END MODULE

MODULE KERNELC-DESUGARED-SYNTAX
  IMPORTS K-LATEX
  IMPORTS KERNELC-SYNTAX
  MACRO  ! E = E ? 0 : 1

  MACRO  E1 && E2 = E1 ? E2 : 0

  MACRO  E1 || E2 = E1 ? 1 : E2

  MACRO  if( E ) St = if( E ) St else {}

  MACRO  NULL = 0

  MACRO  I ( ) = I ( ( ) )

  MACRO  int * PointerId = int PointerId

  MACRO  #include< Stmts > = Stmts

  MACRO  E1 [ E2 ] = * E1 + E2

  MACRO  scanf("%d",& * E ) = scanf("%d", E )

  MACRO  int * PointerId = E = int PointerId = E

  MACRO  int X = E ; = int X ; X = E ;

  MACRO  stdio.h = {}

  MACRO  stdlib.h = {}

END MODULE
```

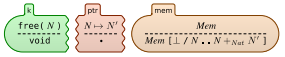
```
MODULE KERNELC-SEMANTICS
  IMPORTS K-SHARED
  IMPORTS K+KERNELC-DESUGARED-SYNTAX+PL-CONVERSION+PL-RANDOM
  CONFIGURATION:
```




```
  RULE  X                          X ↦ V
        V                          V

  RULE  X ++                       X ↦ I
        I                          I +Int 1

  RULE  X = V                      X ↦ _
        V                          V

  RULE  I1 + I2 ⇀ I1 +Int I2

  RULE  I1 - I2 ⇀ I1 −Int I2

  RULE  I1 % I2 ⇀ I1 %Int I2       when I2 !=Int 0

  RULE  I1 <= I2 ⇀ Bool2Int ( I1 ≤Int I2 )

  RULE  I1 < I2 ⇀ Bool2Int ( I1 <Int I2 )

  RULE  I1 == I2 ⇀ Bool2Int ( I1 ==Int I2 )

  RULE  I1 != I2 ⇀ Bool2Int ( I1 !=Int I2 )

  RULE  _?_:_ ⇀ if(_)_else_

  RULE  if( I ) _ else St ⇀ St     when I ==Int 0

  RULE  if( I ) St else _ ⇀ St     when ¬Bool I ==Int 0

  RULE  while( E ) St              if( E ) { St while( E ) St } else {}

  RULE  printf("%d;", I )         S
        void                       S +String Int2String ( I ) +String ";"

  RULE  scanf("%d", N )            I
        + N = I                    •

  RULE  scanf("%d",& X )           I
        X = I                      •

  RULE  V ; ⇀ •

  RULE  { Sts } ⇀ Sts

  RULE  {} ⇀ •

  RULE  St Sts ⇀ St ↷ Sts

  RULE  int X Xl { Sts }           •
        •                          X ↦ int X Xl { Sts }

  RULE  void X Xl { Sts }
        int X                      Sts return void ;
```

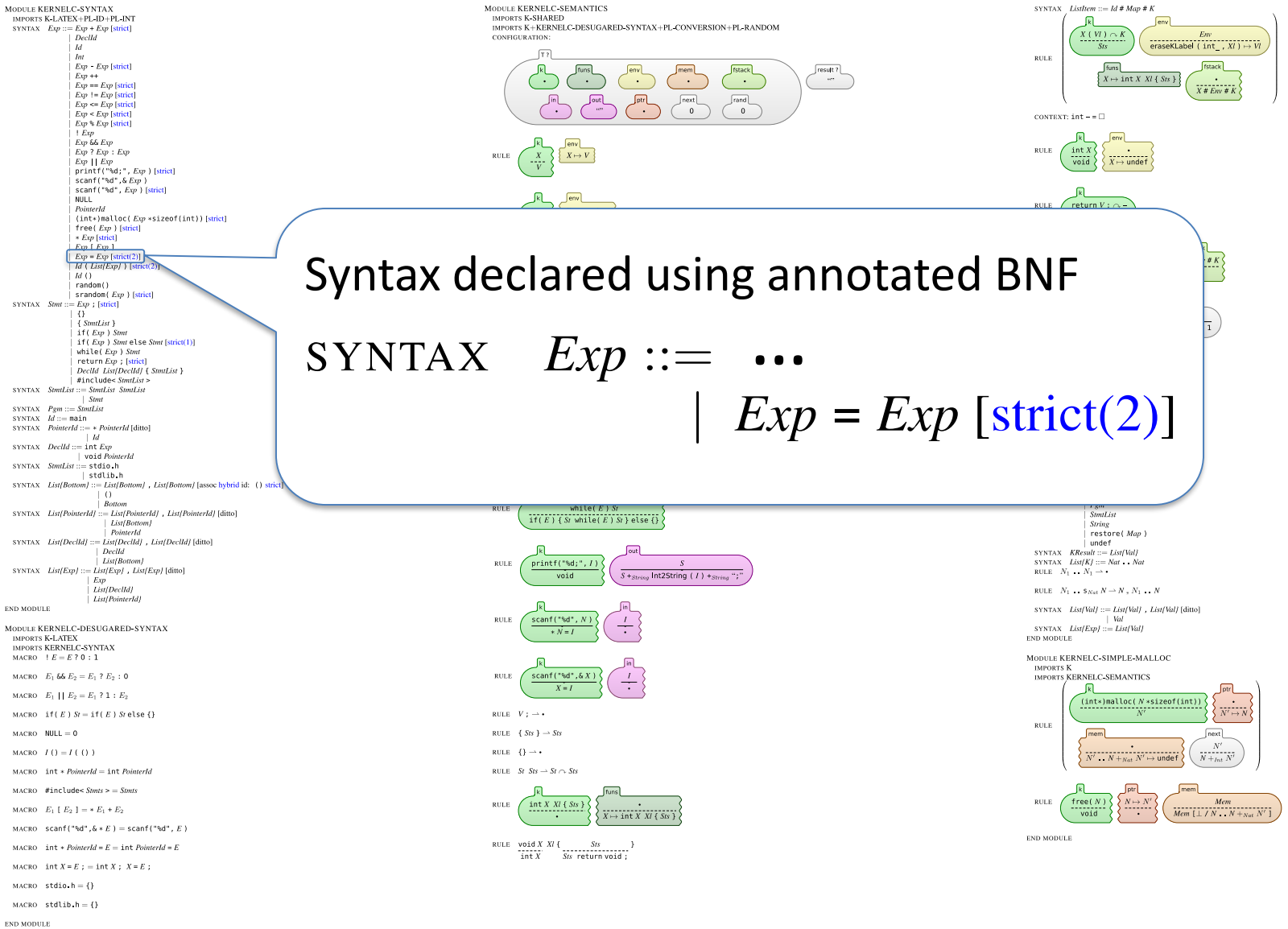
```
  SYNTAX  ListItem ::= Id # Map # K

  RULE    k: X ( Vl ) ↷ K ; Sts
          env: Env ; eraseKLabel ( int _ , Xl ) ↦ Vl
          funs: X ↦ int X Xl { Sts }
          fstack: • ; X # Env # K

  CONTEXT: int _ = □

  RULE    k: int X ; void
          env: • ; X ↦ undef

  RULE    k: return V ; ↷ _ ; V

  RULE    k: V ↷ • ; K
          env: _ ; Env
          fstack: _ # Env # K ; •

  RULE    k: random() ; randomRandom ( N' )
          rand: N' ; N' +Nat 1

  RULE    k: srandom( I ) ; void

  CONTEXT: * □ = _

  CONTEXT: * □ ++

  SYNTAX  Val ::= Int
                | void
  SYNTAX  Exp ::= Val
  SYNTAX  K ::= List{DeclId}
                | List{Exp}
                | List{PointerId}
                | Pgm
                | StmtList
                | String
                | restore( Map )
                | undef
  SYNTAX  KResult ::= List{Val}
  SYNTAX  List{K} ::= Nat .. Nat
  RULE    N1 .. N1 ⇀ •

  RULE    N1 .. sNat N ⇀ N , N1 .. N

  SYNTAX  List{Val} ::= List{Val} , List{Val} [ditto]
                | Val
  SYNTAX  List{Exp} ::= List{Val}
END MODULE

MODULE KERNELC-SIMPLE-MALLOC
  IMPORTS K
  IMPORTS KERNELC-SEMANTICS
```



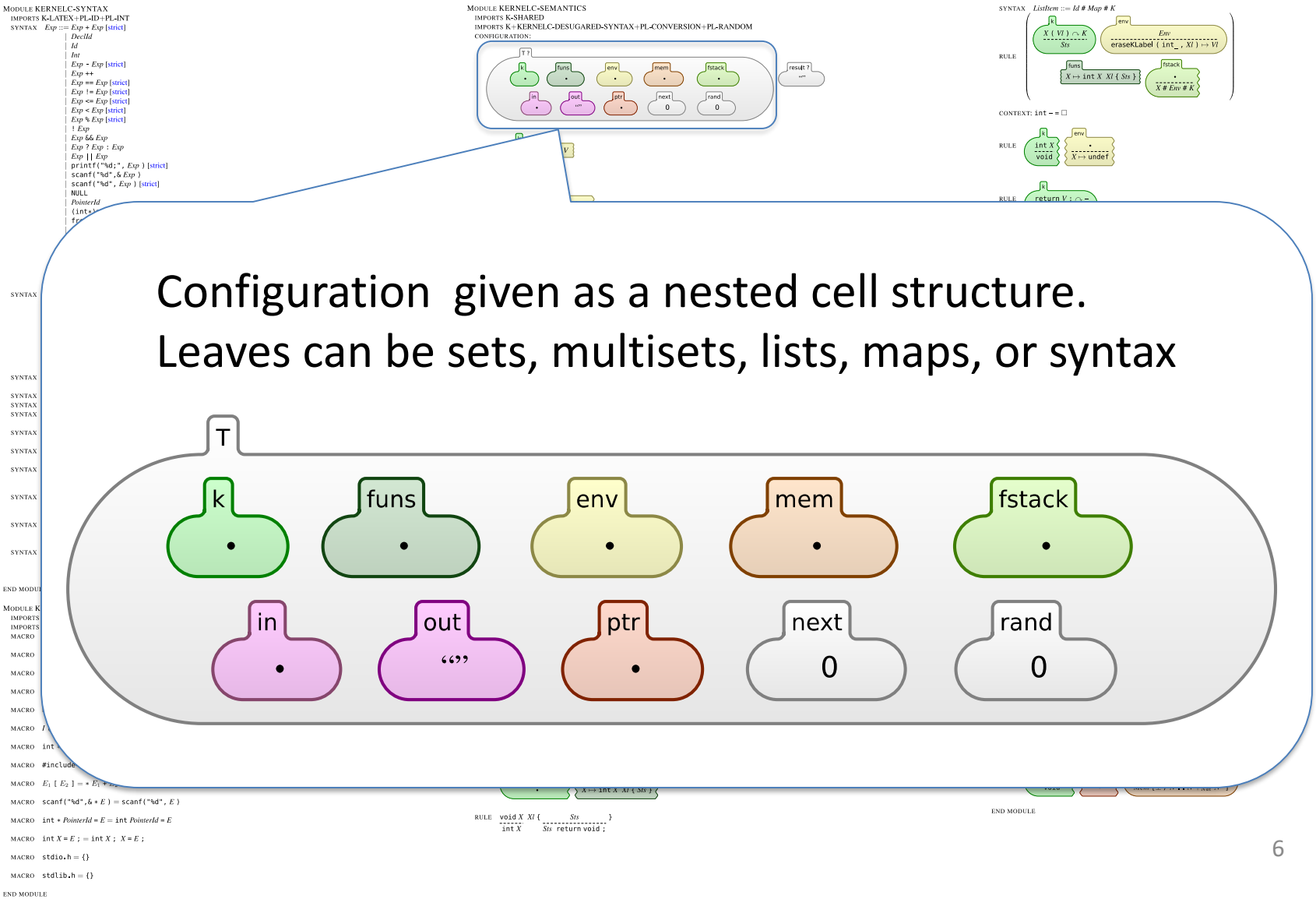



```
END MODULE
```

Complete K Definition of KernelC

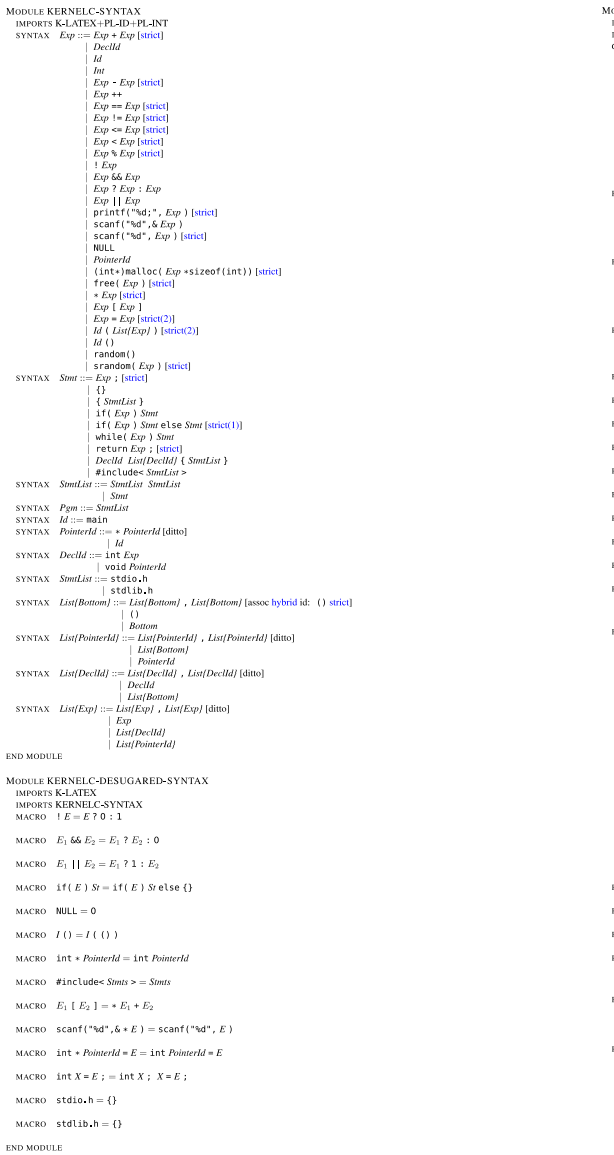


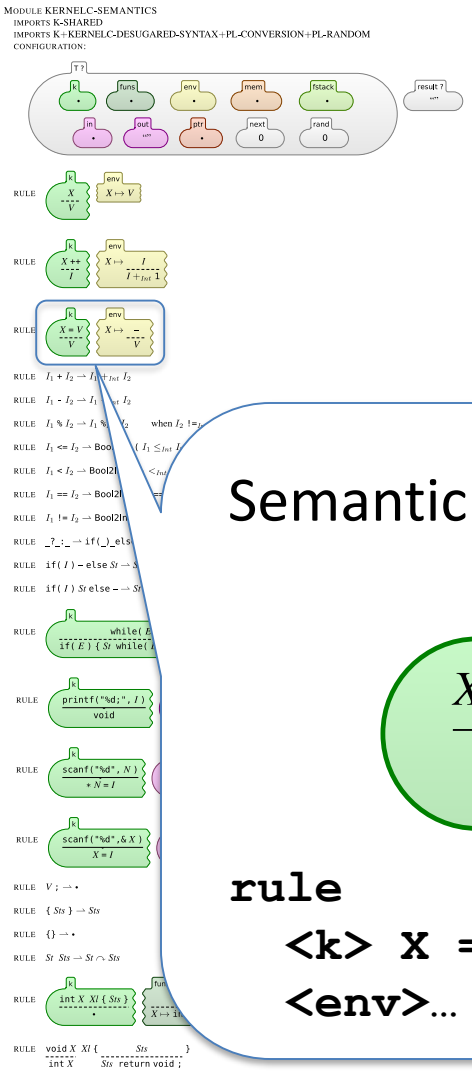
Complete K Definition of KernelC

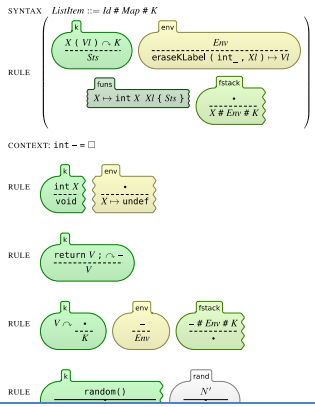


Configuration given as a nested cell structure.
Leaves can be sets, multisets, lists, maps, or syntax

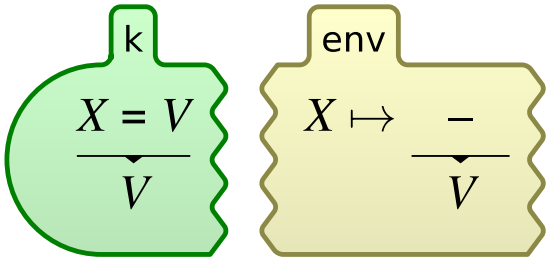
Complete K Definition of KernelC







Semantic rules given contextually




```
rule
  <k> X = V => V ...</k>
  <env>... X |-> (_ => V) ...</env>
```

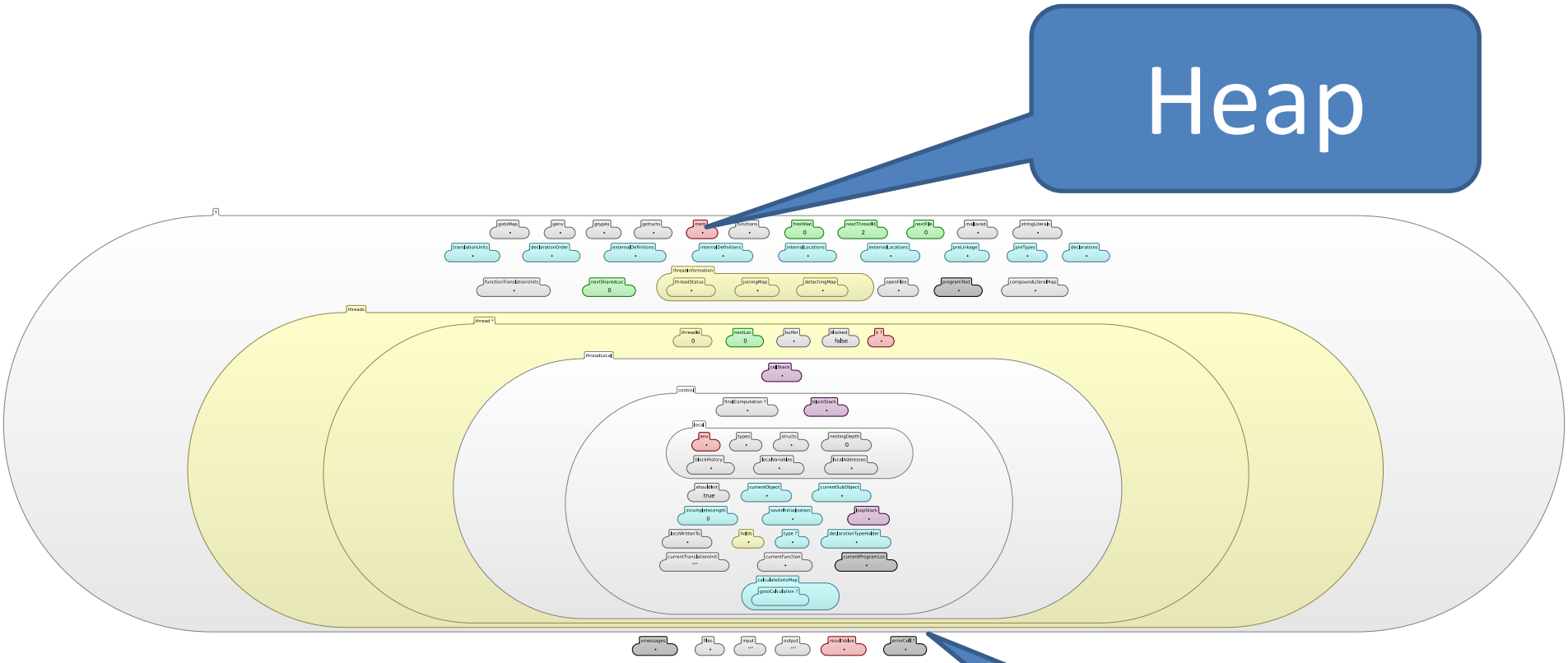
K Scales

Several large languages were recently defined in K:

- Java 1.4: by Bogdanas etal [POPL'15]
  - 800+ program test suite that covers the semantics
- JavaScript ES5: by Park etal [PLDI'15]
  - Passes existing conformance test suite (2872 pgms)
  - Found (confirmed) bugs in Chrome, IE, Firefox, Safari
- C11: Ellison etal [POPL'12, PLDI'15]
  - 192 different types of undefined behavior
  - Commercialized by startup (Runtime Verification, Inc.)

…

K Configuration and Definition of C



… plus ~3500 rules …

RV-Match: Commercial tool based on K[OCAML] with the C semantics

Code (6-int-overflow.c)

…

```
int main() {
  short int a = 1;
  int i;
  for (i = 0; i < 15; i++) {
    a *= 2;
  }
  return a;
}
```

RV-Match gives you:

- an automatic debugger for subtle bugs other tools can't find, with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a platform for analyzing programs, boosting standards compliance and assurance

Conventional compilers do not detect problem

```
$ gcc 6-int-overflow.c
$ ./a.out
$
$ kcc 6-int-overflow.c
$ ./a.out
Error: IMPL-CCV2
Description: Conversion to signed integer outside the range that can be represented.
Type: Implementation defined behavior.
See also: C11 sec. 6.3.1.3:3, J.3.5:1 item 4
  at main(6-int-overflow.c:29)
```

RV-Match's kcc tool precisely detects and reports error, and points to ISO C11 standard

RV-Match on Toyota ITC Benchmark - Comparison with Static Analysis Tools -

- We do not have semantics for "inappropriate code" yet
- We miss defects because inherent limited code coverage of RV
  - No false positives for RV-Match!

[CAV'16]

Shiraishi et al., ISSRE '15	RV-Match			GrammaTech CodeSonar			MathWorks Code Prover			MathWorks Bug Finder			GCC			Clang		
	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM
Static memory	100	100	100	100	100	100	97	100	98	97	100	98	0	100	0	15	100	39
Dynamic memory	94	100	97	89	100	94	92	95	93	90	100	95	0	100	0	0	100	0
Stack-related	100	100	100	0	100	0	60	70	65	15	85	36	0	100	0	0	100	0
Numerical	96	100	98	48	100	69	55	99	74	41	100	64	12	100	35	11	100	33
Resource management	93	100	96	61	100	78	20	90	42	55	100	74	6	100	25	3	100	18
Pointer-related	98	100	99	52	96	71	69	93	80	69	100	83	9	100	30	13	100	36
Concurrency	67	100	82	70	77	73	0	100	0	0	100	0	0	100	0	0	100	0
Inappropriate code	0	100	0	46	99	67	1	97	10	28	94	51	2	100	13	0	100	0
Miscellaneous	63	100	79	69	100	83	83	100	91	69	100	83	11	100	34	11	100	34
AVERAGE (Unweighted)	79	100	89	59	97	76	53	94	71	52	98	71	4	100	20	6	100	24
AVERAGE (Weighted)	82	100	91	68	98	82	53	95	71	62	99	78	5	100	22	7	100	26

DR: Percent of programs with defects where defects are reported

FPR: Percent of programs without defects, with defects incorrectly reported; $\underline{\text{FPR}} = 100 - \text{FPR}$

PM: Productivity metric: $\sqrt{\text{DR} \times (100 - \text{FPR})}$

RV-Match on Toyota ITC Benchmark - Comparison with Other Analysis Tools -

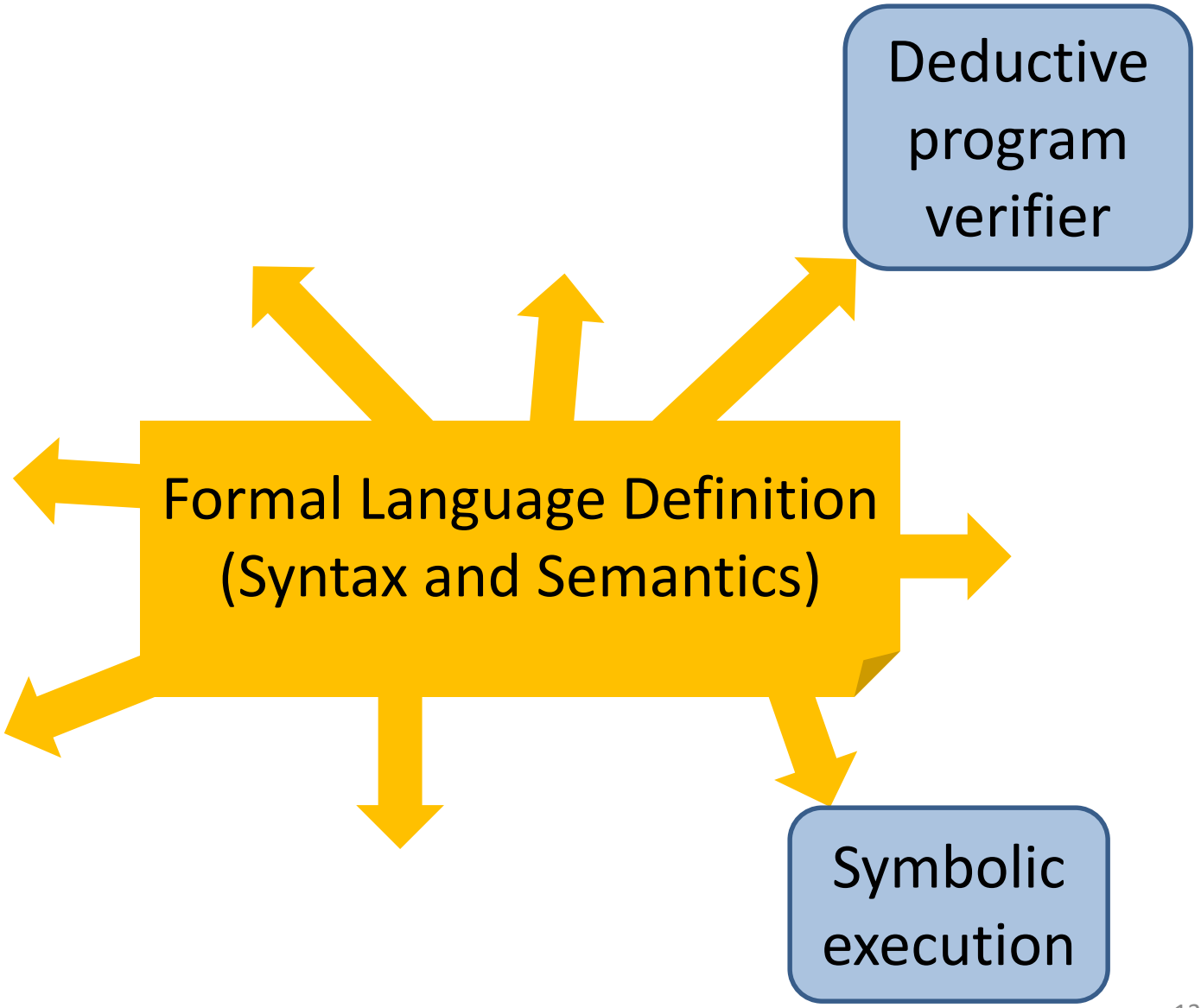
- We have also evaluated other free analysis tools on the Toyota ITC benchmark
- Numbers for other tools may be slightly off; they were not manually checked yet
- Clang cannot be run with UBSan, ASan and TSan together; we ran them separately

Shiraishi et al., ISSRE '15	RV-Match			Valgrind + Helgrind (GCC)			UBSan + TSan + MSan + ASan (Clang)			Frama-C (Value Analysis Plugin)			Compcert Interpreter		
	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM	DR	FPR	PM
Static memory	100	100	100	9	100	30	79	100	89	82	96	89	97	82	89
Dynamic memory	94	100	97	80	95	87	16	95	39	79	27	46	29	80	48
Stack-related	100	100	100	70	80	75	95	75	84	45	65	54	35	70	49
Numerical	96	100	98	22	100	47	59	100	77	79	47	61	48	79	62
Resource management	93	100	96	57	100	76	47	96	67	63	46	54	32	83	52
Pointer-related	98	100	99	60	100	77	58	97	75	81	40	57	87	73	80
Concurrency	67	100	82	72	79	76	67	72	70	7	100	26	58	42	49
Inappropriate code	0	100	0	2	100	13	0	100	0	33	63	45	17	83	38
Miscellaneous	63	100	79	29	100	53	37	100	61	83	49	63	63	71	67
AVERAGE (Unweighted)	79	100	89	44	95	65	51	93	69	61	59	60	52	74	62
AVERAGE (Weighted)	82	100	91	42	97	65	47	95	67	66	55	60	51	76	63

DR: Percent of programs with defects where defects are reported

FPR: Percent of programs without defects, with defects incorrectly reported; $\underline{\text{FPR}} = 100 - \text{FPR}$

PM: Productivity metric: $\sqrt{\text{DR} \times (100 - \text{FPR})}$


Deductive program verifier
Formal Language Definition (Syntax and Semantics)
Symbolic execution

State-of-th

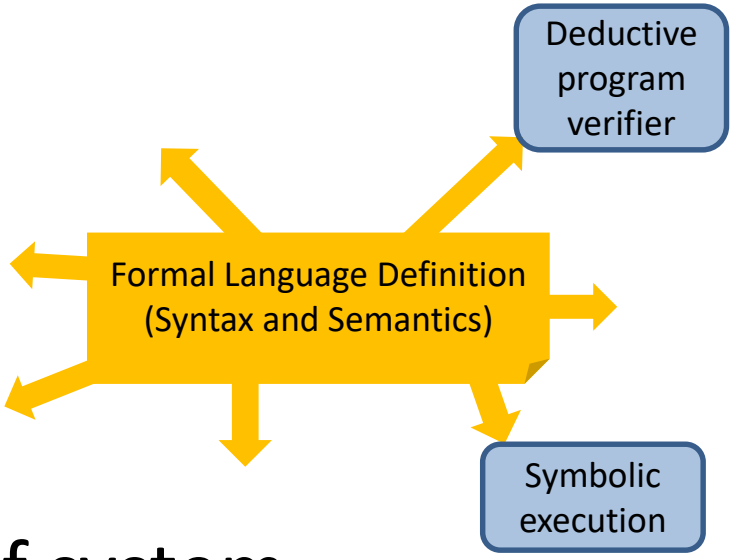
- Redefine the language usi
  approach (Hoare/separatio
- Language specific, non-exec able, error-prone

Many different program logics for "state" properties: FOL, HOL, Separation logic…

$$\frac{\mathcal{H} \vdash \{\psi \wedge \mathtt{e} \neq 0\}\, \mathtt{s}\, \{\psi\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{while(e)\ s}\, \{\psi \wedge \mathtt{e} = 0\}}$$

$$\frac{\mathcal{H} \cup \{\psi\}\, \mathtt{proc()}\, \{\psi'\} \vdash \{\psi\}\, \mathtt{body}\, \{\psi'\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{proc()}\, \{\psi'\}}$$

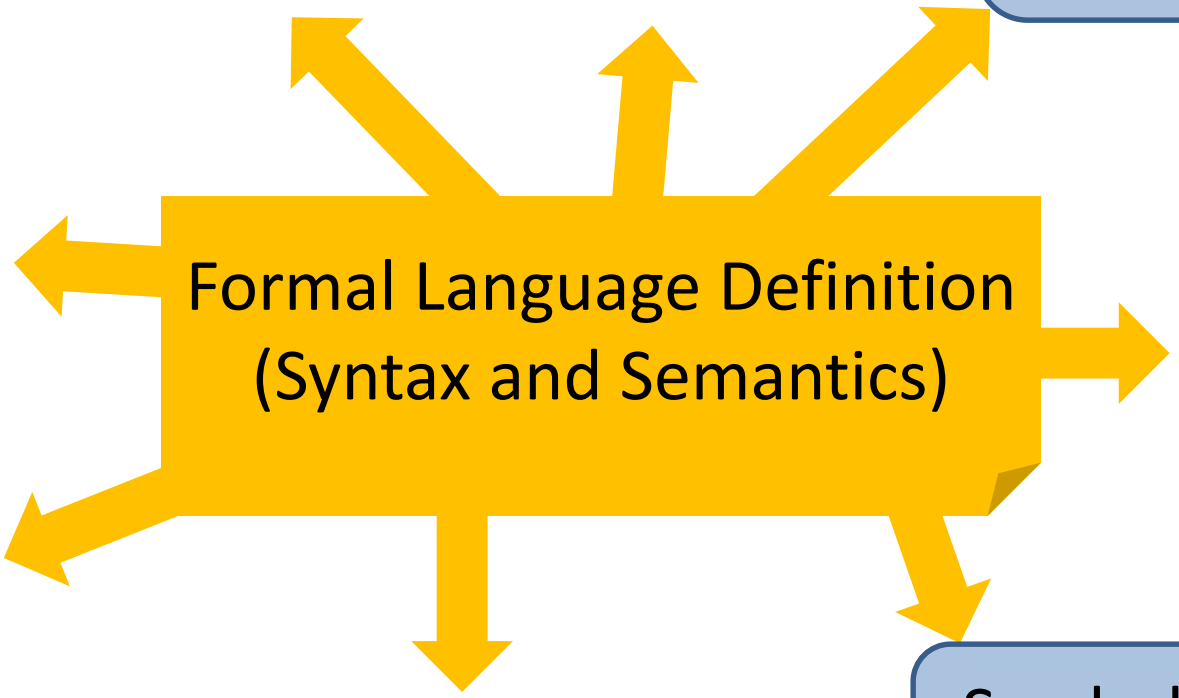
What We Want



- Use directly the trusted executable semantics!
- Language-independent proof system
  - Takes operational semantics as axioms
  - Derives reachability properties
  - Sound and relatively complete for all languages!

Need a means to specify static and dynamic program properties

Deductive program verifier



Symbolic execution

Matching Logic

Patterns (of each sort s)

Structure

Constraints

Binders

$$
\begin{array}{lll}
\varphi_s & ::= & x \in Var_s \\
 & | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
 & | & \neg \varphi_s \\
 & | & \varphi_s \wedge \varphi_s \\
 & | & \exists x \,.\, \varphi_s \quad with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

Matching Logic Models

$$
\begin{array}{rcl}
\varphi_s & ::= & x \in Var_s \\
& | & \sigma(\varphi_{s_1}, ..., \varphi_{s_n}) \quad with\ \sigma \in \Sigma_{s_1 ... s_n, s} \\
& | & \neg \varphi_s \\
& | & \varphi_s \wedge \varphi_s \\
& | & \exists x \,.\, \varphi_s \ with\ x \in Var \quad (of\ any\ sort)
\end{array}
$$

$$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$$

Patterns interpreted as *sets* (all elements that *match* them)

$\neg$ as complement, $\wedge$ as intersection, $\exists$ as union over all *x*

From FOL to Matching Logic

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$$t_s ::= x \in Var_s$$
$$\mid \sigma(t_{s_1}, ..., t_{s_n}) \text{ with } \sigma \in \Sigma_{s_1 ... s_n, s}$$
$$\varphi ::= \pi(t_{s_1}, ..., t_{s_n}) \text{ with } \pi \in \Pi_{s_1 ... s_n, s}$$
$$\mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \text{ with } x \in Var$$

Models

$$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s \text{ for } \sigma \in \Sigma_{s_1 ... s_n, s}$$
$$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n} \text{ for } \pi \in \Pi_{s_1 ... s_n, s}$$

Matching Logic

From FOL to Matching Logic

Collapse symbols

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Matching Logic

From FOL to Matching Logic

Collapse symbols

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

From FOL to Matching Logic

Collapse terms and predicates

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

From FOL to Matching Logic

Collapse terms and predicates

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

Formulae (called Patterns)

$\varphi_s ::= x \in Var_s$

$\quad | \quad \sigma(\varphi_{s_1}, ..., \varphi_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \exists x.\varphi_s$ with $x \in Var$

From FOL to Matching Logic

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Interpret symbols into powerdomains

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

Formulae (called Patterns)

$\varphi_s ::= x \in Var_s$

$\quad | \quad \sigma(\varphi_{s_1}, ..., \varphi_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \exists x.\varphi_s$ with $x \in Var$

From FOL to Matching Logic

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Interpret symbols into powerdomains

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

Formulae (called Patterns)

$\varphi_s ::= x \in Var_s$

$\quad | \quad \sigma(\varphi_{s_1}, ..., \varphi_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \exists x.\varphi_s$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

Matching Logic vs. FOL

First Order Logic

Signatures

S = sorts

Σ = S-sorted operation symbols

Π = S-sorted predicate symbols

Formulae

$t_s ::= x \in Var_s$

$\quad | \quad \sigma(t_{s_1}, ..., t_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\varphi ::= \pi(t_{s_1}, ..., t_{s_n})$ with $\pi \in \Pi_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\pi_M \subseteq M_{s_1} \times \cdots \times M_{s_n}$ for $\pi \in \Pi_{s_1 ... s_n, s}$

Matching Logic

Signatures

S = sorts

Σ = S-sorted symbols

Formulae (called Patterns)

$\varphi_s ::= x \in Var_s$

$\quad | \quad \sigma(\varphi_{s_1}, ..., \varphi_{s_n})$ with $\sigma \in \Sigma_{s_1 ... s_n, s}$

$\quad | \quad \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \exists x.\varphi_s$ with $x \in Var$

Models

$\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ for $\sigma \in \Sigma_{s_1 ... s_n, s}$

Simpler (no need to distinguish operations from predicates)
Same expressiveness, but patterns more succinct than FOL formulae
When fixing model(s), quite a convenient notation (captures separation logic)

Bottom Line

- No distinction between function and predicate symbols; they build patterns
- In models, patterns evaluate to sets of elements, namely those that match them
- Examples
  - $x \in Var_s$, written $x : s$, matched by singletons of sort s
  - $succ(x)$ matched by successor of x
  - $0 \vee \exists x \,.\, succ(x)$ elements are either zero or successors
  - $list(x, S)$ matched by all linked lists in the heap starting with pointer x and holding mathematical sequence S

Useful Sugar

Derived constructs

$$\begin{aligned} \bot_s &\equiv x:s \wedge \neg x:s \\ \top_s &\equiv \neg \bot_s \\ \varphi_1 \vee \varphi_2 &\equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2) \end{aligned}$$

$$\begin{aligned} \varphi_1 \to \varphi_2 &\equiv \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 &\equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1) \\ \forall x.\varphi &\equiv \neg(\exists x.\neg\varphi) \end{aligned}$$

Definedness, equality, membership

$$\lfloor \_ \rfloor_{s_1}^{s_2} \in \Sigma_{s_1,s_2}$$

$$\lfloor x:s_1 \rfloor_{s_1}^{s_2}$$

$$\lfloor \varphi \rfloor_{s_1}^{s_2}$$

Either the empty set (when φ empty) or otherwise the total set (when φ non-empty)

$$\begin{aligned} \varphi =_{s_1}^{s_2} \varphi' &\equiv \neg\lfloor \neg(\varphi \leftrightarrow \varphi') \rfloor_{s_1}^{s_2} \\ x \in_{s_1}^{s_2} \varphi &\equiv \lfloor x \wedge \varphi \rfloor_{s_1}^{s_2} \end{aligned}$$

Sort subscripts and superscripts can be inferred from the context, so we do not write them

More Sugar

Functions (recovering "operation" symbols)

$\sigma \in \Sigma_{s_1 \ldots s_n, s}$

$\exists y \,.\, \sigma(x_1, \ldots, x_n) = y$

We write it using the functional notation:

$\sigma : s_1 \times \cdots \times s_n \to s$

Similarly we can define partial functions, total relations, etc. Algebraic specification and FOL subsumed notationally. For example:

$0 : \to Nat$

$succ : Nat \to Nat$

$plus : Nat \times Nat \to Nat$

$plus(0, y) = y$

$plus(succ(x), y) = succ(plus(x, y))$

Expressiveness

- Important logics for program reasoning can be framed as matching logic theories / notations
  - First-order logic
    - Equality, membership, definedness, partial functions
  - Lambda / mu calculi (least/largest fixed points)
  - Modal logic
  - Hoare logic
  - Separation logic
  - Reachability logic
  - …

Separation logic = Matching logic [Map]

- Consider map model, with some useful axioms

$$\_ \mapsto \_ : Nat \times Nat \rightharpoonup Map \qquad emp * H = H$$
$$emp : \rightarrow Map \qquad H_1 * H_2 = H_2 * H_1$$
$$\_ * \_ : Map \times Map \rightharpoonup Map \qquad (H_1 * H_2) * H_3 = H_1 * (H_2 * H_3)$$
$$0 \mapsto a = \bot \qquad x \mapsto a * x \mapsto b = \bot$$

$$\_ \mapsto [\_] : Nat \times Seq \to Map \qquad x \mapsto [\epsilon] = emp \qquad x \mapsto [a, S] = x \mapsto a * (x + 1) \mapsto [S]$$

- Then we can define map patterns "a la SL"

$$list \in \Sigma_{Nat \times Seq, Map}$$

$$list(0, \epsilon) = emp$$
$$list(x, n \cdot S) = \exists z \,.\, x \mapsto [n, z] * list(z, S)$$

Sound and complete proof system

[RTA'15]

- Sample derivation for the "separation logic" theory:

$$
\begin{array}{lcll}
1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1 & = & 1 \mapsto [5,0] * 7 \mapsto [9,1] & = \\
1 \mapsto [5,0] * \mathit{list}(0,\epsilon) * 7 \mapsto [9,1] & \rightarrow & (\exists z \,.\, 1 \mapsto [5,z] * \mathit{list}(z,\epsilon)) * 7 \mapsto [9,1] & = \\
\mathit{list}(1,5 \cdot \epsilon) * 7 \mapsto [9,1] & = & \mathit{list}(1,5) * 7 \mapsto [9,1] & \rightarrow \\
\exists z \,.\, 7 \mapsto [9,z] \wedge \mathit{list}(z,5) & = & \mathit{list}(7,9 \cdot 5) &
\end{array}
$$

- Local reasoning globalized ("structural framing" for free!)
  - Above derivation can be lifted to whole configuration

$$
\begin{array}{l}
\forall c : \mathit{Cfg}. \forall h : \mathit{Map} \,.\, (\langle\langle 1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1 * h \rangle_{\mathsf{heap}}\ c \rangle_{\mathsf{cfg}} \\
\qquad \rightarrow\ \langle\langle \mathit{list}(7,9 \cdot 5) * h \rangle_{\mathsf{heap}}\ c \rangle_{\mathsf{cfg}})
\end{array}
$$

Reachability Logic
[LICS'13, RTA'14, RTA'15,OOPLSA'16]

- "Rewrite" rules over matching logic patterns:

$$\varphi \Rightarrow \varphi'$$

(generalize to conditional rules)

- Since patterns generalize terms, matching logic reachability rules capture term rewriting rules
- Moreover, deals naturally with side conditions:

$l \Rightarrow r \text{ if } b$ turn into $l \wedge b \Rightarrow r$

Expressiveness of Reachability Rules

- Capture operational semantics rules:

$$\langle\langle \mathtt{x} = i\,;\mathtt{s}\rangle_{\mathsf{k}}\ \langle \mathtt{x} \mapsto j,\ e\rangle_{\mathsf{env}}\ c\rangle_{\mathsf{cfg}}$$
$$\Rightarrow \langle\langle \mathtt{s}\rangle_{\mathsf{k}}\ \langle \mathtt{x} \mapsto i,\ e\rangle_{\mathsf{env}}\ c\rangle_{\mathsf{cfg}}$$

- Capture Hoare Triples: $\{\psi\}\,\mathsf{code}\,\{\psi'\}$

$$\exists X_{\mathsf{code}}(\langle \mathsf{code}, \sigma_{X_{\mathsf{code}}}\rangle \wedge \psi_X)$$
$$\Rightarrow \exists X_{\mathsf{code}}(\langle \mathtt{skip}, \sigma_{X_{\mathsf{code}}}\rangle \wedge \psi'_X)$$

Reachability Logic

- New: definable in matching logic
  - All proof rules below can be proved as theorems
- Language-independent proof system for deriving sequents of the form

$$\mathscr{A} \vdash_{\mathscr{C}} \varphi \Rightarrow \varphi'$$

where $\mathscr{A}$ (axioms) and $\mathscr{C}$ (circularities) are sets of reachability rules

- Intuitively: symbolic execution with operational semantics + reasoning with cyclic behaviors

Proof System for Reachability

Proves any reachability property of any lang., including anything that Hoare logic can (proofs of comparable size)

[FM'12]

Sound (partially correct) and relatively complete

[ICALP'12], [OOPSLA'12], [LICS'13], [RTA'14]

$$\textsc{Axiom}: \frac{\varphi \Rightarrow \varphi' \in \mathcal{A} \qquad \psi \text{ is a FOL formula (the } \textit{logical frame}\text{)}}{\mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow \varphi' \wedge \psi}$$

$$\textsc{Reflexivity}: \mathcal{A} \vdash_\emptyset \varphi \Rightarrow \varphi$$

$$\textsc{Transitivity}: \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash_\emptyset \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

$$\textsc{Consequence}: \frac{\models \varphi_1 \rightarrow \varphi_1' \qquad \mathcal{A} \vdash_C \varphi_1' \Rightarrow \varphi_2' \qquad \models \varphi_2' \rightarrow \varphi_2}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2}$$

$$\textsc{Case Analysis}: \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi \qquad \mathcal{A} \vdash_C \varphi_2 \Rightarrow \varphi}{\mathcal{A} \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow \varphi}$$

$$\textsc{Abstraction}: \frac{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi' \text{ where } X \cap FV(\varphi') = \emptyset}{\mathcal{A} \vdash_C \exists X\, \varphi \Rightarrow \varphi'}$$

$$\textsc{Circularity}: \frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Traditional Verification vs. Our Approach

Traditional proof systems: language-specific

$$\frac{\mathcal{H} \vdash \{\psi \wedge \mathtt{e} \neq 0\}\, \mathtt{s}\, \{\psi\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{while(e)}\, \mathtt{s}\, \{\psi \wedge \mathtt{e} = 0\}}$$

$$\frac{\mathcal{H} \cup \{\psi\}\, \mathtt{proc()}\, \{\psi'\} \vdash \{\psi\}\, \mathtt{body}\, \{\psi'\}}{\mathcal{H} \vdash \{\psi\}\, \mathtt{proc()}\, \{\psi'\}}$$

Our proof system: language-independent

$$\textbf{Transitivity}: \quad \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash_\emptyset \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

$$\textbf{Circularity}: \quad \frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Traditional Verification vs. Our Approach

Traditional proof systems: language-specific

$$\frac{\begin{array}{l}\{P\}\texttt{e1}\{S * \mathbf{r} \doteq V_1\} \qquad S = R * \gamma(Ls, V_1, V_2) \\ \{S * \mathsf{True}(V_2)\}\texttt{e2}\{P\} \\ Q = S * \mathsf{False}(V_2) * \mathbf{r} \doteq \texttt{undefined} \qquad \mathbf{r} \notin \mathsf{fv}(R)\end{array}}{\{P\}\texttt{while(e1)\{e2\}}\{Q\}}$$

Our proof system: language-independent

$$\textbf{Transitivity}: \quad \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash_\emptyset \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

$$\textbf{Circularity}: \quad \frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Whiteboard example: SUM

```
// SUM
s = 0;
// LOOP
while(--n) {
   s += n;
}
```

$$\textsc{Axiom}: \frac{\varphi \Rightarrow \varphi' \in \mathcal{A} \qquad \psi \text{ is a FOL formula (the } \textit{logical frame}\text{)}}{\mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow \varphi' \wedge \psi}$$

$$\textsc{Reflexivity}: \mathcal{A} \vdash_\emptyset \varphi \Rightarrow \varphi$$

$$\textsc{Transitivity}: \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash_\emptyset \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

$$\textsc{Consequence}: \frac{\models \varphi_1 \rightarrow \varphi_1' \qquad \mathcal{A} \vdash_C \varphi_1' \Rightarrow \varphi_2' \qquad \models \varphi_2' \rightarrow \varphi_2}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2}$$

$$\textsc{Case Analysis}: \frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi \qquad \mathcal{A} \vdash_C \varphi_2 \Rightarrow \varphi}{\mathcal{A} \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow \varphi}$$

$$\textsc{Abstraction}: \frac{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi' \text{ where } X \cap FV(\varphi') = \emptyset}{\mathcal{A} \vdash_C \exists X\, \varphi \Rightarrow \varphi'}$$

$$\textsc{Circularity}: \frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Whiteboard example: SUM

Hoare Logic

1.	$\{\psi_{pre}\}$ SUM′ $\{\psi_{post}\}$	HL-SEQ(2, 6)
2.	$\{\psi_{pre}\}$ s=0;n=n−1; $\{\psi_{inv}\}$	HL-CNSQ(3)
3.	$\{\psi_{pre}\}$ s=0;n=n−1; $\{\psi_1\}$	HL-SEQ(4, 5)
4.	$\{\psi_{pre}\}$ s=0; $\{\psi_{pre} \wedge \mathtt{s} =_{Int} 0\}$	HL-ASGN
5.	$\{\psi_{pre} \wedge \mathtt{s} =_{Int} 0\}$ n=n−1; $\{\psi_1\}$	HL-ASGN
6.	$\{\psi_{inv}\}$ LOOP′ $\{\psi_{post}\}$	HL-WHILE(7), HL-CNSQ
7.	$\{\psi_2\}$ s=s+n;n=n−1; $\{\psi_{inv}\}$	HL-SEQ(8,9)
8.	$\{\psi_2\}$ s=s+n; $\{\psi_3\}$	HL-ASGN
9.	$\{\psi_3\}$ n=n−1; $\{\psi_{inv}\}$	HL-ASGN, HL-CNSQ

Reachability Logic

1.	$\mathcal{S}, \emptyset \vdash_\emptyset \varphi_{pre} \Rightarrow^\exists \varphi_{post}$	TRANS(2, 3)
2.	$\mathcal{S}, \emptyset \vdash_\emptyset \varphi_{pre} \Rightarrow^\exists \exists n'.\varphi_{inv}$	CONSEQUENCE(4)
3.	$\mathcal{S}, \emptyset \vdash_\emptyset \exists n'.\varphi_{inv} \Rightarrow^\exists \varphi_{post}$	ABSTRACTION(5)
4.	$\mathcal{S}, \emptyset \vdash_\emptyset \varphi_{pre} \Rightarrow^\exists \varphi_1$	AXIOM
5.	$\mathcal{S}, \emptyset \vdash_\emptyset \varphi_{inv} \Rightarrow^\exists \varphi_{post}$	CIRCULARITY(6)
6.	$\mathcal{S}, \emptyset \vdash_{\{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\}} \varphi_{inv} \Rightarrow^\exists \varphi_{post}$	TRANS(7, 8)
7.	$\mathcal{S}, \emptyset \vdash_{\{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\}} \varphi_{inv} \Rightarrow^\exists \varphi_2$	AXIOM
8.	$\mathcal{S}, \{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\} \vdash_\emptyset \varphi_2 \Rightarrow^\exists \varphi_{post}$	CASE(9, 10)
9.	$\mathcal{S}, \{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\} \vdash_\emptyset \varphi_2 \wedge n' >_{Int} 1 \Rightarrow^\exists \varphi_{post}$	TRANS(11, 12)
10.	$\mathcal{S}, \{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\} \vdash_\emptyset \varphi_2 \wedge n' \leq_{Int} 1 \Rightarrow^\exists \varphi_{post}$	AXIOM$^+$
11.	$\mathcal{S}, \{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\} \vdash_\emptyset \varphi_2 \wedge n' >_{Int} 1 \Rightarrow^\exists \varphi_5$	AXIOM$^+$
12.	$\mathcal{S}, \{\varphi_{inv} \Rightarrow^\exists \varphi_{post}\} \vdash_\emptyset \varphi_3 \Rightarrow^\exists \varphi_{post}$	AXIOM(μ)

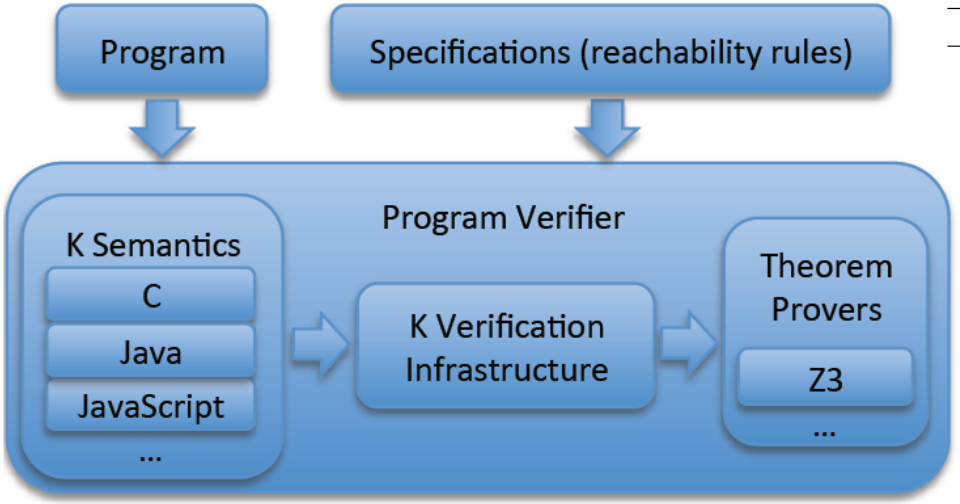
Notations:

$$
\begin{array}{lcl}
\psi_{pre} & \equiv & \mathtt{n} =_{Int} n \wedge n \geq_{Int} 1 \\
\psi_{post} & \equiv & \mathtt{n} =_{Int} 0 \wedge \mathtt{s} =_{Int} n *_{Int} (n -_{Int} 1) /_{Int} 2 \\
\psi_1 & \equiv & \mathtt{n} =_{Int} n -_{Int} 1 \wedge n \geq_{Int} 1 \wedge \mathtt{s} =_{Int} 0 \\
\Sigma_i^j & \equiv & (j +_{Int} i) *_{Int} (j -_{Int} i +_{Int} 1) /_{Int} 2 \\
\psi_{inv} & \equiv & \mathtt{n} \geq_{Int} 0 \wedge s =_{Int} \Sigma_{\mathtt{n}+_{Int}1}^{n-_{Int}1} \\
\mathtt{LOOP'} & \equiv & \mathtt{while(n)\{s = s + n; n = n - 1;\}} \\
\psi_2 & \equiv & \mathtt{n} >_{Int} 0 \wedge \mathtt{s} =_{Int} \Sigma_{\mathtt{n}+_{Int}1}^{n-_{Int}1} \\
\psi_3 & \equiv & \mathtt{n} >_{Int} 0 \wedge \mathtt{s} =_{Int} \Sigma_{\mathtt{n}}^{n-_{Int}1}
\end{array}
$$

$$
\begin{array}{lcl}
\varphi_{pre} & \equiv & \langle\langle \mathtt{SUM} \rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto n,\ \mathtt{s} \mapsto s \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \wedge n \geq_{Int} 1 \\
\varphi_{post} & \equiv & \langle\langle\rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto 0,\ \mathtt{s} \mapsto n *_{Int} (n -_{Int} 1) /_{Int} 2 \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \\
\mathtt{LOOP} & \equiv & \mathtt{while(--n)\{\ s = s + n;\ \}} \\
\varphi_1 & \equiv & \langle\langle \mathtt{LOOP} \rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto n,\ \mathtt{s} \mapsto 0 \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \wedge n \geq_{Int} 1 \\
\varphi_{inv} & \equiv & \langle\langle \mathtt{LOOP} \rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto n',\ \mathtt{s} \mapsto \Sigma_{n'}^{n-_{Int}1} \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \wedge n' \geq_{Int} 1 \\
\mathtt{IF} & \equiv & \mathtt{if(--n)\{s = s + n;\ LOOP\}\ else\ \{\}} \\
\varphi_2 & \equiv & \langle\langle \mathtt{IF} \rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto n',\ \mathtt{s} \mapsto \Sigma_{n'}^{n-_{Int}1} \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \wedge n' \geq_{Int} 1 \\
\varphi_3 & \equiv & \langle\langle \mathtt{LOOP} \rangle_{\mathsf{code}} \langle \mathtt{n} \mapsto n' -_{Int} 1,\ \mathtt{s} \mapsto \Sigma_{n'-_{Int}1}^{n-_{Int}1} \rangle_{\mathsf{state}} \rangle_{\mathsf{cfg}} \wedge n' >_{Int} 1
\end{array}
$$

(Best Effort) Implementation

- Reachability logic implemented in K, generically



	C	Java	JavaScript
Semantics development (months)	40	20	4
Semantics size (#rules)	2,572	1,587	1,378
Semantics size (LOC)	17,791	13,417	6,821

[OOPLSA'16]

- Evaluated it with the existing semantics of C, Java, and JavaScript, and several tricky programs

OK Performance

Time (seconds) spent on applying semantic steps (symbolic execution)

Time (seconds) spent on domain reasoning (matching logic + querying Z3)

Programs	KernelC				C				Java				JavaScript			
	Execution		Reasoning		Execution		Reasoning		Execution		Reasoning		Execution		Reasoning	
	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query	Time	#Step	Time	#Query
BST find	0.6	192	1.2	95	10.4	1,028	3.6	246	1.9	322	2.8	244	4.5	1,736	1.8	93
BST insert	0.8	336	2.9	160	23.0	2,481	7.2	414	4.1	691	4.5	342	5.4	3,394	2.8	158
BST delete	1.4	582	5.6	420	55.1	4,540	16.6	938	9.8	1,274	15.1	1,125	15.6	5,052	5.6	373
AVL find	0.6	192	1.2	95	9.9	1,028	3.1	214	2.2	322	2.7	244	4.5	1,736	1.9	93
AVL insert	6.2	1,980	42.1	1,133	210.7	12,616	70.6	1,865	42.4	3,753	62.8	2,146	102.5	26,977	32.5	1,221
AVL delete	9.5	2,933	45.4	1,758	514.8	26,003	118.9	3,883	122.2	8,144	149.4	4,866	184.3	38,591	55.3	2,233
RBT find	0.6	192	1.1	95	11.5	1,064	3.0	214	2.1	322	2.9	244	4.9	1,736	1.9	93
RBT insert	7.6	2,331	48.1	1,392	722.0	30,924	181.8	4,394	39.9	4,240	75.7	2,547	84.9	28,082	29.6	1,381
RBT delete	10.6	3,891	33.7	2,033	1593.8	50,389	308.3	15,429	95.8	8,312	75.4	4,460	144.2	51,356	39.4	2,009
Treap find	0.6	200	1.4	118	11.2	1,064	3.2	214	2.0	322	2.9	244	4.6	1,736	1.9	116
Treap insert	1.4	753	4.5	247	52.4	4,954	15.3	724	12.7	1,469	10.4	563	13.7	7,738	5.2	243
Treap delete	2.0	831	9.4	509	73.9	5,512	16.5	656	12.0	1,694	16.4	1,021	24.8	8,333	8.4	460
List reverse	0.4	142	0.3	21	6.6	815	4.8	76	1.5	222	2.6	46	5.0	1,162	0.5	20
List append	0.4	171	0.5	45	7.4	909	7.4	128	1.8	239	5.5	106	4.5	1,392	0.8	46
Bubble sort	0.9	391	26.8	190	28.4	2,401	38.0	357	3.4	589	35.4	345	5.6	2,688	25.7	145
Insertion sort	1.1	468	24.5	300	26.6	2,555	35.3	451	4.1	731	27.0	371	8.3	3,119	36.5	213
Quick sort	1.1	604	31.6	269	31.0	3,601	48.2	518	7.1	958	40.0	413	15.0	5,046	33.1	252
Merge sort	1.7	970	55.0	478	81.6	6,589	89.0	1,070	14.1	1,566	72.9	737	22.8	7,021	43.2	480
Total	47.7	17,159	335.2	9,358	3470.5	158,473	970.6	31,791	379.3	35,170	604.5	20,064	654.9	196,895	326.3	9,629

- Properties very challenging to verify automatically. We only found one such prover for C, based on a separation logic extension of VCC
  - Which takes 260 sec to verify AVL insert (ours takes 280 sec; see above)

Conclusion: It can be done.