# Extended Static Checking

Michael Clarkson
CS 711
November 15, 2005

## Authors

| Author | Defn. | Num.* |
|---|---|---|
| K. Rustan M. Leino | MJS | 13 |
| Greg Nelson | MJ | 6 |
| James B. Saxe | MJ | 6 |
| Wolfram Schulte | S | 3 |
| David Detlefs | M | 2 |
| Raymie Stata | J | 2 |
| Mike Barnett | S | 2 |
| Cormac Flanagan | J | 1 |
| Mark Lillibridge | J | 1 |
| Robert DeLine | | 1 |
| Manuel Fähndrich | | 1 |
| Silvija Seres | | 1 |

** w.r.t. bibliography at end of talk*

| | |
|---|---|
| M | ESC/Modula-3 |
| J | ESC/Java |
| S | Spec# |



## Verification of Safety Properties

- Purpose: finding bugs, not full verification
- Nine out of the last twelve seminar papers:
  - ESP, buffer overflows, race detection, ownership types, pointer assertions
- Approach so far:
  - Define a clever abstraction
  - Use (clever) algorithm to verify property in the abstraction



## Extended Static Checking (ESC)

- SRC project ca. 1995-2000
- Abstraction: predicates
  - Encode program and property into (first-order) predicate(s)
  - Truth of predicates implies program satisfies property
- Algorithm: theorem prover
  - Invoke prover on predicates
- Idea has been around since early 1970s



## Extended Static Checking (ESC)

- Cons:
  - Theorem prover is a blunt tool
    - It may need help from the user (interaction, annotations)
    - It may diverge
  - Bug-finder, not full verifier
    - "We aren't proving that the program meets its full functional specification, only that it doesn't crash"
    - "Without discipline, you can quickly slide into the black hole of full correctness verification."
- Pros:
  - General purpose
  - Conceptually elegant



## Overview

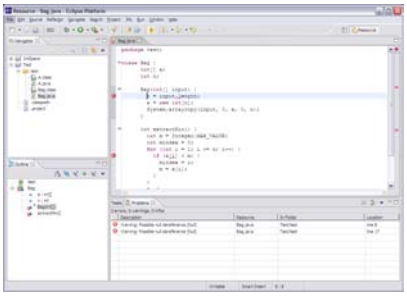
- ESC/Java
  - Demo
- Spec#
- Data abstraction

## ESC/Java Design

- Priority: useful
  - Check (statically) for runtime errors
    - Null dereference, buffer overrun, type cast, division by 0, etc.
  - Check for common synchronization errors
    - Race conditions and deadlocks
  - Check programmer-supplied specifications
    - Preconditions, postconditions, invariants
  - Be modular
  - Be automatic
- Sacrifice soundness and completeness



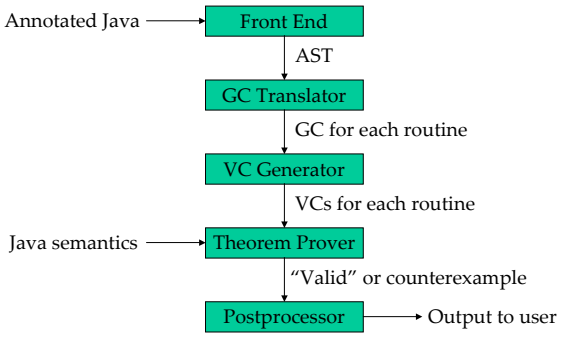
## ESC/Java Demo



*Actually ESC/Java2*



## ESC/Java Architecture






## ESC/Java Assertion Language

- Two primitives:
  - assume P
  - assert P
- Variables:
  - non_null
- Method specifications:
  - requires P
  - ensures P
  - exsures (T t) P
  - pure
  - modifies V



## ESC Assertion Language

- Class declaration:
  - invariant P
- Predicates:
  - Any side-effect free Java expression
  - \result
  - \old(E)
  - \forall T V; E
  - \exists T V; E
- …



## Translating Java to GCs

- Target language is:

S ::= x = E | skip | raise | assert P | assume P
| var x in S end | S ; S | S ! S | S [] S
| loop {inv P} S end | call m (E*)

- Then loop and call are translated away

## Translating Java to GCs

- wlp and VCgen easy to define for remaining GCs
  - wlp.S.R,X,Z
  - Goal is to show that assert never fails
- Full translation takes 40 pages to document
  - Example:

$$[\![\ t = (T)\ s;\ ]\!] = \text{assert } (s = \text{null} \vee \text{typeof}(s) <: T);\ t = s;$$

  - typeof and <: are relations defined by *background predicate*



## Using Theorem Prover

- Effort to use must be low:
  - Fully automatic
  - Counterexample generation
  - Reasonably fast
  - Behaves like a type checker
- Simplify (ESC/Modula-3, ESC/Java, Spec#)
  - Engineered to work well for the kinds of formulas that VCgen produces
  - Performs heuristic search for satisfying assignment to ¬VC
  - Labels predicates with program location to produce human-readable error messages



## Sources of Unsoundness

- Finite unrolling of loops (default is 1.5)
  - Avoids need for programmers to supply invariants
- Object invariants not universally enforced
  - Invariants should hold for *all* allocated objects at *all* routine boundaries
  - But checking would be
    - Too expensive: too many objects to check
    - Too strict: sometimes programmers temporarily violate invariants
  - So instead:
    - At call sites, only check invariants for parameters
    - Use heuristics to reduce set of invariants



## Sources of Unsoundness

- Modifies lists not enforced
  - Aliasing and subclassing make it impossible to write down an accurate modifies list anyway
  - But prover still assumes that modifies list is correct
- Overriding methods can change strengthen precondition
  - Similar to allowing covariant arguments
  - Included so that a class can mention its fields when overriding a specification inherited from an interface
- Multiple inheritance: super types' specifications not all enforced
- Arithmetic overflow, string semantics



## Sources of Unsoundness

- Most Java errors and exceptions ignored
  - NullPointer, IndexOutOfBounds, ClassCast, ArrayStore, Arithmetic, NegativeArraySize are the only one checked
- Constructors that terminate abnormally can leak uninitialized objects
- …



## Sources of Incompleteness

- Simplify
  - Theory of arithmetic is undecidable: May abort attempted proof and report a counterexample to avoid potential infinite loop
  - No semantics for multiplication
  - No support for induction
- Java semantics not fully modeled
  - Floating–points, strings, exceptions, JDK, dynamic typing of arrays, integer overflow, reflection
- …

## Spec#

- MSR project: ESC for C#
- Does not attempt to prove absence of unchecked exceptions
- Major goal: recover soundness
  - "[The verifier] attempts to completely verify a program without missing errors; its ability to do so is bound to depend on the simplicity of the specifications"

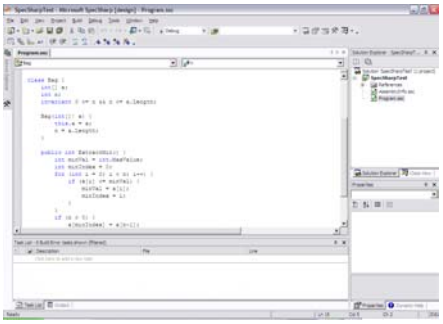


## Spec# Soundness

- Loops
  - Use abstract interpretation to synthesize invariant
- Modifies clauses
  - Checked statically
  - Introduce mechanism to abstract over heap
- Overriding specifications
  - No changes to preconditions allowed
- Multiple inheritance
  - Disallow shared implementation of methods with differing preconditions
- …



## Spec# Demo





## Evaluation (ESC/Java)

- Annotating a program increases LOC by 10%
- Annotation rate is 300-600 LOC/hour
- Time to check a routine correlates with size of routine
  - Reasonable (0-50 LOC): 0-10 sec
  - Large (50-1000 LOC): up to 5 min deadline
  - About 3 hours to check 41KLOC in 2300 routines
- These results are for their own front end
- There seems to be no reported, thorough evaluation?



## Evaluation

- "The start-up cost [for a preexisting code base] is still too high"
- "[We] found about [6] errors…assessed as not having been worth 3 weeks to discover"



## Open Problems in ESC

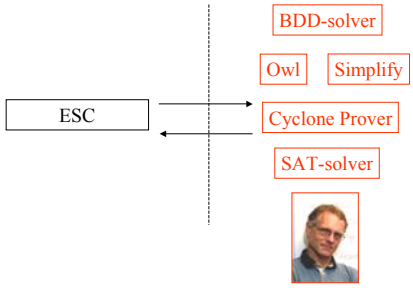
- Reduce annotation burden (Houdini, Daikon)
- Sound checking
- Sound and complete logic for higher-order functions
- Temporary violation of invariants
- Reasoning about machine arithmetic
- Instructional use

## Pseudo-Cornell Work on ESC

- Yanling Wang, ESC for Cyclone:
  - Safety policies supplied by code consumer rather than producer
  - Pluggable theorem provers



## Extensible Architecture






## Conclusion

- Extended static checking
  - Find bugs in programs
  - User-supplied predicates as annotations
  - Theorem prover as backend
- Still searching for sweet spot between soundness, usefulness, completeness
- Wide-spread adoption requires reducing annotation burden and improving safety guarantees



## Bibliography


- Extended static checking for Java.
- The Spec# programming system: An overview.
- ESC/Java user's manual.
- Data abstraction and information hiding.
- Verification of object-oriented programs with invariants.
- Applications of extended static checking.
- Extended static checking: A ten-year perspective.
- Houdini, an annotation assistant for ESC/Java.
- Static verification of dynamically detected program invariants: Integrating Daikon and ESC/Java.
- Checking Java programs via guarded commands.
- ESC/Java quick reference.
- Simplify: A theorem prover for program checking.
- Exception safety for C#