

software analysis: a roadmap

Daniel Jackson & Martin Rinard
MIT Laboratory for Computer Science
ICSE · Limerick, Ireland · June 7, 2000

analysis zoo

Promela	LTL	SPIN
assigns	CTL	SMV
Statechart	ad hoc	Statemate
CSP	CSP	FDR
LTSA	FSP	LTL

VDM	ad hoc	IFAD
Z	Z+	Z/EVES
Alloy	Alloy	Alloy

Java	Java types	javac
C++	FO logic	PSA
C	ad hoc	lint
any	RMT	graphs
C	SQL	CIA

every analysis involves

- an artifact language
model or code?
- a property language
operational or declarative?
- a mechanism
sound, scalable, accurate?

PSA: Parametric Shape Analysis
Sagiv, Reps, Wilhelm
RMT: Reflexion Model Tool
Murphy, Notkin, Sullivan

topics

analyzing models

- why analysis?
- incrementality
- do errors matter?

analyzing code

- compositionality
- unsoundness
- evaluation

analyzing code vs. models

- rhinos and tick birds

some myths

- about analysis generally

6/2/00

© Daniel Jackson, 2000

3

disclaimers

this roadmap

- will leave you at forks without directions
- doesn't stick to main roads
- doesn't take you to your destination

6/2/00

© Daniel Jackson, 2000

4

why analysis?

clashing cultures?

- Z: language matters
- SMV: analysis matters

two kinds of result

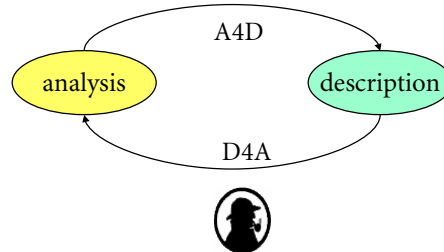
- checking an adder
want errors
- modelling a railway
want a model

morals

- in software, both matter
- design language *with* analysis
- no formal basis, no analysis



*This is not a good game
Said our fish as he lit
"No, I do not like it,
Not one little bit!"*



the apple doesn't fall far from the tree



*Oxford, home of Z
elegant expression*



*Pittsburgh, home of SMV
mechanization*

Oxford photo from
Jonathan Bowen's
collection: Pittsburgh
photo by Norm
Schumm, with
permission

incrementality

want

- gentle slope: say a bit, analyze a bit
- few environmental assumptions
- maximal implementation freedom

language effects

- conjunction: refine by extension
- if property language is artifact language
 - can use property as model
 - can continue analysis without fixing bug

morals

- need interactive tools
- language can help

model expressed as
abstract program
easy for programmers
no frame problem

model expressed as
logical formula
can start with less
inconsistency
underconstraint

do errors matter?

standard justification

- catching errors early saves costly fix later

in software

- few showstopper bugs
- subtle bugs are cheap to fix
- real cost is flawed design

morals

- analysis can help simplify design
- “a difficulty deferred is a difficulty doubled”*
- find a new way to sell software model checking ...



*Michael Jackson

compositionality

analysis in the face of

- missing source code
- libraries, frameworks
- multiple languages

compositional analyses

- process unit once
- use *spec* for clients

morals

- can't escape specs
- user's help may be inevitable

where do specs come from?

- the tool (eg, Rinard's PEG)
 - handle all contexts
 - but need just a few
 - huge or imprecise spec
- the user (eg, ESC)
 - laborious
- a wizard (eg, Houdini)
 - expensive but effective?

unsoundness

conservative analysis

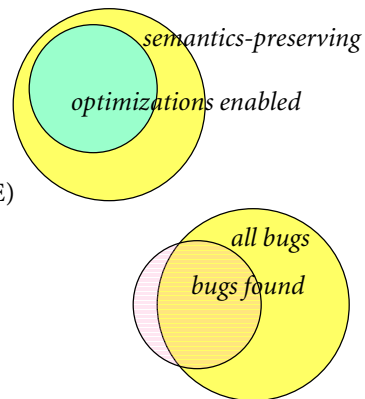
- sound: results can be trusted

why be unsound?

- often good enough (PreFIX, Murphy's LSME)
- easier to implement (Womble)
- soundness compromises accuracy (ESC)

morals

- soundness & accuracy conflict
- engineer can do for a dime ...
- depends on task at hand
- no guarantees, ever?



evaluation

is it good enough?

- how detailed alias analysis?
- flow sensitive? context sensitive?
- higher-order functions?

morals

- need both approaches
- can't predict anyway

the short view

- only end-to-end results count
- show value for a given task

the long view

- the internet took 30 years
maybe slicing will too?
- surprises (eg, ML types for C)
- judge ideas, not applications

sybiosis: models & code

check conformance of code to models

- Bandera, RMT, Pathfinder, etc

how analysis helps models

- if not conforming, what predictive use?

how models help analysis

- focus on properties of interest
- no fumbling in the dark (eg, k-limiting)
- models as induction hypotheses

morals

- look for good matches
object models & shape analysis?

some myths

complex implementation ok

- easy to implement, more implementors (eg, JVM)
- tool writers pick easy languages (eg, Java vs. C++)

can run for a week

- occasionally true
- but interactive tools far better

can evaluate analyses by reading papers

- not your examples, numbers suspect
- you have more/less expertise
- publication culture rewards hiding failures

must be linear time, etc

- better: how will Moore help?
- 1980–2000: 3 hours to 1 second