AI$_4$FM

Another way to use AI ideas to support formal methods a new project: AI4FM

Cliff Jones
Newcastle University

Abstract

AI$_4$FM

We hope to improve the productivity of industrial engineers who are using formal methods. Some prejudices (based of decades of experience) will be presented but the main point is about using Artificial Intelligence (AI) ideas. Theorem proving was seen as an early challenge by AI researchers but any set of heuristics will have a limit. In a recently funded project (AI4FM) we are aiming to learn from the efforts of human theorem proving attempts to improve tool support for classes of proof obligations.

What are "formal methods"? AI_4FM

- all software is in a formal language
  - but users live in an informal world!
  - so … using formalism earlier
- FMs won't solve all problems
  - but stupid not to use
    - cf. effects of sanitation on health!
    - … checklists (MAJ)
  - I argue for RoI (impact) in design
- recent survey: Woodcock et al. ACM CSurv


formal languages
code
formailty
natural languages
tractablity
ability to reason about

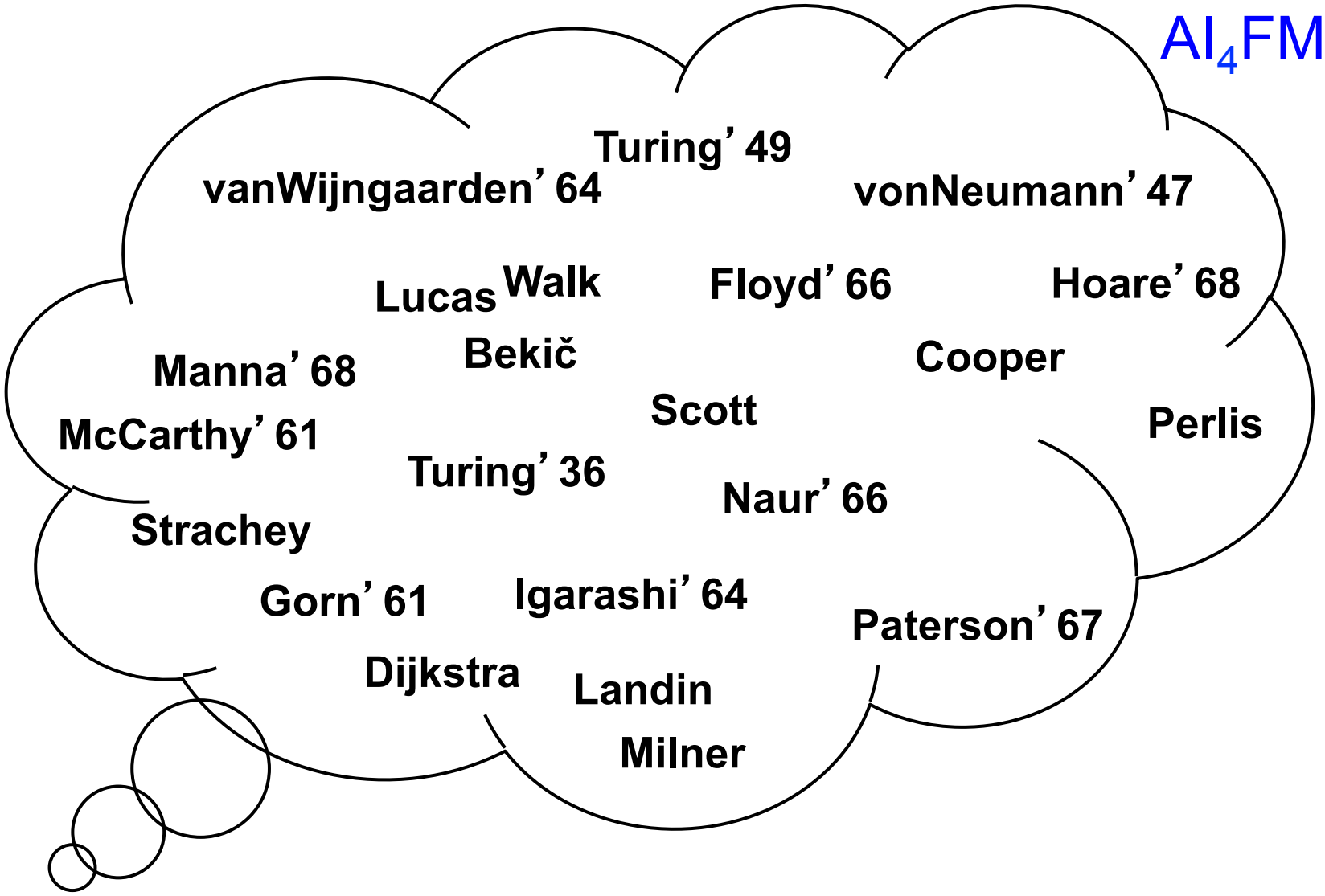
There are many FMs

AI_4FM

- VDM
  - created in IBM Lab Vienna
- Z
- B
- Event-B
- ASMs
- …

A lightning history

Early Search for Tractable Ways of Reasoning about Programs


AI4FM
Turing’ 49
vanWijngaarden’ 64
vonNeumann’ 47
Lucas
Walk
Floyd’ 66
Hoare’ 68
Bekič
Manna’ 68
Cooper
Scott
McCarthy’ 61
Perlis
Turing’ 36
Naur’ 66
Strachey
Gorn’ 61
Igarashi’ 64
Paterson’ 67
Dijkstra
Landin
Milner
Influence?

Floyd-Hoare Axioms

AI$_4$FM

$$\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}\ \{P \wedge \neg b\}}$$

note: does not handle termination

$$\{\, x = r + q*y \,\}$$

while $y \leq r$ **do**
$r \leftarrow r - y\ ;\ q \leftarrow q + 1$
od

$$\{\, x = r + q*y \ \wedge\ \neg(y \leq r) \,\}$$

[Floyd67 (1966)]

AI$_4$FM

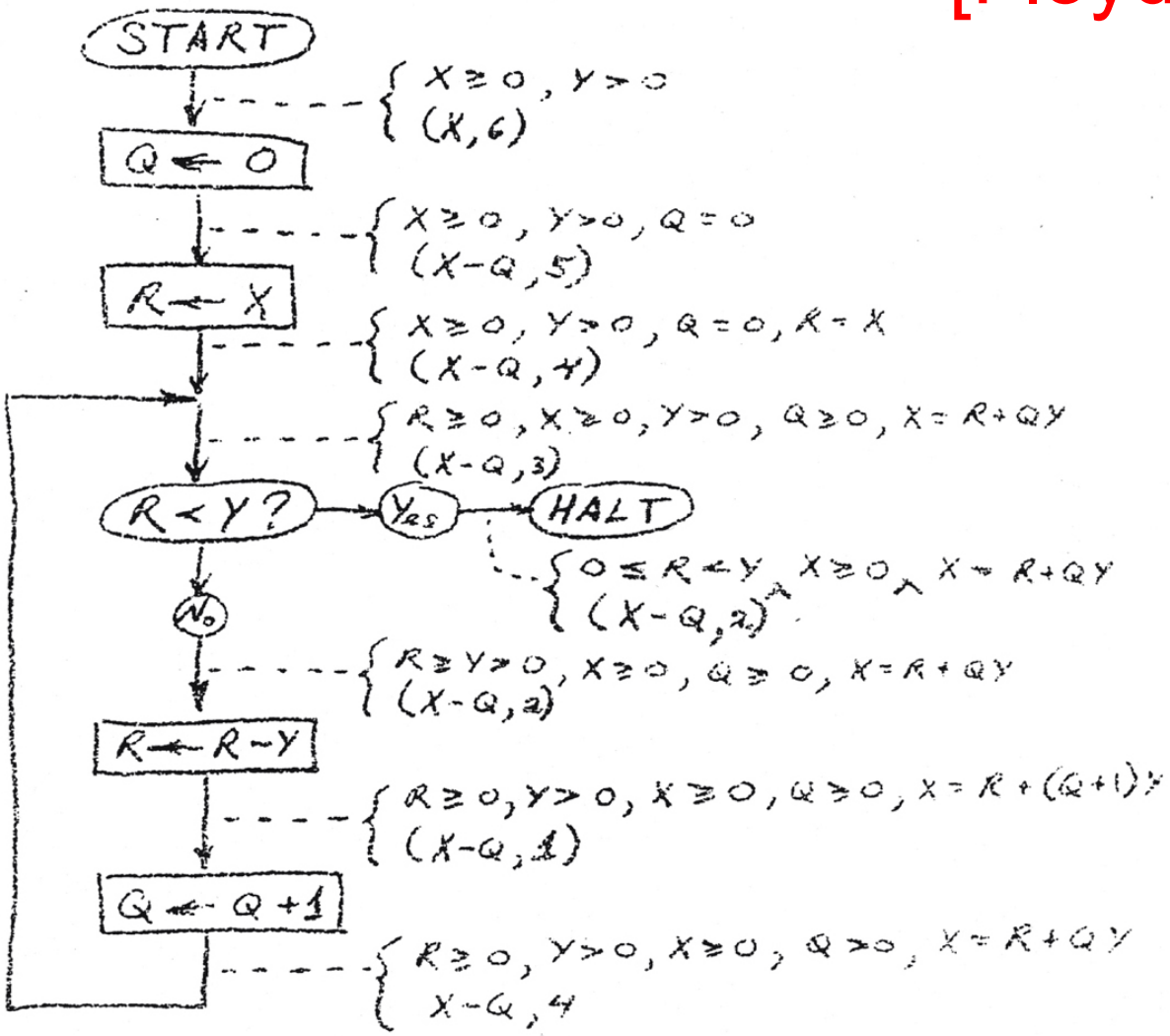


Figure 5.

Algorithm to compute quotient Q and remainder R of X ÷ Y, for integers X≥0, Y>0.

1

Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
------
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
------
 3974
 2213
------
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3794
 2213
```

e.g. it might be

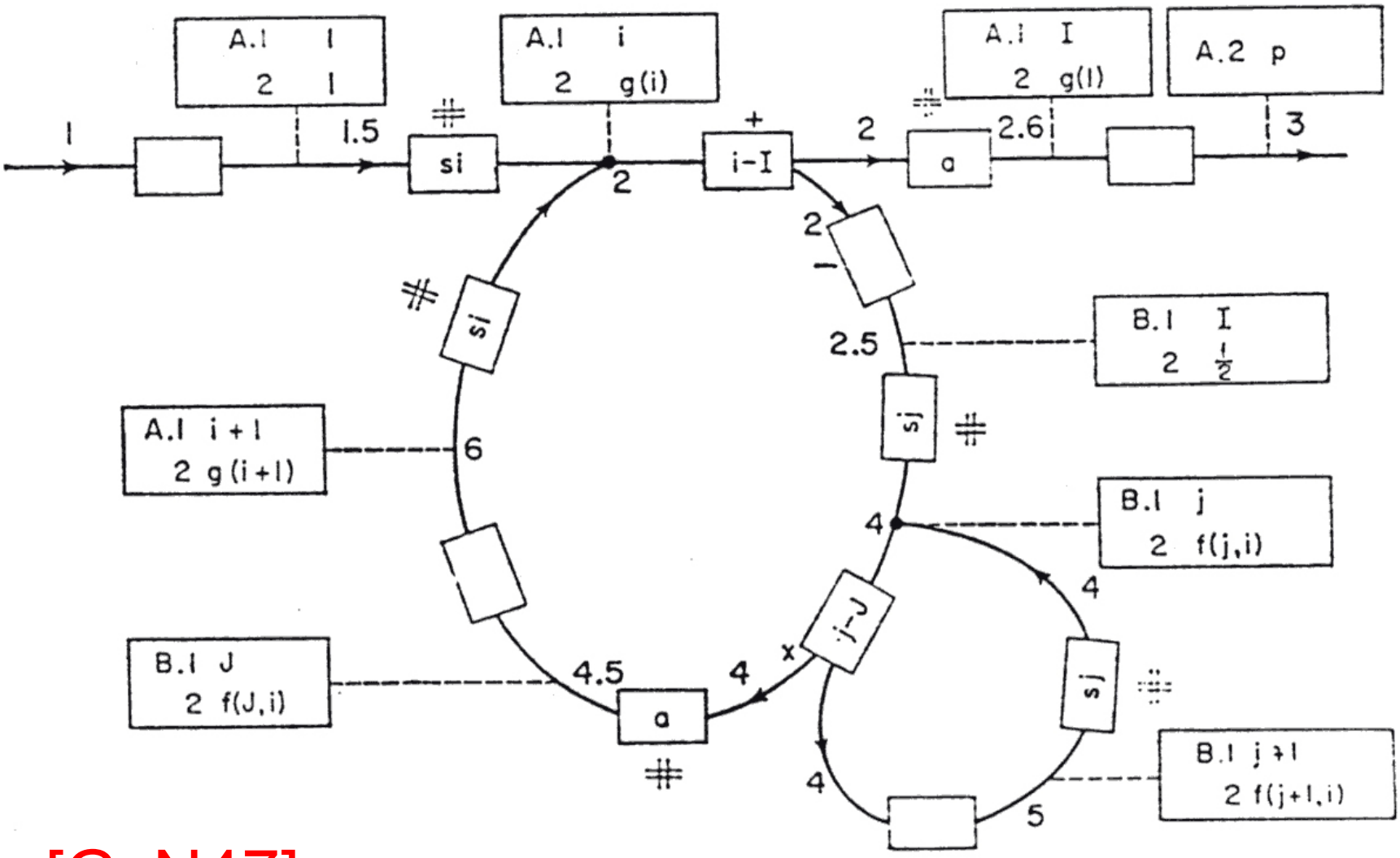
assertions
sertions may
tates when
the ringed
for each such
specify the
nt to give
s or the store
on the
wer part

he initial
e made for
art with
ntity in
s than 240,

half of
any order
rgue.
s. From
i.e.
tries are

d.
definite
ch is
To the
a problem
m of the

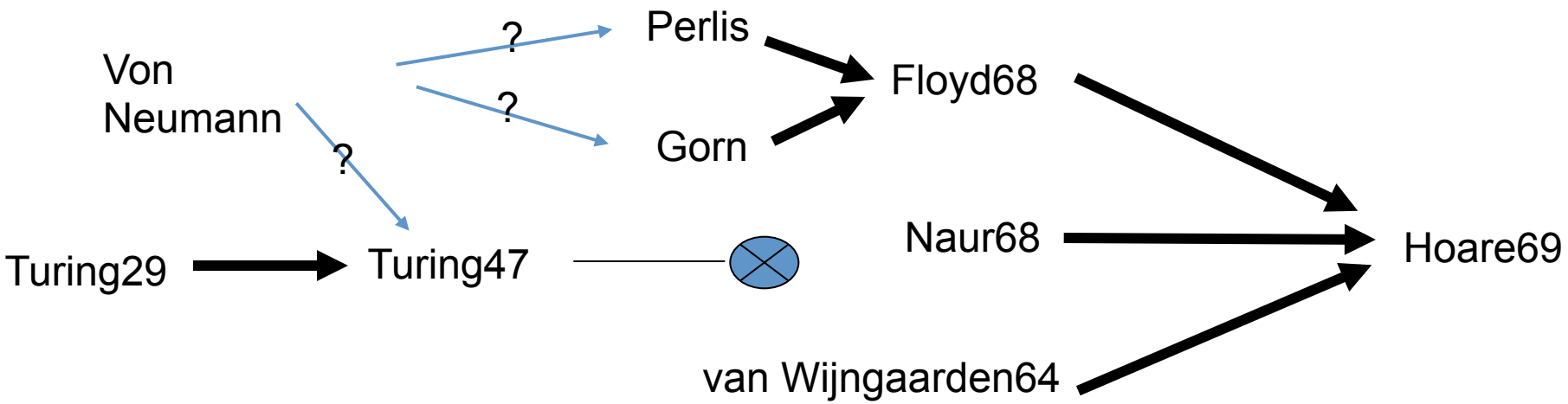
[Turing49]

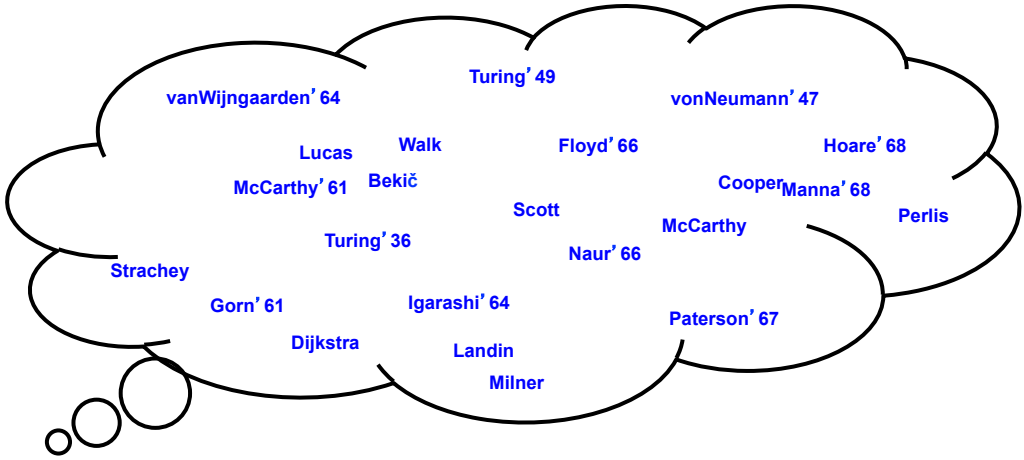

A.1 1
2 1
A.1 i
2 g(i)
A.1 I
2 g(I)
A.2 p
1
1.5
si
2
i-I
2
a
2.6
3
2
1
2.5
B.1 I
2 1/2
si
A.1 i+1
2 g(i+1)
6
si
B.1 j
2 f(j,i)
4
4
j-J
x
4
B.1 J
2 f(J,i)
4.5
a
sj
B.1 j+1
2 f(j+1,i)
4
5

[GvN47]

FIG. 7.9

AI$_4$FM


Von
Neumann
?
?
?
Perlis
Gorn
Floyd68
Turing29
Turing47
Naur68
Hoare69
van Wijngaarden64


Turing' 49
vanWijngaarden' 64
vonNeumann' 47
Lucas
Walk
Floyd' 66
Hoare' 68
McCarthy' 61
Bekič
Cooper
Manna' 68
Scott
Perlis
McCarthy
Turing' 36
Naur' 66
Strachey
Gorn' 61
Igarashi' 64
Paterson' 67
Dijkstra
Landin
Milner

A few issues

AI$_4$FM

- message of my Annals paper
  - importance of formality was obvious
  - search has been for "tractability"
- termination (partial/total correctness)
- predicates of two states
- "clean termination" – R. Sites
- controversies
  - [de Milo et al. 79] *Social Processes ...*
  - [Naur82] *Formalization in Program Development*
  - [Fetzer88] *Program Verification: the very idea!*

(continue)

What are "Formal Methods"?

examples that really ought be thought of as FMs

AI_4FM

- type checking
- …
- SAT
- SMT
- Slayer
- MS driver verification
- hypervisor

my sort of FM

AI$_4$FM

- posit and prove
  - designer uses intuition: posits a design step
    - documents in a formal notation
  - system generates “Proof Obligations” (POG)
  - proof discharged
    - automatically ☺
    - interactively ☹
- examples
  - VDM, Event-B, …

Key role of abstractions

AI$_4$FM

- pre/post specification ("what not how")
  - operation decomposition (sequential)
- delicate/dynamic ownership constraints
  - separation logic
- interference specification
  - rely/guarantee reasoning
- data abstraction
  - data reification
- fiction of atomicity
  - splitting atoms safely

The problem (that we hope to ameliorate)

AI$_4$FM

- anecdotes!
- not all POs are discharged automatically
  - one DEPLOY partner at first review
    - 300 POs/200 automatic 5 ideas for the 100 but ...
  - Steve Wright's figures
    - 5000 POs; varying stages 30%-100%
- often, when they fail, engineers find hard
  - but POs/application follow a very similar pattern

(view from)

A project with heavy involvement from industry

DEPLOY

Background to Deploy project AI$_4$FM

- RODIN project
  - 2004-2007
  - "STREP" = research led
  - industrial partners: Nokia, ClearSy, VTEC, Praxis
  - created a tool set for formal methods
  - some deployment

Deploy project itself

AI$_4$FM

- “IP” = industrial emphasis
  - 2008-2012
- industrial partners:
  - Bosch
  - Siemens Transport
  - SAP
  - Space Systems Finland (SME)
- + “Associate Partners”

A personal view of formal methods (from a long standing proponent) AI_4FM

- "formal methods" (FMs)
  - are not a panacea
    - many problems not even addressed by FMs
    - not intended to replace engineering insight
  - *do* help remove a class of errors
    - one comparison could be role of sanitation in health!
    - I would/will argue biggest RoI for FMs is in design
  - *must* fit into established engineering tool support

Adoption of FMs

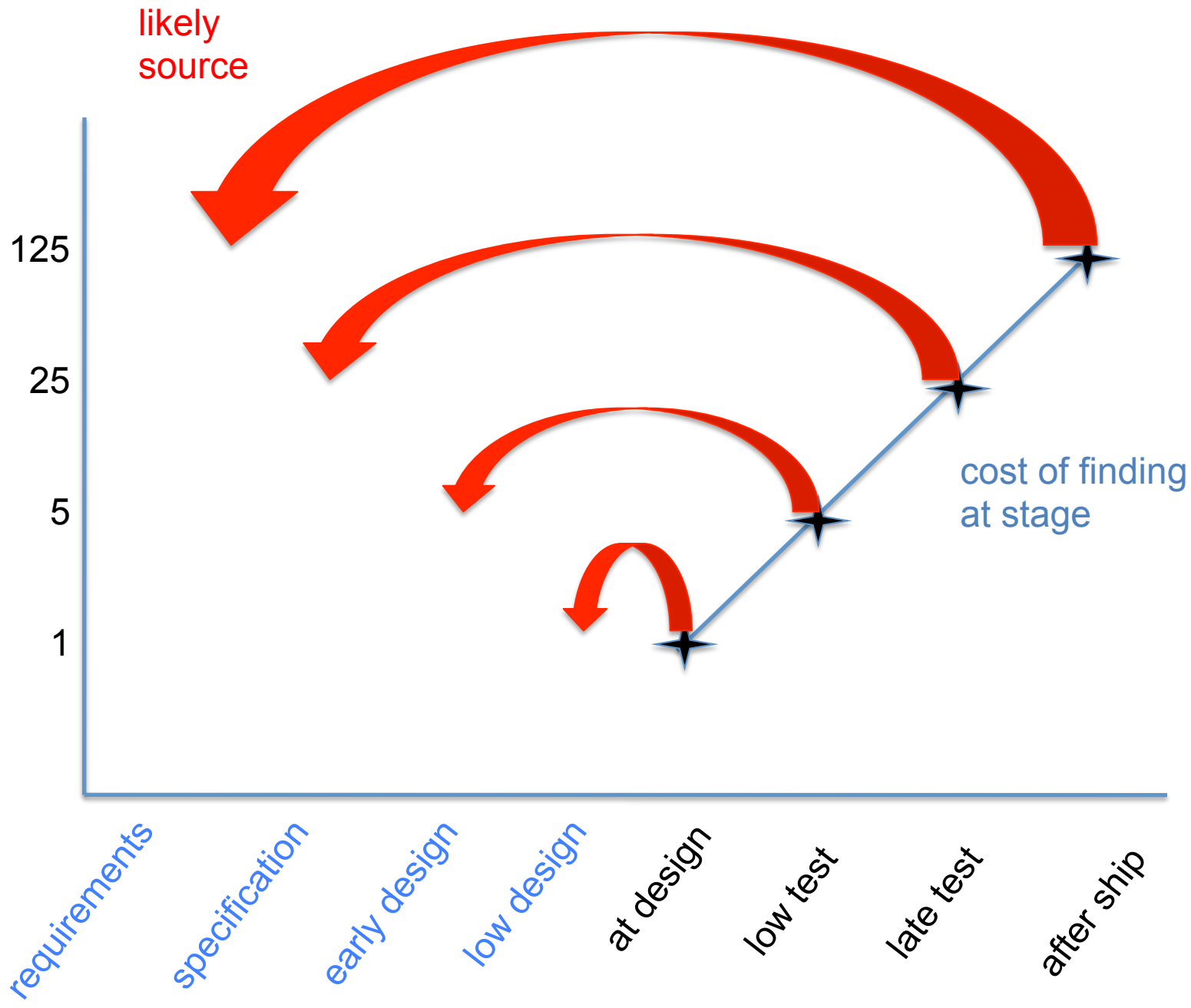
AI$_4$FM

- many things matter
  - management support
  - enthusiasts
  - well chosen pilot project
  - technical support
  - …
- tool support
  - ease of use
  - link to other engineering systems

Experience/view

AI$_4$FM

- what drove me to FMs
  - testing a (poor) IBM compiler
- if there are (deep) bugs in design
  - might find in system test
  - could find by "model checking"
  - would find by proof
    - but expensive *at this stage*
- "scrap and rework" is what costs


likely
source
125
25
5
1
cost of finding
at stage
requirements
specification
early design
low design
at design
low test
late test
after ship

Thus: best place to use formalism

AI$_4$FM

- as early as *possible* in design/development
  - but clearly can't formalise all requirements
- reason for Siemens involvement in Deploy
  - B used in "Météor Line" on-board
  - plan to move to "systems level"
- I argue for use of FM in high-level design
- *not* wait for code as first formal language

Looking at proofs

AI$_4$FM

- examples from VDM
  - satisfiability of Operations
  - adequacy of data representations
  - pre conditions of reified operations
  - post conditions commute …
- note, in some sense, already part of a strategy
  - cf. Nipkow rule
  - vs homomorphic rule

Data reification (i)

AI$_4$FM

- suppose a specification given in terms of sets
  - collection of operations defined on state
- choose a representation on lists
  - write a "retrieve" function
- first: prove "adequacy"
  - for every element of abstraction, …
  - … there exists a representation
- (simple) induction over sets

Data reification (ii) more realistic

AI$_4$FM

- abstraction (for equivalence relations)
  - sets of sets
- representation
  - Fischer/Galler trees

- …, bigger
  - abstraction: state of Java store
  - representation: run time state

How do we *construct* proofs? AI$_4$FM

- backwards from goal
  - hide tracks: present forward from knowns!
- actually, often drop ideas into the middle
- key: system should not constrain thinking
  - FLM anecdote
- example
  - reification

(starting with a good GUI)

governs the way one thinks about proof construction

mural

Other ingredients (cf. *mural*)

AI$_4$FM

- logic frame: not first, certainly not the last
- theories of data types
- dependant types
  - predicate restriction (including recursive)
- good taste in (abstract) lemmas
- sensible handling of partial functions
- never (well almost never) delete anything

Tactics (vs strategy language)

AI$_4$FM

- goal: reduce the search space
- HOL-like tactics are
  - low level language
  - strictly procedural
  - brittle
- but I don't just want a HLL
  - we want to get away from sequential view

AI4FM

setting forth!

(remember, this is a brand new project)

Who's going to fix it? (aka the team)

AI_4FM

- CBJ
  - plus Leo Freitas + Andrius Velykis
- Alan Bundy (Edinburgh)
  - plus Gudmund Grov + PhD
- Andrew Ireland (H-W)
- Michael Butler
- 4 years
- *this was actually intended to be Phase II*

AI and TP

AI$_4$FM

- one of first problems tackled
  - Herb Simon *"Logic Theory Machine"*
  - John McCarthy *"Advice Taker"*
- heuristics
  - enormously successful
  - but, always a horizon
- Bundy on "mining proofs"

A dichotomy

AI$_4$FM

- tackle by model revision (was to be Phase I)
  - how will model revision work with evolution?
  - doubt it works with large models
    - debate about avoiding "large models"
- vs
- tackle via proofs (was to be Phase II)
- so – not "vs" – we need both approaches
  - we will start with proofs

Sooo

AI$_4$FM

- we will focus on POs from FMs
  - try to be generic across FMs
- AI aim: learning from human TP
  - not: better (and better) heuristics
    - they always have a limit
- not even learn from finished proofs
  - learn from proof attempts
    - maybe even failures – "proof critics"

Approach to strategy language AI_4FM

- "good things to try"
  - with POG/theorem shape
  - e.g. generalise; generalise IH
- how these things work
  - with each data type
  - ways to do induction (of course)
  - ways to be more general in induction

Finished proofs vs proof process

AI$_4$FM

- not just backwards (nor just forwards)
  - cf. GUI that lets user jump about
- others have analysed complete proofs
- we want to instrument tools to see how the steps evolve
- what can we learn from dead ends?

Ideas from AI

AI$_4$FM

- proof planning
- "proof critics"
- "patching proofs"
- rippling
- so the "bottom line" is
  - to learn from *proof attempts*
  - then be able to tackle *similar* proofs
  - initial objective POs from *software development*

Whacky ideas (i)

AI$_4$FM

- strategy language should not be sequential
  - key in proof explanation is splitting
  - programming by "desiderata" (Prolog-like)
- proofs are graphs (not lists)
  - even finished ones
- incomplete proofs should be big search graphs
- use parallelism at this level?

Whacky ideas (ii)

AI$_4$FM

- J. Moore's "toy problems"
  - can we generate automatically?
- < for "complete induction"
- cutting search space – categorize lemmas
  - those within a type (use when …)
  - those that bring in new operators (…)

Whacky ideas (iii)

AI$_4$FM

- split info
  - store properties of data structures with them
    - not amidst other tactics
  - store tips for task shapes ?where?
  - store ideas related to POs with POG
    - e.g. examples of ways to do “adequacy”

Theorem Prover's House re-revisited

AI_4FM

- Alan Wills – Appendix E of BoM
- (expert) theorem prover
  - figures out strategy
  - returns proof +
  - … strategy
  - at a price

Meanwhile …

AI$_4$FM

- we have to analyse a lot of proofs
- … and failed attempts
- data from RodinTools
- … and at least one other system
- choice
  - gamble on strategy language adoption
  - which one system we can bend