On Typechecking B

Antti-Juhani Kaijanaho
University of Jyväskylä
Finland

SPLST 2003, Kuopio, Finland
Motivation

• Software should be reliable, yes?

• Software should be correct, yes?

• Formal methods promise to deliver the above. . .

• . . . so how come I can find an inconsistency in one of the most rigorous of methods?

• In this presentation I will show you an inconsistency in the foundations of the B method. . .

• . . . and I will propose a fix.
Question

• How many of you know B?

• How many of you know the details of B typing?
The B Method

- Formal method for developing safety-critical software
- Focus on refinement and proof
- Developed along with industrial tools (B-Toolkit, Atelier-B)
- Paris metro line 14 (Météor) traffic control (e.g. Behm et al, 1999)
  - An industrial success case
  - 80 KLOC of B text, essentially bugfree
B’s layered architecture

- Abstract Machine Notation (AMN)
  - Constructs for specification, refinement-diffs and implementation
  - Familiar Algol-like notation

- Generalized Substitution Language (GSL)
  - A variant of Dijkstra’s weakest precondition calculus
  - Provides formal semantics for AMN

- A typed set calculus
  - Formalized in standard untyped first-order logic
Types in B (1)

- All syntactically well-formed B text must be typechecked

- Types are introduced by the set calculus
  - AMN is translated into GSL and GSL is (partially) translated into set calculus
  - An explicit type must be given by the quantified predicate:
    \[ \forall x \cdot (x \in \mathbb{N} \Rightarrow x \in \mathbb{Z}) \]

- Typechecker (as specified in Abrial’s book) consists of 29 inference rules which a Prolog-like engine interprets
  - Most rules decompose formulae, e.g.
    \[ \frac{E \vdash \text{check}(p) \quad E \vdash \text{check}(q)}{E \vdash \text{check}(p \land q)} \]
Types in B (2)

• Quantifiers introduce types in the environment*:

\[
\begin{align*}
& \quad \textit{i} \notin s \quad \text{\textit{i} \notin q \text{ for each } q \text{ in } E} \quad E, i \in s \vdash \text{check}(p) \\
& \quad E \vdash \text{check}(\forall i \cdot (i \in s \Rightarrow p))
\end{align*}
\]

• Quantification of a variable pair (and thus, recursively, of a variable tuple) is also allowed:

\[
\begin{align*}
& \quad E \vdash \text{check}(\forall x \cdot (x \in s \Rightarrow \forall y \cdot (y \in t \Rightarrow p))) \\
& \quad E \vdash \text{check}(\forall(x, y) \cdot (x, y \in s \times t \Rightarrow p))
\end{align*}
\]

• That was the only rule which decomposes tuple quantification.

*\textit{x} \notin p \text{ means "}\textit{x} \text{ does not occur free in } p\text{"}.
A problem in B typing

• As a reminder:
  \[
  E \vdash \text{check}(\forall x \cdot (x \in s \Rightarrow \forall y \cdot (y \in t \Rightarrow p)))
  \]
  \[
  E \vdash \text{check}(\forall(x, y) \cdot (x, y \in s \times t \Rightarrow p))
  \]

• So, what about this (false) predicate?
  \[
  \forall x, y \cdot (x \in \mathbb{N} \land y \in \mathbb{N} \Rightarrow x > y)
  \]
  – It ought to pass but does not!
  
  – Such predicates and other similar constructs are frequent in the literature
  
  – Such constructs are used even in normative specification of B!

• The following AMN construct is another example:

  \[
  \begin{align*}
  \text{CONSTANTS} & \quad \text{minint, maxint} \\
  \text{PROPERTIES} & \quad \text{minint} \in \mathbb{Z} \land \text{maxint} \in \mathbb{Z}
  \end{align*}
  \]
Idea for a fix

- Enlarge the set of formulae the typechecker accepts.

- Not too much! Types rule out the famous set antinomies.

- Add typing rules that copy potential type-giving predicates to a position where the original system will see them.

- Copying instead of moving induces a defensive failure mode (if the new system does something terribly wrong, it will red-flag the input).
Proposed fix

\[ p \not\sim x \in f \quad \vdash \text{typeQ}(x, p, e) \quad E \vdash \text{check}(\forall x \cdot (x \in e \Rightarrow (p \Rightarrow q))) \]

\[ E \vdash \text{check}(\forall x \cdot (p \Rightarrow q)) \]

\[ \vdash \text{typeQ}(i, p, e) \]

\[ \vdash \text{typeQ}(i, p \land q, e) \]

\[ \vdash \text{typeQ}(i, q, e) \]

\[ \vdash \text{typeQ}(i, p \land q, e) \]

\[ \vdash \text{typeQ}(i, p, e) \]

\[ \vdash \text{typeQ}(i, p \Rightarrow q, e) \]

\[ \vdash \text{typeQ}(i, i \in e, e) \]

\[ \vdash \text{typeQ}(x, p, e) \quad \vdash \text{typeQ}(y, p, f) \]

\[ \vdash \text{typeQ}((x, y), p, e \times f) \]

("\not\sim" means "not of the form" or, equivalently, "does not unify with")

We call this extended typechecking or extended typechecker.
Analysis

• If original terminates, the extended type-checking terminates.

• The extended typechecker passes anything the original passes.

• The thus extended set calculus is as expressive as the original
  – This is Theorem 1 of the paper.
  – Corollary: the extended set calculus is consistent if the original is.
  – Proof is by structural induction and rather messy though not very deep.
Conclusion

• The B method as described by Abrial in B-Book (1996) is inconsistent.

• The inconsistency is remediable, and I proposed a fix.

• The fix is valid in the sense that it does not break anything.

• Curious question: how far apart Abrial’s B-Book and the tools actually are?