IEEE SecDev | 2021

IEEE COMPUTER SOCIETY

The Correctness-by-Construction Approach to Programming Using CorC

Tobias Runge, Loek Cleophas,
Ina Schaefer, and Bruce W. Watson

#IEEESecDev https://secdev.ieee.org/2021

Technische Universität Braunschweig

UNIVERSITEIT STELLENBOSCH UNIVERSITY

TU/e Technische Universiteit Eindhoven University of Technology

ISF

The Correctness-by-Construction Approach to Programming Using CorC

Tobias Runge, Loek Cleophas, Ina Schaefer, and Bruce W. Watson

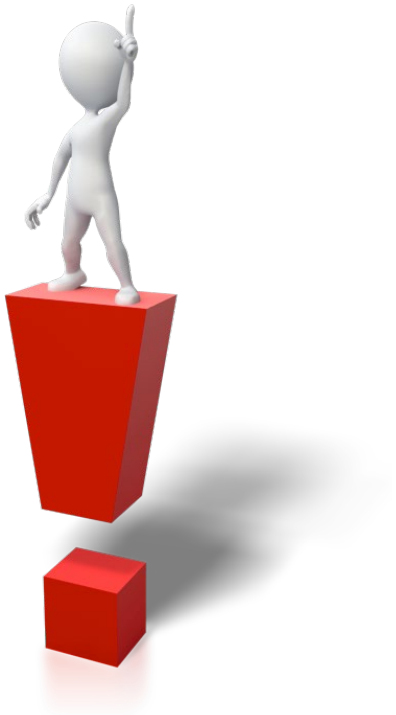
CbC Tutorial, SecDev 2021, 18 October 2021

Motivation

- Small fraction of SW is correctness-critical, but then it really matters.
- For instance, standards for automotive, aviation, medical, …
- Usually correctness is ensured by post-hoc testing (and sometimes post-hoc verification), but that is usually expensive.

Alternative: Correctness-by-Construction

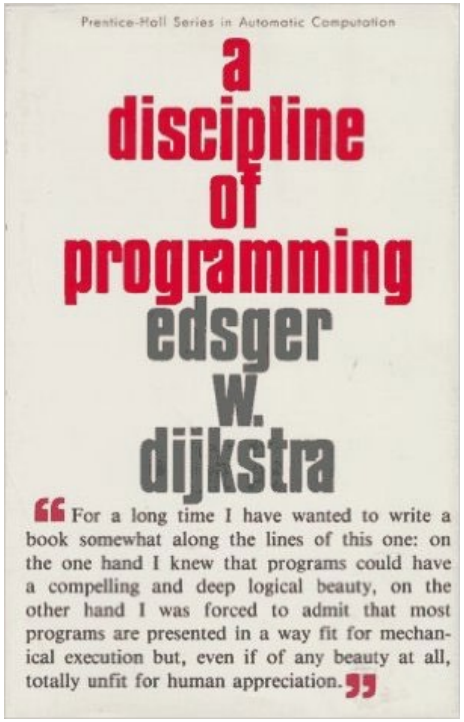
- *Lightweight* correctness-by-construction, historically, the “other” camp
- Start with pre- and postcondition
- Co-develop program and annotations
- Use these annotations to kick-start
  post-hoc verification, or testing, or…



Agenda

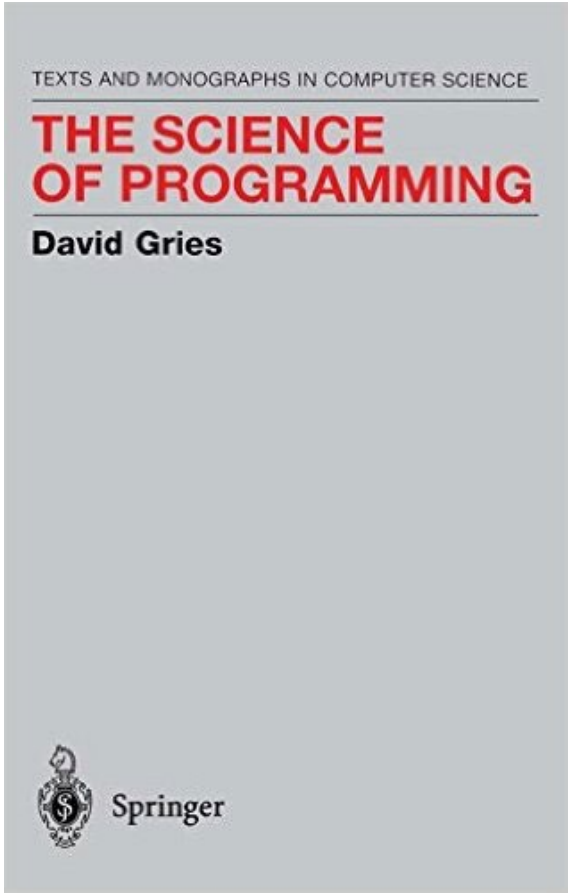
1. Foundations of CbC
2. Simple CbC Examples
3. Hands-On with WebCorC
4. Advanced CbC Examples
5. Information-Flow-by-Construction
6. Further Aspects of CbC

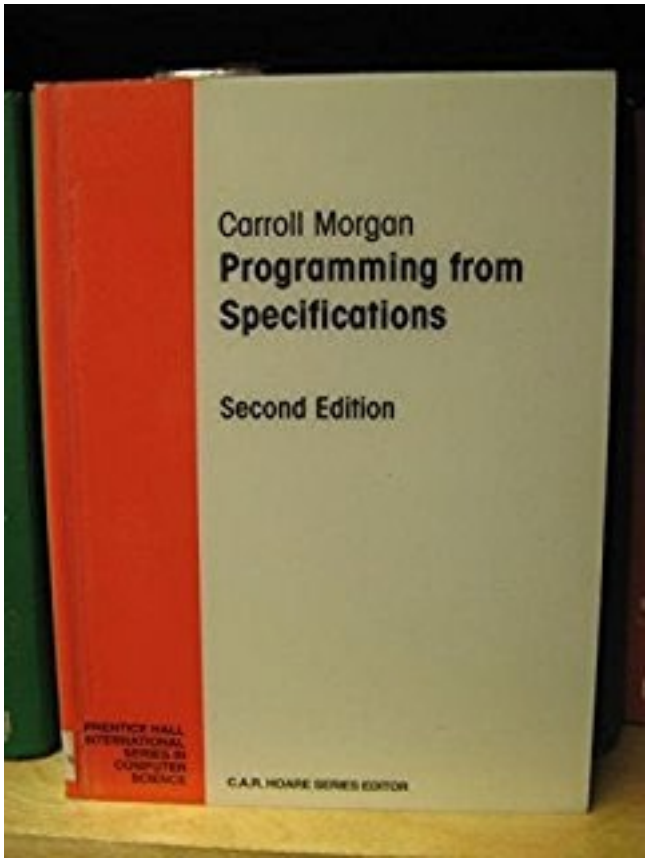
Correctness-by-Construction (CbC)

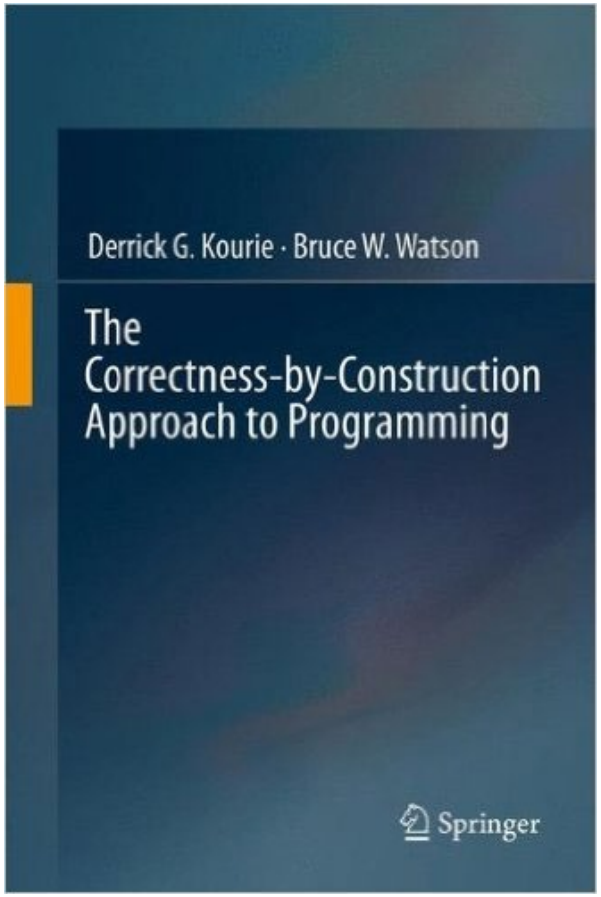


Worthless to the Working Programmer - Great for Computer Scientists
It's like someone writing a book entitled "A Discipline of Calculus" and then claiming that every engineer should use it to "properly" develop their projects, allowing the formalism to do their thinking for them.
James R. Pannozzion November 12, 2011

CbC Round 2+


TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE
THE SCIENCE OF PROGRAMMING
David Gries
Springer


Carroll Morgan
Programming from Specifications
Second Edition
C.A.R. HOARE SERIES EDITOR


Derrick G. Kourie · Bruce W. Watson
The Correctness-by-Construction Approach to Programming
Springer

What is CbC?

CbC ==

Construct a program/algorithm
from a specification
using refinement/correctness-preserving transformations

We consider

- Imperative programs (Guarded Command Language)
- First-order Predicate Logic over Program Variables

Guarded Command Language (GCL)

GCL has only six commands

Command	Symbol
1. Empty command	**skip**
2. Chaotic command	**abort**
3. Assignment	**:=**
4. Composition/concatenation	**;**
5. Selection	**if**
6. Repetition	**do**
7. Comments	{ This is a comment }

Hoare triples as program specification

Definition

{P} y : S {Q} is a predicate, where:

- S is a program
- y (optional) a list of variables changed by S
- P and Q are predicates (first-order logic formulae over program variables)

Objective of Hoare triples

- Specify requirements for S satisfying pre P and post Q.
- Multiple instances of S may satisfy spec.

Total and Partial Correctness

Total Correctness

{P} S {Q} is TRUE if and only if the following holds:

- If P is TRUE and S executes, then
  - S will terminate
  - and Q will be TRUE

Partial Correctness

{P} S {Q} is TRUE if and only if the following holds:

- If P is TRUE and S executes and S terminates, then:
  - Q will be TRUE

Weakest Precondition

For a program S and a postcondition Q,

wp(S, Q) denotes the weakest precondition of S wrt Q

Properties of wp(S, Q)

- {wp(S, Q)} S {Q} is true
- ∀P : ({P} S {Q}) implies (P => wp(S, Q))

Refinement – General approach

1. Start with {P} S {Q} where S is an abstraction of code
2. Refine iteratively using refinement rules (and common sense)
3. Certain rules replace parts of abstract code with GCL commands
4. Each rule preserves correctness of {P} S {Q}
5. Eventually arrive at GCL code, C, such that {P} C {Q} is true

Refinement Rule 1: Skip Statement

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$$\{\mathtt{P}\}\ \mathit{skip}\ \{\mathtt{Q}\}\ \mathit{iff}\ \mathtt{P}\ \mathit{implies}\ \mathtt{Q}$$

Refinement Rule 2: Assignment

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$$\{\mathtt{P}\}\ x := E\ \{\mathtt{Q}\}\ \textit{iff}\ \mathtt{P}\ \textit{implies}\ \mathtt{Q}[\mathtt{x} := \mathtt{E}]$$

Refinement Rule 3: Composition

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$\{\mathtt{P}\}\ \mathtt{S1}\,;\,\mathtt{S2}\ \{\mathtt{Q}\}$ *iff there is an intermediate condition* $\mathtt{M}$ *such that* $\{\mathtt{P}\}\ \mathtt{S1}\ \{\mathtt{M}\}$ *and* $\{\mathtt{M}\}\ \mathtt{S2}\ \{\mathtt{Q}\}$

Refinement Rule 4: Selection

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$\{\mathtt{P}\}$ **if** $\mathtt{G_1} \rightarrow \mathtt{S_1}$ **elseif** $\ldots\ \mathtt{G_n} \rightarrow \mathtt{S_n}$ **fi** $\{\mathtt{Q}\}$ *iff* ($\mathtt{P}$ *implies* $\mathtt{G_1} \vee \mathtt{G_2} \vee \ldots \vee \mathtt{G_n}$) *and* $\{\mathtt{P} \wedge \mathtt{G_i}\}\ \mathtt{S_i}\ \{\mathtt{Q}\}$ *holds for all* i.

The idea of loop invariants

Consider predicate P

P is a loop Invariant, then

- If P is **true** at †, then P is **true** at ▽:

$$
\begin{array}{llll}
\dagger & \textbf{do} & \triangledown & G_1 \rightarrow S_1 \triangledown \\
 & [] & \triangledown & G_2 \rightarrow S_2 \triangledown \\
 & & & \ldots \\
 & [] & \triangledown & G_n \rightarrow S_n \triangledown \\
 & \textbf{od} & \triangledown &
\end{array}
$$

- or, assumming $G \equiv (G_1 \vee \cdots \vee G_n)$:

$$\dagger \quad \textbf{do} \quad \triangledown \quad G \rightarrow S \triangledown \quad \textbf{od} \quad \triangledown$$

(the 2's are sup
name: dr-brucev
that's what it is :
bruce@fastar.or
web; and my ph
+1-650-488-49
position (under r
thinking of "Fou

The idea of loop variants

The variant of a loop is …

An *integer expression*, say V, such that

- letting V_0 denote the value of V at the start of each iteration;
- then $(0 \leq V < V_0$ is TRUE at the end of each iteration
- **Note:** Strictly speaking $(0 \leq V < V_0)$ should be written as $(0 \leq V) \wedge (V < V_{0)}$

Loop variants are monotonically decreasing and bounded from below.

Example:

$i := 10;$ **do** $(i > 0) \rightarrow i := i - 1$ **od**

i is a variant of the loop,

$\because (0 \leq i < i_0) =$ TRUE at the end of each iteration

(the 2's are superscript); Skype name: dr-brucewatson (I think that's what it is :-D); email: bruce@fastar.org; fastar.org for web; and my phone number +1-650-488-4916. For job title/ position (under my name?), I was thinking of "Founder" or "Chief Scientist" or "Co-founder"

Refinement Rule 5: Repetition

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$\{\mathtt{P}\}\ \mathbf{do}\ [\mathtt{I},\mathtt{V}]\ \mathtt{G} \rightarrow \mathtt{S}\ \mathbf{od}\ \{\mathtt{Q}\}$ *iff* $(\mathtt{P}$ *implies* $\mathtt{I})$ *and* $(\mathtt{I} \wedge \neg\mathtt{G}$ *implies* $\mathtt{Q})$ *and* $\{\mathtt{I} \wedge \mathtt{G}\}\ \mathtt{S}\ \{\mathtt{I}\}$ *and* $\{\mathtt{I} \wedge \mathtt{G} \wedge \mathtt{V}{=}\mathtt{V}_0\}\ \mathtt{S}\ \{\mathtt{I} \wedge 0{\leq}\mathtt{V} \wedge \mathtt{V}{<}\mathtt{V}_0\}$

Refinement Rule 6: Weaken Precondition

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$$\{\mathtt{P}'\}\ S\ \{\mathtt{Q}\}\ \textit{iff}\ \mathtt{P}\ \textit{implies}\ \mathtt{P}'$$

Refinement Rule 7: Strengthen Postcondition

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$$\{\mathtt{P}\}\ S\ \{\mathtt{Q}'\}\ \textit{iff}\ \mathtt{Q}'\ \textit{implies}\ \mathtt{Q}$$

Refinement Rule 8: Subroutines

$\{\mathtt{P}\}\ \mathtt{S}\ \{\mathtt{Q}\}$ *can be refined to*

$$\{\mathtt{P}\}\ \mathit{Sub}\ \{\mathtt{Q}\}\ \mathit{with\ subroutine}\ \{\mathtt{P}'\}\ \mathit{Sub}\ \{\mathtt{Q}'\}$$
$$\mathit{iff}\ \mathtt{P}\ \mathit{is\ equal\ to}\ \mathtt{P}'\ \mathit{and}\ \mathtt{Q}'\ \mathit{is\ equal\ to}\ \mathtt{Q}$$

Different Refinement Steps

$$\{P\}\ S\ \{Q\}$$

$$\{P\}\ S_0\ \{R\}\ ;\ S_1\ \{Q\}$$

$$\begin{array}{l}
\{\ P\ \} \\
\textbf{if}\ \ G_0 \rightarrow \{\ P \wedge G_0\ \}\ S_0\ \{\ Q\ \} \\
[\!]\ \ G_1 \rightarrow \{\ P \wedge G_1\ \}\ S_1\ \{\ Q\ \} \\
\textbf{fi} \\
\{\ Q\ \}
\end{array}$$

$$\begin{array}{l}
\{\ P\ \} \\
\{\ \textbf{invariant:}\ I\ \} \\
\textbf{do}\ \ G_0 \rightarrow \{\ I \wedge G_0\ \}\ S_0\ \{\ I\ \} \\
[\!]\ \ G_1 \rightarrow \{\ I \wedge G_1\ \}\ S_1\ \{\ I\ \} \\
\textbf{od} \\
\{\ I \wedge \neg G_0 \wedge \neg G_1\ \} \\
\{\ Q\ \}
\end{array}$$

Summary – Foundations of CbC

Hoare Triples as Program Specification

Hoare Logic Rules for

- Assignment
- Selection
- Repetition

Refinement Rules

- Weaken Precondition
- Strengthen Postcondition
- Program Logic Rules

CbC by Example

- **Simple examples**
  - Linear search
  - Pattern matching

- **Advanced example**
  - Three Banner Flag (Dutch National Flag)

Derrick G. Kourie, and Bruce W. Watson. The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

Linear search

Determine which index i in array A contains x

Assumptions:

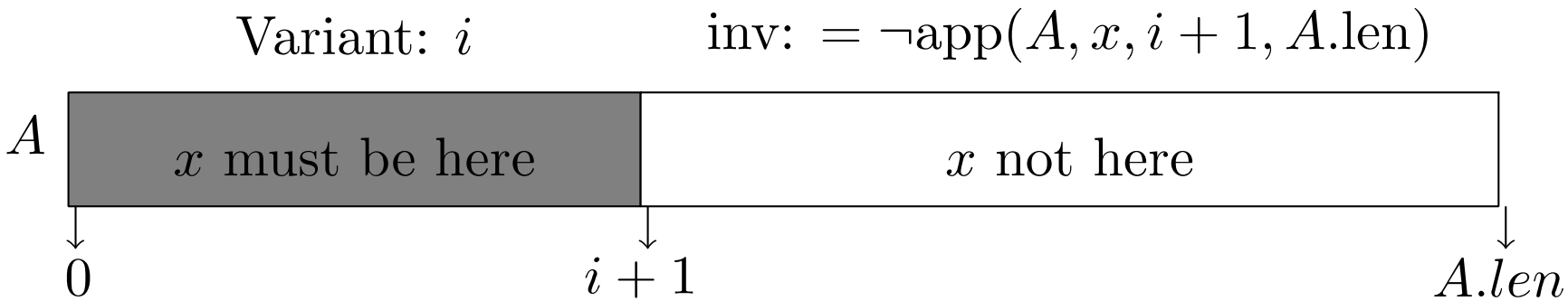
- At least one i such that $A_i = x$
- No preferred i if x occurs multiply in A.

Linear search – more formally

Formally:

- **Pre-condition:** $\mathrm{app}(A, x, k, \ell) \triangleq \exists i \in [k, \ell) : (A_i = x)$
- **Post-condition:** $A_i = x$
- **Problem:** $\{\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : S\ \{(A_i = x)\}$

Linear search – choosing (in)variant



- Strategic choice: sweep from right to left
- Suggests invariant and variant indicated above

Linear search – initialization

$$\{\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : S\ \{(A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Strengthen postcon: } \mathrm{inv} \wedge (A_i = x) \Rrightarrow (A_i = x)\}$$

$$\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : S\ \{\mathrm{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Composition: Mid predicate is inv}\}$$

$$\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : \mathrm{init}; \{\mathrm{inv}\} S_2\ \{\mathrm{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Assignment: } \mathrm{app}(A, x, 0, A.\mathrm{len}) \Rrightarrow \neg\mathrm{app}(A, x, i+1, A.\mathrm{len})[i \backslash A.\mathrm{len} - 1]\}$$

$$\mathrm{app}(A, x, 0, A.\mathrm{len})\}\ i : i := A.\mathrm{len} - 1; \{\mathrm{inv}\}\ S_2\ \{\mathrm{inv} \wedge (A_i = x)\}$$

Linear search – loop structure

$$\{\text{inv}\}\ S_2\ \{\text{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Repetition: } \neg G \text{ is } (A_i = x)\ \}$$

$$\{\text{inv}\}\ \textbf{do}\ (A_i \neq x) \rightarrow B\ \textbf{od}\ \{\text{inv} \wedge (A_i = x)\}$$

$$\sqsubseteq \quad \{\text{Insert implicit pre- and post con for } B\ \}$$

$$\{\text{inv}\}$$

$$\textbf{do}\ (A_i \neq x) \rightarrow$$

$$\{(A_i \neq x) \wedge inv\}\ B\ \{\text{inv}\}$$

$$\textbf{od}$$

$$\{\text{inv} \wedge (A_i = x)\}$$

Linear search – loop body

$$\{(A_i \neq x) \wedge \text{inv}\}\ B\ \{\text{inv}\}$$

$$\sqsubseteq \quad \{\text{Variant bounds strengthen postcon: } \text{inv} \wedge (0 \leq i < i_0)\}$$

$$\{(A_i \neq x) \wedge \text{inv}\}\ B\ \{\text{inv} \wedge (0 \leq i < i_0)\}$$

$$\sqsubseteq \quad \{\text{Assignment: } \text{inv} \wedge (A_i \neq x) \Rrightarrow (\text{inv} \wedge (0 \leq i < i_0)[i \backslash i - 1]\ \}$$

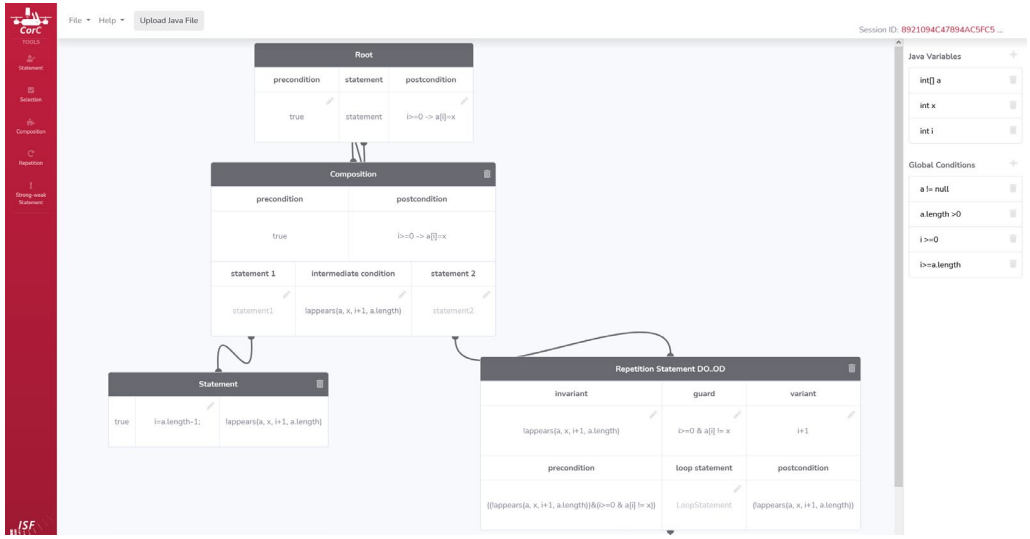
$$\{\text{inv} \wedge (A_i \neq x))\}\ i := i - 1\ \{\text{inv} \wedge (0 \leq i < i_0)\}$$

Linear search – final version

$$
\begin{array}{l}
\{\ \ \mathrm{app}(A, x, 0, A.\mathrm{len})\}\ \ \} \\
i := A.\mathrm{len} - 1; \\
\{\ \ \mathrm{inv} \triangleq \neg\mathrm{app}(A, x, i+1, A.\mathrm{len})\ \ \} \\
\mathbf{do}\ \ (A_i \neq x) \rightarrow \\
\qquad i := i - 1 \\
\mathbf{od} \\
\{\ \ \{\mathrm{inv} \wedge (A_i = x)\}\ \ \}
\end{array}
$$

Hands-On with WebCorC

- https://www.isf.cs.tu-bs.de/WebCorC/
- Web-based tool for correctness-by-construction
- Supports 5 refinement rules
- KeY used to verify the statements



Tobias Runge, Ina Schaefer, Loek Cleophas, Thomas Thüm, Derrick G. Kourie, and Bruce W. Watson. "Tool support for correctness-by-construction." International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, 2019.

CbC by Example

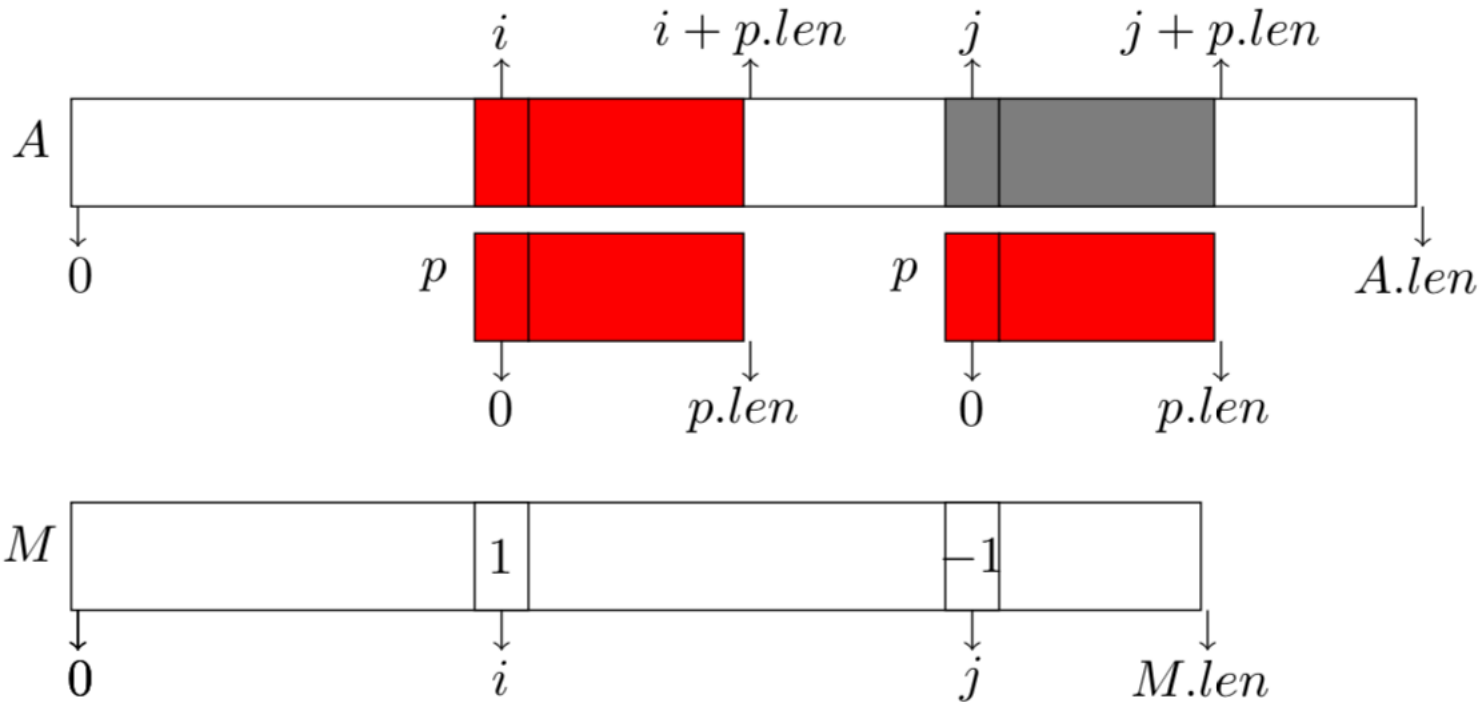
- **Simple examples**
  - Linear search
  - Pattern matching

- **Advanced example**
  - Three Banner Flag (Dutch National Flag)

Derrick G. Kourie, and Bruce W. Watson. The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

Pattern Matching – overview

Indicate in array *M* where pattern *p* matches in *A*



$$\text{Note: } M.len \geq \max(0, (A.len - p.len + 1))$$

Pattern Matching – problem specification

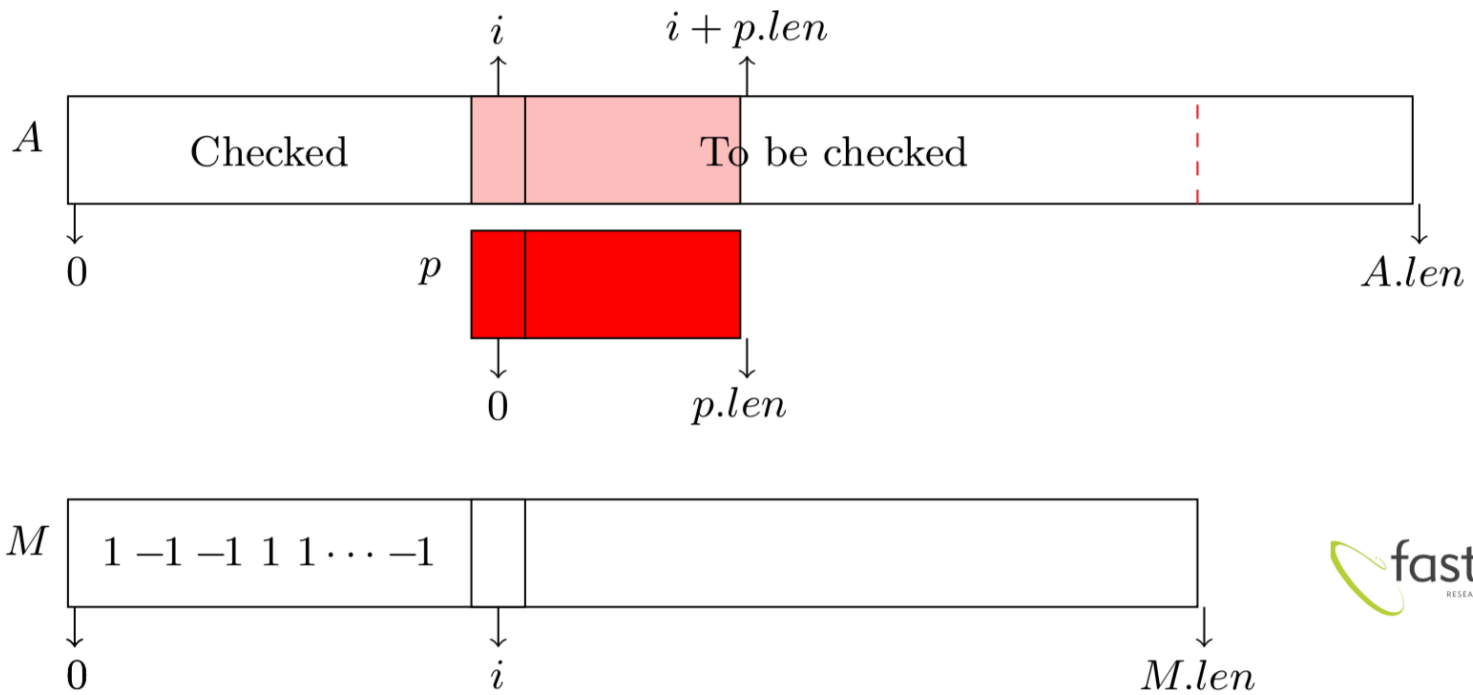
Problem Specification: $M : [Pre, Post]$

$$Pre \equiv M.len \geq \max(0, (A.len - p.len + 1))$$

$$\begin{aligned} Post \equiv\ & \forall k \in [0, A.len - p.len + 1) : \\ & (M_k = 1 \ \wedge \ A_{[k,k+p.len)} = p) \vee \\ & (M_k = -1 \ \wedge \ A_{[k,k+p.len)} \neq p) \end{aligned}$$

Pattern Matching – choosing the invariant

$$
\begin{aligned}
inv(i) \quad \equiv \quad & \forall k \in [0, i) : \\
& (M_k = 1 \ \wedge \ A_{[k,k+p.len)} = p) \ \vee \\
& (M_k = -1 \ \wedge \ A_{[k,k+p.len)} \neq p)
\end{aligned}
$$



Pattern Matching – composition and establishing the invariant

$$M : [Pre, Post]$$

$$\equiv \quad \{\text{Write } Post \text{ in alternative format; introduce frame variable } i\}$$

$$M, i : [Pre, inv(i) \wedge i \geq \max(0, (A.len - p.len + 1))]$$

$$\sqsubseteq \quad \{\text{Composition}\}$$

$$M, i : [Pre, inv(i)]; M, i : [inv(i), inv(i) \wedge i \geq \max(0, (A.len - p.len + 1))]$$

$$\sqsubseteq \quad \{\text{Assignment: } inv(i)[i \backslash 0] \equiv \text{TRUE and } Pre \Rrightarrow \text{TRUE } \}$$

$$i := 0;$$

$$M, i : [inv(i), inv(i) \wedge i \geq \max(0, (A.len - p.len + 1))]$$

Pattern Matching – developing the loop

$$M, i : [inv(i), inv(i) \wedge i \geq \max(0, (A.len - p.len + 1))]$$

$$\sqsubseteq \quad \{\text{Repetition: } \neg G \equiv i \geq \max(0, (A.len - p.len + 1)); V(i) = A.len - i\}$$

$$\mathbf{do}(i < \max(0, (A.len - p.len + 1))) \rightarrow$$

$$\quad M, i : [inv(i) \wedge i \neq (A.len - p.len + 1), inv(i) \wedge (0 \leq V(i) < V(i_0))]$$

$$\mathbf{od}$$

$$\sqsubseteq \quad \{\text{Decrease variant using “Following assignment rule”. }\}$$

$$\mathbf{do}(i < (A.len - p.len + 1)) \rightarrow$$

$$\quad M, i : [inv(i) \wedge i \neq (A.len - p.len + 1), inv(i)[i \backslash i + 1])];$$

$$\quad i := i + 1$$

$$\mathbf{od}$$

Pattern Matching – developing the loop (cont)

- Drop variant part of postcon dropped for brevity
- Informally $inv(i)[i \backslash i+1]$ asserts that $M_{[0,i+1)}$ is ok.

$$inv(i)[i \backslash i+1] \equiv \forall k \in [0, i+1) : (M_k = 1 \ \wedge \ A_{[k,k+p.len)} = p) \vee (M_k = -1 \ \wedge \ A_{[k,k+p.len)} \neq p)$$

- $M, i : [inv(i) \wedge i \neq (A.len - p.len + 1), inv(i)[i \backslash i+1])]$

$\sqsubseteq$ {Use selection rule??}

if $(A_{[i,i+p.len)} = p) \rightarrow M_i := 1$

$[\!]$ $(A_{[i,i+p.len)} \neq p) \rightarrow M_i := -1$

fi

- Not a practical solution
- Loop & if-statement to check — Self study!

Pattern Matching – final version

Put all the pieces together:

$$
\begin{array}{l}
i := 0; \\
\textbf{do}\ \ i < A.len - p.len + 1 \rightarrow \\
\quad j := 0; \\
\quad \textbf{do}\ \ (j \neq p.len \wedge p_j = A_{i+j}) \rightarrow j := j + 1\ \ \textbf{od}; \\
\quad \textbf{if}\ \ (j = p.len) \rightarrow M_i := 1\ \ [\!]\ \ (j \neq p.len) \rightarrow M_i := -1\ \ \textbf{fi}; \\
\quad i := i + 1 \\
\textbf{od}
\end{array}
$$

CbC by Example

- **Simple examples**
  - Linear search
  - Pattern matching

- **Advanced example**
  - Three Banner Flag (Dutch National Flag)

Derrick G. Kourie, and Bruce W. Watson. The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

Three Banner Flag a.k.a. Dutch National Flag

Coined as Dutch National Flag problem by Dijkstra in 1976

Based on arrays of red, white, blue values to be sorted into DNF order

More generally any 3-valued array
or e.g. any 3-bannered flag, e.g.
Sierra Leone National Flag

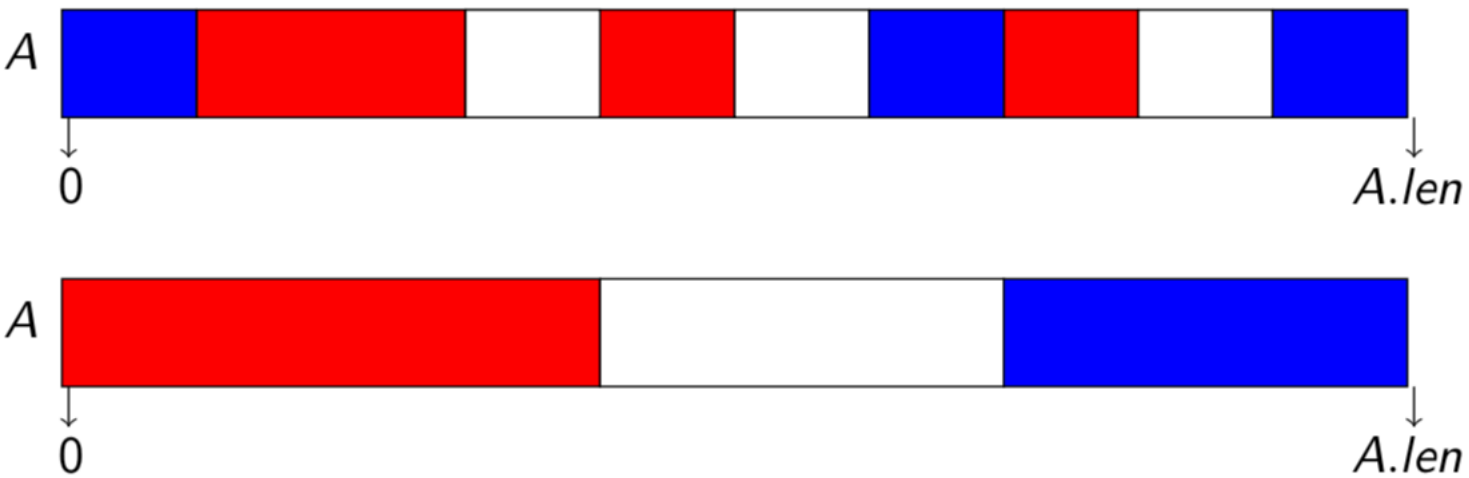
Irish National Flag

German National Flag

…

TBF/DNF – overview

Sort array A containing red, white & blue values in $\mathcal{O}(N)$



TBF/DNF – some definitions

Definition (A coloured region)

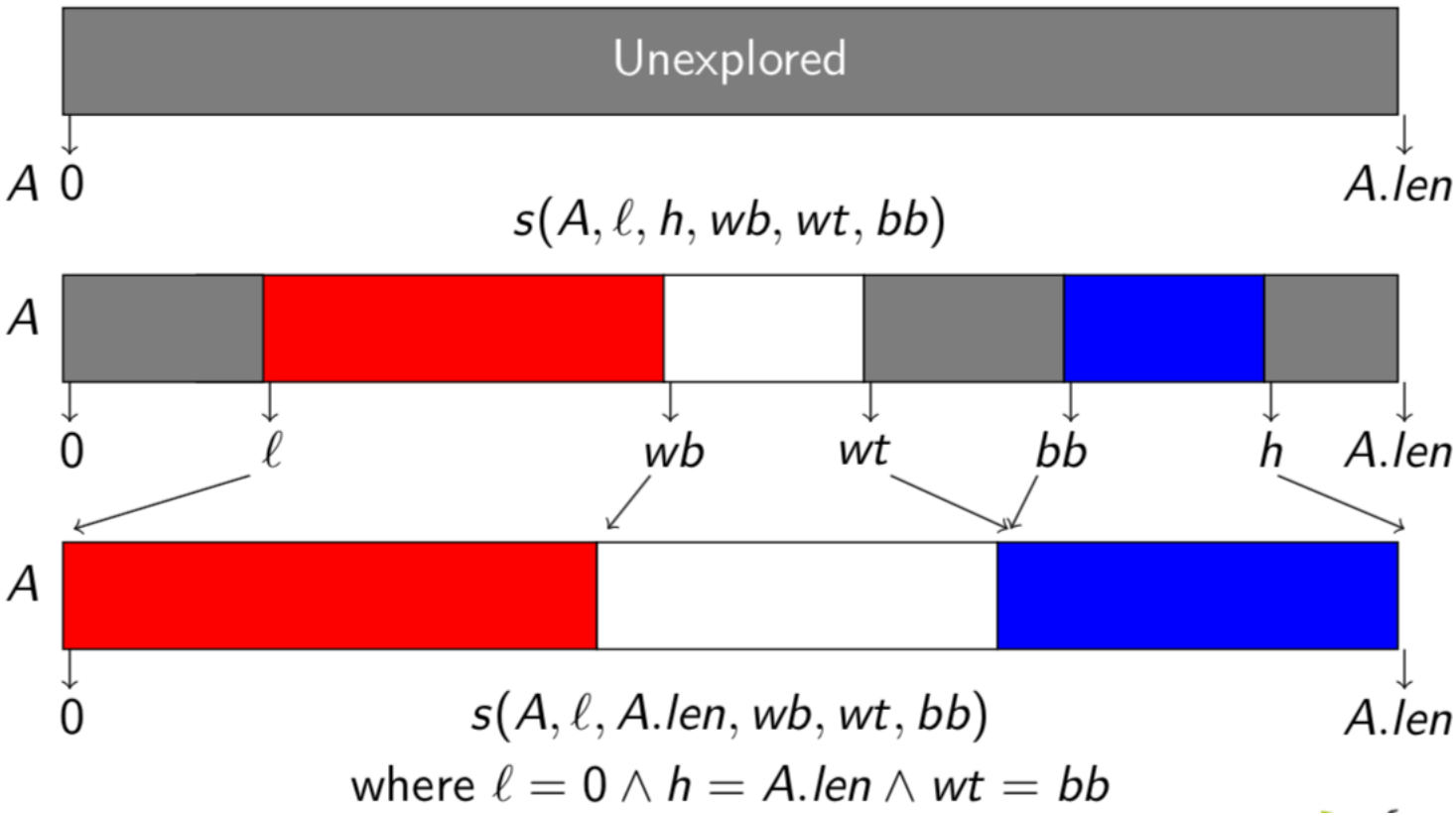
$c(A, \ell, h, x) \triangleq \forall i : [\ell, h) \cdot (A_i = x)$

i.e. everything in $A_{[\ell,h)}$ has colour x

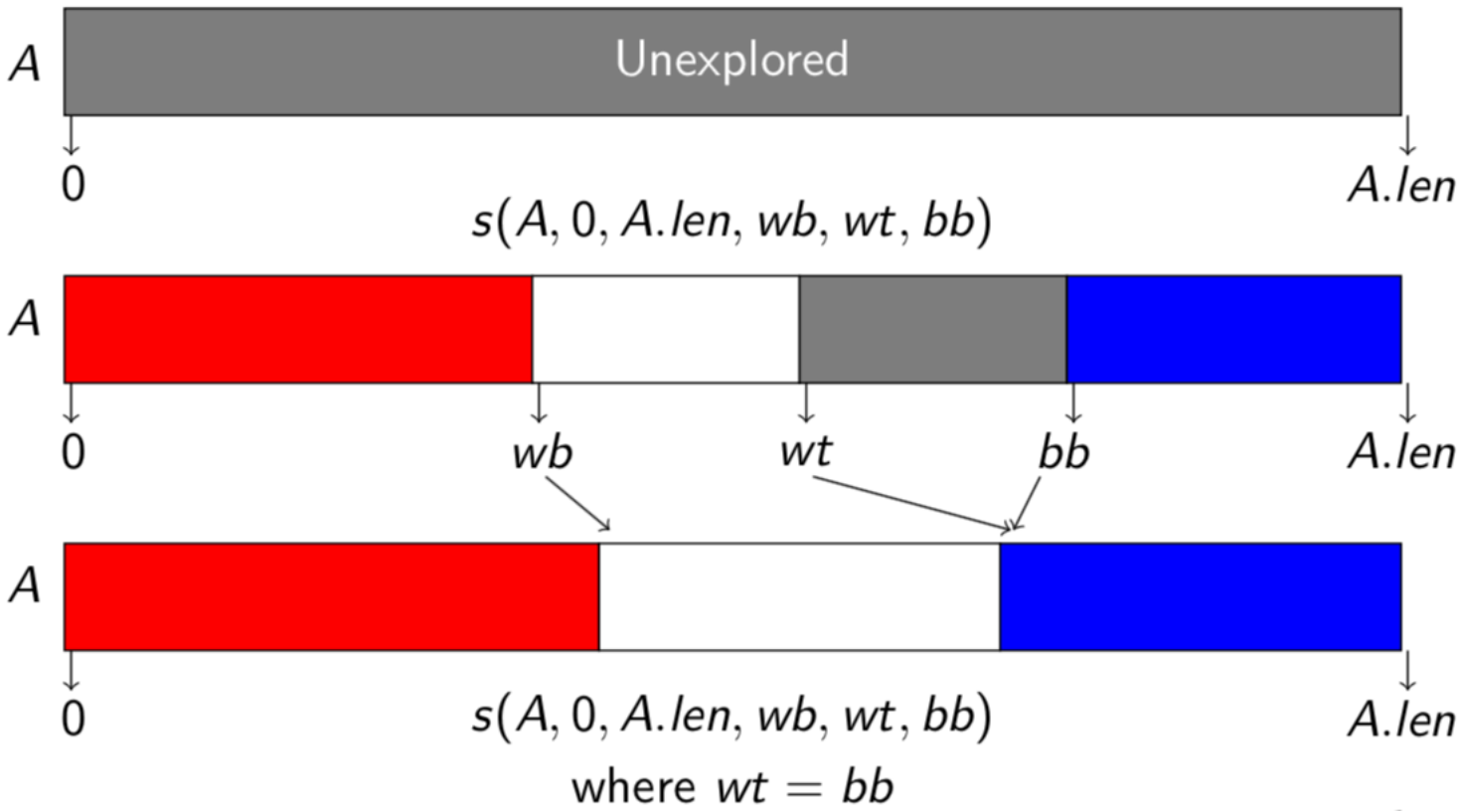
Definition (An arbitrary state)

$$
\begin{aligned}
s(A, \ell, h, wb, wt, bb) \quad \triangleq \quad & c(A, \ell, wb, r) \wedge \\
& c(A, wb, wt, w) \wedge \\
& c(A, bb, h, b) \wedge \\
& 0 \leq \ell \leq wb \leq wt \leq bb \leq h \leq A.len
\end{aligned}
$$

TBF/DNF – a possible route



TBF/DNF – a better route: $\ell = 0 \wedge h = A.len$



TBF/DNF – problem specification and invariant

- Use $s(A, wb, wt, bb)$ instead of $s(A, 0, A.len, wb, wt, bb)$
- Spec: $A, wb, wt, bb : [Pre, Post]$ where
  - Pre: $s(A, 0, 0, A.len)$
  - Post: $s(A, wb, wt, bb) \wedge (wt = bb)$
- Invariant: $s(A, wb, wt, bb)$

TBF/DNF – composition and establishing the invariant

$$A, wb, wt, bb : [Pre, Post]$$

$$\equiv \quad \{\text{Substitute. (Drop frame variables)}\}$$

$$[s(A, 0, 0, A.len), s(A, wb, wt, bb) \wedge (wt = bb)]$$

$$\sqsubseteq \quad \{\text{Sequence to bring in inv}\}$$

$$[s(A, 0, 0, A.len), s(A, wb, wt, bb)] \ ;$$

$$[s(A, wb, wt, bb), s(A, wb, wt, bb) \wedge (wt = bb)]$$

$$\sqsubseteq \quad \{\text{Assignment: } pre \Rrightarrow inv[wb, wt, bb \setminus 0, 0, A.len]\}$$

$$wb, wt, bb := 0, 0, A.len \ ;$$

$$[s(A, wb, wt, bb), s(A, wb, wt, bb) \wedge (wt = bb)]$$

TBF/DNF – developing the loop

Focus on deriving the loop:

$$[s(A, wb, wt, bb), s(A, wb, wt, bb) \wedge (wt = bb)]$$

$$\sqsubseteq \quad \{\text{Repetition. Variant is } bb - wt\}$$

$$\textbf{do } (wt \neq bb) \rightarrow$$

$$[(wt \neq bb) \wedge s(A, wb, wt, bb),$$

$$s(A, wb, wt, bb) \wedge 0 \leq (bb - wt) < (bb_0 - wt_0)]$$

$$\textbf{od}$$

TBF/DNF – developing the loop (2)

Use *inv* and *var* for invariant and variant predicates

$A, wb, wt, bb : [inv \wedge (wt \neq bb), inv \wedge var]$

$\sqsubseteq$ {selection: $pre \Rrightarrow (A_{wt} = r) \vee (A_{wt} = w) \vee (A_{wt} = b) \equiv \mathrm{TRUE}$}

if $(A_{wt} = r) \rightarrow [inv \wedge (wt \neq bb) \wedge (A_{wt} = r), inv \wedge var]$

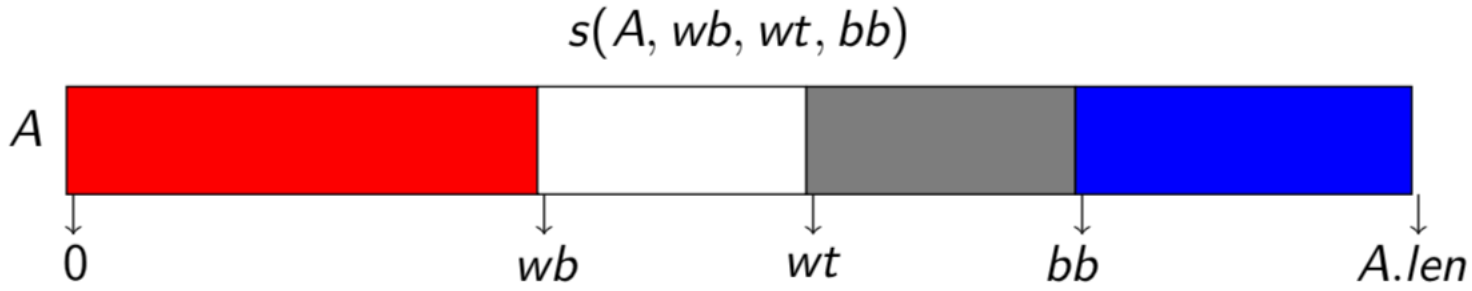
$[\!]$ $(A_{wt} = w) \rightarrow [inv \wedge (wt \neq bb) \wedge (A_{wt} = w), inv \wedge var]$

$[\!]$ $(A_{wt} = b) \rightarrow [inv \wedge (wt \neq bb) \wedge (A_{wt} = b), inv \wedge var]$

fi

$\sqsubseteq$ {assignment rule}

TBF/DNF – developing the loop (3)



$$
\begin{array}{l}
\textbf{if } (A_{wt} = r) \rightarrow A_{wt}, A_{wb}, wt, wb := A_{wb}, A_{wt}, wt + 1, wb + 1 \\
\quad [\!] \ (A_{wt} = w) \rightarrow wt := wt + 1 \\
\quad [\!] \ (A_{wt} = b) \rightarrow A_{wt}, A_{bb-1}, bb := A_{bb-1}, A_{wt}, bb - 1 \\
\textbf{fi}
\end{array}
$$

TBF/DNF – final algorithm

$$
\begin{array}{l}
wb, wt, bb := 0, 0, A.len; \\
\textbf{do}\ \ (wt \neq bb) \rightarrow \\
\quad \textbf{if}\ \ (A_{wt} = r) \rightarrow A_{wt}, A_{wb}, wt, wb := A_{wb}, A_{wt}, wt + 1, wb + 1 \\
\quad [\!]\ \ (A_{wt} = w) \rightarrow wt := wt + 1 \\
\quad [\!]\ \ (A_{wt} = b) \rightarrow A_{wt}, A_{bb-1}, bb := A_{bb-1}, A_{wt}, bb - 1 \\
\quad \textbf{fi} \\
\textbf{od}
\end{array}
$$

CbC by Example

- **Simple examples**
  - Linear search
  - Pattern matching

- **Advanced example**
  - Three Banner Flag (Dutch National Flag)

Derrick G. Kourie, and Bruce W. Watson. The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.
Tobias Runge, Ina Schaefer, Loek Cleophas, Thomas Thüm, Derrrick G. Kourie, and Bruce W. Watson. "Tool support for correctness-by-construction." International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, 2019.

XbC – Non-functional Properties by Construction

So far, we have only considered functional correctness by construction.

But there are many other non-functional properties such as performance, memory/energy consumption, security and privacy etc.

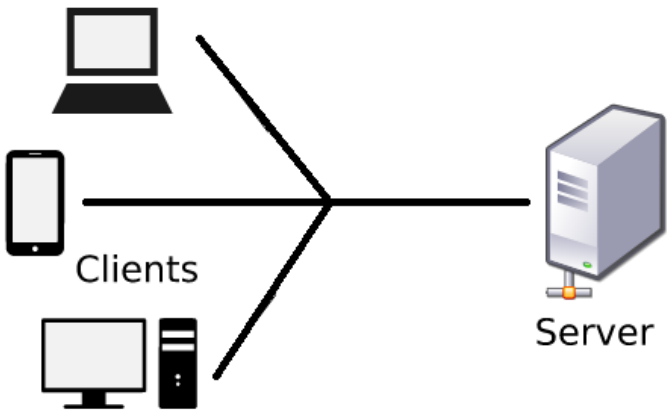
Usually, there is a tradeoff between those properties, e.g., faster computation needs more energy, …

What if we could construct functionally correct programs which also have required non-functional properties by construction?

M. H. ter Beek, L. Cleophas, I. Schaefer, and B. W. Watson. X-by-Construction. In Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'18), volume 11244 of LNCS, pages 359–364. Springer, 2018.
M. H. ter Beek, L. Cleophas, A. Legay, I. Schaefer, and B. W. Watson. X-by-Construction: Correctness Meets Propability. In Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'20), volume 12476 of LNCS, pages 211–215. Springer, 2020.

Information Flow-by-Construction

- **Client** and **server** have different security levels
- **Client** should not read **server** data uncontrolled
  - Ensure confidentiality
- **Server** should not read **client** data uncontrolled
  - Ensure integrity



- Usually security and correctness is ensured by post-hoc analysis (and sometimes post-hoc verification)

- Alternative:
  - Information Flow-by-Construction (IFbC)

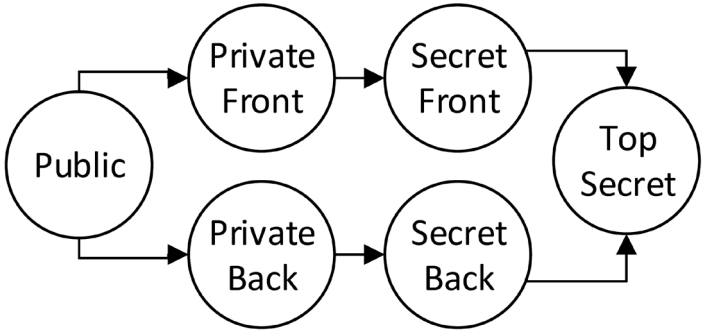
Schaefer I., Runge T., Knüppel A., Cleophas L., Kourie D., Watson B.W. Towards Confidentiality-by-Construction. In: Leveraging Applications of Formal Methods, Verification and Validation. Modeling (ISoLA'18). volume 11244 of LNCS. Springer, 2018.
Runge T., Knüppel A., Thüm T., Schaefer I. Lattice-based Information Flow Control-by-Construction for Security-by-Design. In Proceedings of the 8th International Conference on Formal Methods in Software Engineering (FormaliSE'20). ACM, pp. 44-54.

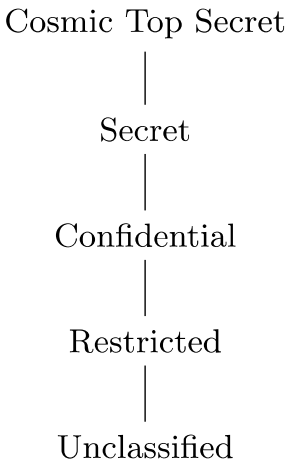
Information Flow Control

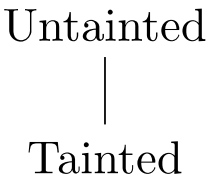
- Non-interference:
  - Confidential data may not be deducible from public data

- Direct Information Flow: Assignments
- Indirect Information Flow: Usage of confidential data in loop-guards and if-conditions

Information Flow Policies (Lattices):







Language-based Security

```
boolean low paymentAction := true;
if (paymentAction) {
  int high creditCard := getNumber();
  while (!valid(creditCard)) {
    creditCard = getNumber();
  }
  String low output := declassify(mask(creditCard));
} else {
  skip
}
```

Security Type Systems [cf. Sabelfeld, Myers, Volpano et al.]

$$\vdash \texttt{exp} : \texttt{high} \qquad \frac{\texttt{h} \notin \texttt{Vars}(\texttt{exp})}{\vdash \texttt{exp} : \texttt{low}} \qquad [\texttt{ct}] \vdash \texttt{skip} \qquad [\texttt{ct}] \vdash \texttt{h} = \texttt{exp} \tag{1–4}$$

$$\frac{\texttt{exp} : \texttt{low}}{[\texttt{low}] \vdash \texttt{l} = \texttt{exp}} \qquad \frac{[\texttt{ct}] \vdash \texttt{S}_1 \quad [\texttt{ct}] \vdash \texttt{S}_2}{[\texttt{ct}] \vdash \texttt{S}_1; \texttt{S}_2} \qquad \frac{\vdash \texttt{exp} : \texttt{ct} \quad [\texttt{ct}] \vdash \texttt{S}}{[\texttt{ct}] \vdash \texttt{while exp do S}} \tag{5–7}$$

$$\frac{\vdash \texttt{exp} : \texttt{ct} \quad [\texttt{ct}] \vdash \texttt{S}_1 \quad [\texttt{ct}] \vdash \texttt{S}_2}{[\texttt{ct}] \vdash \texttt{if exp then S}_1 \texttt{ else S}_2} \tag{8}$$

Information Flow Control-by-Construction by Example

```
1   pre: publishBid = 0
2   post: \forall int x; ((x >= 0
3     & x < bids.length)
4     -> (publishBid >= bids[x]))
5   void auction(private int[] bids,
6     public int publishBid) {
7     public int i = 0;
8     secret int highestBid = 0;
9     do (i < bids.length) {
10      if (highestBid < bids[i]) {
11        highestBid = bids[i];
12      else {
13        skip
14      } fi
15      i = i + 1;
16    } od
17    publishBid = declassify(highestBid);
18  }
```

Initialization

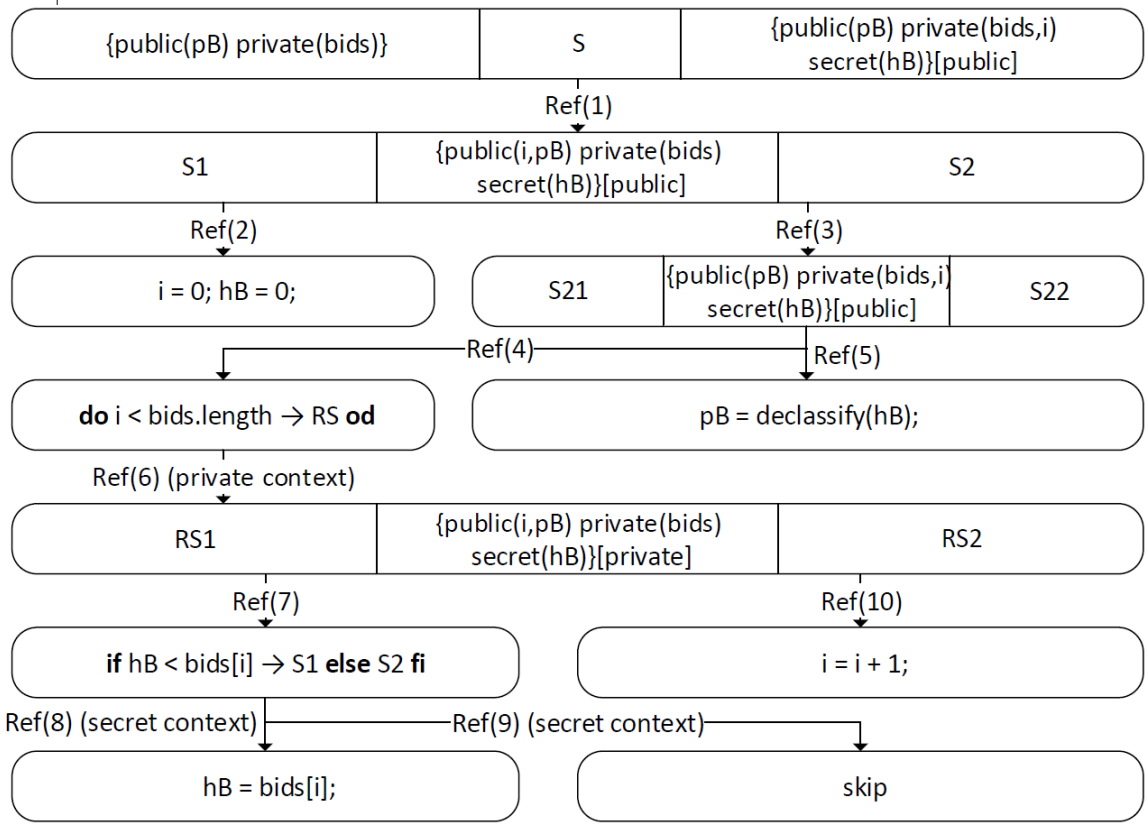
Check for higher bidding

Loop through the array of bids

Publishing the result

Incremental Construction of the Algorithm

```
1  pre: publishBid = 0
2  post: \forall int x; ((x >= 0
3    & x < bids.length)
4    -> (publishBid >= bids[x]))
5  void auction(private int[] bids,
6    public int publishBid) {
7    public int i = 0;
8    secret int highestBid = 0;
9    do (i < bids.length) {
10     if (highestBid < bids[i]) {
11       highestBid = bids[i];
12     else {
13       skip
14     } fi
15     i = i + 1;
16   } od
17   publishBid = declassify(highestBid);
18 }
```



Refinement Rules – Assignment

- Check that expression E and variable x have the same security level
- Check that variable x has a equal or greater security level than the context

- If not: raise security level of x

RULE 2 (ASSIGNMENT).
$\{\mathcal{V}^{pre}\}\ \mathsf{S}\ \{\mathcal{V}^{post}\}[\eta]$ *is refinable to* $\{\mathcal{V}^{pre}\}\ \mathsf{x} := \mathsf{E}\ \{\mathcal{V}^{post}\}[\eta]$ *iff* $\mathcal{V}^{post}(\mathsf{y}) = \mathcal{V}^{pre}(\mathsf{y})$ *for all* $\mathsf{y} \in Vars \setminus \{x\}$, *and* $\mathcal{V}^{post}(\mathsf{x}) = lub(\{\mathcal{V}^{pre}(\mathsf{v}) \mid \mathsf{v} \in vars(\mathsf{E})\} \cup \{\mathcal{V}^{pre}(\mathsf{x}), \eta\})$.

Refinement Rules – Repetition

- Update context of Hoare triple
  - Least upper bound of old context and security level of guard G

RULE 5 (REPETITION).
$\{\mathcal{V}^{pre}\}$ S $\{\mathcal{V}^{post}\}[\eta]$ *is refinable to* $\{\mathcal{V}^{pre}\}$ **do** G $\rightarrow$ S1 **od** $\{\mathcal{V}^{post}\}$ $[\eta]$ *iff* $\{\mathcal{V}^{pre}\}$ S1 $\{\mathcal{V}^{post}\}[\eta']$ *with* $\eta' = lub(\{\mathcal{V}^{pre}(\mathrm{v}) \mid \mathrm{v} \in vars(\mathrm{G})\} \cup \{\eta\})$

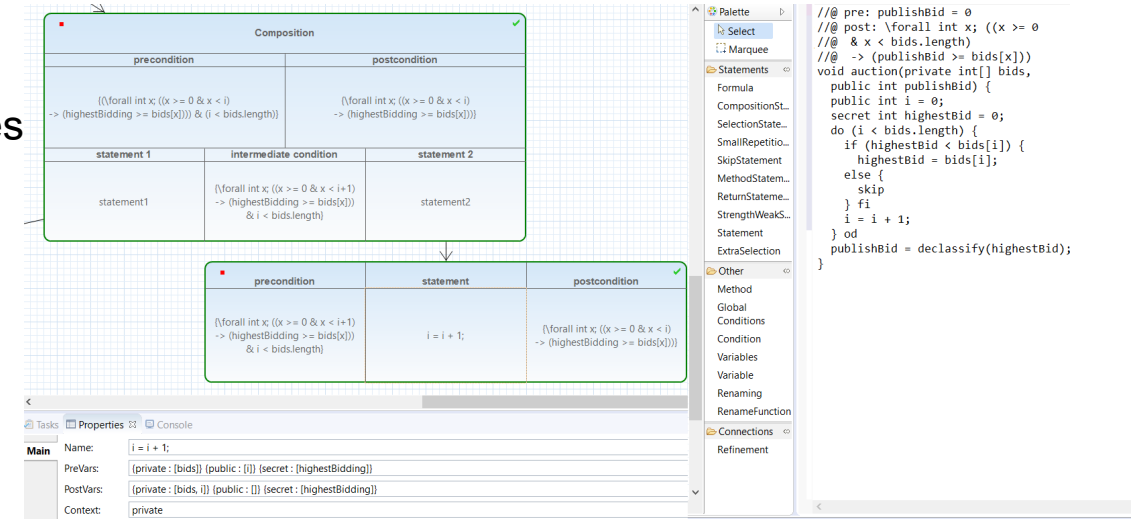
Refinement Rules – Method Call

- Call of other secure programs
- Similar to substitution of method body

RULE 6 (METHOD CALL).
$\{\mathcal{V}^{pre}\}\ \mathsf{S}\ \{\mathcal{V}^{post}\}[\eta]$ *is refinable to* $\{\mathcal{V}^{pre}\}\ \mathsf{M}(\mathsf{a}_1, \ldots, \mathsf{a}_n)\ \{\mathcal{V}^{post}\}[\eta]$ *iff for a method* $\{\mathcal{V}^{pre}_{call}\}\ \mathsf{M}(\mathsf{z}_1, \ldots, \mathsf{z}_n)\ \{\mathcal{V}^{post}_{call}\}[\eta]$ *and for all parameters:* $\mathcal{V}^{pre}(\mathsf{a}_i) \leq \mathcal{V}^{pre}_{call}(\mathsf{z}_i) \wedge \mathcal{V}^{post}_{call}(\mathsf{z}_i) \leq \mathcal{V}^{post}(\mathsf{a}_i)$ *where* a_i *are the actual parameters and* z_i *are the formal parameters.*

CorC – Tool Support for CbC and Information Flow Control

- Eclipse plug-in
- Hybrid: Textual and graphical editor
- KeY used as verification backend
- Implementation of refinement rules for information flow control



Available at https://github.com/TUBS-ISF/CorC

Information Flow Rules for Object-Oriented Languages

- Objects: Aliases could lead to problems
  - Updates not always feasible

$$\mathrm{low}\ Object\ o = \mathrm{new\ Object}();$$
$$\mathrm{high}\ Object\ p = o;$$
$$p.set(somethingPrivate);$$
$$o.get();$$ //leaks the private information

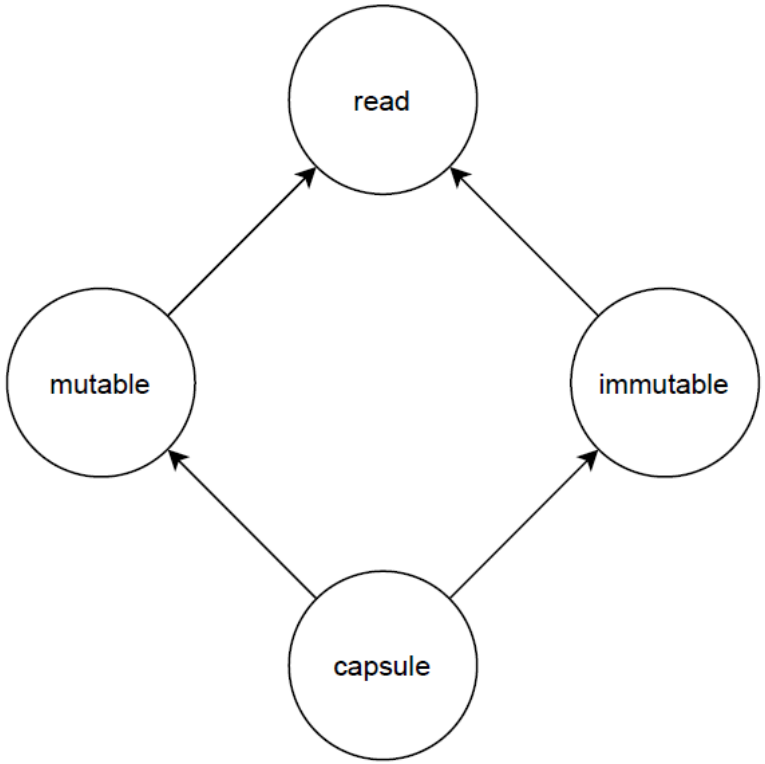
  - Rules use type modifiers to extend expressiveness
    - E.g. if object is immutable, .set() would be prohibited

Runge T., Servetto M., Potanin A., Schaefer I. Immutability and Encapsulation for Sound OO Information Flow Control. In: ACM Transactions on Programming Languages and Systems (TOPLAS). Under Review.

Information Flow in Object-Oriented Languages with Type Modifiers

- Type modifiers increase the precision
- Utilizing immutability and uniqueness properties

- Type modifiers:
  - **Read**: No modification, no aliases
  - **Mutable**: modification, aliases
  - **Immutable**: No modification, aliases
  - **Capsule**: modification, isolated portion of store



Information Flow-by-Construction

- Refinement rules that comply with the information flow policy
  - Any lattice of security levels

- CorC tool support
  - Functional correctness
  - Information flow control

- Future work:
  - Extension for object-oriented languages

The other camp: Post-Hoc Verification (PhV)

- Program annotated with pre/post-conditions and/or class invariants
- Auxiliary annotations, e.g., loop (in)variants
- 2 main verification approaches
  - Verification condition generation
  - Symbolic execution

Bruce W. Watson, Derrick G. Kourie, Ina Schaefer, Loek Cleophas: Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience? ISoLA (1) 2016: 730-748

CbC vs PhV – Code Structure

- PhV constructs machine-checked proof that is correct, subject to correctness of proof calculus and correctness of prover
- PhV weakness in articulating the predicates to verify code developed in an ad-hoc fashion with poor structure
- CbC generally results in well-structured code, a byproduct of articulating the specifications and annotations needed by PhV proof tools.
- CbC allows the taxonomization of algorithmic families

CbC vs PhV - Formality

- CbC is concerned with correctness at level of intuitive meaning.
- But if CbC specifications are too informal, we risk errors!
- PhV can fill gap, allowing some informality of CbC-development, and then PhV-checking with invariants, variants, pre- and postconditions worked out by CbC

CbC vs. PhV – Tool Support

Main drawback of CbC: Lack of tool support

Tool support for CbC strongly relies on advances made for PhV program verification, e.g.,

- uninterpreted predicates and unknown program parts while a program is refined in CbC
- different interaction and editing capabilities for developer

Combining CbC and PhV

1. CbC-derive elegant algorithmic, simultaneously providing pre/post-specifications and variant/invariant annotations.

2. Translate the CbC-derived program into the programming language required by PhV proof tool.

3. Necessary to translate annotations into the logical notation syntax used by tool.

4. Possibly: Add auxiliary annotations.

5. Use the prover tool to apply PhV to the translated CbC-derived program.

CbC - Advantages

- CbC generally results in well-structured code.
- CbC allows the taxonomization of algorithmic families.
- CbC saves testing effort and can be used to bootstrap post-hoc verification.
- CbC allows faster time to market.

CbC - Disadvantages

- CbC is concerned with correctness at level of intuitive meaning.
- But if CbC specifications are too informal, we risk errors!
- Main drawback of CbC: Lack of tool support
- Tool support for CbC strongly relies on advances made for PhV program verification.

Applying CbC to other Domains and Languages

- CbC for Parallel and Cloud-based Programming
- CbC for Data-flow Languages, such as Matlab/Simulink
- CbC for Software Product Lines and Variability

Prerequisite is CbC-Readiness:

- Formal Language Syntax
- Formal Specification Language
- Proof Rules for Specifications over Programs
- Refinement Rules for Programs and Specifications

Industrial Uptake – Success Stories

Case: Large semi-conductor manufacturing company

- Mix of software for robotics (motion planning), metrology, mechatronics, optimization for semi-conductor manufacturing machines
- High robustness, sensitivity, performance requirements
- Physical and financial consequences to a software crash
- Tens of millions of lines of C and C++ code
- 100% faithful simulation software for optimization purposes
- Simulation code's kernel developed using CbC

Comparing CbC against traditional SE process:

- Factor 10 reduction in time to test-suite validation
- Factor 3 reduction in size of team required
- Cleaner and more maintainable code
- Clearly requires some CbC-skills on-team

Conclusions

- CbC *lite* should be widely used, even in the presence of verification tools, or if verification is mandated by standards.
- CbC should be taught more widely.
- Tool Support needs to be extended for wide-spread adoption.

Literature and References

Kourie D. G., Watson B. W. The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

Runge T., Schaefer I., Cleophas L., Thüm T., Kourie D. G., Watson B. W. "Tool support for correctness-by-construction." International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, 2019.

ter Beek M. H., Cleophas L., Schaefer I., Watson and B. W. X-by-Construction. In Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'18), volume 11244 of LNCS, pages 359–364. Springer, 2018.

ter Beek M. H., Cleophas L., Legay A., Schaefer I., Watson B. W. X-by-Construction: Correctness Meets Propability. In Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'20), volume 12476 of LNCS, pages 211–215. Springer, 2020.

Schaefer I., Runge T., Knüppel A., Cleophas L., Kourie D. G., Watson B.W. Towards Confidentiality-by-Construction. In: Leveraging Applications of Formal Methods, Verification and Validation. Modeling (ISoLA'18). volume 11244 of LNCS. Springer, 2018.

Runge T., Knüppel A., Thüm T., Schaefer I. Lattice-based Information Flow Control-by-Construction for Security-by-Design. In Proceedings of the 8th International Conference on Formal Methods in Software Engineering (FormaliSE'20). ACM, pp. 44-54.

Runge T., Servetto M., Potanin A., Schaefer I. Immutability and Encapsulation for Sound OO Information Flow Control. In: ACM Transactions on Programming Languages and Systems (TOPLAS). Under Review.