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
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. Pannozzio November 12, 2011
CbC Round 2+
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

<table>
<thead>
<tr>
<th>Command</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Empty command</td>
<td>skip</td>
</tr>
<tr>
<td>2. Chaotic command</td>
<td>abort</td>
</tr>
<tr>
<td>3. Assignment</td>
<td>:=</td>
</tr>
<tr>
<td>4. Composition/concatenation</td>
<td>;</td>
</tr>
<tr>
<td>5. Selection</td>
<td>if</td>
</tr>
<tr>
<td>6. Repetition</td>
<td>do</td>
</tr>
<tr>
<td>7. Comments</td>
<td>{ This is a comment }</td>
</tr>
</tbody>
</table>
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 $Q$,

$\text{wp}(S, Q)$ denotes the weakest precondition of $S$ wrt $Q$

**Properties** of $\text{wp}(S, Q)$

- $\{\text{wp}(S, Q)\} S \{Q\}$ is true
- $\forall P : (\{P\} S \{Q\})$ implies $(P \Rightarrow \text{wp}(S, Q))$
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

\[ \{ P \} \text{ skip } \{ Q \} \quad \text{can be refined to} \quad \{ P \} \text{ implies } \{ Q \} \]
Refinement Rule 2: Assignment

\[
\{ P \} \; S \; \{ Q \} \quad \text{can be refined to}
\]

\[
\{ P \} \; x := E \; \{ Q \} \quad \text{iff} \quad P \text{ implies } Q[x := E]
\]
Refinement Rule 3: Composition

\{P\} S \{Q\} can be refined to

\{P\} S1; S2 \{Q\} iff there is an intermediate condition M such that \{P\} S1 \{M\} and \{M\} S2 \{Q\}
Refinement Rule 4: Selection

\{P\} S \{Q\} \quad \text{can be refined to}

\{P\} \text{ if } G_1 \rightarrow S_1 \text{ elseif } \ldots \text{ } G_n \rightarrow S_n \text{ fi } \{Q\} \text{ iff } (P \text{ implies } G_1 \lor G_2 \lor \ldots \lor G_n) \text{ and } \{P \land G_i\} S_i \{Q\} \text{ holds for all } i.
The idea of loop invariants

Consider predicate $P$

$P$ is a loop Invariant, then

- If $P$ is true at $\dagger$, then $P$ is true at $\triangledown$:
  $$\dagger \quad \text{do} \quad \triangledown \quad G_1 \rightarrow S_1 \quad \triangledown$$
  $$\none \quad \triangledown \quad G_2 \rightarrow S_2 \quad \triangledown$$
  $$\ldots$$
  $$\none \quad \triangledown \quad G_n \rightarrow S_n \quad \triangledown$$
  $$\text{od} \quad \triangledown$$

- or, assuming $G \equiv (G_1 \lor \cdots \lor G_n)$:
  $$\dagger \quad \text{do} \quad \triangledown \quad G \rightarrow S \quad \text{od} \quad \triangledown$$
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 $(0 \leq V) \land (V < V_0)$

Example:

\[ i := 10; \quad \textbf{do} \quad (i > 0) \rightarrow i := i - 1 \quad \textbf{od} \]

$i$ is a variant of the loop,

\[ \therefore (0 \leq i < i_0) = \text{true} \quad \text{at the end of each iteration} \]
Refinement Rule 5: Repetition

\{P\} S \{Q\} \quad can \ be \ refined \ to

\{P\} \textbf{do} [I, V] G \rightarrow S \textbf{od} \{Q\} \iff (P \implies I) \ and \ (I \land \neg G \implies Q) \ and \ \{I \land G\} S \{I\} \ and \ \{I \land G \land V = V_0\} \ S \{I \land 0 \leq V \land V < V_0\}
Refinement Rule 6: Weaken Precondition

\[
\{ P \} \ S \ \{ Q \} \quad \text{can be refined to} \quad \{ P' \} \ S \ \{ Q \} \ \text{iff} \ P \ \text{implies} \ P'
\]
Refinement Rule 7: Strengthen Postcondition

\{P\} S \{Q\} \quad can \ be \ refined \ to \\

\{P\} S \{Q'\} \iff \text{Q' implies Q}
Refinement Rule 8: Subroutines

\( \{P\} S \{Q\} \) can be refined to

\( \{P\} \text{ Sub } \{Q\} \text{ with subroutine } \{P’\} \text{ Sub } \{Q’\} \)

iff \( P \) is equal to \( P’ \) and \( Q’ \) is equal to \( Q \)
Different Refinement Steps

\{ P \} S \{ Q \}

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

\{ P \}
if \ G_0 \rightarrow \{ P \land G_0 \} S_0 \{ Q \}
\| \ G_1 \rightarrow \{ P \land G_1 \} S_1 \{ Q \}
fi
\{ Q \}

\{ P \}
\{ \textbf{invariant: } I \}
do \ G_0 \rightarrow \{ I \land G_0 \} S_0 \{ I \}
\| \ G_1 \rightarrow \{ I \land G_1 \} S_1 \{ I \}
od\{ I \land \neg G_0 \land \neg G_1 \}
\{ Q \}
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)

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:** app\( (A, x, k, \ell) \triangleq \exists i \in [k, \ell) : (A_i = x) \)

- **Post-condition:** \( A_i = x \)

- **Problem:** \( \{ \text{app}(A, x, 0, A.\text{len}) \} i : S \ (A_i = x) \)
Linear search – choosing (in)variant

- Variant: $i$
- inv: $= \neg \text{app}(A, x, i + 1, A.\text{len})$

$A$

$x$ must be here  $x$ not here

0 $i + 1$ $A.\text{len}$

- Strategic choice: sweep from right to left
- Suggests invariant and variant indicated above
Linear search – initialization

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

\sqsubseteq \ \{ \text{Strengthen postcon: inv} \land (A_i = x) \Rightarrow (A_i = x) \}\n
\text{app}(A, x, 0, A.\text{len}) \ i : S \ \{ \text{inv} \land (A_i = x) \}\n
\sqsubseteq \ \{ \text{Composition: Mid predicate is inv} \}\n
\text{app}(A, x, 0, A.\text{len}) \ i : \text{init}; \ \{ \text{inv} \} S_2 \ \{ \text{inv} \land (A_i = x) \}\n
\sqsubseteq \ \{ \text{Assignment: app}(A, x, 0, A.\text{len}) \Rightarrow \neg \text{app}(A, x, i + 1, A.\text{len})[i \backslash A.\text{len} - 1] \}\n
\text{app}(A, x, 0, A.\text{len}) \ i : i := A.\text{len} - 1; \ \{ \text{inv} \} S_2 \ \{ \text{inv} \land (A_i = x) \}
Linear search – loop structure

\{inv\} S_2 \{inv \land (A_i = x)\}

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

\{inv\} \textbf{do} (A_i \neq x) \rightarrow B \quad \textbf{od} \quad \{inv \land (A_i = x)\}

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

\{inv\}

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

\quad \{(A_i \neq x) \land inv\} \quad B \quad \{inv\}

\textbf{od}

\{inv \land (A_i = x)\}
Linear search – loop body

\{(A_i \neq x) \land \text{inv}\} B \{\text{inv}\}

\[\Box \quad \{(\text{Variant bounds strengthen postcon: } \text{inv} \land (0 \leq i < i_0))\}\]

\{(A_i \neq x) \land \text{inv}\} B \{\text{inv} \land (0 \leq i < i_0)\}\n
\[\Box \quad \{(\text{Assignment: } \text{inv} \land (A_i \neq x) \Rightarrow (\text{inv} \land (0 \leq i < i_0))[i \leftarrow i - 1]\} \}

\{\text{inv} \land (A_i \neq x)\}\  i := i - 1 \{\text{inv} \land (0 \leq i < i_0)\}\
Linear search – final version

\[
\{ \text{app}(A, x, 0, A.\text{len}) \} \ \\
i := A.\text{len} - 1;
\{ \text{inv} \triangleq \neg \text{app}(A, x, i + 1, A.\text{len}) \} \ \\
\textbf{do} (A_i \neq x) \rightarrow \\
i := i - 1
\textbf{od}
\{ \{\text{inv} \land (A_i = x)\} \} 
\]
Hands-On with WebCorC

- [https://www.isf.cs.tu-bs.de/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

CbC by Example

• Simple examples
  • Linear search
  • Pattern matching

• Advanced example
  • Three Banner Flag (Dutch National Flag)

Pattern Matching – overview

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

Note: $M\.\text{len} \geq \max(0, (A\.\text{len} - p\.\text{len} + 1))$
Problem Specification: $M : [Pre, Post]$

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

$$Post \equiv \forall k \in [0, A\text{.len} - p\text{.len} + 1) :$$

$$(M_k = 1 \land A_{[k,k+p\text{.len}]} = p) \lor$$

$$(M_k = -1 \land A_{[k,k+p\text{.len}]} \neq p)$$
Pattern Matching – choosing the invariant

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

Diagram:

- \( A \)
  - Checked
  - To be checked
- \( M \)
  - 1 -1 -1 1 1 \ldots -1
  - 0 \rightarrow i \rightarrow i + p.len
- \( 0 \) to \( p + p.len \)
- \( 0 \) to \( A.len \)
- \( 0 \) to \( M.len \)
Pattern Matching – composition and establishing the invariant

\[ M : [Pre, Post] \]
\[ \equiv \{ \text{Write } Post \text{ in alternative format; introduce frame variable } i \} \]
\[ M, i : [Pre, inv(i) \land i \geq \max(0, (A.len - p.len + 1))] \]
\[ \square \{ \text{Composition} \} \]
\[ M, i : [Pre, inv(i)]; M, i : [inv(i), inv(i) \land i \geq \max(0, (A.len - p.len + 1))] \]
\[ \square \{ \text{Assignment: } inv(i)[i\backslash 0] \equiv \text{TRUE and } Pre \Rightarrow \text{TRUE } \} \]
\[ i : = 0; \]
\[ M, i : [inv(i), inv(i) \land i \geq \max(0, (A.len - p.len + 1))] \]
Pattern Matching – developing the loop

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

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

\[ \textbf{do}(i < \max(0, (A\.len - p\.len + 1))) \rightarrow \]
\[ M, i : [inv(i) \land i \neq (A\.len - p\.len + 1), \, inv(i) \land (0 \leq V(i) < V(i_0))] \]
\textbf{od}

\[ \sqsubseteq \{ \text{Decrease variant using “Following assignment rule”.} \} \]

\[ \textbf{do}(i < (A\.len - p\.len + 1)) \rightarrow \]
\[ M, i : [inv(i) \land i \neq (A\.len - p\.len + 1), \, inv(i)[i\lceil i + 1]]]; \]
\[ i : = i + 1 \]
\textbf{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.
  
  $inv(i)[i \backslash i + 1] \equiv \forall k \in [0, i + 1) : (M_k = 1 \land A_{[k,k+p.len)} = p) \lor (M_k = -1 \land A_{[k,k+p.len)} \neq p)$

- $M, i : [inv(i) \land i \neq (A.len - p.len + 1), inv(i)[i \backslash i + 1)]$
  
  $\sqsubseteq \{ \text{Use selection rule??} \}$

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

  $\not\iff (A_{[i,i+p.len)} \neq p) \rightarrow M_i : = -1$

  fi

- Not a practical solution

- Loop & if-statement to check — Self study!
Put all the pieces together:

\[
\begin{align*}
  i &:= 0; \\
  \textbf{do} & ~ i < A\. len - p\. len + 1 \rightarrow \\
  & \quad j := 0; \\
  & \quad \textbf{do} \ (j \neq p\. len \land p_j = A_{i+j}) \rightarrow j := j + 1 \ \textbf{od}; \\
  & \quad \textbf{if} \ (j = p\. len) \rightarrow M_i := 1 \ | \ (j \neq p\. len) \rightarrow M_i := -1 \ \textbf{fi}; \\
  & \quad i := i + 1 \\
  \textbf{od}
\end{align*}
\]
CbC by Example

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

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

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**

...
Sort array $A$ containing red, white & blue values in $O(N)$
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)
\[ s(A, \ell, h, wb, wt, bb) \triangleq c(A, \ell, wb, r) \wedge \\
c(A, wb, wt, w) \wedge \\
c(A, bb, h, b) \wedge \\
0 \leq \ell \leq wb \leq wt \leq bb \leq h \leq A\text{.len} \]
TBF/DNF – a possible route

\[ s(A, \ell, h, wb, wt, bb) \]

where \( \ell = 0 \land h = A \cdot \text{len} \land wt = bb \)
TBF/DNF – a better route:

\[ \ell = 0 \land h = A.\text{len} \]

\[ s(A, 0, A.\text{len}, wb, wt, bb) \]

where \( wt = bb \)
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: $s(A, wb, wt, bb) \land (wt = bb)$

Invariant: $s(A, wb, wt, bb)$
TBF/DNF – composition and establishing the invariant

\[ A, \ wb, \ wt, \ bb : [Pre, \ Post] \]
\[ \equiv \ \{ \text{Substitute. (Drop frame variables)}\} \]
\[ [s(A, 0, 0, A\.len), s(A, wb, wt, bb) \land (wt = bb)] \]
\[ \sqsubset \ \{ \text{Sequence to bring in inv}\} \]
\[ [s(A, 0, 0, A\.len), s(A, wb, wt, bb)] ; \]
\[ [s(A, wb, wt, bb), s(A, wb, wt, bb) \land (wt = bb)] \]
\[ \sqsubset \ \{ \text{Assignment:} \ pre \ \Rightarrow \ inv[wb, wt, bb \ \backslash \ 0, 0, A\.len]\} \]
\[ wb, wt, bb := 0, 0, A\.len ; \]
\[ [s(A, wb, wt, bb), s(A, wb, wt, bb) \land (wt = bb)] \]
TBF/DNF – developing the loop

Focus on deriving the loop:

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

\[\subseteq \{\text{Repetition. Variant is } bb - wt\}\]

\[\textbf{do } (wt \neq bb) \rightarrow\]

\[[(wt \neq bb) \land s(A, wb, wt, bb),\]

\[\quad s(A, wb, wt, bb) \land 0 \leq (bb - wt) < (bb_0 - wt_0)]\]

\[\textbf{od}\]
Use \textit{inv} and \textit{var} for invariant and variant predicates

\begin{align*}
A, wb, wt, bb : [ & \text{inv} \land (wt \neq bb), \text{inv} \land \text{var}] \\
\sqsubseteq & \{\text{selection: pre} \Rightarrow (A_{wt} = r) \lor (A_{wt} = w) \lor (A_{wt} = b) \equiv \text{TRUE}\} \\
\text{if} & (A_{wt} = r) \rightarrow [\text{inv} \land (wt \neq bb) \land (A_{wt} = r), \text{inv} \land \text{var}] \\
\quad & (A_{wt} = w) \rightarrow [\text{inv} \land (wt \neq bb) \land (A_{wt} = w), \text{inv} \land \text{var}] \\
\quad & (A_{wt} = b) \rightarrow [\text{inv} \land (wt \neq bb) \land (A_{wt} = b), \text{inv} \land \text{var}] \\
\text{fi} \\
\sqsubseteq & \{\text{assignment rule}\}
\end{align*}
TBF/DNF – developing the loop (3)

```
if (Awt = r) → Awt, Awb, wt, wb := Awb, Awt, wt + 1, wb + 1
  (Awt = w) → wt := wt + 1
  (Awt = b) → Awt, Abb−1, bb := Abb−1, Awt, bb − 1
fi
```
TBF/DNF – final algorithm

\[
\begin{align*}
\text{wb, wt, bb} &:= 0, 0, A.\text{len}; \\
\text{do } (\text{wt} \neq \text{bb}) &\rightarrow \\
\quad \text{if } (A_{\text{wt}} = r) &\rightarrow A_{\text{wt}}, A_{\text{wb}}, \text{wt}, \text{wb} := A_{\text{wb}}, A_{\text{wt}}, \text{wt} + 1, \text{wb} + 1 \\
\quad &\quad | (A_{\text{wt}} = w) \rightarrow \text{wt} := \text{wt} + 1 \\
\quad &\quad | (A_{\text{wt}} = b) \rightarrow A_{\text{wt}}, A_{\text{bb}-1}, \text{bb} := A_{\text{bb}-1}, A_{\text{wt}}, \text{bb} - 1 \\
\quad \text{fi} \\
\text{od}
\end{align*}
\]
CbC by Example

• Simple examples
  • Linear search
  • Pattern matching

• Advanced example
  • Three Banner Flag (Dutch National Flag)

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?


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)

---


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):**

- Cosmic Top Secret
- Secret
- Confidential
- Restricted
- Unclassified
- Untainted
- Tainted
Language-based Security

```java
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.]

\[
\begin{align*}
\vdash \text{exp} : \text{high} & \quad h \notin \text{Vars} (\text{exp}) \\
\vdash \text{exp} : \text{low} & \quad [\text{ct}] \vdash \text{skip} \\
\vdash \text{exp} : \text{low} & \quad [\text{ct}] \vdash S_1 \quad [\text{ct}] \vdash S_2 \\
[\text{low}] \vdash l = \text{exp} & \quad [\text{ct}] \vdash S_1; S_2 \\
\vdash \text{exp} : \text{ct} & \quad [\text{ct}] \vdash S_1 \quad [\text{ct}] \vdash S_2 \\
& \quad [\text{ct}] \vdash \text{if} \: \text{exp} \: \text{then} \: S_1 \: \text{else} \: S_2
\end{align*}
\]
Information Flow Control-by-Construction by Example

1. **pre:** publishBid = 0
2. **post:** \(\forall x \in \text{bids} \mid (x \geq 0 \land x < \text{bids.length}) \rightarrow (\text{publishBid} \geq \text{bids}[x])\)
3. ```java
   void auction(private int[] bids,
                public int publishBid) {
   public int i = 0;
   secret int highestBid = 0;
   do (i < bids.length) {
      if (highestBid < bids[i]) {
         highestBid = bids[i];
      } else {
         skip
      }
      i = i + 1;
   } od
   publishBid = declassify(highestBid);
   }
```
Incremental Construction of the Algorithm

```java
1  pre: publishBid = 0
2  post: \forall int x; ((x >= 0
3      & x < bids.length)
4      -> (publishBid >= bids[x]))
5
6  void auction(private int[] bids, 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).**

$$\{V^{pre}\} S \{V^{post}\}[\eta] \text{ is refifiable to } \{V^{pre}\} x := E \{V^{post}\}[\eta] \iff\]$$

$$V^{post}(y) = V^{pre}(y) \text{ for all } y \in Vars \setminus \{x\}, \text{ and } V^{post}(x) = lub(\{V^{pre}(v) \mid v \in \text{vars}(E)\} \cup \{V^{pre}(x), \eta\}).$$
Refinement Rules – Repetition

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

\[
\text{RULE 5 (REPETITION).}
\{V^{\text{pre}}\} S \{V^{\text{post}}\}[\eta] \text{ is refinable to } \{V^{\text{pre}}\} \text{ do } G \rightarrow S1 \text{ od } \{V^{\text{post}}\}\[\eta]\text{ iff } \{V^{\text{pre}}\} S1 \{V^{\text{post}}\}[\eta'] \text{ with } \eta' = \text{lub}(\{V^{\text{pre}}(v) \mid v \in \text{vars}(G)\} \cup \{\eta\})
\]
Refinement Rules – Method Call

• Call of other secure programs

• Similar to substitution of method body

Rule 6 (Method Call).

\{V^{pre}\}_S \{V^{post}\}_\eta \text{ is refinable to } \{V^{pre}\}_M(a_1, \ldots, a_n) \{V^{post}\}_\eta \text{ iff for a method } \{V^{pre}_{\text{call}}\}_M(z_1, \ldots, z_n) \{V^{post}_{\text{call}}\}_\eta \text{ and for all parameters: } V^{pre}_{\text{call}}(a_i) \leq V^{pre}_{\text{call}}(z_i) \land V^{post}_{\text{call}}(z_i) \leq V^{post}_{\text{call}}(a_i) \text{ where } a_i \text{ are the actual parameters and } z_i \text{ 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
    
    ```java
    low Object o = new Object();
    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

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

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.


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


