Confidentiality in *Circus*

Based on the work of Michael J. Banks

Jeremy L. Jacob

Department of Computer Science, The University of York

23 January 2013

Headlines

- Integrated notation to express confidentiality & functionality. (Functionality is a lower limit on information flow; confidentiality is an upper limit.)
- Inconsistency handled by miracles.
- Single notion of refinement.

Secondary headlines

- Confidentiality annotations integrated into *Circus*.
- Semantics in Hoare & He's UTP.
- Semantics based on standard *Circus* semantics.

Intellectual ancestry

A very partial list

Confidentiality	Goguen & Meseguer; Jacob; Morgan; Mantel.
Functionality	Dijkstra; Morgan; Hoare & He; Woodcock, Cavalcanti, Oliveira.

Trivial example

A one-shot buffer that protects the parity of the input

Key: original *Circus*; addition to *Circus*.

channel $h, \ell : \mathbb{N}$
process $CMD \mathrel{\hat=}$ **begin**
state $S \mathrel{\hat=} [v : \mathbb{N}]$
$Init \mathrel{\hat=} [S' \mid v' = 0]$ — initialisation
$C \mathrel{\hat=} \langle \{\ell\} : v \in Odd \iff \tilde{v} \notin Odd \rangle$ — confidentiality annotation
$H \mathrel{\hat=} h?n \rightarrow v := n?$ — action
$L \mathrel{\hat=} \ell!2 * (v \div 2) \rightarrow Stop$ — action
$\bullet \langle \{\ell\} : Init; H \rangle; C; \langle \{\ell\} : L \rangle$ — behaviour
end

C: “If $v \in Odd$ then communication on channels in $\{\ell\}$ cannot allow $v \notin Odd$ to be ruled out, and *vice versa*.”

$\{\ell\}$ ‘sees program counter’ between blocks and own actions within blocks.

Trivial example

A one-shot buffer that simultaneously leaks and protects the parity of the input

Key: original *Circus*; addition to *Circus*.

channel $h, \ell : \mathbb{N}$
process $CMD \mathrel{\widehat{=}}$ **begin**
state $S \mathrel{\widehat{=}} [v : \mathbb{N}]$
$Init \mathrel{\widehat{=}} [S' \mid v' = 0]$ — initialisation
$C \mathrel{\widehat{=}} \langle \{\ell\} : v \in Odd \iff \tilde{v} \notin Odd \rangle$ — confidentiality annotation
$H \mathrel{\widehat{=}} h?n \rightarrow v := n?$ — action
$L \mathrel{\widehat{=}} \ell!v \rightarrow Stop$ — action, inconsistent with C
$\bullet \langle \{\ell\} : Init; H \rangle; C; \langle \{\ell\} : L \rangle$ — behaviour
end

This specification is only implementable by a miracle, because it encodes inconsistent requirements.

An analogy for miracles

Consider $\mathbb{N}$-valued expressions composed of $\mathbb{N}$ constants, addition and partial subtraction.

An instance such as $(4 - 5) + 7$ has a value, but contains an undefined term.

Three possibilities:

- Do nothing.

  $(4 - 5) + 7$ is lost to us as a legal expression.

- Invent theory of rearrangement.

  Separate into added and subtracted terms; sum each; then (x, y) with $x < y$ is inconsistent, otherwise compute $x - y$.

  $(4 - 5) + 7 \rightsquigarrow (4 + 7, 5) \rightsquigarrow (11, 5) \rightsquigarrow 6$.

- Totalise.

  Introduce special values to represent the result of subtracting a bigger number from a smaller number. If final result is -ve, then inconsistent, otherwise OK.

  $(4 - 5) + 7 = (-1) + 7 = 6$.

Extending programs with miracles

- Least upper bound $P \sqcup Q$ of specifications is partial.
- Miracles are a way of totalising.
- Lattice of implementable specifications mirrored to add ‘upper half’.
- The name of the (alas, impossible to implement) perfect program is $\top$; mirrors the (alas, easily implementable) worst specification, $\bot$.
- “Naked guarded command”:
  $wp(g \textbf{ guard } S, P) \iff (g \implies wp(S, P))$.
  Miraculous if guard g false. $\top = \textsf{false} \to S$, for any statement S.
- $\textbf{if } x < y \textbf{ then } (x \leq y \textbf{ guard } w := 0) \textbf{ else } w := 1 \textbf{ fi}$ same as $\textbf{if } x < y \textbf{ then } w := 0 \textbf{ else } w := 1 \textbf{ fi}$ — OK.
- $\textbf{if } x > y \textbf{ then } (x \leq y \textbf{ guard } w := 0) \textbf{ else } w := 1 \textbf{ fi}$ same as $\textbf{if } x > y \textbf{ then } \top \textbf{ else } w := 1 \textbf{ fi}$ — KO.

Circus

- A specification language: control flow described in CSP; state transitions in Z.
- Semantics in UTP.
- Refinement calculus.

Example: one-shot buffer that zeroes the lowest bit.

channel $h, \ell : \mathbb{N}$	— declarations
process $CMD \mathrel{\widehat=}$ **begin**	
state $S \mathrel{\widehat=} [v : \mathbb{N}]$	— state components (Z)
$Init \mathrel{\widehat=} [S' \mid v' = 0]$	— initialisation (Z)
$H \mathrel{\widehat=} h?n \rightarrow [\Delta S; n? : \mathbb{N} \mid v' = n?]$	— piece of behaviour (CSP/Z)
$L \mathrel{\widehat=} \ell!2 * (v \div 2) \rightarrow Stop$	— piece of behaviour (CSP)
$\bullet\ Init; H; L$	— overall behaviour (CSP)
end	

UTP

A semantic framework

- UTP uses alphabetised predicates to describe things.
  Particles have a mass-at-rest, m, and an energy equivalent, e.
  Sequential programs have value-before x and value-after x' for each program variable x; and also a special pair ok/ok' to record program started/finished.
- Classes of predicates described by "healthiness conditions", couched as fixed-point constructions of predicate transformers.
  (Predicates describing) particles, P, satisfy $P \iff (P \wedge e = m.c^2)$.
  (Predicates describing) sequential programs, P, satisfy, among others, healthiness condition 'H1': $P \iff (ok \implies P)$ ("values on termination not predicted until programme starts").

Circus semantics

- For each program variable x, a pair of variables x, x' in alphabet.
- Special variables ok, ok', $wait$, $wait'$, tr, tr', ref, ref' in alphabet.
- Predicates that describe *Circus* actions must satisfy the healthiness conditions for reactive systems, further restricted by ‘CSP’ conditions.
- The refinement lattice is standard for the UTP: $P \sqsubseteq Q \iff [Q \implies P]$.

Example operators

- Nondeterministic choice:
  $P \sqcap Q \iff P \vee Q$
- A coercion, $[\mathit{predicate}]$, is a special annotation:
  $[P] \iff \mathit{Skip} \lhd P \rhd \textbf{miracle}$

$$
\begin{aligned}
h := 0 \sqcap h := 1; [h = 0] &\equiv h := 0; [h = 0] \sqcap h := 1; [h = 0] \\
&\equiv h := 0 \sqcap \textbf{miracle} \\
&\equiv h := 0
\end{aligned}
$$

Confidentiality annotations and blocks

Additions to *Circus* for specifying confidentiality

CAs

Syntax $\langle \mathit{channelset} : \mathit{predicate} \rangle$

Intuition The ‘fogging’ captured in the predicate must be enforced throughout the process about the current state.

Example $\langle \{\ell\} : v \in \mathit{Odd} \iff \tilde{v} \notin \mathit{Odd} \rangle$

Blocks

Syntax $\langle \mathit{channelset} : \mathit{action} \rangle$

Intuition Region in which location of ‘program counter’ cannot be deduced by viewing communications on *channelset*.

Example $\langle \{\ell\} : \mathit{Init}; H \rangle$

Confidentiality predicates

- Relate values of semantic variables with non-repudiable values.
- May refer to program variables (x), plus a 'fog' copy ($\tilde{x}$).
- The value in the semantic variable is the actual value; the possible values of the fog variables may not be ruled out as the actual value by communications on the channels in $channelset = Low$.
- Examples:
  1. $v \in Odd \implies \tilde{v} \notin Odd$ "if v odd, then a correct implementation must not allow Low to rule out v even (but may allow Low to be sure that v is even)".
  2. $v \in Odd \iff \tilde{v} \notin Odd$ "Low may not know the parity of v."
  3. $a < b \implies \tilde{a} \geq \tilde{b}$ "Low may not be sure that a is below b."
  4. $x + y > 99 \implies \tilde{x} + \tilde{y} \leq 99$ "the state summing to more than 99 is secret."

Extended semantics

- Add a copy of the action to record desired explanations for observations: $\mathbf{U}A \mathrel{\widehat{=}} A \wedge A[\tilde{\mathbf{v}}/\mathbf{v}]$.
  Variables in copy indicated by a tilde.
  ($A[\tilde{\mathbf{v}}/\mathbf{v}]$ is predicate A but with each variable decorated by a tilde.)
- Synchronise each copy on $\mathcal{L}$'s interface: $\mathbf{UC}(\mathcal{L}, A)) \mathrel{\widehat{=}} \mathbf{U}A \wedge \mathbf{I}\mathcal{L}$ where

$$\begin{aligned}\mathbf{I}\mathcal{L} \mathrel{\widehat{=}} & ok = \tilde{ok} \wedge ok' = \tilde{ok}' \wedge wait = \tilde{wait} \wedge wait' = \tilde{wait}' \\ & \wedge (tr' - tr) \upharpoonright \mathcal{L} = (\tilde{tr}' - \tilde{tr}) \upharpoonright \mathcal{L} \\ & \wedge wait' \implies ref' \cap \mathcal{L} = \tilde{ref}' \cap \mathcal{L}\end{aligned}$$

- Each *Circus* operator, $\_ \oplus \_$ lifted to an operator, $\_\hat{\oplus}\_$, in extended space.

Semantics of CAs and Blocks

- Blocks:
  $\langle \mathcal{L} : A \rangle \widehat{=} \mathbf{UC}(\mathcal{L}, A)$
  Apply ordinary *Circus* rules, then apply **UC**.
  Atomic constructs refined by composite: $\langle A \oplus B \rangle \sqsubseteq \langle A \rangle \hat{\oplus} \langle B \rangle$
- CAs:
  $\langle \mathcal{L} : P \rangle \widehat{=} \mathbf{UC}(\mathcal{L}, Skip) \wedge (ok \wedge \neg wait \implies P)$
  CAs generalise the notion of coercions in extended space.
  Main purpose is to desynchronise the two copies of A. (Extra synchronisation can be added to record extra information known to $\mathcal{L}$.)

Refinement and Verification

- Proof technique for verifying miraculousness: backward propagation of CAs [c.f weakest precondition].
- Refinement same definition as other UTP-based theories: $P \sqsubseteq Q \mathrel{\widehat{=}} [Q \implies P]$.
- Inconsistent refinement (removing too much fog) gives a miraculous action: avoids so-called ‘refinement paradox’.

Future work

- Idioms for writing confidentiality annotations.
- Pragmatics of refinement and verification, especially route to code.
- Tools.