Safety-critical Software Development Tools Implementation Using an Applicative Language

Abstract
Safety-critical software have mostly been developed using subsets of C or Ada (SPARK). As software become more complex, tools are necessary to help reducing the development cost and enhance software reliability.
In safety-critical domains, "development tools" (which may introduce bugs if they are wrong) are submitted to a similar process as for embedded programs.
Using a high-level applicative language to develop such tools means having to interpret the norms or guides for such languages while they have been written with simple imperative languages in mind.

We present in this poster [1]
- a grammar of a (rather rich) mini ML
- inductive definitions for "condition/decision"-related code coverage criteria
- a code coverage report generated by Zamcov [5], a tool for OCaml [6] code coverage.

Safety-critical software development
- guides/norms: **DO-178B [2]**, EN 50128, EN 50129, IEC-61508
  - everything is ruled and must always be verified
  - conformity assessment by independent authority
- rather conservative domains, mostly using C or SPARK

How to reduce costs and have better reliability?
- Use tools to enhance software!

Two categories of tools
- development tools: may introduce bugs (*e.g.*, compilers)
- verification tools: cannot introduce bugs

Developments tools
- similar certification process as for embedded software
- but different approach (*e.g.*, dynamic memory allocations are fine)

An applicative language for development tools
- very well-suited to develop compilers
- but richer-than-usual runtime library
  - provide rarely existent documentation
- have to interpret the guides/norms for such languages
  - unclear code coverage criteria semantics
- must provide tools for code coverage measurement

Mini ML

e ::=	`x`	variable
|	`c`	constant
|	`C`	constant constructor
|	`C(e)`	constructor with an argument
|	$e_1\ e_2$	application
|	`fun x -> e`	abstraction
|	`let x = e1 in e2`	let-in
|	`let rec` x_1 `= fun` x_2 `->` e_1 `in` e_2	rec-in
|	e_1 `;` e_2	sequence
|	`{` f_1 `=` e_1 `;` ... `;` f_n `=` e_n `}`	record
|	`e.f`	access
|	e_1`.f <-` e_2	modify
|	`for x = e1 to e2 do e3 done`	for
|	`while e1 do e2 done`	while
|	`if e1 then e2 else e3`	conditional
|	e_1 `&&` e_2	sequential conjunction
|	e_1 `||` e_2	sequential disjunction
|	`not e`	negation
|	`(e)`	parentheses
|	`match e with` p_1 `->` e_1 `|` ... `|` p_n `->` e_n	match
|	`raise e`	raise
|	`try` e_1 `with x ->` e_2	try

Classical structural code coverage criteria
- statement coverage (SC)
  - which statements have been activated
- condition coverage (CC)
  - which conditions have been both T and F
- decision coverage (DC)
  - which decisions have been both T and F
- condition/decision coverage (C/DC)
  - union of CC and DC
- multiple condition coverage (MCC)
  - which decisions for which conditions have taken all possible Boolean configurations
- modified condition/decision coverage (MC/DC) [3]
  - C/DC with the constraint that each condition has been shown to **independently** (from other conditions) affect the result of the decision

Condition and decision
- condition: a Boolean expression containing no Boolean operator
- decision: a Boolean expression containing zero or more Boolean operators

What is a Boolean operator?
- conjunction (&&) and disjunction (||) operators
- but not only: by essence, it depends on the language (and on coding rules)

A semantics for conditions and decisions (1/3)

Not-in-a-decision Boolean expressions.
$\mathcal{C}_1(\emptyset, \texttt{true : bool}) = []$
$\mathcal{C}_1(\emptyset, \texttt{false : bool}) = []$
$\mathcal{C}_1(\emptyset, \texttt{e : bool}) =$ coding-rule/error : not-in-a-decision

Not-in-a-decision non-Boolean expressions.
$\mathcal{C}_1(\emptyset, \texttt{while } e_1 \texttt{ do } e_2 \texttt{ done}) = \mathcal{C}_1(d, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3) = \mathcal{C}_1(d, e_1) \cup \mathcal{C}_1(\emptyset, e_2) \cup \mathcal{C}_1(\emptyset, e_3)$
d is a fresh variable representing a new decision.
$\mathcal{C}_1(\emptyset, c) = []$
$\mathcal{C}_1(\emptyset, x) = []$
$\mathcal{C}_1(\emptyset, e_1\ e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{fun x -> e}) = \mathcal{C}_1(\emptyset, e)$
$\mathcal{C}_1(\emptyset, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{let x = } e_1 \texttt{ in } e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, e_1\ ;\ e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{C(e)}) = \mathcal{C}_1(\emptyset, e)$
$\mathcal{C}_1(\emptyset, \{\ f_1 = e_1\ ;\ \ldots\ ;\ f_n = e_n\ \}) = \mathcal{C}_1(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_1(\emptyset, e_n)$
$\mathcal{C}_1(\emptyset, e\ .\ f) = \mathcal{C}_1(\emptyset, e)$
$\mathcal{C}_1(\emptyset, e_1\ .\ f\ \texttt{<-}\ e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{for x = } e_1 \texttt{ to } e_2 \texttt{ do } e_3 \texttt{ done}) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2) \cup \mathcal{C}_1(\emptyset, e_3)$
$\mathcal{C}_1(\emptyset, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n) = \mathcal{C}_1(\emptyset, e) \cup \mathcal{C}_1(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_1(\emptyset, e_n)$
$\mathcal{C}_1(\emptyset, \texttt{raise e}) = \mathcal{C}_1(\emptyset, e)$
$\mathcal{C}_1(\emptyset, \texttt{try } e_1 \texttt{ with x -> } e_2) = \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(\emptyset, \texttt{(e)}) = \mathcal{C}_1(\emptyset, e)$

In-a-decision expressions.
$\mathcal{C}_1(d, \texttt{true : bool}) = []$
$\mathcal{C}_1(d, \texttt{false : bool}) = []$
$\mathcal{C}_1(d, \texttt{x : bool}) = \{(\ulcorner \texttt{x} \urcorner, d \to c)\}$
$\mathcal{C}_1(d, e_1\ e_2 \texttt{ : bool}) = \{(\ulcorner e_1\ e_2 \urcorner, d \to c)\} \cup \mathcal{C}_1(\emptyset, e_1) \cup \mathcal{C}_1(\emptyset, e_2)$
$\mathcal{C}_1(d, e\ .\ f \texttt{ : bool}) = \{(\ulcorner e\ .\ f \urcorner, d \to c)\} \cup \mathcal{C}_1(\emptyset, e)$
$\mathcal{C}_1(d, e_1 \texttt{ \&\& } e_2 \texttt{ : bool}) = \mathcal{C}_1(d, e_1) \cup \mathcal{C}_1(d, e_2)$
$\mathcal{C}_1(d, e_1 \texttt{ || } e_2 \texttt{ : bool}) = \mathcal{C}_1(d, e_1) \cup \mathcal{C}_1(d, e_2)$
$\mathcal{C}_1(d, \texttt{not e : bool}) = \mathcal{C}_1(d, e)$
$\mathcal{C}_1(d, \texttt{(e) : bool}) = \mathcal{C}_1(d, e)$

Other rejected expressions.
$\mathcal{C}_1(d, e_1\ ;\ e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{let x = } e_1 \texttt{ in } e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{raise e : bool}) =$ coding-rule/error
$\mathcal{C}_1(d, \texttt{try } e_1 \texttt{ with x -> } e_2 \texttt{ : bool}) =$ coding-rule/error

The C_1 semantics makes our mini ML more similar to classical subsets of C or Ada by forbidding all "exotic" expressions.

The C_2 semantics allows decisions to appear anywhere the language allows, but conditions still have to be not too "exotic".

The C_3 semantics allows the full language to be used by replacing coding rules with additional measurement obligations.

new decisions

A semantics for conditions and decisions (2/3)

Not-in-a-decision Boolean expressions.
$\mathcal{C}_2(\emptyset, \texttt{true : bool}) = []$
$\mathcal{C}_2(\emptyset, \texttt{false : bool}) = []$
$\mathcal{C}_2(\emptyset, \texttt{x : bool}) = \{(\ulcorner \texttt{x} \urcorner, d)\}$
$\mathcal{C}_2(\emptyset, e\ .\ f \texttt{ : bool}) = \{(\ulcorner e\ .\ f \urcorner, d)\} \cup \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, e_1\ e_2 \texttt{ : bool}) = \{(\ulcorner e_1\ e_2 \urcorner, d)\} \cup \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, e_1 \texttt{ \&\& } e_2 \texttt{ : bool}) = \{(\ulcorner e_1 \texttt{ \&\& } e_2 \urcorner, d)\} \cup \mathcal{C}_2(d, e_1) \cup \mathcal{C}_2(d, e_2)$
$\mathcal{C}_2(\emptyset, e_1 \texttt{ || } e_2 \texttt{ : bool}) = \{(\ulcorner e_1 \texttt{ || } e_2 \urcorner, d)\} \cup \mathcal{C}_2(d, e_1) \cup \mathcal{C}_2(d, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{not e : bool}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \texttt{(e) : bool}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, e_1\ ;\ e_2 \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{let x = } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2) \cup \mathcal{C}_2(\emptyset, e_3)$
$\mathcal{C}_2(\emptyset, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e) \cup \mathcal{C}_2(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_2(\emptyset, e_n)$
$\mathcal{C}_2(\emptyset, \texttt{raise e : bool}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \texttt{try } e_1 \texttt{ with x -> } e_2 \texttt{ : bool}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$

Not-in-a-decision non-Boolean expressions.
$\mathcal{C}_2(\emptyset, c) = []$
$\mathcal{C}_2(\emptyset, x) = []$
$\mathcal{C}_2(\emptyset, e\ .\ f) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, e_1\ .\ f\ \texttt{<-}\ e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, e_1\ e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{(e)}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \texttt{C(e)}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \{\ f_1 = e_1\ ;\ \ldots\ ;\ f_n = e_n\ \}) = \mathcal{C}_2(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_2(\emptyset, e_n)$
$\mathcal{C}_2(\emptyset, e_1\ ;\ e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{let x = } e_1 \texttt{ in } e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{fun x -> e}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2) \cup \mathcal{C}_2(\emptyset, e_3)$
$\mathcal{C}_2(\emptyset, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n) = \mathcal{C}_2(\emptyset, e) \cup \mathcal{C}_2(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_2(\emptyset, e_n)$
$\mathcal{C}_2(\emptyset, \texttt{raise e}) = \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(\emptyset, \texttt{try } e_1 \texttt{ with x -> } e_2) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(\emptyset, \texttt{for x = } e_1 \texttt{ to } e_2 \texttt{ do } e_3 \texttt{ done}) = \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2) \cup \mathcal{C}_2(\emptyset, e_3)$

In-a-decision Boolean expressions.
$\mathcal{C}_2(d, \texttt{true : bool}) = []$
$\mathcal{C}_2(d, \texttt{false : bool}) = []$
$\mathcal{C}_2(d, \texttt{x : bool}) = \{(\ulcorner \texttt{x} \urcorner, d \to c)\}$
$\mathcal{C}_2(d, e\ .\ f \texttt{ : bool}) = \{(\ulcorner e\ .\ f \urcorner, d \to c)\} \cup \mathcal{C}_2(\emptyset, e)$
$\mathcal{C}_2(d, e_1\ e_2 \texttt{ : bool}) = \{(\ulcorner e_1\ e_2 \urcorner, d \to c)\} \cup \mathcal{C}_2(\emptyset, e_1) \cup \mathcal{C}_2(\emptyset, e_2)$
$\mathcal{C}_2(d, e_1 \texttt{ \&\& } e_2 \texttt{ : bool}) = \mathcal{C}_2(d, e_1) \cup \mathcal{C}_2(d, e_2)$
$\mathcal{C}_2(d, e_1 \texttt{ || } e_2 \texttt{ : bool}) = \mathcal{C}_2(d, e_1) \cup \mathcal{C}_2(d, e_2)$
$\mathcal{C}_2(d, \texttt{not e : bool}) = \mathcal{C}_2(d, e)$
$\mathcal{C}_2(d, \texttt{(e) : bool}) = \mathcal{C}_2(d, e)$
$\mathcal{C}_2(d, e_1\ ;\ e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{let x = } e_1 \texttt{ in } e_2 \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n \texttt{ : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{raise e : bool}) =$ coding-rule/error
$\mathcal{C}_2(d, \texttt{try } e_1 \texttt{ with x -> } e_2 \texttt{ : bool}) =$ coding-rule/error

new decisions

A semantics for conditions and decisions (3/3)

Not-in-a-decision Boolean expressions.
$\mathcal{C}_4(\emptyset, \texttt{x : bool}) = \{(\ulcorner \texttt{x} \urcorner, d)\} \cup []$
$\mathcal{C}_4(\emptyset, \texttt{c : bool}) = []$
$\mathcal{C}_4(\emptyset, e_1\ e_2 \texttt{ : bool}) = \{(\ulcorner e_1\ e_2 \urcorner, d)\} \cup \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ : bool}) = \{(\ulcorner \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \urcorner, d)\} \cup \mathcal{C}_4(d, e_2) \cup \mathcal{C}_4(d, e_3)$
$\mathcal{C}_4(\emptyset, \texttt{let x = } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, e_1\ ;\ e_2 \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, e\ .\ f \texttt{ : bool}) = \{(\ulcorner e\ .\ f \urcorner, d)\} \cup \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, e_1 \texttt{ \&\& } e_2 \texttt{ : bool}) = \{(\ulcorner e_1 \texttt{ \&\& } e_2 \urcorner, d)\} \cup \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(\emptyset, e_1 \texttt{ || } e_2 \texttt{ : bool}) = \{(\ulcorner e_1 \texttt{ || } e_2 \urcorner, d)\} \cup \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{not e : bool}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{(e) : bool}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e) \cup \mathcal{C}_4(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_4(\emptyset, e_n)$
$\mathcal{C}_4(\emptyset, \texttt{raise e : bool}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{try } e_1 \texttt{ with x -> } e_2 \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$

Not-in-a-decision non-Boolean expressions.
$\mathcal{C}_4(\emptyset, x) = []$
$\mathcal{C}_4(\emptyset, c) = []$
$\mathcal{C}_4(\emptyset, e_1\ e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{fun x -> e}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2) \cup \mathcal{C}_4(\emptyset, e_3)$
$\mathcal{C}_4(\emptyset, \texttt{let x = } e_1 \texttt{ in } e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{C(e)}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \{\ f_1 = e_1\ ;\ \ldots\ ;\ f_n = e_n\ \}) = \mathcal{C}_4(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_4(\emptyset, e_n)$
$\mathcal{C}_4(\emptyset, e\ .\ f) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, e_1\ .\ f\ \texttt{<-}\ e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{for x = } e_1 \texttt{ to } e_2 \texttt{ do } e_3 \texttt{ done}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2) \cup \mathcal{C}_4(\emptyset, e_3)$
$\mathcal{C}_4(\emptyset, \texttt{while } e_1 \texttt{ do } e_2 \texttt{ done}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(\emptyset, \texttt{(e)}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n) = \mathcal{C}_4(\emptyset, e) \cup \mathcal{C}_4(\emptyset, e_1) \cup \ldots \cup \mathcal{C}_4(\emptyset, e_n)$
$\mathcal{C}_4(\emptyset, \texttt{raise e}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(\emptyset, \texttt{try } e_1 \texttt{ with x -> } e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$

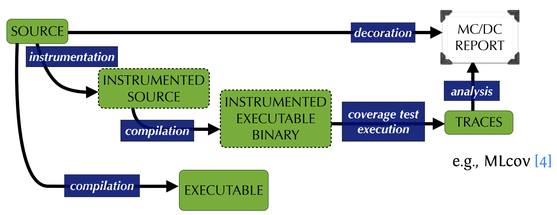
In-a-decision Boolean expressions.
$\mathcal{C}_4(d, \texttt{x : bool}) = \{(\ulcorner \texttt{x} \urcorner, d \to c)\} \cup []$
$\mathcal{C}_4(d, \texttt{c : bool}) = []$
$\mathcal{C}_4(d, e_1\ e_2 \texttt{ : bool}) = \{(\ulcorner e_1\ e_2 \urcorner, d \to c)\} \cup \{(\ulcorner e_1\ e_2 \urcorner, d_2)\} \cup \mathcal{C}_4(d_2, e_1) \cup \mathcal{C}_4(d_2, e_2)$
$\mathcal{C}_4(d, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2) \cup \mathcal{C}_4(d, e_3)$
$\mathcal{C}_4(d, \texttt{let x = } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2 \texttt{ : bool}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, e_1\ ;\ e_2 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, e\ .\ f \texttt{ : bool}) = \{(\ulcorner e\ .\ f \urcorner, d \to c)\} \cup \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, e_1 \texttt{ \&\& } e_2 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, e_1 \texttt{ || } e_2 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{not e : bool}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, \texttt{(e) : bool}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n \texttt{ : bool}) = \mathcal{C}_4(d, e) \cup \mathcal{C}_4(d, e_1) \cup \ldots \cup \mathcal{C}_4(d, e_n)$
$\mathcal{C}_4(d, \texttt{raise e : bool}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, \texttt{try } e_1 \texttt{ with x -> } e_2 \texttt{ : bool}) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$

In-a-decision non-Boolean expressions.
$\mathcal{C}_4(d, x) = []$
$\mathcal{C}_4(d, c) = []$
$\mathcal{C}_4(d, e_1\ e_2) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{fun x -> e}) = \mathcal{C}_4(\emptyset, e)$
$\mathcal{C}_4(d, \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2) \cup \mathcal{C}_4(d, e_3)$
$\mathcal{C}_4(d, \texttt{let x = } e_1 \texttt{ in } e_2) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{let rec } x_1 \texttt{ = fun } x_2 \texttt{ -> } e_1 \texttt{ in } e_2) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{C(e)}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, e_1\ ;\ e_2) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \{\ f_1 = e_1\ ;\ \ldots\ ;\ f_n = e_n\ \}) = \mathcal{C}_4(d, e_1) \cup \ldots \cup \mathcal{C}_4(d, e_n)$
$\mathcal{C}_4(d, e\ .\ f) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, e_1\ .\ f\ \texttt{<-}\ e_2) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$
$\mathcal{C}_4(d, \texttt{for x = } e_1 \texttt{ to } e_2 \texttt{ do } e_3 \texttt{ done}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2) \cup \mathcal{C}_4(\emptyset, e_3)$
$\mathcal{C}_4(d, \texttt{while } e_1 \texttt{ do } e_2 \texttt{ done}) = \mathcal{C}_4(\emptyset, e_1) \cup \mathcal{C}_4(\emptyset, e_2)$
$\mathcal{C}_4(d, \texttt{(e)}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, \texttt{match e with } p_1 \texttt{ -> } e_1 \texttt{ | } \ldots \texttt{ | } p_n \texttt{ -> } e_n) = \mathcal{C}_4(d, e) \cup \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_n)$
$\mathcal{C}_4(d, \texttt{raise e}) = \mathcal{C}_4(d, e)$
$\mathcal{C}_4(d, \texttt{try } e_1 \texttt{ with x -> } e_2) = \mathcal{C}_4(d, e_1) \cup \mathcal{C}_4(d, e_2)$

Source code instrumentation approach
- rewrites the source code for the instrumented programme to generate execution traces.

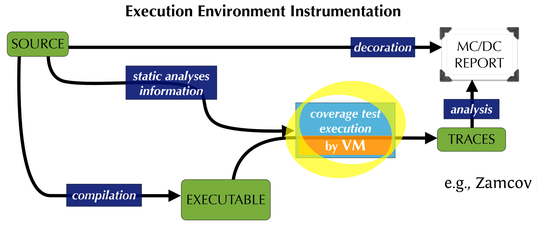
Execution environment instrumentation approach
- uses source code to machine code coverage-specific information at runtime to generate execution traces.

Source Code Instrumentation



e.g., MLcov [4]

Execution Environment Instrumentation



e.g., Zamcov [5]

An example of a code coverage measurement report using Zamcov

ZamCov Expression Coverage for scalene.ml

MC/DC measurement : scalene.ml.mcdc

```
1  let all_different a b c =
2    a <> b && b <> c && c <> a
3
4  let all_positive a b c =
5    a > 0 && b > 0 && c > 0
6
7  let is_triangle a b c =
8    all_positive a b c
9    && a + b > c
10   && a + c > b
11   && b + c > a
12
13 let is_scalene a b c =
14   is_triangle a b c
15   && all_different a b c
```

ZamCov: MC/DC tabulars (scalene.ml)

line 2: a <> b && b <> c && c <> a	a <> b	b <> c	c <> a
T : 4	T	T	T
F : 2	F	–	–
F : 1	T	T	F
F : 1	T	F	–

unreachable conditions

line 5: a > 0 && b > 0 && c > 0	a > 0	b > 0	c > 0
T : 8	T	T	T

line 8: all_positive a b c && a + b > c && a + c > b && b + c > a	all_positive a b c	a + b > c	a + c > b	b + c > a
T : 8	T	T	T	T

line 14: is_triangle a b c && all_different a b c	is_triangle a b c	all_different a b c
T : 4	T	T
F : 4	T	F

Legend (color information)

```
I'm a covered decision, I'm underlined in green.
I'm a not-covered decision, I'm underlined in red.
I'm a covered condition, I'm overlined in green.
I'm a not-covered condition, I'm overlined in red.
I'm a covered statement, I'm highlighted in light green.
I'm a not-covered statement, I'm highlighted in light red.
I don't have red nor green, it means that I'm free from coverage measurement.
```

With a test set of 3 values for function `is_scalene`, all expressions have been activated, but only one decision has taken both true and false values.
From this report, one cannot assure that they have observed any behaviour when function `is_scalene` is given, say for instance, negative values.

Either the test set is not large enough, or the code is wrong (e.g., dead code).

Conclusion and Future Work
- This work was part of SYSTEMATIC project "Couverture".
- For more details, *cf.* [1].
- This approach may (and should) be applied to other languages (*e.g.*, C, Ada, Java, Haskell, Scala, etc.).
- Formal specification of code coverage for pattern-matching.
- Formal specification of code coverage for polymorphic expressions.
- Code coverage for object-oriented programming features, in particular method dispatch (*cf.* DO-178C)
- Code coverage for concurrent programs.
- A theorem prover (*e.g.*, Coq) could be used to extract constructive proofs of code coverage properties.

References

1. Philippe Wang, Applicative Languages and Abstract Machines for Structural Code Coverage. PhD thesis of Université Pierre et Marie Curie (in French), 2012. http://tel.archives-ouvertes.fr/tel-00741549
2. RTCA Special Committee 167 and EUROCAE Working Group 12. DO-178B/ED-12B – Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, December 1992.
3. K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. A Practical Tutorial on Modified Condition/Decision Coverage. Technical report, NASA/TM-2001-210876, May 2001.
4. MLcov (software). http://www-algo-prog.info/mlcov
5. Zamcov (software). http://www-algo-prog.info/zamcov
6. X. Leroy *et al.*. The Objective Caml system release 3.12: Documentation and user's manual, 2011.