Formal Methods – An Appetizer

Chapter 3: Program Verification

Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

`FormalMethods.dk`

Program Verification

The Aim of Program Verification

We want to provide guarantees about the behaviour of models and programs.

Correctness

Is the program correct? that is, does it have the intended functionality?

Safety and Security

Does the program give the required safety and security guarantees?

Performance

Does the program live up to the required performance criteria?

What is needed to reason about such properties?

- We need a precise understanding of the meaning of programs.
  This is provided by the semantics!
- We need a precise description of the property of interest.
  For this we shall introduce predicates.

Four Techniques for Reasoning about Properties

Program Verification (FM p 31)

We attach general predicates to the nodes of the program graph and formally prove that they hold whenever control reaches the nodes – they are invariants.

Program Analysis (FM p 47)

We approximate the properties holding at the nodes. Compared to program verification, program analysis is less expressive but then it is fully automatic.

Model Checking (FM p 77)

We express the properties in a restricted logic – and obtain a fully automatic technique. Model checking is more restrictive than program verification and more precise than program analysis; however, it is also more costly.

Language-based Security (FM p 61)

We restrict the attention to two specific properties of information flow, namely confidentiality and integrity. The resulting analyses are fully automatic and can be used to reject programs violating the security properties.

Section 1

3.1 Predicates

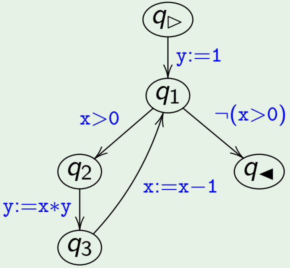
Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

FormalMethods.dk

Example: Factorial Function

How can we express that the program computes the factorial function?

Program Graph



Predicates (FM p 31)

We associate predicates with the nodes:

- $q_\triangleright$: $\mathtt{x} = \underline{\mathtt{n}} \geq 0$
- $q_\blacktriangleleft$: $\mathtt{y} = \mathit{fac}\,(\underline{\mathtt{n}})$

Here $\underline{\mathtt{n}}$ is a variable not used (and hence not modified) in the program – it is called a *logical* variable.

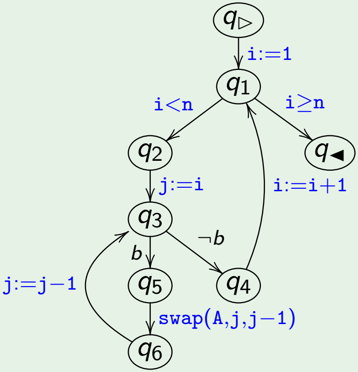
Mathematical Function

$$\mathit{fac}(z) = \begin{cases} 1 & \text{if } z \leq 0 \\ z \cdot \mathit{fac}(z-1) & \text{if } z > 0 \end{cases}$$

Example: Insertion Sort

How can we express that the program sorts the array $\mathtt{A}$?

Program Graph



Predicates (FM p 32)

We associate predicates with the nodes:

- $q_\triangleright$: $\mathtt{A} = \underline{\mathtt{A}}$
- $q_\blacktriangleleft$: $\mathit{sorted}(\mathtt{A}, 0, \mathtt{n}) \wedge \mathit{permuted}(\mathtt{A}, \mathtt{n}, \underline{\mathtt{A}})$

$\mathit{sorted}(\mathtt{A}, \mathtt{m}, \mathtt{n})$ abbreviates

$$\forall \underline{i}, \underline{j} : \mathtt{m} \leq \underline{i} < \underline{j} < \mathtt{n} \Rightarrow \mathtt{A}[\underline{i}] \leq \mathtt{A}[\underline{j}]$$

$\mathit{permuted}(\mathtt{A}, \mathtt{n}, \underline{\mathtt{A}})$ abbreviates

$$\exists \underline{\pi} : \left(\begin{array}{l} (\forall \underline{i}, \underline{j} : 0 \leq \underline{i} < \underline{j} < \mathtt{n} \\ \qquad\qquad \Rightarrow 0 \leq \underline{\pi}(\underline{i}) \neq \underline{\pi}(\underline{j}) < \mathtt{n}) \wedge \\ \forall \underline{i} : \mathtt{A}[\underline{i}] = \underline{\mathtt{A}}[\underline{\pi}(\underline{i})] \end{array} \right)$$

b abbreviates $\mathtt{j > 0\ \&\&\ A[j-1] > A[j]}$

Predicates for Program Verification

Wishlist

We would like to use predicates and expressions going beyond boolean and arithmetic expressions:

- universal quantifiers ($\forall$)
- existential quantifiers ($\exists$)
- mathematical functions (as *fac*)
- pre-defined predicates (as *sorted* and *permuted*)
- logical variables (as $\underline{\mathtt{n}}$, $\underline{\mathtt{A}}$ and $\underline{\pi}$) that are not used in the program

In addition we use

- program variables (as `n`, `x` and `A`) that occur in the programs

Predicates and expressions (FM p 33)

$$
\begin{aligned}
\phi ::= \; & \mathit{true} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_0 \\
\mid \; & \phi_1 \Rightarrow \phi_2 \mid \exists \underline{x} : \phi_0 \mid \forall \underline{x} : \phi_0 \\
\mid \; & e_1 = e_2 \mid \cdots \mid p(e_1, \cdots, e_n)
\end{aligned}
$$

$$
\begin{aligned}
e ::= \; & x \mid \underline{x} \mid e_1 + e_2 \mid \cdots \\
\mid \; & f(e_1, \cdots, e_n)
\end{aligned}
$$

The truth value $(\sigma, \underline{\sigma}) \models \phi$

- σ is a concrete memory giving values to program variables
- $\underline{\sigma}$ is a virtual memory giving values to logical variables

A predicate evaluates to either true or false – never undefined.

Section 2

3.2 Predicate Assignments

Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

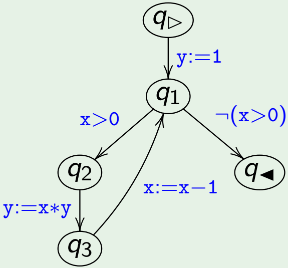
`FormalMethods.dk`

Predicate Assignments

Predicate Assignment (FM p 34)

A *predicate assignment* is a mapping $\mathbf{P} : \mathbf{Q} \to \mathbf{Pred}$ where $\mathbf{Pred}$ is a (non-empty) set of predicates.

Factorial Program (FM p 31)



Predicate Assignment (FM p 34)

$$\mathbf{P}(q_\triangleright) : \quad \mathtt{x} = \underline{\mathtt{n}} \land \underline{\mathtt{n}} \geq 0$$

$$\mathbf{P}(q_1) : \quad \underline{\mathtt{n}} \geq \mathtt{x} \land \mathtt{x} \geq 0 \land \mathtt{y} \cdot \mathit{fac}\,(\mathtt{x}) = \mathit{fac}\,(\underline{\mathtt{n}})$$

$$\mathbf{P}(q_2) : \quad \underline{\mathtt{n}} \geq \mathtt{x} \land \mathtt{x} > 0 \land \mathtt{y} \cdot \mathtt{x} \cdot \mathit{fac}\,(\mathtt{x} - 1) = \mathit{fac}\,(\underline{\mathtt{n}})$$

$$\mathbf{P}(q_3) : \quad \underline{\mathtt{n}} \geq \mathtt{x} \land \mathtt{x} > 0 \land \mathtt{y} \cdot \mathit{fac}\,(\mathtt{x} - 1) = \mathit{fac}\,(\underline{\mathtt{n}})$$

$$\mathbf{P}(q_◀) : \quad \mathtt{y} = \mathit{fac}\,(\underline{\mathtt{n}})$$

What Does Correctness Mean?

Correct Predicate Assignment (FM p 35)

A predicate assignment $\mathbf{P}$ is *correct* if all edges $(q_\circ, \alpha, q_\bullet)$ of the program graph and all pairs of suitable memories $(\sigma, \underline{\sigma})$ satisfy

$$(\sigma, \underline{\sigma}) \models \mathbf{P}(q_\circ) \text{ and } \sigma' = \mathcal{S}[\![\alpha]\!](\sigma)$$

imply

$$(\sigma', \underline{\sigma}) \models \mathbf{P}(q_\bullet)$$

Try It Out (FM p 35)

Show that the predicate assignment given for the factorial program is correct.

Proposition: Partial Correctness (FM p 35)

Let $\mathbf{P}$ be a correct predicate assignment. Then

$$(\sigma, \underline{\sigma}) \models \mathbf{P}(q_\triangleright) \text{ and } \langle q_\triangleright; \sigma\rangle \overset{\omega}{\Longrightarrow}{}^{*} \langle q_\blacktriangleleft; \sigma'\rangle$$

imply

$$(\sigma', \underline{\sigma}) \models \mathbf{P}(q_\blacktriangleleft)$$

$$\xrightarrow{\alpha_1} \quad \xrightarrow{\alpha_2} \qquad\qquad \xrightarrow{\alpha_n}$$

$$\overset{\omega=\alpha_1\alpha_2\cdots\alpha_n}{\Longrightarrow}{}^{*}$$

Section 3

3.3 Partial Predicate Assignments

Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

FormalMethods.dk

Partial Predicate Assignments

Partial Predicate Assignment (FM p 36)

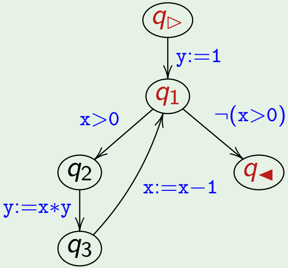
A *partial predicate assignment* is a mapping $\mathbf{P} : \mathbf{Q} \hookrightarrow \mathbf{Pred}$ associating predicates to some of the nodes of **PG**.

P Covers PG (FM p 37)

A partial predicate assignment $\mathbf{P} : \mathbf{Q} \hookrightarrow \mathbf{Pred}$ *covers* the program graph **PG** if

- the nodes $q_\triangleright$ and $q_\blacktriangleleft$ are in $\mathrm{dom}(\mathbf{P})$
- each loop in **PG** contains a node in $\mathrm{dom}(\mathbf{P})$

Factorial Program (FM p 31)



Partial Predicate Assignment (FM p 36)

$\mathbf{P}(q_\triangleright) : \mathtt{x} = \underline{\mathtt{n}} \wedge \underline{\mathtt{n}} \geq 0$

$\mathbf{P}(q_1) : \underline{\mathtt{n}} \geq \mathtt{x} \wedge \mathtt{x} \geq 0 \wedge \mathtt{y} \cdot \mathit{fac}\,(\mathtt{x}) = \mathit{fac}\,(\underline{\mathtt{n}})$

$\mathbf{P}(q_\blacktriangleleft) : \mathtt{y} = \mathit{fac}\,(\underline{\mathtt{n}})$

Short Path Fragments

Short Path Fragment (FM p 37)

A *short path fragment* is a path that starts and ends at nodes in $\mathrm{dom}(\mathbf{P})$ but that do not pass through other nodes in $\mathrm{dom}(\mathbf{P})$.

Correctness: Proof Obligations (FM p 39)

A partial predicate assignment $\mathbf{P}$ is *correct* if all short path fragments $q_\circ \alpha_1 \cdots \alpha_n q_\bullet$ and all pairs of suitable memories $(\sigma, \underline{\sigma})$ satisfy

$$(\sigma, \underline{\sigma}) \models \mathbf{P}(q_\circ) \text{ and } \sigma' = \mathcal{S}[\![\alpha_1 \cdots \alpha_n]\!](\sigma)$$

imply

$$(\sigma', \underline{\sigma}) \models \mathbf{P}(q_\bullet)$$

Short Path Fragments (FM p 37)

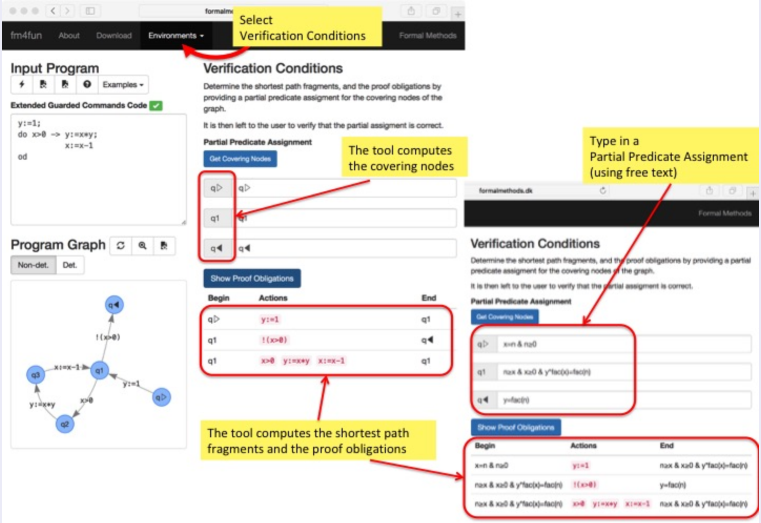
- $q_\triangleright \quad \mathtt{y := 1} \quad q_1$
- $q_1 \quad \mathtt{x > 0} \quad \mathtt{y := x * y}$ $\quad \mathtt{x := x - 1} \quad q_1$
- $q_1 \quad \neg(\mathtt{x > 0}) \quad q_\blacktriangleleft$

Try It Out (FM p 39)

Show that the partial predicate assignment for the factorial program is correct:

- first identify the proof obligations and
- next argue that they hold.

FormalMethods.dk/fm4fun



Hands On: Partial Predicate Assignments

Insertion Sort

```
i := 1;
do i < n →
 j := i;
 do (j > 0)&&(A[j-1] > A[j]) →
     t := A[j]; A[j] := A[j-1];
     A[j-1] := t; j := j-1
 od;
 i := i + 1
od
```

almost(A, m, p, n) abbreviates

$$\forall \underline{i}, \underline{j} : (\mathtt{m} \le \underline{i} < \underline{j} < \mathtt{n}) \Rightarrow (\mathtt{A}[\underline{i}] \le \mathtt{A}[\underline{j}] \lor \underline{j} = \mathtt{p})$$

FormalMethods.dk/fm4fun

Use the tool to construct a partial predicate assignment for the insertion sort program and identify the proof obligations. You may use the predicates *sorted*, *permuted* and *almost*.

Try It Out: Smaller Cover (FM p 37)

The tool does not necessarily compute the smallest set of nodes covering the program graph. Find a smaller set for the insertion sort program. Identify the corresponding shortest path fragments and the associated proof obligations.

Section 4

3.4 Guarded Commands with Predicates

Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

FormalMethods.dk

Guarded Commands with Annotations

Annotation with Predicates

When specifying a Guarded Command:

- Specify a predicate that should hold initially
- Specify a predicate that should hold at termination
- Specify a predicate for each do loop

Factorial Program (FM p 40)

$$
\begin{array}{l}
\texttt{begin}[x = \underline{n} \land \underline{n} \geq 0] \\
y := 1; \\
\texttt{do}\ \ [\underline{n} \geq x \land x \geq 0 \land \\
\qquad y \cdot \mathit{fac}(x) = \mathit{fac}(\underline{n})] \\
\quad x > 0 \to y := x * y; \\
\qquad\qquad\quad x := x - 1 \\
\texttt{od} \\
\texttt{end}[y = \mathit{fac}(\underline{n})]
\end{array}
$$

Extended Syntax (FM p 40)

$$
\begin{array}{lcl}
AP & ::= & \texttt{begin}[\phi_{\triangleright}]\ AC\ \texttt{end}[\phi_{\blacktriangleleft}] \\
AC & ::= & x := a \mid A[a_1] := a_2 \mid \texttt{skip} \\
 & \mid & AC_1\,;AC_2 \mid \texttt{if}\ AG\ \texttt{fi} \\
 & \mid & \texttt{do}[\phi]\ AG\ \texttt{od} \\
AG & ::= & b \to AC \mid AG_1 \, [] \, AG_2
\end{array}
$$

Partial Predicate Assignment (FM p 41, 42)

The construction of program graphs can be extended to produce the partial predicate assignment at the same time.

Section 5

3.5 Reverse Postorder

Flemming Nielson, Hanne Riis Nielson:
Formal Methods – An Appetizer.
ISBN 9783030051556, Springer 2019.

FormalMethods.dk

Nodes Covering a Program Graph

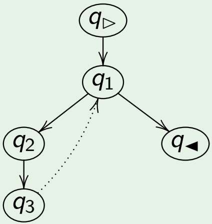
Given a program graph, how can we determine the set a nodes covering it?

Construct a depth first spanning tree for the program graph and the associated reverse post-order.

The reverse post-order will identify all the back edges (that is, all the loops) of the program graph.

The targets of the back edges together with the nodes $q_\triangleright$ and $q_\blacktriangleleft$ will cover the program graph.

Depth First Spanning Tree (FM p 43)



Reverse post-order

The numbering **rP**:

$q_\triangleright$	q_1	q_2	q_3	$q_\blacktriangleleft$
1	2	3	4	5

(q_3, q_1) is a back edge because

$$\mathbf{rP}(q_3) \geq \mathbf{rP}(q_1)$$

Covering Nodes

$\{q_\triangleright, q_\blacktriangleleft, q_1\}$