Guarded Commands, Nondeterminancy and Formal Derivation of Programs

Edsger W. Dijkstra

CACM 18, 8 pp. 453-457 (Aug. 1975).

A Discipline of Programming*, Prentice-Hall, 1976.*
book

2. Two statements made from guarded commands

⟨statement⟩ **::=** *⟨alternative construct⟩* | *⟨repeatative_construct⟩*
| *"⟨other statements⟩"*
⟨alternative construct⟩ **::=** ***if*** *⟨guarded command set⟩* ***fi***
⟨repeatative construct⟩ **::=** **do** *⟨guarded comman set⟩* **od**

"other statements" *assignment statements, procedure calls, ...*

*⟨guarded command set⟩***::=** *⟨guarded command⟩* **{** | *⟨guarded command⟩***}**
⟨guarded command⟩ **::=** *⟨guard⟩* → *⟨guarded list⟩*
⟨guard⟩ **::=** *⟨boolean_expression⟩*
⟨guarded list⟩ **::=** *⟨statement⟩* **{ ;** *⟨statement⟩* **}**

{ *...* **}**: *"followed by zero or more instances of the enclosed ... "*
$= (\ldots)^{*}$.
Extended BNF(Backus-Nauer Form) *ALGOL 60*

*Two separators **;** in ⟨guarded list⟩ and | in ⟨guarded command set⟩*

⟨guarded list⟩ **::=** *⟨statement⟩* **{ ;** *⟨statement⟩* **}**
*⟨guarded command set⟩***::=** *⟨guarded command⟩* **{** | *⟨guarded command⟩***}**

S_1 ; S_2 ; ... ; S_n.

A sequence of S_i’s.

G_1 | G_2 | ... | G_n.

An arbitrarily ordered enumeration of an unordered set

Altenative construct

if *...* ***fi****.*

If ***none*** *of the guard is* ***true****, the program will* ***abort****;*
otherwise an ***arbitrary*** *guarded list with a* ***true guard*** *is selected for* ***execution****.*

(Note) ***if fi*** $\equiv$ ***abort***
An example - illustrating the ***nondeterminancy*** *in a very modest fashion*
For fixed x and y assigns to m the maximum value of x and y.

if $x \geq y \rightarrow m := x$
$|$ $x \leq y \rightarrow m := y$
fi*.*

Repeatative constructs
do … od

None *of the guards is* ***true*** *will* ***not*** *leads to* ***abortion*** *but* ***termination***.

When initially or upon completed execution of selected a guarded list one or more ***guards*** *are* ***true***, *a new selection for* ***execution*** *of a guarded list with a ture will take place, and so on.*

When the repeatative constuct had ***terminated*** *properly, we know* ***all guards*** *are* ***false***.

(Note) ***do od*** $\equiv$ ***skip***.

*An example - showing the **nondeterminancy** in some what greater glory*

Assigns to a variables q_1, q_2, q_3, and q_4
*a **permutation** of the values Q_1, Q_2, Q_3, and Q_4,*
such that $q_1 \le q_2 \le q_3 \le q_4$.

$q_1, q_2, q_3, q_4 := Q_1, Q_2, Q_3, Q_4;$
do $q_1 > q_2 \rightarrow q_1, q_2 := q_2, q_1$
$\mid q_2 > q_3 \rightarrow q_2, q_3 := q_3, q_2$
$\mid q_3 > q_4 \rightarrow q_3, q_4 := q_4, q_3$
od.

$$\neg(q_1 > q_2) \wedge \neg(q_2 > q_3) \wedge \neg(q_3 > q_4) = (q_1 \le q_2) \wedge (q_2 \le q_3) \wedge (q_3 \le q_4)$$
$$= (q_1 \le q_2 \le q_3 \le q_4)$$

An example - Not only computation but also final state is not necessary uniquely determined!

Determine k such that

for fixed value $n(n > 0)$ and a fixed function $f(i)$ defined for $0 \le i < n$:

k will eventually satisfy $0 \le k < n$ and $(\forall i: 0 \le i < n: f(k) \ge f(i))$.

(Eventually k should be the place of a maximun)

$k := 0; j := 1;$
do $j \ne n \rightarrow$ **if** $f(j) \le f(k) \rightarrow j := j + 1$
$\quad | \; f(j) \ge f(k) \rightarrow k := j; j := j + 1$
fi
od.

3. Formal Definitos of the Semantics
3.1 Notational Prelude

P, Q, R to denote (predicate defining)
__boolean__ function defined on all points of the state space.
"__conditions__" satisfied by all states
for which the boolean function is true.

T denotes the condition that is satisfied by __all__ states.
F denotes the condition that is satisfied by __no__ state __at all__.

Hoare sufficient pre-condition
.϶. the mechanisms will __not__ produce the __wrong__ result
(but may fail to terminate)

Dijkstra __necessary__ and sufficinet pre-condition,
i.e, so called "__weakest__" pre-condition
.϶. the mechanisms are guranteed to produce the __right__ result.

wp(S, R) *S: a statement list and*
R: some condition of the system

to denote the ***weakest pre-condition*** *for the initial state of the system .э. activation of S is guranteed to lead to a properly* ***terminating*** *activity leaving the system in a final state* ***satisfying*** *the* ***post-condition*** *R.*

wp *"predicate transformer" has following properties.*

1. $\forall S$: $wp(S, F) = F$. *Law of* ***Excluded Miracle***

2. $\forall S$: If $P \Rightarrow Q$, then $wp(S, P) \Rightarrow wp(S, Q)$.

3. $\forall S$: $(wp(S, P)$ **and** $wp(S, Q)) = wp(S, P$ **and** $Q)$.

4. $\forall$ ***deterministic*** *S*: $(wp(S, P)$ **or** $wp(S, Q)) = wp(S, P$ **or** $Q)$.
   4'. $\forall$ ***nondeterministic*** *S*: $(wp(S, P)$ **or** $wp(S, Q)) \Rightarrow wp(S, P$ **or** $Q)$.

*We know the **semantics** of a mechanism S sufficiently well,*
*if we know its "**predicate transformer**", i.e.,*
can derive wp(S, R) for any post-condition R.

***Example 1**. The semantics of empty statement, denoted by "**skip**" is*
$\forall$*post-condition R: wp("**skip**", R) = R.*

***Example 2**. The semantics of **assignment** statement "x := E" is*
wp("x := E", R) = R_E^x *where*
R_E^x *denotes a copy of the predicate defining R*
in each occurence of the variable x is replaced by (E).

***Example 3**. The semantics of semicolon ";" as **concatenation** operator*
wp("S_1*;* S_2*", R) = wp(*S_1*, wp(*S_2*, R)).*

3.2 The Alternative Construct

Let IF $\equiv$ ***if*** $B_1 \to SL_1 \mid \ldots \mid B_n \to SL_n$ ***fi***.

Let BB $\equiv (\exists i\colon 1 \le i \le n\colon B_i)$. *Then*

$wp(IF, R) = BB \wedge (\forall i\colon 1 \le i \le n\colon B_i \Rightarrow wp(SL_i, R))$.

BB: IF will not lead to ***abortion***.

$B_i \Rightarrow wp(SL_i, R)$: *each guareded list eligible for execution will lead to an acceptable final state.*

Theorem 1. *If* $(\forall i\colon 1 \le i \le n\colon (Q \wedge B_i) \Rightarrow wp(SL_i, R)$, *then*

$(Q \wedge BB) \Rightarrow wp(IF, R)$.

Let t denotes some integer function defined on the state space and wdec(t) denotes the weakest precondition .э. activation S is guranteed to leads to a proper termination activity leaving the system in a final state such that the value of t is decreased by at least 1(compre to its initial value)

Theorem 2. *If* $(\forall i: 1 \leq i \leq n: (Q \wedge B_i) \Rightarrow wp(SL_i, R)$, *then*

$(Q \wedge BB) \Rightarrow wdec(IF, R)$.

3.3 The Repeatative Construct

Let $DO \equiv$ ***do*** $B_1 \rightarrow SL_1 \mid \ldots \mid B_n \rightarrow SL_n$ ***od***.

Let $H_0(R) = R \wedge \neg BB$ *and*

$H_k(R) = wp(IF, H_k(R)) \vee H_0(R)$ *for* $k > 0$. *Then*

$wp(DO, R) = (\exists_k : k \geq 0 : H_k(R))$.

BB: IF will not lead to ***abortion***.

$B_i \Rightarrow wp(SL_i, R)$: *each guareded list eligible for execution will lead to an acceptable final state.*

4. Formal Derivation of Programs

$m = max(x, y)$

$R: ((m = x) \vee (m = y)) \wedge (m \geq x) \wedge (m \geq y).$

*The assignment statement "m := x" will make **true** for (m = x).*
weakest preconditon to make R is true after the statement "m := x".

$$wp(\text{"}m := x\text{"}, R) \equiv (x = x \vee x = y) \wedge (x \geq x) \wedge (x \geq y) = x \geq y.$$

$\therefore$ **if** $x \geq y \rightarrow m := x$ **fi** *the weakest precondition as guard.*

BB = $(x \geq y) \neq T$. Weakening BB. add($\vee$) guards.
The alternative guard is precondion of assignment "m := y".

$$wp(\text{"}m := y\text{"}, R) \equiv (y = x \vee y = y) \wedge (y \geq x) \wedge (y \geq y) = y \geq x.$$

$\therefore$ **if** $x \geq y \rightarrow m := x$
$\quad | \; y \geq x \rightarrow m := y$
fi.

$BB = (x \geq y) \vee (y \geq x) = T.$

Example of ***repeatative construct***

Given two positive numbers X and Y, find x .϶. x = gcd(X, Y).

"establish the relation P to be kept invariant"

do *"decrease t as long as possible uncer variance of P"* ***od***

$P: (gcd(X, Y) = gcd(x, y)) \wedge (x > 0) \wedge (y > 0).$

Easy initialization of P

$x := X;\ y := Y$

Do something under the loop invariance of P

$P \wedge B \Rightarrow wp(\text{"}x, y := E1, E2\text{"}, P) =$

$= (gcd(X, Y) = gcd(E1, E2)) \wedge (E1 > 0) \wedge (E2 > 0).$

$gcd(x, y) = gcd(x - y, y) = gcd(x, y - x) = gcd(y, x) = \ldots$

Consider $t = x + y.$

$wp(\text{"}x := x - y\text{"}, t \le t_0) = wp(\text{"}x := x - y\text{"}, x + y \le t_0) = (x \le t_0).$

$tmin = x. \therefore wdec(\text{"}x := x - y\text{"}, t) = (x < x + y) = (y > 0). \therefore P \Rightarrow wdec.$

$$wp(\text{"}x := x - y\text{"}, P) = (gcd(X, Y) = gcd(x - y, y)) \wedge (x - y > 0) \wedge (y > 0)$$
$$= x > y.$$

$x := X;\ \ y := Y;$
do $x > y \rightarrow x := x - y$ **od**.

$(\neg BB = x \leq y) \not\Rightarrow (x = gcd(X, Y))$
Alternate assignment $y := y - x$ *is required as its guard* $y > x$.
$$wp(\text{"}y := y - x\text{"}, P) = (gcd(X, Y) = gcd(x, y - x)) \wedge (x > 0) \wedge (y - x > 0)$$
$$= y > x.$$

$x := X;\ \ y := Y;$
do $x > y \rightarrow x := x - y$
$|\ \ y > x \rightarrow y := y - x$
od.

$\neg BB = x = y.\ (P \wedge x = y) \Rightarrow x = gcd(X, Y).$

(Note) If you select $t = x + 2y$

$x := X;\ \ y := Y;$
do $x > y \rightarrow x := x - y$
$|\ \ y > x \rightarrow x, y := y, x$
od.

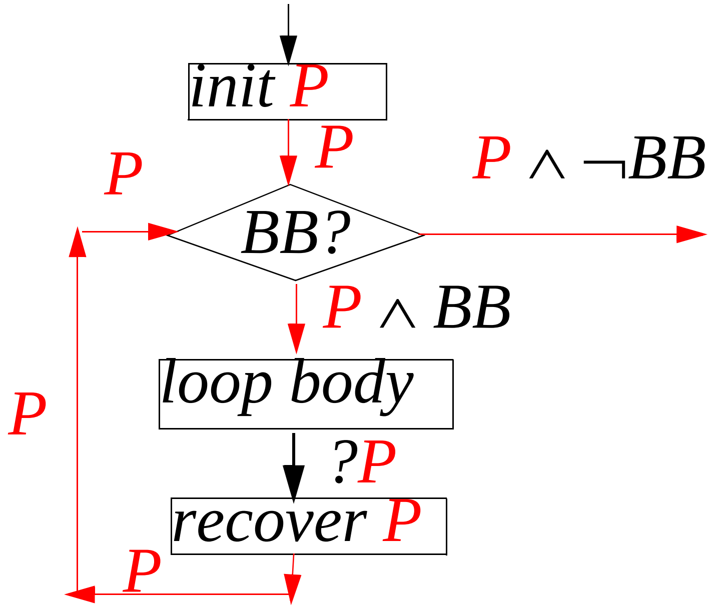
$x := X;\ y := Y;$ *(version A)*
while $x \neq y$ **do if** $x > y$ **then** $x := x - y$
else $y := y - x$ **fi od**.

$x := X;\ y := Y;$ *(version B)*
while $x \neq y$ **do while** $x > y$ **do** $x := x - y$ **od;**
while $y > x$ **do** $y := y - x$ **od.**

$x := X;\ y := Y;$ *(original)*
do $x > y \rightarrow x := x - y$
$|\ \ y > x \rightarrow y := y - x$
od

$$
\begin{array}{l}
\textbf{\textit{if}}\ X > 0\ \textbf{\textit{and}}\ Y > 0 \rightarrow \\
\qquad x := X;\ y := Y; \\
\qquad \textbf{\textit{do}}\ x > y \rightarrow x := x - y \\
\qquad \quad \mid y > x \rightarrow y := y - x \\
\qquad \textbf{\textit{od}} \\
\textbf{\textit{fi}}
\end{array}
$$

syntactic sugar *in C* *(init P; test BB; recover P) loop_body;*



Another Example

Sum, i := 0, 1; $P \equiv (Sum = \sum_{k=1}^{i-1} k)$ ***initialize*** *P*

do $i \le 100 \rightarrow$ $P \equiv (Sum = \sum_{k=1}^{i-1} k)$ *P is still* ***valid***

Sum := Sum + i; $P' \equiv (Sum = \sum_{k=0}^{i} k)$ *P is* ***distroyed***

i := i +1 $P \equiv (Sum = \sum_{k=1}^{i-1} k)$ ***recover*** *P*

od

$$P \wedge \neg(i \le 100) \equiv (Sum = \sum_{k=1}^{i-1} k) \wedge (i = 101)$$

$$= (Sum = \sum_{k=1}^{100} k) = (Sum = 5050).$$

101 – i is the nonnegative decreasing functon
loop terminate