Final VCG Rules

Артем Дмитриевич Ищенко

February 2024

1 Introduction

Processing of Reflex expressions is done throw usage of recursive function $expr(E, U) -> (E_{res}, U_{res}, D)$ which takes expression U and initial state U and returns symbolically evaluated expression E_{res}, updated state U_{res} and calculated limitations on domain of values D presented through logical formula. It uses several auxiliary functions:

- $mark(x)$ - marks variable for future actuation when side effects will be applied
- $act(x, u)$ - actuates variable value
- $typeDet$ - determines resulting type
- $toIsablleOp(op)$ - parses Reflex operation into it's Isabelle representation

Expression of the form $(T)e$ means that e is casted to type T. With given annotations, $expr$ is defined as follows:

1. $expr(c, u) = (c,\ u,\ True)$, If c - constant
2. $expr(x, u) = (mark(x),\ u,\ True)$, If x - variable
3. $expr(\textbf{process } p \textbf{ in state stop}, u) = (getPstate(u, p) = stop,\ u,\ True)$
4. $expr(\textbf{process } p \textbf{ in state error}, u) = (getPstate(u, p) = error,\ u,\ True)$
5. $expr(\textbf{process } p \textbf{ in state inactive}, u) = (getPstate(u, p) = stop \vee getPstate(u, p) = error,\ u,\ True)$
6. $expr(\textbf{process } p \textbf{ in state active}, u) = (\neg(getPstate(u, p) = stop) \wedge \neg(getPstate(u, p) = error),\ u,\ True)$
7. $expr((e), u) = expr(e, u)$

8. Если $\mathbf{op} \in \{+, -, *, <, >, <=, >=, \&\&, ||, ==, !=, |, \&, <<, >>\}$

$function\ expr(e_1\ \mathbf{op}\ e_2, u)\{$

$(E_1, U_1, D_1) = expr(e_1, u)$

$(E_2, U_2, D_2) = expr(e_2, U_1)$

$T = typeDet(E_1, E_2, \mathbf{op})$

$E = ((T)act(E_1, U_2))\ toIsabelleOp(\mathbf{op})\ ((T)act(E_2, U_2))$

$D = D_1 \wedge D_2$

$return(E, U_2, D)\}$

9. Если $\mathbf{op} \in \{/, \%\}$

$function\ expr(e_1\ \mathbf{op}\ e_2, u)\{$

$(E_1, U_1, D_1) = expr(e_1, u)$

$(E_2, U_2, D_2) = expr(e_2, U_1)$

$T = typeDet(E_1, E_2, \mathbf{op})$

$E = ((T)act(E_1, U_2))\ toIsabelleOp(\mathbf{op})\ ((T)act(E_2, U_2))$

$D = D_1 \wedge D_2 \wedge (\neg(((T)act(E_2, U_2)) == 0))$

$return(E, U_2, D)\}$

10. $function\ expr(x = e, u)\{$

$(E_1, U_1, D_1) = expr(e, u)$

$T = typeDet(x, E_1)$

$U = setVar(U_1, x, (T)act(E_1, U_1))$

$return(mark(x), U, D_1)\}$

11. Если $\mathbf{op} \in \{*, +, -, \&, |, <<, >>\}$

$function\ expr(x\ \mathbf{op} = e, u)\{$

$(E_1, U_1, D_1) = expr(e, u)$

$T = typeDet(x, E_1, \mathbf{op})$

$U = setVar(U_1, x, getVar(U_1, x)\ toIsabelleOp(\mathbf{op})\ (T)act(E_1, U_1))$

$return(mark(x), U, D_1)\}$

12. Если $\mathbf{op} \in \{/, \%\}$

$function\ expr(x\ \mathbf{op} = e, u)\{$

$(E_1, U_1, D_1) = expr(e, u)$

$T = typeDet(x, E_1, \mathbf{op})$

$U = setVar(U_1, x, getVar(U_1, x)\ toIsabelleOp(\mathbf{op})\ (T)act(E_1, U_1))$

$D = D_1 \wedge (\neg(((T)act(E_1, U_1)) == 0))$

$return(mark(x), U, D)\}$

13. Если $\mathbf{op} \in \{+, -, !, \tilde{}\}$

$function\ expr(\mathbf{op}\ e, u)\{$
$(E_1, U_1, D_1) = expr(e, u)$
$checkT(\mathbf{op}, typeDet(E_1))$
$E = toIsabelleOp(\mathbf{op})\ act(E_1, U_1)$
$return(E, U_1, D_1)\}$

14. $function\ expr(\mathbf{(type)}\ e, u)\{$
$(E_1, U_1, D_1) = expr(e, u)$
$T = (typeDet(\mathbf{type}))$
$E = (T)\ act(E_1, U_1)$
$return(E, U_1, D_1)\}$

15. $function\ expr(++x, u)\{$
$T = (x)$
$U = setVar(u, x, getVar(u, x) + ((T)1))$
$E = (mark(x))$
$return(E, U, True)\}$

16. $function\ expr(\mathbf{--}x, u)\{$
$T = (x)$
$U = setVar(u, x, getVar(u, x) - ((T)1))$
$E = (mark(x))$
$return(E, U, True)\}$

17. $function\ expr(x++, u)\{$
$T = (x)$
$U = setVar(u, x, getVar(u, x) + ((T)1))$
$E = (getVar(u, x))$
$return(E, U, True)\}$

18. $function\ expr(x\mathbf{--}, u)\{$
$T = (x)$
$U = setVar(u, x, getVar(u, x) - ((T)1))$
$E = (getVar(u, x))$
$return(E, U, True)\}$

1.1 Правила выражений

Вычисление выражений задается функцией $gen((P, U), W) -> (P_{new}, U_{new})$ где P и P_{new} начальное и итоговое предусловие, U и U_{new} начальное и итоговое состояние, W - обрабатываемая конструкция языка. W имеет ссылку на родительскую конструкцию получаемую функцией $parent(W)$.

Также в процессе вычисления исползуются функции $stackPush(P, U, W, i)$, $stackPop() -> (P, U, W, i)$ и $isStackEmpty() -> bool$ отвечающие за взаимодействие с глобальным стеком, сохраняющим не пройденные генератором на ветвлениях пути. P означает уакопленное к текущему моменту предусловие. W - конструкция языка на которой произошло ветвление. Индекс i обозначает не пройденный ветвлением путь и зависит от типа ветвления

1. Для конструкции if **exp** $= true$ $\{\}$ $else\{\}$, i принимает значения 0 и 1 соответствующие **exp** $= true$ и **exp** $= false$.

2. Для конструкции $switch$ **exp** $case\ c_0\ \ldots\ case\ c_n\ default$, i принимает значения от 0 до $n+1$(включительно) соответствующие **exp** $= c_i$ для $i = 0..n$ и $default$ для $i = n+1$

3. Для конструкции $timeout$, i принимает значения 0 и 1 соответствующие сработавшему и не сработавшему таймауту

4. Для конструкции $process$, i принимает значения от -2 до n, $i = -2$ соответствует процессу в состоянии $error$, $i = -1$ соответствует процессу в состоянии $stop$, i от 0 до n соответствуют состоянию в котором находится процесс.

Сгенерированные условия корректности сохраняются в структуре vcs, добавление происходит операцией $+=$. Дополнительно определены следующие функции:

1. $constTrue(D) -> bool$ - Для предиката области выражения D проверяет не является ли он постоянно истинным

2. $markRestart(P)$, $unmarkRestart(P)$ и $checkRestart(P) -> bool$ - добавляет, убирает и проверяет пометку о том что в текущей цепочке операторов была команда restart

3. $markStateChange(P)$, $unmarkStateChange(P)$ и $checkStateChange(P) -> bool$ - добавляет, убирает и проверяет пометку о том что в текущей цепочке операторов была комнда имзменения состояния

   На практике unmark будет оставлять свою пометку, а check проходить справа на лево по цепочки до первой метки

4. $initialize(U, p) -> U$ - проводит инициализацию всех переменных процесса

5. $getStartState(p) -> pstate$ - возвращает имя первого состояния

6. $getNextState(p, s) -> pstate$ - возвращает имя состояния следубщего за состоянием s

Символами p_0 и s_0 обозначаются имена текущих процесса и состояния соответственно. $this$ обозначает ссылку на текущую обрабатываемою кострукцию

Правила для комманд:

1. Вычисление выражения. сначало происходит символьное вычисления выражения, в процессе образуется обновленное состояние и может оьразоваться ограничение области определения, в случае чеог оно дописывается в предусловие

   $function\ gen((P,U),e;)\{$
   $(E_1,U_1,D_1) = expr((P,U),e);$
   $if\ !constTrue(D_1)\{$
   $vcs+ = (P \longrightarrow D_1)$
   $P_1 = (P \wedge D_1)$
   $\}else\{$
   $P_1 = P$
   $\}$
   $return\ (P_1,U_1)\}$

2. Команда изменения состояния процесса. Вычисляется новое состояние и делается пометка что состояние было обновлено.

   $function\ gen((P,U),set\ state\ s)\{$
   $U_1 = setPstate(U,p_0,s)$
   $markStateChange(P)$
   $return\ (P,U_1)\}$

3. Команда изменения состояния процесса. Вычисляется новое состояние и делается пометка что состояние было обновлено.

   $function\ gen((P,U),set\ next\ state)\{$, где s - следующие состояние за текущим
   $U_1 = setPstate(U,p_0,getNextState(p_0,s_0))$
   $markStateChange(P)$
   $return\ (P,U_1)\}$

4. Команда обновления таймера. Вычисляется новое состояние.

   $function\ gen((P,U),reset\ timer)\{$
   $U_1 = reset(U,p_0)$
   $return\ (P,U_1)\}$

5. Команда перезапуска текущего процесса. Добавляется пометка что был произведен рестарт.

   $function\ gen((P,U),restart)\{$
   $markRestart()$
   $return\ (P,U)\}$

6. Команда начала процесса. Если имя процесса соответствует текущему, что повторяется сценарий перезапуска, иначе указанный процесс переходит в первое состояние.

$function\ gen((P,U), start\ process\ p)\{$
$\quad if\ p == p_0\{$
$\quad\quad markRestart()$
$\quad\quad U_1 = U$
$\quad \}else\{$
$\quad\quad U_1 = setPstate(U, p, getStartState(p))$
$\quad \}$
$return\ (P, U_1)\}$

7. Команда остановки текущего процесса. Изменяется текущее стостояние процесса. Добавляется пометка что было изменение текущего состояния процесса.

$function\ gen((P,U), stop)\{$
$\quad U_1 = setPstate(U, p_0, stop)$
$\quad markStateChange()$
$return\ (P, U_1)\}$

8. Команда остановки другого процесса. Изменяется текущее стостояние процесса. Добавляется пометка что было изменение текущего состояния процесса.

$function\ gen((P,U), stop\ process\ p)\{$
$\quad U_1 = setPstate(U, p, stop)$
$\quad if\ p == p_0\{$
$\quad\quad markStateChange()$
$\quad \}$
$return\ (P, U_1)\}$

9. Команда остановки с ошибкой текущего процесса. Изменяется текущее стостояние процесса. Добавляется пометка что было изменение текущего состояния процесса.

$function\ gen((P,U), error)\{$
$\quad U_1 = setPstate(U, p_0, error)$
$\quad markStateChange()$
$return\ (P, U_1)\}$

10. Команда остановки с ошибкой текущего процесса. Изменяется текущее стостояние процесса. Добавляется пометка что было изменение текущего состояния процесса.

$function\ gen((P,U), error\ process\ p)\{$
$\quad U_1 = setPstate(U, p, error)$
$\quad if\ p == p_0\{$
$\quad\quad markStateChange()$
$\quad\}$
$return\ (P, U_1)\}$

11. Команда обработки последовательности комманд. команды обрабатываются последовательно. $function\ gen((P,U), q_1 \dots q_n)\{$
$U_1 = U$
$P_1 = P$
$for\ i\ in\ (1.. = n)\{$
$(P_1, U_1) = gen((P_1, U_1), q_i)$
$\}$
$return\ (P_1, U_1)\}$

12. Команда ветвления. Вычисляется выражение (условие). В стэк добавляется пометка что необработанно условие ложности. К предусловию добавляется утверждение что условие истинно и обрабатывается соотвестующая ветвь.

$function\ gen((P,U), if\ exp\ then\ w_1\ else\ w_2)\{$
$\quad (E_1, U_1, D_1) = expr(exp, U)$
$\quad if\ !constTrue(D_1)\{$
$\quad\quad vcs+ = (P => D_1)$
$\quad\quad P_1 = (P \wedge D_1)$
$\quad\}else\{$
$\quad\quad P_1 = P$
$\quad\}$
$\quad stackPush(P_1, U_1, this, 1)$
$\quad P_1 = (P_1 \wedge E_1 = True)$
$return\ gen((P_1, U_1), w_1)\}$

13. Команда ветвления switch. Вычисляется выражение (условие). В стэк добавляются пометки о необработанных условиях 1..n и когда все условия ложны. К предусловию добавляется утверждение что условие равно метке 0 и обрабатывается соотвестующая ветвь. Если не было команды break;, то далее обрабатываются последующие ветви пока такая команда не будет встреченна или не будет обработано все.

$function\ gen((P,U),$
$switch(exp)\{$
$case\ c_0\{w_0; break_1\}$
$\ldots$
$case\ c_n\{w_n; break_n\}$
$default\{w_def\})\{$
$(E_1,U_1,D_1) = expr(exp,U)$
$if\ !constTrue(D_1)\{$
$vcs+ = (P => D_1)$
$P_1 = (P \wedge D_1)$
$\}else\{$
$P_1 = P$
$\}$
$for\ i\ in\ (1.. = n+1)\{$
$stackPush(P_1,U_1,this,i)$
$\}$
$U_1 = U$
$break_{def} = false$
$P_1 = (P_1 \wedge E_1 = c_0)$
$for\ i\ in\ (0.. = n)\{$
$(P_1,U_1) = gen((P_1,U_1),w_i)$
$if\ break_i\{$
$break_{def} = true$
$break$
$\}$
$\}$
$if\ (!break_{def})\{$
$(P_1,U_1) = gen((P_1,U_1),w_{def})$
$\}$
$return\ (P_1,U_1)\}$

14. Команда таймаута. Проверяется было ли в текущем проходе изменение состояния или рестарт. Если было то дальнейшей обрабокти не происходит т.к. уловие всегда будет ложно. Иначе в стэк добавляется пометка о необработанном условии не преывшения таймаута. После в предусловие добавляется уловие что время больше таймаута

$function\ gen((P,U), timeout\ e\ \{q_1 \dots q_n\})\{$
$\quad if(checkStateChange(P)||checkRestart(P))\{$
$\quad\quad return(P,U)$
$\quad \{$
$\quad stackPush(P,U,this,1)$
$\quad P_1 = (P \wedge e < ltime(U,p_0))$
$return\ gen((P_1,U_1), q_1 \dots q_n)\}$

15. Описание состояния. Происходит вычисление внутренних комманд состояния и снимается метка что состояние было изменено.

$function\ gen((P,U), state\ s\ \{q_1 \dots q_n\})\{$
$\quad (P_1,U_1) = gen((P,U), q_1 \dots q_n)$
$\quad unmarkStateChange(P_1)$
$return\ (P_1,U_1)\}$

16. Описание процесса. В стэк добавляется метка о не вычисленных состояних. В предусловие добавлется что текущее состояние равно error.

$function\ gen((P,U), process\ p\ \{s_0 \dots s_n\})\{$
$\quad for\ i\ in\ (-1..n)\{$
$\quad\quad stackPush(P,U,this,i)$
$\quad \}$
$\quad P_1 = (P \wedge getPstate(U,p) = error)$
$return\ (P_1,U)\}$

17. Описание программы. Добавляется условие корректности соотвествующее утверждению, что из состояния программы после инициализации следует инвариант. После вычисляется программа и добавляется условие корректности соотвествующее утверждению, что из начального сотояния программы сответствующего инварианту следует что новое состояние полученное после однократного прохода программы соответствует инварианту.

$function\ gen((P,U), program\ p\ \{pr_0 \dots pr_n\})\{$
$U_0 = U$
$for\ i\ in\ (0..n)\{$
$U_0 = initialize(U_1, pr_i)$
$\}$
$U_0 = setPstate(U_0, pr_0, getStartState(pr_0))$
$vcs+ = (u_0 = U_0 \longrightarrow inv(u_0))$
$(P_{res}, U_{res}) = gen((inv(U), U), pr_0 \dots pr_n)$
$U_{upd} = toEnv(U_{res})$
$vcs+ = P_{res} \longrightarrow inv(U_{upd})$
$return\}$

В процессе генерации образуется *stack* не пройденных путей. Его обработка происходит в с помощью функций $genRest(P,U,W,W_{child}) -> (P,U)$ и $genMiss(P,U,W,i) -> (P,U)$ внутри функции $genOther(P,U)$ до тех пор пока *stack* не опустеет:

$function\ genOther(P,U)\{$
$U_0 = U$
$while\ !isStackEmpty()\{$
$(P_{res}, U_{res}) = genMiss(stackPop())$
$U_{upd} = toEnv(U_{res})$
$vcs+ = P_{res} \longrightarrow inv(U_{upd})$
$\}$
$return\}$

Правила для $genMiss(P,W,i)$:

1. $function\ genMiss(P,U,if\ exp\ then\ w1\ else\ w2,i)\{$
$(E_1, U_1, D_1) = expr(exp, U)$
$U_1 = U$
$if\ i = 1\{$
$P_1 = (P \wedge E_1 = False)$
$(P_1, U_1) = gen((P_1, U_1), w2)$
$\}else\{$
$P_1 = (P \wedge E_1 = True)$
$(P_1, U_1) = gen((P_1, U_1), w1)$
$\}$
$return\ genRest(P_1, U_1, parent(this), this)\}$

2. $function\ genMiss(P, U,$
   $switch(exp)\{$
   $case\ c_0\{w_0; break_1\}$
   $\ldots$
   $case\ c_n\{w_n; break_n\}$
   $default\{w_{def}\}, i)\{$
   $(E_1, U_1, D_1) = expr(exp, U)$
   $U_1 = U$
   $break_{def} = false$
   $for\ j\ in\ (0..i)\{$
   $P_1 = (P_1 \wedge E_1! = c_j)$
   $\}$
   $P_1 = (P_1 \wedge E_1 = c_i)$
   $for\ j\ in\ (i.. = n)\{$
   $(P_1, U_1) = gen((P_1, U_1), w_j)$
   $if\ break_j\{$
   $break_{def} = true$
   $break$
   $\}$
   $\}$
   $if\ (!break_{def})\{$
   $(P_1, U_1) = gen((P_1, U_1), w_def)$
   $\}$
   $return\ genRest(P_1, U_1, parent(this), this)\}$

3. $function\ genMiss(P, U, timeout\ e\ \{w\}, i)\{$
   $(E_1, U_1, D_1) = expr(exp, U)$
   $U_1 = U$
   $if\ i = 0\{$
   $P_1 = (P \wedge e > ltime(U, p_0))$
   $(P_1, U_1) = gen((P_1, U_1), w)$
   $\}else\{$
   $P_1 = (P \wedge e <= ltime(U, p_0))$
   $\}$
   $return\ genRest(P_1, U_1, parent(this), this)\}$

4. $function\ genMiss(P, U, process\ p\ \{w_0 \dots w_n\}, i)\{$
   $U_1 = U$
   $if\ i = -1\{$
      $P_1 = (P \wedge getPstate(U, p) = stop)$
   $(P_1, U_1) = gen((P_1, U_1), w)$
   $\}else\{$
      $P_1 = (P \wedge getPstate(U, p) = w_i)$
      $(P_1, U_1) = gen((P_1, U_1), w_i)$
   $\}$
   $unmarkRestart(P_1)$
  $return\ genRest(P_1, U_1, parent(this), this)\}$

Правила для $genRest(P, U, W, W_{child})$:

1. $function\ genRest(P, U, q_1 \dots q_n, W_{child})\{$
   $for\ i\ in\ (1..n)\{$
      $if\ q_i = W_{child}\{break\}$
   $\}$
   $(P_1, U_1) = gen((P, U), q_{i+1} \dots q_n)$
  $return\ genRest(P_1, U_1, parent(this), this)\}$

2. $function\ genRest(P, U, if \dots, W_{child})\{$
  $return\ genRest(P, U, parent(this), this)\}$

3. $function\ genRest(P, U,$
   $switch(exp)\{$
   $case\ c_0\{w_0; break_1\}$
   $\ldots$
   $case\ c_n\{w_n; break_n\}$
   $default\{w_{def}\},$
   $W_{child})\{$
   $for\ i\ in\ (0.. = n)\{$
   $if\ w_i = W_{child}\{break\}$
   $\}$
   $if\ break_i\{$
   $return\ genRest(P, U, parent(this), this)$
   $\}\ else\{$
   $break_{def} = false;$
   $\}$
   $for\ j\ in\ (i + 1.. = n)\{$
   $(P_1, U_1) = gen((P_1, U_1), w_j)$
   $if\ break_j\{$
   $break_{def} = true$
   $break$
   $\}$
   $\}$
   $if\ (!break_{def})\{$
   $(P_1, U_1) = gen((P_1, U_1), w_def)$
   $\}$
   $return\ genRest(P, U, parent(this), this)\}$

4. $function\ genRest(P, U, timeout \ldots, W_{child})\{$
   $return\ genRest(P, U, parent(this), this)\}$

5. $function\ genRest(P, U, state\ s\ \{q_1 \ldots q_n\}, W_{child})\{$
   $for\ i\ in\ (1..n)\{$
   $if\ q_i = W_{child}\{break\}$
   $\}$
   $(P_1, U_1) = gen((P, U), q_{i+1} \ldots q_n)$
   $unmarkStateChange(P_1);$
   $return\ genRest(P_1, U_1, parent(this), this)\}$

6. $function\ genRest(P, U, process\ p\ \{s_1 \ldots s_n\}, W_{child})\{$
   $unmarkReset(P);$
   $return\ genRest(P, U, parent(this), this)\}$

7. $function\ genRest(P, U, program\ p\ \{pr_1 \ldots pr_n\}, W_{child})\{$
   $for\ i\ in\ (1..n)\{$
   $if\ pr_i = W_{child}\{break\}$
   $\}$
   $(P_1, U_1) = gen((P, U), pr_{i+1} \ldots pr_n)$
   $return\ (P_1, U_1)\}$