Demo file of Aligator's main components

Loading the package

\[ In[1]:= \text{\texttt{\textless \textless Aligator.m}} \]

Input Check

Checks whether an expression is syntactically correct Aligator input. Correct Aligator input is of the form:

\[
\text{WHILE[cond1, s1; IF[cond2, s2, IF[cond3, s3, ...]; ...]]; sk]}
\]

where
- si (i=1,...,k) are sequences of assignments,
- condi (i=1,...,k-1) are boolean conditions or ...
- the nested conditional might be omitted in the loop body.

Input for InputCheck: expression
Output of InputCheck:
- 1 if expression is syntactically correct Aligator input
  or
- 0, otherwise. In this case, the error in the input is pointed out.

\[ \text{\textbullet InputCheck} \]

\[ In[2]:= \text{InputCheck[a]} \]

\text{aligator::InputError4 :}
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body].
Given input must be a WHILE loop!

\[ Out[2]= 0 \]

\[ In[3]:= \text{InputCheck[WHILE[g, u, x]]} \]

\text{aligator::InputError3 :}
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body].
WHILE has 2 arguments only!

\[ Out[3]= 0 \]
In[4]:= InputCheck[WHILE[g, u]]

aligator::InputError2 :=
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :=)! Hint: check also for spellings (e.g., := instead of =, IF instead of If, ; instead of ,)! 

Out[4]= 0

In[5]:= InputCheck[While[]]

aligator::InputError4 := 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given input must be a WHILE loop!

Out[5]= 0

In[6]:= InputCheck[WHILE[, j]]

aligator::InputError1 := 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-condition is not appropriate - it has to be a NON-EMPTY BOOLEAN CONDITION (without :=)!

Out[6]= 0

In[7]:= InputCheck[WHILE[b > 0, x := 1]]

Out[7]= 1

In[8]:= InputCheck[g; WHILE[b > 0, x := 1]]

aligator::InputError4 := 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given input must be a WHILE loop!

Out[8]= 0

In[9]:= InputCheck[WHILE[b > 0, x := 1; j := 2]]

Out[9]= 1

In[10]:= InputCheck[WHILE[, x := 1; j := 2]]

aligator::InputError1 := 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-condition is not appropriate - it has to be a NON-EMPTY BOOLEAN CONDITION (without :=)!

Out[10]= 0

In[11]:= InputCheck[WHILE[b > 0,]]

aligator::InputError2 := 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :=)! Hint: check also for spellings (e.g., := instead of =, IF instead of If, ; instead of ,)!

Out[11]= 0
In[12]:= InputCheck[WHILE[b > 0, x1]]

g::InputError2 :
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUNEC E OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :-).

Hint: check also for spellings (e.g. := instead of =, IF instead of If, ; instead of ,)

Out[12]= 0

In[13]:= InputCheck[WHILE[b > 0, IF[j > 1, x := 1, u := 2]]]

Out[13]= 1

In[14]:= InputCheck[WHILE[b > 0, IF[,] x := 1, u := 2]]

g::InputError2 :
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUNEC E OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :-).

Hint: check also for spellings (e.g. := instead of =, IF instead of If, ; instead of ,)

Out[14]= 0

In[15]:= InputCheck[WHILE[b > 0, s := 1; IF[j > 1, x := 1, u := 2]]]

Out[15]= 1

In[16]:= InputCheck[WHILE[b > 0, s := 1; IF[j > 1, x := 1, u := 2]; g := 0]]

Out[16]= 1

In[17]:= InputCheck[WHILE[b > 0, IF[j > 1, x := 1, u := 2]; g := 0]]

Out[17]= 1

In[18]:= InputCheck[WHILE[b > 0, s := 1; IF[j > 1, x := 1, IF[j > 8, u := 2]]; g := 0; l := 9]]

Out[18]= 1

Check if given loop contains conditionals

CheckIfSeq is called ONLY if the input is correct, i.e. InputCheck returns 1 as result. CheckIfSeq check whether the loop contains conditional statements, or not.

Input: While loop
Output: False, if the loop has only assignments, and True if the loop is with conditionals.

- CheckIfSeq

    CheckIfSeq[WHILE[... , x := 1; u := 9]]

    False
Loop Transform

Loop Transform gets as input a syntactically correct loop with/without nested conditional statements.

It returns the list of paths (inner loops) in the given loop body.

We use the auxiliary function Body to keep the content of each path unevaluated.

In the input of Loop Transform, the 2nd and 3rd arguments are both Body[], and denote the empty statement.

They are used to specify the statements in the loop body before and after the first argument of Loop Transform.

Output is of the form:

- [Body[S1], ..., Body[Sk]], where S1, ..., Sk are sequences of assignments representing the possible (disjunctive) paths in the given loop.

or

- Body[S], where S is a sequence of assignments representing the body of a given loop without conditions.

```
In[20]:= LoopTransform[u := 1; y := 9; u := 1, Body[], Body[]]
Out[20]= Body[u := 1; y := 9; u := 1]

In[21]:= LoopTransform[y := 9; IF[g > 0, u := 1], Body[], Body[]]
Out[21]= {Body[y := 9; u := 1], Body[y := 9]}

In[22]:= LoopTransform[IF[g > 0, u := 1], Body[], Body[]]
Out[22]= {Body[u := 1], Body[]} 

In[23]:= LoopTransform[y := 9; IF[g > 0, u := 1, u := 8], Body[], Body[]]
Out[23]= {Body[y := 9; u := 1], Body[y := 9; u := 8]}

In[24]:= LoopTransform[y := 9; IF[g > 0, u := 1, u := 8]; k := 0, Body[], Body[]]
Out[24]= {Body[y := 9; u := 1; k := 0], Body[y := 9; u := 8; k := 0]}

In[25]:= LoopTransform[IF[g > 0, u := 1]; k := 0; y := 9, Body[], Body[]]
Out[25]= {Body[u := 1; k := 0; y := 9], Body[k := 0; y := 9]}

In[26]:= LoopTransform[k := 0; y := 9; IF[g > 0, u := 1], Body[], Body[]]
Out[26]= {Body[k := 0; y := 9; u := 1], Body[k := 0; y := 9]}

In[27]:= LoopTransform[y := 9; x := x + 8; IF[g > 0, u := 1, v := 2]; k := 0, Body[], Body[]]
Out[27]= {Body[y := 9; x := x + 8; u := 1; k := 0], Body[y := 9; x := x + 8; v := 2; k := 0]}
```
Recurrence System of Inner Loops

RecSystem builds the system of recurrences of a given sequence of assignments (representing the body of a loop with only assignments).

It uses a fresh new variable standing for the iteration counter of the loop.

Loop body is simplifies and flattenes in the construction of recurrence system.

Input: sequence of assignments, smth of the form x1:=expr1; x2:=expr2; x2:=expr3

Output: list containing
- recurrence equations of loop variables,
- correspondence between variables and recurrence sequences
- loop iteration counter.

RecSystem
\textbf{Recurrence Solving}

- \textbf{RecSolve}

Computes the closed forms of a given recurrence system.

Input: system of recurrences, variable correspondence and recurrence index.

Output: a list containing the
- system of closed forms
- recurrence index
- new variables standing for the exponential sequences in the closed forms
- final values of variables
- initial values of variables
- algebraic dependencies among the new variables (standing for the exponential sequences).

\textbf{In}[103]:=
\textbf{sys} = \{y[1 + n] = 1 + y[n], x[1 + n] = 2 + x[n]\}

\textbf{Out}[103]:=
\{y[1 + n] = 1 + y[n], x[1 + n] = 2 + x[n]\}
\textbf{In}[104]:=
\texttt{vars = \{\{y[n], y\}, \{x[n], x\}\}}

\texttt{Out}[104]=
\texttt{\{\{y[n], y\}, \{x[n], x\}\}}

\textbf{In}[105]:=
\texttt{RecSolve[sys, vars, \{n\}\}}

\texttt{Out}[105]=
\texttt{\{\{y = n + y[0], x = 2 n + x[0]\}, \{n\}, \{\}, \{y, x\}, \{y[0], x[0]\}, \{\}\\}}

\textbf{In}[106]:=
\texttt{sys = \{y[1 + n] = 1 + 2 y[n], x[1 + n] = 2 + 4 x[n]\}}

\texttt{Out}[106]=
\texttt{\{y[1 + n] = 1 + 2 y[n], x[1 + n] = 2 + 4 x[n]\}}

\textbf{In}[107]:=
\texttt{vars = \{\{y[n], y\}, \{x[n], x\}\}}

\texttt{Out}[107]=
\texttt{\{\{y[n], y\}, \{x[n], x\}\}}

\textbf{In}[108]:=
\texttt{RecSolve[sys, vars, \{n\}\}}

\texttt{Out}[108]=
\texttt{\{\{y = -1 + (1 + y[0]) 25[1], x = -\frac{2}{3} + \left(\frac{2}{3} + x[0]\right) 25[2]\}, \{n\}, \{25[1], 25[2]\}, \{y, x\}, \{y[0], x[0]\}, \{25[1]^2 - 25[2] = 0\}\}}

\textbf{In}[109]:=
\texttt{sys = \{a[n + 1] = 2 a[n] + 1, b[n + 1] = b[n] - a[n] + c[n + 1], c[n + 1] = c[n] + a[n + 1]\}}

\texttt{Out}[109]=
\texttt{\{a[1 + n] = 1 + 2 a[n], b[1 + n] = -a[n] + b[n] + c[1 + n], c[1 + n] = a[1 + n] + c[n]\}}

\textbf{In}[110]:=
\texttt{varList = \{\{a[n], a\}, \{b[n], b\}, \{c[n], c\}, \{b[n + 1], e\}\}}

\texttt{Out}[110]=
\texttt{\{\{a[n], a\}, \{b[n], b\}, \{c[n], c\}, \{b[1 + n], e\}\}}

\textbf{In}[111]:=
\texttt{RecSolve[sys, varList, \{n\}\}}

\texttt{Out}[111]=
\texttt{\{\{a = -1 + (1 + a[0]) 26[1], b = \frac{1}{2} (-6 - 3 n - n^2 - 6 a[0] - 4 n a[0] + 2 b[0] + 2 n c[0]) + 3 (1 + a[0]) 26[3], c = -2 - 2 a[0] + c[0] + 2 (1 + a[0]) 26[2], e = \frac{1}{2} (-6 - 3 \left(1 + n\right) - (1 + n)^2 - 6 a[0] - 4 \left(1 + n\right) a[0] + 2 b[0] + 2 \left(1 + n\right) c[0]) + 6 (1 + a[0]) 26[3], \{a[0], b[0], c[0], e[0]\}, \{26[1], 26[2], 26[3]\}, \{a, b, c, e\}, \{a[0], b[0], c[0], e[0]\}, \{26[1] - 26[3] = 0, 26[2] - 26[3] = 0\}\}}
In[112]:=

sys = \{z[1+n] = x[n] + z[n], x[1+n] = 2x[n], y[1+n] = -1+y[n]\}

Out[112]=
\{z[1+n] = x[n] + z[n], x[1+n] = 2x[n], y[1+n] = -1+y[n]\}

In[113]:=

varList = \{(z[n], z), (x[n], x), (y[n], y)\}

Out[113]=
\{(z[n], z), (x[n], x), (y[n], y)\}

In[115]:=

RecSolve[sys, varList, \{n\}]

Out[115]=
\{\{(n = 29 - x[0] + z[0] + x[0] \times 28[2], x = x[0] \times 28[1], y = -2+y[0])\}, \{(28[1], 28[2]), (z, x, y), (x[0], x[0], y[0]), (28[1] - 28[2] = 0)\}\}

In[116]:=

sys = \{a[n+1] = 2a[n] + 1, b[n+1] = b[n] - c[n], c[n+1] = c[n]^2 + a[n]\}

Out[116]=
\{a[1+n] = 2a[n] + 1, b[1+n] = b[n] - c[n], c[1+n] = a[n] + c[n]^2\}

In[117]:=

RecSolve[sys, \{x\}, \{n\}]

Out[117]=
\{\{n = 29 - a[0] + b[0] + a[0] \times 28[2], a = a[0] \times 28[1], b = b[0] \times 28[2]\}\}

In[122]:=

sys = \{a[n+1] = 2a[n], b[n+1] = b[n] - 2\}

Out[122]=
\{a[1+n] = 2a[n], b[1+n] = -2+b[n]\}

In[123]:=

varList = \{(a[n], a), (b[n], b)\}

Out[123]=
\{(a[n], a), (b[n], b)\}

In[124]:=

RecSolve[sys, varList, \{n\}]

Out[124]=
\$Aborted
EqRecSolve

Computes the closed form of a given recurrence equation.

Input: recurrence equation and recurrence index

Output:
- True and the closed form system of the recurrence equation with the list of the bases of exponential sequences that appear in the closed form
- False (and error message) if the recurrence equation is not (solvable) Gosper-summable or C-finite.

```
In[88]:= EqRecSolve[c[1 + n] == -1 + 2^{1+n} (1 + a[0]) + c[n], n]
Out[88]= {True, c[n] == -2 - n - 2 a[0] + 2^{1+n} (1 + a[0]) + c[0],
  {c, {2, 1}, {-2 + 6 (1 + a[0]) + 2 c[0] - 2 (-1 + 2 (1 + a[0]) + c[0])},
   {2 - 6 (1 + a[0]) - c[0] + 2 (-1 + 2 (1 + a[0]) + c[0]),
    2 - 6 (1 + a[0]) - 3 c[0] + 3 (-1 + 2 (1 + a[0]) + c[0])}, {2}}}

In[89]:= EqRecSolve[y[1 + n] == 1/2 * (-1 + y[n]), n]
Out[89]= {True, y[n] == -1 + 2^{-n} (1 + y[0]),
  {y, {1, 1/2}, {-1}, {2 (1/2 (1 - y[0]) + y[0])}}, {1/2}}

In[90]:= EqRecSolve[y[1 + n] == 1/2 * (-1 + y[n]), n]
Out[90]= {True, y[n] == -1 + 2^{-n} (1 + y[0]),
  {y, {1, 1/2}, {-1}, {2 (1/2 (1 - y[0]) + y[0])}}, {1/2}}

In[91]:= EqRecSolve[y[1 + n] == -1/2 - 1/2 * y[n], n]
Out[91]= {True, y[n] == -1/2 + (1/2)^n (1/2 + y[0]),
  {y, {1, -1/2}, {1/3 (2 (-1/2 + y[0]/2) + y[0])}, {2/3 (1/2 + 3 y[0]/2)}}, {1/2}}

In[92]:= EqRecSolve[x[n + 1] == -1/2 * x[n] - 2^n, n]
Out[92]= {True, x[n] == -2^{1+n}/5 + (1/2)^n (2/5 + x[0]),
  {x, {2, -1/2}, {1/5 (2 (-1 - x[0]/2) + x[0])}, {2/5 (1 + 5 x[0]/2)}}, {2, -1/2}}

In[93]:= EqRecSolve[x[n + 1] == x[n] + 1, n]
Out[93]= {True, x[n] == n + x[0],
  {x, {1}, {n + x[0]}}, {}}

In[94]:= EqRecSolve[x[n + 1] == 2 x[n] + 1, n]
Out[94]= {True, x[n] == -1 + 2^n (1 + x[0]),
  {x, {2, 1}, {1 + x[0]}, {-1}}, {2}}

In[95]:= EqRecSolve[x[n + 2] == x[n] + 1, n]
Out[95]= {True, x[n] == 1/4 (-1)^n (1 + 2 x[0] - 2 x[1]) + 1/4 (-1 + 2 n + 2 x[0] + 2 x[1]),
  {x, {1, -1}, {1/4 (-1 + 2 x[0] + 2 x[1]), 1/2}, {1/4 (1 + 2 x[0] - 2 x[1])}}, {-1}}
Invariant Generation for Loops without Conditionals

For more examples, please look at the AligatorDemo.nb file.

Input: list of inner loops

Output:

 Polynomial invariants for the P-solvable loop of P-solvable inner loops, together with completeness result.

The method is complete if the returned set of invariants are a basis for the polynomial invariant ideal.

If the loops are not P-solvable, execution is aborted

Main steps of the invariant generation process are:
- Transforming loops into recurrences,
- solving recurrences (and computing algebraic dependencies),
- variable elimination.

```
In[180]:=
   L = Body[x := x + 2; y := y + 1]
Out[180]=
   Body[x := x + 2; y := y + 1]

In[181]:=
   InvLoopAssg[L]

   Method is complete!
Out[181]=
   {-x + 2 y + x[0] - 2 y[0]}

In[182]:=
   L = Body[x := 4 x + 2; y := y + 1]
Out[182]=
   Body[x := 4 x + 2; y := y + 1]

In[183]:=
   InvLoopAssg[L]

   Not P-solvable loop! No algebraic dependencies among exponentials!
Out[183]=
   $Aborted

In[184]:=
   L = Body[y := 2 y - 1; x := 2 x]
Out[184]=
   Body[y := 2 y - 1; x := 2 x]

In[185]:=
   InvLoopAssg[L]

   Method is complete!
Out[185]=
   {x + (-1 + y) x[0] - x y[0]}

In[189]:=
   L = Body[t := y; y := y + x; x := t]
Out[189]=
   Body[t := y; y := y + x; x := t]

In[190]:=
   InvLoopAssg[L]

   Method is complete!
Out[190]=
   \{-x^4 - 2 x^3 y + x^2 y^2 + 2 x y^3 - y^4 + (x[0]^2 + x[0] y[0] - y[0]^2)^2\}
```
Invariant Generation for Loops with Conditionals

For more examples, please look at the AligatorDemo.nb file.

Input: list of inner loops
Output:
Polynomial invariants for the P-solvable loop of P-solvable inner loops, together with completeness result.
The method is complete if the returned set of invariants are a basis for the polynomial invariant ideal.

If the loops are not P-solvable, execution is aborted

Main steps of the invariant generation process are:
- Transforming loops into recurrences,
- solving recurrences (and computing algebraic dependencies),
- merging of closed forms,
- computing invariant ideals of sequences of inner loops (by variable elimination)
- invariant filtering on a set of polynomial relations.

```plaintext
In[135]:= L = {Body[x := x + 2; y := y + 1], Body[y := y + 1; x := x + 2]}
Out[135]= {Body[x := x + 2; y := y + 1], Body[y := y + 1; x := x + 2]}

In[136]:= InvLoopCond[L]
    Method is complete!
Out[136]= {-x + 2 y + x[0] - 2 y[0]}

In[133]:= L = {Body[x := 4 x + 2; y := y + 1], Body[y := 2 y - 1; x := 2 x]}
Out[133]= {Body[x := 4 x + 2; y := y + 1], Body[y := 2 y - 1; x := 2 x]}

In[134]:= InvLoopCond[L]
    Not P-solvable loop! No algebraic dependencies among exponentials!
Out[134]= $Aborted

In[137]:= L = {Body[x := x + 2; y := y + 1], Body[y := 2 y - 1; z := 2 z + 1]}
Out[137]= {Body[x := x + 2; y := y + 1], Body[y := 2 y - 1; z := 2 z + 1]}
```
In[138]:= 
InvLoopCond[L]

Out[138]=
{} 

ALIGATOR

For illustrative, textbook examples, please check the AligatorDemo.nb file.

Input: P-solvable loop with/without nested conditionals

Output: polynomial invariants of the loop and completeness report.

The method is complete if the returned set of invariant is a bases of the polynomial invariant ideal.

In case of failure, execution -with error message- is aborted.

Optional argument: IniVal->{initial values}

In[178]:= ?? Aligator

Aligator[WHILE[c1, s1; IF[c2, s2, s3]; s2], IniVal -> {assignments}] 
generates the polynomial invariant of the given loop if the loop is P-solvable. 
The initial values of the variables are optionally specified by IniVal.

Current usage: Aligator[RecEq,VariableTuples, 
SummationVar] where VariableTuple is a list of [x[n],x]

Attributes[Aligator] = {HoldAll}

Aligator[c,., IniVal -> {seq_.}] :=
Module[{invariants = {}, givenIniRecs, givenIniRules}, invariants = Aligator[c];
givenIniRecs = RecEqseq, (8)]
givenIniRules = InitialSubstitutions[givenIniRecs, ()];
Simplify[invariants /., givenIniRules]}

Aligator[c_] := Module[{sw, ifCheck, loops, invariants = {}, sw = InputCheck[c];
If[sw = 0, Abort[]]; ifCheck = CheckIfSeq[c]; loops = IfWhileTransform[c, Body[], Body[]];
If[!ifCheck, invariants = InvLoopAsg[loops], invariants = InvLoopCond[loops]];
Simplify[And@@(#1 = 0 &) /@ invariants]}

Options[Aligator] = {IniVal -> {}}

In[179]:= ?? IniVal

Option to Aligator for specifying initial values of variables.
- **Aligator - top level command**

In[153]:=
\[\text{Aligator[\text{WHILE}}[g, u; v]]\]

\text{aligator::InputError2 :}
Not appropriate Aligator\(\ldots\)input must be of the form \text{WHILE[loop-condition, loop-body]}. Given loop-body has to be a sequence of the form: \text{s0, IF[test, s1, s2]}, \text{s3}, where \text{s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without \\#)}.

Hint: check also for spellings (\text{e.g. := instead of =, IF instead of If, ; instead of ,})!

Out[153]=
$\text{Aborted}$

In[154]:=
\[\text{Aligator[\text{WHILE}}[g, \text{IF}[y, p, t]; i := 1]]\]

\text{aligator::InputError2 :}
Not appropriate Aligator\(\ldots\)input must be of the form \text{WHILE[loop-condition, loop-body]}. Given loop-body has to be a sequence of the form: \text{s0, IF[test, s1, s2]}, \text{s3}, where \text{s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without \\#)}.

Hint: check also for spellings (\text{e.g. := instead of =, IF instead of If, ; instead of ,})!

Out[154]=
$\text{Aborted}$

In[155]:=
\[\text{Aligator[5]}\]

\text{aligator::InputError4 :}
Not appropriate Aligator\(\ldots\)input must be of the form \text{WHILE[loop-condition, loop-body]}. Given input must be a \text{WHILE} loop!

Out[155]=
$\text{Aborted}$

In[156]:=
\[\text{Aligator[WHILE}}[g1; g2]\]

\text{aligator::InputError1 :}
Not appropriate Aligator\(\ldots\)input. Input must be of the form \text{WHILE[loop-condition, loop-body]}. Given loop condition is not appropriate – it has to be a NON-EMPTY BOOLEAN CONDITION (without \\#)!

Out[156]=
$\text{Aborted}$

In[157]:=
\[\text{Aligator[c; WHILE}}[b, h]\]

\text{aligator::InputError4 :}
Not appropriate Aligator\(\ldots\)input. Input must be of the form \text{WHILE[loop-condition, loop-body]}. Given input must be a \text{WHILE} loop!

Out[157]=
$\text{Aborted}$
In[158]:=  
\text{Aligator}[\text{WHILE}[g_1, u, v]]  

\text{aligator::InputError3 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{WHILE has 2 arguments only!}  
Out[158]=  
\$Aborted  

In[159]:=  
\text{Aligator}[\text{WHILE}[g_1, u, v, k]]  

\text{aligator::InputError3 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{WHILE has 2 arguments only!}  
Out[159]=  
\$Aborted  

In[160]:=  
\text{Aligator}[\text{WHILE}[g, u; u, k]]  

\text{aligator::InputError3 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{WHILE has 2 arguments only!}  
Out[160]=  
\$Aborted  

In[161]:=  
\text{Aligator}[\text{WHILE[]}]]  

\text{aligator::InputError4 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{Given input must be a WHILE loop!}  
Out[161]=  
\$Aborted  

In[162]:=  
\text{Aligator}[\text{WHILE}[u, j]]  

\text{aligator::InputError2 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{Given loop-body has to be a sequence of the form: s_0; IF[test, s_1, s_2]; s_3, where } s_0, s_1, s_3 \text{ are SEQUENCES OF ASSIGNMENTS, s_2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without ::).}  
\text{Hint: check also for spellings [e.g. := instead of =, IF instead of If, ; instead of ,]!}  
Out[162]=  
\$Aborted  

In[163]:=  
\text{Aligator}[c]  

\text{aligator::InputError4 :}  
Not appropriate Aligator-input.\text{Input must be of the form WHILE[loop-condition, loop-body].}  
\text{Given input must be a WHILE loop!}  
Out[163]=  
\$Aborted
In[164]:= 

Aligator[WHILE[g, u]]

aligator::InputError2 : 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :-)! 
Hint: check also for spellings (e.g. := instead of =, IF instead of If, ; instead of ,)! 

Out[164]= 
$Aborted

In[165]:= 

Aligator[WHILE[g, u := s]]

No recursively changed variables! Not P-solvable Loop!

Out[165]= 
$Aborted

In[167]:= 

Aligator[WHILE[g, u := u]]

aligator::InputError2 : 
Not appropriate Aligator-input. Input must be of the form WHILE[loop-condition, loop-body]. Given loop-body has to be a sequence of the form: s0; IF[test, s1, s2]; s3, where s0, s1, s3 are SEQUENCES OF ASSIGNMENTS, s2 is a SEQUENCE OF ASSIGNMENTS or IF-STATEMENT, and all conditions are NON-EMPTY BOOLEAN CONDITIONS (without :-)! 
Hint: check also for spellings (e.g. := instead of =, IF instead of If, ; instead of ,)! 

Out[167]= 
$Aborted

In[168]:= 

Aligator[WHILE[... , x := x + 1; u := u + 9]]

Method is complete!

Out[168]= 
u + 9 x[0] = 9 x + u[0]

In[169]:= 

Aligator[WHILE[... , x := 2 x + 1; u := 1/2 u + 9]]

Method is complete!

Out[169]= 
u + u x + 18 x[0] = 18 x + 1[0] + 1[0] x[0]

In[170]:= 

Aligator[WHILE[... , x := 2 x + 1; u := 1/2 u + 9], IniVal \rightarrow \{x := 0; u := 1\}]

Method is complete!

Out[170]= 
u + u x = 18 x + 1[0]

In[171]:= 

Aligator[WHILE[g, u := u + 1]]

Method is complete!

Out[171]= 
True
In[172]:=
    Aligator[WHILE[g, u := 1; IF[x, h := 0]]]
    No recursively changed variables! Not P-solvable Loop!
Out[172]=
    $Aborted

In[173]:=
    Aligator[WHILE[a > 0, IF[b, c := 1]]]
    No recursively changed variables! Not P-solvable Loop!
Out[173]=
    $Aborted

In[174]:=
    Aligator[WHILE[a > 0, IF[b, c := 1, IF[,..., d := d + 4, j := 7]]] // Timing
    No recursively changed variables! Not P-solvable Loop!
Out[174]=
    $Aborted

In[176]:=
    Aligator[WHILE[,..., x := 2 x; y := 1/2 + y + 1]]
    Method is complete!
Out[176]=
    x (-2 + y) = x[0] (-2 + y[0])

In[177]:=
    Aligator[WHILE[,..., x := 2 x; y := 1/2 + y + 1], IniVal → {x := 1; y := 0}]
    Method is complete!
Out[177]=
    x (-2 + y) = -2