Regular path clauses and their application in solving loops

Bishoksan Kafle[1] **John P. Gallagher**[1,2] Manuel Hermenegildo[1,3]
Maximiliano Klemen[1] Pedro López-García [1] José F. Morales[1,3]

[1]IMDEA Software Institute, [2]Roskilde University, and [3]T.U. Madrid (UPM)

HCVS'21, Virtual

Motivation

Standard approach for solving loops:

- extract recurrences from the loop, and
- solve them to get a closed-form expression (possibly) using a (combination of) Computer Algebra Systems (CASs).

Resource analysis: [Wegbreit Comm. of the ACM'74, Debray et al. PLDI'90 and TOPLAS'93, Navas et al. ICLP'07, Albert et al. TOCL'13].

Invariant synthesis: [Farzan et al. FMCAD'15, Kincaid et al. POPL'18-POPL'19, Humenberger et al. VMCAI'18].

Computer Algebra Systems (CASs)

Pros:

- Can derive non-linear functions including polynomial, exponential, logarithmic, . . .
- Can produce very precise solutions for some classes of recurrences.

Cons:

- Can only solve a subset of all possible recurrences.
- Typically:
  - Recurrences with a single recursive case.
  - Recurrences involving univariate functions.

Program recurrences and Computer Algebra Systems

Recurrences derived from programs may not be *solvable* by CASs:

- Usually have multiple paths (if ... then ... else inside a loop) $\longrightarrow$ multiple recursive cases.
- Manipulate multiple variables $\longrightarrow$ multivariate recurrences.

Goal

Use CASs to solve program loops (infer loop invariants) by:

- Systematically transforming programs, expressed as constrained Horn clauses (CHCs), to obtain recurrences that are solvable by CASs

Example: a program and its CHC representation

```
int a, b; //input
while (a > 0) {
if (b > 0) then
    b − −;
else   b = b + a;
    a − −; }
```

(a) Ex. program.

c_1: $\mathrm{wh}(a, b) \leftarrow a > 0, b > 0, \mathrm{wh}(a, b-1).$
c_2: $\mathrm{wh}(a, b) \leftarrow a > 0, b \leq 0, \mathrm{wh}(a-1, b+a).$
c_3: $\mathrm{wh}(a, b) \leftarrow a \leq 0.$

(b) CHCs (c_i is a clause identifier).

- It exhibit a multi-path loop (with paths c_1 and c_2).
- It manipulates two variables, a, b.

$\longrightarrow$ CASs cannot be directly applied.

Overview

1. Multi-path loops $\longrightarrow$ single-path loops
2. Multi-argument functions $\longrightarrow$ single-argument functions

Removing multi-path loops: Procedure

Multi-path loops $\longrightarrow$ single-path loops

Input: a CHC program P and its CFG (w/ entry and exit nodes).
Output: an "equivalent" CHC program P' without multi-path loops.
Process sketch:

1. Compute a regexp e describing all paths from entry to exit (using Tarjan's alg.)
2. Transform e into an equivalent regexp e' without $+$ op. within a $*$.
3. Construct path clauses P' using P and e'; return P'.

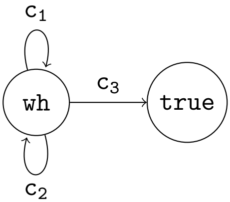
Related work:

- Merge paths: [Albert et al. TOCL'13, Farzan et al. FMCAD'15, Kincaid et al. POPL'18-POPL'19].
- Deal with each path separately and combine their polynomial ideals: [Humenberger et al. VMCAI'18].
- Control-flow refinement: [Sharma et al. CAV'11, Puebla et al. JLP'99, Doménech et al. TPLP'19].

- Program paths can be described by regular expressions, eg.:

  $(c_1 + c_2)^* c_3$ describes all paths through the following loop

  c_1: $\mathrm{wh}(a, b) \leftarrow a > 0, b > 0, \mathrm{wh}(a, b - 1).$
  c_2: $\mathrm{wh}(a, b) \leftarrow a > 0, b \leq 0, \mathrm{wh}(a - 1, b + a).$
  c_3: $\mathrm{wh}(a, b) \leftarrow a \leq 0.$



- Regular expressions can be transformed into other equivalent expressions, e.g.:

$$(c_1 + c_2)^* c_3 \equiv c_1^*(c_2 c_1^*)^* c_3 \equiv c_2^*(c_1 c_2^*)^* c_3.$$

Construction of path clauses corresponding to * expression

Given c_1: $\text{wh}(a, b) \leftarrow \underbrace{a > 0, b > 0, a' = a, b' = b - 1}_{\phi}, \text{wh}(a', b')$, the path clauses given by c_1^* starting from node wh are:

$$\begin{array}{lll} path_{c_1^*}(wh(a,b), wh(a,b)) & \leftarrow \text{true}. & \text{(note: } e^* = e^*e) \\ path_{c_1^*}(wh(a,b), wh(a'',b'')) & \leftarrow path_{c_1^*}(wh(a,b), wh(a',b')), & \\ & \quad path_{c_1}(wh(a',b'), wh(a'',b'')). & \end{array}$$

OR

$$\begin{array}{lll} path_{c_1*}(wh(a,b), wh(a,b)) & \leftarrow \text{true}. & \text{(note: } e^* = ee^*) \\ path_{c_1*}(wh(a,b), wh(a'',b'')) & \leftarrow path_{c_1}(wh(a,b), wh(a',b')), & \\ & \quad path_{c_1^*}(wh(a',b'), wh(a'',b'')). & \end{array}$$

where $path_{c_1}(wh(a,b), wh(a',b')) = \phi$

Simplification and renaming of path clauses

Given

$$
\begin{array}{lll}
path_{c_1^*}(wh(a,b), wh(a,b)) & \leftarrow & \text{true}. \\
path_{c_1^*}(wh(a,b), wh(a'',b'')) & \leftarrow & path_{c_1^*}(wh(a,b), wh(a',b')), \\
 & & path_{c_1}(wh(a',b'), wh(a'',b'')). \\
 & & \\
path_{c_1}(wh(a',b'), wh(a'',b'')) & \leftarrow & a' > 0, b' > 0, a'' = a', b'' = b' - 1.
\end{array}
$$

Renaming and simplifying we obtain:

$$
\begin{array}{lll}
\mathtt{wh}_2(\mathtt{a},\mathtt{b},\mathtt{a},\mathtt{b}) & \leftarrow & \text{true}. \\
\mathtt{wh}_2(\mathtt{a},\mathtt{b},\mathtt{a}',\mathtt{b}'-1) & \leftarrow & \mathtt{wh}_2(\mathtt{a},\mathtt{b},\mathtt{a}',\mathtt{b}'),\ \mathtt{a}' > 0,\ \mathtt{b}' > 0.
\end{array}
$$

Construction of path clauses corresponding to an expr. e

c	$path_c(p(\mathbf{x}), q(\mathbf{x}')) \leftarrow \phi.$, where clause $p(\mathbf{x}) \leftarrow \phi, q(\mathbf{x}') \in P$ has identifier c
ϵ	$path_\epsilon(p(\mathbf{x}), p(\mathbf{x})) \leftarrow$ true.
$\emptyset$	no clause
$e_1 e_2$	$path_{e_1 e_2}(p(\mathbf{x}), z) \leftarrow path_{e_1}(p(\mathbf{x}), q(\mathbf{x}')), path_{e_2}(q(\mathbf{x}'), z).$, for each $q \in firstpred(e_2)$
$e_1 + e_2$	$path_{e_1+e_2}(p(\mathbf{x}), z) \leftarrow path_{e_1}(p(\mathbf{x}), z).$ $path_{e_1+e_2}(p(\mathbf{x}), z) \leftarrow path_{e_2}(p(\mathbf{x}), z).$
e^*	$path_{e^*}(p(\mathbf{x}), p(\mathbf{x})) \leftarrow$ true. $path_{e^*}(p(\mathbf{x}), p(\mathbf{x}'')) \leftarrow path_{e^*}(p(\mathbf{x}), p(\mathbf{x}')), path_e(p(\mathbf{x}'), p(\mathbf{x}'')).$

Path clauses for the example program

c_1: $\mathrm{wh}(a, b) \leftarrow a > 0, b > 0, \mathrm{wh}(a, b-1).$
c_2: $\mathrm{wh}(a, b) \leftarrow a > 0, b \leq 0, \mathrm{wh}(a-1, b+a).$
c_3: $\mathrm{wh}(a, b) \leftarrow a \leq 0.$

Path clauses based on $(c_1^*(c_2 c_1^*)^*)c_3$

$\mathtt{path}(\mathtt{wh}(\mathtt{a}, \mathtt{b}), \mathtt{true}) \leftarrow \mathtt{wh}_2(\mathtt{a}, \mathtt{b}, \mathtt{a}', \mathtt{b}'), \mathtt{wh}_5(\mathtt{a}', \mathtt{b}', \mathtt{a}'', \mathtt{b}''),\ \mathtt{a}'' \leq 0.$

$\mathtt{wh}_2(\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{b}) \leftarrow \mathtt{true}.$
$\mathtt{wh}_2(\mathtt{a}, \mathtt{b}, \mathtt{a}', \mathtt{b}' - 1) \leftarrow \mathtt{wh}_2(\mathtt{a}, \mathtt{b}, \mathtt{a}', \mathtt{b}'),\ \mathtt{a}' > 0,\ \mathtt{b}' > 0.$

$\mathtt{wh}_5(\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{b}) \leftarrow \mathtt{true}.$
$\mathtt{wh}_5(\mathtt{a}, \mathtt{b}, \mathtt{a}'', \mathtt{b}'') \leftarrow \mathtt{wh}_5(\mathtt{a}, \mathtt{b}, \mathtt{a}', \mathtt{b}'),\ \mathtt{a}' > 0,\ \mathtt{b}' \leq 0, \mathtt{wh}_2(\mathtt{a}' - 1, \mathtt{b}' + \mathtt{a}', \mathtt{a}'', \mathtt{b}'').$

Result: multi-paths $\longrightarrow$ nested single-paths

Overview

1. Multi-path loops $\longrightarrow$ single-path loops
2. Multi-argument functions $\longrightarrow$ single-argument functions

Procedure

1. Instrument path clause with a counter k.
   - Given $\mathtt{path}_{e^*}(\mathbf{x}, \mathbf{x_2}) \leftarrow \phi(\mathbf{x_1}, \mathbf{x_2}), \mathtt{path}_{e^*}(\mathbf{x}, \mathbf{x_1})$.
   - Path clauses with a counter k are:

$$\mathtt{path}_{e^*}(\mathrm{k}, \mathbf{x}, \mathbf{x_2}) \leftarrow \mathrm{k} > 0, \phi(\mathbf{x_1}, \mathbf{x_2}), \mathtt{path}_{e^*}(\mathrm{k}-1, \mathbf{x}, \mathbf{x_1}).$$
$$\mathtt{path}_{e^*}(\mathrm{k}, \mathbf{x}, \mathbf{x}) \leftarrow \mathrm{k} = 0.$$

2. Then, given an input tuple $\mathbf{x}$ and k, set up equations for output tuple $\mathbf{x_2}$ using the path clause, using methods such as in (Debray et al., TOPLAS'93).
3. Detect and remove symbolic constants from the equations; obtaining single-argument functions.
4. Solve the resulting equations using CASs and replace k in the solution with the value of the ranking function in the initial state of the original loop clause.

Multi-args $\longrightarrow$ single-arg: [Farzan et al. FMCAD'15, Albert et al. TOCL'13]

Example

Given $\mathtt{wh}(x_1, y_1) \leftarrow x_1 > 0, y_1 > 0, x_2 = x_1 - 1, y_2 = y_1 + x_1, \mathtt{wh}(x_2, y_2)$, the path clauses with a counter k are:

$$
\begin{array}{lcl}
\mathtt{path}(k, x, y, x_2, y_2) & \leftarrow & k > 0, k_1 = k - 1, x_2 = x_1 - 1, y_2 = y_1 + x_1, \\
 & & \mathtt{path}(k_1, x, y, x_1, y_1), \\
 & & x_1 > 0, y_1 > 0. \\
\mathtt{path}(k, x, y, x, y) & \leftarrow & k = 0.
\end{array}
$$

The outputs x_2, y_2 represent the values of x, y after k iterations of the loop and is completely determined by the inputs k, x, y.

Extracted Recurrences

Hence, given

$$\begin{array}{lll}\mathtt{path}(\mathtt{k},\mathtt{x},\mathtt{y},\mathtt{x}_2,\mathtt{y}_2) & \leftarrow & \mathtt{k}>0,\mathtt{k}_1=\mathtt{k}-1,\mathtt{x}_2=\mathtt{x}_1-1,\mathtt{y}_2=\mathtt{y}_1+\mathtt{x}_1,\\ & & \mathtt{path}(\mathtt{k}_1,\mathtt{x},\mathtt{y},\mathtt{x}_1,\mathtt{y}_1),\mathtt{x}_1>0,\mathtt{y}_1>0.\end{array}$$

we obtain the following recurrences

$$\mathtt{wh}^{\mathtt{x}}(k,x,y)=\begin{cases}\mathtt{wh}^{\mathtt{x}}(k-1,x,y)-1, & \textit{for} \quad k>0,\\ x, \quad \textit{for} \quad k=0\end{cases}$$

$$\mathtt{wh}^{\mathtt{y}}(k,x,y)=\begin{cases}\mathtt{wh}^{\mathtt{y}}(k-1,x,y)+\mathtt{wh}^{\mathtt{x}}(k-1,x,y), & \textit{for} \quad k>0,\\ y, \quad \textit{for} \quad k=0\end{cases}$$

where $\mathtt{wh}^{\mathtt{v}}(k,x,y)$ defines the values of $\mathtt{v}$ after k iterations of the $\mathtt{wh}$ loop.

Since the recurrences involve multi-argument functions, they cannot be solved by the CASs.

Multi-args functions to single-arg functions

Using data-flow analysis (reaching definitions analysis), we can detect that x, y are symbolic constants in the following equation.

$$\mathtt{wh}^{\mathtt{x}}(k, x, y) = \begin{cases} \mathtt{wh}^{\mathtt{x}}(k-1, x, y) - 1, & \textit{for} \quad k > 0, \\ x, \quad \textit{for} \quad k = 0 \end{cases}$$

since they cannot affect the solution of the equations

- remove them as arguments, and
- replace their occurrence elsewhere by a constant function returning their values.

Solving recurrences for $\mathtt{wh}^{\mathtt{x}}(k, x, y)$

$$\mathtt{wh}^{\mathtt{x}}(k) = \begin{cases} \mathtt{wh}^{\mathtt{x}}(k-1) - 1, & \textit{for} \quad k > 0, \\ c_x, & \textit{for} \quad k = 0 \end{cases}$$

- Can be solved using existing CASs, obtaining $\mathtt{wh}^{\mathtt{x}}(k) = c_x - k$ as a closed-form solution.
- This is also the solution of the original equation, thus we have $\mathtt{wh}^{\mathtt{x}}(k, x, y) = x - k$.

Solving recurrences for $\mathtt{wh}^{\mathtt{y}}(k, x, y)$

$$\mathtt{wh}^{\mathtt{y}}(k, x, y) = \begin{cases} \mathtt{wh}^{\mathtt{y}}(k-1, x, y) + \mathtt{wh}^{\mathtt{x}}(k-1, x, y), & \textit{for} \quad k > 0, \\ y, \quad \textit{for} \quad k = 0 \end{cases}$$

Reusing the solution of $\mathtt{wh}^{\mathtt{x}}(k, x, y)$, we obtain

$$\mathtt{wh}^{\mathtt{y}}(k, x, y) = \begin{cases} \mathtt{wh}^{\mathtt{y}}(k-1, x, y) + x - k + 1, & \textit{for} \quad k > 0, \\ y, \quad \textit{for} \quad k = 0 \end{cases}$$

Since x, y are symbolic constants, we get

$$\mathtt{wh}^{\mathtt{y}}(k) = \begin{cases} \mathtt{wh}^{\mathtt{y}}(k-1) + c_x - k + 1, & \textit{for} \quad k > 0, \\ c_y, \quad \textit{for} \quad k = 0 \end{cases}$$

- which can be solved to yield $\mathtt{wh}^{\mathtt{y}}(k) = c_y - 1/2k(k - 2x - 1)$ and hence $\mathtt{wh}^{\mathtt{y}}(k, x, y) = y - 1/2k^2 + kx + 1/2k$.

What is the value of the counter variable k?

- Assuming the *wh* loop has a ranking function, $k \in [0, r]$ where r is the value of ranking function in the initial state.
- assume wh(x1,y1) is the call at the start of the loop. Then the loop executes $k \in [0, x1]$ times.
- Thus the final value of x_1 and x_2 resp. are $[0, x_1]$ and $[y_1 - 1/2x_1^2, y_1 + 1/2x_1 + x_1^2]$; obtained using interval arithmetic.

Putting it all together

We achieved

- Multi-path loops $\longrightarrow$ Single-path loops (through transformation of regular expressions).
- Multi-argument functions $\longrightarrow$ single-argument functions (through counter instrumentation and detection and removal of symbolic constants).

Discussion and future work

Pros:

- Besides overcoming these limitations of CASs, also ensures that the resulting recurrences.
  - Contain no mutual recursion.
  - Contain only a decreasing argument(k).

Cons:

- Shifts the problem to finding bounds on k.
- Success of our method depends on external tools such as ranking function synthesizers and CASs.
- The use of interval arithmetic to infer bounds on variable values can result in imprecision.
- Transforming a given program into a multi-path loop could cause an exponential blow-up in size.

Discussion and future work

Future:

- Investigate the choice of regular expressions.
- Extend our approach to extract and solve recurrence inequations.
- Extend it to handle non-linear loops.

The end!

Thanks for your attention!