# Precise Dependence Partitioning Algoritm

Archana Kale*   Supratim Biswas†
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

IS is the Iteration Space of a Loop. This algorithms generates PDP partition as Set of Components i.e. $P = \{C\}$. Each component is a Set of Dependent Iterations $i \in IS = [l, u]$. This solves LDE $ax + by = c$. The inputs to the algorithm are:

- $a \Rightarrow$ coefficient of $x$.
- $b \Rightarrow$ coefficient of $y$.
- $c \Rightarrow$ constant on the RHS for LDE.
- $l \Rightarrow$ Lower Bound.
- $u \Rightarrow$ Upper Bound.

## 1 Formation of Components

The objective of this section is to formulate the structure of a $C$ and relationship between its constituent iterations. The solutions, to LDE in parametric form, are $\{(\alpha - mb), (\alpha + ma)\}$, where m is an integer and $\{\alpha, \alpha\}$ is a particular solution form $C$s of length 2.

Longer $C$s are formed if $m$ is a multiple of $a$ or $b$ ($m = qa$) or ($m = qb$). As $(\alpha + qab)$ is a common iteration of $\{(\alpha - qbb), (\alpha + qab)\}$ and $\{(\alpha + qab), (\alpha - qaa)\}$ longer $C = \{(\alpha - qbb), (\alpha + qab), (\alpha - qaa)\}$ is formed.

The general structure of $C$s of length $l+1$ where $m$ is not a multiple of $a$ or $b$ is: $\{(\alpha \pm ma^l), (\alpha \mp ma^{l-1}b), (\alpha \pm ma^{l-2}b^2), ....., (\alpha \pm (-1)^{l-2}.ma^2b^{l-2}), (\alpha \pm (-1)^{l-1}.mab^{l-1}), (\alpha \pm (-1)^l.mb^l)\}$

An iteration $i$, of a component $C$, is defined to be a seed of $C$ if all the iterations of $C$ can be generated using $i$. Seed is a representative iteration of a $C$ which can generate it. For a single LDE single LOOP case, all iterations are seeds of corresponding $C$s. If $\alpha$ is not an integer, $C$ and seeds are computed using particular solution $\{\beta, \gamma\}$ of LDE as: $\{(\beta - mb), (\gamma + ma)\}$.

## 2 PD for 2 variable LDE

The following 2 results, known for 2 variable LDE L and its solution $S$, provide the basis for the correctness of PDP for Type 1 LDEs.

*archanak@cse.iitb.ac.in
†sb@cse.iitb.ac.in
Algorithm for PDP submitted to PACT 2015
Draft—Do not Distribute

1. All the dependent iterations in L are elements of $S$. (i.e if iterations $i_1$ and $i_2$ are dependent then $(i_1, i_2) \in S$ or $(i_2, i_1) \in S$.
2. All elements in $S$ represent dependent iterations in L. i.e if $(i_1, i_2) \in S$ or $(i_2, i_1) \in S$ then iterations $i_1$ and $i_2$ are dependent.

# 3 Non-Integer $\alpha$

If $\alpha = c/(a+b)$ exists and is not an integer, solutions for $ax + by = c$ can be computed from specific solutions of $ax + by = 0$.

Solution of $ax + by = c(a + b)$ can be translated from solutions of solution of $ax + by = 0$ for $\alpha = c$.

Result to be proved:
If $S^0$ is the solution of $ax + by = c$ where, $(a + b) \not| c$
and
If $S^t$ is the solution of $ax + by = c(a + b)$, then establish relation between $S^t$ and $S^0$.

PROOF of $S^t = f_1(S^0)$
Let $(x', y') \in S^0$, then $(x'(a + b), y'(a + b)) \in S^t$.
$\therefore \forall\ (x', y') \in S^0$, $(x'(a + b), y'(a + b)) \in S^t$.
$f_1$ is one-to-one function.

PROOF of $S^0 = f_2(S^t)$
Let $(x", y")$ be a solution of $ax + by = c(a + b)$.
If $((a + b)|x")$ AND $((a + b)|y")$ then $(x"/(a + b), y"/(a + b))$ will be a solution of $ax + by = c$.
Function $f_2$ is defined only for a subset of domain$S^t$.

Thus all solutions of 2 variable LDEs having $\alpha \notin Z$ can be obtained from $ax + by = 0$.

# 4 Algorithm for All-C-Size2

This is the case when $a * b == 1$, i.e. the LDE is of form $x + y = c$. The Steps are:

1. $k_1 = c/2$.
2. if $(k_1 \in Z)$, $k_2 = k_1$.
   else $k_2 = k_1 + 0.5, k_1 = k_1 - 0.5$.
3. Add $\{k_1, k_2\}$ to $P$.
4. $d = min(|c/2 - l|, |u - c/2|)$.
5. for $i = 1$ to $d$ Add $\{k_1 - i, k_2 + i\}$ to $P$.
6. Add remaining iterations as singular sets of components to $P$ as they are all independent.
7. Return $P$, Goto Step 14 in Algorithm 9.

## 5 Algorithm for All-C-spirals

This is the case when $a$ and $b$ have same sign. The components have iterations from both sides of $\alpha$.

1. if ($\alpha \in Z$)
   (a) Translate iteration space $B = [l, u] = [l - \alpha, u - \alpha]$
   (b) Generate chains as per formula in section 1 for $B$ and add them to $P'$.
   (c) $P" = \forall\ C \in P'$ Add $C" \in P"$ such that $\forall\ i \in C$ Add $-i \in C"$.
   (d) Add $P' + \{0, 0\} + P"$ to $P$.
   (e) Return $P$, Goto Step 13 in Algorithm 9.
2. else $B = [l, u]$
   (a) Initialize 2D array I[B][2] as follows
   (b) for j = l to u { I[j-l][0] = ((j-l)*(a+b)); I[j-l][1] = 1)}
3. Initialize $P'$ to empty.
4. $i = 1$, $cno = 1$, set $I[i][1] = 0$.
5. While( some element in $I[i][1]$ is equal to 1):
   (a) Add $I[i] \in C_{cno}$, $Chain(i, C_{cno})$.
   (b) $i = nextelement()$.
6. $P" = \forall\ C \in P'$ Add $C" \in P"$ such that $\forall\ i \in C$ Add $-i \in C"$.
7. Add $P' + \{0, 0\} + P"$ to $P$.
8. $\forall\ i \in C \in P\ i \Leftarrow i/(a + b)$
9. Return $P$, Goto Step 13 in Algorithm 9.

## 6 Algorithm for Chain

Use formula given in section 1 to generate components. Set corresponding element in $I$ to zero.

## 7 Algorithm for GenComp

This is the case when $a$ and $b$ have different sign. The components have all iterations on same side of $\alpha$.
Input is B the size of range to be used to compute $P'$

1. Let $I$ be a boolean array of size $B$ initialized to 1.
2. $i = 1$, $cno = 1$, set $I[i] = 0$.
3. While( some element in $I$ is equal to 1):
   (a) Add $i \in C_{cno}$, $Chain(i, C_{cno})$.
   (b) $i = nextelement()$.
4. Return $P$, Goto Step 12d in Algorithm 9.

# 8 Algorithm for Gen-nonint

This is the case when $\alpha$ is not an integer.

1. $B = max((\alpha - l), (u - \alpha))$.
2. Initialize 2D array I[B][2] as follows
   (a) for j = 1 to B { I[j][0] = (j*(a+b)); I[j][1] = 1)}
3. $i = 1$, $cno = 1$, set $I[i][1] = 0$.
4. While( some element in $I[i][1]$ is equal to 1):
   (a) Add $I[i] \in C_{cno}$, $Chain(i, C_{cno})$.
   (b) $i = nextelement()$.
5. $\forall\ i \in C \in P\ i \Leftarrow i/(a+b)$
6. Return $P$, Goto Step 13 in Algorithm 9.

# 9 Algorithm for 2 variable LDE

**Inputs:** $a, b, c, l, u$.

**Output:** $P = \{C\} = \{\{i\}\}$.

**Steps:**

1. if ($!a$ OR $!b$) $\Rightarrow$ "No LDE"
2. Initialize $P$ = empty.o
3. if ($!a$ & $!b$) $\Rightarrow P = \{\{l, l+1, l+2, .., u\}\}$ ("All iterations are dependent"), Goto Step 14.
4. if ($gcd(a, b) \not| c$) $\Rightarrow$ "No Solutions".
5. a /= gcd(a,b). b /= gcd(a,b). c /= gcd(a,b).
6. if ($a*b == -1$) & $!c \Rightarrow P = \{\{l\}, \{l+1\}, \{l+2\}, .., \{u\}\}$ ("All iterations are independent"), Goto Step 14.
7. if ($a*b == -1$) $\Rightarrow P = \{\{l, l+c, l+2c, ..\}, \{i+1, l+1+c, l+1+2c, ..\}, ..\{l+c-1, l+2c-1, ..\}\}$ ("Constant dependence"), Goto Step 14.
8. if ($a * b == 1$) $\Rightarrow P = All - C - size2()$.
9. if $c \neq 0$ Compute Loop independent iteration $\alpha = c/(a+b)$.
   else $\alpha = 0$.
10. if ($a * b > 0$) $\Rightarrow P = All - C - spirals()$.
11. if ($\alpha \notin Z$), Gen-nonint().

12. else
    (a) Compute range $R' = [1, max(|\alpha - l|, |u - \alpha|)]$.
    (b) $P' = GenComp(R')$ (Single sided part of $P$).
    (c) Generate other part of $P$ i.e. $P$" by translating every iteration of every $C' \in P'$ as $\forall i, i \in C', -i \in C$".
    (d) Add $P' + \{0\} + P$" to $P$.
13. If($\alpha$) Translate $P$ by $\alpha$ i.e. for every $C \in P$, $\forall i, i \in C, i \Leftarrow i + \alpha$.
14. Output $P$.