A General Approach of Infeasibility in ILP-based WCET Estimation Methods

Pascal Raymond

Verimag/CNRS

Grenoble-Alpes University

SYNCHRON14, Aussois, Dec. 2014

Presented at EMSOFT 14

Supported by the French ANR project W-SEPT

WCET estimation

Principles

- Computes a safe upper bound to the worst case execution time
- Relevance: performed at the binary level, for a know architecture
- Main problems/challenges for accuracy:

$\hookrightarrow$ Precise modeling of the micro-architecture

$\hookrightarrow$ Reject infeasible executions

State of the art organization

- Input: Control Flow graph (CFG) of the binary code, whose vertices are Basic Blocks of sequential instructions (BB)

- Mainly 3 steps:

  $\hookrightarrow$ Data-flow analysis: find semantic information, in order to prune infeasible executions, at least: find loop bounds to reject infinite executions

  $\hookrightarrow$ Micro-architecture analysis: mainly local to avoid intractability, assigns local “weights” to each BB and/or transition (expressed in CPU cycles)

  $\hookrightarrow$ Search of the worst (heaviest) path in the CFG annotated with the local weights
  Here: we focus on this step

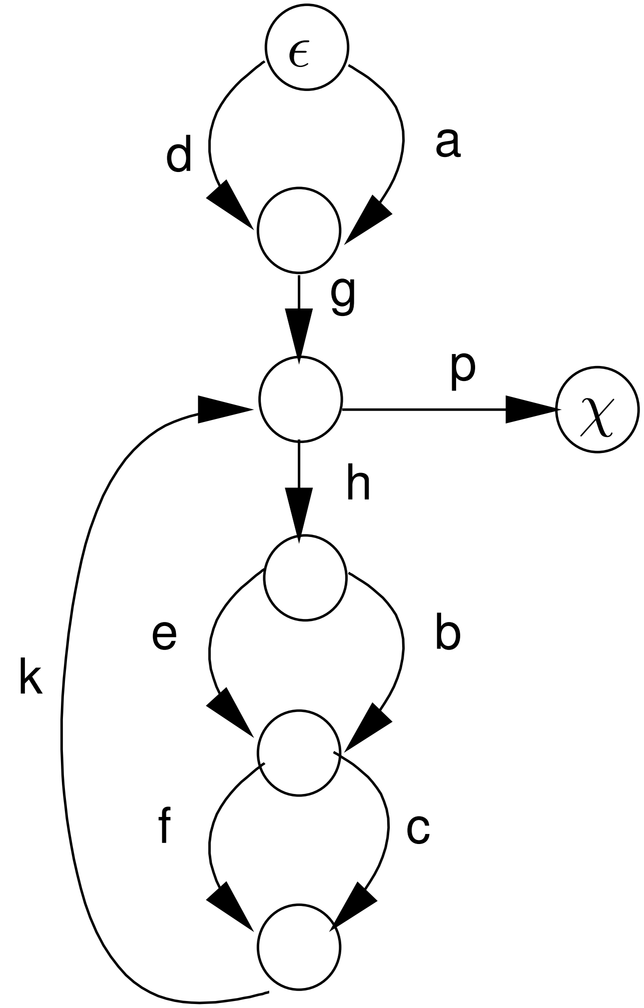
- Implicit Path Enumeration Technique:

  $\hookrightarrow$ Encodes Path search as a numerical optimization problem

  $\hookrightarrow$ Use Integer Linear Programming (ILP) techniques

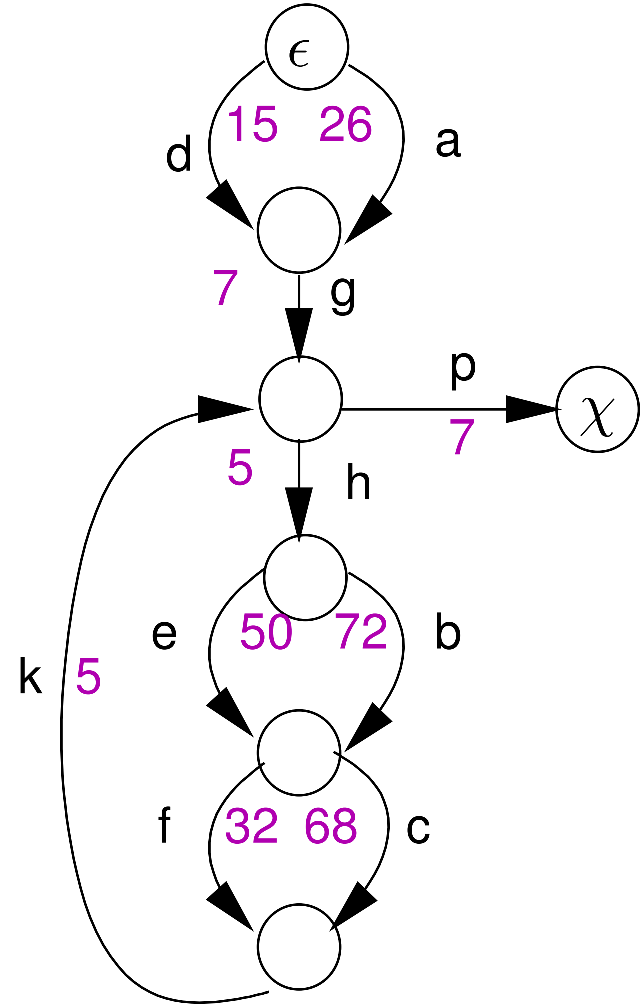
ILP encoding on an example

- μ-archi analysis has assigned weights



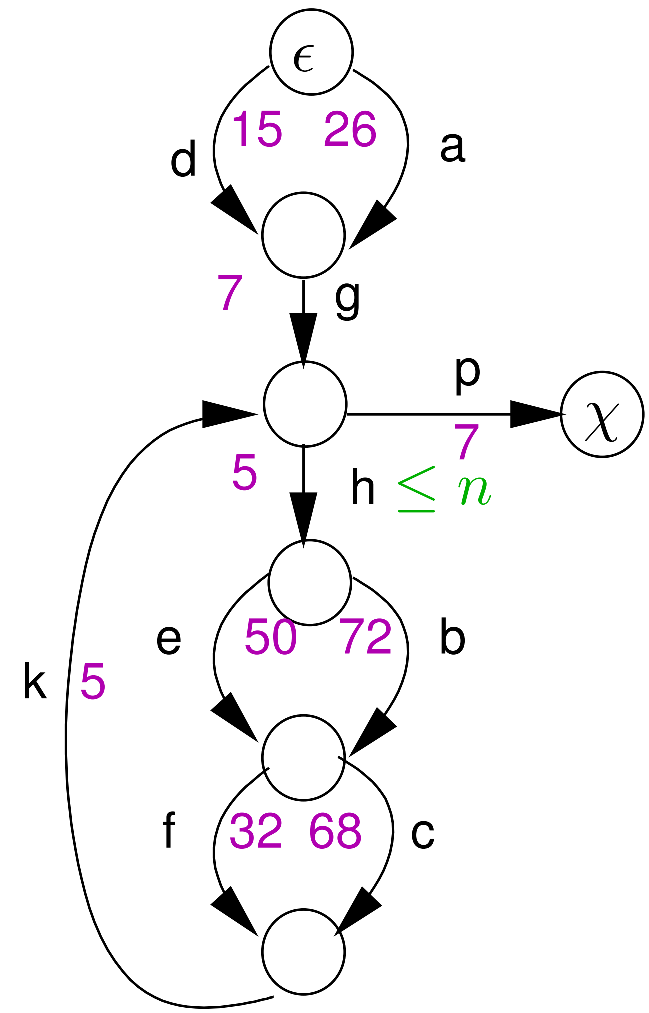
ILP encoding on an example

- μ-archi analysis has assigned weights
  e.g. $w_a = 26$, $w_b = 72$ etc.



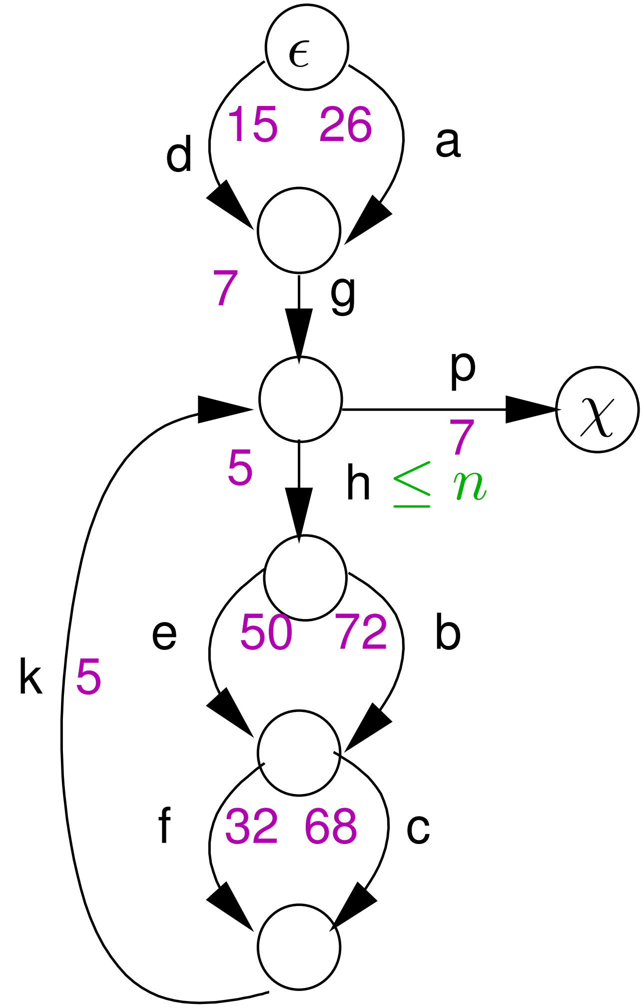
ILP encoding on an example

- μ-archi analysis has assigned weights e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds 'h' taken at most n times



ILP encoding on an example

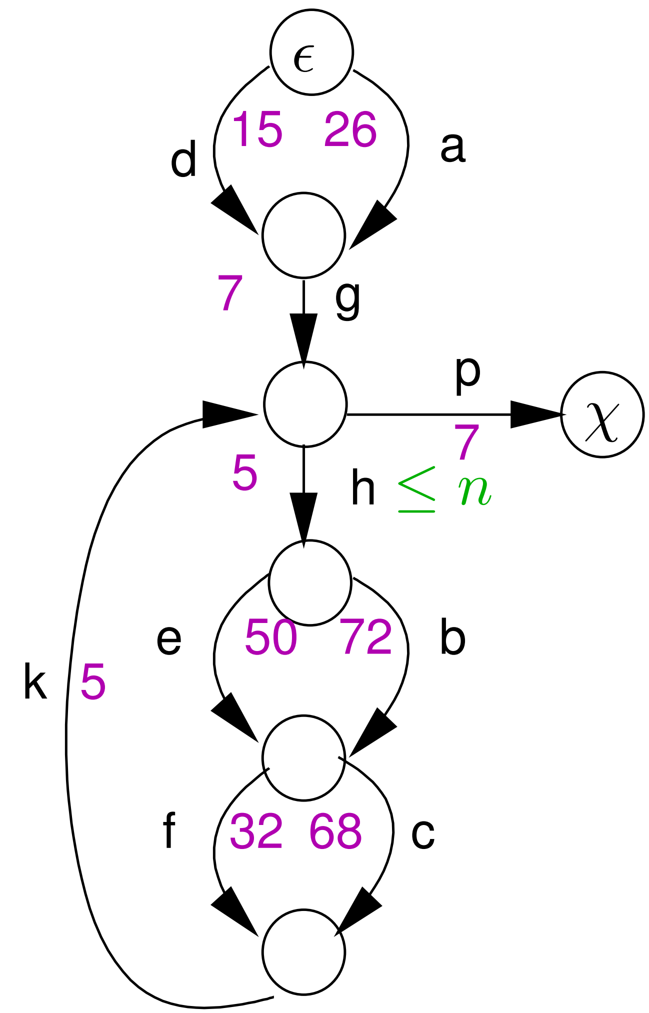
- μ-archi analysis has assigned weights e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds 'h' taken at most n times
- ILP encoding:



ILP encoding on an example

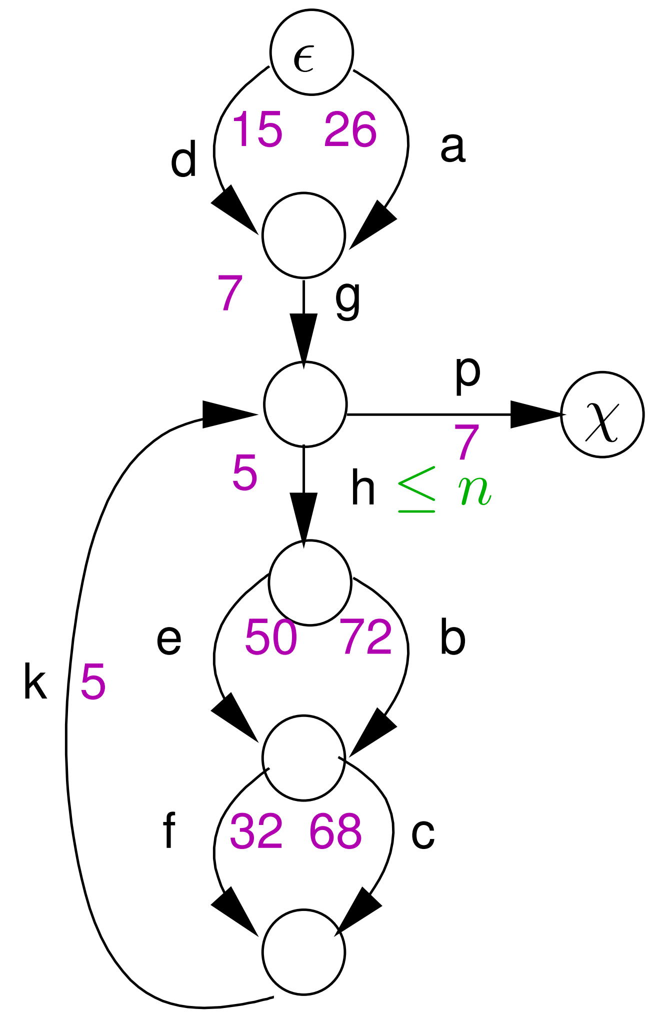
- μ-archi analysis has assigned weights
  e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds
  'h' taken at most n times
- ILP encoding:
  $\hookrightarrow$ Structural constraints

  $a + d = 1$
  $g = a + d$
  $g + k = p + h$
  etc.



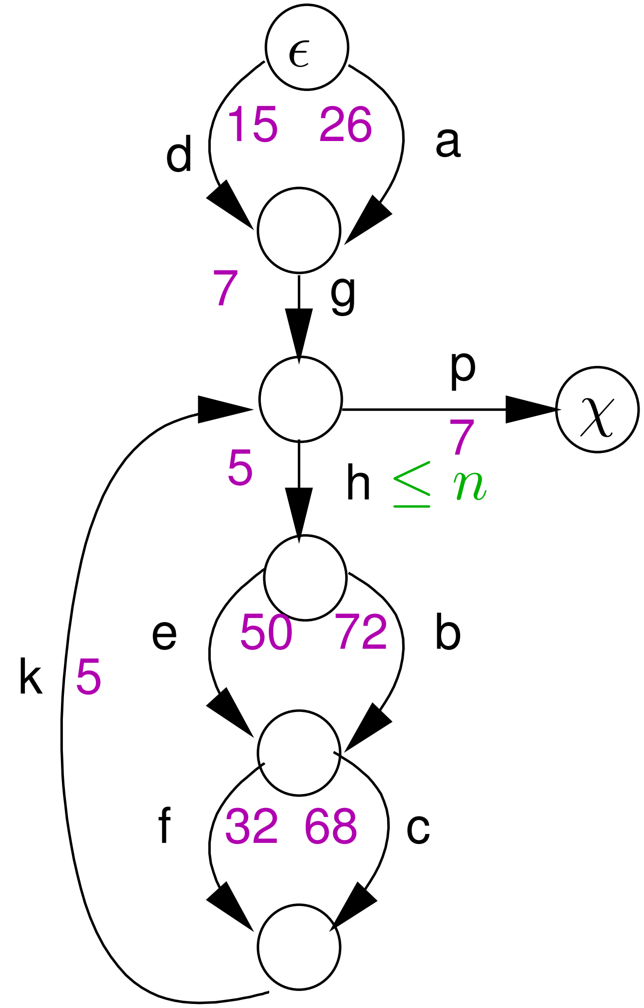
ILP encoding on an example

- μ-archi analysis has assigned weights
  e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds
  'h' taken at most n times
- ILP encoding:
  $\hookrightarrow$ Structural constraints
  $a + d = 1$
  $g = a + d$
  $g + k = p + h$
  etc.
  $\hookrightarrow$ Semantic constraints
  $h \leq n$



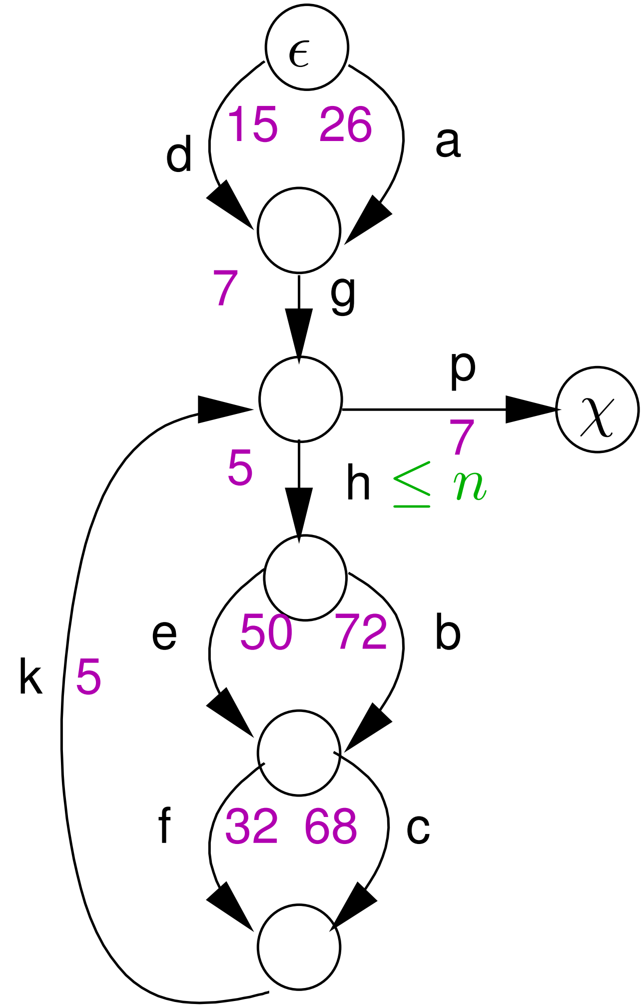
ILP encoding on an example

- μ-archi analysis has assigned weights e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds 'h' taken at most n times
- ILP encoding:
  - ↪ Structural constraints
    $a + d = 1$
    $g = a + d$
    $g + k = p + h$
    etc.
  - ↪ Semantic constraints
    $h \leq n$
  - ↪ Objective function
    MAX$(\sum_{x \in \mathcal{E}} w_x x)$



ILP encoding on an example

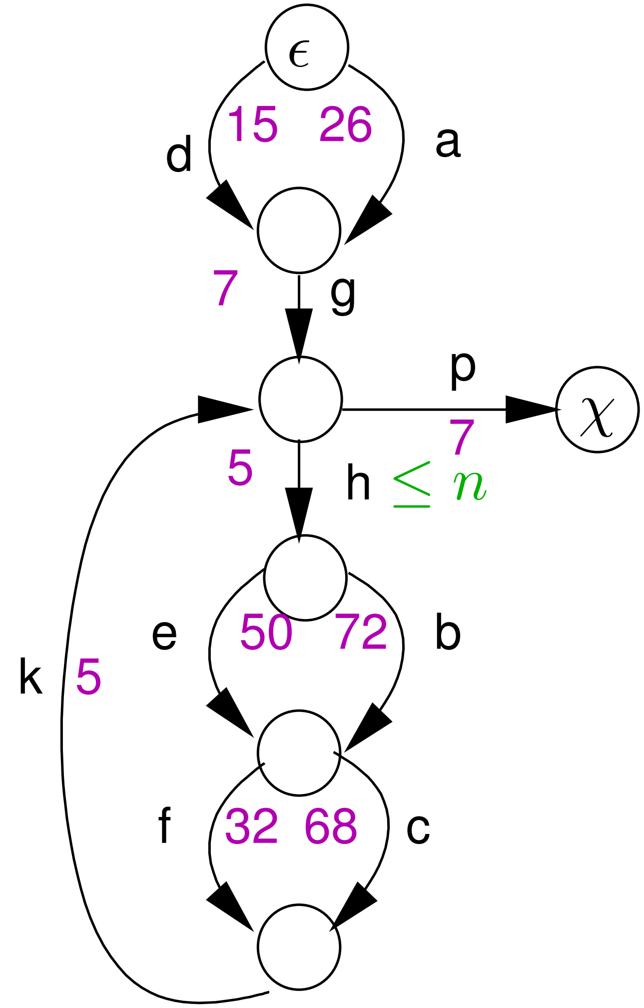
- μ-archi analysis has assigned weights
  e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds
  'h' taken at most n times
- ILP encoding:
  $\hookrightarrow$ Structural constraints
  $a + d = 1$
  $g = a + d$
  $g + k = p + h$
  etc.
  $\hookrightarrow$ Semantic constraints
  $h \leq n$
  $\hookrightarrow$ Objective function
  MAX$(\sum_{x \in \mathcal{E}} w_x x)$



Interest ? against (e.g.) graph traversal techniques ?

ILP encoding on an example

- μ-archi analysis has assigned weights
  e.g. $w_a = 26$, $w_b = 72$ etc.
- data-flow analysis has found loop bounds
  'h' taken at most n times
- ILP encoding:
  $\hookrightarrow$ Structural constraints
  $a + d = 1$
  $g = a + d$
  $g + k = p + h$
  etc.
  $\hookrightarrow$ Semantic constraints
  $h \leq n$
  $\hookrightarrow$ Objective function
  MAX$(\sum_{x \in \mathcal{E}} w_x x)$



Interest ? against (e.g.) graph traversal techniques ?

$\hookrightarrow$ Yes, as far as *infeasibility constraints* can be found/expressed

Infeasibility constraints

```
if (init) {
 /* a */
} else {
 /* d */
}
for(i=0;i<n;i++){
 if (Y[i]) {
  cond = not init and Z[i];
  /* b */
 } else {
  cond = true;
  /* e */
 }
 /* ... */
 if (cond){
  /* c */
 } else {
  /* f */
 }
}
```

Infeasibility constraints

```
if (init) {
 /* a */
} else {
 /* d */
}
for(i=0;i<n;i++){
 if (Y[i]) {
  cond = not init and Z[i];
  /* b */
 } else {
  cond = true;
  /* e */
 }
 /* ... */
 if (cond){
  /* c */
 } else {
  /* f */
 }
}
```

- for each iteration, edges e and f are incompatible
  - $\hookrightarrow$ notion of conflicting pair
  - $\hookrightarrow$ largely used in literature
  - $\hookrightarrow$ restricted to simple scopes
  - $\hookrightarrow$ here: $e + f \leq n$

Infeasibility constraints

```
if (init) {
 /* a */
} else {
 /* d */
}
for(i=0;i<n;i++){
 if (Y[i]) {
  cond = not init and Z[i];
  /* b */
 } else {
  cond = true;
  /* e */
 }
 /* ... */
 if (cond){
  /* c */
 } else {
  /* f */
 }
}
```

- for each iteration, edges e and f are incompatible
  - $\hookrightarrow$ notion of conflicting pair
  - $\hookrightarrow$ largely used in literature
  - $\hookrightarrow$ restricted to simple scopes
  - $\hookrightarrow$ here: $e + f \leq n$
- a makes b and c incompatible at each iteration
  - $\hookrightarrow$ 3 edges involved, conflict across loop scope...
  - $\hookrightarrow$ hardly treated in literature ...

    ... without modifying (unfolding) the graph
  - $\hookrightarrow$ however, ad hoc reasoning gives:

    $na + b + c \leq 2n$

State of the art:

- Almost all IPET related work use extra constraints to prune infeasible path
- Mix two problems: find the properties and express them in ILP
- Mainly focus on particular properties (e.g. pairwise edge exclusion), holding for particular scopes (e.g. intra-loop), leading to particular constraint shapes (e.g. bounded sums)
- since the goal is WCET enhancement, often completed with graph transformation methods

State of the art:

- Almost all IPET related work use extra constraints to prune infeasible path
- Mix two problems: find the properties and express them in ILP
- Mainly focus on particular properties (e.g. pairwise edge exclusion), holding for particular scopes (e.g. intra-loop), leading to particular constraint shapes (e.g. bounded sums)
- since the goal is WCET enhancement, often completed with graph transformation methods

The problem faced here:

- don't consider the discovery of the properties, only their expression in ILP
- how to: characterize pruning constraints ? express them with ILP ?
- remain abstract: as far as possible only reason on *numbers*, not on program shape/pattern
- explore the limits of ILP formulation: graph transformation forbidden

Programs, traces and executions

- a program P = CFG = vertices + edges + start + exit

  edges are named a,b,c etc.

- trace = path from start to exit = sequence of edges

  $\mathcal{T}(P)$ = set of P traces

- for any $t \in \mathcal{T}(P)$, $|a|_t$ = number of occurrence of a in t

  n.b. depending on the context, we often simply note a for $|a|_t$

- $\mathcal{E} \subseteq \mathcal{T}(P)$ = real/exact set of (bounded) executions

  (i.e. $\mathcal{T}(P) \setminus \mathcal{E} =$ set of infeasible executions)

Unfoldings

- Formalize the notion of ”more precise CFG”
- U a CFG, δ a mapping from U edges to P edges

  let a be a P edge, a_1, a_2, $\cdots$ be the U edges s.t. $\delta(a_i) = a$

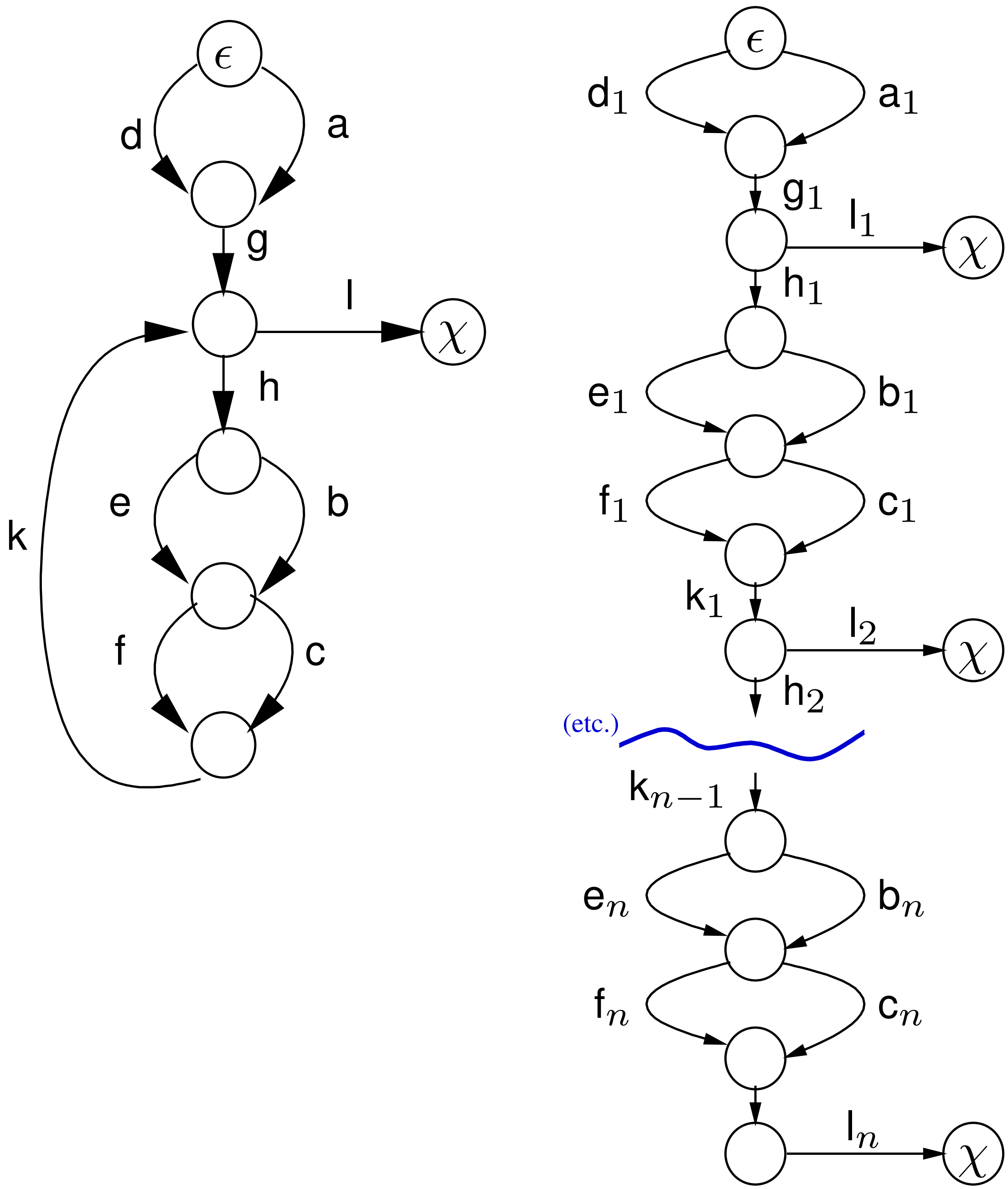
  (let’s call them the *avatars* of a
- Let $\mathcal{T}^{\delta}(U)$ = set of (decoded) traces of U
- (U, δ) is an unfolding of P iff:

$$\boxed{\mathcal{E} \subseteq \mathcal{T}^{\delta}(U) \subseteq \mathcal{T}(P)}$$

- From now on: only consider ACYCLIC unfoldings

  N.B. This is (virtually) what we have with a CFG + loop bounds

Unfolding and ILP



- $\{x_1, \cdots, x_n\}$ = avatars of x
- m_x = # of x avatars
- For any trace $t \in \mathcal{T}(U)$
  - $\hookrightarrow$ let $t' = \delta(t) \in \mathcal{T}(P)$
  - $\hookrightarrow$ then for any edge x:

    $|x|_{t'} = \sum_{i=1}^{m_x} |x_i|_t$
  - $\hookrightarrow$ or (simplified notation):

$$x = \sum x_i$$

- Moreover, for any x_i:

  (acyclic property)

$$0 \leq x_i \leq 1$$

Conflict and completion (example)

- On the unfolding: n (avatar) edges are conflicting if no execution where they are **all** taken is feasible
- N.B. Any set of infeasible paths can be expressed as a conjunction of conflicts

$\hookrightarrow$ It explains why we only focus on conflicts (see paper)

- For instance (example):

  $\{e_1, f_1\}$, ... $\{e_n, f_n\}$ are conflicting sets (all for the "same reason")

  $\{a_1, b_1, c_1\}$, ... $\{a_1, b_n, c_n\}$ are conflicting sets (all for the "same reason")

- Conflict to ILP, at unfolding level is trivial, e.g.:

$\hookrightarrow$ $\forall i = 1 \cdots n,\;\; e_i + f_i \leq 1$

$\hookrightarrow$ $\forall i = 1 \cdots n,\;\; a_1 + b_i + c_i \leq 2$

- Erase avatar details by summing all constraints, e.g.:

$\hookrightarrow$ $\sum_{i=1}^{n}(e_i + f_i) = e + f \leq 1n$

$\hookrightarrow$ $\sum_{i=1}^{n}(a_1 + b_i + c_i) = na + b + c \leq 2n$

Towards a general result ?

Conflict and completion (general 3-edges case)

- Sake of simplicity: case of 3 (concrete) edges, a, b, c
- Suppose existence of a (suitable) acyclic unfolding:

  keep it abstract, reason about numbers only !

  ↪ Numbers of avatars: m_a, m_b, m_c avatars

  ↪ Size of the conflict:

  a set S of s conflicting avatar triples (i, j, k), i.e.

  such that $a_i + b_j + c_k \leq 2$
- The sum of constraints gives:

  $\sum_{(i,j,k)\in S}(a_i + b_j + c_k) \leq 2s$
- Problem: complete this constraint to obtain full versions of a, b, c ?

A (very) bad solution: rough completion

- Let $m = m_a * m_b * m_c$ the number of avatar triples,
- there are s conflicting triples,
- and thus $m - s$ non-conflicting triples, satisfying the (trivial) constraint:
  $a_i + b_j + c_k \leq 3$
- leading to the non-conflict constraint:
  $\sum_{(i,j,k) \notin S} (a_i + b_j + c_k) \leq 3(m - s)$
- Summing conflict and non-conflict constraints gives the formula:

$$\begin{array}{rcl} \displaystyle\sum_{(i,j,k) \notin S} (a_i + b_j + c_k) & \leq & 3(m - s) \\ + \displaystyle\sum_{(i,j,k) \in S} (a_i + b_j + c_k) & \leq & 2s \end{array}$$

$$\boxed{m_b m_c \, a + m_a m_c \, b + m_a m_b \, c \leq 3m - s}$$

Likely to be very imprecise !

Precise completion

Multiplicity

- Conflict constraint is (also) of the form:

$$\sum_{i=1}^{m_a} \alpha_i a_i + \sum_{j=1}^{m_b} \beta_j b_j + \sum_{k=1}^{m_c} \gamma_k c_k \leq 2s$$

- Idea: add as few useless constraints ($a_i \leq 1$) to obtain complete "versions" of a
- focus on the a term ($\sum_{i=1}^{m_a} \alpha_i a_i$):

$\hookrightarrow$ sum of coefs is $\sum_{i=1}^{m_a} \alpha_i = s$

$\hookrightarrow$ the biggest α_i is the multiplicity of a, noted p_a

how many time MAX the whole edge is involved

$\hookrightarrow$ In a precise solution, the "whole" a must appear p_a times, AND NOT MORE!

Lack

- We need p_a times a in the constraint, thus a total of $p_a m_a$ avatars
- It's does not matter which avatars are missing, only their number matters:

  the *lack of a* in the conflict = $\ell_a = p_a m_a - s$
- Conclusion: adding ℓ_a to the right-hand side, allows to replace all a_i details by $p_a a$ in the left hand side

Summary

In order to get a precise ILP formulation of a conflict, one has to identify:

- For each edge, its number of “avatars” m_a, in a suitable unfolding of the program (kept largely virtual),
- The size of the conflict, s: the number of conflicting avatars
- For each edge, its multiplicity p_a: the number of times the edge is “involved” in the conflict,

  from which we compute the lack $\ell_a = p_a m_a - s$ etc.
- Then, the following ILP constraint holds:

$$\boxed{p_a a + p_b b + p_c c \le 2s + \ell_a + \ell_b + \ell_c}$$

n.b. the generalization to any number of conflicting edges is straightforward

e.g. case for 2 edges:

$$p_a a + p_b b \le s + \ell_a + \ell_b$$

Example program of the beginning

- $\{e, f\}$ conflict:
  - $\hookrightarrow$ loop bounds give $m_e = m_f = n$, conflict holds $s = n$ times
  - $\hookrightarrow$ each avatar involved once, thus $p_e = p_f = 1$ and $\ell_e = \ell_f = 0$
  - $\hookrightarrow$ finally: $e + f \leq n$
- $\{a, b, c\}$ conflict:
  - $\hookrightarrow$ $m_a = 1$, $m_b = m_c = n$, conflict holds $s = n$ times
  - $\hookrightarrow$ a involved n times, b and c once, thus $p_a = n$, $p_b = p_c = 1$ and $\ell_a = \ell_b = \ell_c = 0$
  - $\hookrightarrow$ finally: $na + b + c \leq 2n$

Example with lack

- Similar to the previous $\{a, b, c\}$, except that b and c conflict on consecutive iterations i.e. $\{a_1, b_2, c_1\}$, $\{a_1, b_3, c_2\}$ etc.
- $m_a = 1$, $m_b = m_c = n$, conflict holds $s = n - 1$ times
- $p_a = n - 1$, thus $\ell_a = p_a m_a - s = 0$
- $p_b = p_c = 1$, thus $\ell_b = \ell_c = p_b m_b - s = 1$
- right hand side is: $2s + \ell_a + \ell_b + \ell_c = 2(n-1) + 2 = 2n$
- finally: $(n-1)a + b + c \leq 2n$

Auto-conflict

- An edge a within a loop conflict with the *same* edge at the next iteration
- Case covered by the formula: just keep in mind that a plays TWO roles i.e. apply the a, b formula, keeping in mind that $a = b$
- $m_a = m_b = n$, $s = n - 1$
- $p_a = p_b = 1$, $\ell_a = \ell_b = 1$
- thus: $a + b \leq n - 1 + 1 + 1 = n + 1$
- and since $a = b$: $2a \leq n + 1$

Conclusion

- Explores the ability of ILP methods to express infeasibility
- Shows that infeasibility can be expressed as a conjunction of conflicting constraints
- Propose a general method, requiring to “numerically” characterize the conflicts
- Mainly a theoretical work,
- however, helps to understand/classify/compare existing concrete methods
- Open problem: can it help to develop/enhance actual methods ?