Cost Analysis Summary II

Monday, July 18, 2022 9:25 AM

T

Static Bound Infer Analysis.

1. Over lapped with Bound Analysis. Worst Case Runtime analysis

WCET on systems: \cite{}
[GustafssonEL05]Towards a Flow Analysis for Embedded System C Programs
--> abstract interpretation.
--> on embedded system of c program
[AlbertAGP08] Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis
--> invariant generation through ranking functions

General While langue:
[BrockschmidtEFFG16]
Analyzing Runtime and Size Complexity of Integer Programs
--> invariant generation through ranking functions
[AliasDFG10] Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs
--> invariant generation through ranking functions
[Flores-MontoyaH14]Resource Analysis of Complex Programs with Cost Equations
--> invariant generation through cost equations or ranking functions

[GulwaniJK09]Control-flow Refinement and Progress Invariants for Bound Analysis
--> program abstraction and invariant inference
[]Bound Analysis using Backward Symbolic Execution
--> program abstraction and invariant inference

① 1997. Thomas. via graph Reachibility.

② 2008. Upper Bound for Recurrence Relation in Cost Analysis.

⇒ ⓐ generate RR (recurrence Relation). ↓ close form solution / upper bound. → focus on solve the closed form

★. ⓑ solve How to compute RR into closed form..

⇒ by ranking function & loop invariant & partial evaluation

③ ⇒ 2008. Numerical Abstract Domain (Con Expression Abstraction & Max Operator) apply in Timing analysis.

⇒. exp abstraction + linear Relation. & Max

⇒ generate abstract expressions S ↗ programer provide / symbolic execution ⇒ Rules to reason them and infer Bound.

⇒ abstract interpretation

exp abstraction

linear relational abstract domain + disjunctive reasoning ↓ with max.

★ set of inference rules for computing bounds.

eg: $z_x \le c \Rightarrow z_{\exists x} \le \partial^c$

$true \Rightarrow z_{Max} = max(z_x, -z_x)$

$\sum a_i z_{x_i} = 0 \Rightarrow \sum a_i z_{x_i y} = a z_y$

$z_x \le z_y + c \Rightarrow z_{\exists x} \le \partial^c \times z_{\exists y}$.

... ↓ computation algorithm

④ 2009. - Control flow refinement. more detail in RB analysis note

1. refine if-branch.

↓ 2. still solve Bound for outside loop by conjunction of multiple path.

⑤ 2009. - SPEED

↓ Extend to inter-procedure with implementation improvement.

⑥. 2009: Symbolic Execution

similar to SPEED.

⑦. 2010. Reachability Bound Problem. ↝ detail & Example in RB.

↝ by infer closed loop Bound for Location Reachability. ⇒ not for location.

↝ entire cost count the max

⑧. 2012. Bound

⇒ the same as the PLDI'17. Complexity ...

solve Bound for each SCC. by assigning each edge with an dec var.

⇒ compute Bound of Dec var - use it as Bound of While SCC

⇒ Bound of edge.

① Still cannot solve multi Bound eqld.

② ↝ for entire cost. still take the max.

⇒ 2013. Size-Change Abstraction & Max Plus automata technique paper on how to solve equation to infer Bound

ⓐ. automata ×
★ ⓑ. characterizing the computation complexity bounds obtained by size-change.
↓
ⓒ. size-change predicate (SCP).
SCT = $\mathcal{P}$(SCP). $scs \in \mathcal{P}(SCT)$
$\sigma, \sigma' \models xy$ if $\sigma(x) > \sigma'(y)$ holds
$\sigma, \sigma' \models \{x, x\}$.
a bit for how solving Bound.

⑨ 2014. Resource Analysis with Cost Equation.

⇒. by infer Loop Bound & Estimate the Worst Case

⑩ 2014: Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis.

efficiency improvement in solving the Bound

But still estimate the Worst Case cost Consumption. for program

⑪ 2016: Runing time & size Complexity of Integer Programs

⇒ infer Bound for variables, ⇒ then apply to program cost infer.

But still count entire Worst Case cost.

⑫ ⇒ 2017: Sinn: Complexity & Resource Bound analysis of Imperiative Program Using DC

⇒ ① doesn't Consider path sensitive

② still worst case.

Cost Analysis SummaryI

Monday, July 18, 2022 9:25 AM

Type :

1. Relational Cost

⇒ 2019. for imperative Programs. ⇒ typing rules :

typing rules : $\vdash^{U_1}_{L_1} t_1 : A_1$. $\vdash^{U_2}_{L_2} t_2 : A_2$ / $t_1 \ominus t_2 : U_1 - L_2$.

$\vdash^{U_1}_{L_1} t_1' : A_1'$. $t_1 \ominus t_2 \le D_2$ / $t_2 \ominus$ let $x = t_1'$ in t_2 $\le = D_2 - L_t - U_t$

upper bound − lower bound. $U_1 + U_2' + U_{lt} - (L_1 + L_2' + L_{lt}) = U_1 - L_1 + (U_2' - L_2') + (U_{lt} - L_{lt})$ (C_{ld})

2018 ⇒ Monadic refinement for relational Cost analysis

typing rule : Abs : $\vdash e_1 \ominus e_2 \le n : \tau_2$ / $\vdash \lambda x. e_1 \ominus \lambda x. e_2 \le 0$ ⤳ decrease with explicite function abstraction.

= explicite abstraction.

⇒ 2017. Relational Cost analysis: Eiagi.

rules : sync : rules : fix : $\vdash e_1 \ominus e_2 \le t : \tau_2$ / $\vdash$ fix $f(x). e_1 \ominus$ fix $f(x). e_2 \le 0$

iLamd $\vdash e_1 \ominus e_2 \le t$ / $\Lambda. e_1 \ominus \Lambda. e_2 \le 0$.

2. Amortized Cost analysis

⇒ 1983. Tarjan: Amortized Computational Complexity. ⤳ explicit pop.

Σ actual cost.

↓ extension with union-find data

⇒ 2000 : A pontential Based Amortized Analysis of the Union-Find Data Structure.

⇒ 2003: Static Program Prediction of Heap Usage for First-order Functional Programs.

⇒ intro type system & Recurrence for Formal Amortized Analysis. , Potential Based.

(extend potential expression) ⇒. Reuse cost by "explicit" deallocation, & garbage collection ⤳ explicit deallocation & garbage collection.

⇒ 2010. Amortized Resource Analysis with Polynomial Potential.

⇒ 2010. Static Determination of Quantitative Resource Usage for Higher-Order Programs.

extend : higher-order, polymorphic. recursive. possibly-aliased.

⇒ $\Gamma = FV(e)/x$. $x : A \vdash^{q}_{q'} e : C$ $\phi \Rightarrow \bigcup_{DET} \forall (D|D, D)$. $\vec{\gamma} \notin FV(\Gamma) \cup FV(\phi)$ / $\vdash^{kmkFun(|\Gamma|)}_{0}$ $\lambda x. e : \forall \vec{\gamma} \in \psi. B | \phi$.

X. ① explicit abstraction.

(extension) ②. to achieve recursion. limited potential in function closure = 0, then count cost by requiring potential from argument

③. if in case function call have cost. then need explicit costant # of function call.

⇒. 2011. Multi-variate Amortized Resource Analysis.

⇒ 2013. A static Cost Analysis for Higher order Language. → using potential cost save in function. same as Amortized analysis ⇒

$\lambda x. e : [\![r \to y]\!]$: potential → complexity. ⇒ in a denotational semantic way.

t t : $[\![t_1 t_2]\!] = 1 + [\![t_1]\!] + [\![t_2]\!]$

⇒ 2015. Compositional Certified Resource Bounds. ⤳ algorithm automaticalize.

⇒ 2018 Automatic Space Bound Analysis for Functional Programs with Garbage Collection.

✓⑧ Space Bound Analysis for Functional Programs with Garbage Collection.

☆ ⇒ recounting the resource by restituting the potential to type the subexpression ⤳ explicit match in nil & cons. for reducing cost.

extend ⇒ $\dfrac{\Sigma; \Gamma \vdash^{q}_{q'} e_1 : B \quad x_h : A,\ x_t : L^{p}(A) \vdash^{q + p + 1}_{q'} : e_2 : B}{\vdash^{q}_{q'} \text{match } x \{ \text{nil} \to e_1\ \text{cons}(x_h; x_t) \to e_2 \}}$ ☆.

⇒ 2020. Exponential Automatic Amortized Resource Analysis.

⇒ extend the analysis result expressiveness ⇒ exp. no precision improvement

⇒ 2018. Bound Expectation. R.. for Probabilistic. Program.

⇒ 2021. Central Moment analysis for Cost Accumulators in Probabilistic Programs

⇒ bounds on pop. concentration inequality. ⇒ central moments.

⇒ Bounds on variants, central Moment, for ↓.

☆ cost-accumulator : termination time. [...]
rewards. in Markov decision [32]
position information [4. 7. 34].
cash flow [7. 24. 39. 41]. static approach. leverage aggregate infos of cost accumulator.

⇒ 2021. Unifying Type-Theory for higher-order (Amortized) Cost Analysis

⇒ λ-amor. amortized cost analysis.

⇒ potentials : cost have been accounted for, but not yet incurred.

☆: T-tick. only resource consume: T-tick $\dfrac{k : \mathbb{R}}{\cdots \vdash \uparrow^{k} : M\, k\, 1}$

☆ T-release: $\dfrac{\vdash e_1 : [p_1]\, \tau_1 \qquad \vdash e_2 : M(p_1 + p_2)\, \tau_2}{\vdash \text{release } x = e_1 \text{ in } e_2 : M\, p_2\, \tau_2}$ reduce the cost stored in e. ☆.

☆. T-store : $\vdash e : \tau$. $\vdash p : \mathbb{R}^{+}$ / $\vdash$ store $e : M\, p([p]\, \tau)$.

✓ but not account this in eval.

3. other types : Resource Usage Analysis.

Static Bound Infer Analysis.

1 Over lapped with Bound Analysis. Worst Case Runtime analysis

WCET on systems: \cite{}
[GustafssonEL05]Towards a Flow Analysis for Embedded System C Programs
--> abstract interpretation.
--> on embedded system of c program
[AlbertAGP08] Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis
--> invariant generation through ranking functions

General While langue:
[BrockschmidtEFFG16]
Analyzing Runtime and Size Complexity of Integer Programs
--> invariant generation through ranking functions
[AliasDFG10] Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs
--> invariant generation through ranking functions
[Flores-MontoyaH14]Resource Analysis of Complex Programs with Cost Equations
--> invariant generation through cost equations or ranking functions

[GulwaniJK09]Control-flow Refinement and Progress Invariants for Bound Analysis
--> program abstraction and invariant inference
[]Bound Analysis using Backward Symbolic Execution
--> program abstraction and invariant inference

① 1997. Thomas. via graph Reachibility.

②. 2008. Upper Bound for Recurrence Relation in Cost Analysis.

⇒ ① generate RR (recurrence Relation).

☆. ② solve How to compute RR into closed form.. ↓ close form solution / upper bound. → focus on solve the closed form

⇒ by ranking function & loop invariant & partial evaluation

③ ⇒ 2008. Numerical Abstract Domain (on Expression Abstraction & Max Operator) apply in Timing analysis.

⇒. exp abstraction + linear Relation. & Max

⇒ generate abstract expressions S ↗ programer provide / symbolic excution ⇒ Rules to reason them and infer Bound.

⇒ abstract interpretation

< exp abstraction
{linear relational abstract domain + disjunctive reasoning. } with max.

☆ set of inference rules for computing bounds.

eg : $z_x \le c \Rightarrow z_{2^x} \le 2^c$

true $\Rightarrow z_{\sqrt{x^2}} = \max(z_x, -z_x)$

$\sum a_i z_{x_i} = a \Rightarrow \sum a_i z_{x_i y} = a z_y$

$z_x \le z_y + c \Rightarrow z_{2^x} \le 2^c \times z_{2^y}$.

... ⇓ computation algorithm

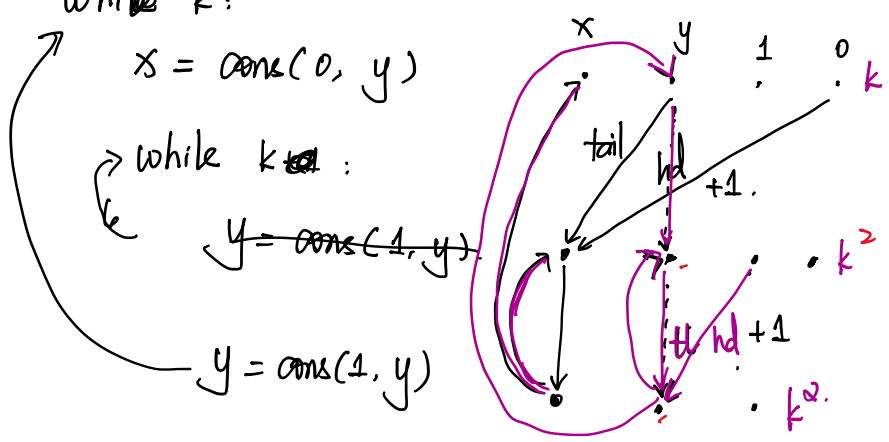
Memory resources

Monday, July 11, 2022 9:03 AM

Shape Analysis :

possible "shapes" that heap-allocated structure can take :

⇒ example :



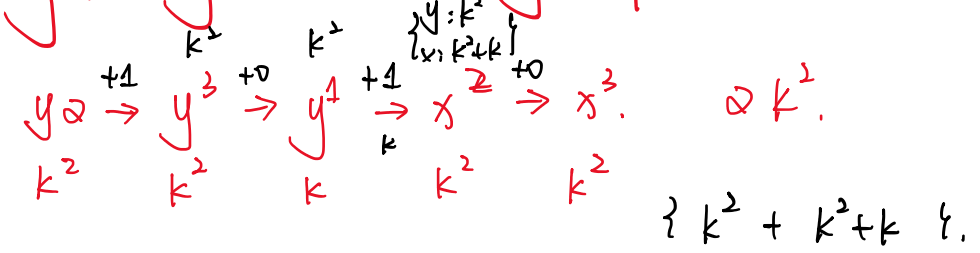
⇒ const(x) = k^3 or $k+k^2$.

actually

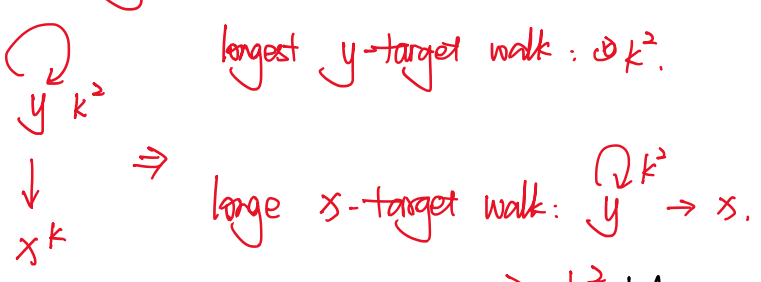
const(x) = k^ω.

⇒ by reduce to single-target path problem:

longest weighted x-target path is:



⇒ by longest finite walk



More on Cost Analysis

Wednesday, July 13, 2022 9:28 AM

Static Resource analysis of Program. with cost Decreasing. via finite walk.

Check are they able to handle cost decrease.

① via type system, mainly marco. & effect system. Dagstuhl
graded Hoare Logic. & its Categorical Semantics.

A Type-Theory for Higher-Order (Amortized) Cost analysis

Monadic Refinement for Relational Cost Analysis.

② Via Static Program Analysis [static cost analysis]
1. static cost analysis. for higher-Order Language.
⇒ Mainly Jan Hoffman. Zhong Shao.
⇒ Multivariate Amortized Resource analysis.

⇒ via type system:

① Relational Cost analysis: either { upper Bound - lower Bound. / sum all cost of subexpression. } ⇒ same. ⇒
⇒ Extended. Relational Cost analysis for Functional-Imperative Programs.

② Monadic Refinement for Rel-cost.
⇒ higher-order refinements, and cost monads.

one issue: Lambda rule: $e \ominus e' \le t \rightarrow \lambda x.e \ominus \lambda x.e' \le 0$. ☆.

③. Graded Hoare Logic & its Categorical Semantics. ⇒ only increase.

④. A Type-Theory for Higher-Order (Amortized) Cost analysis ~ 2021.
⇒ λ-amor. amortized cost analysis.
⇒ potentials: cost have been accounted for, but not yet incurred.

☆: T-tick. only resource consume: $\dfrac{k : \mathbb{R}.}{\cdots \vdash \uparrow^k : M\ k\ 1}$ T-tick

☆ T-release: $\dfrac{\vdash e_1 : [p_1]\tau_1 \qquad \vdash e_2 : M\ (p_1 + p_2)\ \tau_2}{\vdash \text{release}\ x = e_1\ \text{in}\ e_2 : M\ p_2\ \tau_2}$ reduce the cost stored in e. ☆.

☆. T-store: $\dfrac{\vdash e : \tau. \qquad \vdash p : \mathbb{R}^+.}{\vdash \text{store}\ e : M\ p\ ([p]\tau).}$

✓ but not account this in eval.

⑤. Central Moment Analysis for Cost accumulators. in Probabilistic. <> ⑥ Bounded Expectations.
Resource Analysis for Probabilistic Programs.
⇒ bounds on prop. concentration inequality. ⇒ central moments.
⇒ Bounds on variants, central Moment, for.

☆ cost-accumulator: termination time. [...]
rewards. in Markov decision [32]
position information [4. 7. 34].
cash flow [7. 24. >?. 41]. static approach. leverage aggregate infos of cost accumulator.

Amortized: Potential:

⑥. ⑦*: 2020. Exponential Automatic Amortized Resource analysis. ⇒ Resource Bound exp in size of inputs.
✓⑧ Space Bound Analysis for Functional Programs with Garbage Collection.
☆ ⇒ recounting the resource by restituting the potential to type the subexpression

⇒ $\dfrac{\Sigma ; \Gamma \vdash {}^{q}/_{q'}\ e_1 : B \qquad x_h : A,\ x_t : L^p(A) \vdash {}^{q+p\cdot 1}/_{q'}\ : e_2 : B}{\vdash {}^{q}/_{q'}\ \text{match}\ x\ \{\text{nil} \to e_1 \mid \text{cons}(x_h, x_t) \to e_2\}.}$ ☆.

⑨ 2011. Multivariate Amortized Resource Analysis.

⑩ 2010. Amortized Resource analysis with Polynomial Potential.

⑪ 2003. potential Based amortized analysis. ⇝ Types & Recurrences for Formal Amortized Analysis
⇒ Steffen Jost. Martin Hofman.
Static Prediction of Heap Space Usage for first-order Function Program.

⑫ 2000, A potential-Based Amortized Analysis of Union-find data Structure.

⑬ 1985: Tarjan. Amortized Computational Complexity
Σ actual cost - net gain of potential incured by data structure invocation.

2007 Extended: $\Gamma, n \vdash_\Sigma e : A, n'$: n memory resource available, n' resources left over.

☆. Upper Bound decrease: in λ. rule. ⇝

$$\Rightarrow \quad \frac{\Sigma(f) = (A_1, \ldots A_p, k) \to (C, k') \qquad n \ge k \qquad n - k + k' \ge n'}{\Gamma \quad n \vdash f(x_1 \ldots x_p) : C : n'} \quad \Rightarrow n' = n - k + k'$$

$n = n' + k \boxed{-k'}$ → estimated upper bound decreased by k', this k' is encapsulated as potential into the type of function

⑭. Unifying Type-Theory for Higher-Order (Amortized) Cost Analysis

④. T-release: $\dfrac{\cdots \vdash e_1 : [p_1]\tau_1 \qquad \cdots \vdash e_2 : M(p_1 + p_2)\ \tau_2}{\cdots \vdash \text{release}\ x = e_1\ \text{in}\ e_2 : M\ p_2\ \tau_2}$

the potential for e_1. will not be assumed by e_2 after release. ⇒ But need explicit Release.
resource (cost) decreased by p_2.

⇓ T-store

⑤ fix: fix.x.e ? recursive function?

Reduce to walk

Monday, July 11, 2022 9:58 PM

general Resource analysis

Better, More accurate. to reduce to finite walk

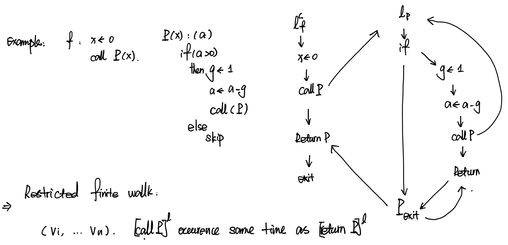
CFL-Reachability : labels of edges composing into a matched word

label matching can be reduced to restrict the occurence times equality

⇒ higher efficient, & symbolic occurence time.

for example:

① interprocedure call can be reduced to valid walk by



⇒ Restricted finite walk:

$(v_i, \dots v_n)$. call_P^i occurence some time as Return_P^i

can generalize to.

② Program slicing Problem:

Example Application ②:

program slicing Problem:

Backword / Forward slicing:

⇒ by define as a single-target L(slice)-problem:

where L defined as:

unbalanced-right := unbalanced-right matched | unbalanced-right); $1 \le i \le \text{callsite}$. ⇒ ✓

| ε

slice := unbalanced-right realizable.

⇒ by restricting inequality between occurence times of vertices.

③. Data flow

standard Data reachability definition analysis is a special case of valid walk.

by set the vertice kill x. weight 0

vertice gen x weight 1.

⇒ valid walk = feasible path.

① IFDS: (Interprocedural, Finite, Distributive, Subset problems)

⇒: 1. Supergraph G^*

2. $\mathbb{D}$: Domain of data flow facts. ⇝ $\Lambda \in \mathbb{D}$. the initial data fact.

Each program point ⇝ associated. exploded to a memember ⇝ $\in 2^{\mathbb{D}}$.

3. Assign of distributive dataflow functions: $2^{\mathbb{D}} \to 2^{\mathbb{D}}$, to edges of G^*.

⇒ solve it as realizable-path reachability problem.

⇒ exploded supergraph: $G^\#$. for each node in G^*:

⇒ $G^\#(n) = \{ V_{G^\#(n)}, E_{G^\#(n)} \}$.

$d \in V_{G^\#(n)}$: $d \in \mathbb{D}$: a dataflow fact.

$e = (d_1, d_2)$. $d_1 \in V_{G^\#(n_1)}$

$d_2 \in V_{G^\#(n_2)}$.

dataflow facts d_2 in node n_2. if d_1 is true in n_1.

↓ ↓ if : the fact transformation function from n_1 to n_2

... d_2 in node n_2. then $f(d_1) = d_2$ is true in n_2

⇒ Formally: $G^\#$:

$V = \forall$ node $n \in G^*$, $\exists$ a node $\langle n, \Lambda \rangle$ in $G^\#$

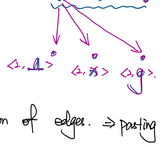
$\cup$ $\forall n \in G^*$. $d \in \mathbb{D}$. $\exists$ a node $\langle n, d \rangle$ in $G^\#$.

E: ① = edge $e = (\langle m, \Lambda \rangle, \langle n, d \rangle)$ $\forall d \in f(c)$.

$\cup$ ② edge $e = (\langle m, d_1 \rangle, \langle n, d_2 \rangle)$ $\forall d_1, d_2 \in \mathbb{D}$. $d_2 \in f(c, d_1) \wedge d_2 \notin f(c)$

$\cup$ ③ edge $e = (\langle m, \Lambda \rangle, \langle n, \Lambda \rangle)$.

Example:



data flow facts: $\Lambda \cup \{x, g\}$: • x: x is the possible uninitialized variable.

<1, x>: the node for start point in program where data flow fact x is true.

fact transformation function f: $\lambda S. \{x, g\}$

i.e. $x \; f(\Lambda) = \{x, g\} \Rightarrow$ edges: $(\langle 1, \Lambda \rangle, \langle 2, x \rangle)$. $(\langle 1, y \rangle, \langle 2, \Lambda \rangle)$

⇒ composition of edges. ⇒ pasting nodes.