CFL Reduce to walk

Monday, July 11, 2022 9:58 PM

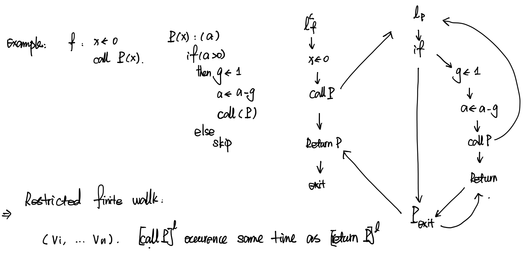
CFL-Reachability : labels of edges composing into a matched word

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] occurence same time as [return P]

can generalize to.

② Program slicing Problem:

Example Application Ω:

program slicing Problem:

Bookword / Forward slicing:

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

where L defined as:

unbalanced-right := unbalanced-right matched

| unbalanced-right); $1 \le i \le$ callsite. ⇒ ✓

| ε

slice := unbalanced-right realizable

⇒ by restricting inequality between occurence times of vertices.

③. Data flow

standard Data reachbility definition analysis is a speeial case of valid walk.

by set the vertice kill x. weight 0

vertice gen x weight 1.

⇒ valid walk = feasible path.

④. IFDS: (Interprocedural, Finite, Distributive, Subset problems)

⇒ : 1. supergraph G^*

2. $\mathbb{D}$: domain of dataflow facts ⇝ $\Lambda \in \mathbb{D}$. the initial data-fact.

Each program point ⇝ 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.

↓ ↓ : 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 $G^\#$.

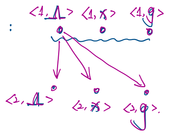
$\cup$ $\forall n \in G^*$. $d \in \mathbb{D}$. $\exists$ a node $\langle n, d \rangle$ in $G^\#$.

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

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

$\cup$ 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.

$\langle 1, x \rangle$: the node for start point in program where data-flow-fact x is true.

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

i.e. $f(\phi) = \{x, g\}$ ⇒ edges : $(\langle \Lambda, x \rangle, \langle \Lambda, y \rangle, \langle \Lambda, \Lambda$

⇒ composition of edges. ⇒ pasting nodes.

CFL-II

Friday, June 3, 2022 3:40 PM

⇒ Dyck-language : Balanced "[" "]" → kleene Closure.

⇒ IFDS, ⇝ Interprocedural. finite. distributive. subset problem.
Sharir-Pnueli -framework.

IDE ⇝ Interprocedural Distributive Environment.

CFL ⇒ chain program

⇒ data-flow fact is set of environment.

f : environment transition function,

variable propagation. → though chain.

⇒ interconvertable to Set Constraints Problem. ⇝

David Melski & Thomas Reps. Dec 1997.

https://research.cs.wisc.edu/wpis/papers/pepm97.pdf

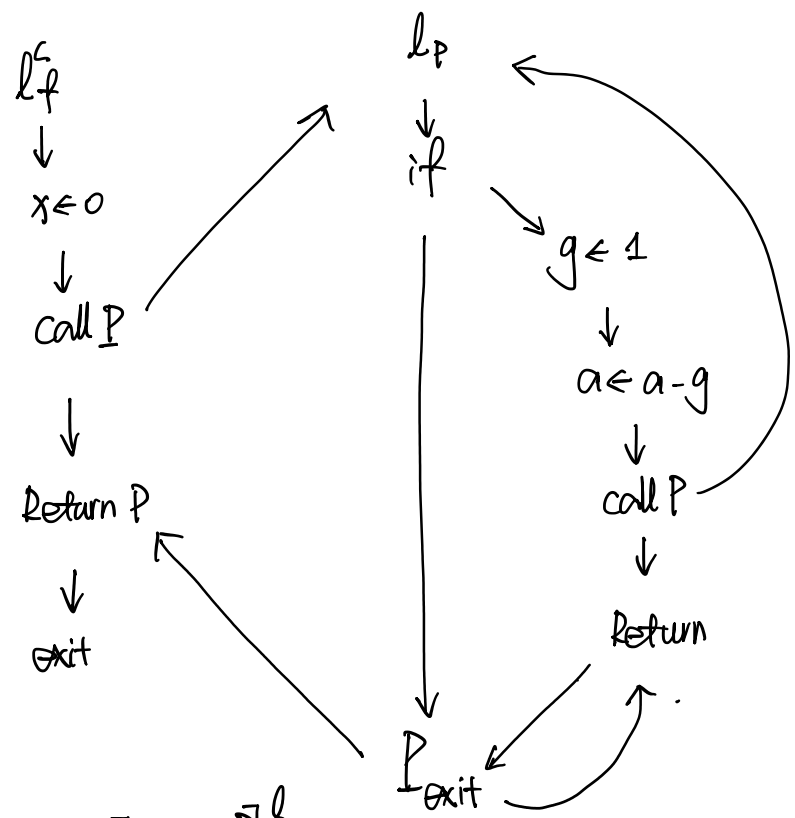
CFL-III-InterProcedure

Monday, July 11, 2022 9:06 AM

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

Example: f : x ← 0
call P(x).

```
P(x) : (a)
  if (a > 0)
    then g ← 1
         a ← a - g
         call (P)
    else
         skip
```



⇒ Restricted finite walk:

$(v_i, \dots v_n)$. $[\text{call } P]^l$ occurence some time as $[\text{return } P]^l$

can generalize. to.

Context Free language reachability

Wednesday, May 25, 2022 4:04 PM

convert program-analysis ⇒ CFL-reachability

⇒ ordinary graph-reachability.

1. generalization of graph-reachability.

with restrictions as labels on edge, (restricting visiting orders of edges?)

excluded infeasible path (i.e. paths on graph which don't corresponds to actual program executions.)

⇒ compare to longest walk definition.

example: while without ctl dep. recursive val dep ✓.

```
while k > 0
    i = k.
    k = k-1
    y = f(x)
    while i > 0 :
        i = i-1
        x = f(x)
```

Adaptivity (able to generalize to general resource consumption) computed by longest walk. more precise than the "longest path" computed from the Best/Most Accurate CFL-graph (as the L-path problem).

CFL-reachability problem:

feasible paths: $x^1 \to y^k$, $x^2 \to x^1 \to y^k$

both are corresponding to actual execution. can't be excluded.

⇒ solve as all-pairs L-path problem ⇒ $1+k!+k$ is the longest path ⇝ over-approximated

⇒ solve as longest finite walk ⇒ $1+k!+1$ is the actual (Resource consumption) Adaptivity.

Example Application on interprocedural data-flow-analysis problem:

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

⇒: 1. Supergraph G^*

2. $\mathcal{D}$: domain of data-flow facts ⇝ $\Lambda \in \mathcal{D}$: the initial data-flow. associated.

Each program point ⇝ exploded to a mememember ⇝ $\in \mathcal{D}$.

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

⇒ solve it as realizable-path reachability problem.

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

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

$d \in V_{G^\#(n)}$: $d \in \mathcal{D}$: 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_2 is true in n_2.

↓ if: the fact transformation function from n_1 to n_2

... d_2 in node n_1. then $f(d_1) = d_2$ is true in n_1

⇒ Formally: $G^\#$:

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

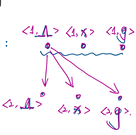
$\cup \; \forall\, n \in G^*.\ d \in \mathcal{D}.\ \exists$ a node $\langle n, d \rangle$ in $G^\#$.

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

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

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

Example:



data flow facts: $\Lambda \cup \{x, y\}$: x: x is a possible uninitialized variable.

$\langle 1, x \rangle$: the node for start point in program where data-flow fact x is true.

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

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

⇒ composition of edges. ⇒ pasting nodes.

Example Application ②:

program slicing Problem:

Backward / Forward slicing.

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

where L defined as:

unbalanced-right := unbalanced-right matched

| unbalanced-right); $1 \le i \le$ callsite. ⇒ ✓

| ε

slice := unbalanced-right realizable.

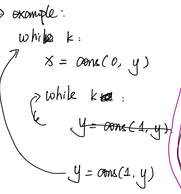
Application ③:

Shape Analysis:

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

⇒ example:

```
while k:
    x = cons(0, y)
    while k:
        y = cons(1, y)
    y = cons(1, y)
```



⇒ cons(x) = k^2 or $k + k^2$.

actually cons(x) = k^2.

⇒ by reduce to single-target path problem:

longest weighted x-target path is:

$y \xrightarrow{+1} \cdot \xrightarrow{k^2} y^2 \xrightarrow{+0} \cdot \xrightarrow{k^2} y^4 \xrightarrow{+1} \cdot \xrightarrow{k} x^2 \xrightarrow{+0} x^2$. ⊗ k^2.

$\{k^2 + k^2 + k\}$.

⇒ by longest finite walk

longest y-target walk: ⊗ k^2.

y^{k^2}

$\downarrow$ ⇒ longest x-target walk: $\otimes k^2 \to x$.

x^k

⇒ $k^2 + 1$.

✓

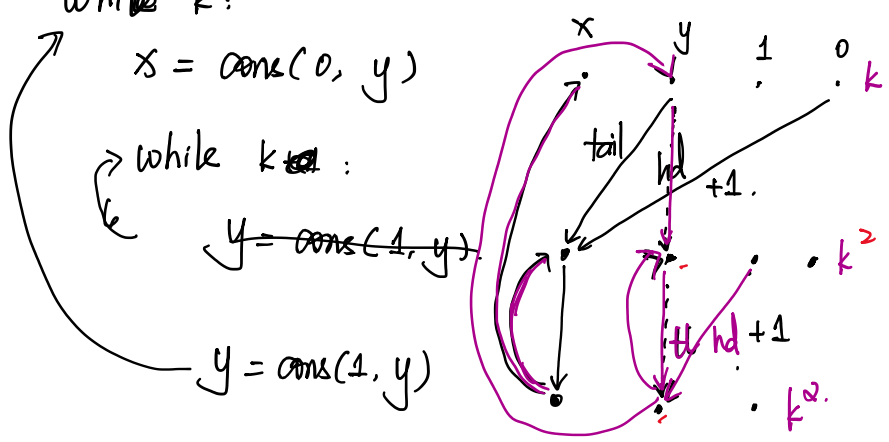
Memory resources

Monday, July 11, 2022 9:03 AM

Shape Analysis :

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

⇒ example :



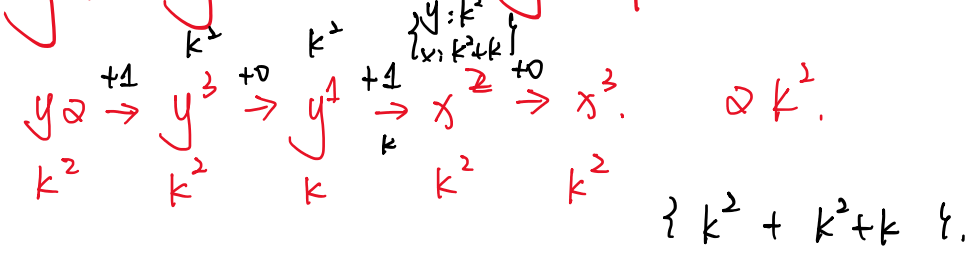
⇒ const(x) = k^3 or $k+k^2$.

actually

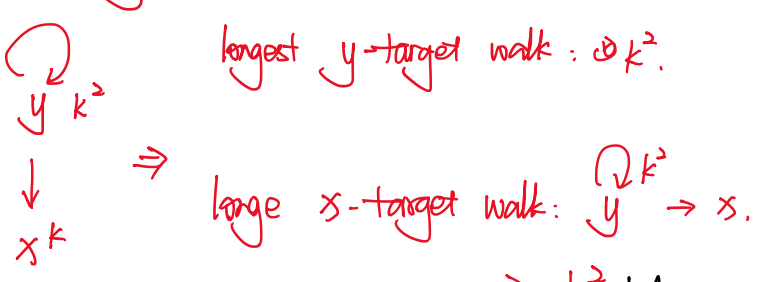
const(x) = k^ω.

⇒ by reduce to single-target path problem:

longest weighted x-target path is:



⇒ by longest finite walk



Reduce to walk

Monday, July 11, 2022 9:58 PM

general Resource analysis

Better, More accurate. to reduce to finite walk

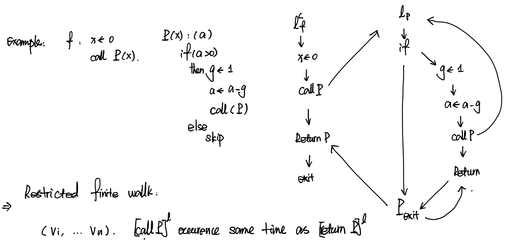
CFL-Reachability : labels of edges composing into a matched word

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 $G^\#$

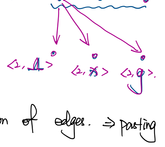
$\cup$ $\forall n \in G^*$. $d \in \mathbb{D}$. $\exists$ a node $\langle n, d \rangle$ in $G^\#$.

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.