Verified C Implementation of Dijksta's Algorithm

Anshuman Mohan, Aquinas Hobor

NUS
National University of Singapore

APLAS Student Research Competition
September 22, 2020

Verified
Software
Toolchain

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

VST + CompCert + 25000 LOC library

Powerful enough to verify executable code
against realistic specifications
expressed with mathematical graphs

[Wang *et. al.*, PACMPL OOPSLA 2019]

This Work

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

Never used edge labels...
Not unlike vertex labels, but let's try anyway

Verify Dijkstra's one-to-all shortest path algorithm

Challenges

Using CompCert C

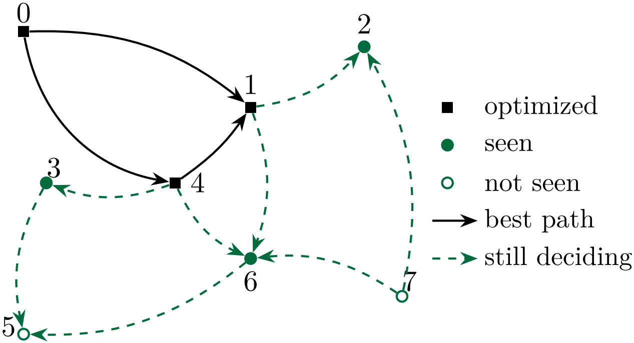
executable and realistic

real-world complications

Aiming for full functional correctness

and not just safety

Refresher



Instantiating DijkGraph

A PreGraph is a hextuple (`VType`, `EType`, `vvalid`, `evalid`, `src`, `dst`)

$$\textbf{Dijk\_PG}(\gamma) \stackrel{\text{def}}{=} \begin{array}{l} \texttt{VType := Z} \\ \texttt{EType := VType * VType} \\ \texttt{src := fst} \\ \texttt{dst := snd} \\ \forall v.\ \texttt{vvalid}(\gamma, v) \Leftrightarrow 0 \leqslant v < \texttt{SIZE} \\ \forall s, d.\ \texttt{evalid}(\gamma, (s, d)) \Leftrightarrow \texttt{vvalid}(\gamma, s) \wedge \texttt{vvalid}(\gamma, d) \end{array}$$

Instantiating DijkGraph

A LabeledGraph is a quadruple (PreGraph, `VL`, `EL`, `GL`)

$$\textbf{Dijk\_LG}(\gamma) \stackrel{\text{def}}{=} \text{Dijk\_PG as shown}$$

```
VL := list EL
EL := Z
GL := unit
```

Instantiating DijkGraph

A GeneralGraph adds arbitrary soundness conditions

$$
\begin{aligned}
\textbf{DijkGraph}(\gamma) \overset{\text{def}}{=}\ & \text{Dijk\_LG as shown, and} \\
& \mathit{FiniteGraph}(\gamma) \wedge \\
& \forall i, j.\ \texttt{vvalid}(\gamma, i) \wedge \texttt{vvalid}(\gamma, j) \Rightarrow \\
& \qquad i = j \Rightarrow \texttt{elabel}(\gamma, (i, j)) = 0 \wedge \\
& \qquad i \neq j \Rightarrow 0 \leqslant \texttt{elabel}(\gamma, (i, j)) \leqslant \lfloor \texttt{MAX/SIZE} \rfloor
\end{aligned}
$$

Representing DijkGraph in Memory

$$\mathsf{list\_rep}(\gamma, i) \stackrel{\text{def}}{=} \mathtt{data\_at\ array\ graph2mat}(\gamma)[i]\ \mathtt{list\_addr}(\gamma, i)$$

$$\mathsf{graph\_rep}(\gamma) \stackrel{\text{def}}{=} \mathop{\circledast}_{\mathtt{vvalid}(\gamma,v)} v \mapsto \mathsf{list\_rep}(\gamma, v)$$

Code and Specification

```
#define IFTY INT_MAX - INT_MAX/SIZE
void dijkstra (int graph[SIZE][SIZE], int src,
                         int *dist, int *prev) {
```

$\{$ DijkGraph$(\gamma)\}$

```
 int pq[SIZE];
 int i, j, u, cost;
 for (i = 0; i < SIZE; i++)
 {  dist[i] = INF; prev[i] = INF; pq[i] = INF;  }
 dist[src] = 0; pq[src] = 0; prev[src] = src;
```

$\{$ DijkGraph$(\gamma) \wedge$ $dijk\_correct(\gamma, src, prev, dist, priq)\}$

```
 // big while loop
```

Code and Specification

$\{\ \mathsf{DijkGraph}(\gamma) \land \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq})\}$

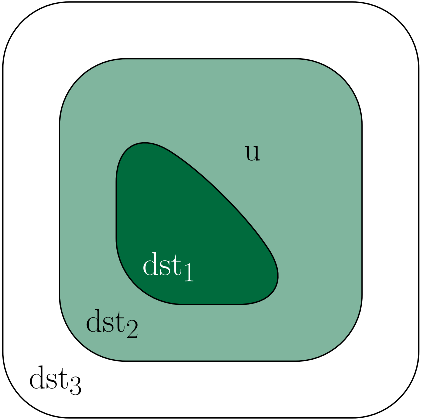
```
 while (!pq_emp(pq)) {
  u = popMin(pq);
  for (i = 0; i < SIZE; i++) {
   cost = graph[u][i];
    if (cost < INF) {
     if (dist[i] > dist[u] + cost) {
      dist[i] = dist[u] + cost; prev[i] = u; pq[i] = dist[i];
 }}}}
```

$\left\{\begin{array}{l} \mathsf{DijkGraph}(\gamma) \land \forall \mathit{dst} \in \mathit{priq}.\ \mathit{priq}[\mathit{dst}] = \mathtt{INF} \land \\ \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \end{array}\right\}$

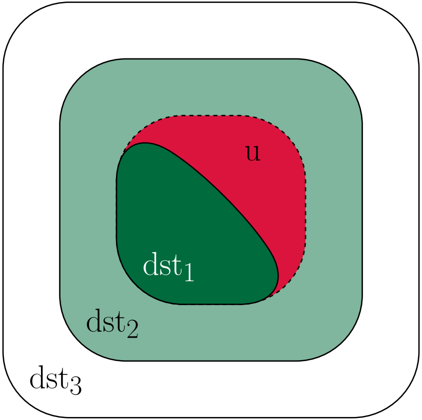
```
return;
}
```

$$
\begin{aligned}
&\textit{dijk\_correct}(\gamma, \textit{src}, \textit{prev}, \textit{dist}, \textit{priq}) \stackrel{\text{def}}{=} \\
&\forall \textit{dst}.\ \textit{dst} \in \textit{popped}(\textit{priq}) \Rightarrow \\
&\qquad\qquad \exists \textit{path}.\ \textit{path\_correct}(\gamma, \textit{prev}, \textit{dist}, \textit{path}) \wedge \\
&\qquad\qquad \textit{path\_glob\_optimal}(\gamma, \textit{dist}, \textit{path}) \wedge \\
&\qquad\qquad \textit{path\_entirely\_in\_popped}(\gamma, \textit{prev}, \textit{path}) \wedge \\
&\qquad \textit{priq}[\textit{dst}] < \texttt{INF} \Rightarrow \\
&\qquad\qquad \texttt{let}\ m\ \texttt{:=}\ \textit{prev}[\textit{dst}]\ \texttt{in}\ m \in \textit{popped}(\textit{priq}) \wedge \\
&\qquad\qquad \forall m' \in \textit{popped}(\textit{priq}).\ \textit{cost}(\textit{path2m}+ :: (m, \textit{dst})) \leqslant \\
&\qquad\qquad\qquad\qquad \textit{cost}(\textit{path2m}'+ :: (m', \textit{dst})) \wedge \\
&\qquad \textit{priq}[\textit{dst}] = \texttt{INF} \Rightarrow \\
&\qquad\qquad \forall m \in \textit{popped}(\textit{priq}).\ \textit{cost}(\textit{path2m}+ :: (m, \textit{dst})) = \texttt{INF}
\end{aligned}
$$

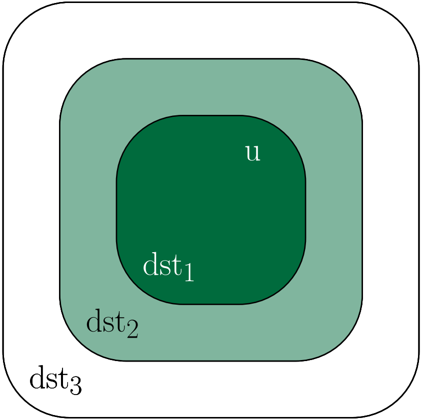
Key Transformation: Growing the Subgraph



Key Transformation: Growing the Subgraph



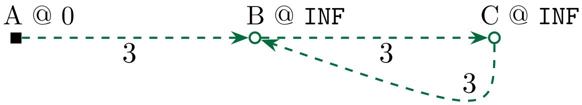
Key Transformation: Growing the Subgraph



Overflow Strikes Again

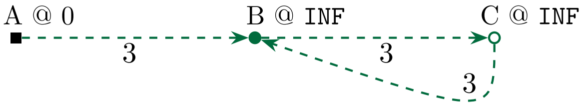
The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

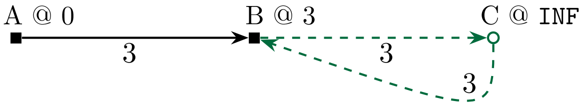


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

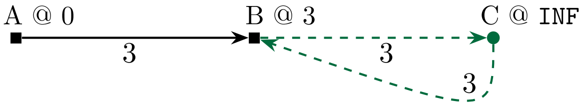
`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

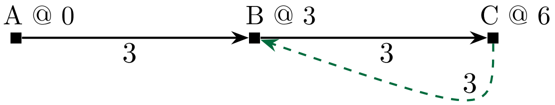
`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

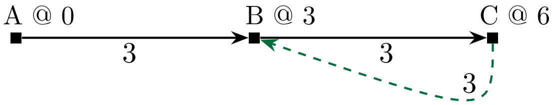
`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor \texttt{MAX/(SIZE-1)} \rfloor$

$\texttt{MAX} = 7$, $\texttt{SIZE} = 3$, so $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.

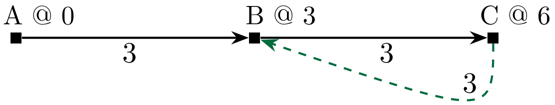


if $3 > 9$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

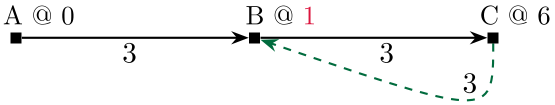


if $3 > 1$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

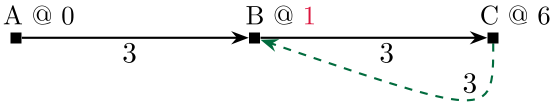


if $3 > 1$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



if $3 > 1$ then relax C $\rightsquigarrow$ B

One solution: Conservatively set upper bound to $\lfloor$`MAX/SIZE`$\rfloor$

Max path cost is then $\lfloor$`MAX/SIZE`$\rfloor$ `* (SIZE-1) = MAX -` $\lfloor$`MAX/SIZE`$\rfloor$

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

That is not the point

- Intuition supports `INF = MAX`
- No reason to do any of the above... until today