Expanding CertiGraph: Dijkstra, Prim, and Kruskal

Anshuman Mohan, Wei Xiang Leow, Aquinas Hobor

NUS Programming Language and Verification Seminar
December 15, 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 + CertiGraph

A Coq library to verify executable code
against realistic specifications
expressed with mathematical graphs

This Work

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

We verify Dijkstra, Prim, Kruskal

This Work

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

We verify Dijkstra, Prim, Kruskal

In doing so, we:

- Test existing features [Dijk labels edges]
- Expand into undirectedness [Prim, Krus]
- Make nontrivial calls to verified methods [Krus calls UF]

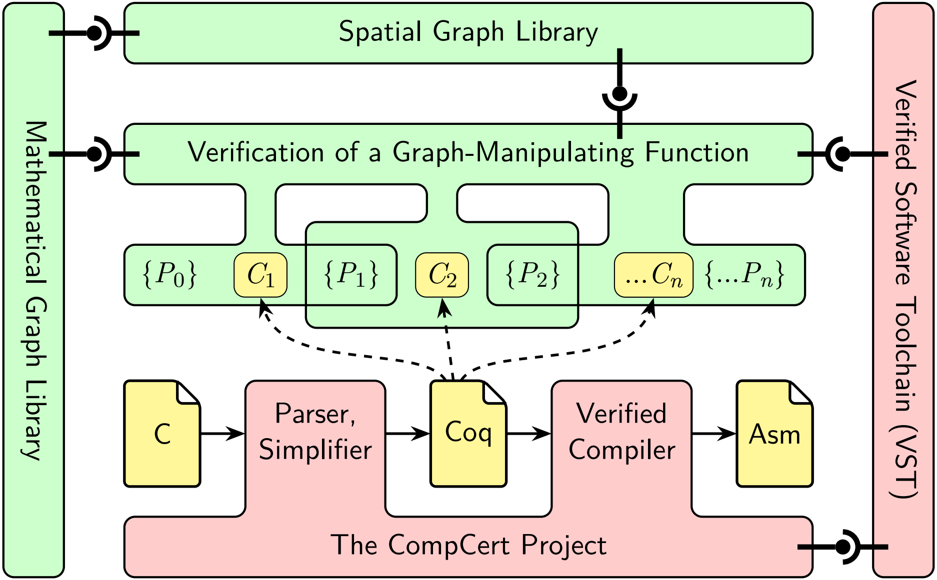
Challenges

Using CompCert C, which
  is executable and realistic
  but has real-world complications

Aiming for full functional correctness

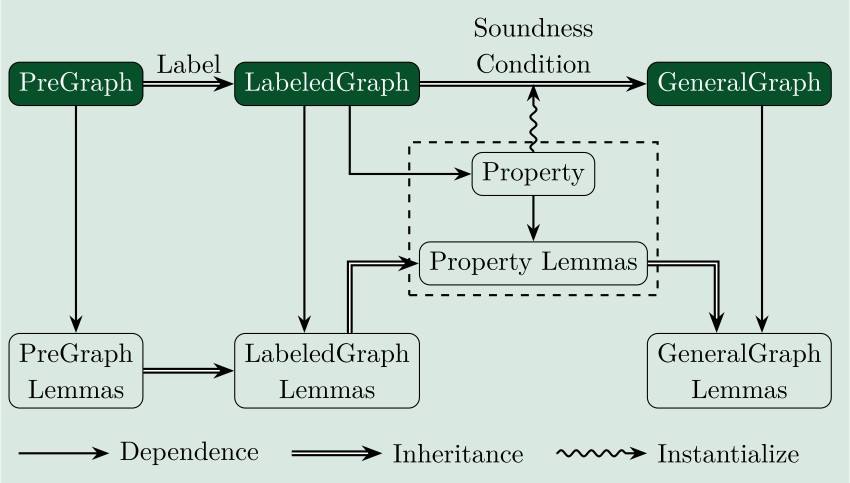
Maintaining modularity and reuse

Workflow



Spatial Graph Library
Mathematical Graph Library
Verification of a Graph-Manipulating Function
Verified Software Toolchain (VST)
$\{P_0\}$
C_1
$\{P_1\}$
C_2
$\{P_2\}$
$\ldots C_n$
$\{\ldots P_n\}$
C
Parser, Simplifier
Coq
Verified Compiler
Asm
The CompCert Project

Math Graph Architecture



Instantiating DijkGraph

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

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

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

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

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

A GeneralGraph adds arbitrary soundness conditions

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

A GeneralGraph adds arbitrary soundness conditions

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

Representing DijkGraph in Memory

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

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

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

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

Relies on restrictions placed at the Math level

Code and Specification

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

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

Code and Specification

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

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

```
 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;
```

Code and Specification

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

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

```
 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;
```

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

```
 // big while loop
```

Code and Specification

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

Code and Specification

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

```
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];
}}}}
```

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];
}}}}
```

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

```
return;
}
```

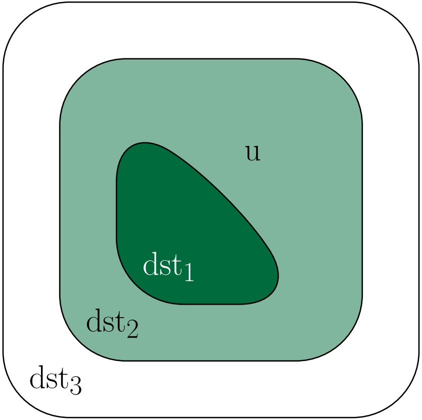
$$
\begin{aligned}
& \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \stackrel{\text{def}}{=} \\
& \quad \forall \mathit{dst}.\ \mathit{dst} \in \mathit{popped}(\mathit{priq}) \Rightarrow \\
& \qquad\qquad\qquad \exists \mathit{path}.\ \mathit{path\_correct}(\gamma,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

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

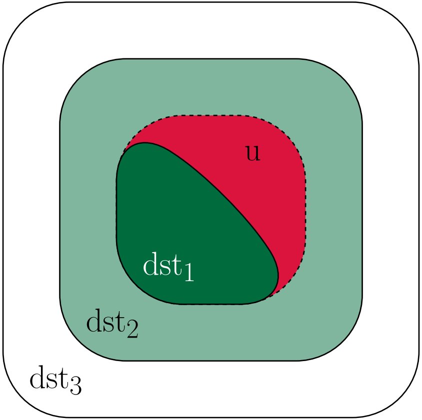
$$
\begin{aligned}
&\mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \stackrel{\text{def}}{=} \\
&\quad \forall \mathit{dst}.\ \mathit{dst} \in \mathit{popped}(\mathit{priq}) \implies \\
&\qquad\qquad \exists \mathit{path}.\ \mathit{path\_correct}(\gamma, \mathit{prev},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

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

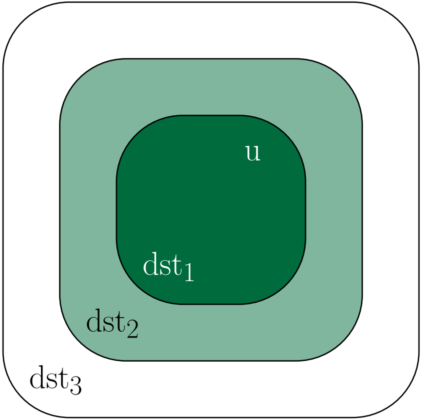
Key Transformation: Growing the Subgraph



Key Transformation: Growing the Subgraph



Key Transformation: Growing the Subgraph



Overflow Strikes Again

The longest optimal path has `size-1` links

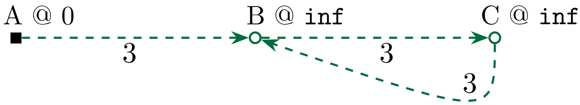
Overflow Strikes Again

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

Overflow Strikes Again

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

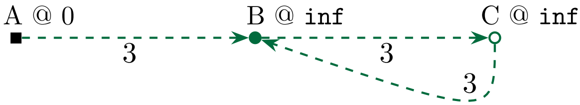
`MAX` $= 7$, `size` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



Overflow Strikes Again

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

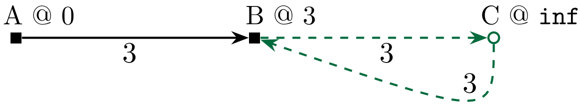
$\texttt{MAX} = 7$, $\texttt{size} = 3$, so $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.



Overflow Strikes Again

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

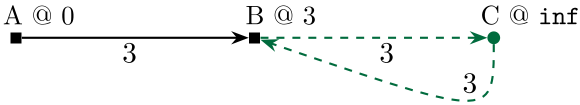
`MAX` $= 7$, `size` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



Overflow Strikes Again

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

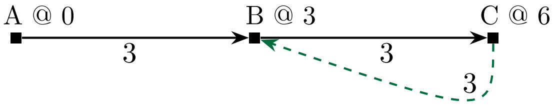
$\texttt{MAX} = 7$, $\texttt{size} = 3$, so $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.



Overflow Strikes Again

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

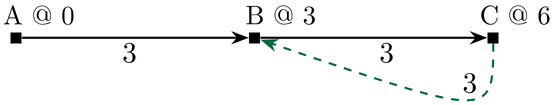
$\texttt{MAX} = 7$, $\texttt{size} = 3$, so $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.



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 $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.

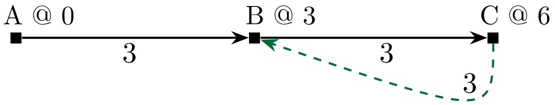


if $3 > 9$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

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

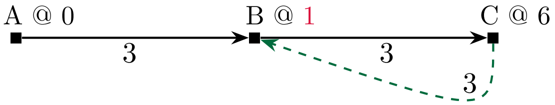
`MAX` $= 7$, `size` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



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

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 $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$.

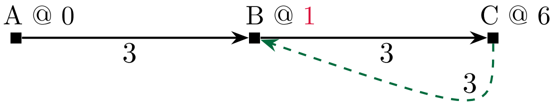


if $3 > 1$ then relax C $\rightsquigarrow$ B

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`$(\gamma, e) \leqslant 3$.

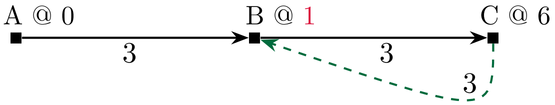


if $3 > 1$ then relax C $\rightsquigarrow$ B

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

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!

Overflow Strikes Again

There are other ways to fix this!

Refactor troublesome addition as subtraction

Overflow Strikes Again

There are other ways to fix this!

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

Overflow Strikes Again

There are other ways to fix this!

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

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

Overflow Strikes Again

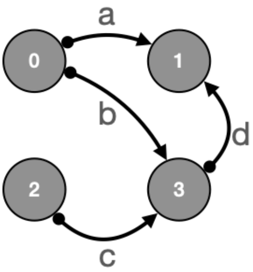
There are other ways to fix this!

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

Sadly, intuition supports `inf = MAX`

Extending to get undirectedness

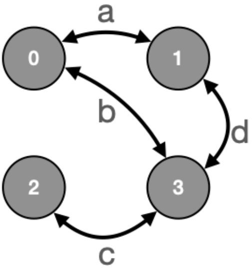
Consider the AdjMat representation of a directed graph:



	0	1	2	3
0	0	a	inf	b
1	inf	0	inf	inf
2	inf	inf	0	c
3	inf	d	inf	0

Extending to get undirectedness

Versus that of an undirected graph:



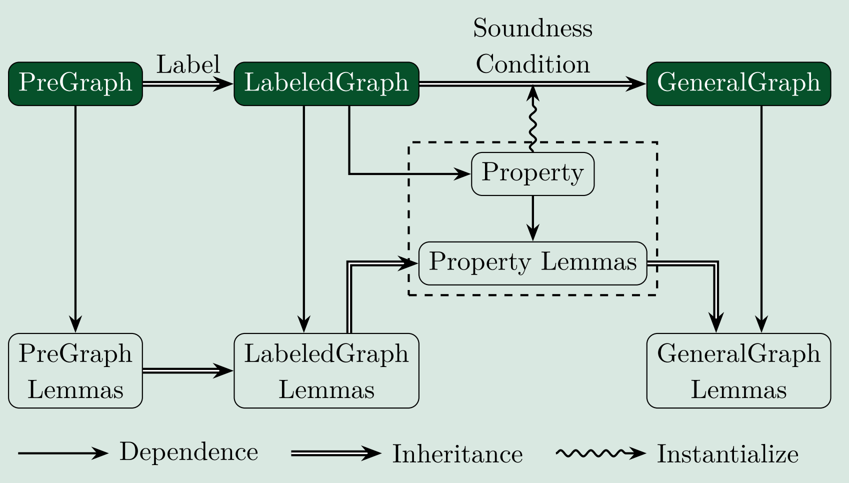
	0	1	2	3
0	0	a	inf	b
1	a	0	inf	d
2	inf	inf	0	c
3	b	d	c	0

Extending to get undirectedness

Thanks to our model, this is remarkably easy to model:

```
Class SoundUAdjMat (g: UAdjMatLG) := {
  sadjmat: @SoundAdjMat size inf g;
  undirec: forall e, evalid g e -> src g e <= dst g e;
}.
```

Recall: Math Graph Architecture



A note on modularity