MACHINE-CHECKED C IMPLEMENTATION OF DIJKSTRA

Anshuman Mohan   Aquinas Hobor
National University of Singapore

Overview

We report on a machine-checked proof of correctness for Dijkstra's one-to-all shortest path algorithm. Unlike previous work, we use classic textbook code written in executable C [2].

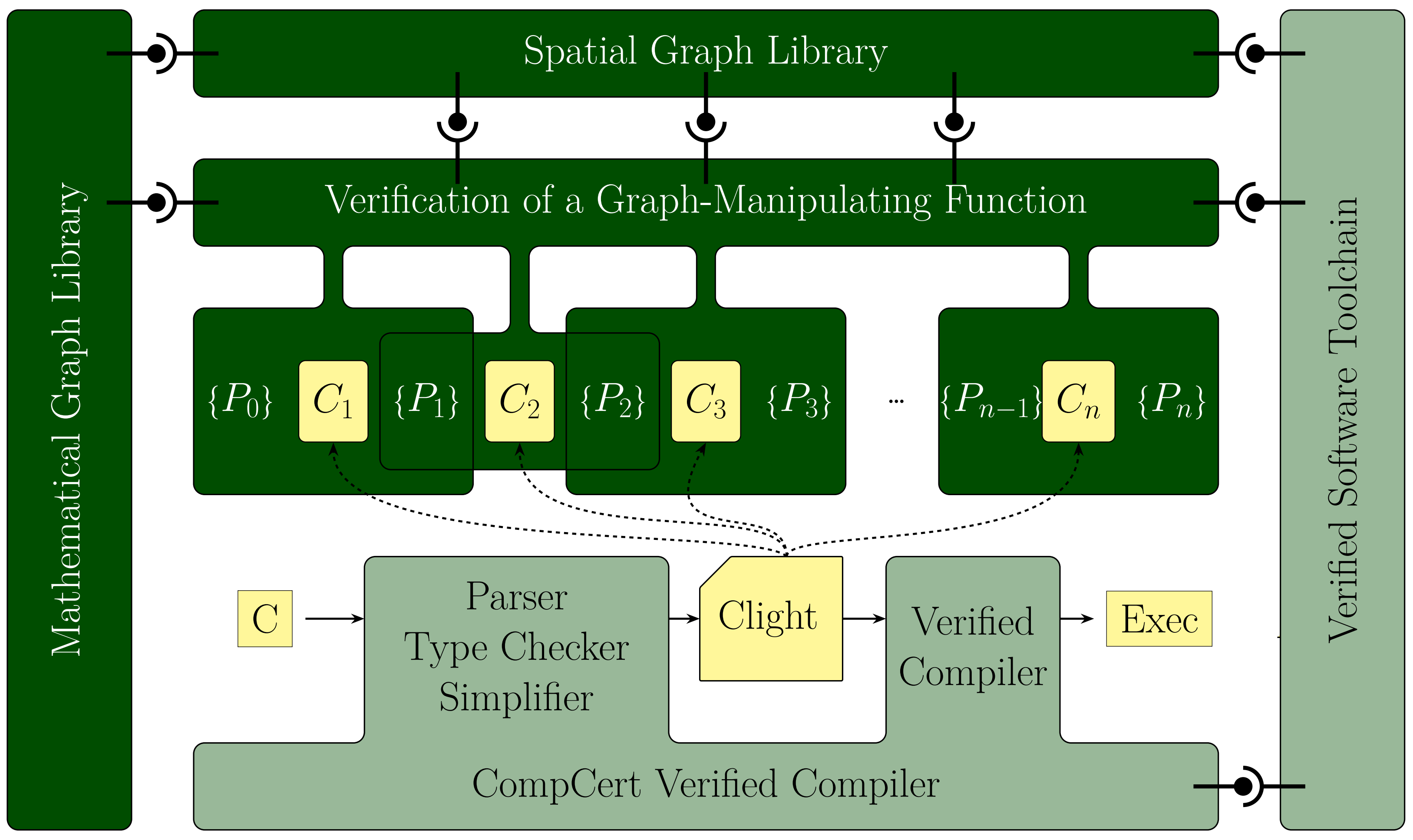
Challenges: We use CompCert C, which is executable and realistic but also has real-world complications. We prove full functional correctness, and not just program safety.

Solution: We use the Verified Software Toolchain [1] and our Mathematical and Spatial Graph Libraries [4] to establish machine-checked correctness at the C level (3 files, 3k LOC). The CompCert compiler [3] then guarantees this correctness on the executable code.

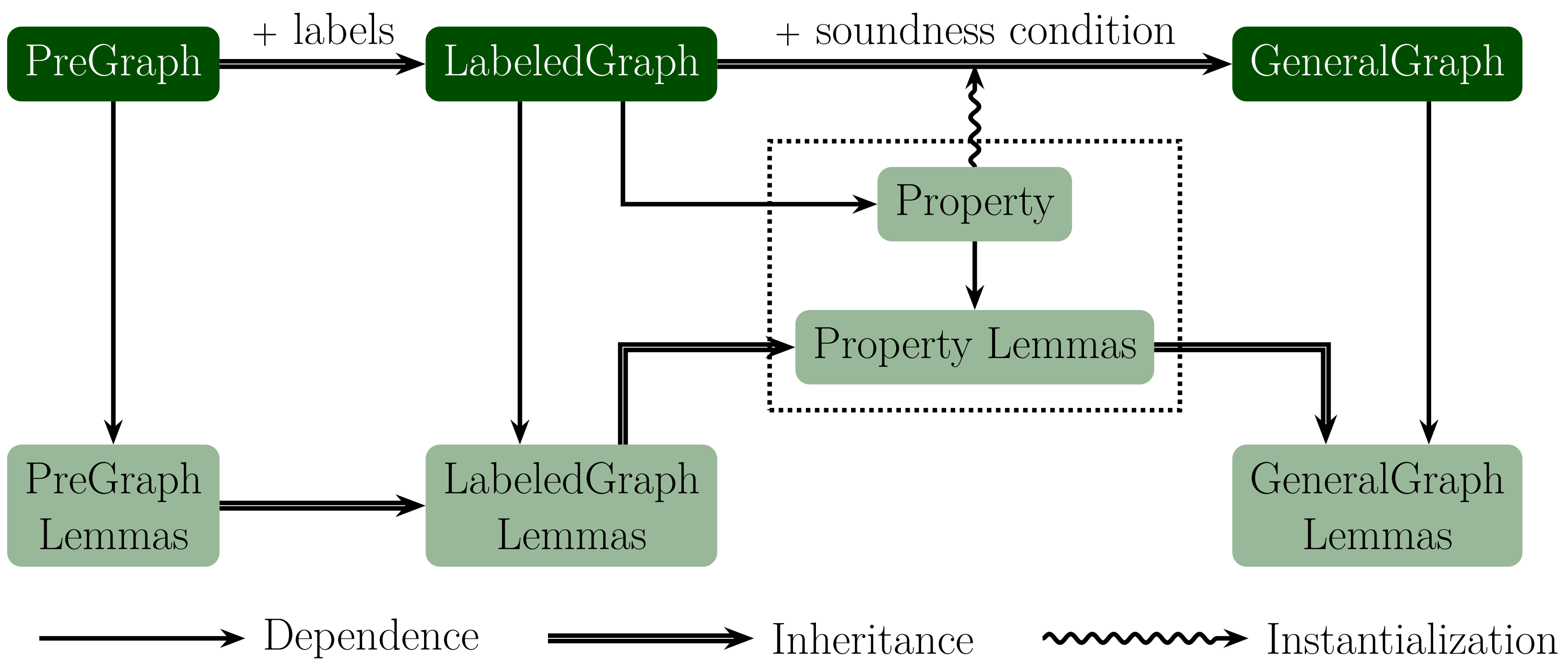
Key findings: The algorithm suffers from potential overflow issues. The precise bound is nontrivial: we show that the intuitive guess fails, and provide a workable refinement.

Workflow

A sketch of how we verify C programs. Note where we integrate with other projects.



The structure of our Mathematical Graph Library. The soundness condition is entirely customizable. Lemmas and properties can be composed, and are automatically inherited.



Instantiating DijkGraph

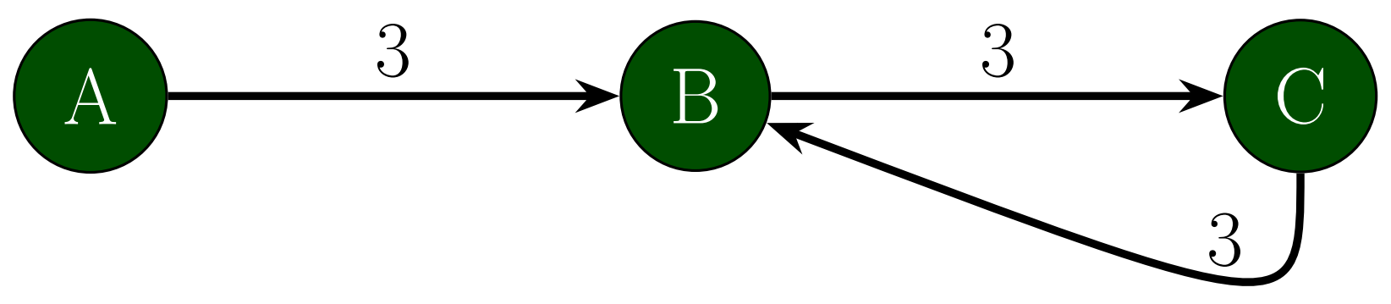
PreGraph: `VType := Z`, `EType := Z * Z`, `src := fst`, `dst := snd`,
$\forall v.\ \texttt{vvalid}(\gamma, v) \Leftrightarrow 0 \leqslant v < \texttt{SIZE}$,
$\forall s, d.\ \texttt{evalid}(\gamma, (s, d)) \Leftrightarrow \texttt{vvalid}(\gamma, s) \wedge \texttt{vvalid}(\gamma, d)$

LabeledGraph: `ELType := Z`, `VLType := list ELType`, `GLType := unit`

GeneralGraph: $\mathit{FiniteGraph}(\gamma) \wedge$
$\forall i, j.\ \texttt{vvalid}(\gamma, i) \wedge \texttt{vvalid}(\gamma, j) \Rightarrow$
$i = j \Rightarrow \texttt{elabel}(\gamma, (i, j)) = 0 \wedge$
$i \neq j \Rightarrow 0 \leqslant \texttt{elabel}(\gamma, (i, j)) \leqslant \texttt{MAX/SIZE}$

Upper Bound on Path Cost

The longest optimal path has `SIZE-1` links, so say we set `elabel`'s upper bound to $\lfloor\texttt{MAX/(SIZE-1)}\rfloor$. Consider the following example, where $\texttt{MAX} = 7$ and $\texttt{SIZE} = 3$, and so $0 \leqslant \texttt{elabel}(\gamma, e) \leqslant 3$. The check on line 20 overflows when scanning C's neighbors: $(\lfloor\texttt{MAX/(SIZE-1)}\rfloor\ \texttt{* (SIZE-1)}) + \lfloor\texttt{MAX/(SIZE-1)}\rfloor\ \texttt{> MAX}$.



Solution: Conservatively set the upper bound for an individual edge to $\lfloor\texttt{MAX/SIZE}\rfloor$.
New worst case: line 20 calculates $(\lfloor\texttt{MAX/SIZE}\rfloor\ \texttt{* (SIZE-1)})\ \texttt{+}\ \lfloor\texttt{MAX/SIZE}\rfloor\ \texttt{= MAX}$.
The true maximum path cost is $\lfloor\texttt{MAX/SIZE}\rfloor\ \texttt{* (SIZE-1) = MAX -}\ \lfloor\texttt{MAX/SIZE}\rfloor$.

Code and Specification

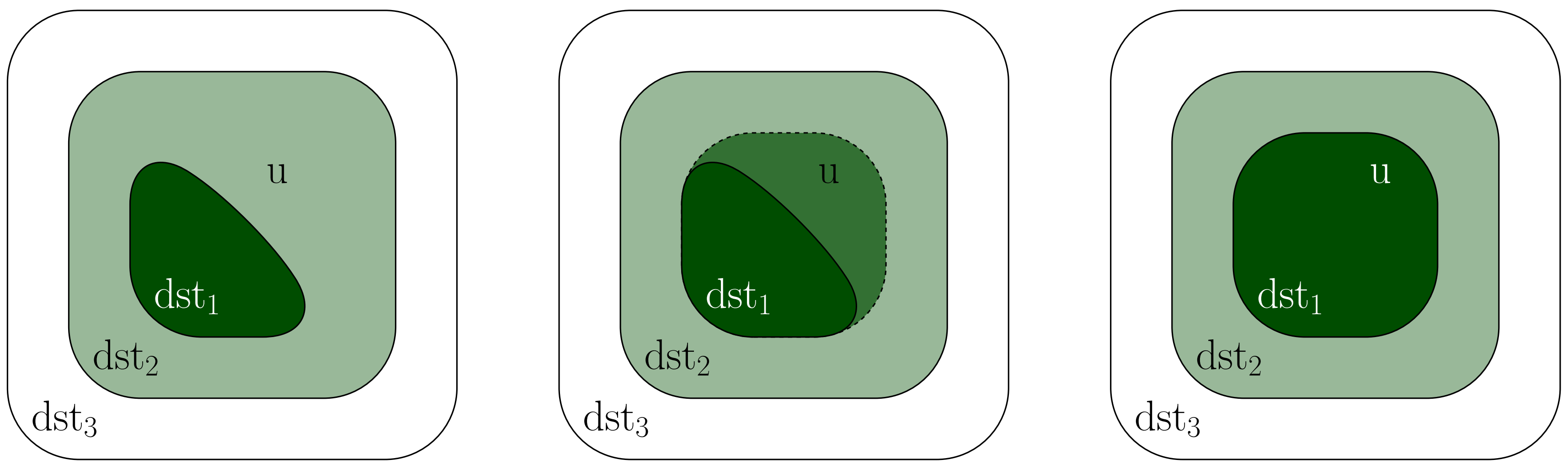
```
1   void dijkstra (int graph[SIZE][SIZE], int src,
2                  int *dist, int *prev) {
3 // { DijkGraph(γ) }
4     int pq[SIZE];
5     int i, j, u, cost;
6     for (i = 0; i < SIZE; i++) {
7       dist[i] = INF;
8       prev[i] = INF;
9       pq[i] = INF;
10    }
11    dist[src] = 0;
12    pq[src] = 0;
13    prev[src] = src;
14 // { DijkGraph(γ) ∧
      dijk_correct(γ, src, prev, dist, priq) }
15    while (!pq_emp(pq)) {
16      u = popMin(pq);
17      for (i = 0; i < SIZE; i++) {
18        cost = graph[u][i];
19        if (cost < INF) {
20          if (dist[i] > dist[u] + cost) {
21            dist[i] = dist[u] + cost;
22            prev[i] = u;
23            pq[i] = dist[i];
24          }
25        }
26      }
27    }
28 // { DijkGraph(γ) ∧
      ∀dst ∈ priq.  priq[dst] = INF ∧
      dijk_correct(γ, src, prev, dist, priq) }
29    return;
30  }
```

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

$$\mathsf{graph\_rep}(\gamma) \triangleq \mathop{\circledast}_{\texttt{vvalid}(\gamma,v)} v \mapsto \mathsf{list\_rep}(\gamma, v)$$

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

Key Transformation: Growing the Subgraph



To begin, vertices dst_1, dst_2, and dst_3 obey the first, second, and third clauses of the invariant *dijk_correct* respectively. Vertex u obeys the second clause with minimal cost.

The invariant is broken when relaxing u's neighbors, and reestablished thereafter: u now obeys the first clause. Eventually no vertices obey the second clause, and we are done.

References

[1] Andrew Appel. "Verified Software Toolchain". In: *NFM 2012, Norfolk, USA, April 3-5*. 2012, p. 2.

[2] Thomas Cormen et al. *Introduction to Algorithms*. MIT Press, 2009.

[3] Xavier Leroy. "Formal certification of a compiler back-end or: programming a compiler with a proof assistant". In: *POPL 2006, Charleston, USA, January 11-13*. 2006, pp. 42–54.

[4] Shengyi Wang et al. "Certifying graph-manipulating C programs via localizations within data structures". In: *OOPSLA 2019, Athens, Greece, October 20-25*. 2019, pp. 171:1–171:30.