CoVaC

Compiler Validation by Program Analysis of the Cross-Product

Ramanuj Goel

November 9, 2023

What is CoVaC

CoVaC is a dedutive framework for proving program equivalence, and also a tool by the same name. Although the paper specifically talks about its uses for verification of compilations, it can be generally used for proving equivalence of any two programs.

Differences from what we've learned so far

- CoVaC is different from Alive. While Alive proves the correctness of **compilers**, CoVaC proves the correctness of **compilations**.
- This means that while CoVaC can be used to test compilers, it cannot be used to prove compilers like Alive. It can instead be used to prove correctness of compilations.
- Furthermore, due to solving an easier problem than Alive, CoVaC is more general and can work on more kinds of transforms.
- CoVaC is also different from Infer or ESC/Java. These tools help to test the source code against bugs. CoVaC on the other hand will ensure that if the source has bugs then so does the compiled code.

How it works

- The implementation of CoVaC given in the paper works on intraprocedural transformations (ie: changees within the same function).
- Furthermore it requires that the source and target are structurally similar or consonant, which we define formally later.
- This means that passes like polly with tiling cannot be transformed. However it is still more generalized than the peephole optimizations that are verified by Alive.
- At a high level, CoVaC creates a graph of the program (with instructions as edges) for both the source and the target and then combines them together in a ”cross product”, so a node/edge in the new graph is a pair of nodes/edges from the source and target program graph.
- Because this method can potentially lead to a quadratic blowup and also have incorrect edges (edges which are never used), CoVaC uses an SMT solver to ”Align the branches” as the graph gets created.

Transition Graphs

Now we formally define the model which is used by the paper.

- We denote program as a list of functions as $f_1 \dots f_m$ and also a *main* function.
- Functions in the paper are represented as transition graphs. Each function can be defined as a tuple of variables $\vec{y}$, nodes (locations) $\mathcal{N}$ and labeled edges $\mathcal{E}$.
- The variables $\vec{y}$ contains three kinds of variables, $\vec{x}, \vec{z}$ and $\vec{w}$. $\vec{x}$ are the pass-by-value input variables. $\vec{z}$ are the pass-by-reference input variables and $\vec{w}$ are the local variables.
- The nodes $\mathcal{N}$ are numbered 0 to k like: $\mathcal{N} = \{r = n_0, n_1, \dots n_k = t\}$. $n_0 = r$ is the unique starting node while $n_k = t$ is the unique exit node. Every node is on a path from r to t.
- The edges $\mathcal{E}$ are directed and are labeled with instructions.
- We use the notation $E(\vec{y})$ for a list of expressions over the variables $\vec{y}$.

Instructions

The paper generalizes instructions into 4 kinds. These are:

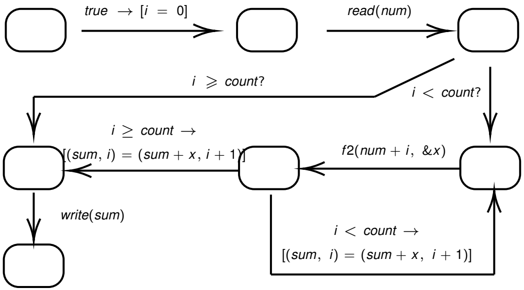
- **Guarded Assignments:** Denoted as $c \to [\vec{u} := E(\vec{y})]$, where c is a boolean expression, $\vec{u}$ is a list of variables. It means that if c is true then the variables $\vec{u}$ are each assinged the expressions $E(\vec{y})$. If there is a boolean condition with no assignments, we write c?.
- **Function Calls:** Denoted as $f(E(\vec{y}), \vec{u})$. Here the first tuple of arguments, $E(\vec{y})$, are pass-by-value and the next, $\vec{u}$, are pass-by-reference. The return values are just returned using the pass-by-reference variables.
- **Reads and Writes:** Denoted as $read(\vec{u})$ and $write(\vec{u})$, they are used to denote IO operations.

A property which we require from the transition graphs is deterministic and non-blocking. Deterministic means that for every node, the guards on all outgoing edges are mutually exclusive. Non-blocking means that the disjunction of all guards on all outgoing edges is a tautology.

Example of a Transition Graph

We will take an example of a simple function written in psuedocode.

```
def f1(count, &sum):
    i = 0
    read(num)
    while (i < count):
        x = f2(num+i)
        sum = sum + x
        i = i + 1
    write(sum)
```



States and Computations

- A **state** is a pair $\langle n; \vec{d} \rangle$ with the node n and $\vec{d}$ is the initialization on the values $\vec{y} = (\vec{x}, \vec{z}, \vec{w})$.
- A **computation** of a module is essentially a run of the module on some specific input.
- The paper defines a $(\vec{\xi}, \vec{\zeta}) -$ *computation* on a function $f(\vec{x}, \&\vec{z})$ as its run with the input $\vec{x} = \vec{\xi}$ and $\vec{z} = \vec{\zeta}$.
- A computation looks like:

  $$\sigma : \langle r; (\vec{\xi}, \vec{\zeta}, \vec{T}) \rangle \xrightarrow{\lambda_1} \langle n_1; \vec{d_1} \rangle \xrightarrow{\lambda_2} \langle n_2; \vec{d_2} \rangle \ldots$$

  Here, $\vec{T}$ denotes uninitialized local variables while the labels of the arrows $(\lambda_1, \lambda_2 \ldots)$ are the lables of the edges in the transition graph or the special lable *ret*. Each transition must be justified by either an intra-procedural transition, a call transition, or a return transition such that the call and return transitions are balanced .
- $Cmp(f)$ denotes all the computations of a module f (or equivalently, a transition graph f). For a procedural program $\mathcal{A}$, we denote the set of computations $Cmp(\mathcal{A}) = Cmp(main)$

Observable variables

The correctness of a translation, which means equivalence of the source and the target, depends upon the values which get assigned to the variables. However not all values matter and we shouldn't really have a one-to-one mapping of varialbes in the source and the target. Hence we define V_s as the set of **observable variables** at a state $s = \langle n, \vec{d} \rangle$ as:

- If s is a state immediately after transition $read(\vec{u})$, $V_s \supseteq \vec{u}$
- If s is a state immediately before transition $write(\vec{u})$, $V_s \supseteq \vec{u}$
- If $n = r$ is the entry node of procedure $f(\vec{x}, \&\vec{z})$, $V_s \supseteq \vec{x} \wedge V_s \supseteq \vec{z}$
- If $n = t$ is the exit node of procedure $f(\vec{x}, \&\vec{z})$, $V_s \supseteq \vec{z}$

Intuitively, we wish to maintain that in both the source and target programs these variables take the same values.

Observation function

We also define an **Observation function** $\mathcal{O}$ which maps the states and labels (the ones above arrows) of a computation to a common domain. $\mathcal{O}$ is defined as follows:

- For a state s, if $V_s = \emptyset$ then $\mathcal{O} = \perp$
- For a state $s = \langle n, \vec{d} \rangle$, if $V_s \neq \emptyset$ then $\mathcal{O} = \vec{d_{V_s}}$, where $\vec{d_{V_s}}$ is the restriction of $\vec{d}$ on the variables V_s, so just the values of the variables V_s at that state.
- For a label λ, if it is a label of a read, write, call (to procedure g) or return (from procedure g) then $\mathcal{O}(\lambda)$ takes the values $read$, $write$, $call_g$ and ret_g respectively. Otherwise $\mathcal{O}(\lambda) = \perp$.

An **observartion** of a computation σ, denoted $o(\sigma)$ is obtained by the application of $\mathcal{O}$ on all states and labels of σ. That is, for $\sigma : s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} s_3 \ldots$ we get $o(\sigma) : \mathcal{O}(s_1) \xrightarrow{\mathcal{O}(\lambda_1)} \mathcal{O}(s_2) \xrightarrow{\mathcal{O}(\lambda_2)} \mathcal{O}(s_3) \ldots$

Correct translation

We define two computations σ and σ' to be **stuttering equivalent**, denoted $\sigma \sim_{st} \sigma'$, if their observations $o(\sigma)$, $o(\sigma')$ only differ from each other by a finite sequence of pairs $\perp \xrightarrow{\perp}$ or $\xrightarrow{\perp} \perp$.

Intuitively, for two computations to be stuttering equivalent, functions need to be called in the same order with the same input parameters and also return the same values, read calls should get the same inputs and write calls should write the same outputs. We can have extra or fewer computations in between these things, but that is ignored due to them translating to $\perp$. It is called "stuttering" because the only way for a user to distinguish the two computations is the time taken, or "stutter".

$f_{\mathcal{T}}$ is a **correct translation** of procedure $f_{\mathcal{S}}$ if for every $(\vec{\xi}, \vec{\zeta})$-computation $\sigma_{\mathcal{T}}$ in $Cmp(f_{\mathcal{T}})$ there exists a $(\vec{\xi}, \vec{\zeta})$-computation $\sigma_{\mathcal{S}}$ in $Cmp(f_{\mathcal{S}})$ such that $\sigma_{\mathcal{T}} \sim_{st} \sigma_{\mathcal{S}}$, and vice-versa. Program $\mathcal{T}$ is a correct translation of program $\mathcal{S}$ if $main_{\mathcal{T}}$ is a correct translation of $main_{\mathcal{S}}$.

Intuitively, $\mathcal{T}$ is a correct translation of $\mathcal{S}$ if for every computation for every input on $\mathcal{S}$ we can have a stuttering equivalent computation in $\mathcal{T}$ on the same input.

Comparison Graphs

We assume that we have two programs, $\mathcal{S}$ and $\mathcal{T}$ which we have to prove equivalent. For each pair of the corresponding source and target procedures $f^{\mathcal{S}} = (\vec{y}^{\mathcal{S}}, \mathcal{N}^{\mathcal{S}}, \mathcal{E}^{\mathcal{S}})$ and $f^{\mathcal{T}} = (\vec{y}^{\mathcal{T}}, \mathcal{N}^{\mathcal{T}}, \mathcal{E}^{\mathcal{T}})$ we define a **comparison transition graph** $f = (\vec{y}, \mathcal{N}, \mathcal{E}) = f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$ as a graph which satisfies a specific set of 3 rules. The collection of comparison graphs for all procedures constitutes the comparison program $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$.

Rule1 (Structural Requirement)

1. The variables $\vec{y} = (\vec{x}, \vec{z}, \vec{w})$ are concatenations of the variables of the source and target programs, that is: $\vec{x} = \vec{x}^{\mathcal{S}} \circ \vec{x}^{\mathcal{T}}$, $\vec{z} = \vec{z}^{\mathcal{S}} \circ \vec{z}^{\mathcal{T}}$ and $\vec{w} = \vec{w}^{\mathcal{S}} \circ \vec{w}^{\mathcal{T}}$
2. Each node of f is a pair of source and target nodes: $\mathcal{N} \subseteq \mathcal{N}^{\mathcal{S}} \times \mathcal{N}^{\mathcal{T}}$. The entry and exit nodes are $r = \langle r^{\mathcal{S}}, r^{\mathcal{T}} \rangle$ and $t = \langle t^{\mathcal{S}}, t^{\mathcal{T}} \rangle$ respectively.
3. Every edge of the graph $(\langle n^{\mathcal{S}}, n^{\mathcal{T}} \rangle \xrightarrow{\langle op^{\mathcal{S}}; op^{\mathcal{T}} \rangle} \langle m^{\mathcal{S}}, m^{\mathcal{T}} \rangle) \in \mathcal{E}$ should satisfy one of these conditions:
   - $(n^{\mathcal{S}} \xrightarrow{op^{\mathcal{S}}} m^{\mathcal{S}}) \in \mathcal{E}^{\mathcal{S}}$; $(n^{\mathcal{T}} \xrightarrow{op^{\mathcal{T}}} m^{\mathcal{T}}) \in \mathcal{E}^{\mathcal{T}}$; and $op^{\mathcal{S}}$ and $op^{\mathcal{T}}$ have the same type (either both reads, writes, assignments, or calls to procedures with the same name).
   - $(n^{\mathcal{S}} \xrightarrow{op^{\mathcal{S}}} m^{\mathcal{S}}) \in \mathcal{E}^{\mathcal{S}}$ where $op^{\mathcal{S}}$ is an **assignment**; $n^{\mathcal{T}} = m^{\mathcal{T}}$ and $op^{\mathcal{T}} = \epsilon$
   - $(n^{\mathcal{T}} \xrightarrow{op^{\mathcal{S}}} m^{\mathcal{T}}) \in \mathcal{E}^{\mathcal{T}}$ where $op^{\mathcal{T}}$ is an **assignment**; $n^{\mathcal{S}} = m^{\mathcal{S}}$ and $op^{\mathcal{S}} = \epsilon$

   Here $\epsilon = $ *true*?, equivalent to nop.

Comparison Graphs

A composed transition $\langle n; \vec{d} \rangle \xrightarrow{e^{\mathcal{S}}; e^{\mathcal{T}}} \langle n'; \vec{d}' \rangle$ in interpreted as a sequential composition of the source and target transitions with one exception, which is when $e^{\mathcal{S}}; e^{\mathcal{T}}$ are $read(\vec{u}^{\mathcal{S}}); read(\vec{u}^{\mathcal{T}})$. Here the variables written by the read need to have the same value in $\vec{d}'$.

Given σ in $Cmp(f)$, we use $\sigma\uparrow_{\mathcal{S}}$ to denote a path obtained by projection of σ onto the states and transitions related to module $f^{\mathcal{S}}$.

Rule 2 There doesn't exist σ in $Cmp(f)$ such that $\sigma\uparrow_{\mathcal{S}}$ or $\sigma\uparrow_{\mathcal{T}}$ contain an infinite sequence of ϵ-transitions.

From Rule 1 and 2, we get:

$$\forall \sigma \in Cmp(f) : (\exists \sigma^{\mathcal{S}} \in Cmp(f^{\mathcal{S}}) : \sigma^{\mathcal{S}} \sim_{st} \sigma\uparrow_{\mathcal{S}}) \wedge (\exists \sigma^{\mathcal{T}} \in Cmp(f^{\mathcal{T}}) : \sigma^{\mathcal{T}} \sim_{st} \sigma\uparrow_{\mathcal{T}})$$

This means that every computation of the comparison graph has the corresponding computation in both the source and the target. We would also like the reverse to hold.

We say that the computations of an input system, say $Cmp(f^{\mathcal{S}})$ are **covered** by $Cmp(f)$ when the following condition holds:

$$\forall \sigma^{\mathcal{S}} \in Cmp(f^{\mathcal{S}}, \exists \sigma \in Cmp(f) : \sigma^{\mathcal{S}} \text{ differs from } \sigma\uparrow_{\mathcal{S}} \text{ by only finite padding}$$

Padding is defined as ϵ-transitions. This notion is stronger than stuttering equivalence so it follows that $\sigma^{\mathcal{S}} \sim_{st} \sigma\uparrow_{\mathcal{S}}$

Comparison Graphs

Rule 3 Computations of $f^{\mathcal{S}}$ and computations of $f^{\mathcal{T}}$ are covered by $Cmp(f)$.

This means that every run of programs that could have happened in the source or the target can also happen in the combined comparison graph. Note that this allows the comparison graph to discard some nodes/edges from the source or the target if they are not reachable.

A comparison graph $f = f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$ is a **witness of correct translation** if for every $((\xi \circ \xi), (\zeta \circ \zeta))$-computation of f, its target and source projections have equal observations. As you can see, we restrict the input of f to be the same for both $\mathcal{S}$ and $\mathcal{T}$.

Theorem 1 The target $f^{\mathcal{T}}$ is a correct translation of source function $f^{\mathcal{S}}$ iff there exists a witness comparison graph $f = f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$. In addition, if $f^{\mathcal{T}}$ is a correct translation of $f^{\mathcal{S}}$ then every comparison graph $f = f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$ is a witness of correct translation.

Thus, in order to determine the correctness of translation, it is sufficient to construct a comparison graph and check if it is, indeed, a witness.

Witness Verification Conditions

An assertion network $\phi = \{\varphi_n | n \in \text{ locations of } \mathcal{C}\}$ for a program $\mathcal{C}$ is said to be **invariant** if for every execution of state $\langle n; \vec{d}\rangle$ in a computation, $\vec{d} \vDash \varphi_n$. That is, on every visit of a computation of node n, the visiting data state satisfies the corresponding assertion φ_n associated with n.

If we have a comparison program $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$ and can construct the strongest invariant network for it, then we can generate and verify the **Witness Verification Conditions**. If these are valid then they prove the correctness of translation. The witness verification conditions are:

- For a write edge $n \xrightarrow{write(\vec{u}^{\mathcal{S}});write(\vec{u}^{\mathcal{T}})} m$, we check that the same values were written:
$$\varphi_n \implies (\vec{u}^{\mathcal{S}} = \vec{u}^{\mathcal{T}})$$
- For a call edge $n \xrightarrow{g^{\mathcal{S}}(E^{\mathcal{S}},\vec{u}^{\mathcal{S}});g^{\mathcal{T}}(E^{\mathcal{T}},\vec{u}^{\mathcal{T}})} m$, we check that the same arguments were passed:
$$\varphi_n \implies (\vec{E}^{\mathcal{S}} = \vec{E}^{\mathcal{T}}) \wedge (\vec{u}^{\mathcal{S}} = \vec{u}^{\mathcal{T}})$$
- If n is the exit node of the transition graph $f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$ where $f^{\mathcal{S}}(x^{\mathcal{S}}, \&\vec{z}^{\mathcal{S}})$ and $f^{\mathcal{T}}(x^{\mathcal{T}}, \&\vec{z}^{\mathcal{T}})$ we check that the pass-by-reference values are same:
$$\varphi_n \implies (\vec{z}^{\mathcal{S}} = \vec{z}^{\mathcal{T}})$$

Consonant Transition Graphs

For a transition graph $f = (\vec{y}, \mathcal{N}, \mathcal{E})$, we classify every node $n \in \mathcal{N}$ to 6 types (denoted $\tau(n)$):

- Read (*rd*): If the outgoing edge from n is a read.
- Write (*wt*): If the outgoing edge from n is a write.
- Call (*cl*): If the outgoing edge from n is a function call.
- Branch (*br*): If n has multiple outgoing edges.
- Unconditional assignment (*wa*): If n has one outgoing edge which is an assignment.
- Exit (*tl*): When $n = t$.

We define a set of cut points, denoted $\mathcal{P}$ as:

$$\mathcal{P} \subseteq \{n : n \in \mathcal{N} \wedge \tau(n) \neq wa\} \quad \text{[Bug]}$$

Every computation σ defines a corresponding sequence of cut points, which is just the nodes in σ intersection with $\mathcal{P}$.

Consonant Transition Graphs

We say that the graphs $f^{\mathcal{S}}$ and $f^{\mathcal{T}}$ are **consonant** if there exists a partial map $\kappa : \mathcal{P}^{\mathcal{S}} \mapsto \mathcal{P}^{\mathcal{T}}$ such that $\forall \sigma^{\mathcal{S}} \in Cmp(f^{\mathcal{S}}), \sigma^{\mathcal{T}} \in Cmp(f^{\mathcal{T}})$: if $\sigma^{\mathcal{S}}$ and $\sigma^{\mathcal{T}}$ are defined by the same input sequence and $n_0^{\mathcal{S}}, n_1^{\mathcal{S}} \ldots$ and $n_0^{\mathcal{T}}, n_1^{\mathcal{T}} \ldots$ are the cut point sequences defined by $\sigma^{\mathcal{S}}$ and $\sigma^{\mathcal{T}}$, then $\kappa(n_i^{\mathcal{S}}) = n_i^{\mathcal{T}} \wedge \tau(n_i^{\mathcal{S}}) = \tau(n_i^{\mathcal{T}})$.

Intuitively, this means that every cut point in $\mathcal{S}$ should map to a cut point in $\mathcal{T}$ such that everytime we go through the cut point in the source, we also go through the cut point in the target in the same order.

Note how consonence of transition graphs depends on the cut point sets that we have chosen. If we take every branch to be a cut point then transforms like loop inversion can not be considered consonant. However, decreasing the size of the cut points (granularity) would make proofs harder. Technically, transforms like tiling and loop interchange could also be proven equivalent if we took the cut point set to not have any branches, however it would be hard for the theorem provers to prove it.

I think a good choice for cut points would be loop heads, reads, writes and exit nodes. Other branch nodes might be added to increase performance, but if they result in the proof failing then we would need to remove them. Keeping loop heads still would result in simple transformations like loop-reordering, loop-fission/fusion or even moving one loop above another to be non-consonant.

Algorithm Compose

The psuedocode for algorithm compose, which constructs comparison graph is given below:

```
$n_0$ := $(n_0^{\mathcal{S}}, n_0^{\mathcal{T}})$; C.Nodes:={$n_0$}; C.Edges:={}; WorkList := {$n_0$};
while(!WorkList.isEmpty()) {
  n := WorkList.removeElement();
  MatchList := matchEdges($n, \mathcal{S}, \mathcal{T}$);
  if(MatchList == NULL) ABORT;
  while(! MatchList.isEmpty()){
    $e_{new}$ := MatchList.removeElement();
    $n_e$ = NewCEdge.toNode();
    C.Nodes.add($n_e$);
    C.Edges.insert($e_{new}$);
    WorkList.add($n_e$);
    WorkList.add(getDescendants($n_e$));
  }
}
```

Function matchEdges

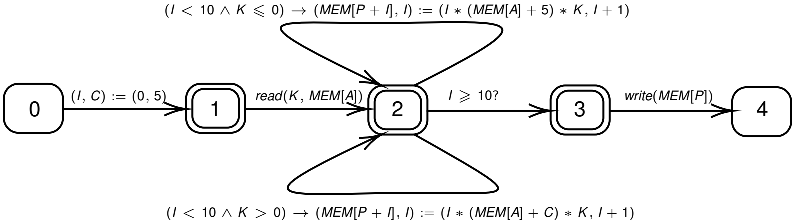
The function matchEdges used in the compose algorithm is supposed to take two nodes and give all pairs of edges which ”match” according to some rules, which are given below:

- **Same type matching**: Given a node $\langle n^{\mathcal{S}}, n^{\mathcal{T}} \rangle$, we match the outgoing edges iff $\tau(n^{\mathcal{S}}) = \tau(n^{\mathcal{T}})$.
- **Adding ϵ-transitions**: If $\tau(n^{\mathcal{S}}) = wa$, we match the cource assignment edge with an an ϵ-transition to the target. The case for $\tau(n^{\mathcal{T}}) = wa$ is handled analogously.
- **Raising error:** If none of the rules are applicable to a node $\langle n^{\mathcal{S}}, n^{\mathcal{T}} \rangle$, matchEdges will return NULL and the construction of $\mathcal{C}$ is aborted.

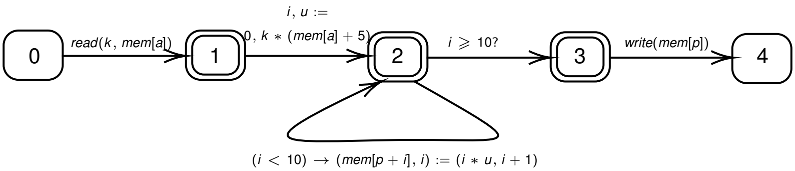
This function isn’t complete right now, as it requires branch alignment, however it is instructive to see how the algorithm works with the given code.

Compose Example

Source:

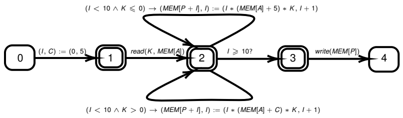

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:


0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

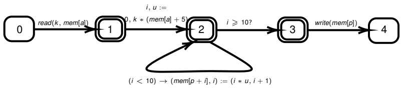
Compose Example

Source:



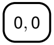
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



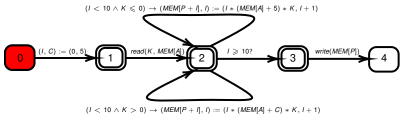
0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(0,0)\}$
Matchlist = $\{\}$

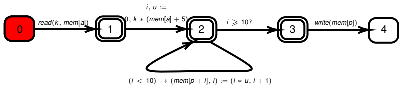


Compose Example

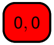
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

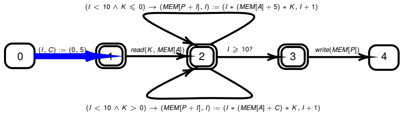


Worklist = $\{\}$
Matchlist = $\{\langle(0 \rightarrow 1), \epsilon\rangle\}$

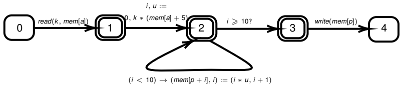


Compose Example

Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

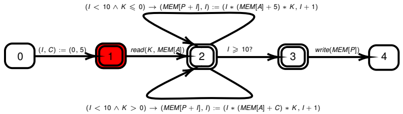

0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(1,0)\}$
Matchlist = $\{\}$

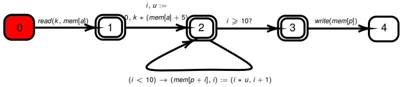
$$\begin{tikzcd}\boxed{0,0} \arrow[r, "{(I, C) := (0,5)}"', "\epsilon"'] & \boxed{1,0}\end{tikzcd}$$

Compose Example

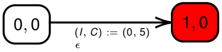
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:

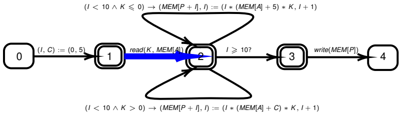


Worklist = $\{\}$
Matchlist = $\{\langle(1 \rightarrow 2), (0 \rightarrow 1)\rangle\}$

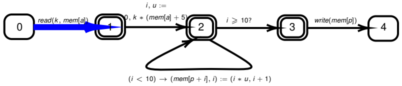


Compose Example

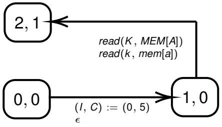
Source:


(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
0
(I, C) := (0, 5)
1
read(K, MEM[A])
2
I ⩾ 10?
3
write(MEM[P])
4
(I < 10 ∧ K > 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + C) ∗ K, I + 1)

Target:

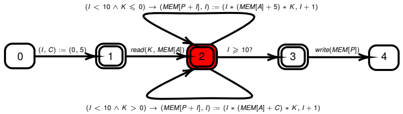


Worklist = $\{(2,1)\}$
Matchlist = $\{\}$

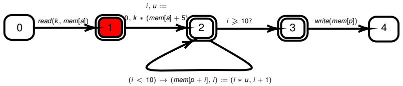


Compose Example

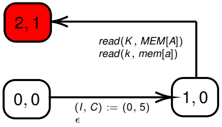
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

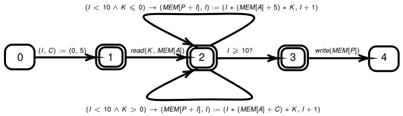


Worklist = $\{\}$
Matchlist = $\{\langle \epsilon, (1 \rightarrow 2) \rangle\}$

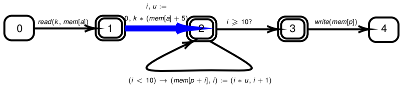


Compose Example

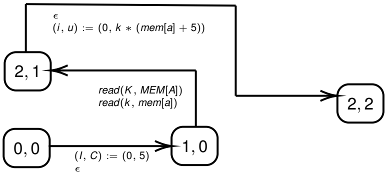
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

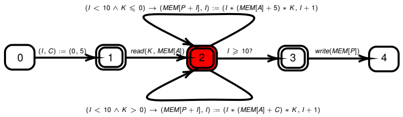

0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(2,2)\}$
Matchlist = $\{\}$

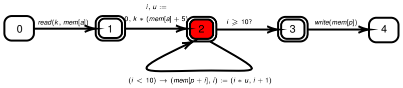


Compose Example

Source:

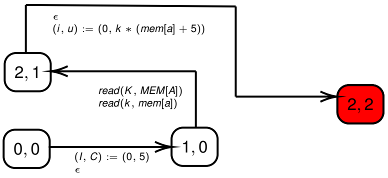

$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



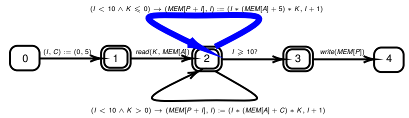
Worklist = $\{\}$

Matchlist = $\{\langle(2 \to 3), (2 \to 3)\rangle, \langle(2 \to 3), (2 \to 2)\rangle, \langle(2 \xrightarrow{2} 2), (2 \to 3)\rangle,$
$\langle(2 \xrightarrow{2} 2), (2 \to 2)\rangle, \langle(2 \xrightarrow{1} 2), (2 \to 3)\rangle, \langle(2 \xrightarrow{1} 2), (2 \to 2)\rangle\}$

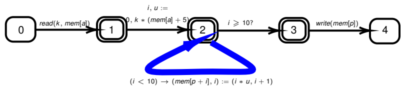


Compose Example

Source:

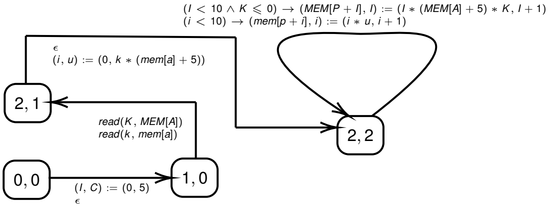

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



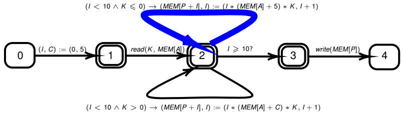
Worklist = $\{(2, 2)\}$

Matchlist = $\{\langle(2 \rightarrow 3), (2 \rightarrow 3)\rangle, \langle(2 \rightarrow 3), (2 \rightarrow 2)\rangle, \langle(2 \xrightarrow{2} 2), (2 \rightarrow 3)\rangle, \langle(2 \xrightarrow{2} 2), (2 \rightarrow 2)\rangle, \langle(2 \xrightarrow{1} 2), (2 \rightarrow 3)\rangle\}$

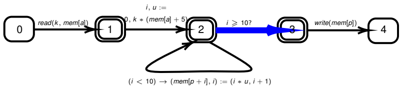

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ

Compose Example

Source:

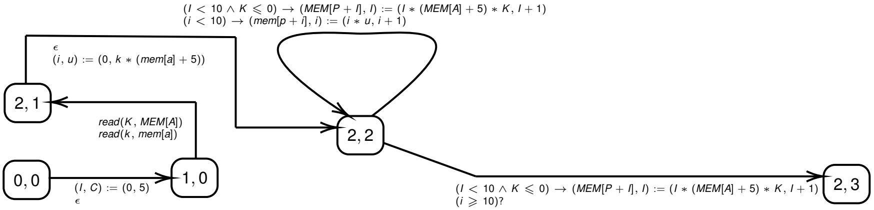

(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
0
(I, C) := (0, 5)
1
read(K, MEM[A])
2
I ⩾ 10?
3
write(MEM[P])
4
(I < 10 ∧ K > 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + C) ∗ K, I + 1)

Target:



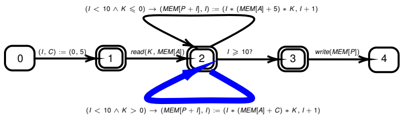
Worklist = $\{(2,2),(2,3)\}$

Matchlist = $\{\langle(2\to 3),(2\to 3)\rangle, \langle(2\to 3),(2\to 2)\rangle, \langle(2\xrightarrow{2} 2),(2\to 3)\rangle,$
$\langle(2\xrightarrow{2} 2),(2\to 2)\rangle\}$

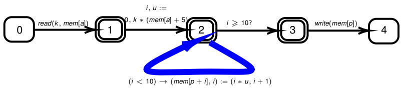

(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
(i < 10) → (mem[p + i], i) := (i ∗ u, i + 1)
ϵ
(i, u) := (0, k ∗ (mem[a] + 5))
2, 1
read(K, MEM[A])
read(k, mem[a])
2, 2
0, 0
(I, C) := (0, 5)
ϵ
1, 0
(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
(i ⩾ 10)?
2, 3

Compose Example

Source:

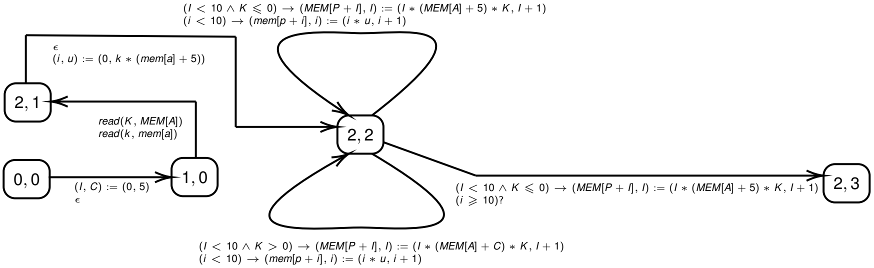

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



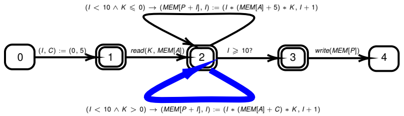
Worklist = $\{(2,2),(2,3)\}$

Matchlist = $\{\langle(2 \rightarrow 3),(2 \rightarrow 3)\rangle, \langle(2 \rightarrow 3),(2 \rightarrow 2)\rangle, \langle(2 \xrightarrow{2} 2),(2 \rightarrow 3)\rangle\}$

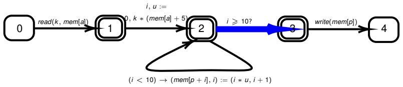

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
$(I, C) := (0, 5)$
ϵ
1, 0
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i \geqslant 10)?$
2, 3
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Compose Example

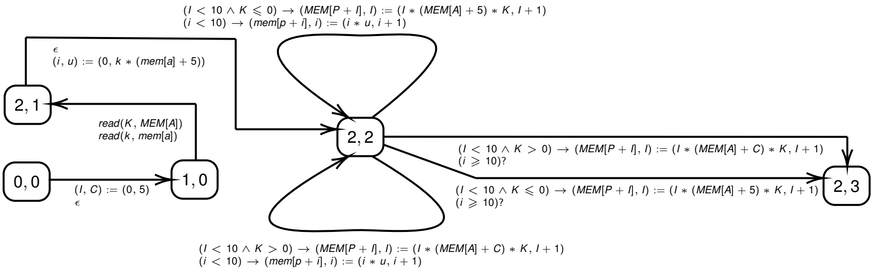
Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

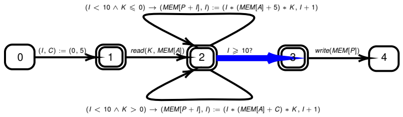


Worklist = $\{(2,2), (2,3)\}$
Matchlist = $\{\langle(2 \to 3), (2 \to 3)\rangle, \langle(2 \to 3), (2 \to 2)\rangle\}$

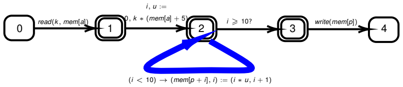

$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i \geqslant 10)?$
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i \geqslant 10)?$
2, 3
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

Compose Example

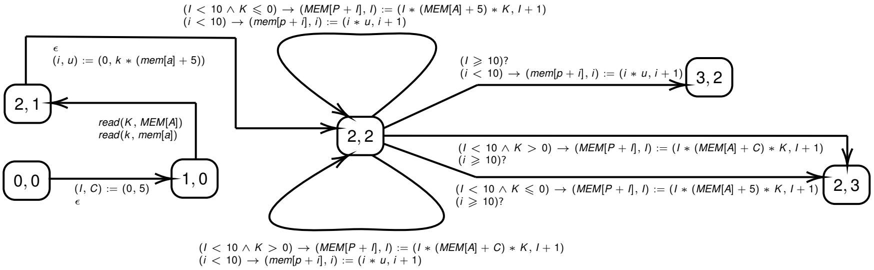
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:

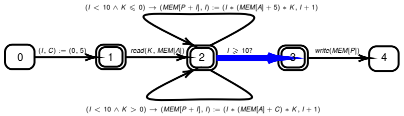


Worklist = $\{(2, 2), (2, 3), (3, 2)\}$
Matchlist = $\{\langle(2 \rightarrow 3), (2 \rightarrow 3)\rangle\}$

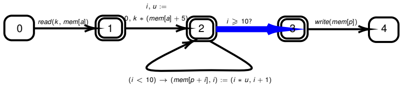

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$
$(i < 10) \rightarrow (mem[p + i], i) := (i * u, i + 1)$
ϵ
$(i, u) := (0, k * (mem[a] + 5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
0, 0
$(I, C) := (0, 5)$
ϵ
1, 0
2, 2
$(I \geqslant 10)?$
$(i < 10) \rightarrow (mem[p + i], i) := (i * u, i + 1)$
3, 2
$(I < 10 \wedge K > 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$
$(i \geqslant 10)?$
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$
$(i \geqslant 10)?$
2, 3
$(I < 10 \wedge K > 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$
$(i < 10) \rightarrow (mem[p + i], i) := (i * u, i + 1)$

Compose Example

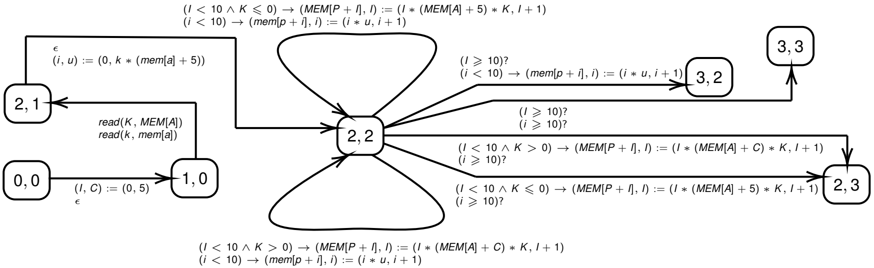
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:

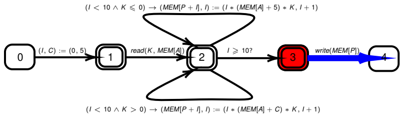

0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(2,2), (2,3), (3,2), (3,3)\}$
Matchlist = $\{\}$


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
3, 3
$(I \geqslant 10)?$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
3, 2
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
$(I \geqslant 10)?$
$(i \geqslant 10)?$
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i \geqslant 10)?$
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i \geqslant 10)?$
2, 3
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

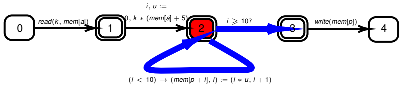
Compose Example

Source:



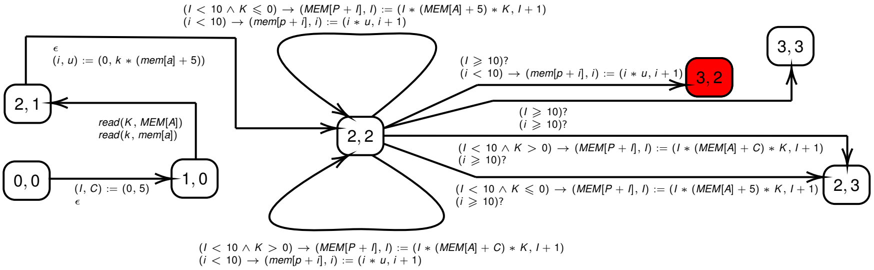
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(2,2),(2,3),(3,3)\}$
Matchlist = $\{ERROR\}$



$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
0, 0
$(I, C) := (0, 5)$
ϵ
1, 0
2, 2
$(I \geqslant 10)?$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
3, 2
3, 3
$(I \geqslant 10)?$
$(i \geqslant 10)?$
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i \geqslant 10)?$
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i \geqslant 10)?$
2, 3
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Branch Alignment

We use the technique of branch alignment to solve the issue of spurious edges being added to the composed graph.

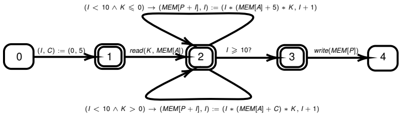
- Assume that we have an algorithm *InvGen*(f_k) that, given a partially constructed graph f_k, obtained after the kth iteration of compose, outputs a set of invariants $\{\varphi_n^k | n \in$ locations of $f_k\}$.
- We match a pair of edges $(e^{\mathcal{S}}, e^{\mathcal{T}}) \in \mathcal{E}_n^{\mathcal{S}} \times \mathcal{E}_m^{\mathcal{T}}$, where $e^{\mathcal{S}}$ and $e^{\mathcal{T}}$ are guarded by $c^{\mathcal{S}}$ and $c^{\mathcal{T}}$ respectively iff $\phi_{n,m} \wedge c^{\mathcal{S}} \wedge c^{\mathcal{T}}$ is satisfiable.
- If φ_n^{fix} is the invariant generated on the fully constructed graph on location n, for any k, φ_n^k is an underapproximation of φ_n^{fix}. ie, $\varphi_n^k \rightarrow \varphi_n^{fix}$.

Theorem 2 The following properties are satisfied by compose:

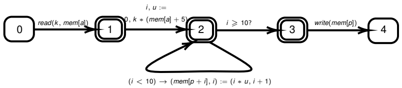
- **Termination:** The algorithm compose terminates.
- **Soundness:** If compose succeeds then the resulting graph $f = f^{\mathcal{S}} \boxtimes f^{\mathcal{T}}$ is a correct comparison graph.
- **Completeness:** Given a strong enough *InvGen*, compose succeeds in construction of a comparison graph.

Compose Example

Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Invariants

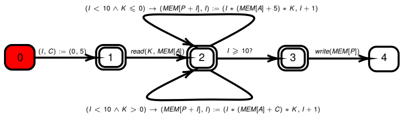
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$

Worklist = $\{(0,0)\}$
Matchlist = $\{\}$

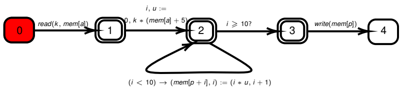
$\boxed{0,0}$

Compose Example

Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



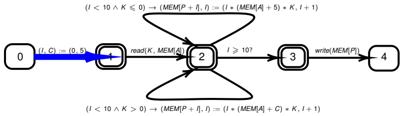
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$

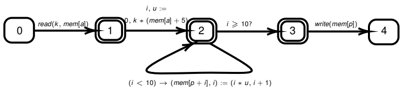
Worklist = $\{\}$
Matchlist = $\{\langle(0 \rightarrow 1), \epsilon\rangle\}$

Compose Example

Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

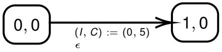
Target:



Invariants

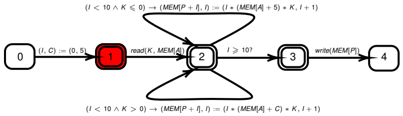
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$

Worklist = $\{(1, 0)\}$
Matchlist = $\{\}$

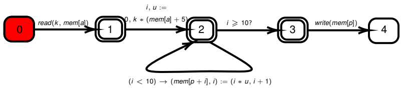


Compose Example

Source:


0
(I, C) := (0, 5)
1
read(K, MEM[A])
2
(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
(I < 10 ∧ K > 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + C) ∗ K, I + 1)
I ⩾ 10?
3
write(MEM[P])
4

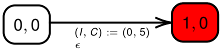
Target:



Invariants

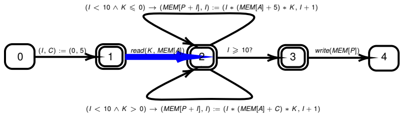
- $\varphi_{0,0} = (MEM = mem) \land (A = a) \land (P = p) \land (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \land (I = 0) \land (C = 5)$

Worklist = $\{\}$
Matchlist = $\{\langle(1 \to 2), (0 \to 1)\rangle\}$



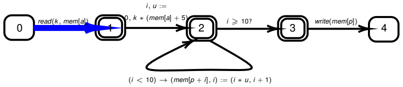
Compose Example

Source:



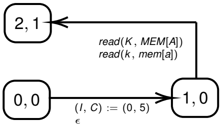
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:



0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a] + 5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i + 1)$

Worklist = $\{(2, 1)\}$
Matchlist = $\{\}$

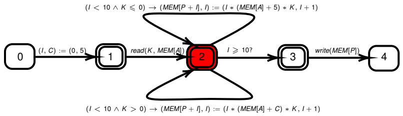


Invariants

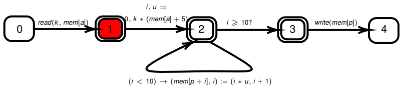
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$

Compose Example

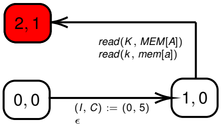
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}
Matchlist = $\{\langle \epsilon, (1 \rightarrow 2) \rangle\}$

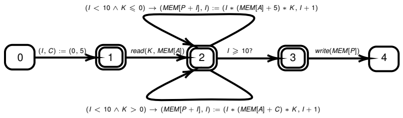


Invariants

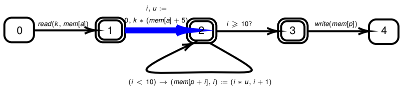
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$

Compose Example

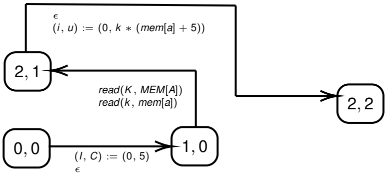
Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = $\{(2,2)\}$
Matchlist = $\{\}$

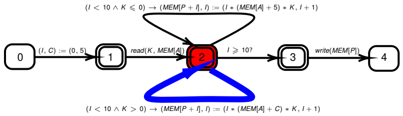


Invariants

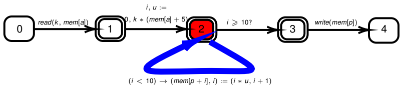
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \dots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

Compose Example

Source:

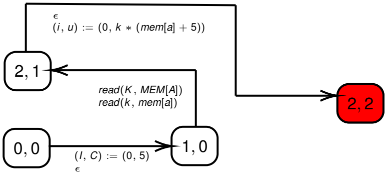

$(I < 10 \land K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \land K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}

Matchlist = $\{\langle (2 \xrightarrow{2} 2), (2 \to 2) \rangle\}$



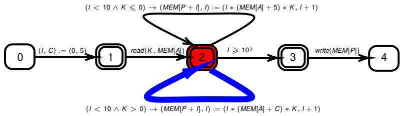
Invariants

- $\varphi_{0,0} = (MEM = mem) \land (A = a) \land (P = p) \land (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \land (I = 0) \land (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \land (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \land (i = 0) \land (u = k * mem[a] + 5)$

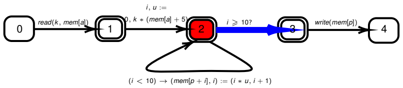
$\varphi_{2,2} \land (I < 10 \land K > 0) \land (i < 10)$ is SAT

Compose Example

Source:

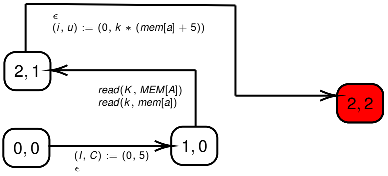

$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}

Matchlist = $\{\langle(2 \xrightarrow{2} 2), (2 \rightarrow 2)\rangle\}$



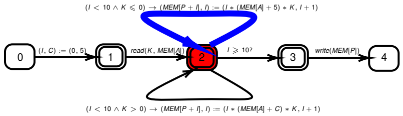
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

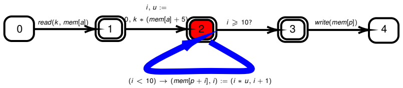
$\varphi_{2,2} \wedge (I < 10 \wedge K > 0) \wedge (i \geq 10)$ is UNSAT

Compose Example

Source:

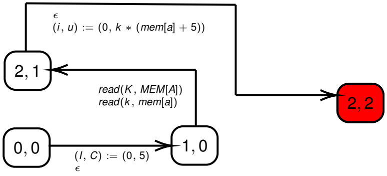

0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = $\{\}$

Matchlist = $\{\langle(2 \xrightarrow{2} 2), (2 \to 2)\rangle, \langle(2 \xrightarrow{1} 2), (2 \to 2)\rangle\}$



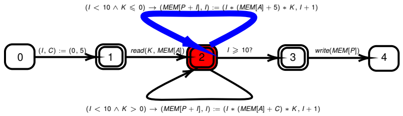
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

$\varphi_{2,2} \wedge (I < 10 \wedge K \leq 0) \wedge (i < 10)$ is SAT

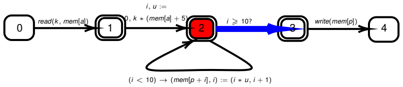
Compose Example

Source:



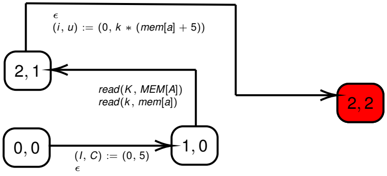
$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:



Worklist = $\{\}$

Matchlist = $\{\langle(2 \xrightarrow{2} 2), (2 \rightarrow 2)\rangle, \langle(2 \xrightarrow{1} 2), (2 \rightarrow 2)\rangle\}$



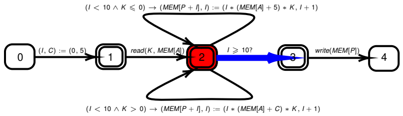
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

$\varphi_{2,2} \wedge (I < 10 \wedge K \leq 0) \wedge (i \geq 10)$ is UNSAT

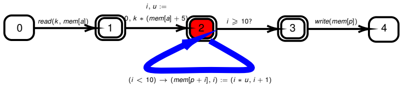
Compose Example

Source:



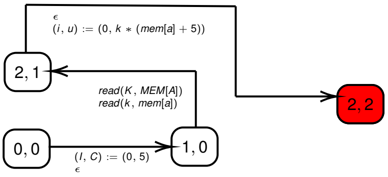
(I < 10 ∧ K ⩽ 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + 5) ∗ K, I + 1)
0
(I, C) := (0, 5)
1
read(K, MEM[A])
2
I ⩾ 10?
3
write(MEM[P])
4
(I < 10 ∧ K > 0) → (MEM[P + I], I) := (I ∗ (MEM[A] + C) ∗ K, I + 1)

Target:



Worklist = {}

Matchlist = $\{\langle(2 \xrightarrow{2} 2), (2 \rightarrow 2)\rangle, \langle(2 \xrightarrow{1} 2), (2 \rightarrow 2)\rangle\}$



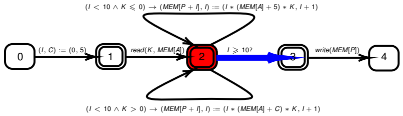
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

$\varphi_{2,2} \wedge (I \geq 10) \wedge (i < 10)$ is UNSAT

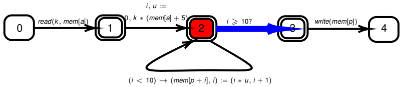
Compose Example

Source:



$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

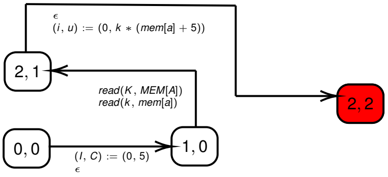
Target:



0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

Worklist = {}

Matchlist = $\{\langle (2 \xrightarrow{2} 2), (2 \to 2)\rangle,$
$\langle (2 \xrightarrow{1} 2), (2 \to 2)\rangle\}$



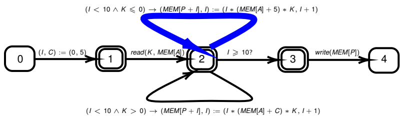
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{2,1} \wedge (i = 0) \wedge (u = k * mem[a] + 5)$

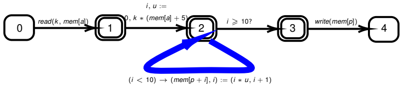
$\varphi_{2,2} \wedge (I \geq 10) \wedge (i \geq 10)$ is UNSAT

Compose Example

Source:

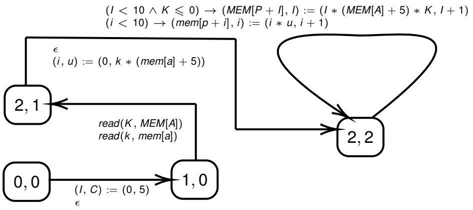

$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = $\{(2,2)\}$

Matchlist = $\{\langle (2 \xrightarrow{2} 2), (2 \to 2) \rangle\}$

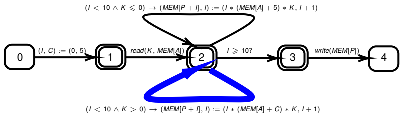

$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ

Invariants

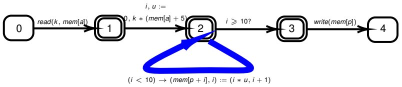
- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

Compose Example

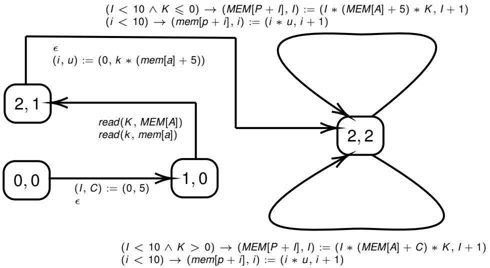
Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:


0
$read(k, mem[a])$
1
$i, u := 0, k * (mem[a]+5)$
2
$i \geqslant 10?$
3
$write(mem[p])$
4
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

Worklist = $\{(2,2)\}$
Matchlist = $\{\}$

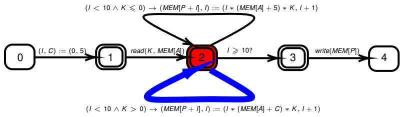

$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

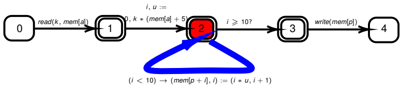
Compose Example

Source:

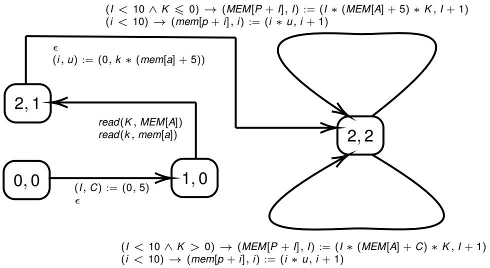


$(I < 10 \wedge K \leqslant 0) \to (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:



Worklist = {}
Matchlist = {}



$(I < 10 \wedge K \leqslant 0) \to (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$
$(i < 10) \to (mem[p + i], i) := (i * u, i + 1)$
ϵ
$(i, u) := (0, k * (mem[a] + 5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \to (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$
$(i < 10) \to (mem[p + i], i) := (i * u, i + 1)$

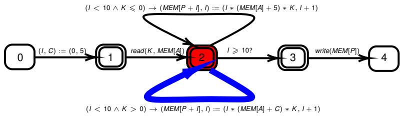
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \dots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

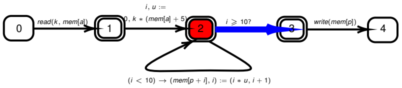
This edge is already done

Compose Example

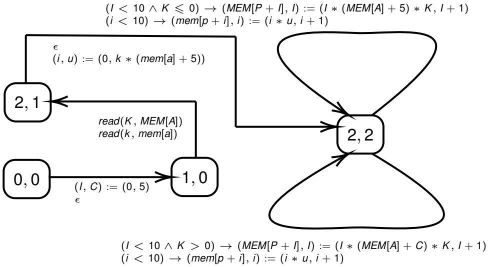
Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}
Matchlist = {}


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i+1)$

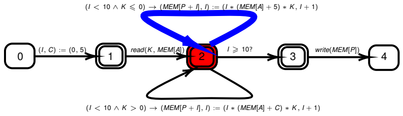
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

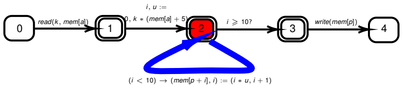
$\varphi_{2,2} \wedge (I < 10 \wedge K > 0) \wedge (i \geq 10)$ is UNSAT

Compose Example

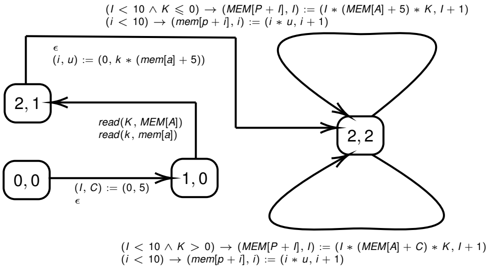
Source:


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A] + 5) * K, I + 1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A] + C) * K, I + 1)$

Target:



Worklist = {}
Matchlist = {}


$(I < 10 \wedge K \leqslant 0) \to (MEM[P+I], I) := (I * (MEM[A] + 5) * K, I + 1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i + 1)$
ϵ
$(i, u) := (0, k * (mem[a] + 5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \to (MEM[P+I], I) := (I * (MEM[A] + C) * K, I + 1)$
$(i < 10) \to (mem[p+i], i) := (i * u, i + 1)$

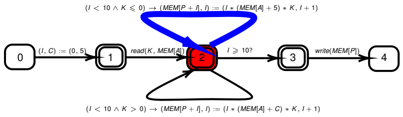
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P + 9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

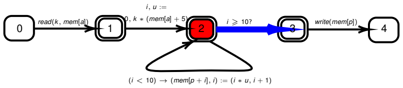
This edge is already done

Compose Example

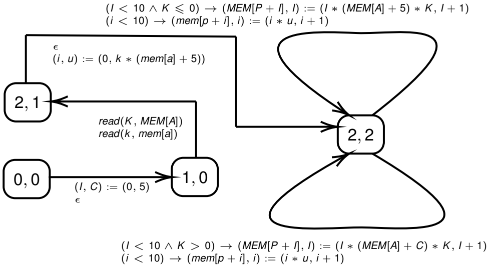
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}
Matchlist = {}


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
2, 2
$read(K, MEM[A])$
$read(k, mem[a])$
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

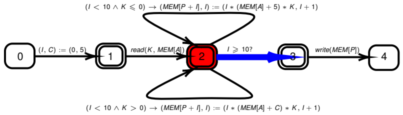
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

$\varphi_{2,2} \wedge (I < 10 \wedge K \leq 0) \wedge (i \geq 10)$ is UNSAT

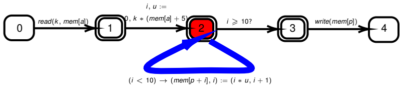
Compose Example

Source:

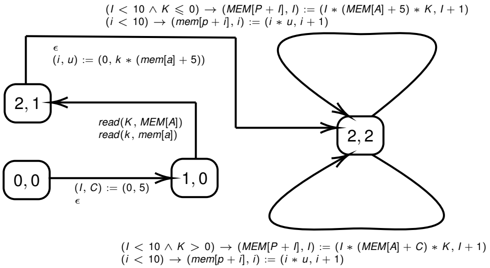


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = {}
Matchlist = {}



$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
1, 0
$(I, C) := (0, 5)$
ϵ
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

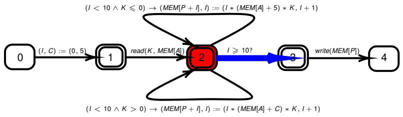
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

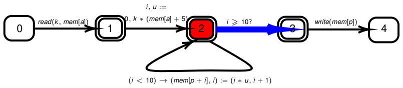
$\varphi_{2,2} \wedge (I \geq 10) \wedge (i < 10)$ is UNSAT

Compose Example

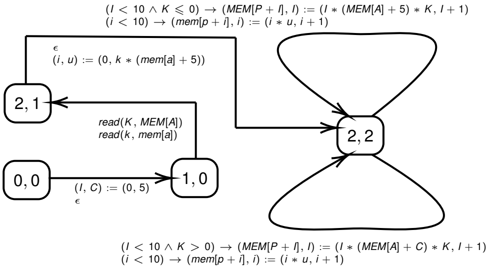
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = $\{\}$
Matchlist = $\{\langle(2 \rightarrow 3), (2 \rightarrow 3)\rangle\}$


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
0, 0
$(I, C) := (0, 5)$
ϵ
1, 0
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

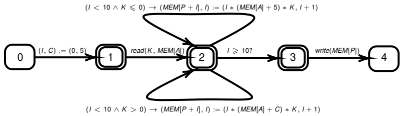
Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$

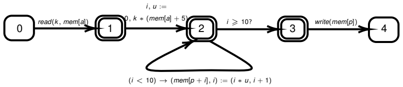
$\varphi_{2,2} \wedge (I \geq 10) \wedge (i \geq 10)$ is SAT

Compose Example

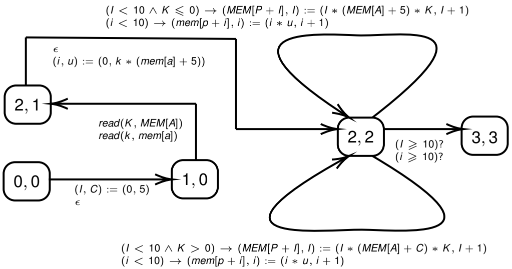
Source:


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
0
$(I, C) := (0, 5)$
1
$read(K, MEM[A])$
2
$I \geqslant 10?$
3
$write(MEM[P])$
4
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$

Target:



Worklist = $\{(3, 3)\}$
Matchlist = $\{\}$


$(I < 10 \wedge K \leqslant 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+5) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$
ϵ
$(i, u) := (0, k * (mem[a]+5))$
2, 1
$read(K, MEM[A])$
$read(k, mem[a])$
2, 2
3, 3
$(I \geqslant 10)?$
$(i \geqslant 10)?$
0, 0
$(I, C) := (0, 5)$
ϵ
1, 0
$(I < 10 \wedge K > 0) \rightarrow (MEM[P+I], I) := (I * (MEM[A]+C) * K, I+1)$
$(i < 10) \rightarrow (mem[p+i], i) := (i * u, i+1)$

Invariants

- $\varphi_{0,0} = (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P \ldots P+9])$
- $\varphi_{1,0} = \varphi_{0,0} \wedge (I = 0) \wedge (C = 5)$
- $\varphi_{2,1} = \varphi_{1,0} \wedge (K = k)$
- $\varphi_{2,2} = \varphi_{0,0} \wedge (i = I) \wedge (u = k * mem[a] + 5)$
- $\varphi_{3,3} = (MEM = Mem) \wedge (P = p)$

$$\varphi_{3,3} \implies MEM[P] = mem[p]$$

CoVaC Tool

The paper also mentions their tool which implements the theory presented. It's written in C++ with a line count of about 7000 and was tested on the compilations of LLVM. The tool was written with a two phase strategy in which they first use a weaker *InvGen* (value numbering algorithm) and then use a stronger but slower one (assertion checking via hoare logic) if that doesn't work.

The tool was tested on programs like heapsort, binary search, print first N primes etc along with the LLVM test suite. Even though LLVM optimized the code highly, with on average 0.53 optmizations per line, CoVaC spent 1 second per 41 lines. The assertion checker (which was the stronger *InvGen*) took the most time and was invoked quite often (once per 8 lines).