

CoVaC:

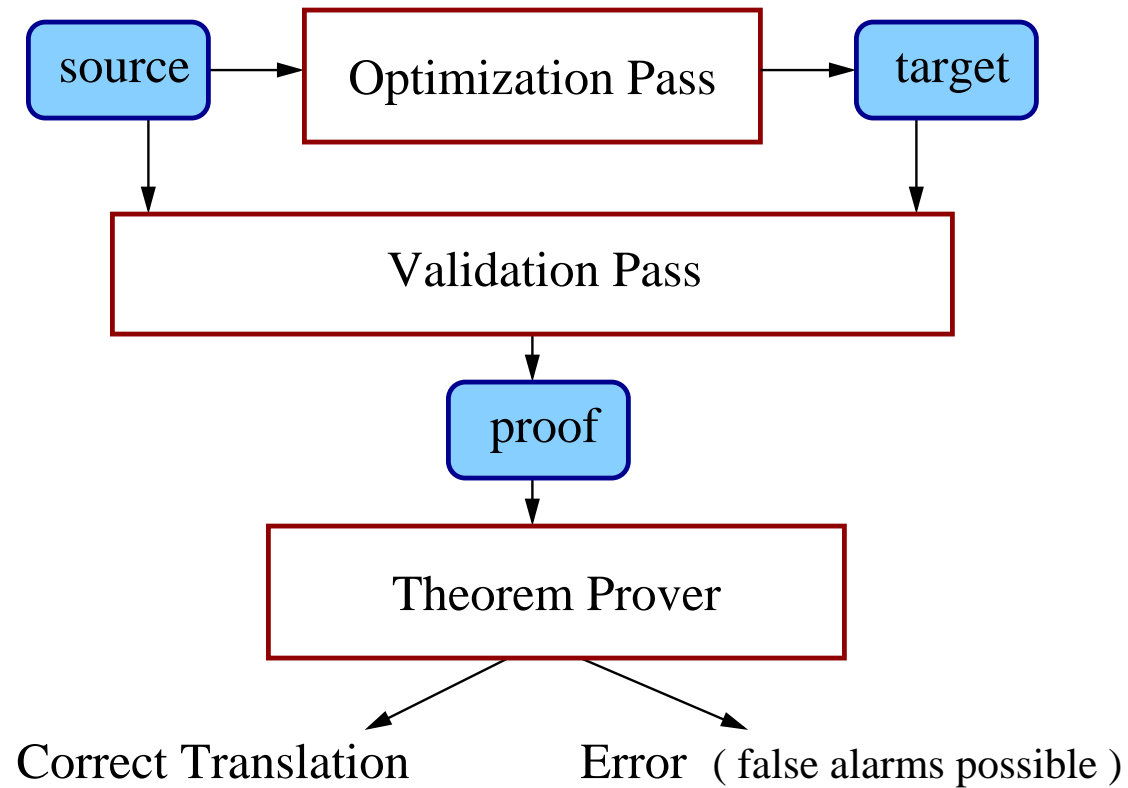
Compiler Verification by Program Analysis of the Cross-Product

Anna Zaks, Amir Pnueli

May 28, 2008

FM'08, Turku, Finland

Translation Validation



Roadmap

- CoVaC Equivalence Checking Framework:

- Construct a Comparison System (a cross-product of the source and target).
- Check if it satisfies a given specification.

- Application:

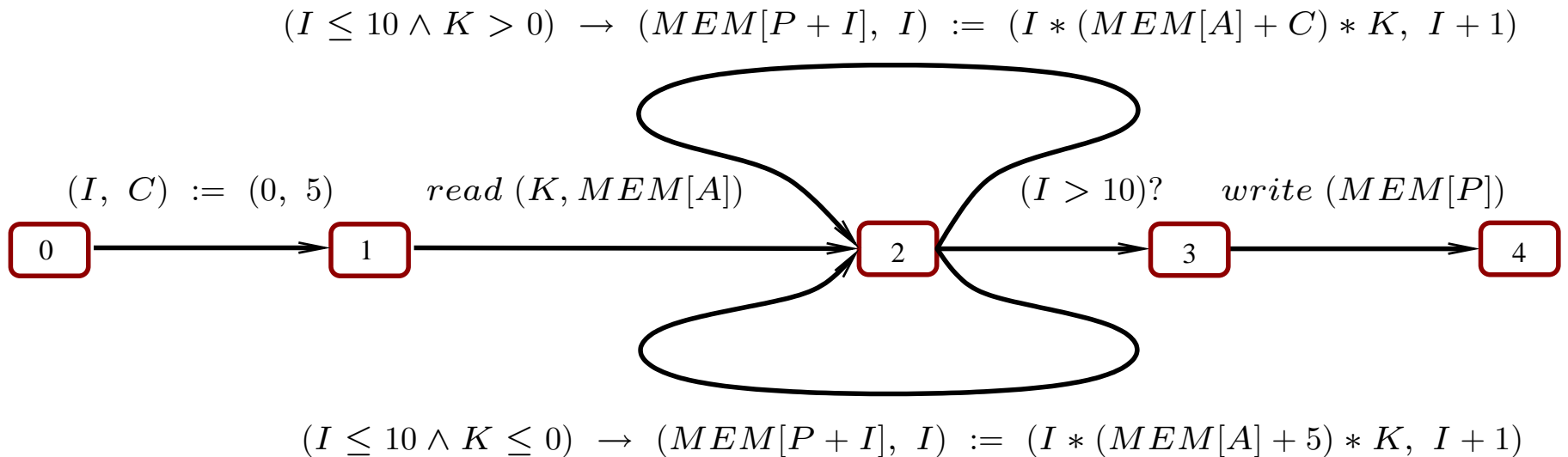
Verification of structure preserving intraprocedural optimizations while accommodating no compiler cooperation.

Example of structure preserving intraprocedural optimizations: constant folding, reassociation, common subexpression elimination, code motion, dead code elimination, branch optimizations, register allocation, instruction scheduling ...

Transition Graphs

- Each **node** corresponds to a location of the source code. The set of program locations represented in the graph must include:
 - both procedure entry and exit,
 - at least one location in each loop,
 - locations just before and right after every procedure call or I/O instruction.
- Each **edge** summarizes the effect of executing a path between the two locations.

Source



Correct Translation

- An **observation** of a program is obtained from a program computation by setting to \top the transitions and data not relevant to the I/O instructions.

$$\top \xrightarrow{\text{read}} (5, 22) \xrightarrow{\top} \top \xrightarrow{\top} \top \xrightarrow{\top} \top \xrightarrow{\top} (110) \xrightarrow{\text{write}} \top$$

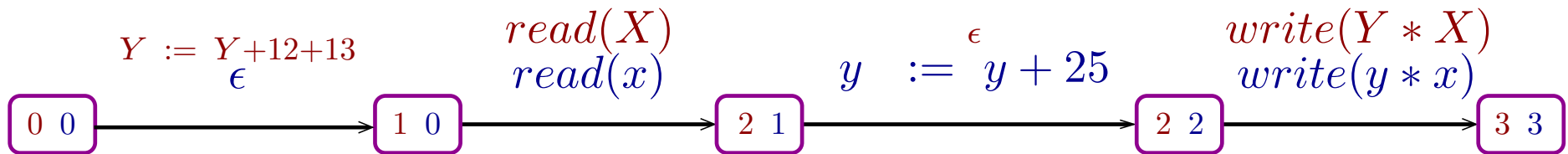
- Computations σ' and σ are **stuttering equivalent** if their observations only differ from each other by finite sequences of pairs $\top \xrightarrow{\top}$ or $\xrightarrow{\top} \top$.

$$\top \xrightarrow{\text{read}} (5, 22) \xrightarrow{\top} \top \xrightarrow{\top} (110) \xrightarrow{\text{write}} \top$$

- Program \mathcal{T} is a **correct translation** of program \mathcal{S} if, for every computation of \mathcal{T} , there exists a stuttering equivalent computation of \mathcal{S} and vice versa.

Comparison Graph

A comparison transition graph $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$ represents simultaneous execution of the source and target procedures, \mathcal{S} and \mathcal{T} .



Requirements:

- No computation of \mathcal{C} contains an infinite sequence of source (or target) ϵ -transitions; thus, every computation of the comparison graph has the corresponding computations in both source and target.
- Each source and target computation is represented in \mathcal{C} .

Witness of Correct Translation

A comparison graph \mathcal{C} is a **Witness of correct translation** if there exists a set of invariants $\{ \varphi_l \mid l \in \text{nodes of } \mathcal{C} \}$ such that for every write edge $e = (n, m)$ labeled by $(\text{write}(\vec{u}^S); \text{write}(\vec{u}^T))$:

$$\varphi_n \rightarrow (\vec{u}^S = \vec{u}^T)$$

Theorem: Target procedure \mathcal{T} is a **correct translation** of source procedure \mathcal{S} if and only if there exists a **witness** comparison graph $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$. In addition, if \mathcal{T} is a correct translation of \mathcal{S} then **every** comparison graph $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$ is a witness.

Implication: In order to check if \mathcal{T} is a correct translation of \mathcal{S} , it is sufficient to

- construct $\mathcal{C} = \mathcal{S} \boxtimes \mathcal{T}$,
- check if \mathcal{C} is a witness of correct translation.

Applications

CoVaC framework can be used in various settings.

- We may assume full knowledge of the inner workings of a particular compiler:
 - A self-certifying compiler may output a comparison graph.
- May have to accommodate minimal (or no) compiler collaboration:
 - Useful to users who may have to work with a particular existing compiler.
 - Can be of service to compiler developers to facilitate testing of immature compilers.

Comparison Graph Construction

Initialization:

- **WorkList** is initialized with the procedure entry node.

On each iteration:

- Remove a node n from the **WorkList**.
- Discover new edges outgoing from n .
- Place all the successors reachable by following the new edges into the **WorkList**.

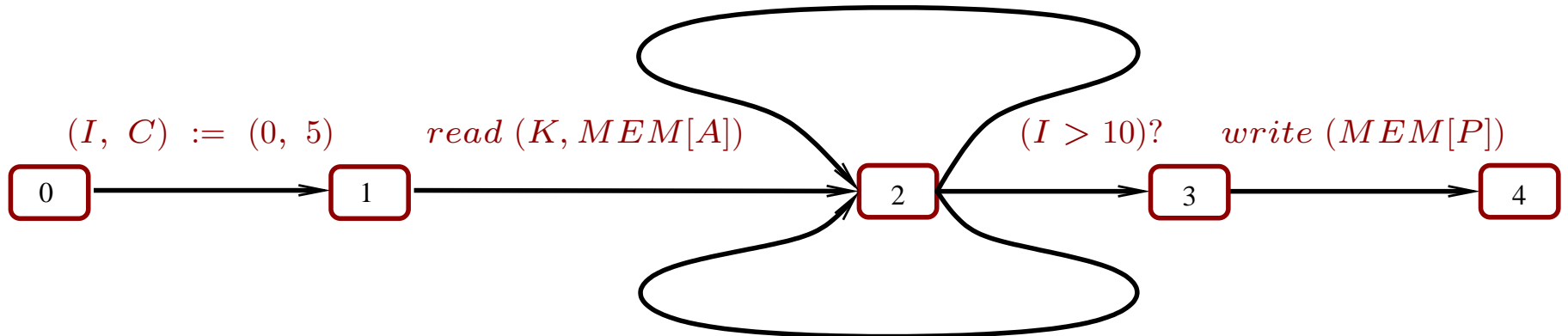
Composition of Source and Target Edges

- **Only compose edges of the same type**; assignments are composed only if either both systems are currently at a loop head or none.
- **Allow mixing an ϵ -label with an assignment**, but require that the ϵ -edge does not introduce an ϵ -cycle for any of the systems.
- **Raise an error** and abort the construction of \mathcal{C} if the above rules are not applicable.

Example of a Source and Target Transition Graphs

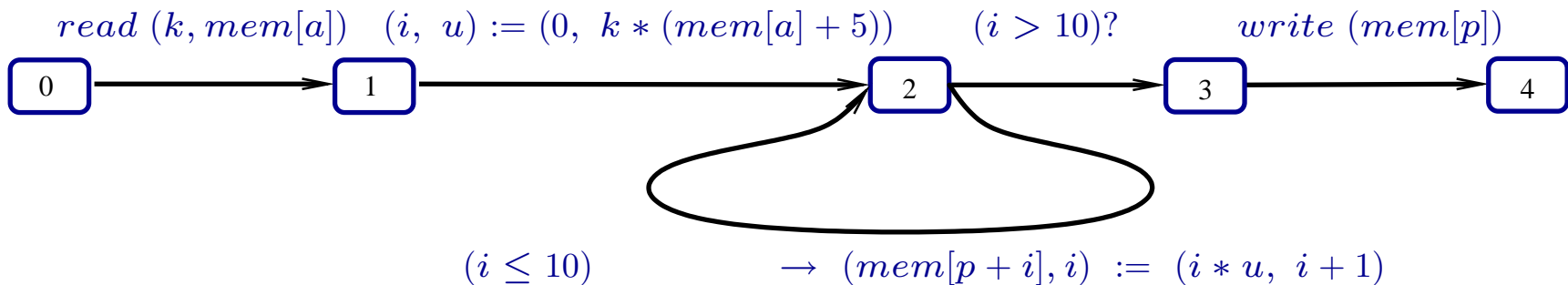
$$(I \leq 10 \wedge K > 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$$

Source



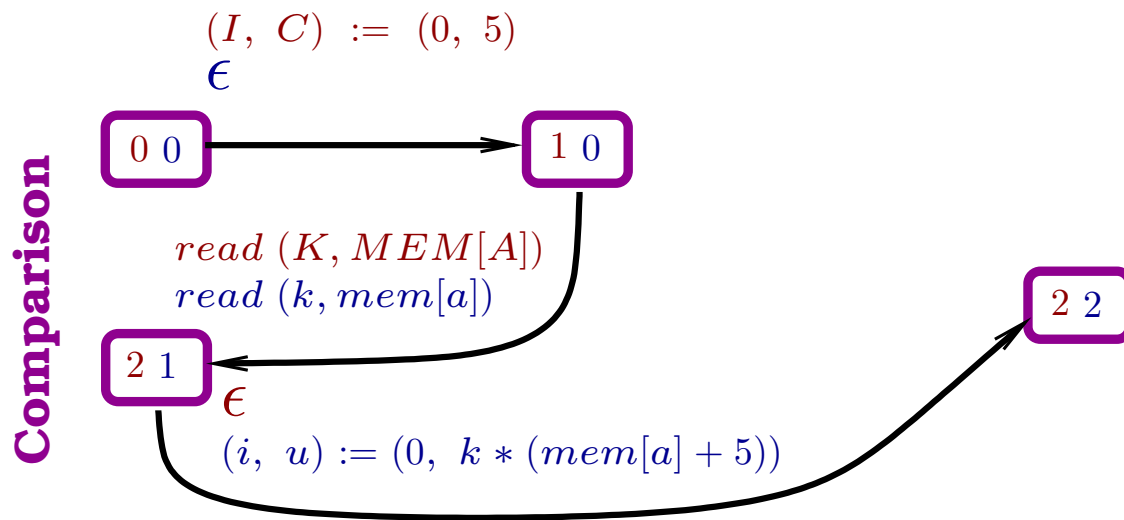
$$(I \leq 10 \wedge K \leq 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$$

Target



The target is obtained from the source after constant copy propagation, if simplification, loop invariant code motion, reassociation, and instruction scheduling.

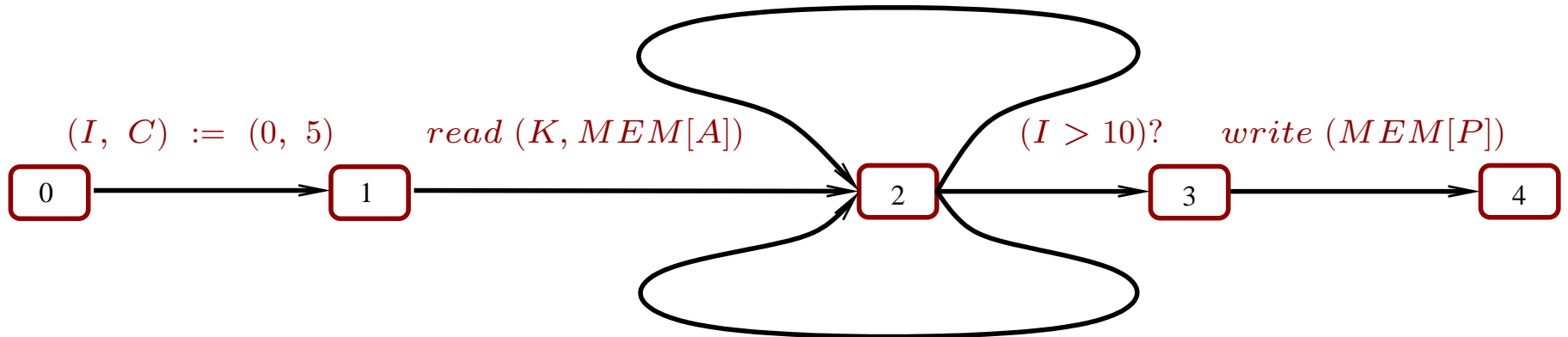
Example: after 3rd iteration



Example of a Source and Target Transition Graphs

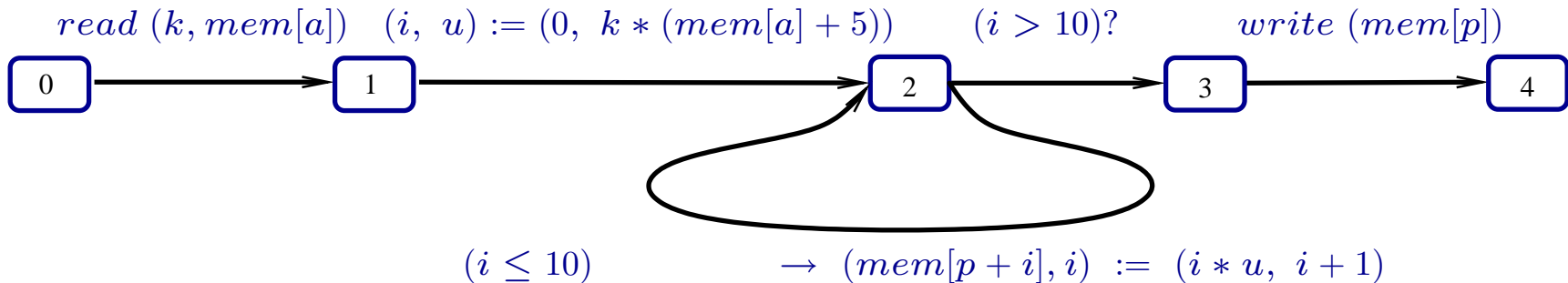
$$(I \leq 10 \wedge K > 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1)$$

Source



$$(I \leq 10 \wedge K \leq 0) \rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1)$$

Target

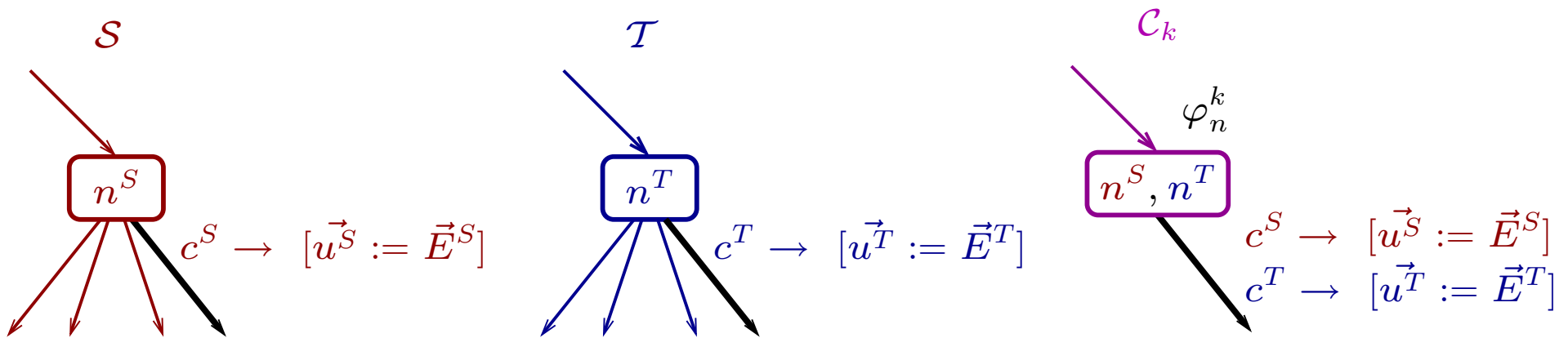


The target is obtained from the source after constant copy propagation, if simplification, loop invariant code motion, reassociation, and instruction scheduling.

Conditional Branch Alignment

We use invariants over the partially constructed graph $\{ \varphi_n^k \mid n \in \text{locations of } \mathcal{C}_k \}$ to facilitate the conditional branch alignment at iteration $k+1$. For each source edge $e^S \in \mathcal{E}_n^S$, conditioned on c^S , and target edge $e^T \in \mathcal{E}_n^T$ conditioned on c^T , we match a pair $(e^S, e^T) \in \mathcal{E}_n^S \times \mathcal{E}_n^T$ iff

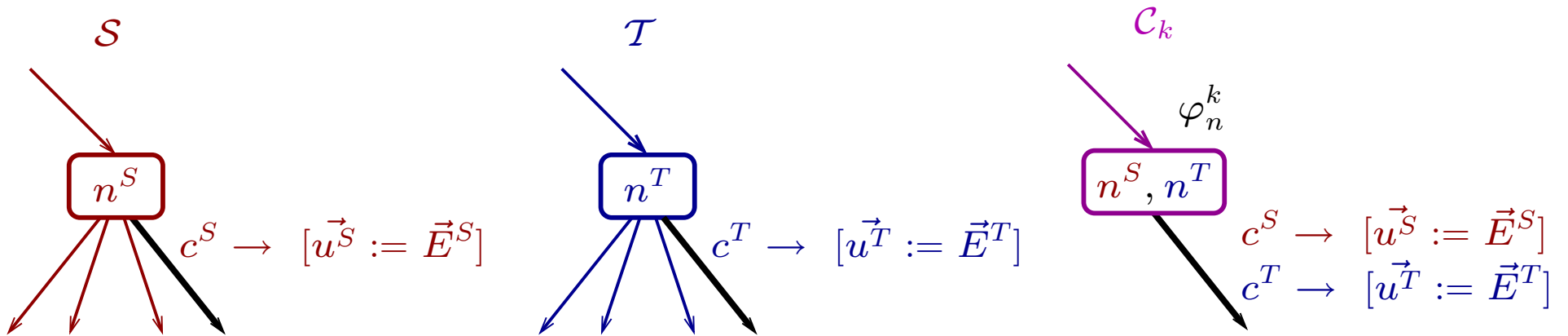
$(\varphi_n^k \wedge c^S \wedge c^T)$ is satisfiable.



Conditional Branch Alignment

We use invariants over the partially constructed graph $\{ \varphi_n^k \mid n \in \text{locations of } \mathcal{C}_k \}$ to facilitate the conditional branch alignment at iteration $k+1$. For each source edge $e^S \in \mathcal{E}_n^S$, conditioned on c^S , and target edge $e^T \in \mathcal{E}_n^T$ conditioned on c^T , we match a pair $(e^S, e^T) \in \mathcal{E}_n^S \times \mathcal{E}_n^T$ iff

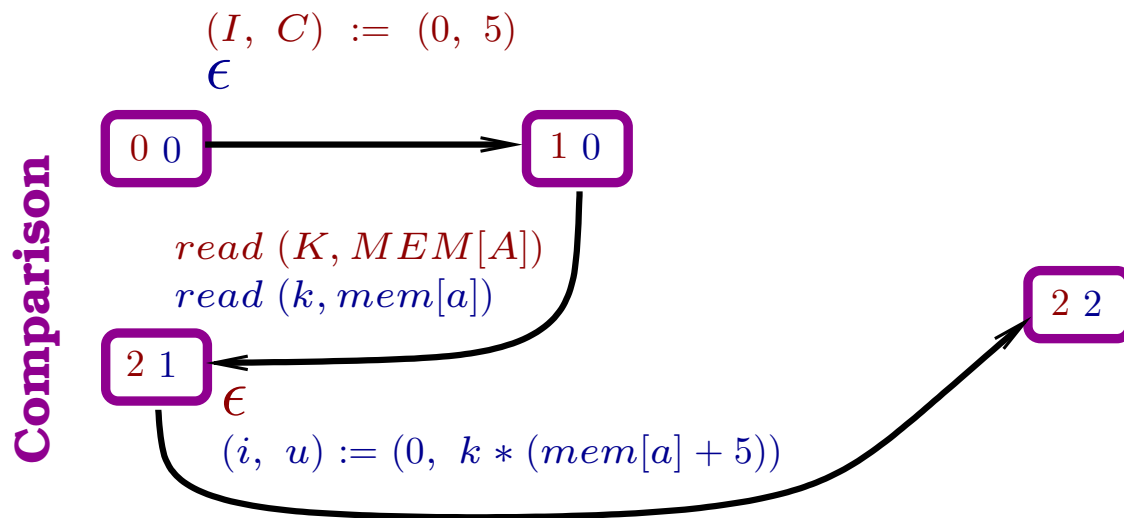
$(\varphi_n^k \wedge c^S \wedge c^T)$ is satisfiable.



Let φ_n^{fix} be an invariant of the complete comparison graph. Then $\varphi_n^k = (\varphi_n^{fix} \wedge \Phi_n^k)$ is an under-approximation of φ_n^{fix} . **No spurious predictions are possible:** if the match (e^S, e^T) is made with φ_n^k , it also complies with φ_n^{fix} .

Example: 4th iteration

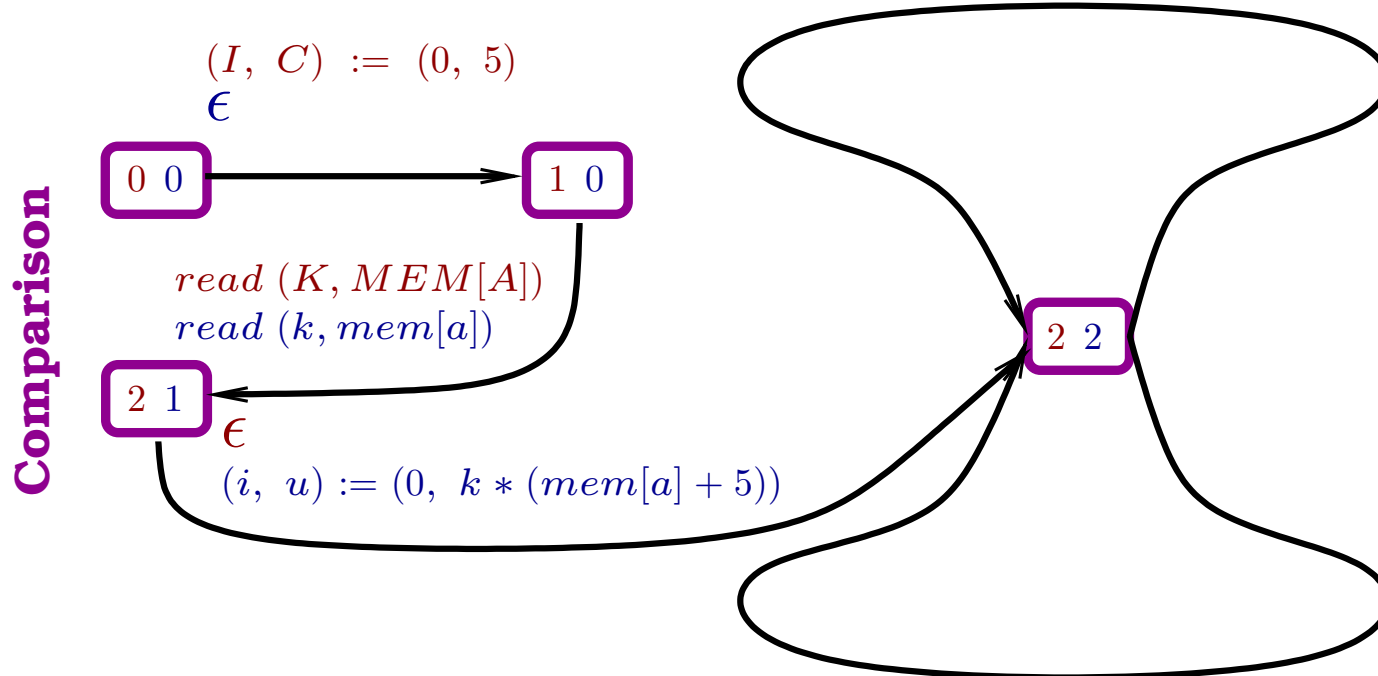
$$\varphi_{\langle 2,2 \rangle}^3 = (I = i = 0)$$



Example: 4th iteration

$$\varphi_{\langle 2,2 \rangle}^3 : (I = i = 0)$$

$$\begin{aligned} (I \leq 10 \wedge K > 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

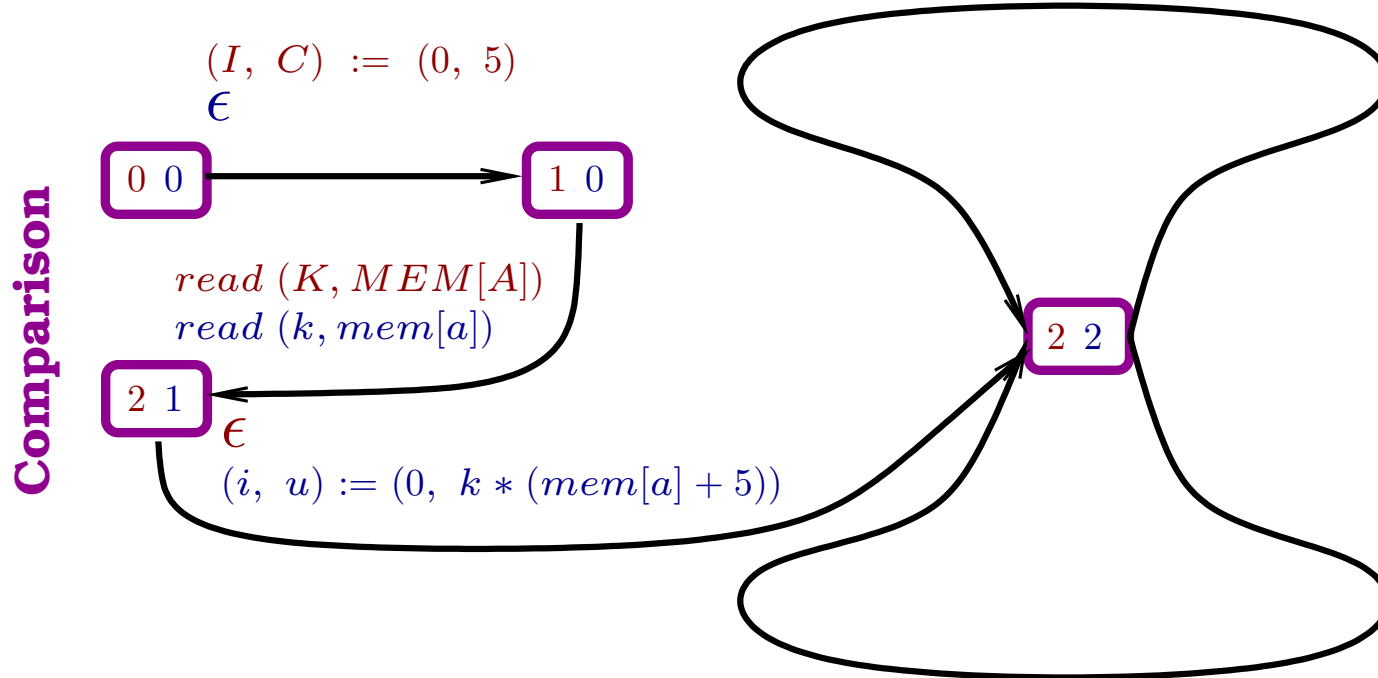


$$\begin{aligned} (I \leq 10 \wedge K \leq 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

Example: 5th iteration

$$\varphi_{\langle 2,2 \rangle}^4 : (I = i)$$

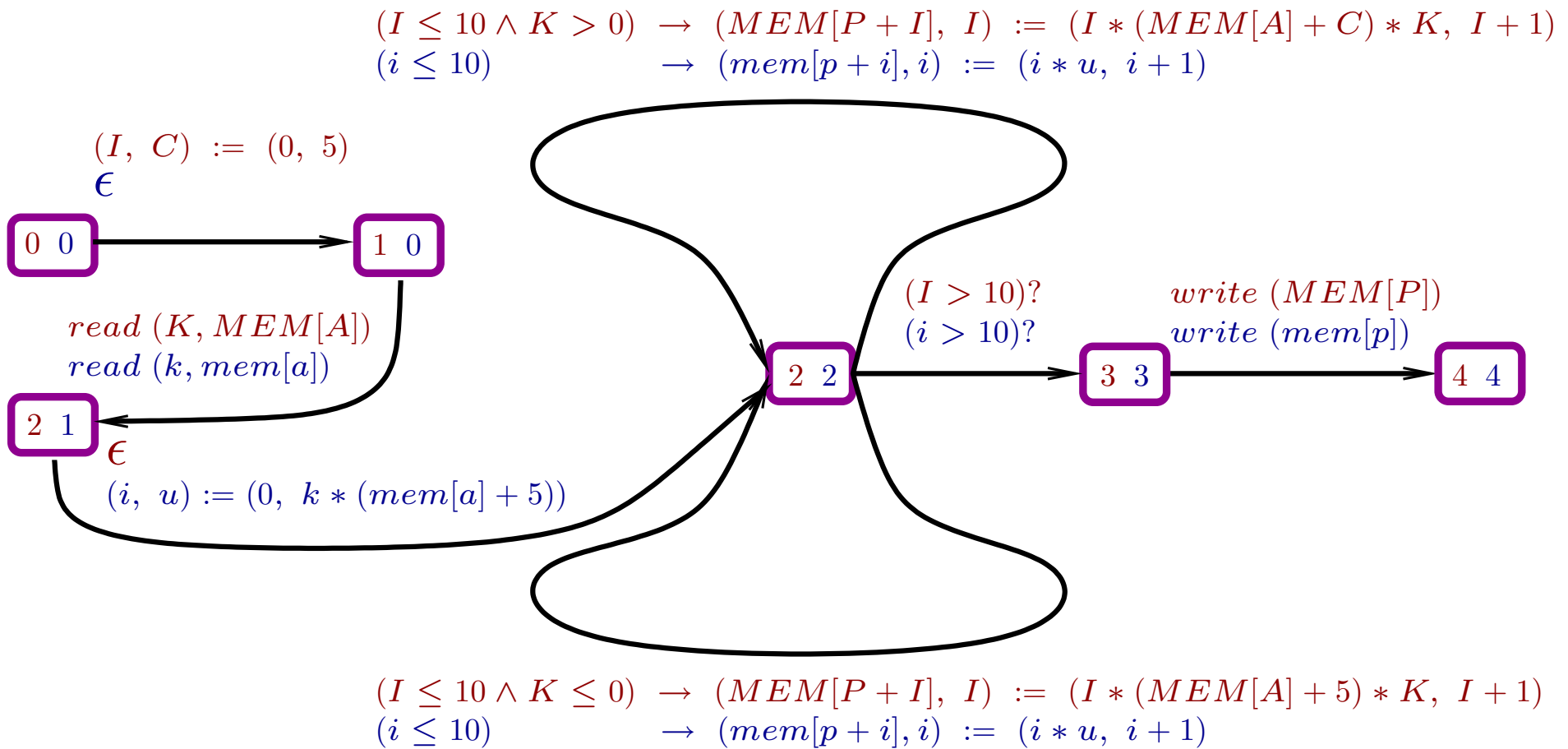
$$\begin{aligned} (I \leq 10 \wedge K > 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$



$$\begin{aligned} (I \leq 10 \wedge K \leq 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

Example: after 6th iteration, we obtain the complete graph

Comparison

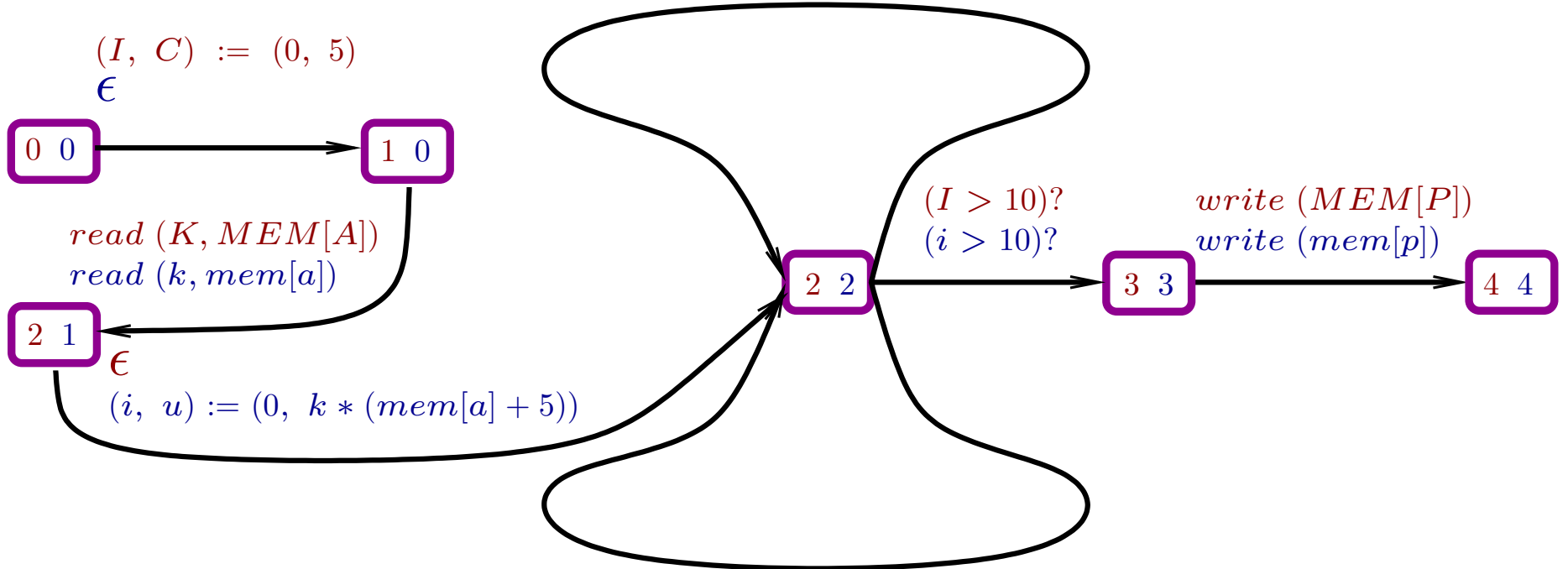


Example: Witness Verification Conditions

$$\varphi_{\langle 3,3 \rangle}^{fix} \rightarrow (MEM[P] = mem[p])$$

$$\begin{aligned} (I \leq 10 \wedge K > 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

Comparison



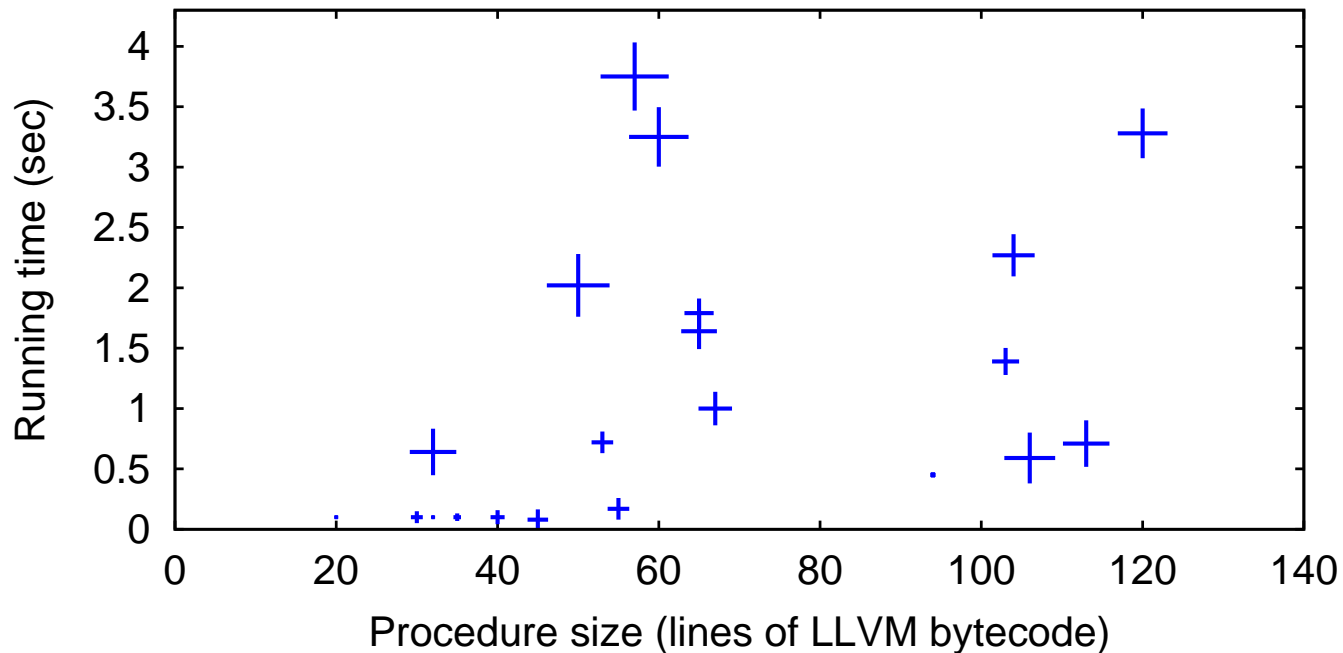
$$\begin{aligned} (I \leq 10 \wedge K \leq 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

Comparison Graph Construction: Summary

- **Soundness:** If the algorithm succeeds it constructs a comparison graph.
- **Completeness:** If the input graphs are consonant (structurally similar), the algorithm succeeds given a strong enough invariant generation algorithm.

CoVaC Prototype Tool

- We used CoVaC tool to verify optimizations performed by [LLVM compiler](#).
- [Value numbering](#) and [assertion checking](#) are used for invariant generation.
- Tested on selected LLVM feature tests and third party implementation of classical algorithms like in-place heapsort, mergesort, qsort, strcmp, shortest paths, etc.
- When validating highly optimized code, $1/2$ optimization per line, CoVaC spends 0.02 sec per line.



Future Work

- Extending of the set of supported optimizations to include, for example, **interprocedural and loop reordering optimizations**.
- Application of the CoVaC framework to development of a **self-certifying compiler**.
- Validation of **language-based security properties**, specifically, checking conformance with information flow policies by self-composition.

Related Work

Our approach can be seen as application of bisimulation equivalence to translation validation.

Existing general purpose translation validation frameworks for C-like languages, which rely on some compiler input to guide their verification effort:

- Translation validation for an optimizing compiler by Necula, 2000;
- A Methodology for the Translation Validation of Optimizing Compilers by Zuck et al., 2003;
- Symbolic transfer function-based approaches to certified compilation by Rival, 2004.

Special-purpose translator validators:

- Catching and identifying bugs in register allocation by Huang et al., 2006.
- Formal verification of translation validators: a case study on instruction scheduling optimizations by Tristan et al., 2008.

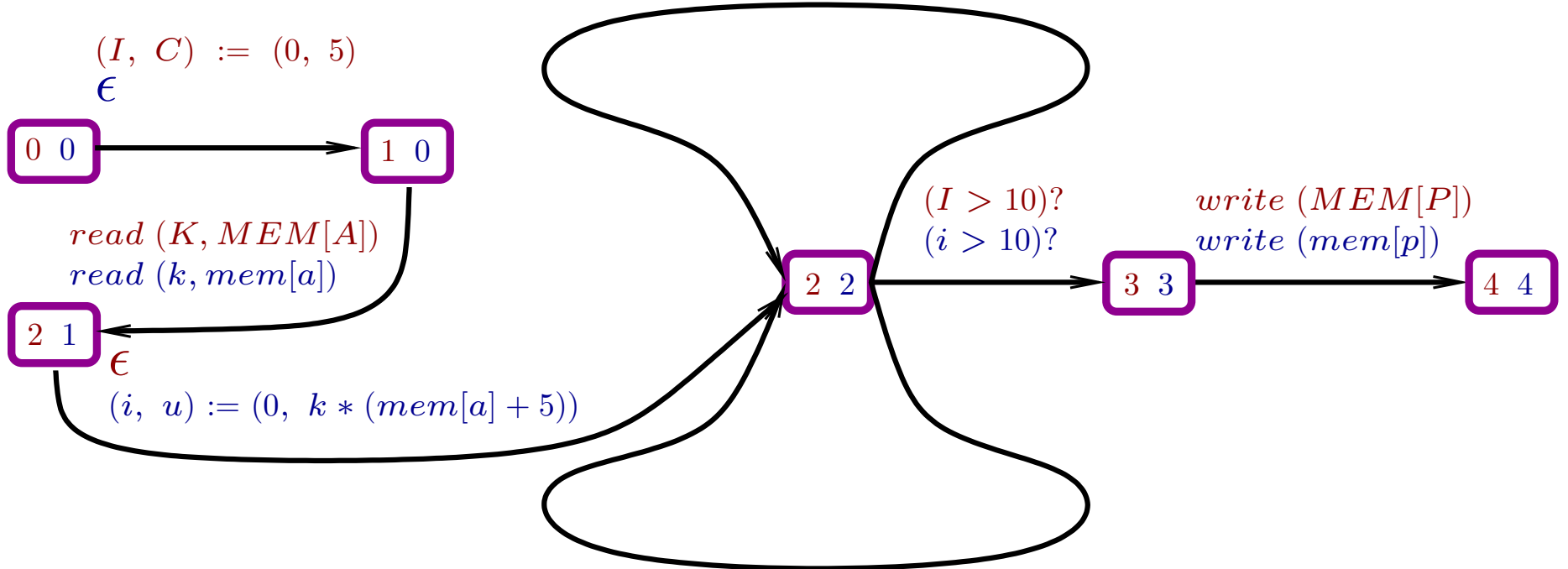
Example: Witness Verification Conditions

$$\varphi_{\langle 3,3 \rangle}^{fix} \rightarrow (MEM[P] = mem[p])$$

$$\begin{aligned} \varphi_{\langle 0,0 \rangle}^{fix} &: (MEM = mem) \wedge (A = a) \wedge (P = p) \wedge (A \notin [P..P + 10]) \\ \varphi_{\langle 1,0 \rangle}^{fix} &: (I = 0) \wedge (C = 5) \wedge \varphi_{\langle 0,0 \rangle}^{fix} \\ \varphi_{\langle 2,1 \rangle}^{fix} &: (K = k) \wedge \varphi_{\langle 1,0 \rangle}^{fix} \\ \varphi_{\langle 2,2 \rangle}^{fix} &: (I = i) \wedge (u = (MEM[A] + 5) * K) \wedge (C = 5) \wedge (K = k) \wedge \varphi_{\langle 0,0 \rangle}^{fix} \\ \varphi_{\langle 3,3 \rangle}^{fix} &: (MEM = mem) \wedge (P = p) \end{aligned}$$

$$\begin{aligned} (I \leq 10 \wedge K > 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + C) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$

Comparison



$$\begin{aligned} (I \leq 10 \wedge K \leq 0) &\rightarrow (MEM[P + I], I) := (I * (MEM[A] + 5) * K, I + 1) \\ (i \leq 10) &\rightarrow (mem[p + i], i) := (i * u, i + 1) \end{aligned}$$