Compiler Validation by Program Analysis of the Cross-Product

Anna Zaks, Amir Pnueli

New York University

2009 LLVM Developer Meeting

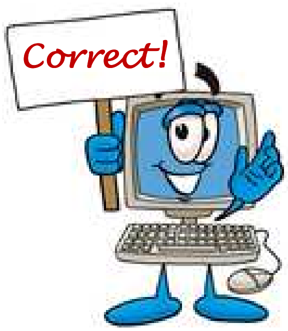
Usually …


```
bool turn, flag[2];
P0:
    flag[0] := 1;
    turn = 1;
    while (flag[1] ==1 && turn == 1)
        ; /* busy wait */
    /* critical section */
    flag[0] = 0;
    goto P0;
...
```

BUGS ARE EVERYWHERE!

- A bug in the Therac-25 radiation therapy machine was responsible for 5 patient deaths (80's)
- Northeast Blackout of 2003 was triggered by a local outage that went undetected due to a race condition in the GE's monitoring software (2003)
- Smartship USS Yorktown had to be towed into a naval base after an unhandled division by zero error caused its propulsion system to fail (1997)

Use a Tool to Check Correctness



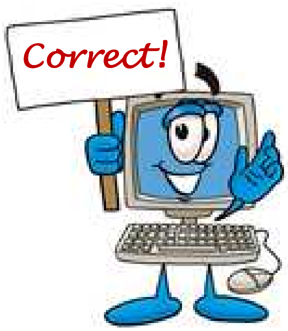

```
bool turn, flag[2];
P0:
    flag[0] := 1;
    turn = 1;
    while (flag[1] ==1 && turn == 1)
        ; /* busy wait */
    /* critical section */
    flag[0] = 0;
    goto P0;
...
```

Much better

BUT

What you check is not what you execute!!!

What you Check is not what you Execute!




```
bool turn, flag[2];
P0:
    flag[0] := 1;
    turn = 1;
    while (flag[1] ==1 && turn == 1)
        ; /* busy wait */
    /* critical section */
    flag[0] = 0;
    goto P0;
...
```

```
static int sh_f0, sh_f1, sh_last;

void * run_thread0 () {
   struct pa_desc d;
   d.f0 = &sh_f0;
   d.f1 = &sh_f1;
   d.last = 1;
   for (;;) {
      *(d.f0)=1;
      sh_last=d.last ;
      while (*(d.f1)==1 && (last==d.last)) {
         ; /* busy wait */
      }
      /* critical section */
      d.f0=0;
   }
}
...
```

```
...
entry:
   %tmp1 = getelementptr %struct.pa* %d, i32 0, i32 0
   %tmp2 = load i32** %tmp1
   store i32 1, i32* %tmp2
   %tmp4 = getelementptr %struct.pa* %d, i32 0, i32 2
   %tmp5 = load i32* %tmp4
   store i32 %tmp5, i32* @pa last
   %tmp8 = getelementptr %struct.pa* %d, i32 0, i32 1
   %tmp9 = load i32** %tmp8
   br label %bb6
bb6:
   %tmp10 = load i32* %tmp9
   %tmp11 = icmp eq i32 %tmp10, .
   br i1 %tmp11, label %cond next, label %return
...
```

ORC

Compiler Verification

Correct!

```
bool turn, flag[2];
```

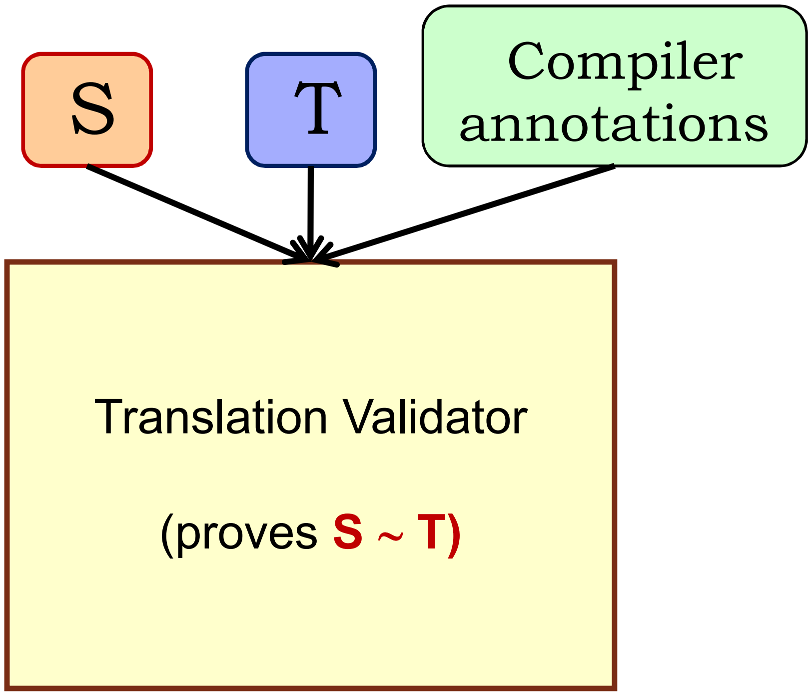
- Verify that the ***optimization*** pass preserves the semantics of the program
- Only intraprocedural, structure preserving optimizations are supported
- The verifier must be sound

```
...
entry:
  %tmp1 = getelementptr %struct.pa* %d, i32 0, i32 0
  %tmp2 = load i32** %tmp1
  store i32 1, i32* %tmp2
  %tmp4 = getelementptr %struct.pa* %d, i32 0, i32 2
  %tmp5 = load i32* %tmp4
  store i32 %tmp5, i32* @pa last
  %tmp8 = getelementptr %struct.pa* %d, i32 0, i32 1
  %tmp9 = load i32** %tmp8
  br label %bb6
bb6:
  %tmp10 = load i32* %tmp9
  %tmp11 = icmp eq i32 %tmp10, ...
  br i1 %tmp11, label %cond next, label %return
...
```

```
        *(d.f0)=1;
        sh_last=d.last ;
        while (*(d.f1)==1 && (last==d.last)) {
            ; /* busy wait */
        }
        /* critical section */
        d.f0=0;
    }
}
...
```

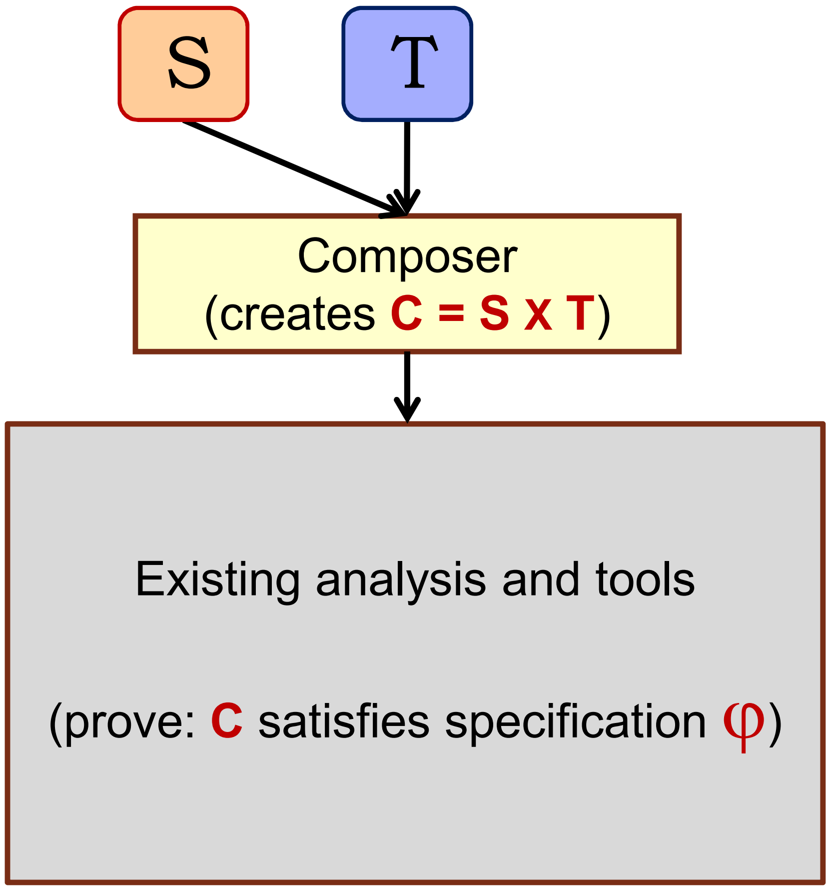
ORC

Existing Translation Validation Tools



- *Translation validation for an optimizing compiler*, Necula, 2000
- *Translation Validation of Optimizing Compilers*, NYU, 2003
- *Symbolic transfer function-based approaches to certified compilation*, Rival, 2004

CoVaC: Compiler Validation via Program Analysis of the Cross Product



- Precision of the analysis and the validator correctness ↑
- Effort ↓

CoVaC: When the Semantics are Preserved?

S $\sim$ T

```
in(X);
in(Y);
S = 0;
for ( i=1; i<Y; i++) { S += X; }
out(S);
```

```
in(x);
in(y);
;

out(x*y);
```

T ~ S if for every observation of S, there exists a stuttering equivalent observation of T and vise versa:

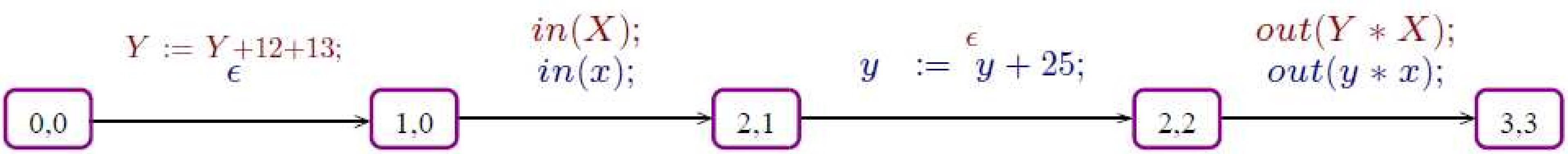
$$\top \xrightarrow{\text{in}} (5,22) \xrightarrow{\top} \top \xrightarrow{\top} \top \xrightarrow{\top} \top \xrightarrow{\top} (110) \xrightarrow{\text{out}} \top \ldots$$

$$\top \xrightarrow{\text{in}} (5,22) \xrightarrow{\top} \top \xrightarrow{\top} (110) \xrightarrow{\text{out}} \top \ldots$$

Comparison Graph

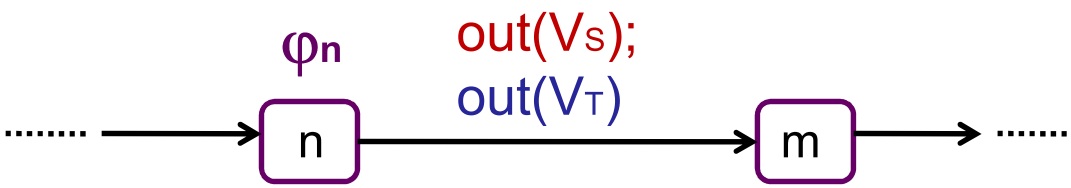
A comparison graph **C = S X T** represents simultaneous execution of the source and target procedures, **S** and **T:**

- each computation of C corresponds to computations in S and T
- each computation of S or T is represented in C



A Witness

C = **S** **X** **T** is a *witness* of correct translation if for every output edge, there exists a program invariant implying the equivalence of the source and target output variables: $\varphi_n \rightarrow V_S = V_T$.



Theorem: To check that **T** ~ **S**, it is sufficient to

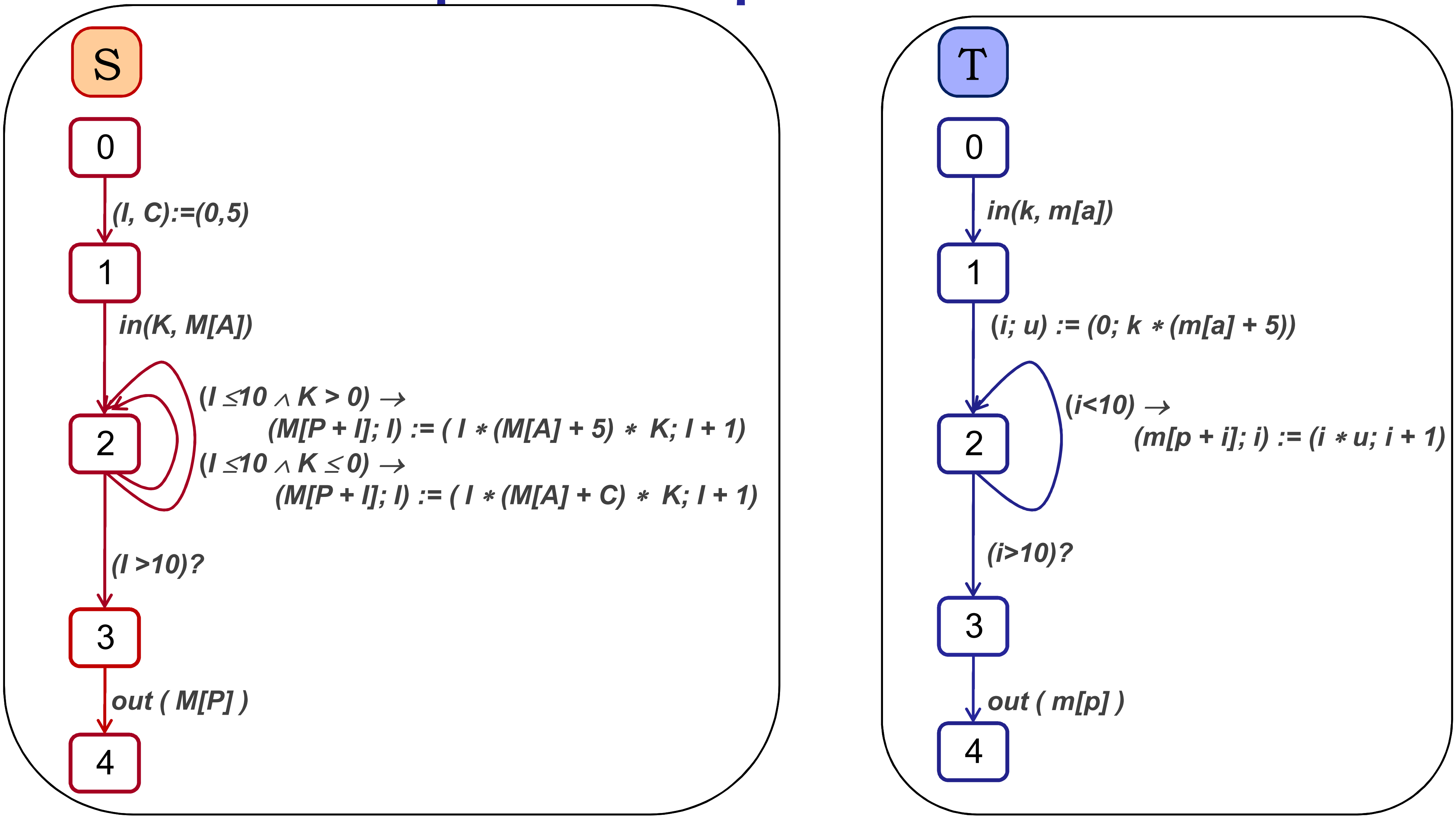
1. construct a comparison graph **C** = **S** **X** **T**
2. check if **C** is a witness

Comparison Graph Construction

Intended uses of the CoVaC framework:

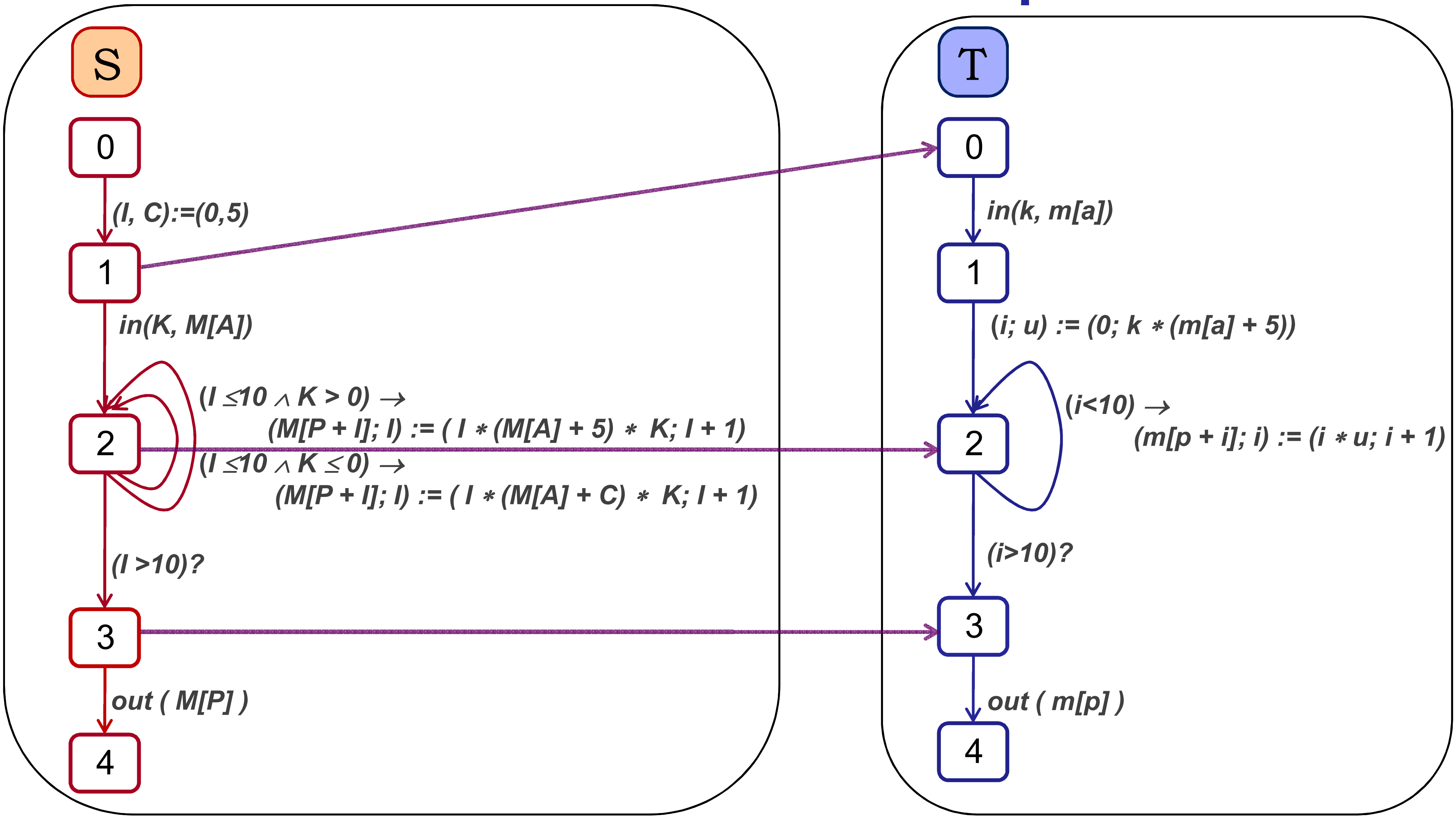
- construction of self-certifying compilers
- high assurance compilation when debug info is available
- testing of immature compilers (no debug info)

Comparison Graph Construction



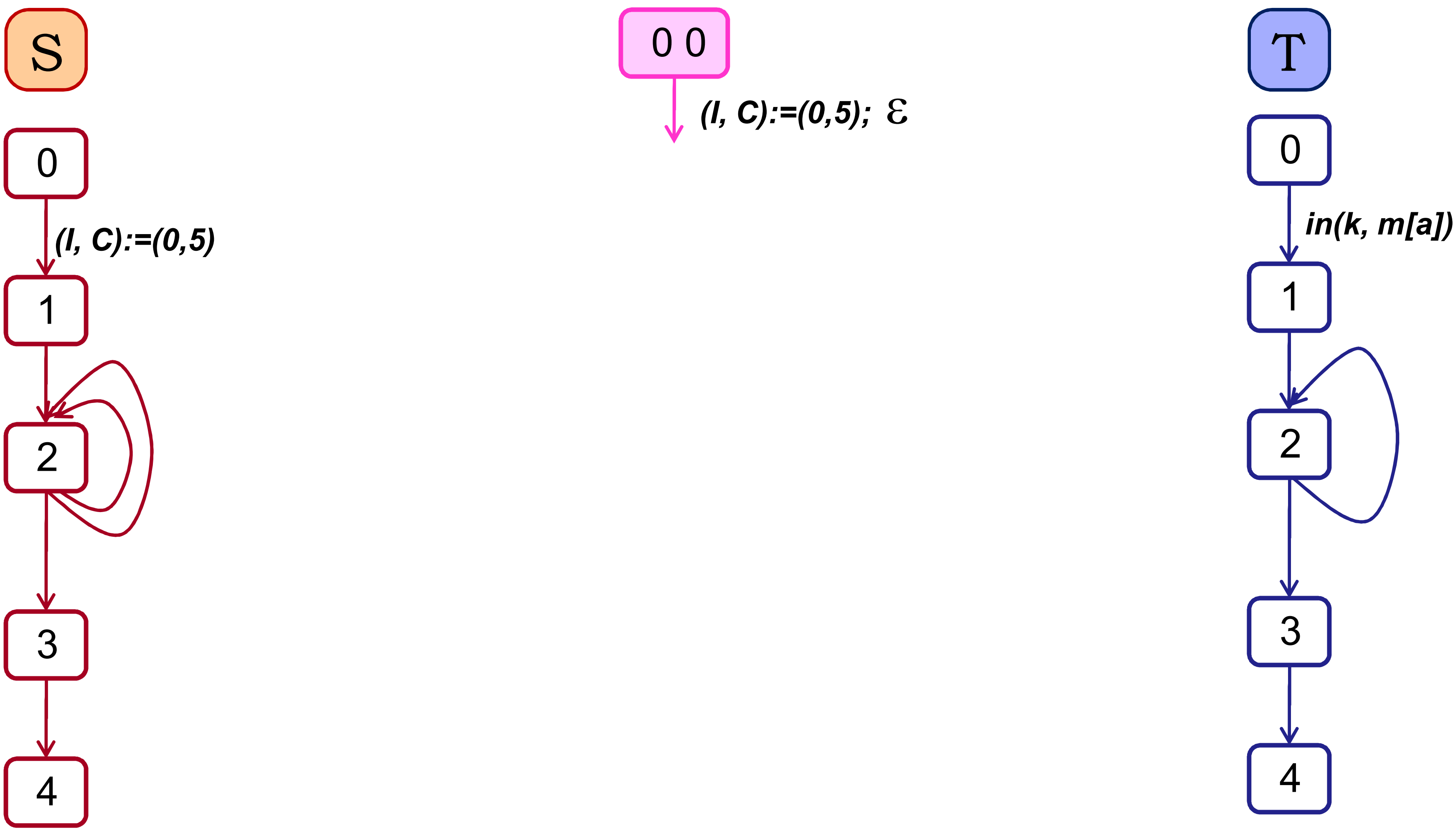
Optimizations: constant copy propagation, if simplification, loop invariant code motion, and instruction scheduling.

Consonant Transition Graphs

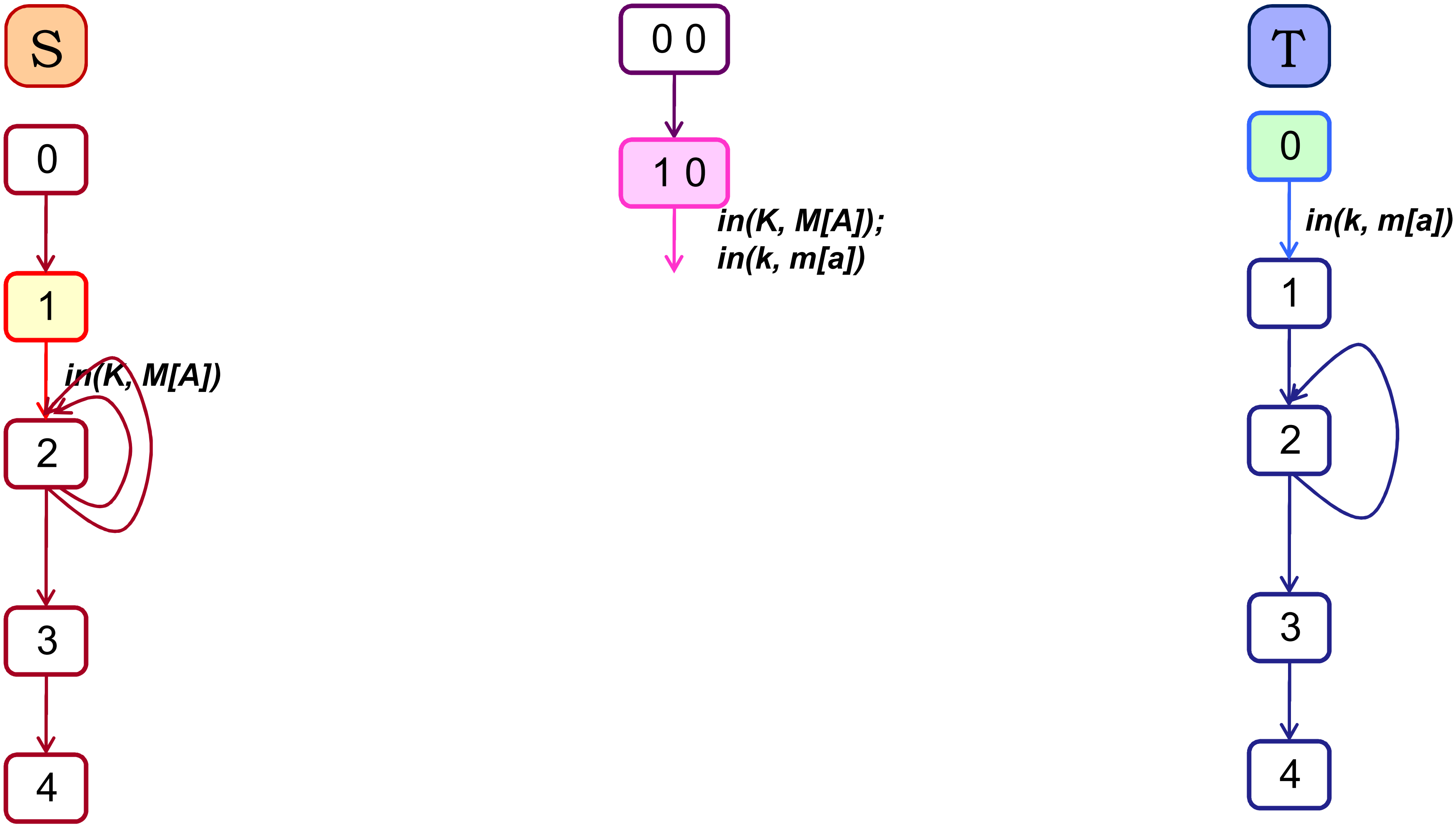


Optimizations: constant copy propagation, if simplification, loop invariant code motion, and instruction scheduling.

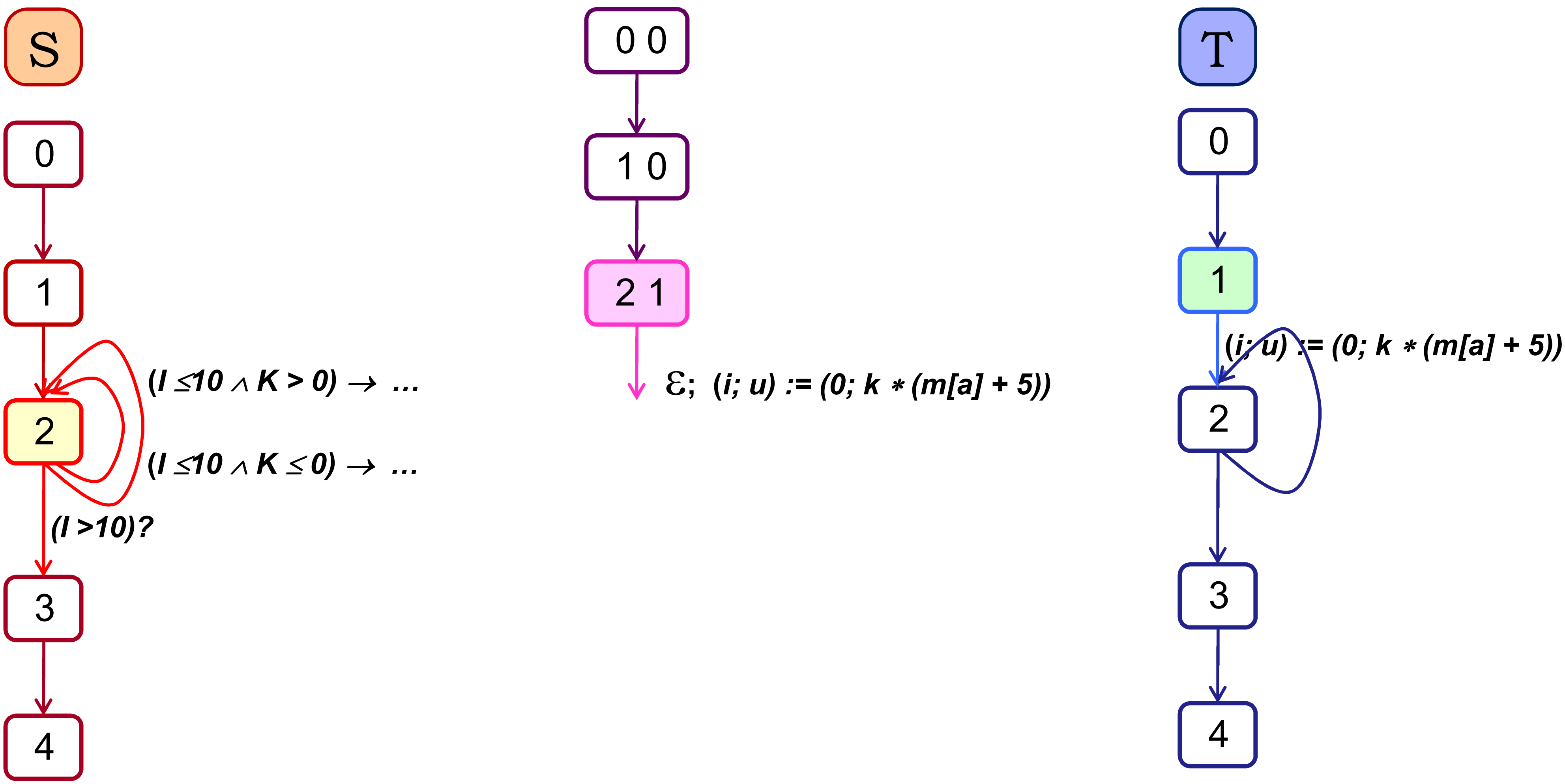
Comparison Graph Construction



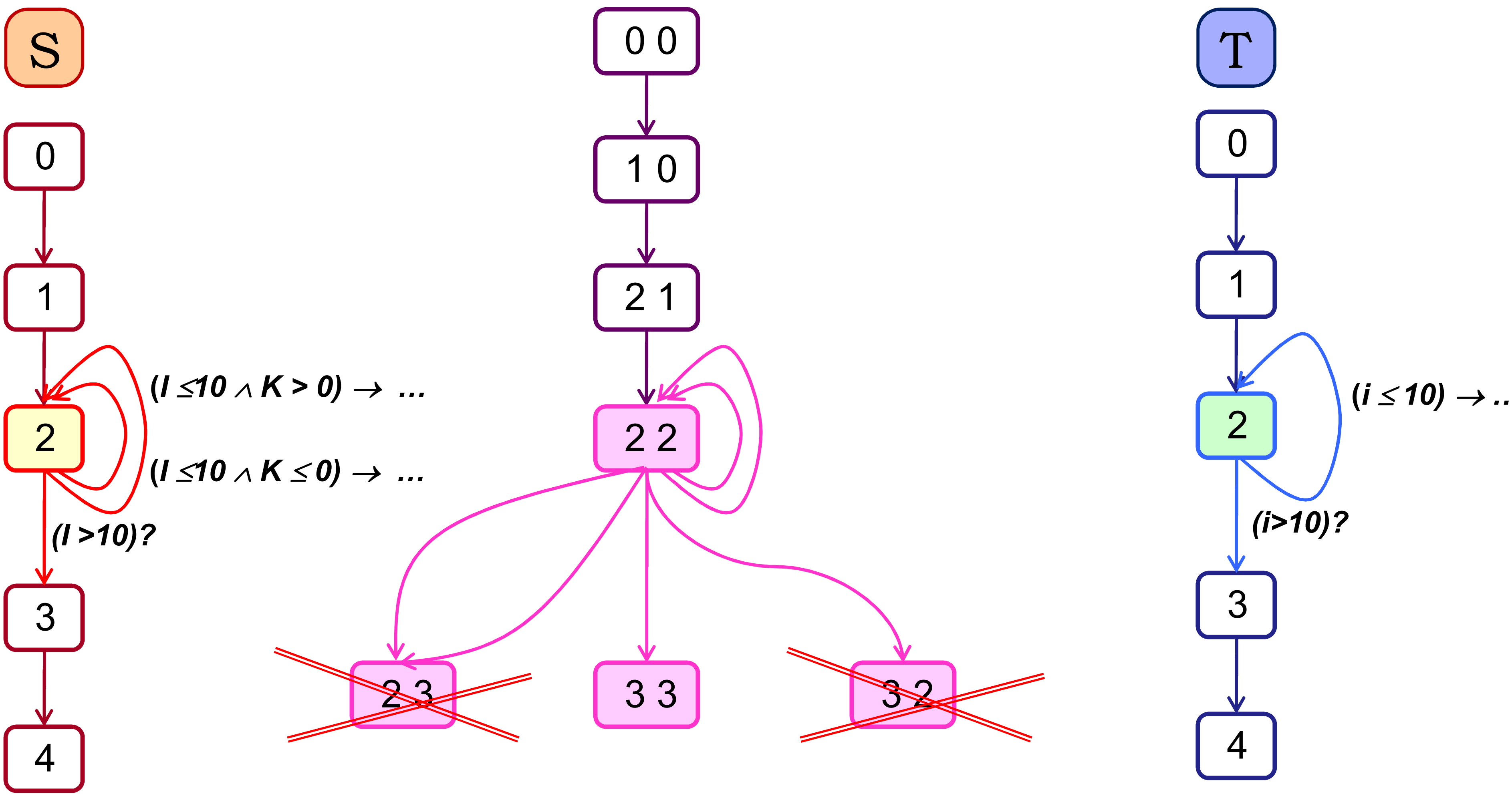
Comparison Graph Construction



Comparison Graph Construction

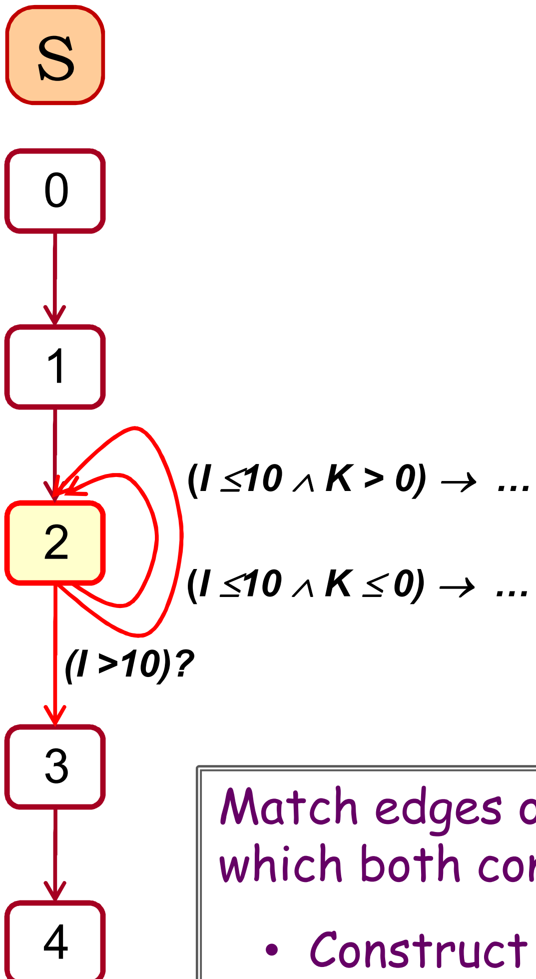


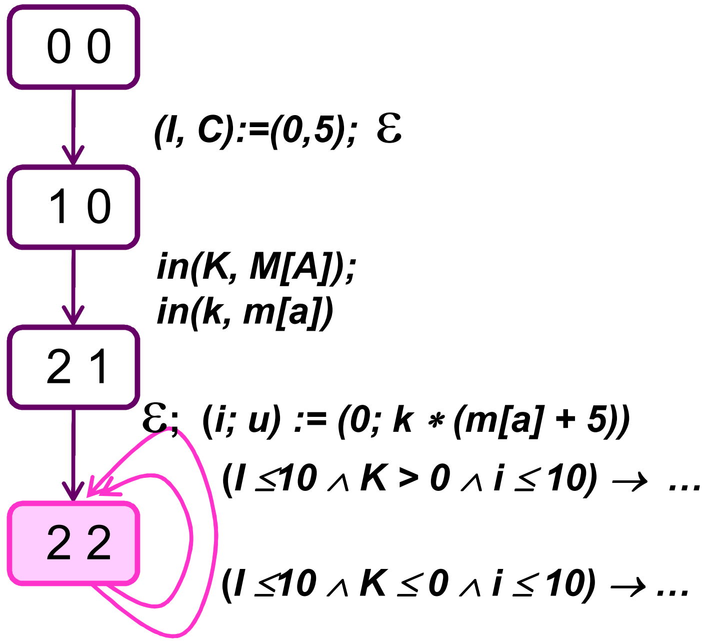
Comparison Graph Construction

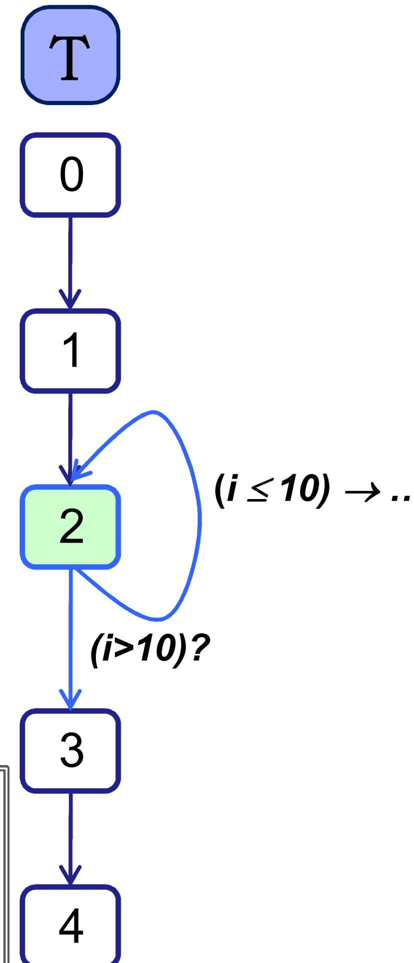


Infeasible paths – inefficient and may lead to false alarms

Comparison Graph Construction



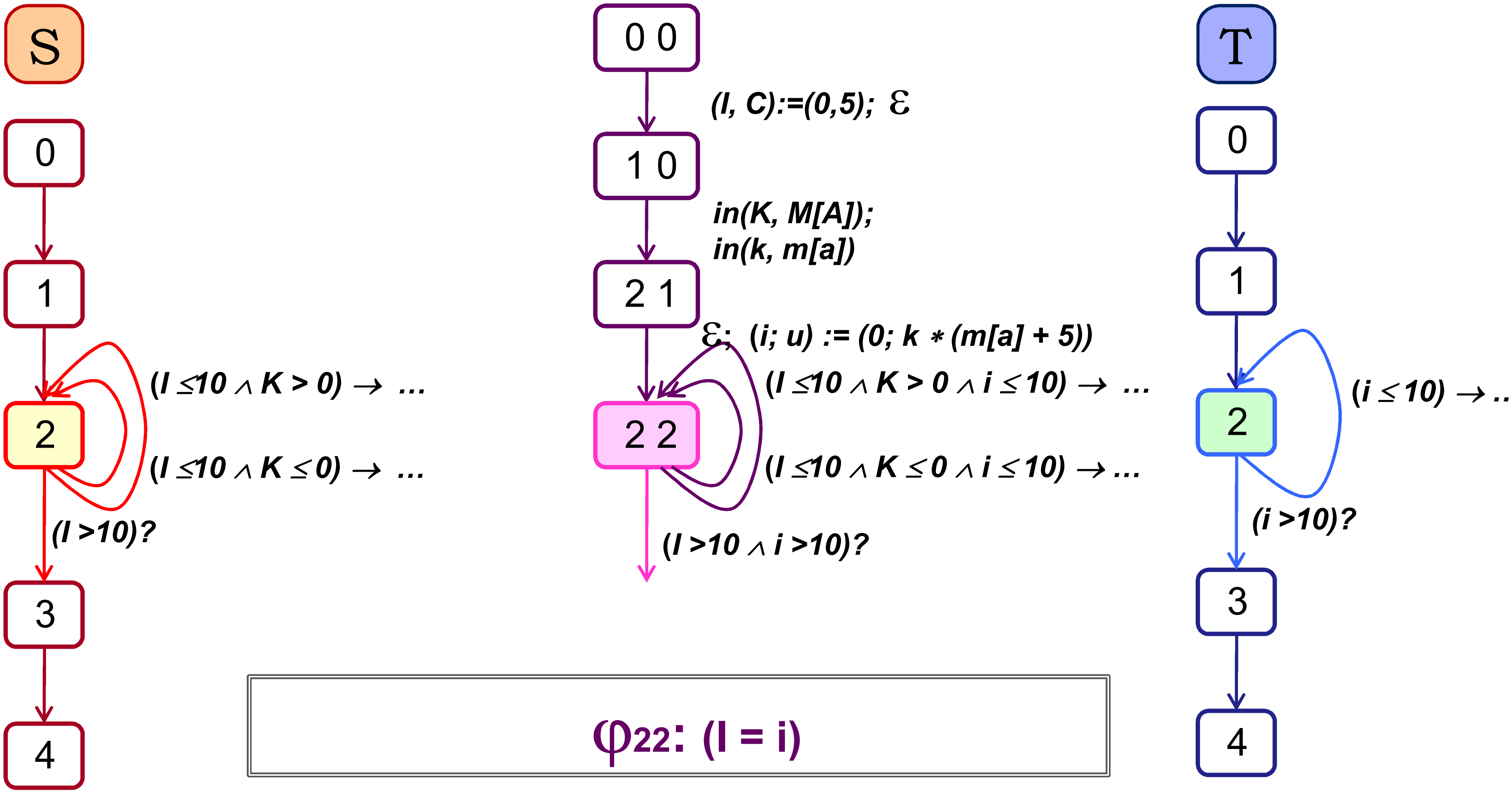




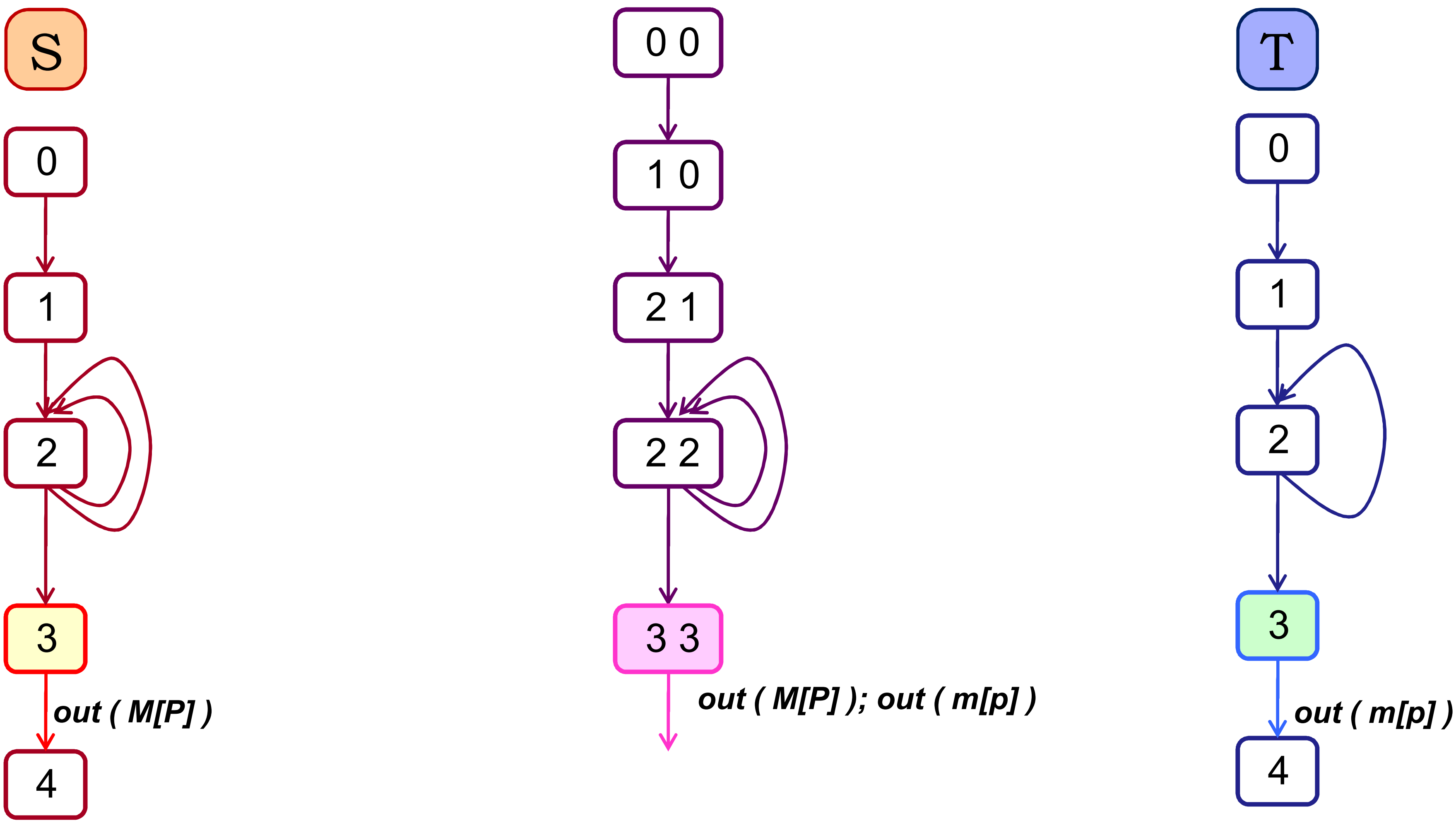
Match edges only if there exist a path through C_K in which both conditions are simultaneously enabled:

- Construct φ_{22}: $(I = 0) \wedge (i = 0)$
- Match edges iff $(\varphi_{22} \wedge C_S \wedge C_T)$ is satisfiable

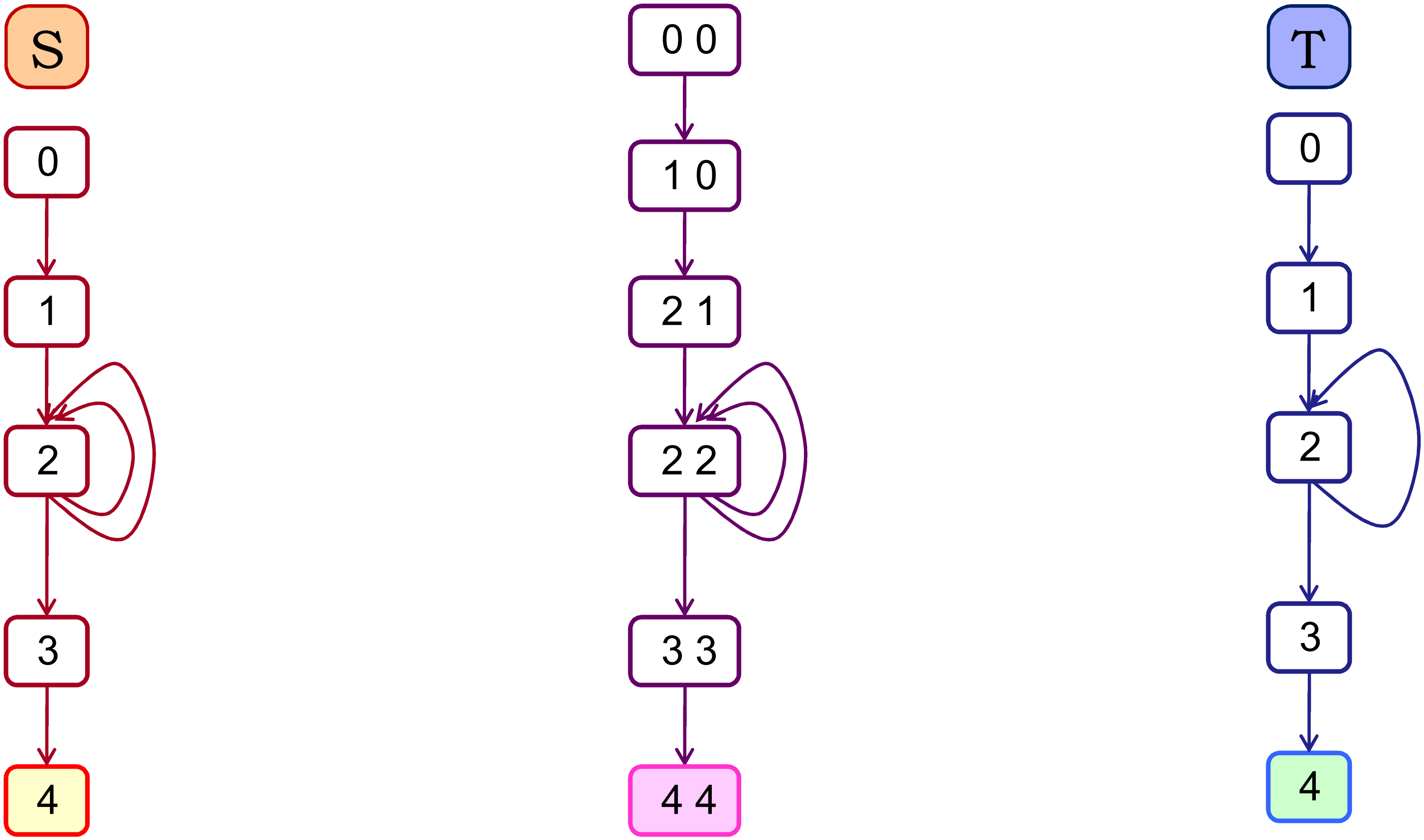
Comparison Graph Construction



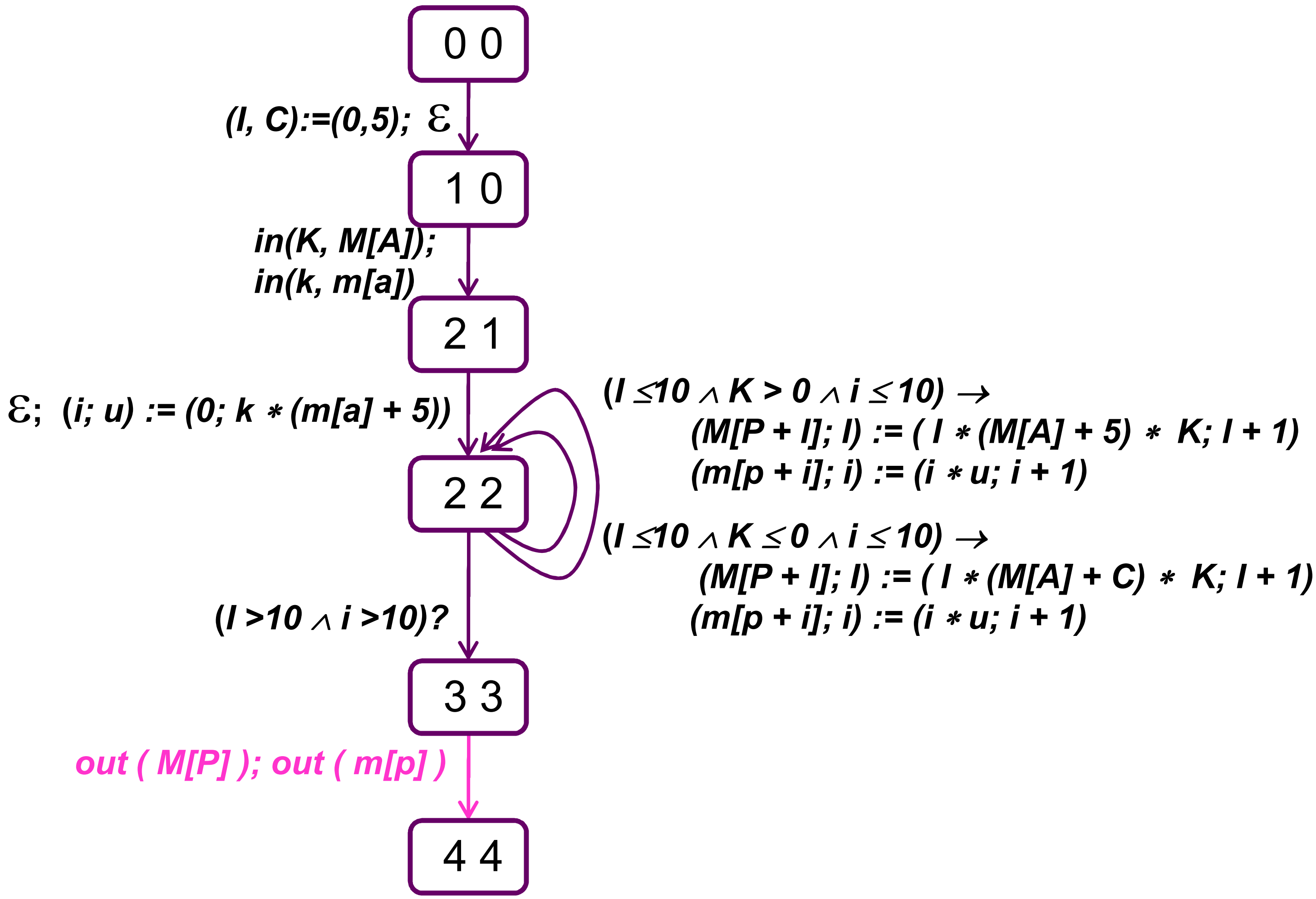
Comparison Graph Construction



Comparison Graph Construction



The Witness Comparison Graph



Construction of $C = S \times T$

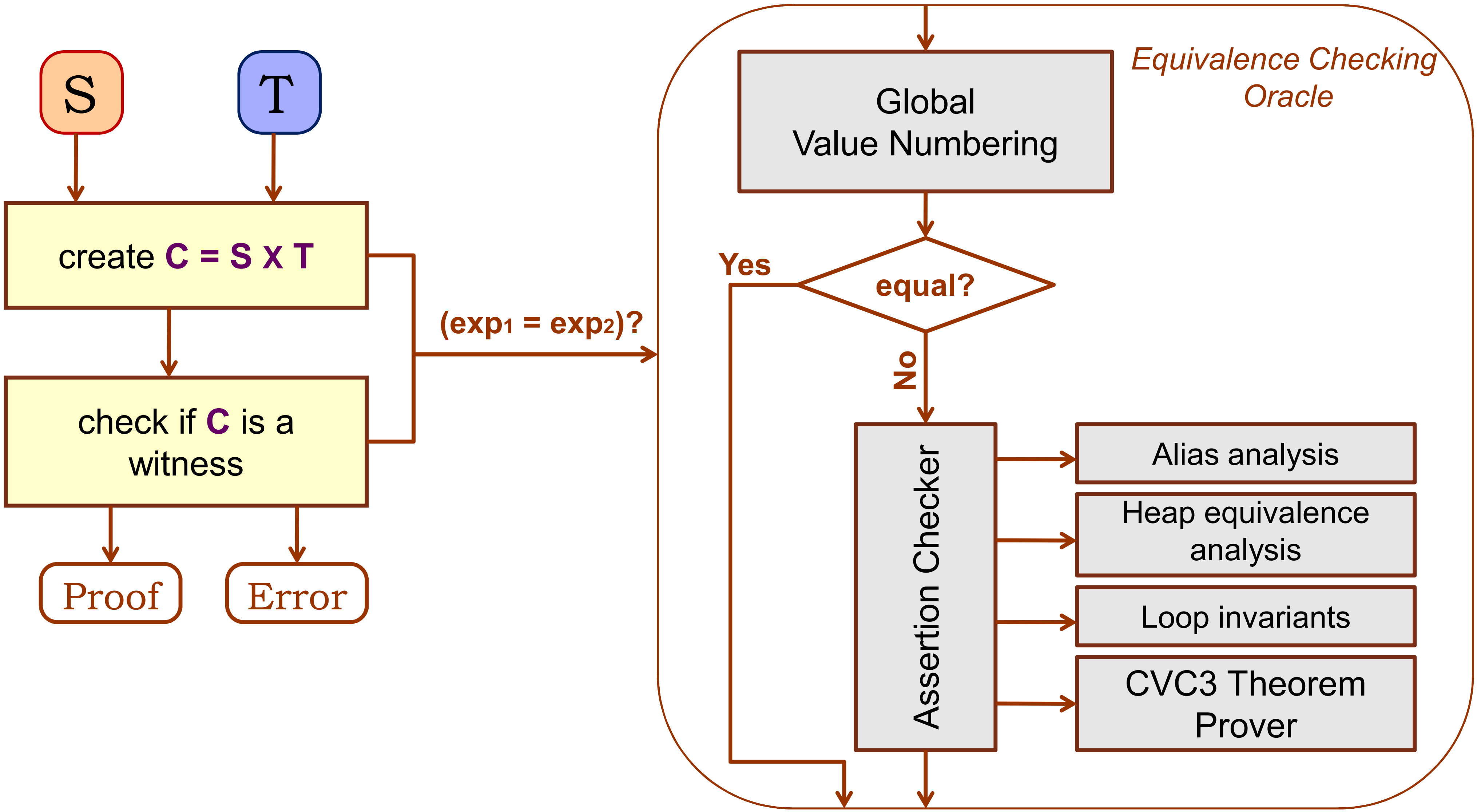
Theorem:

The following are the properties of the construction algorithm when it is applied to consonant programs:

- termination
- soundness
- conditional completeness - false alarms

  (it succeeds only if given strong enough invariants)

The CoVaC Tool

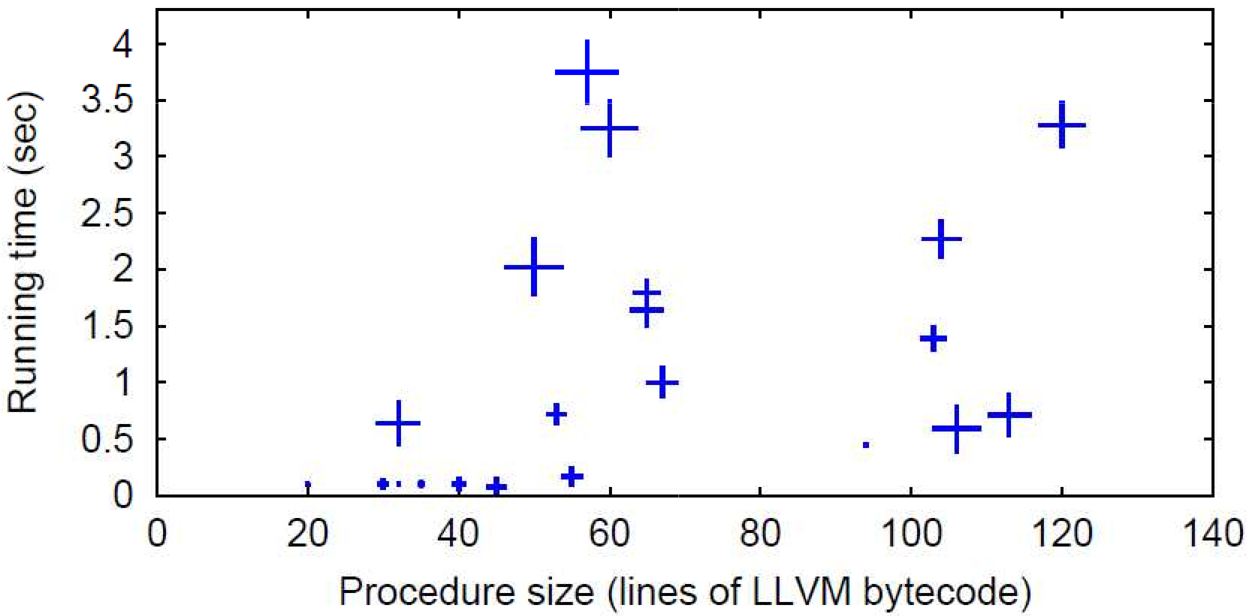


LLVM-based Implementation

- Why LLVM?
  - Typed low level intermediate language
  - Aggressive intraprocedural and interprocedural optimizations and analysis
  - Easy to extend (add an optimization/analysis pass)
  - Very well designed, documented, and supported !!!

CoVaC: Experimental Results

- Tested on third party implementation of classical algorithms like in-place heapsort, mergesort, qsort, strcmp, shortest paths, etc



Related Work

Certified Compilers: Given a source program, it either produces a target program observationally equivalent to the source or raises an error.

- The CompCert verified compiler - a formal certification of a complete compilation chain using the Coq proof assistant
- Cobalt, Rhodium – frameworks for writing compiler optimizations that can be automatically proved sound

Validation algorithms specialized to particular optimizations:

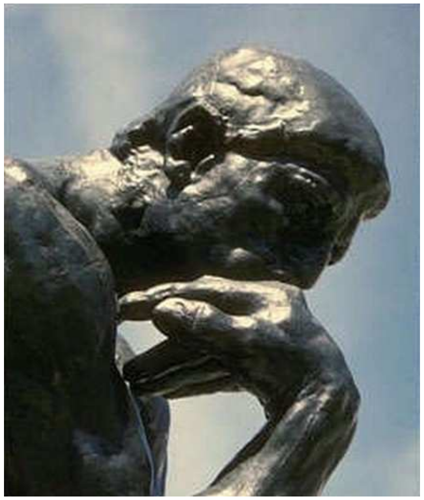
- Catching and identifying bugs in register allocation.(Huang et al. 2006)
- Formal verification of translation validators: A case study on instruction scheduling optimizations and Verified validation of lazy code motion (Tristan and Leroy 2008, 2009)

Compiler Bug Finding:

- Volatiles Are Miscompiled, and What to Do about It. (Eide and Regehr 2008)
- Practical testing of a C99 compiler using output comparison.(Sheridan 2007)

Conclusion

- CoVaC
  - Assumes the program is correct before compilation
  - Constructs a proof that the optimization path of the compiler preserves the semantics of the program
- Directions for future work
  - Apply of CoVaC to development of a self-certifying compiler
    - Currently, the validator is trying to guess the relation between the variables
    - The compiler can provide that infromation
  - Experiment with more lightweight analysis
  - Extend of the set of supported optimizations
    - add interprocedural and loop reordering optimizations



Questions?