Incremental timing analysis of synchronous programs

Partha S Roop

Joint work with Hugh Wang

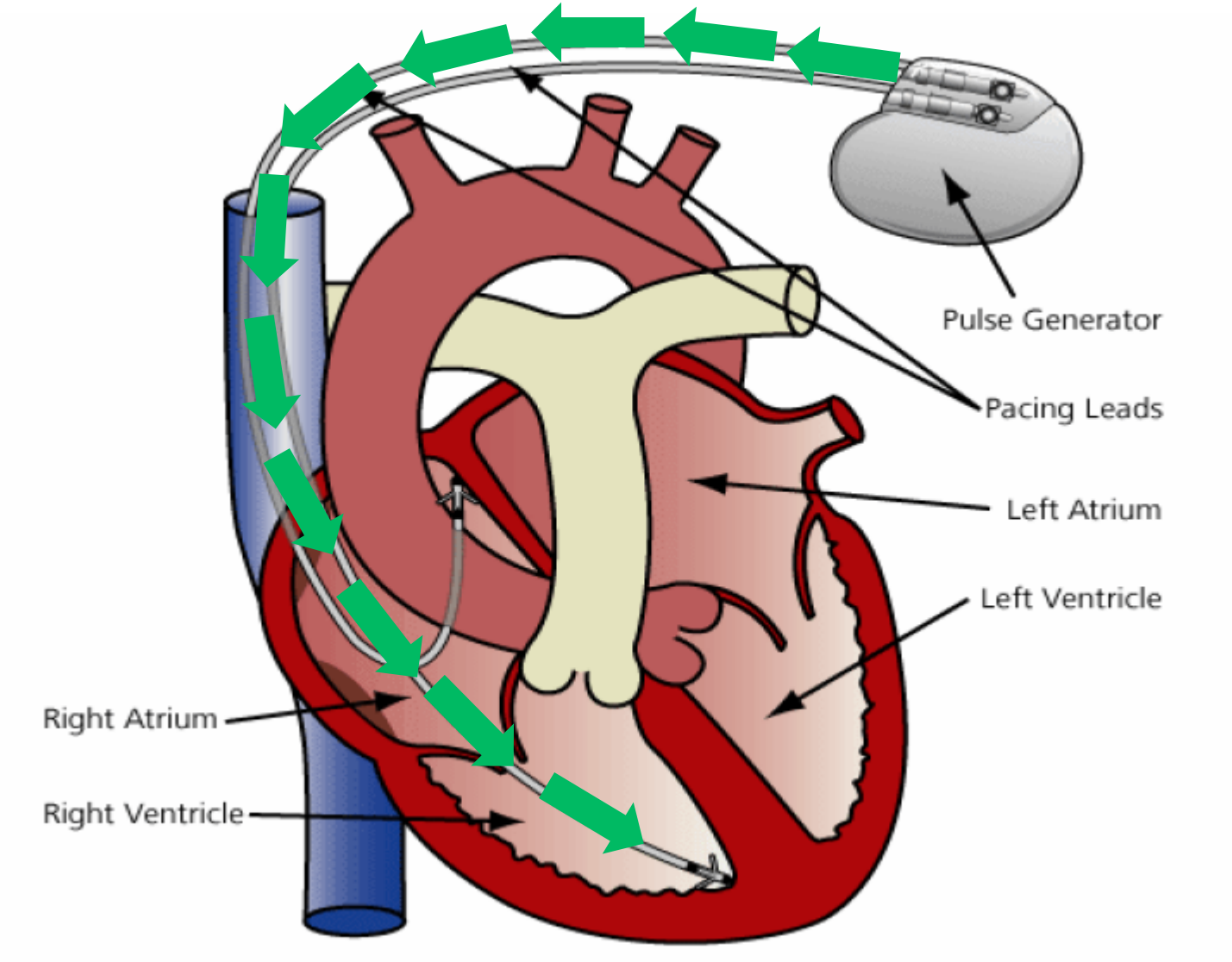
Outline

- Background
  - PRET-C
  - Timed Concurrent Control Flow Graph
- Problem statement
- Methodology
- Results
- Conclusions

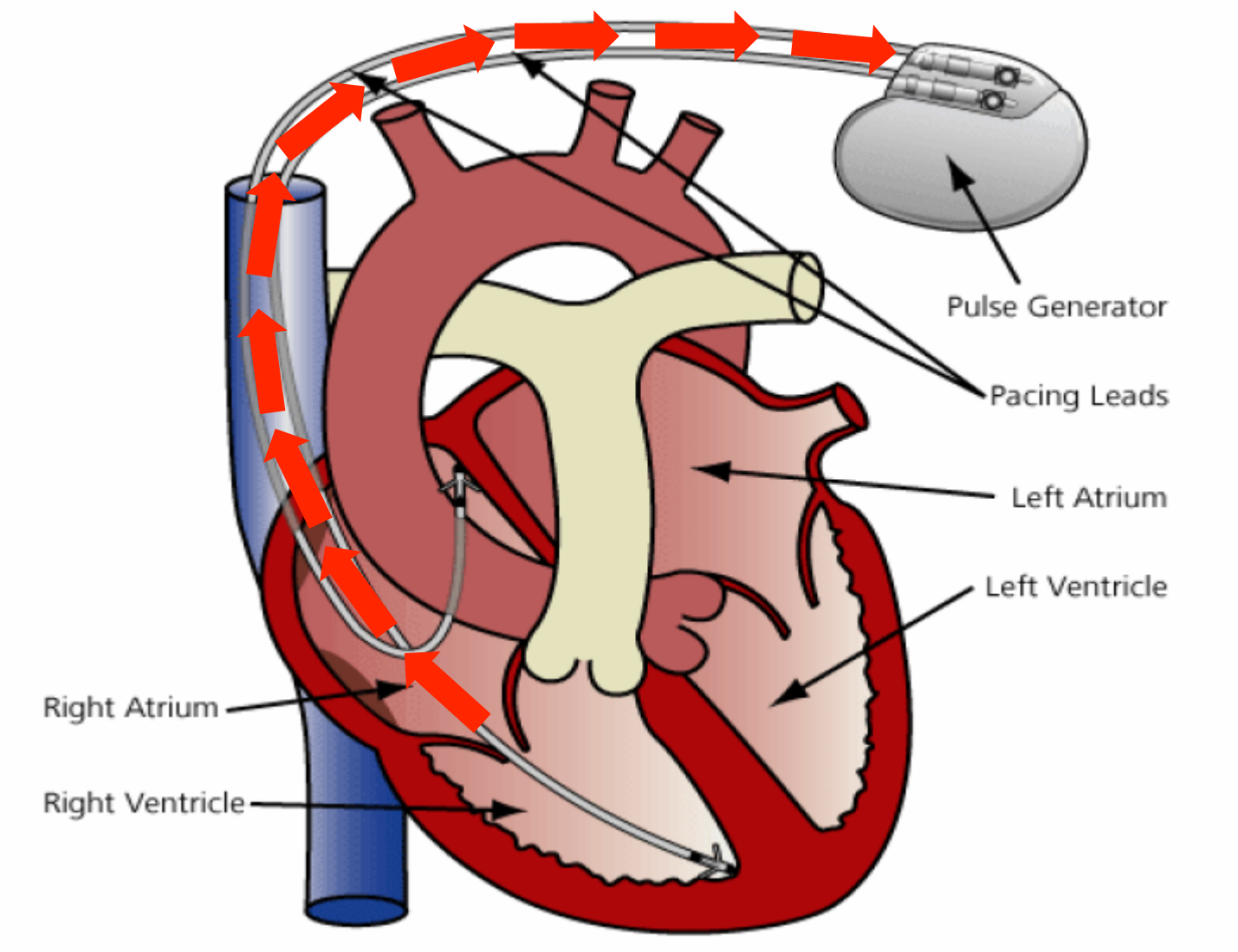
Key terms

- Worst case execution time (WCET) – It is the maximum time taken by a software on a given hardware architecture, and a given application context. WCET analysis is the task of computing a safe over-approximation of the actual WCET [Puschner et al. RTAS'06]
- Worst case response time – While WCET is defined relative to a single task, response time is usually defined for for a set of concurrent tasks. Response time is defined as the time for computing a response (output) relative to a given stimulus (input).

IMPLANTABLE CARDIAC PACEMAKER



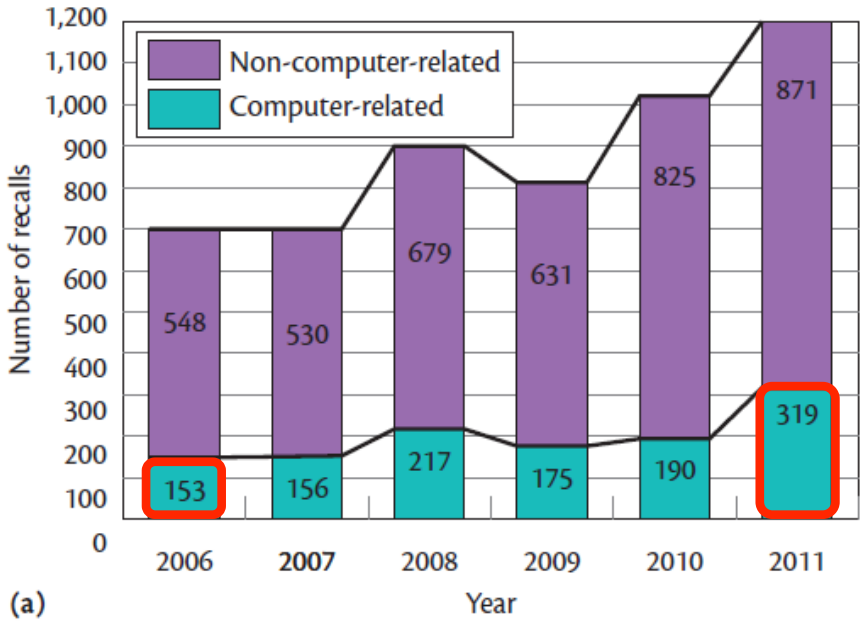
IMPLANTABLE CARDIAC PACEMAKER

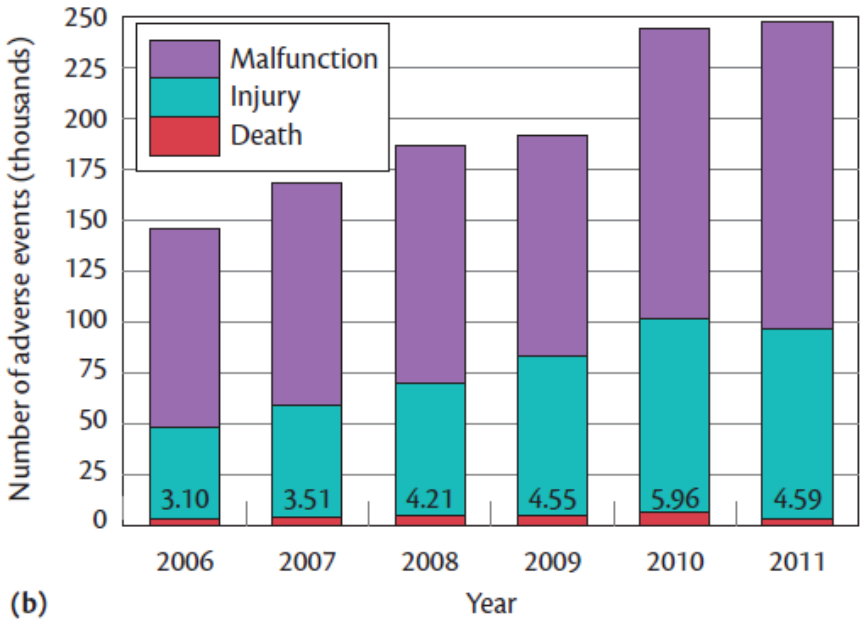


A dual chamber pacemaker

- A pacemaker maintains the timing relationship between the atrial and ventricular events [Jiang et al. TACAS'12].
- These requirements are related to the response time of the pacemaker e.g.,
  - The maximum interval between two ventricular events is less than or equal to TLRI (avoid bradycardia).
  - There must not be any PMT.

FDA recalls and adverse events (2006-2011)

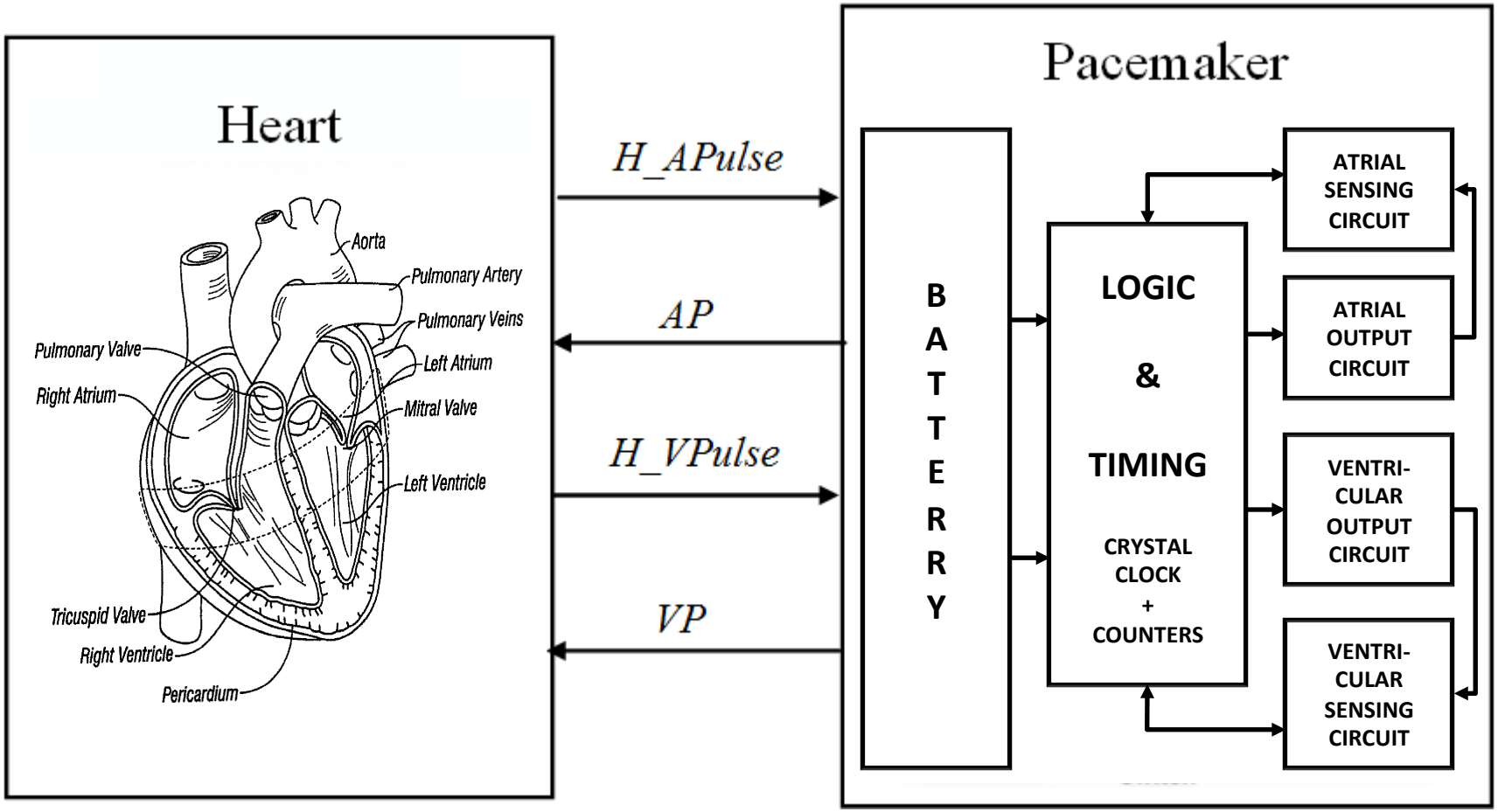




	Class I: high risk	Class II: medium risk	Class III: low risk	Total recalls	Number of devices
Software	14 (33.3%)	718 (65.6%)	46 (75.3%)	778 (64.3%)	2,303,441 (19.2%)
Hardware	8 (19.0%)	158 (14.4%)	13 (27.4%)	179 (14.8%)	4,228,133 (35.2%)
Other	10 (23.8%)	124 (11.3 %)	8 (12.3%)	142 (11.7%)	2,831,048 (23.5%)
Battery	8 (19.0%)	57 (5.2%)	5 (6.8%)	70 (5.8%)	2,385,613(19.8%)
I/O	2 (4.8%)	38 (3.5%)	1 (2.7%)	41 (3.4%)	276,601 (2.3%)
Total recalls	42 (3.5%)	1,095 (90.5%)	73 (6.0%)	1,210	12,024,836

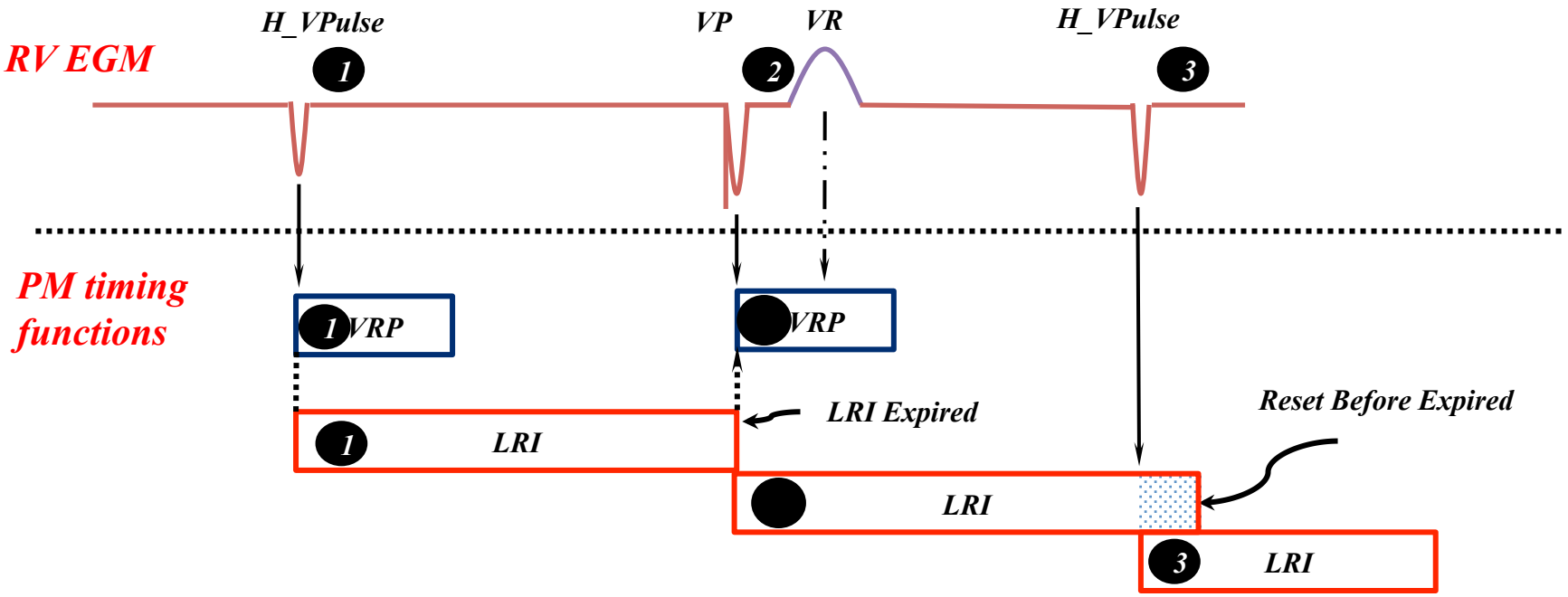
[Ref.]: Alemzadeh, H., Iyer, R.K., Kalbarczyk, Z., Raman, J., "Analysis of Safety-Critical Computer Failures in Medical Devices", *Security and Privacy* , IEEE , vol.11, no.4. pp.14,26. July-Aug, 2013.

IMPLANTABLE CARDIAC PACEMAKER

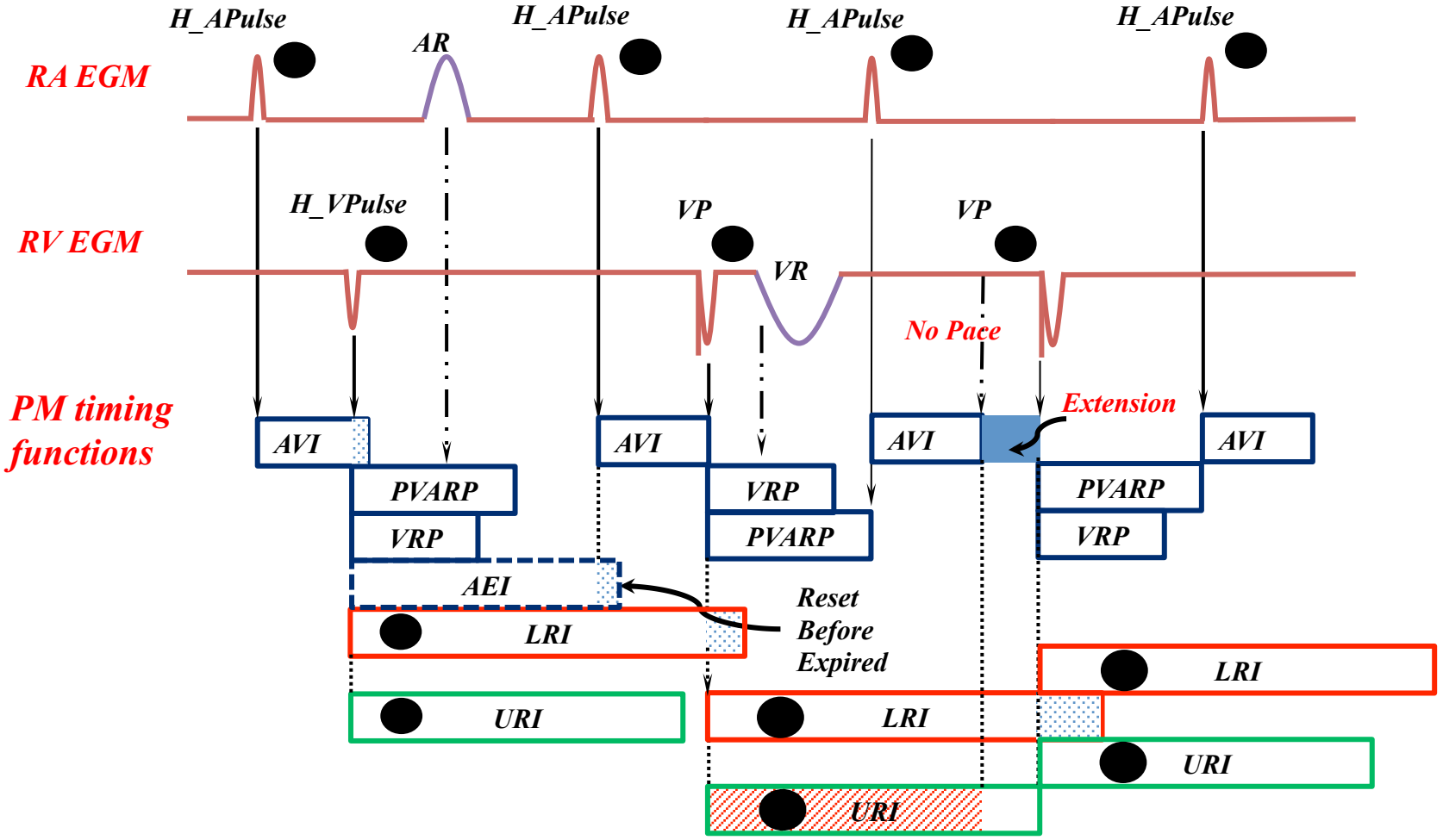


S. Serge Barold, Roland X. Stroobandt, and Alfons F.Sinnaeve, *Cardiac Pacemakers Step by Step*, Futura Publishing, 2004.

TIMING DIAGRAM FOR VVI MODE



TIMING DIAGRAM FOR DDD MODE



WCET analysis of synchronous programs

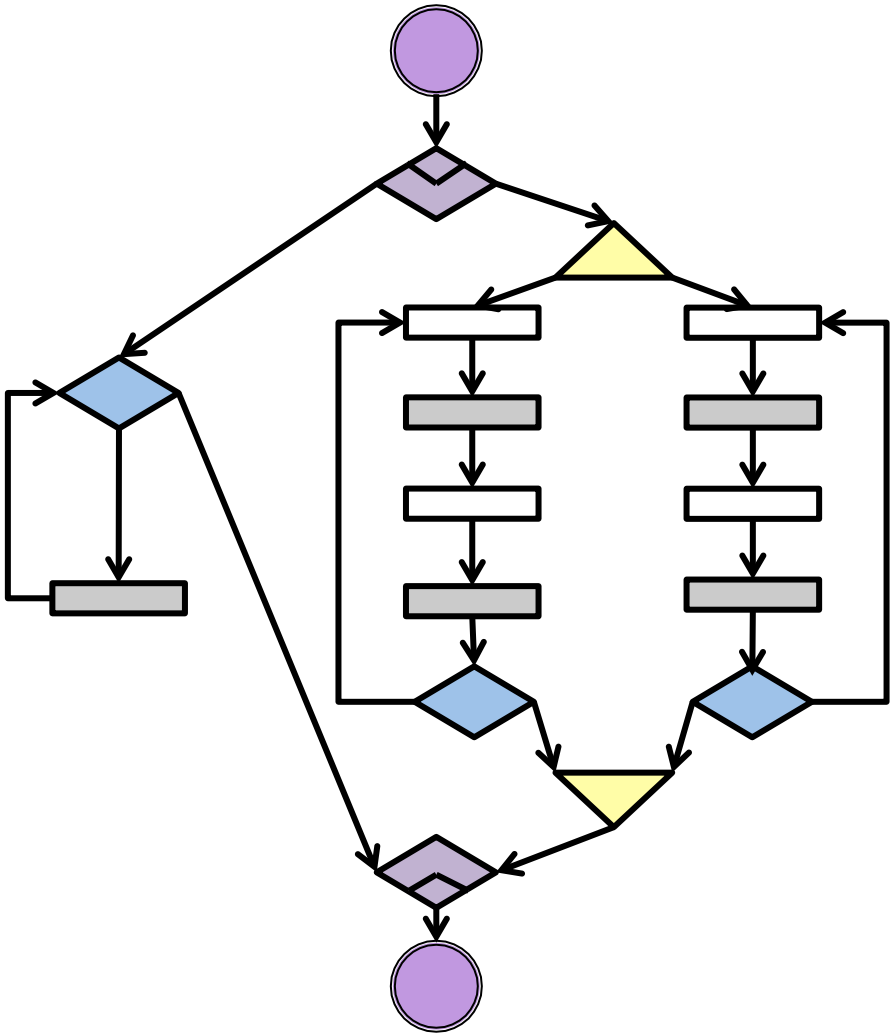
- Also referred to as worst case reaction time (WCRT) analysis.
- Here, system computes a set of reactions indefinitely. Reactions are also known as ticks.
- WCRT refers to the worst case length of any reaction / tick.
- Objectives:
  - How to satisfy timing constraints (Partha)
  - How to also minimize power consumption (Hugh)

Related work

- Plus-max [Boldt et al. SLA++P'07, Mendler et al. DATE'09].
- IPET [Ju et al. journal of RTS'12, DAC'09].
- Model checking [Roop et al. CASES'09].
- Reachability [Kuo et al. DAC'11, Yip et al. ACSD'13].
- Techniques for improved precision:
  - Infeasible path elimination [Andalam et al. DATE'11]
  - Hybrid approach [Raymond et al. RTNS'13]: Model checking +ILP.

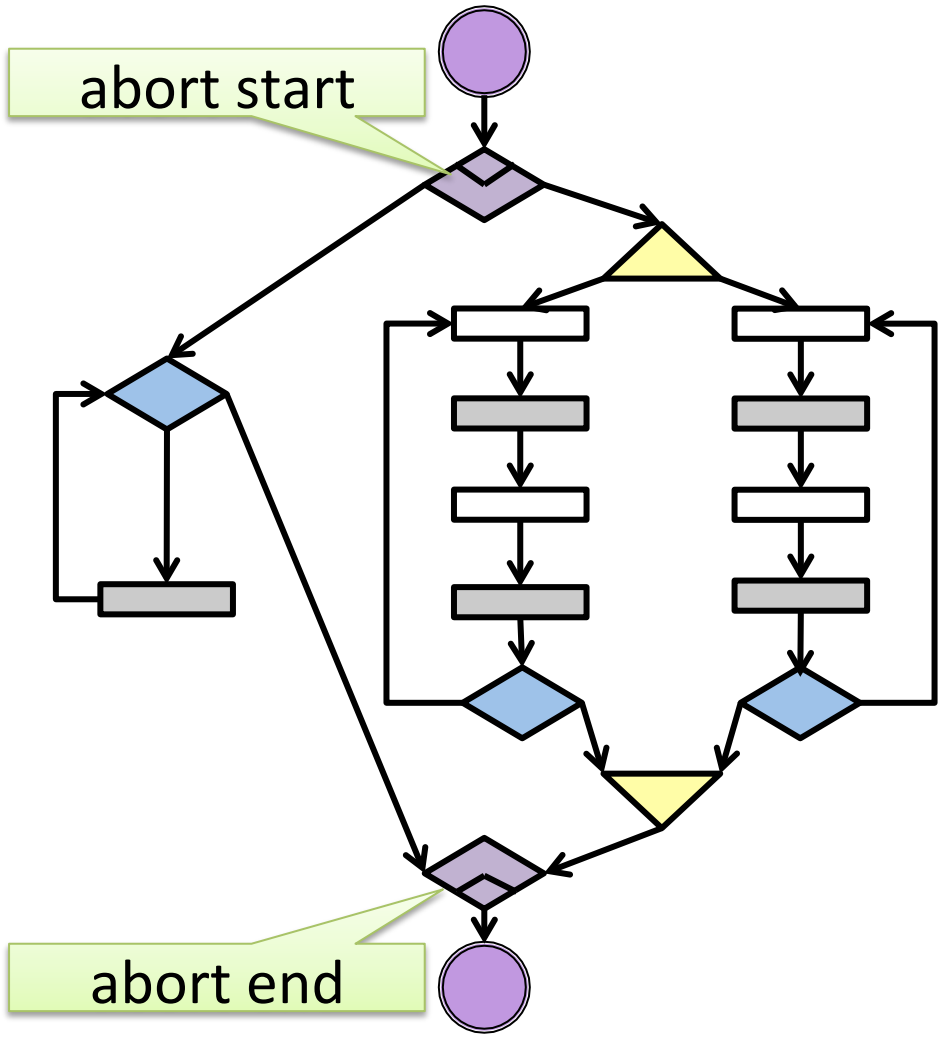
Synchronous program and TCCFG

```
int main (){
  abort{
    PAR(t1,t2);
  }when(S0)
}
```



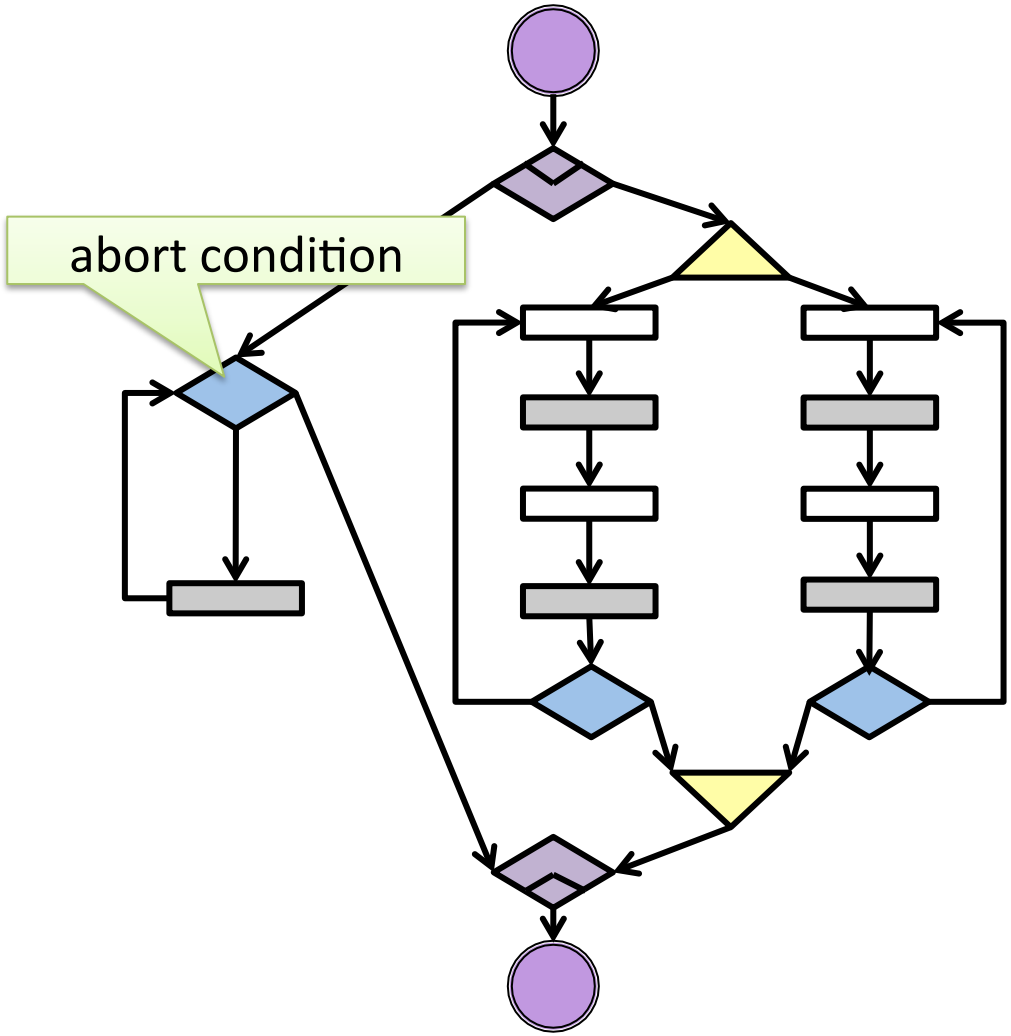
Execution

```
int main (){
 abort{
   PAR(t1,t2);
 }when(S0)
}
```



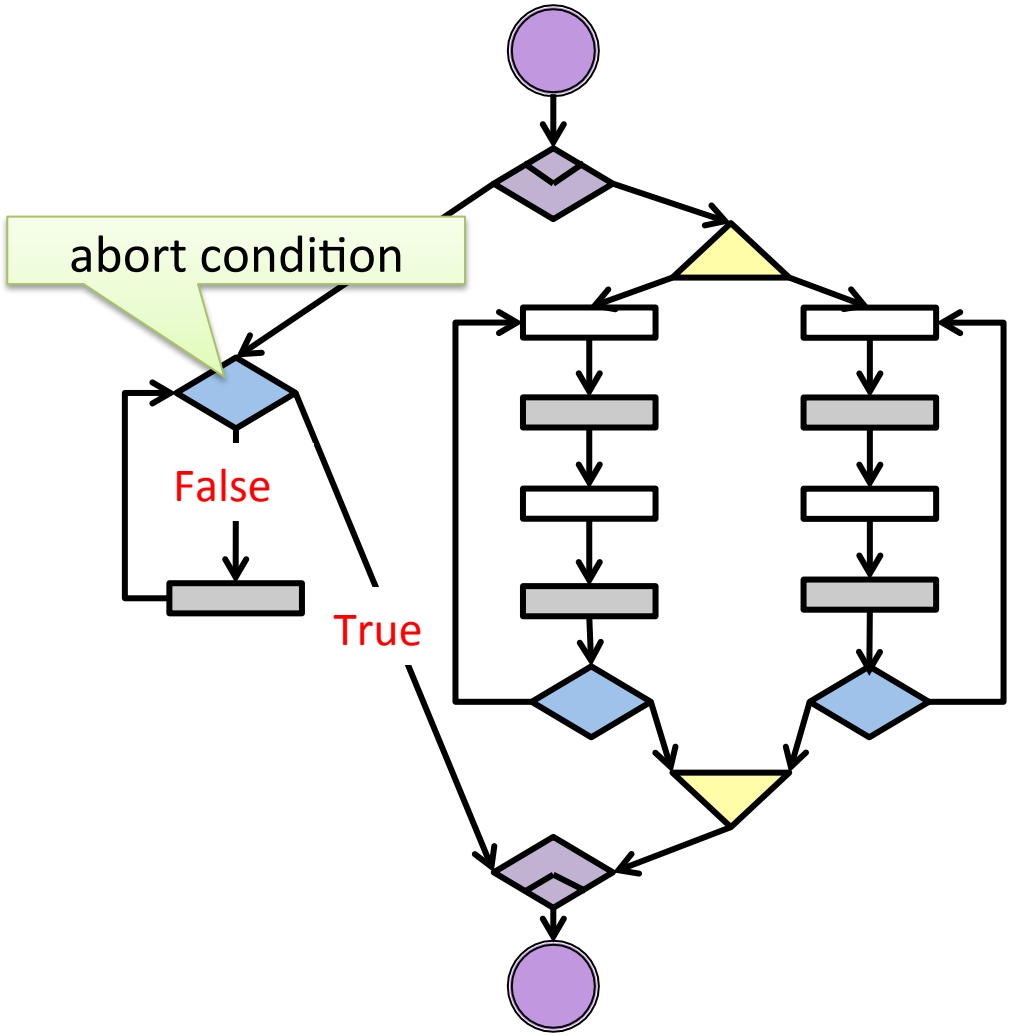
Execution

```
int main (){
 abort{
   PAR(t1,t2);
 }when(S0)
}
```



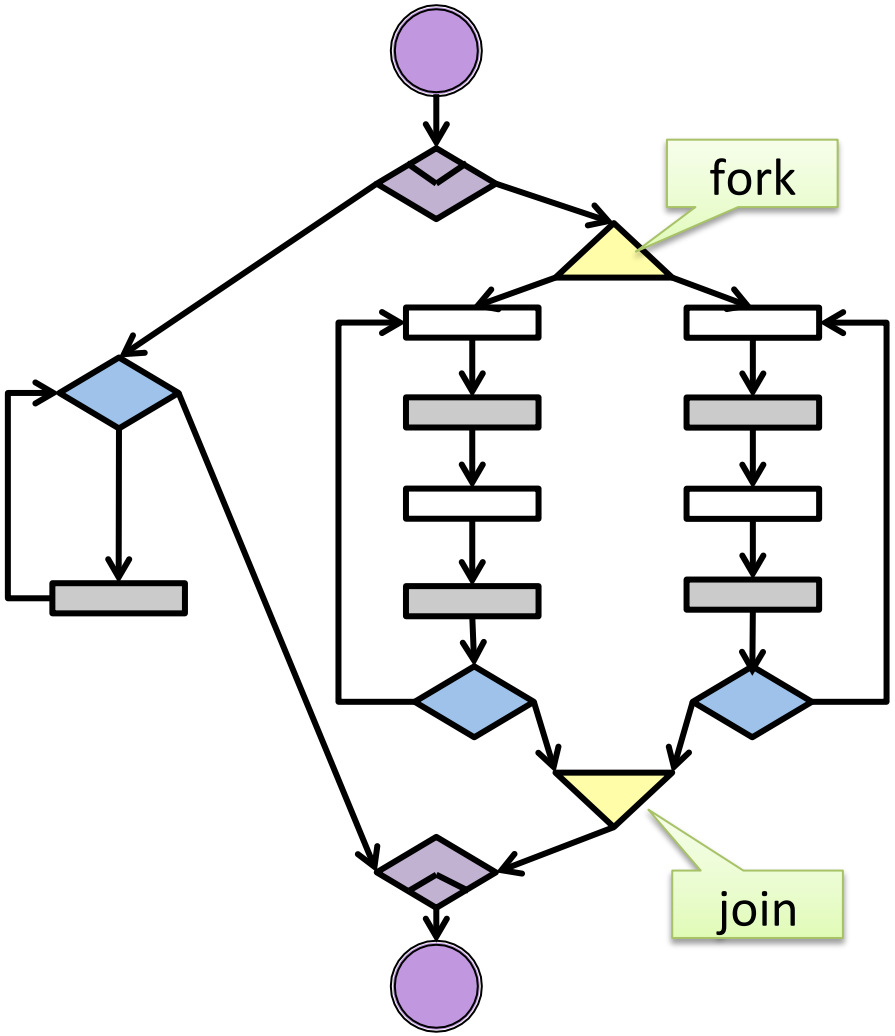
Execution

```
int main (){
 abort{
   PAR(t1,t2);
 }when(S0)
}
```



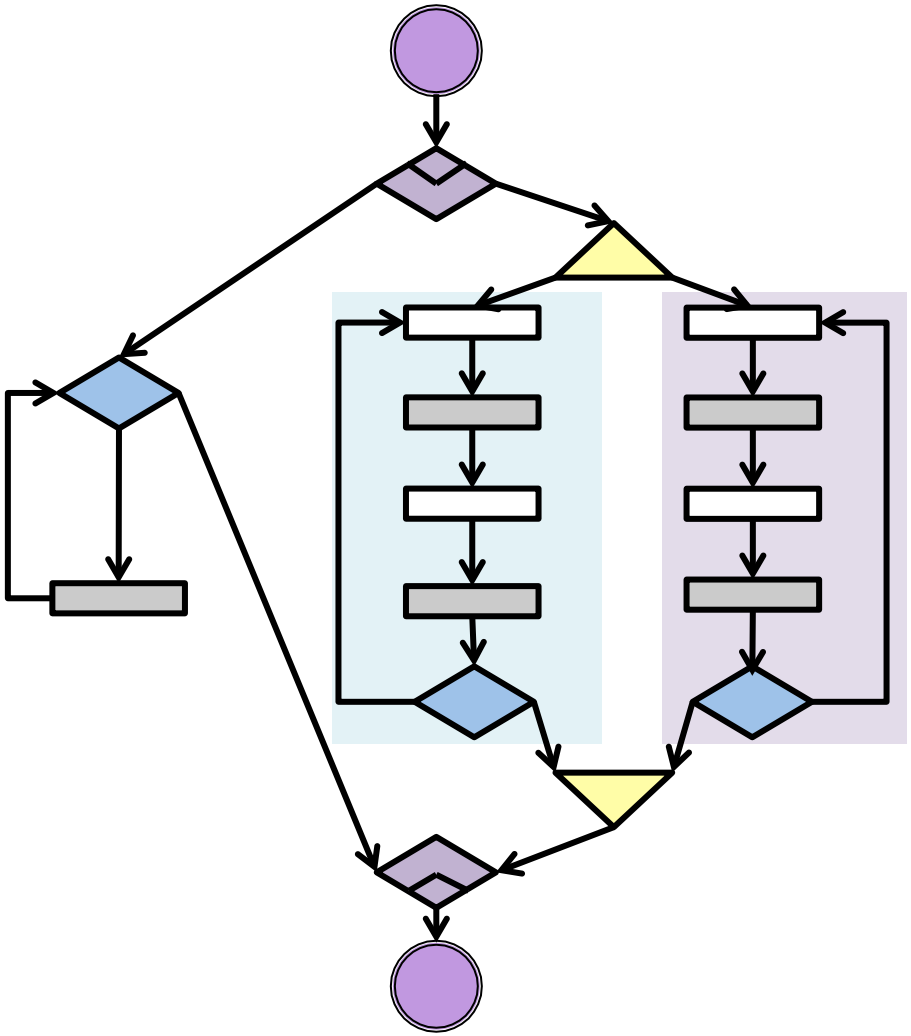
PRET-C and TCCFG

```
int main (){
  abort{
    PAR(t1,t2);
  }when(S0)
}
```



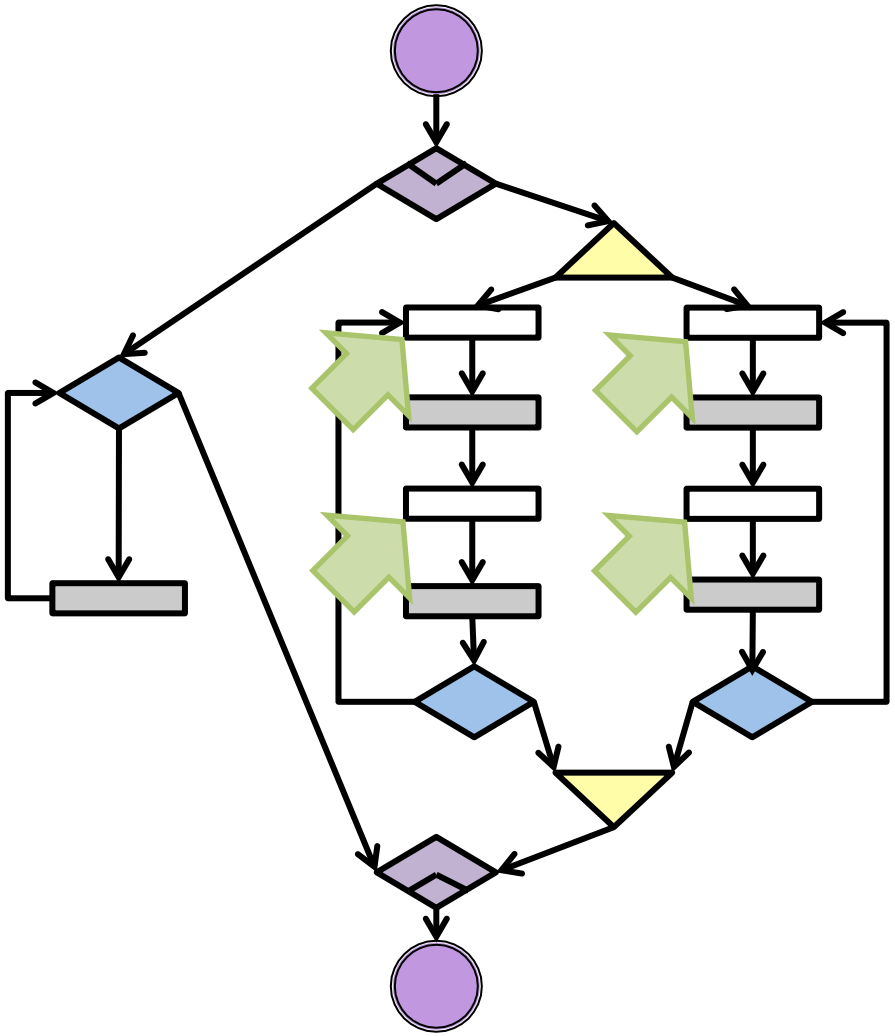
Execution

```
thread t1(){
  do{
    foo1();
    EOT;
    foo2();
    EOT;
  }while(S1)
}

thread t2(){
  do{
    foo3();
    EOT;
    foo4();
    EOT;
  }while(S2)
}
```

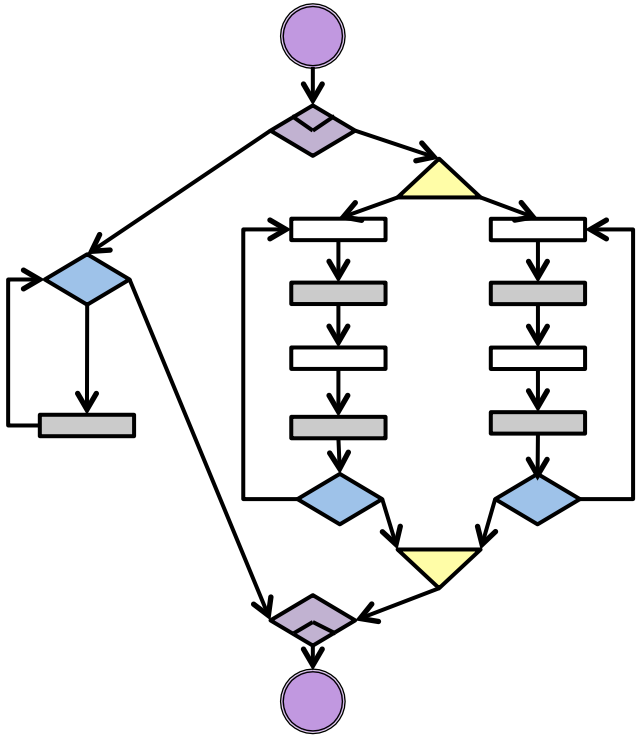


Execution

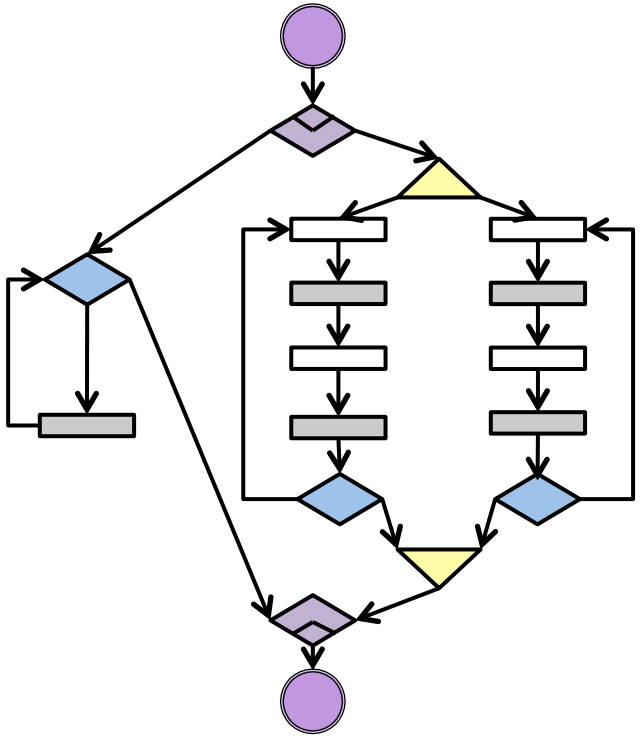
```
thread t1(){
  do{
    foo1();
    EOT;
    foo2();
    EOT;
  }while(S1)
}

thread t2(){
  do{
    foo3();
    EOT;
    foo4();
    EOT;
  }while(S2)
}
```



Execution paths in TCCFG



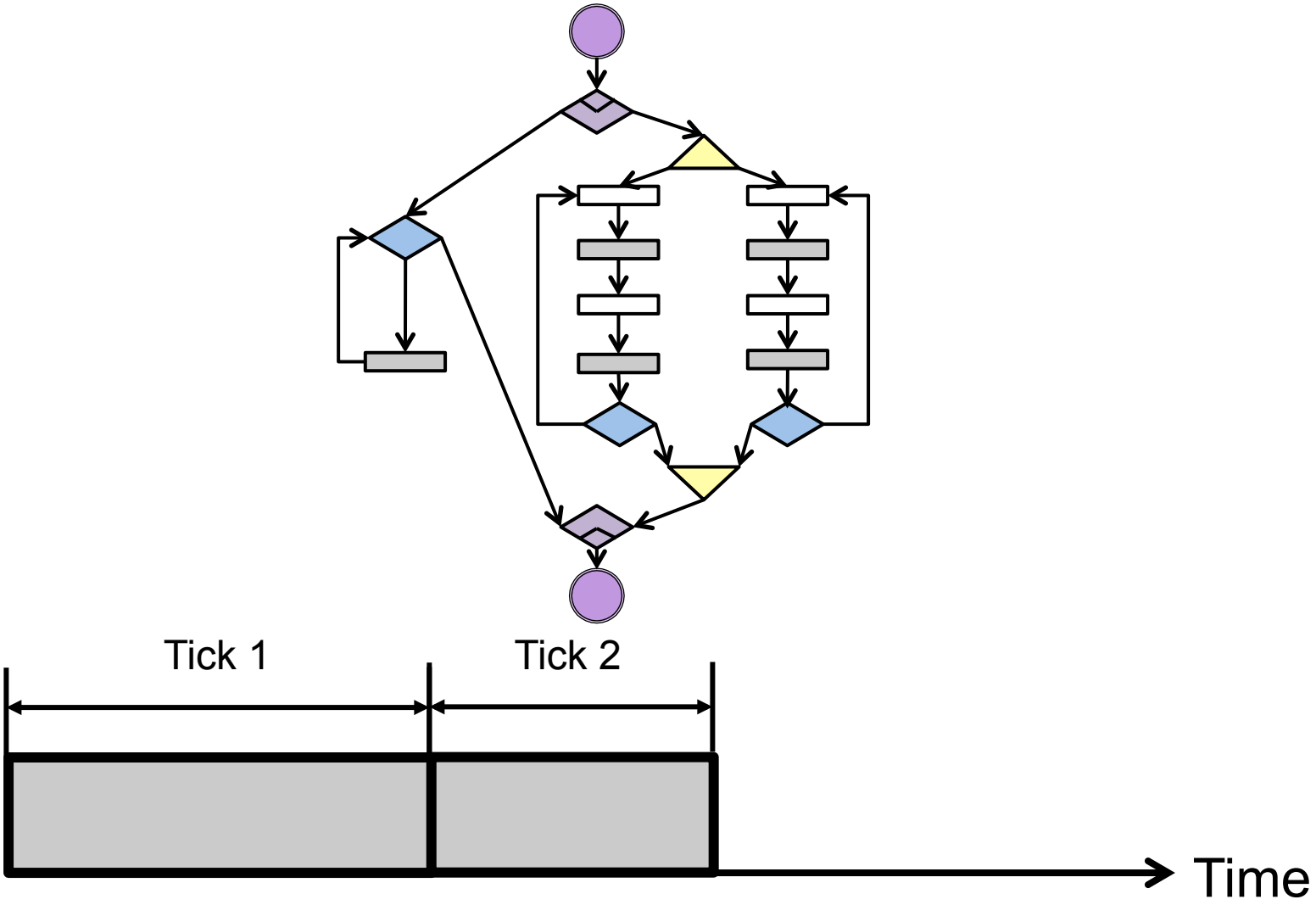
Execution paths in TCCFG



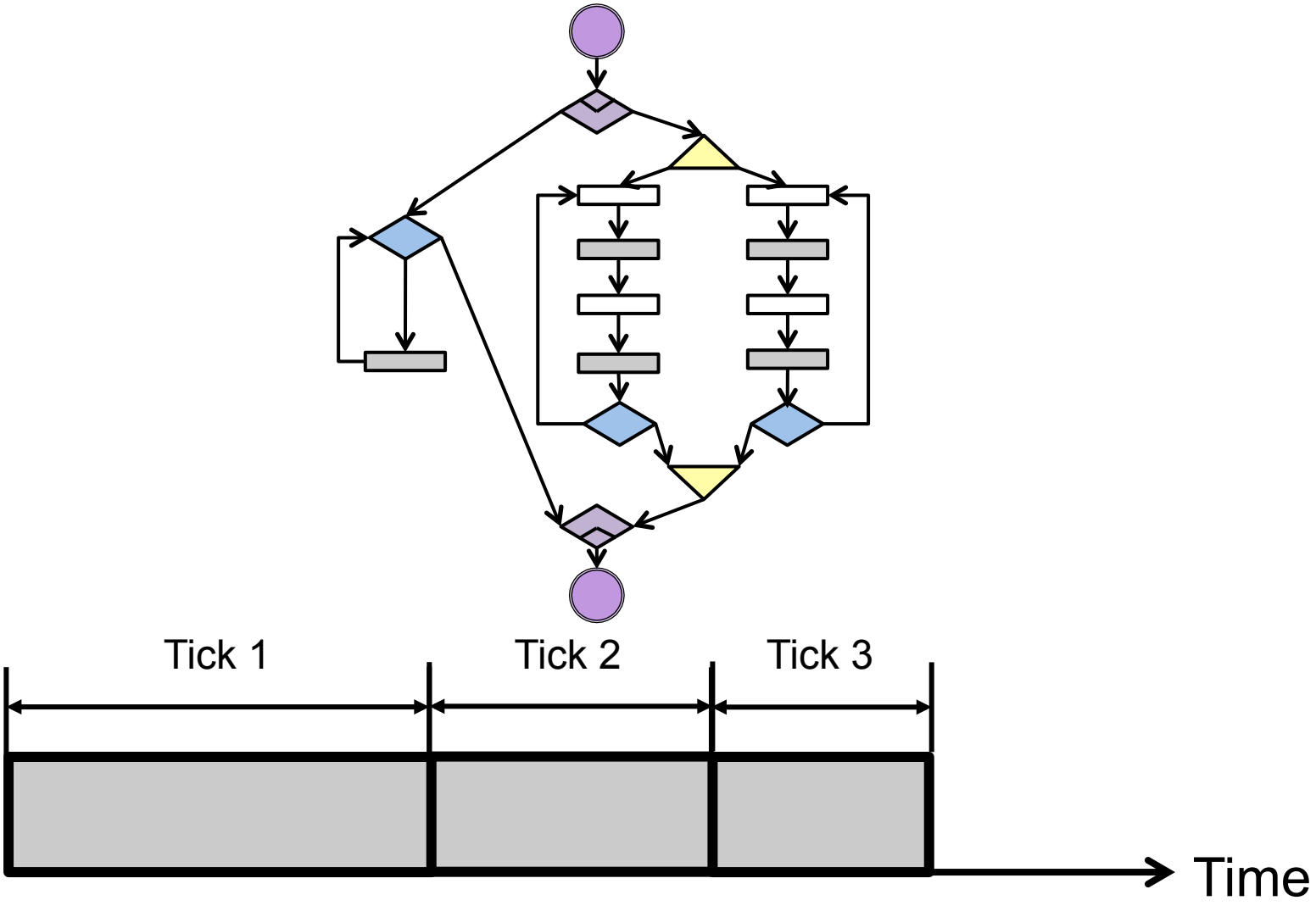
Threads execute in a fixed order.

From Left to Right

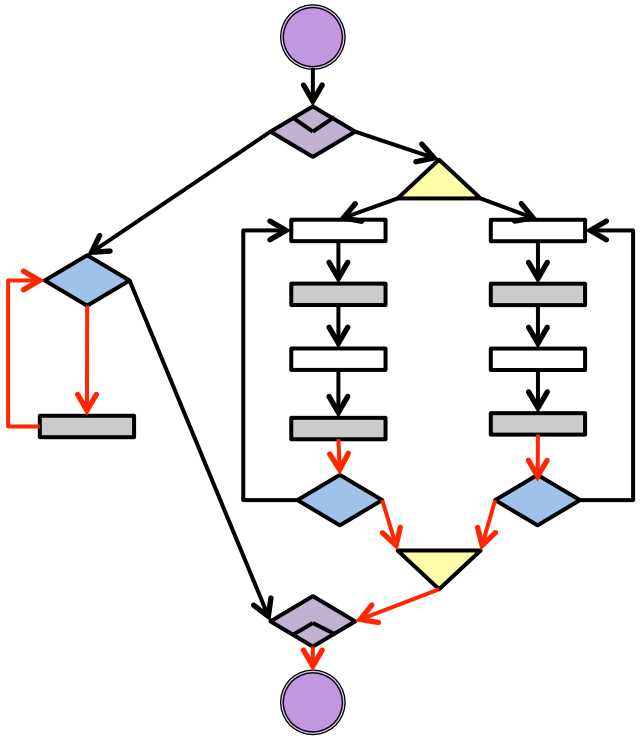
Execution paths in TCCFG



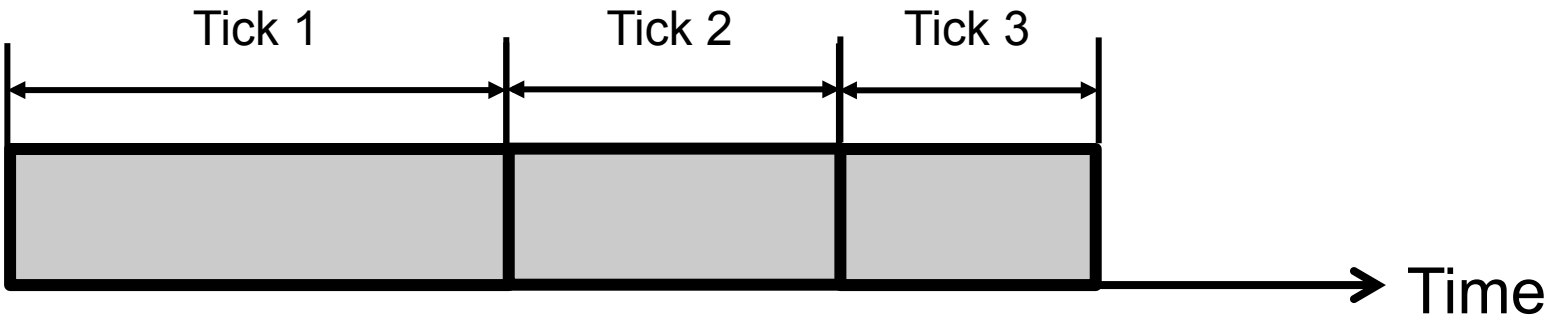
Execution paths in TCCFG



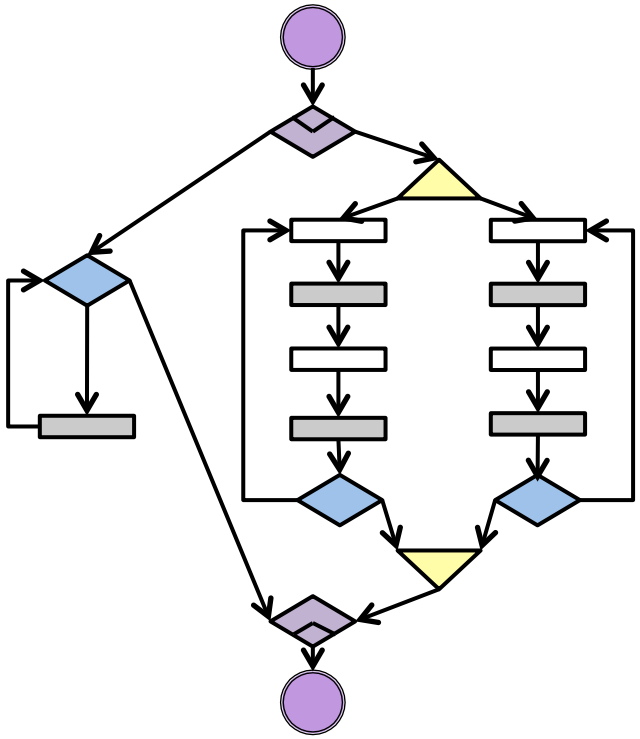
Execution paths in TCCFG



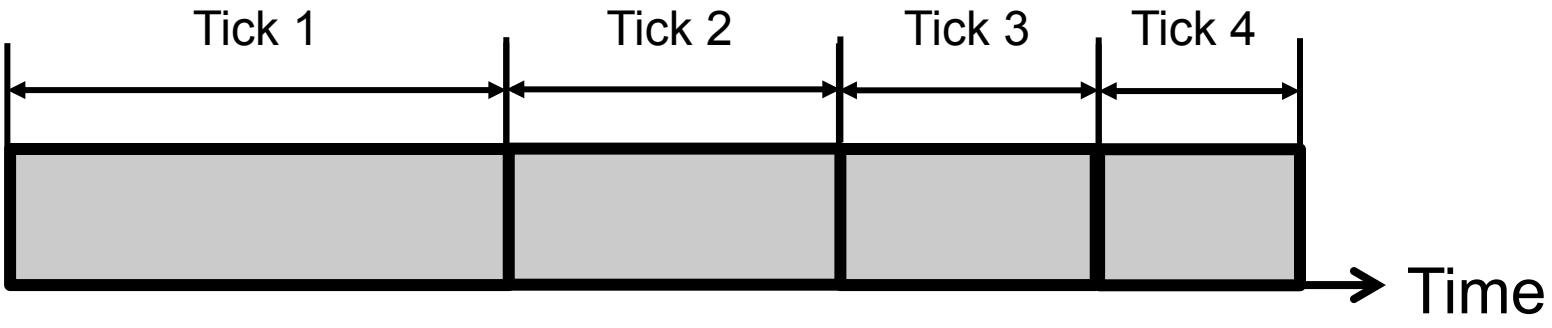
Join



Execution paths in TCCFG

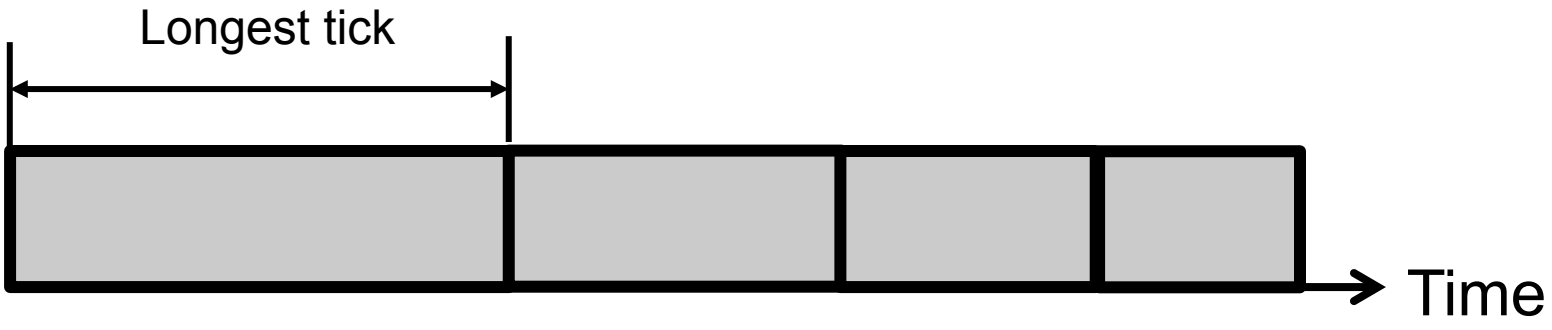


Preemption

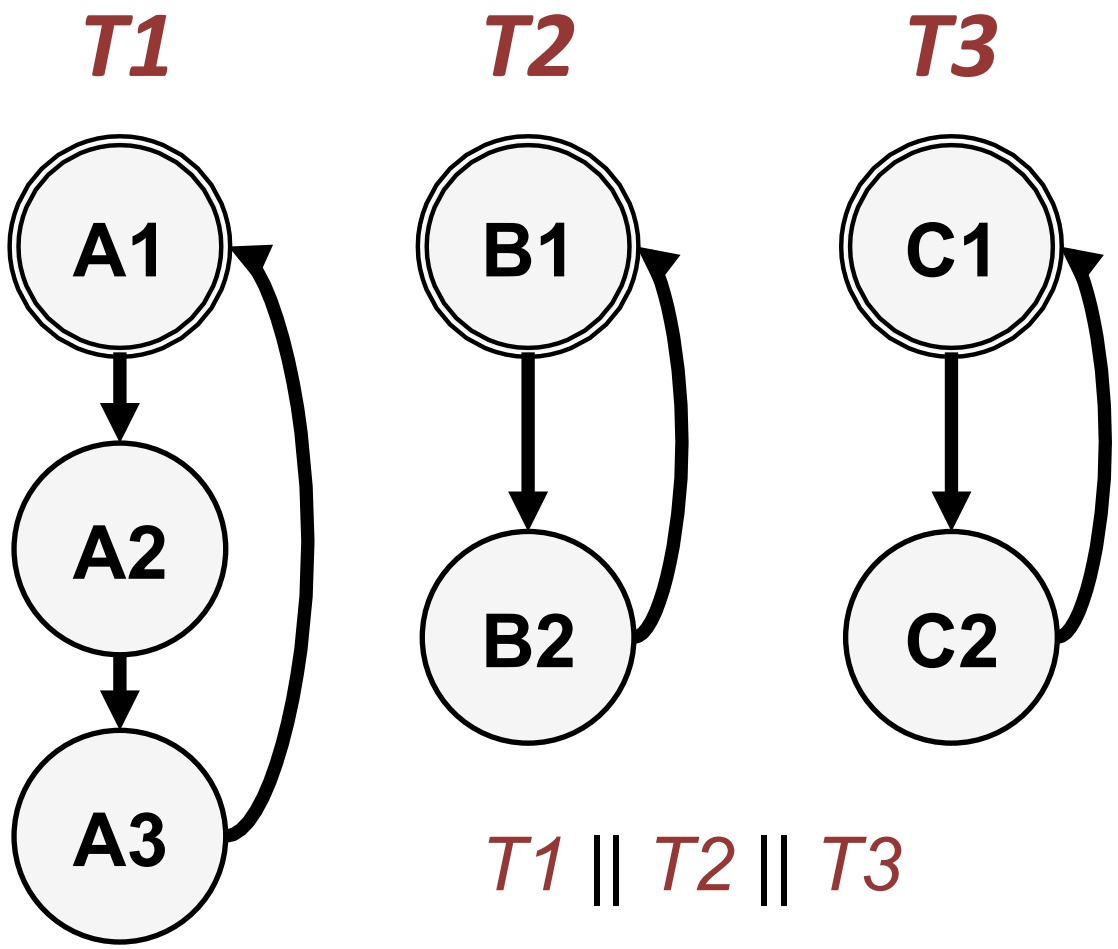


Synchronous languages

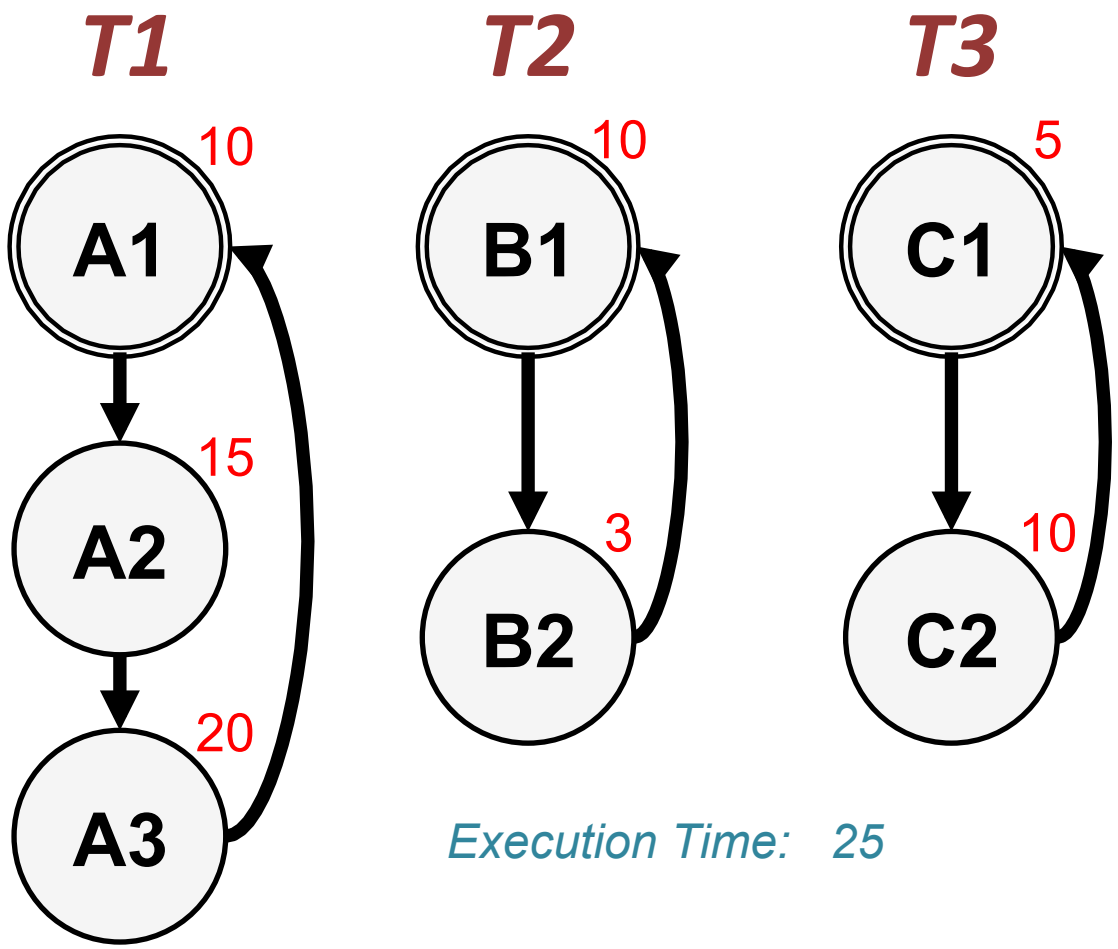
- Worst Case Reaction Time analysis
  - WCRT analysis
- Synchrony hypothesis
  - *Reactive system operates infinitely fast compared to the environment.*
- Validation
  - Min(inter-arrival-time of events) ≥ Max (Reaction Time)
  - Deadline ≥ WCRT



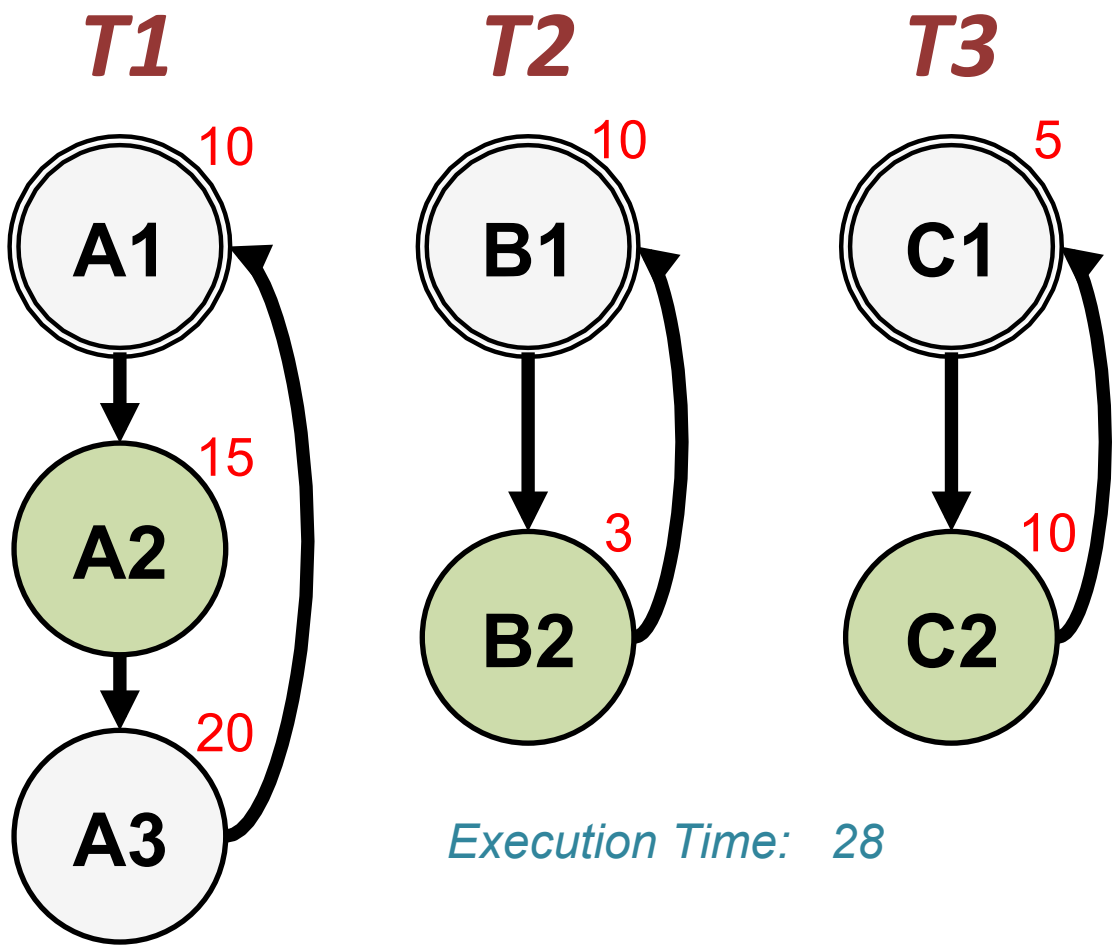
Motivating example



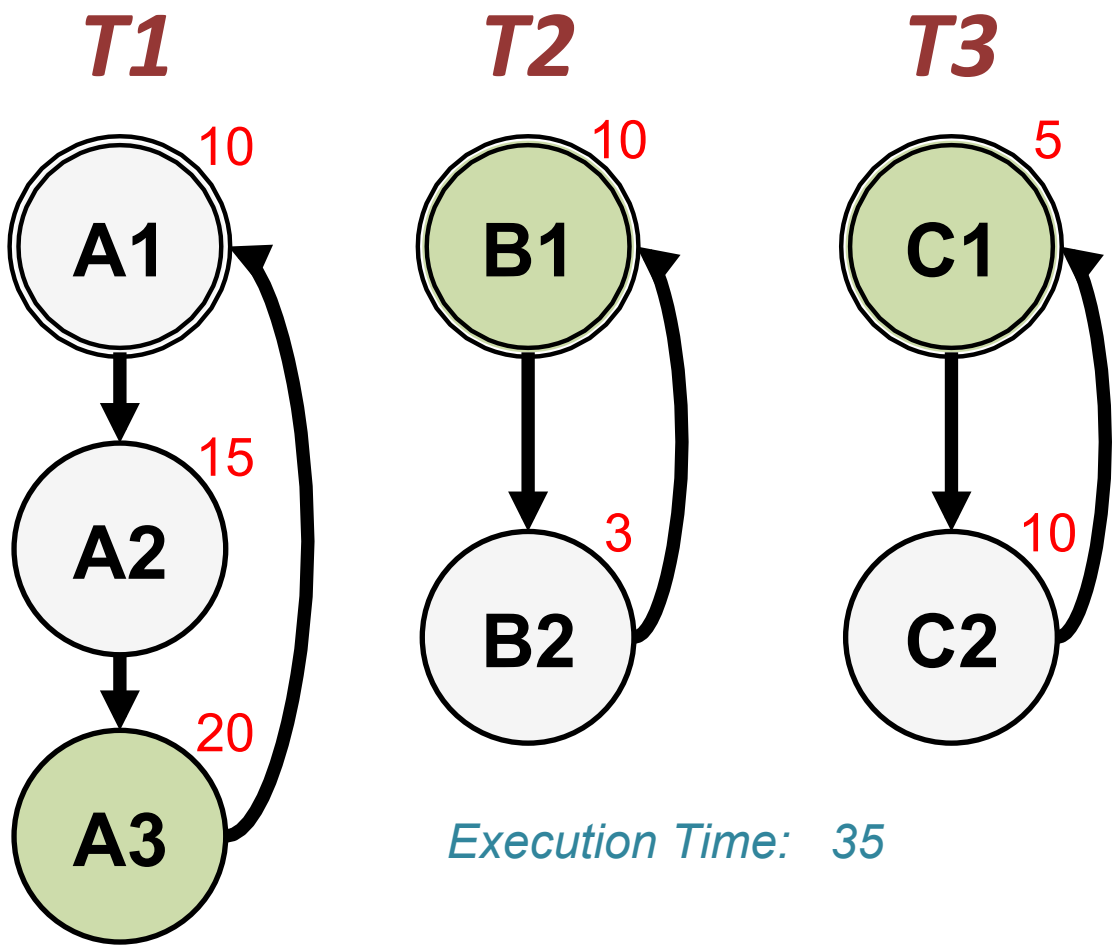
Motivating example



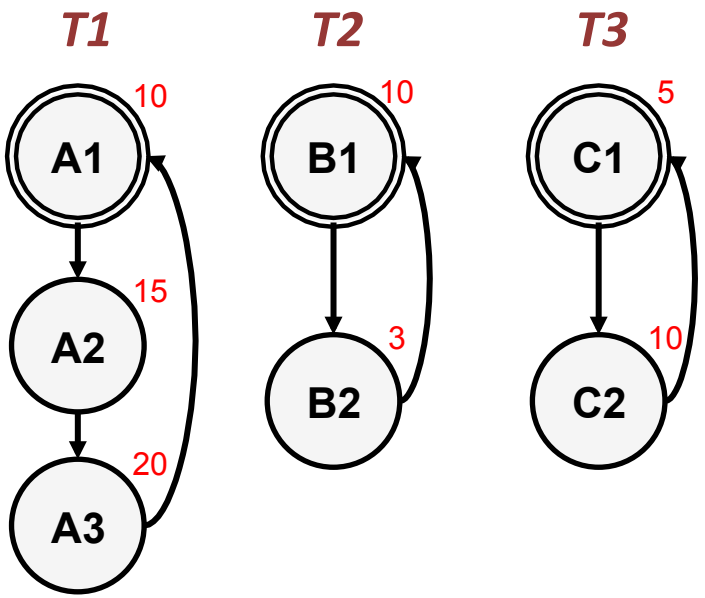
Motivating example



Motivating example



Motivating example



WCRT analysis

-Max-Plus

WCRT = Max(T1) + Max(T2) + Max(T3)

= A3 + B1

cycles

Tick alignment

-State Exploration

A1+B1+C1 = 25

A2+B2+C2 = 28

A3+B1

A1+

23

Scalability

A2+B1+C1 = 30

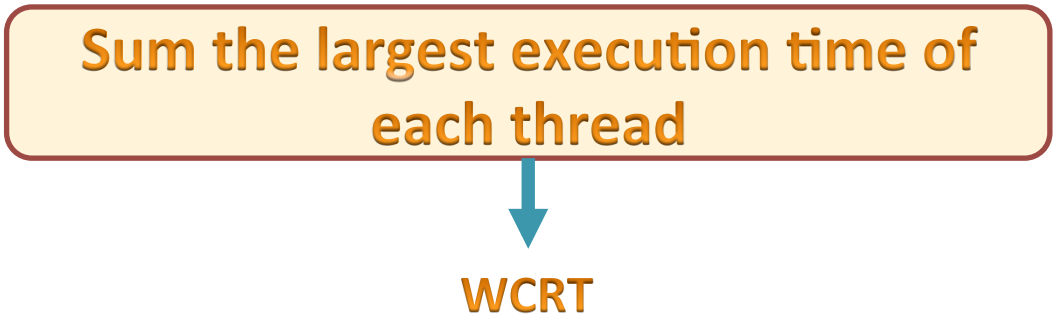
A3+B2+C2 = 33

WCRT = Max(25,28,35,23,30,33)

= 35 cycles

Conventional approaches

- Plus-max

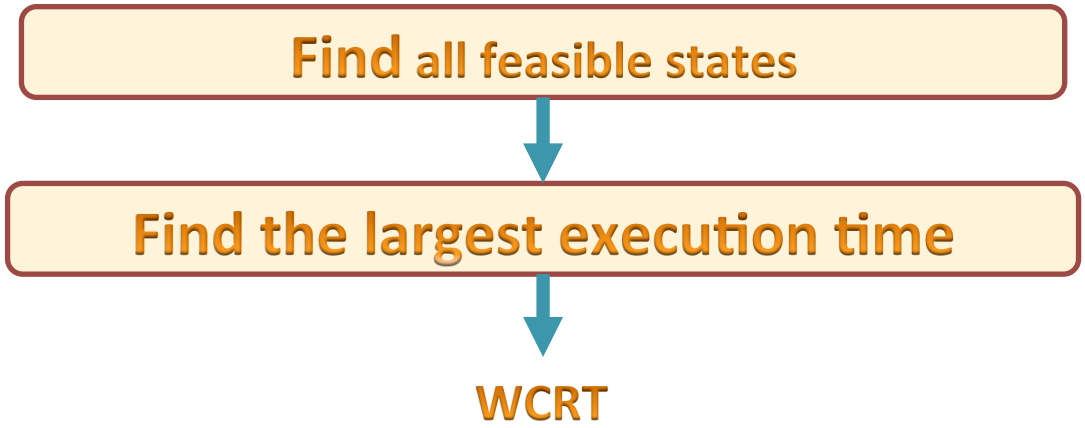


M. Boldt and C. Traulsen and R. Hanxleden. *Worst Case Reaction Time Analysis of Concurrent Reactive Programs. ENTCS, 203(4), 2008.*

L. Ju, B. K. Huynh, A. Roychoudhury, and S. Chakraborty. *Performance debugging of Esterel specifications*. CODES-ISSS 2008.

Conventional approaches

- State exploration



L. Ju, B. K. Huynh, S. Chakraborty, and A. Roychoudhury. *Context-sensitive timing analysis of Esterel programs. DAC,*

ILPs

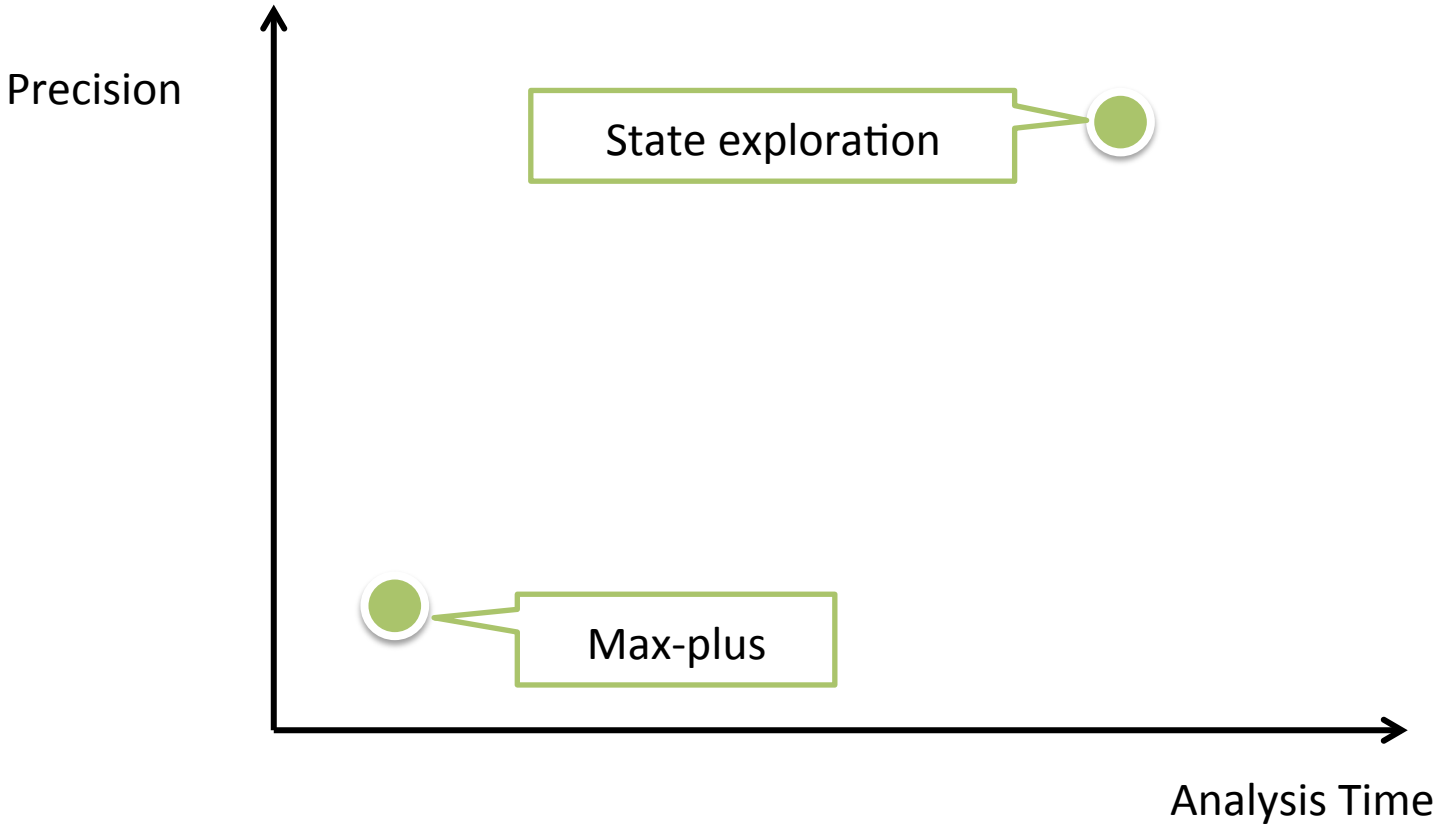
S. Andalam, P. S. Roop, and A. Girault. *Pruning infeasible paths for tight WCRT analysis of synchronous programs. DATE,*

Model checking

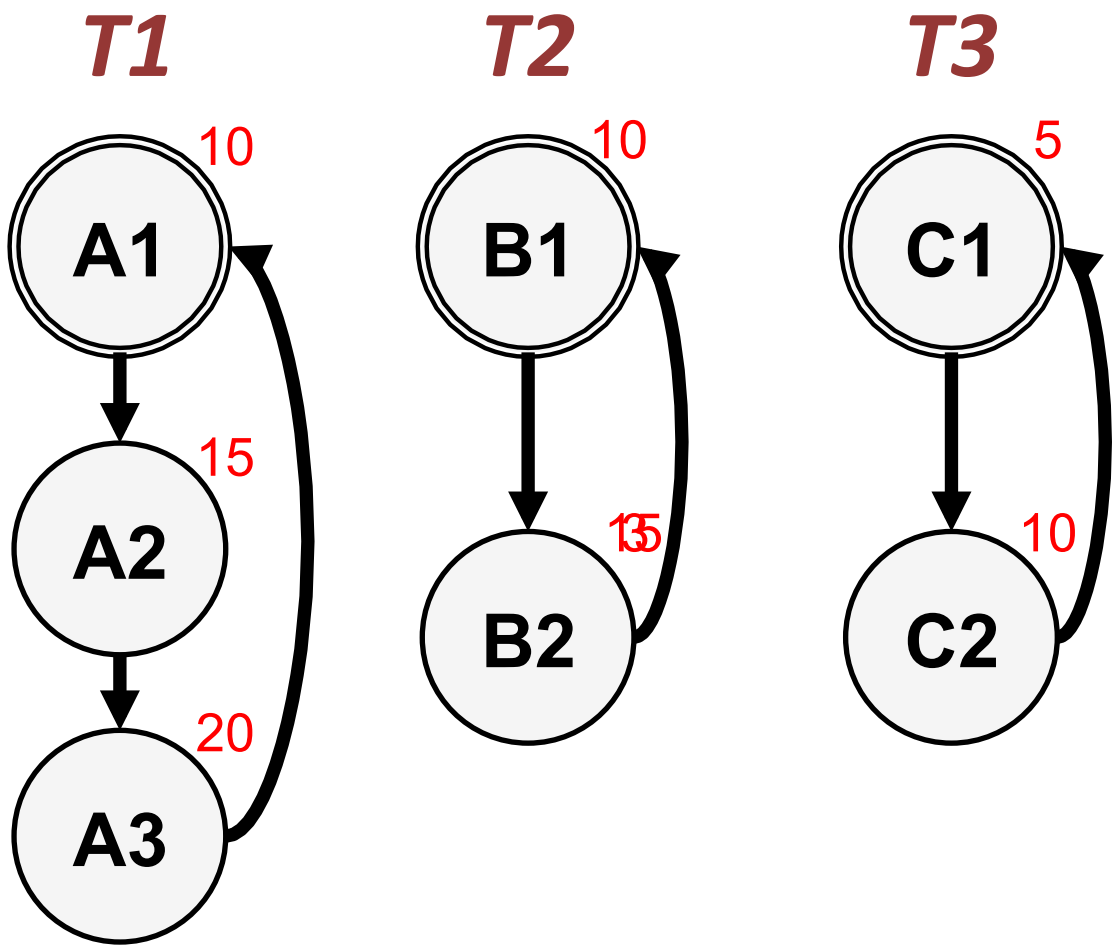
M. Kuo, R. Sinha, and P. S. Roop. *Efficient WCRT analysis of synchronous programs using reachability. DAC,*

Reachability

Trade-off

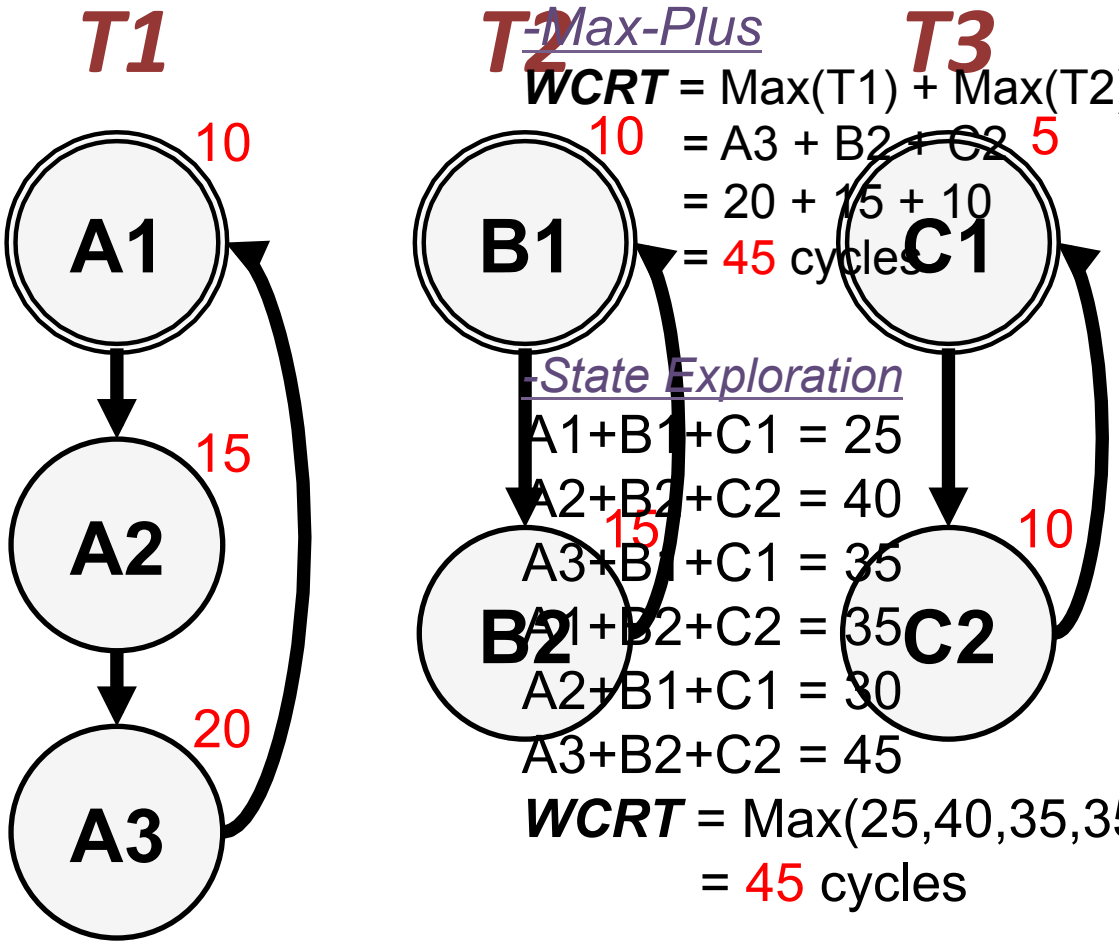


Motivating example



Motivating example

WCRT analysis



-Max-Plus

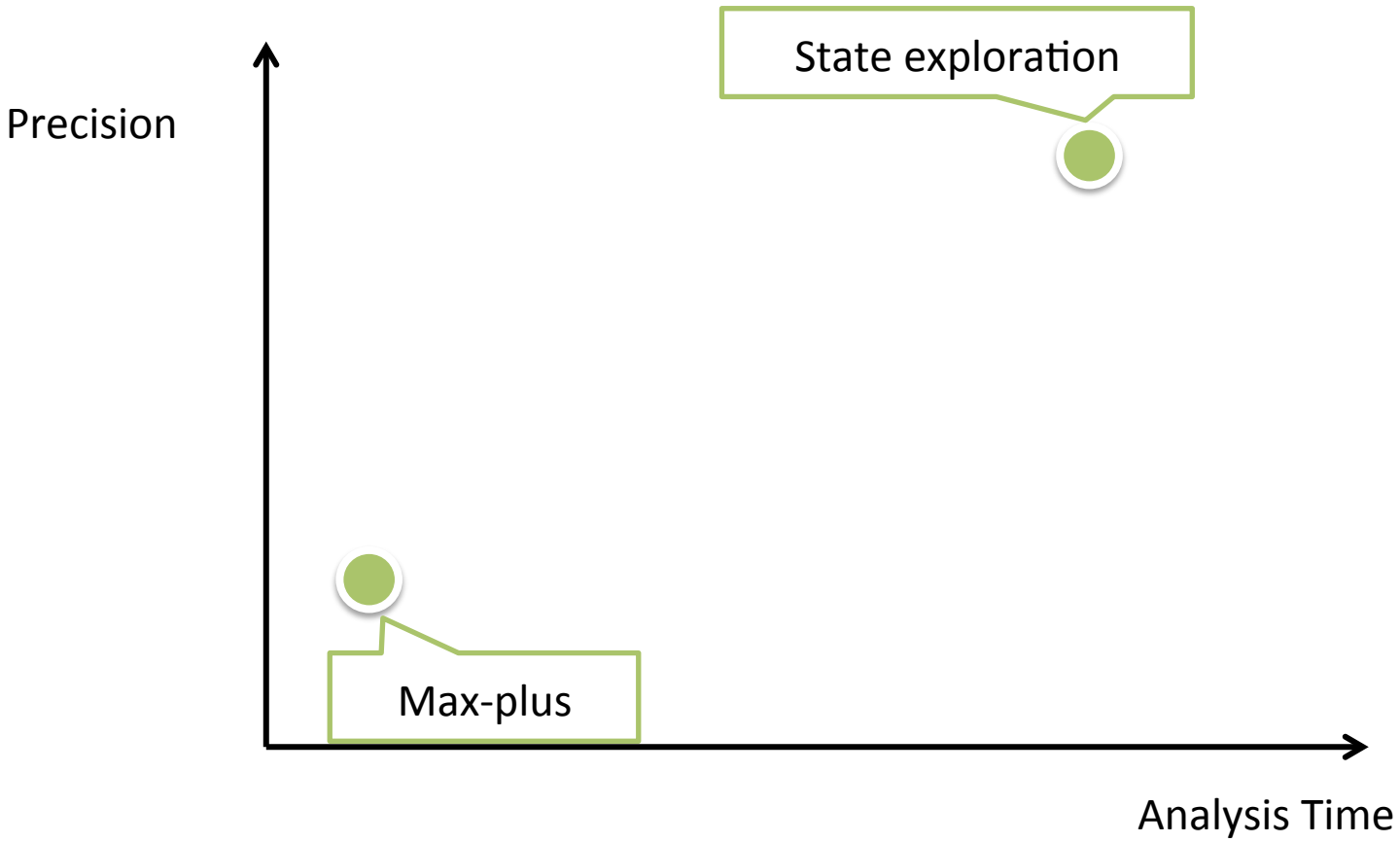
WCRT = Max(T1) + Max(T2) + Max(T3)
= A3 + B2 + C2
= 20 + 15 + 10
= 45 cycles

-State Exploration

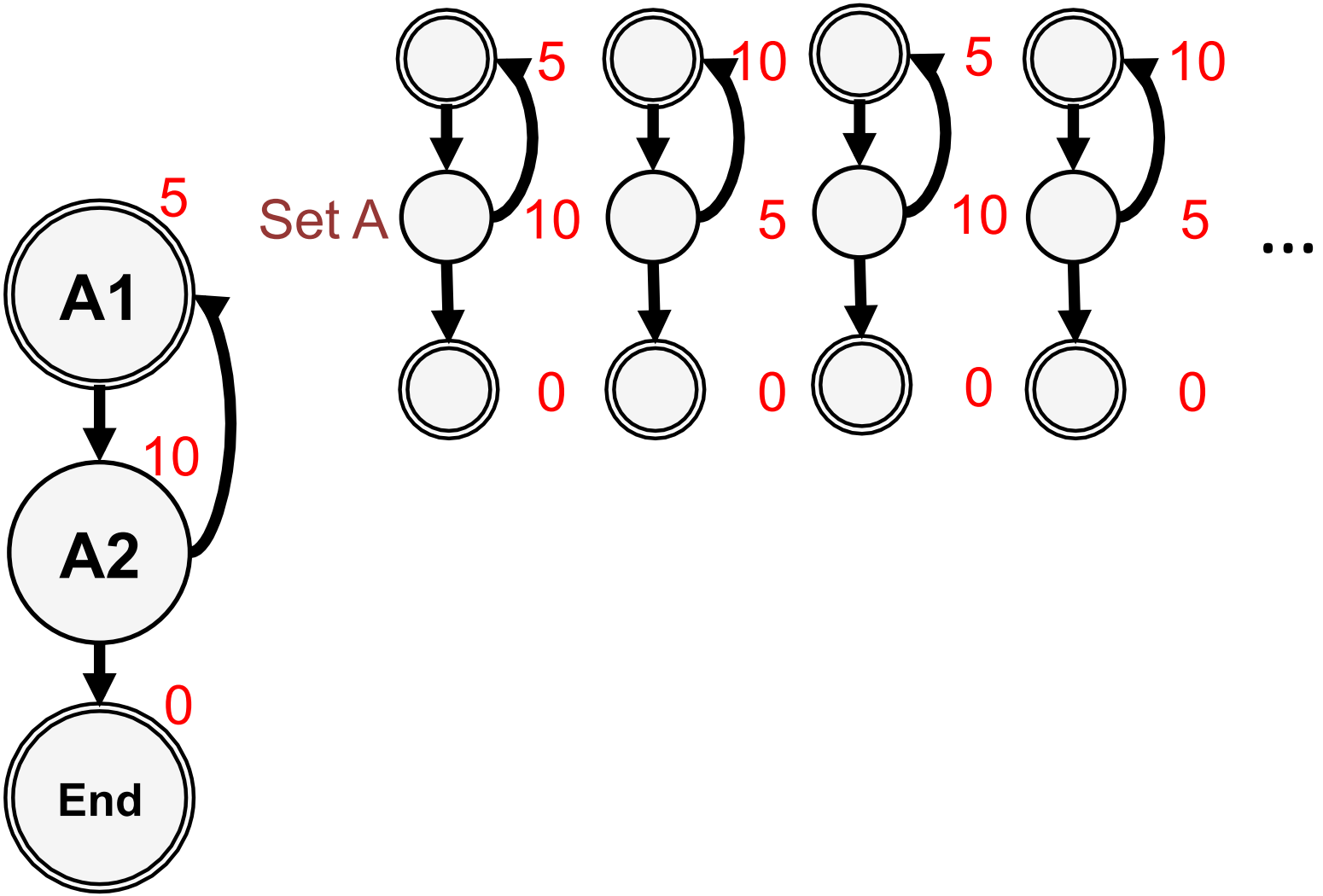
A1+B1+C1 = 25
A2+B2+C2 = 40
A3+B1+C1 = 35
A1+B2+C2 = 35
A2+B1+C1 = 30
A3+B2+C2 = 45

WCRT = Max(25,40,35,35,30,45)
= 45 cycles

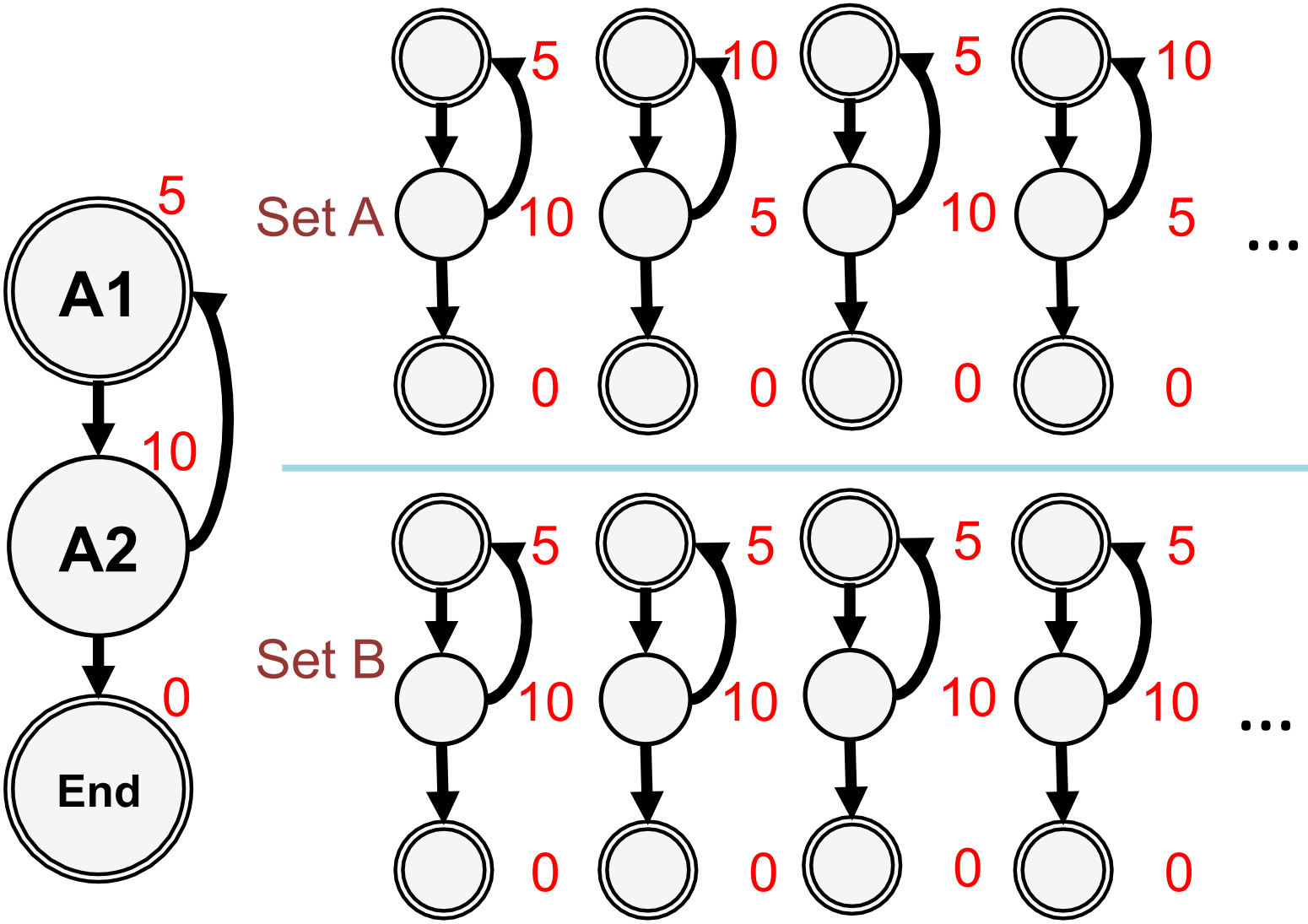
Trade-off



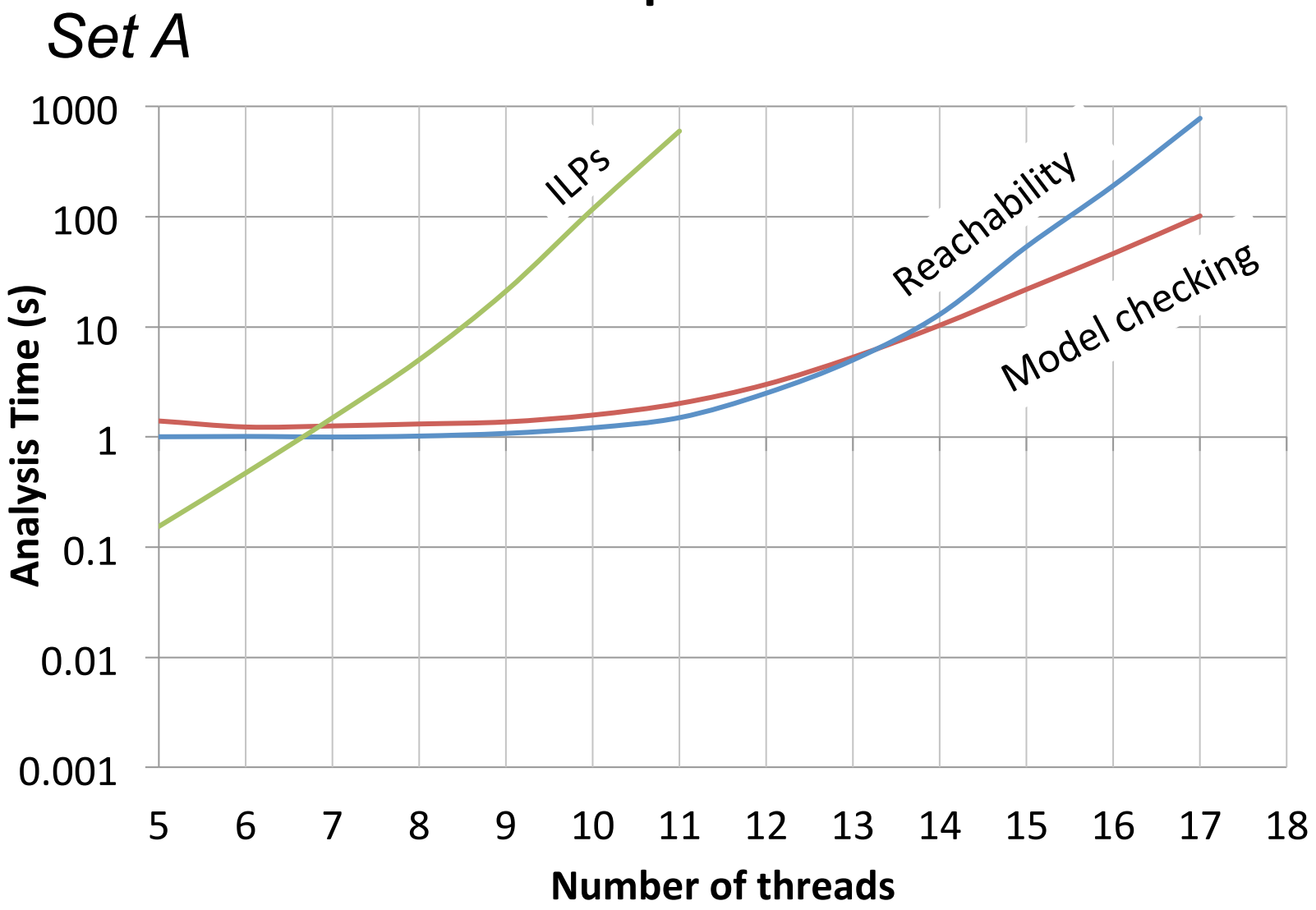
Synthetic examples



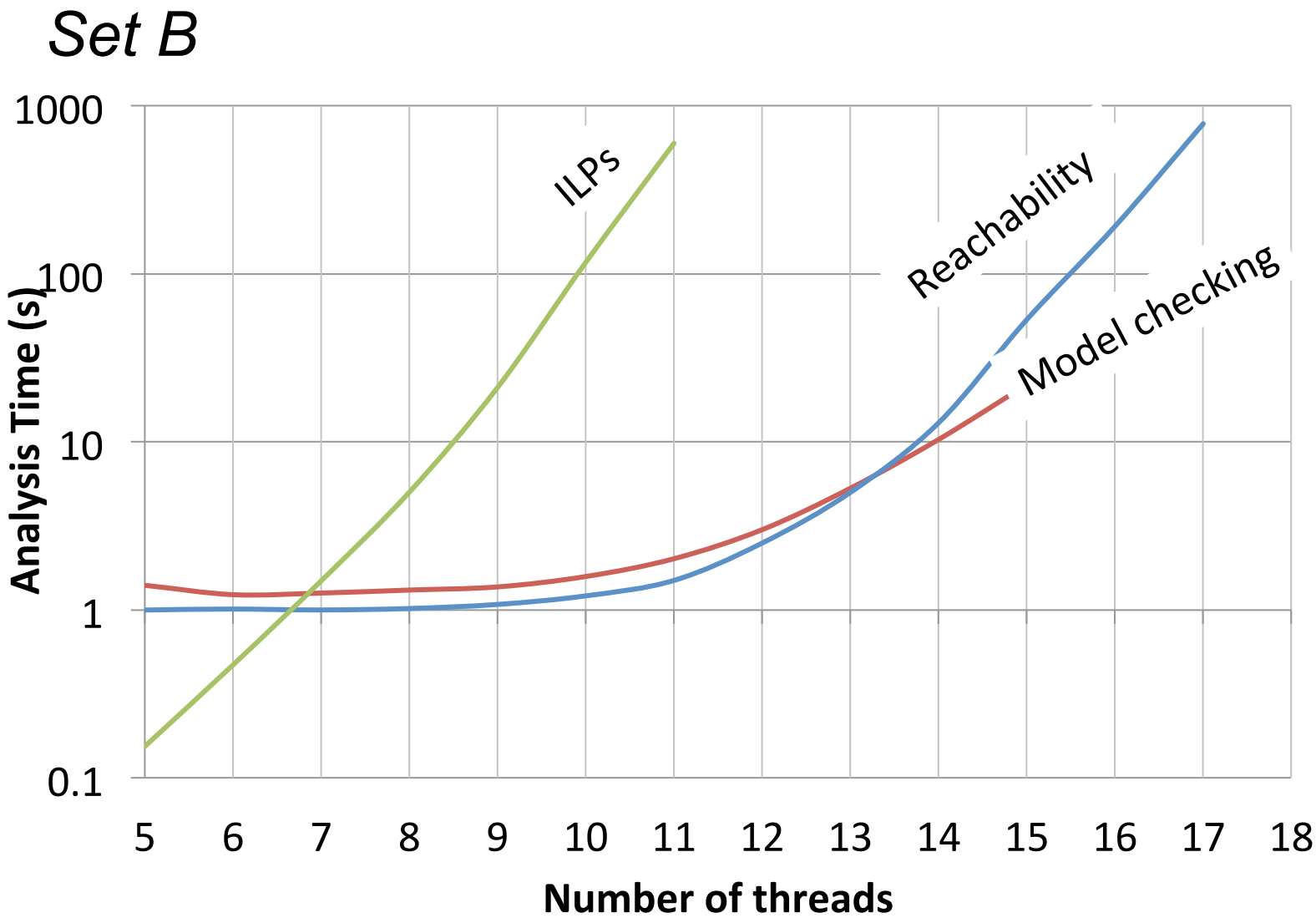
Two sets


5
A1
10
A2
0
End
Set A
5
10
0
5
10
0
5
10
0
5
10
0
...
Set B
5
10
0
5
10
0
5
10
0
5
10
0
...

Comparing state-exploration techniques



Comparison contd.



Observations

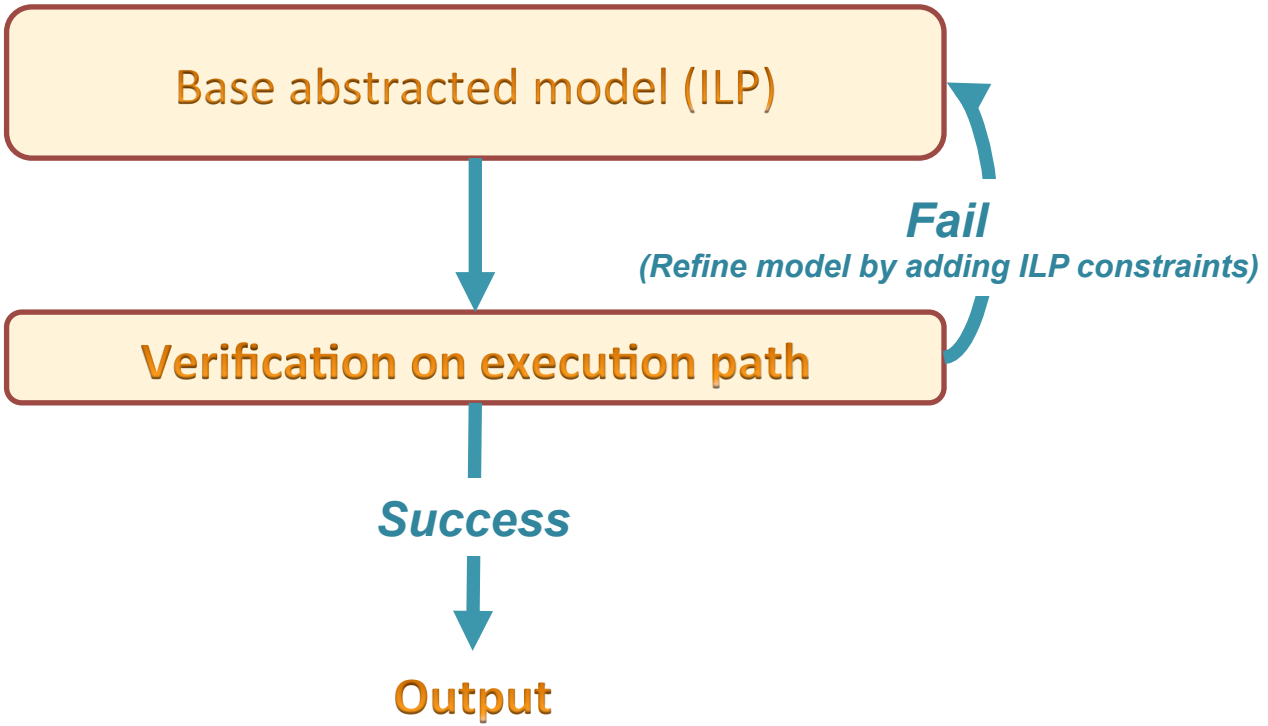
- Longer analysis time does not necessarily translate into better precision.
- Time Complexity: **EXPONENTIAL**

$$\Theta(n\, lcm(\phi_1, \ldots, \phi_n)).$$

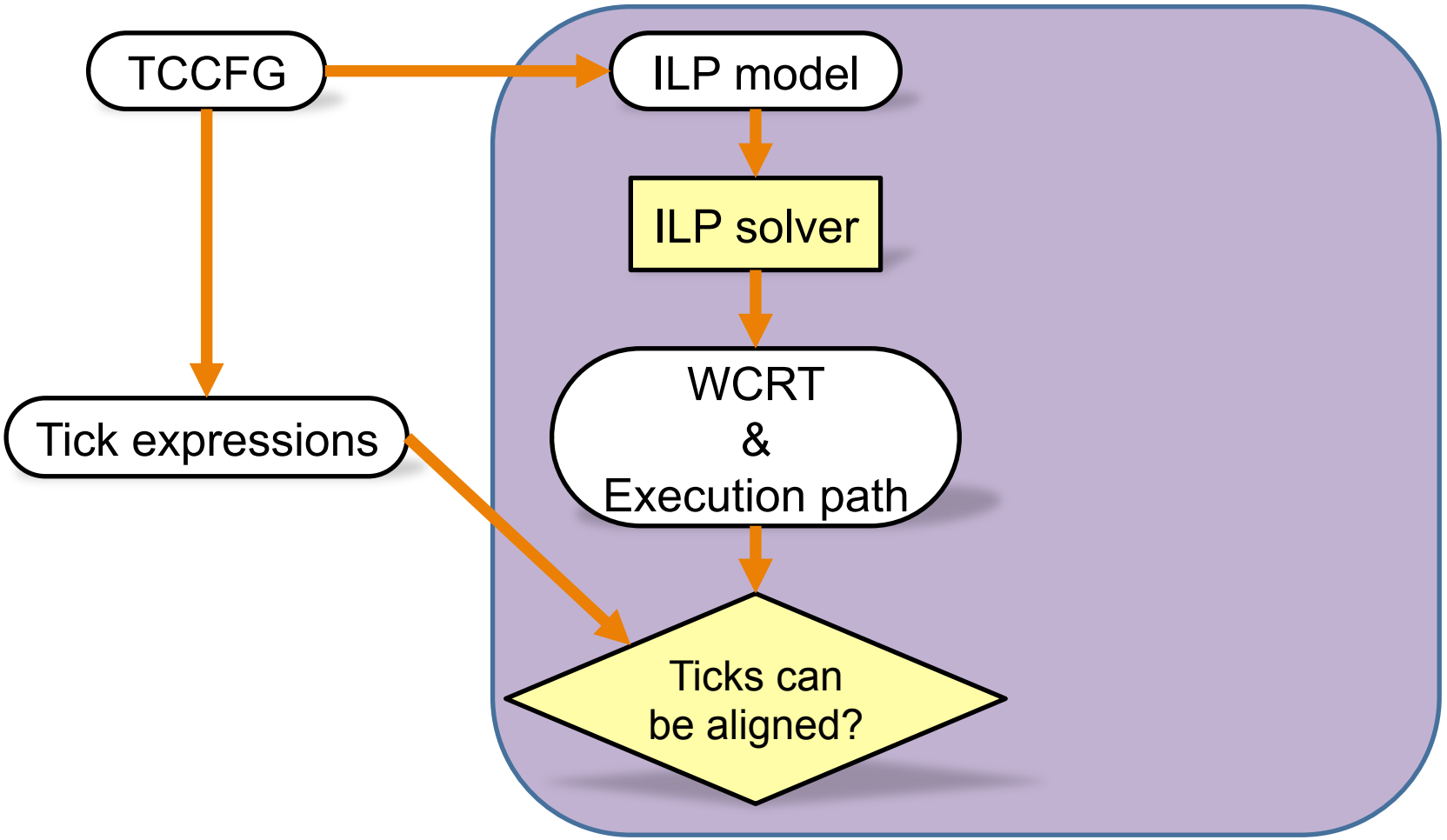
- Similar problem exists for model checking. However, there are algorithms such as CEGAR [Clarke et al, JACM'03] that work well in practice.
- Refine to remove spurious counter examples.

Proposal

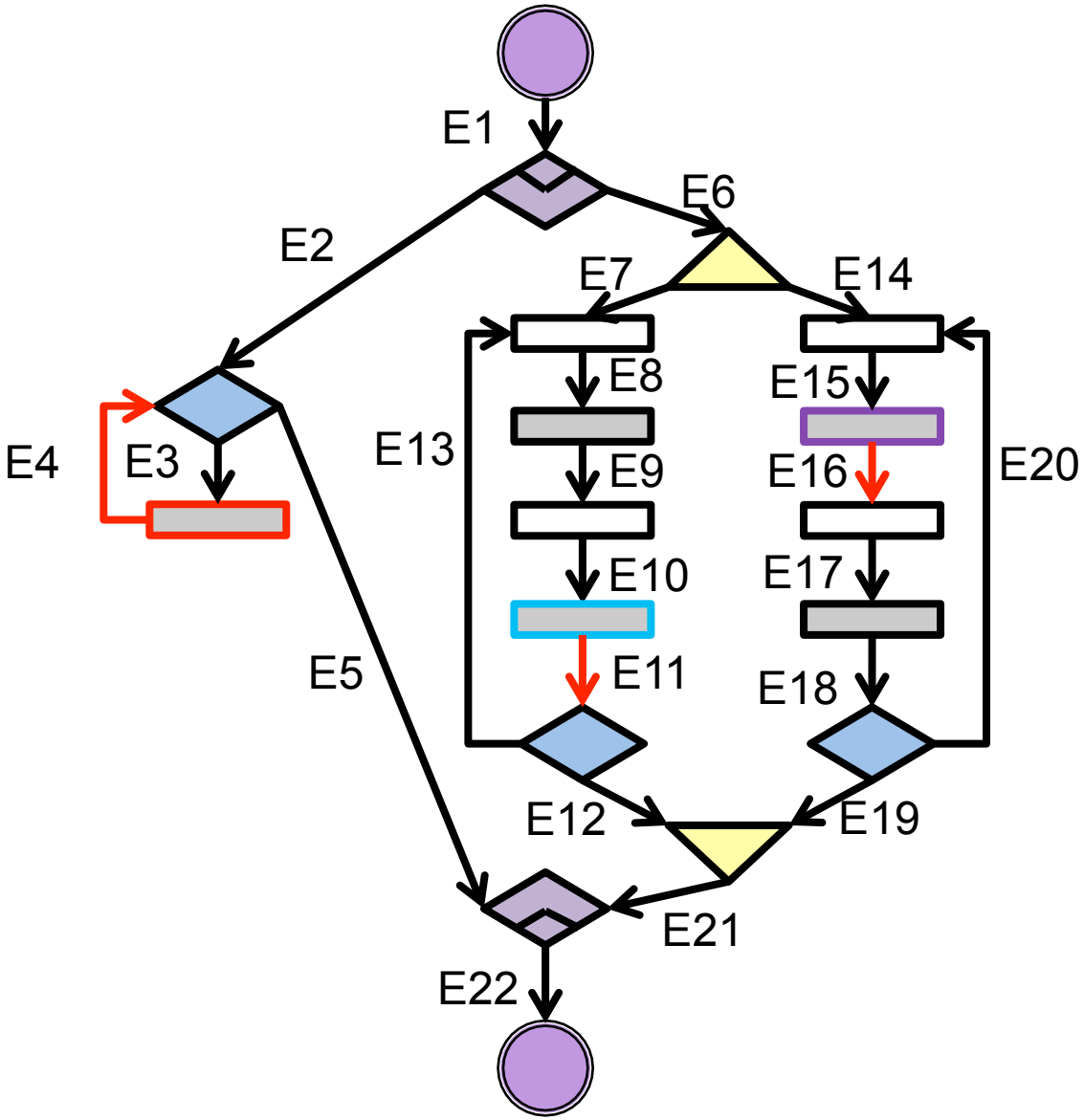
- A scalable and extendable timing analysis framework (ILPc)



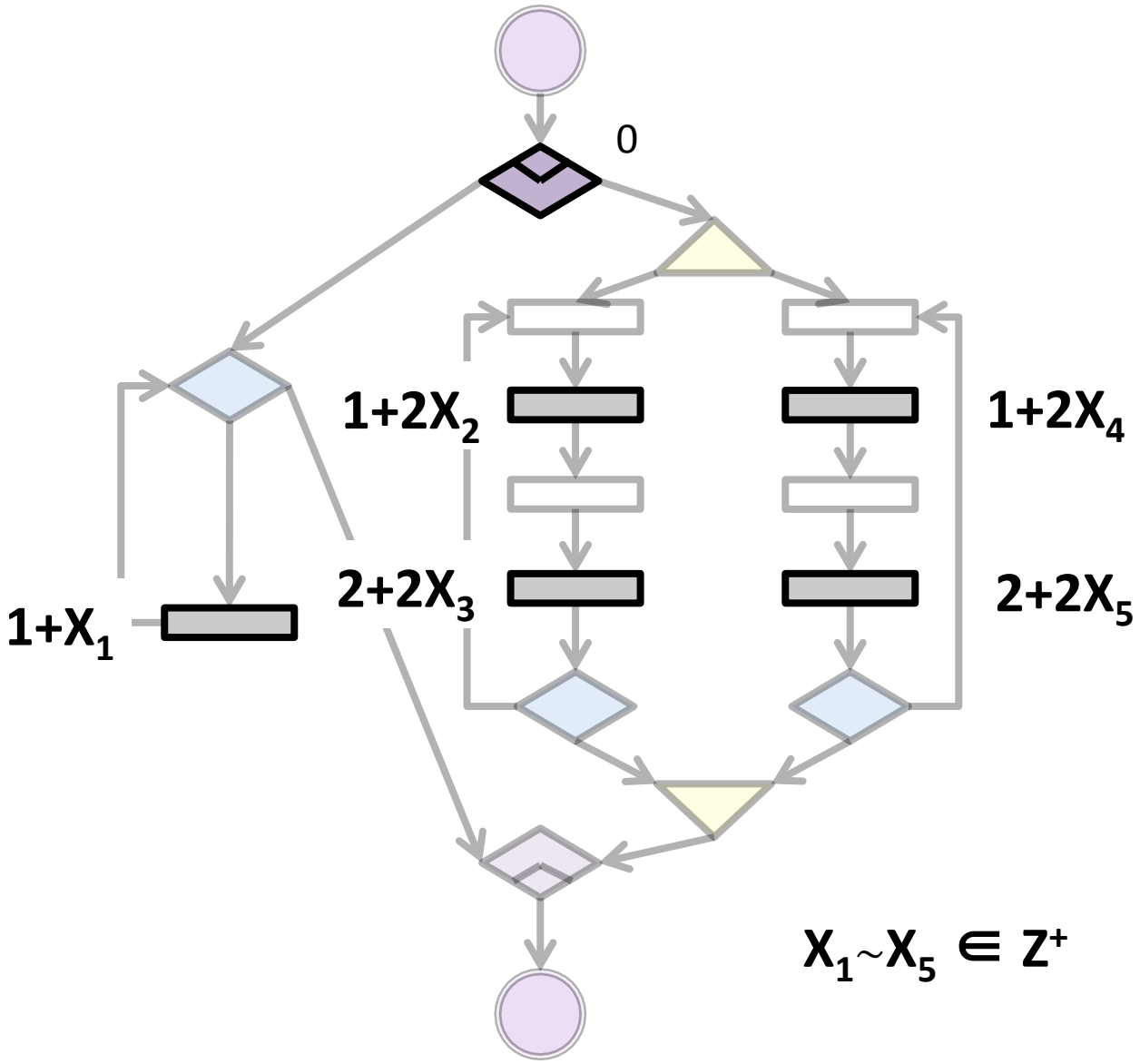
Overview of ILPc



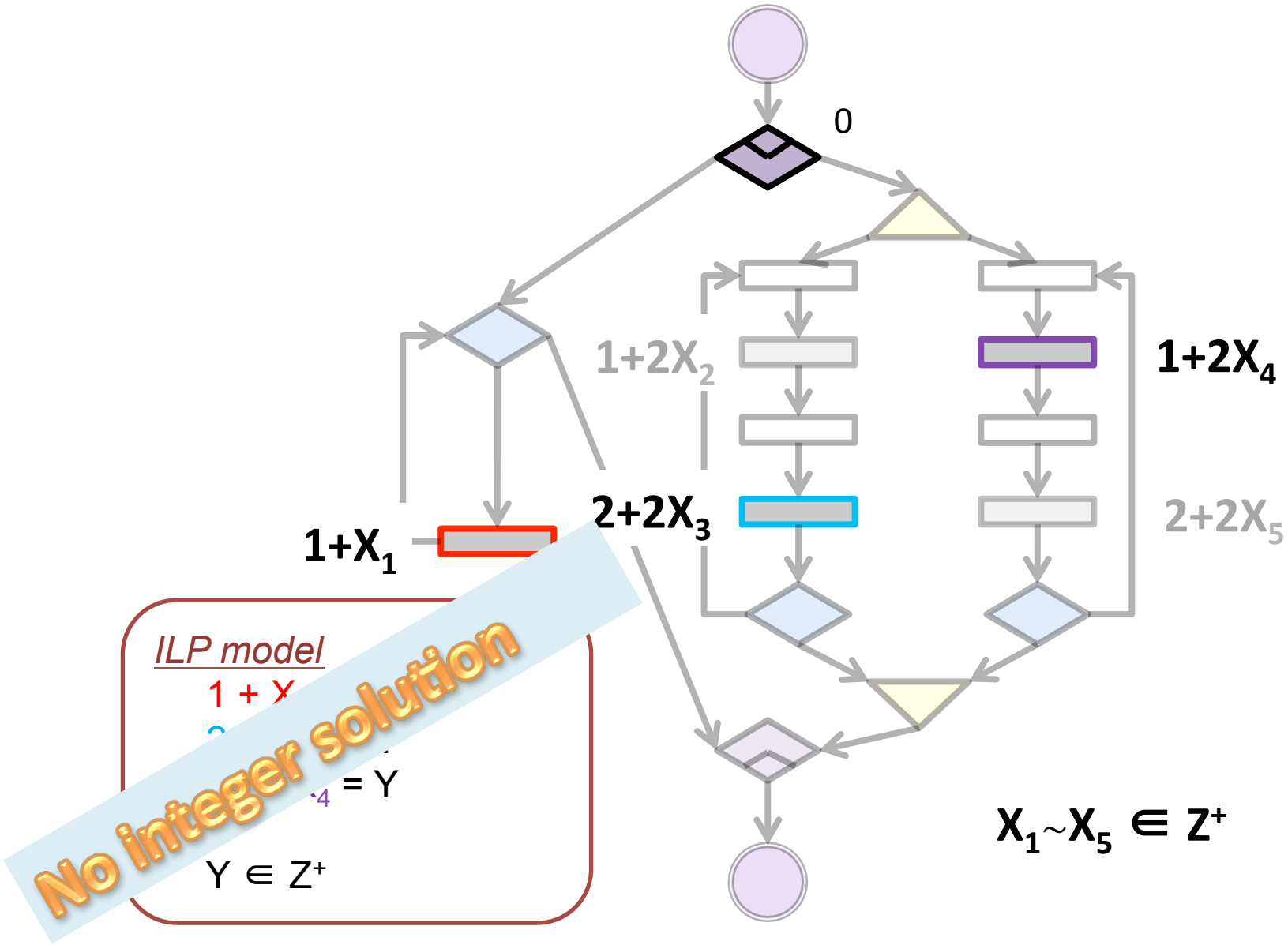
Base ILP model produces a path



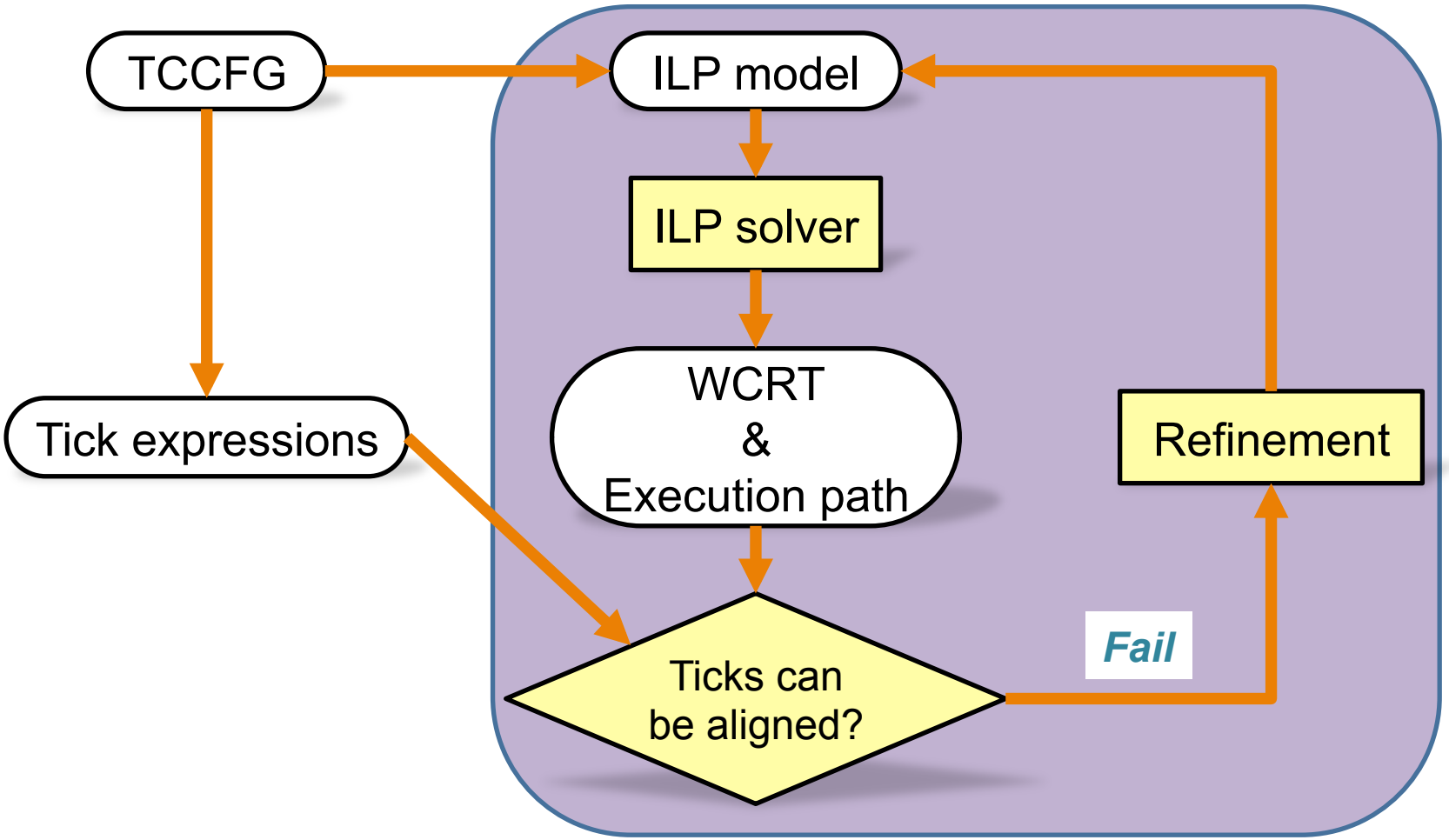
Verifying tick alignment



Verifying tick alignment

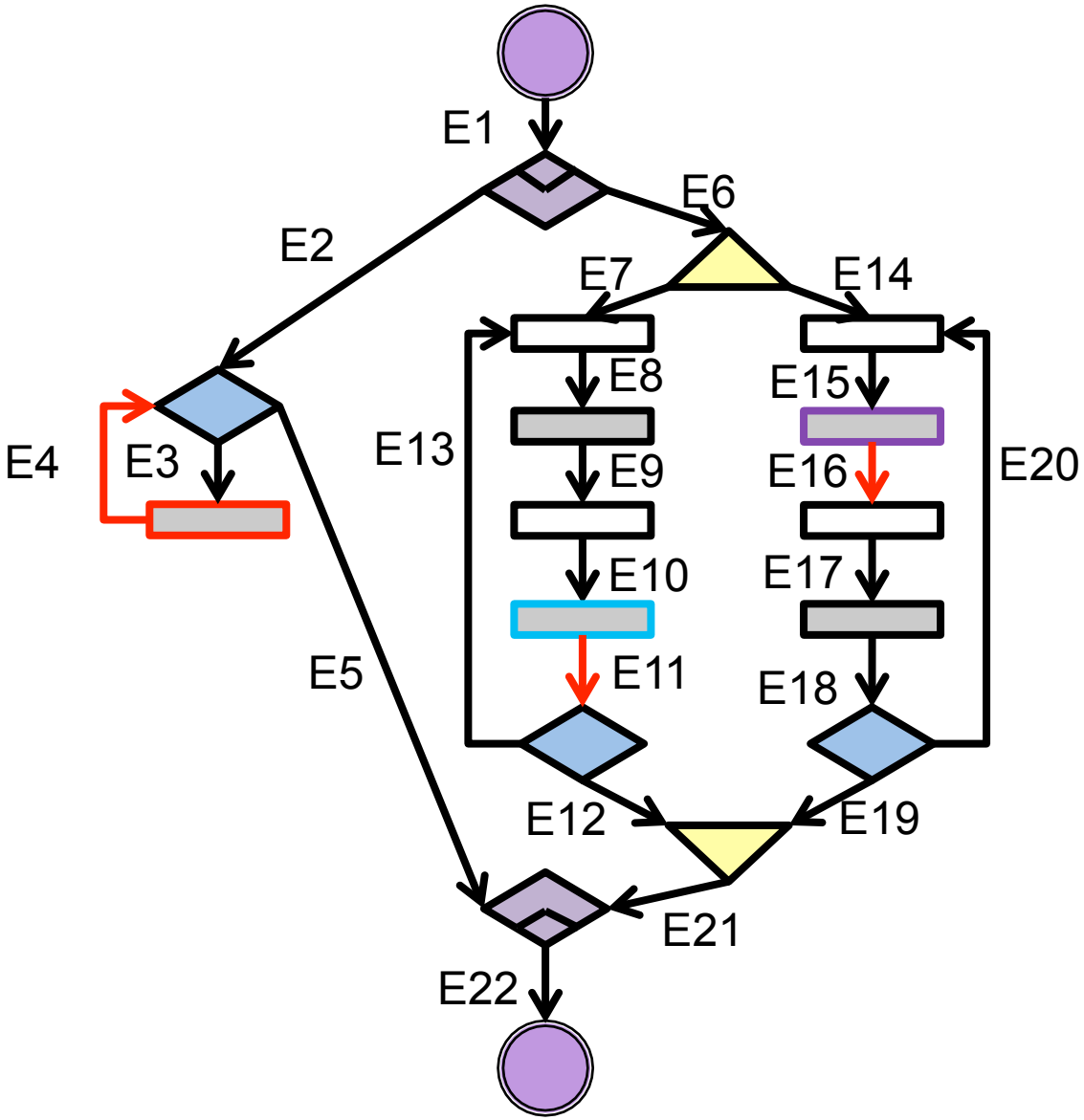


Overview of ILPc

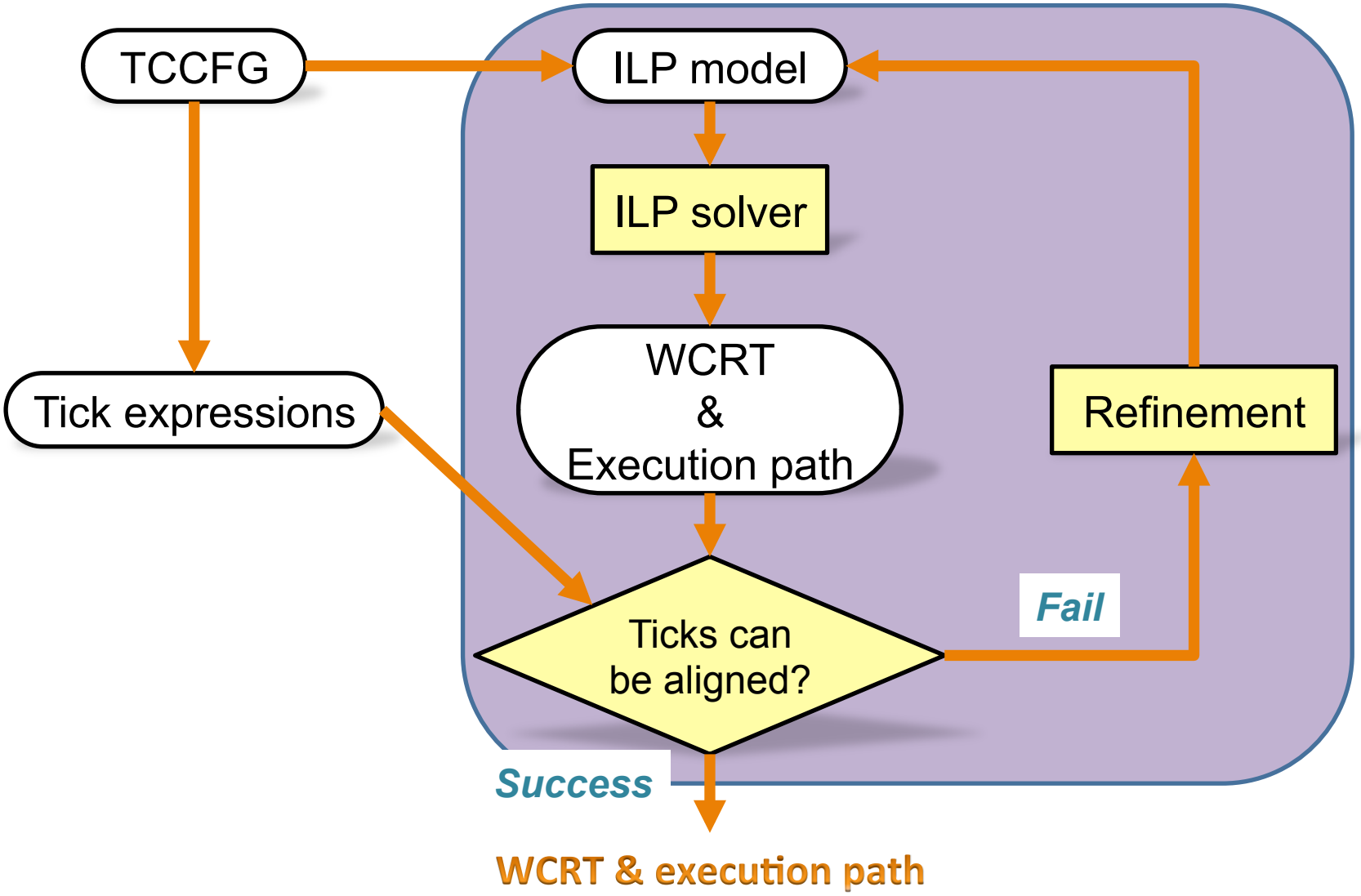


Refinement

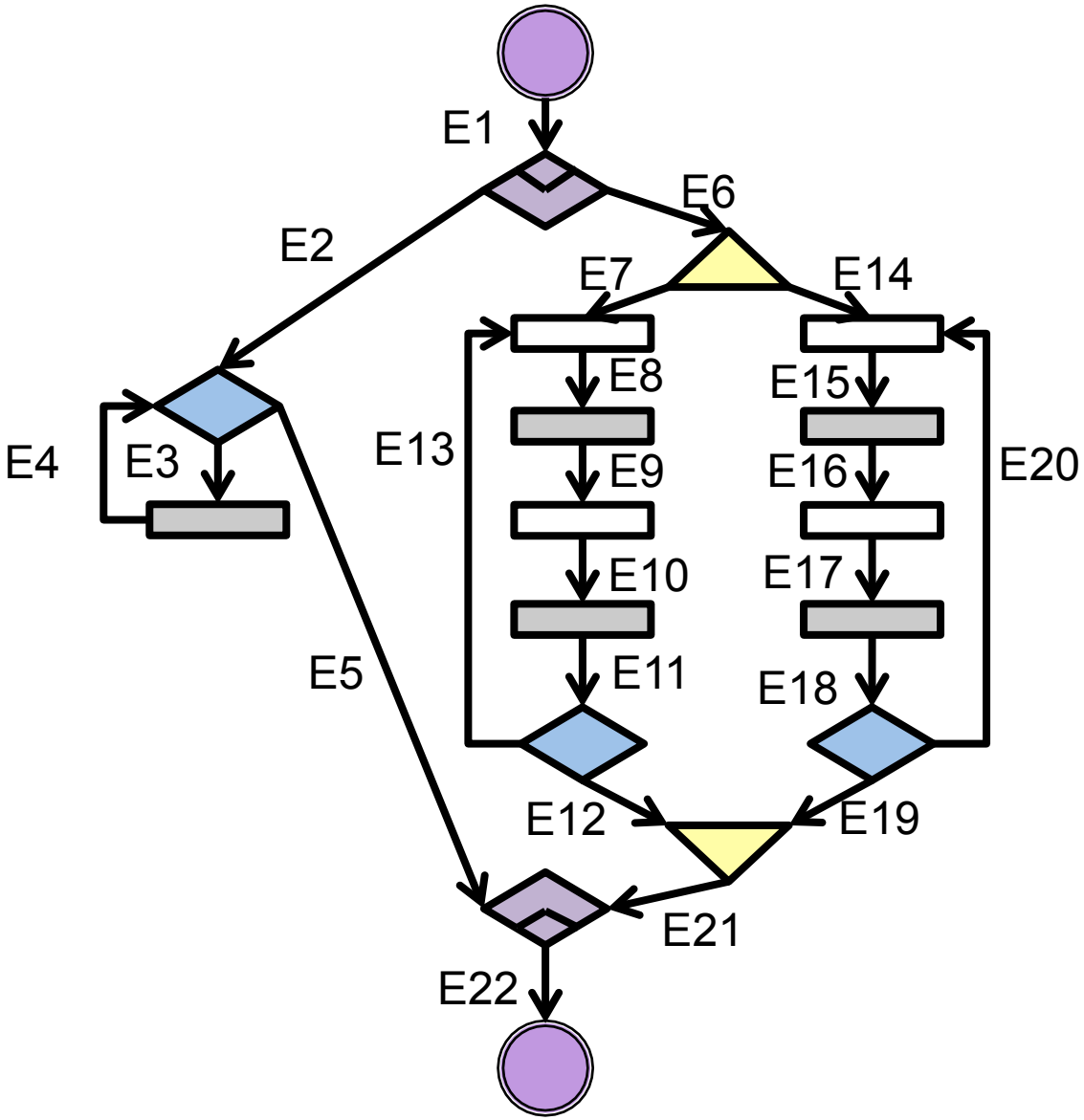
E4 + E11 + E16 ≤ 2



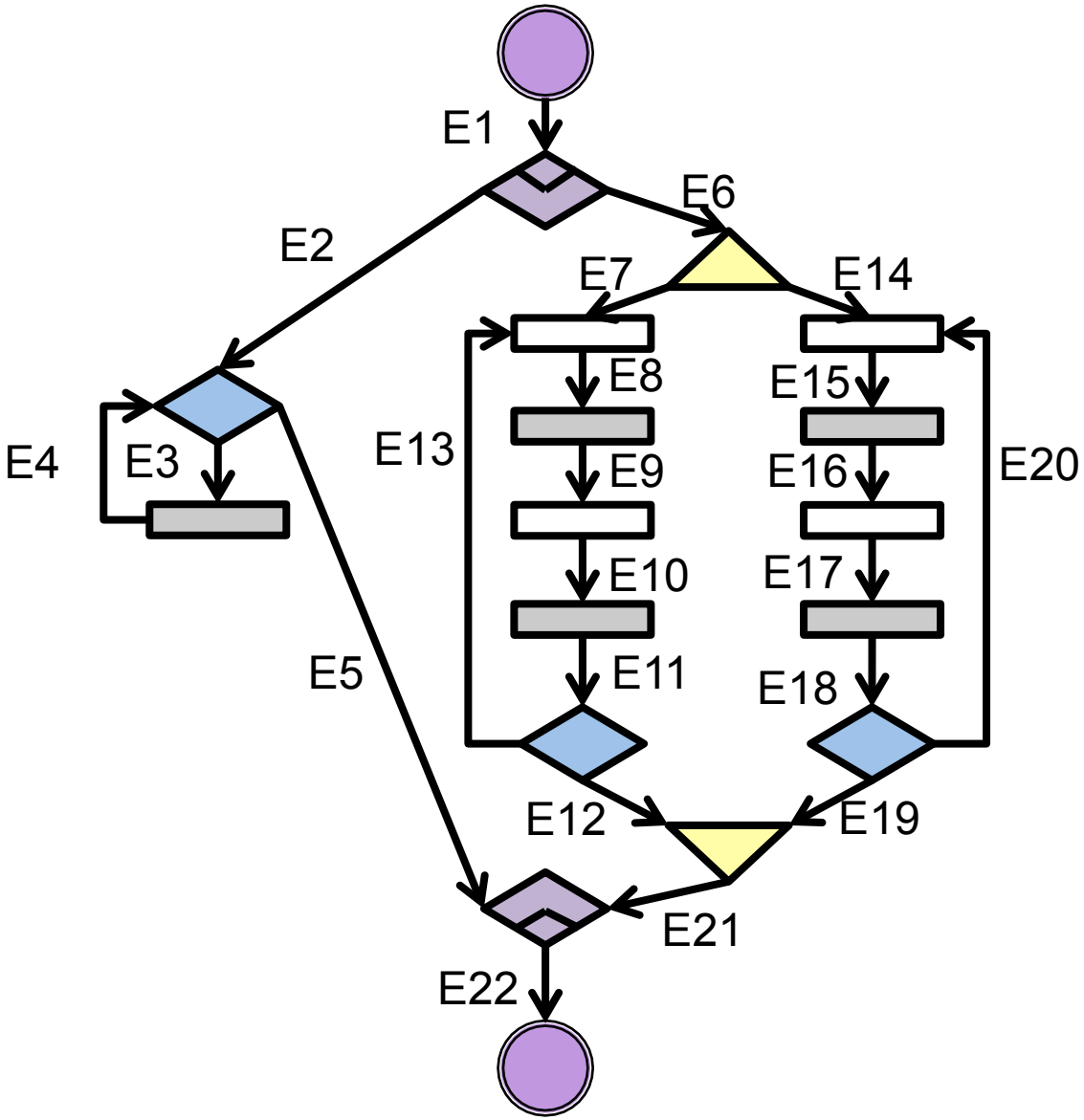
Overview of ILPc



ILP model


E1
E2
E3
E4
E5
E6
E7
E8
E9
E10
E11
E12
E13
E14
E15
E16
E17
E18
E19
E20
E21
E22

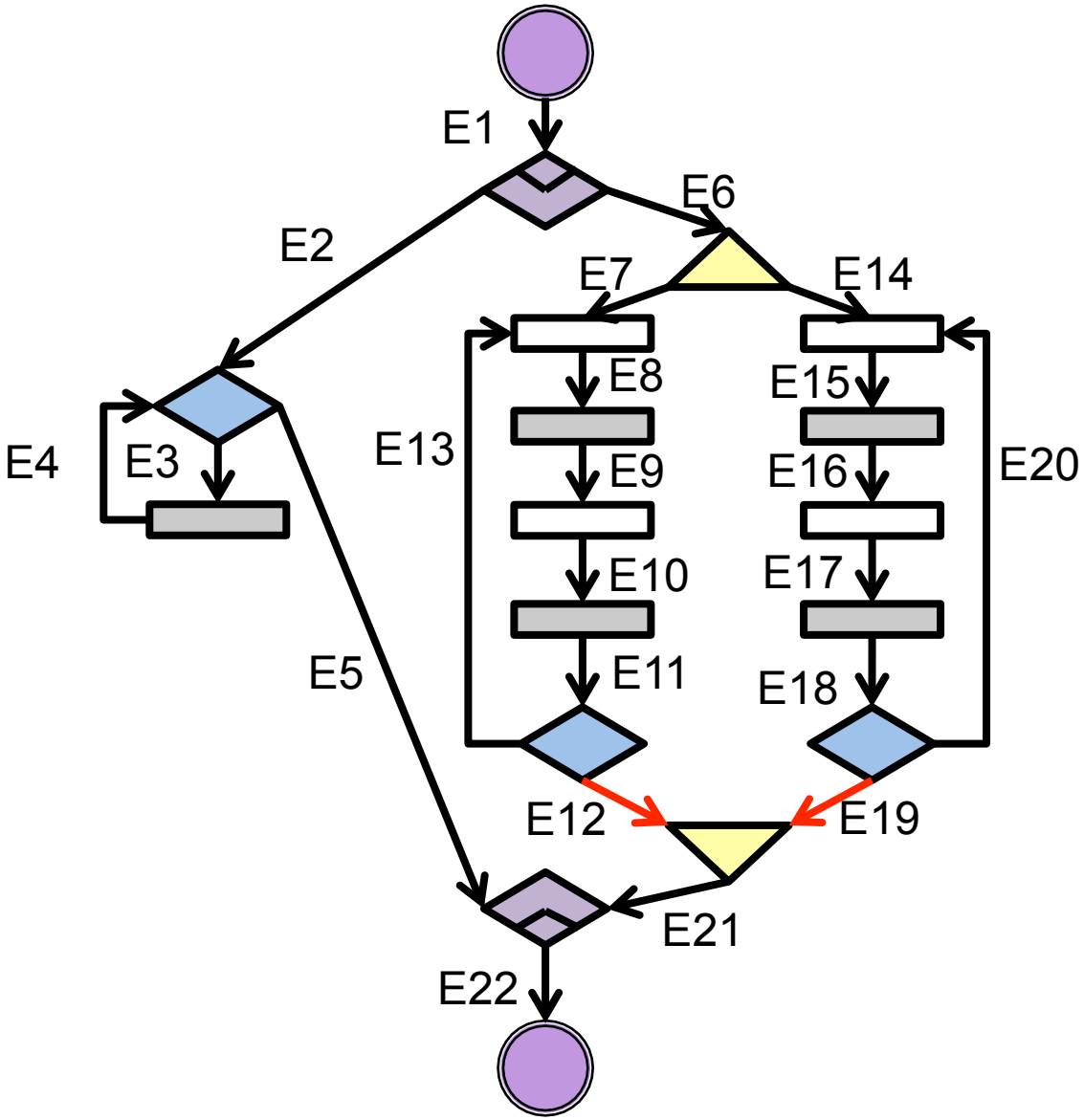
ILP model



Objective function:

$$obj = \sum C_i \times E_i$$

ILP model

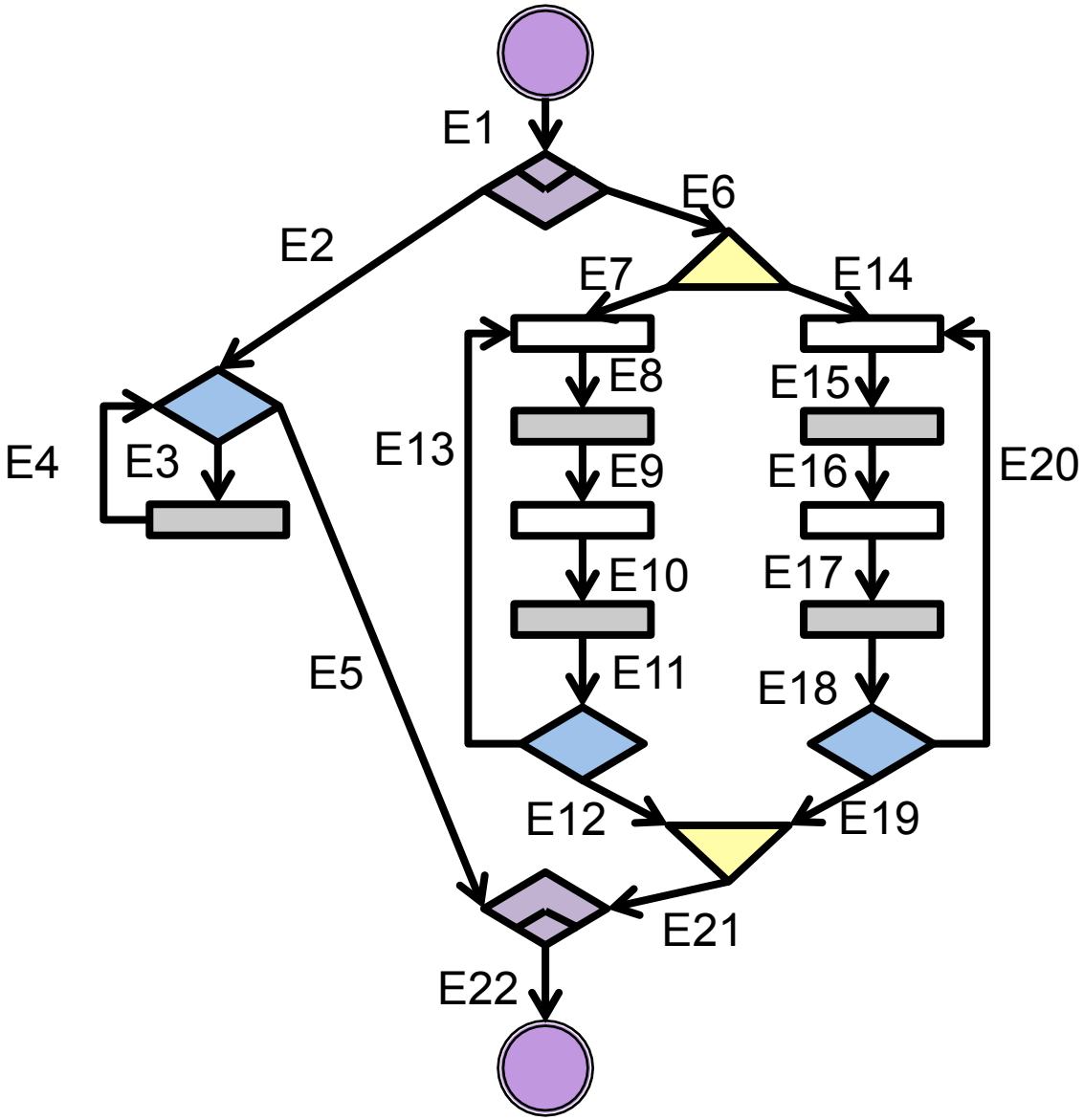


Objective function:

$$obj = \sum C_i \times E_i$$

Execution cost occur upon exiting a node

ILP model

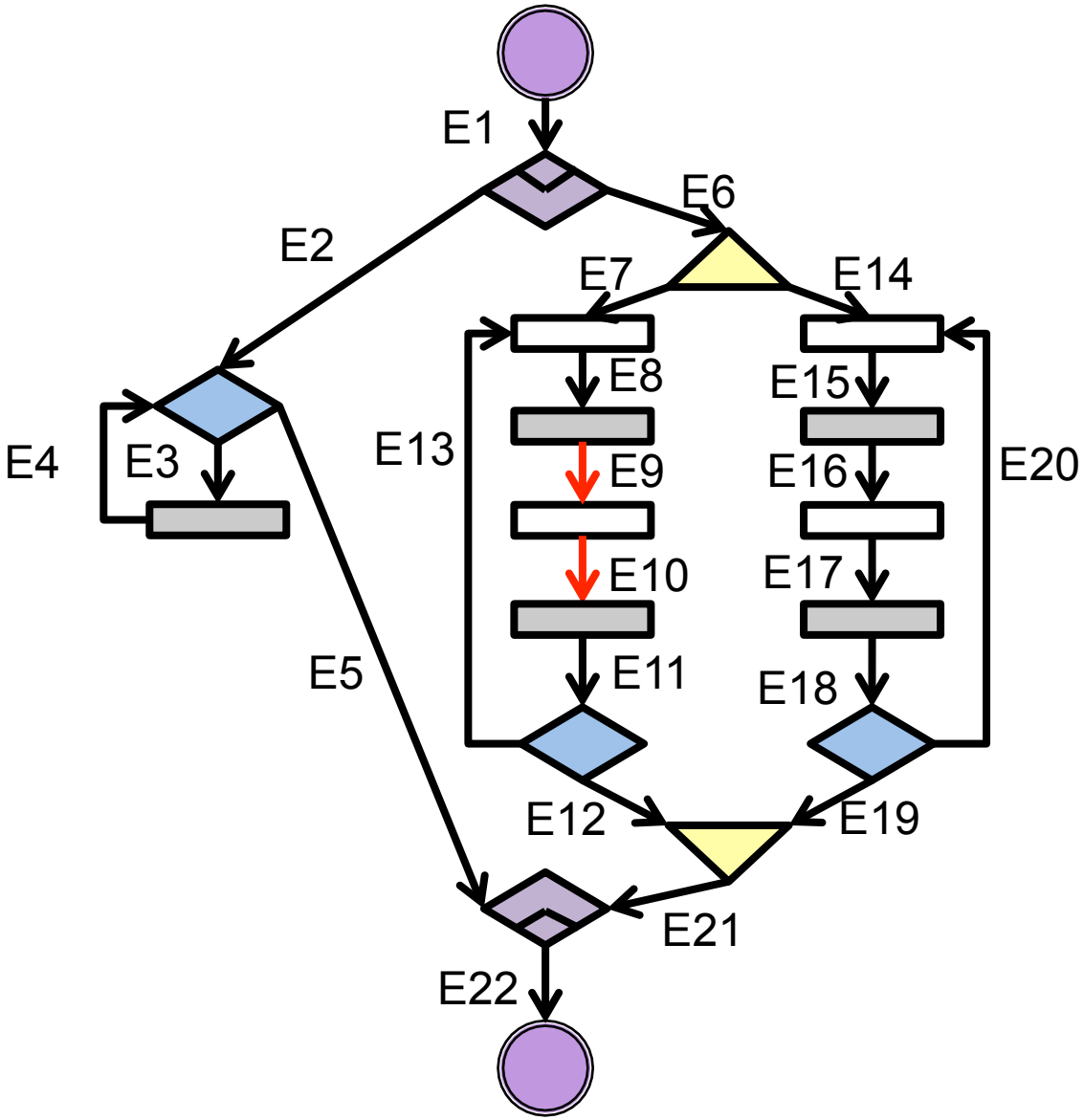


Objective function:

$$obj = \sum C_i \times E_i$$

Conventional nodes:

ILP model



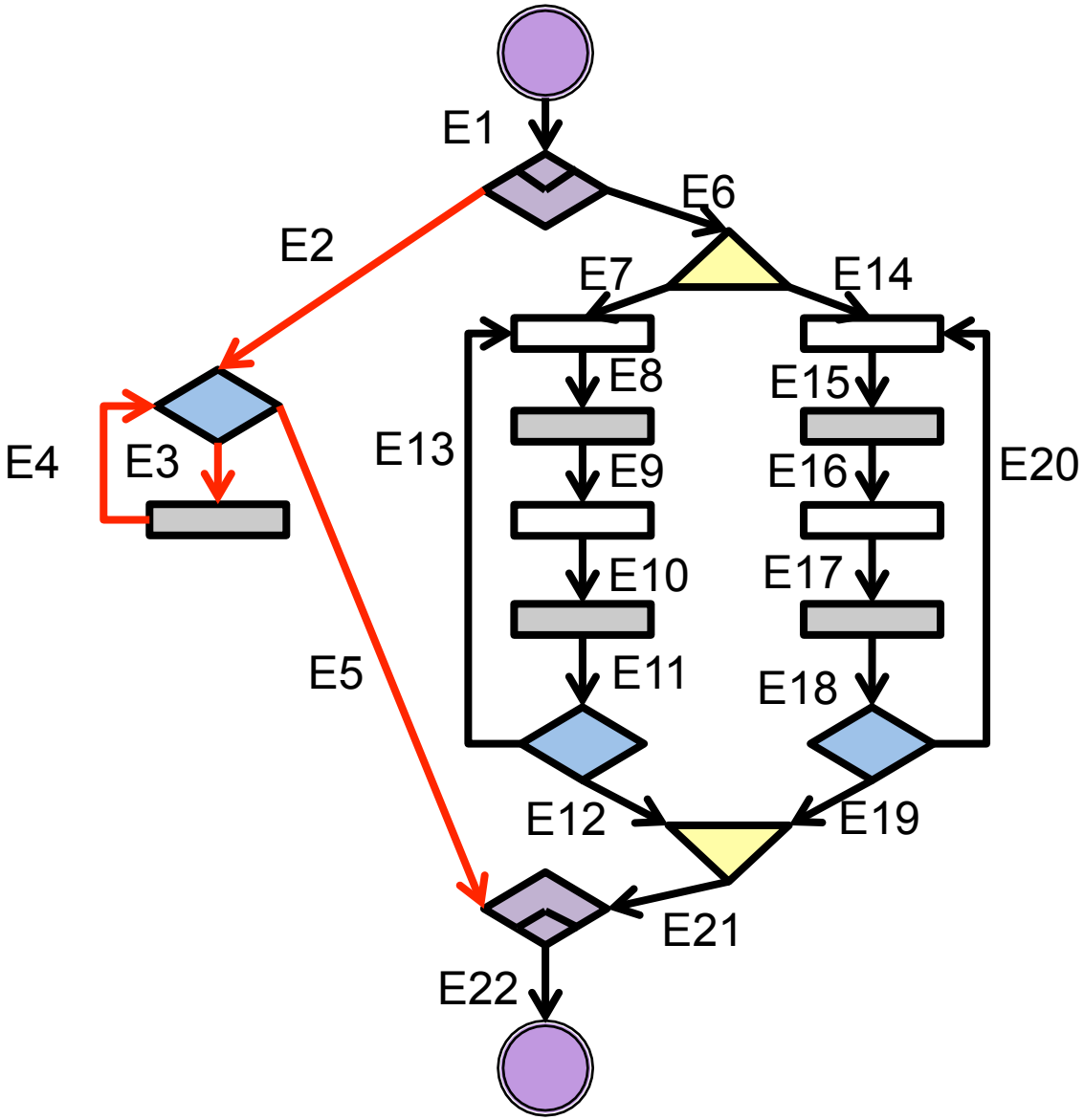
Objective function:

$$obj = \sum C_i \times E_i$$

Conventional nodes:

E9 = E10

ILP model



Objective function:

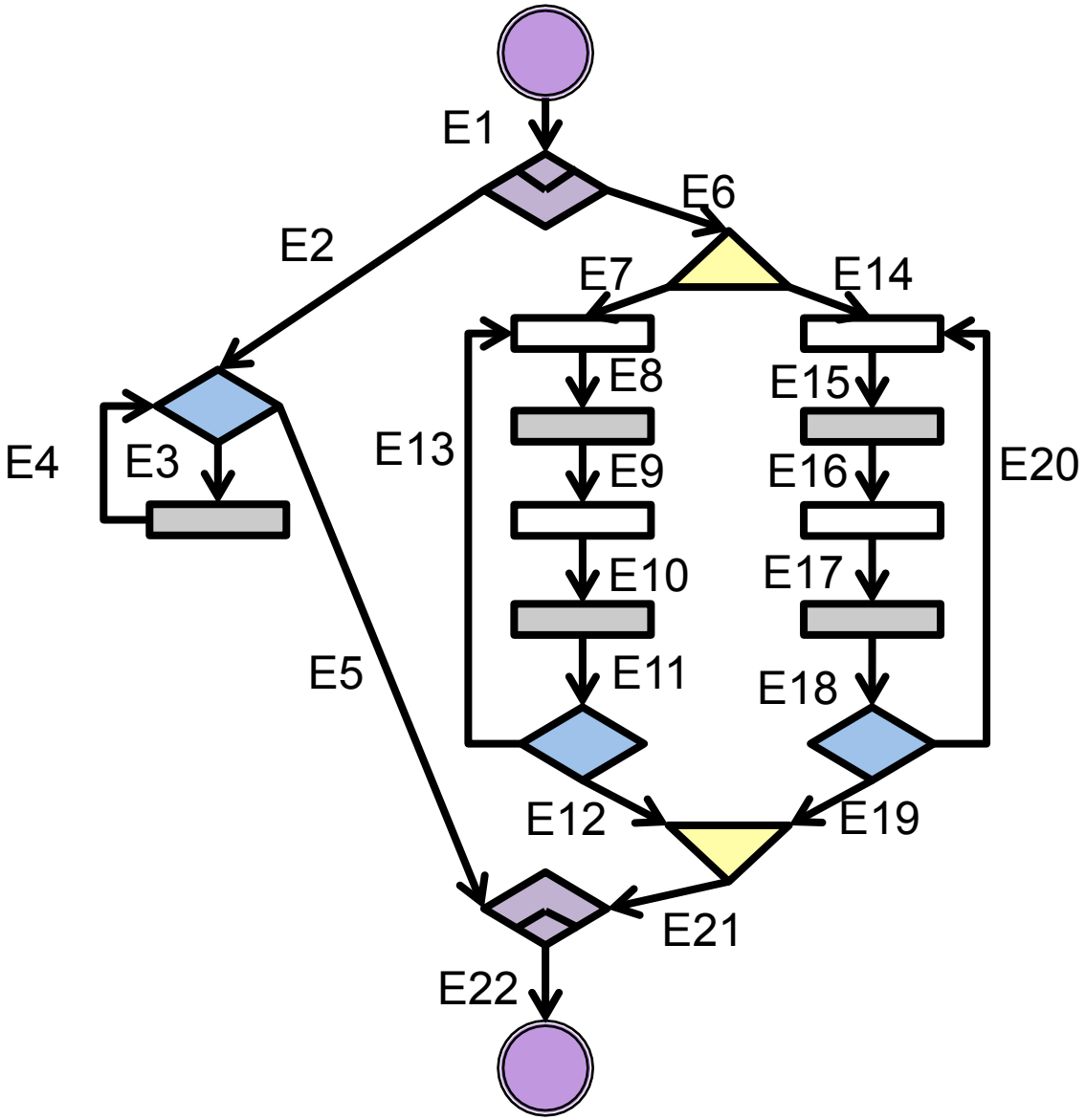
$$obj = \sum C_i \times E_i$$

Conventional nodes:

$$E9 = E10$$

$$E2 + E4 = E3 + E5$$

ILP model



Objective function:

$$obj = \sum C_i \times E_i$$

Conventional nodes:

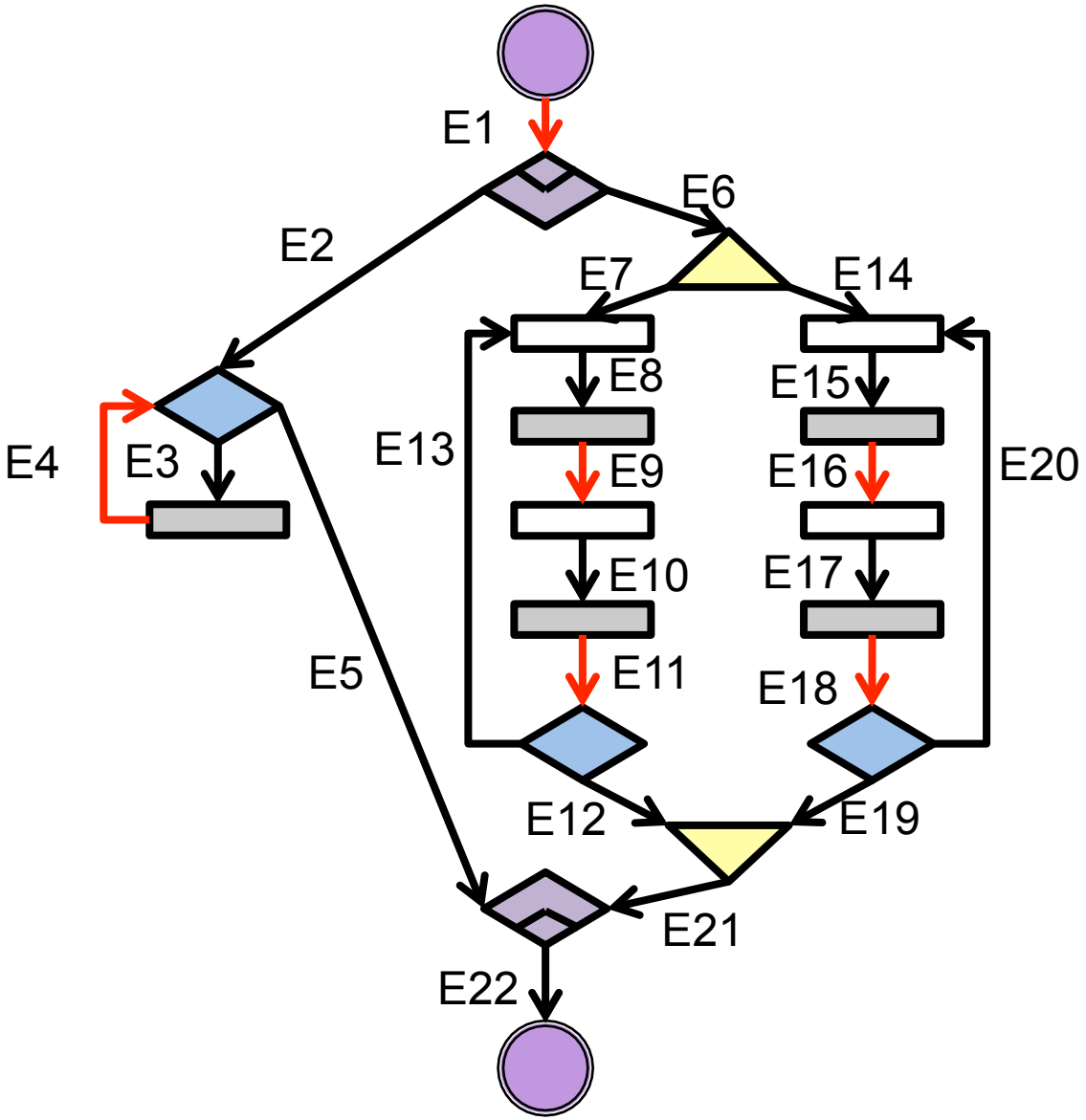
$$E9 = E10$$

$$E2 + E4 = E3 + E5$$

Features:

- $E_i \in \{0,1\}$
- Solver set E_i to 1 whenever is possible
- Inequalities

ILP model



Objective function:

$$obj = \sum C_i \times E_i$$

Conventional nodes:

$$E9 = E10$$

$$E2 + E4 = E3 + E5$$

Features:

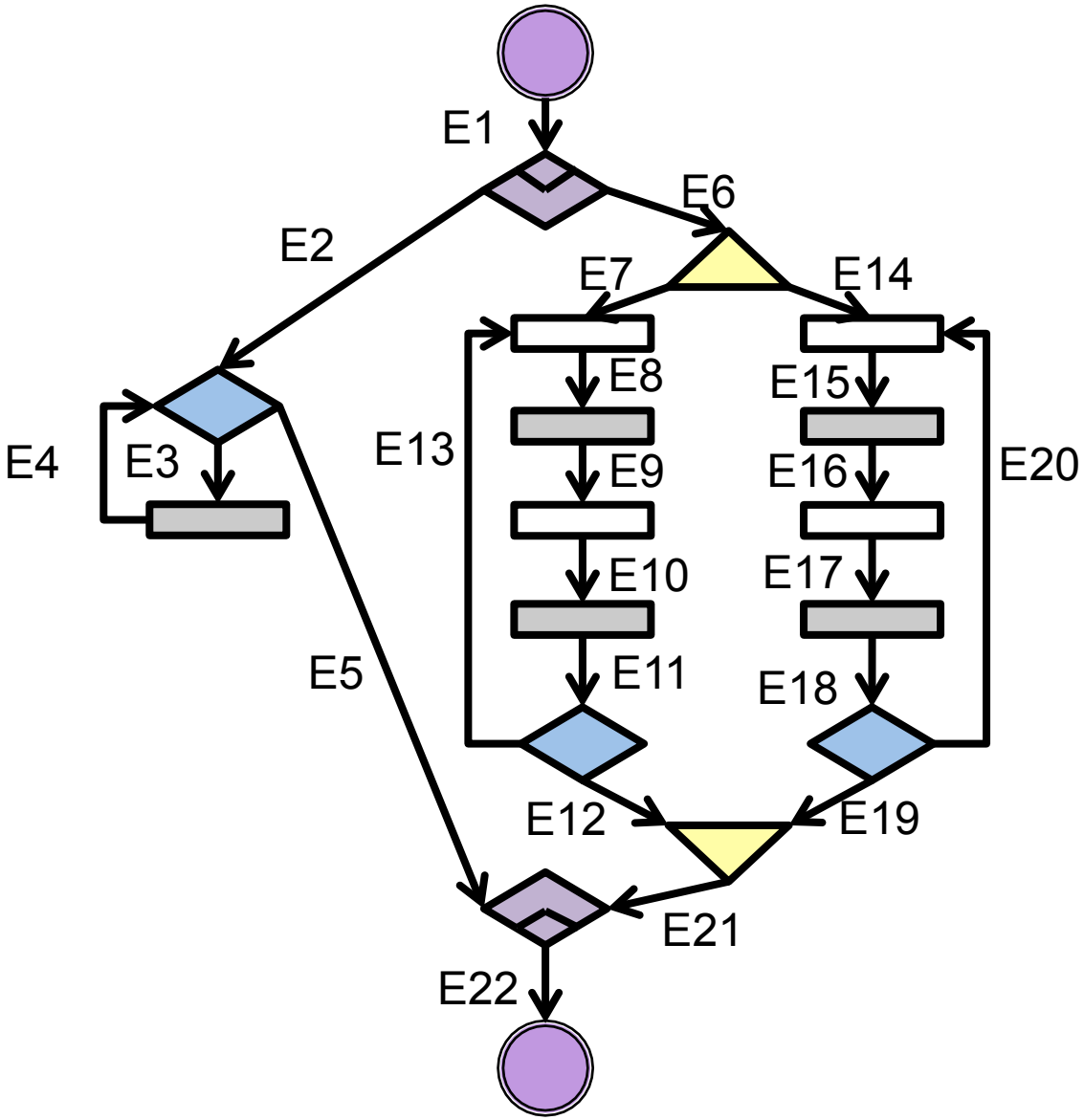
- $E_i \in \{0,1\}$
- Solver set E_i to 1 whenever is possible
- Inequalities

Some special edges:

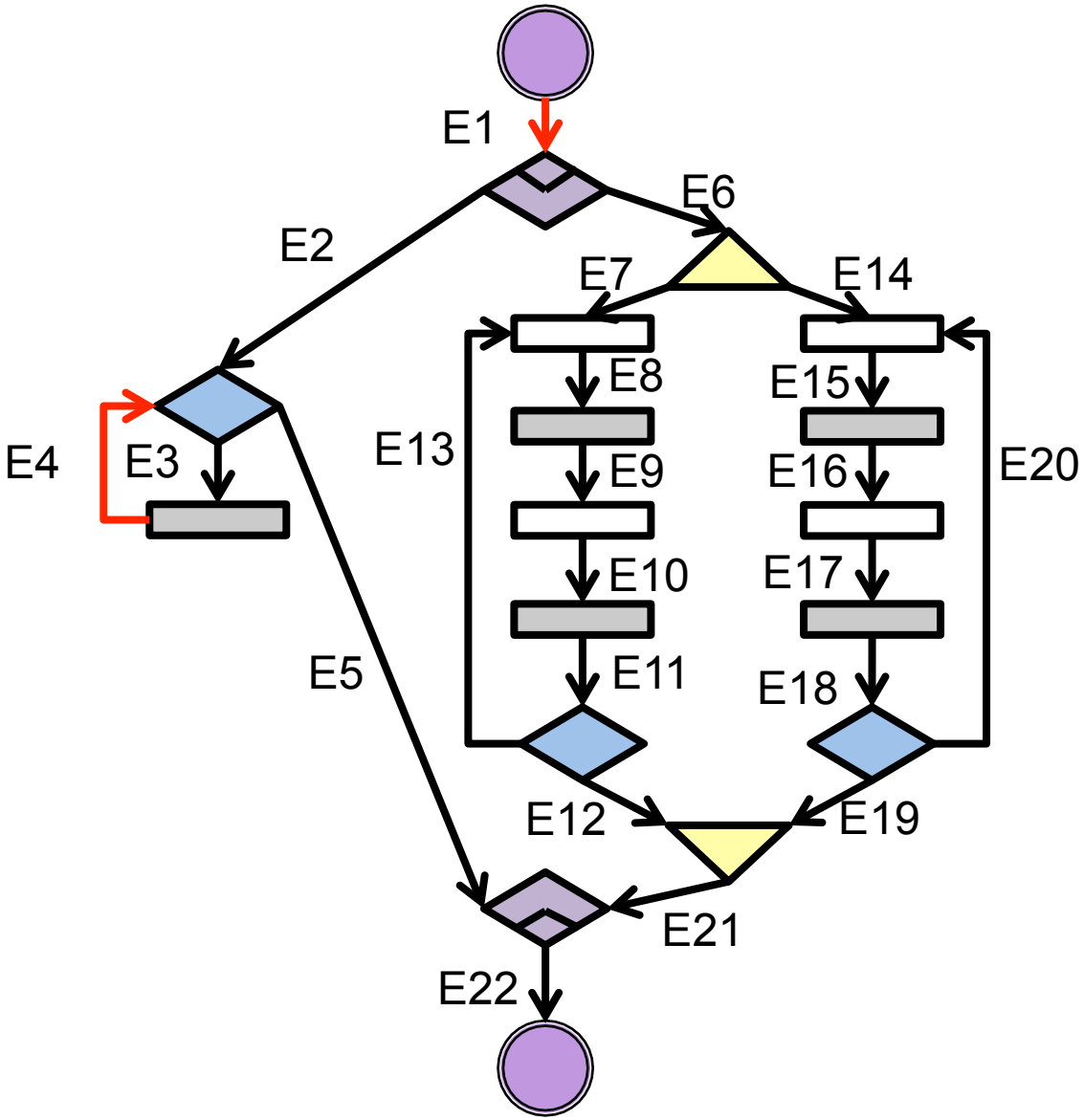
- EOT edges

ILP model

EOT:



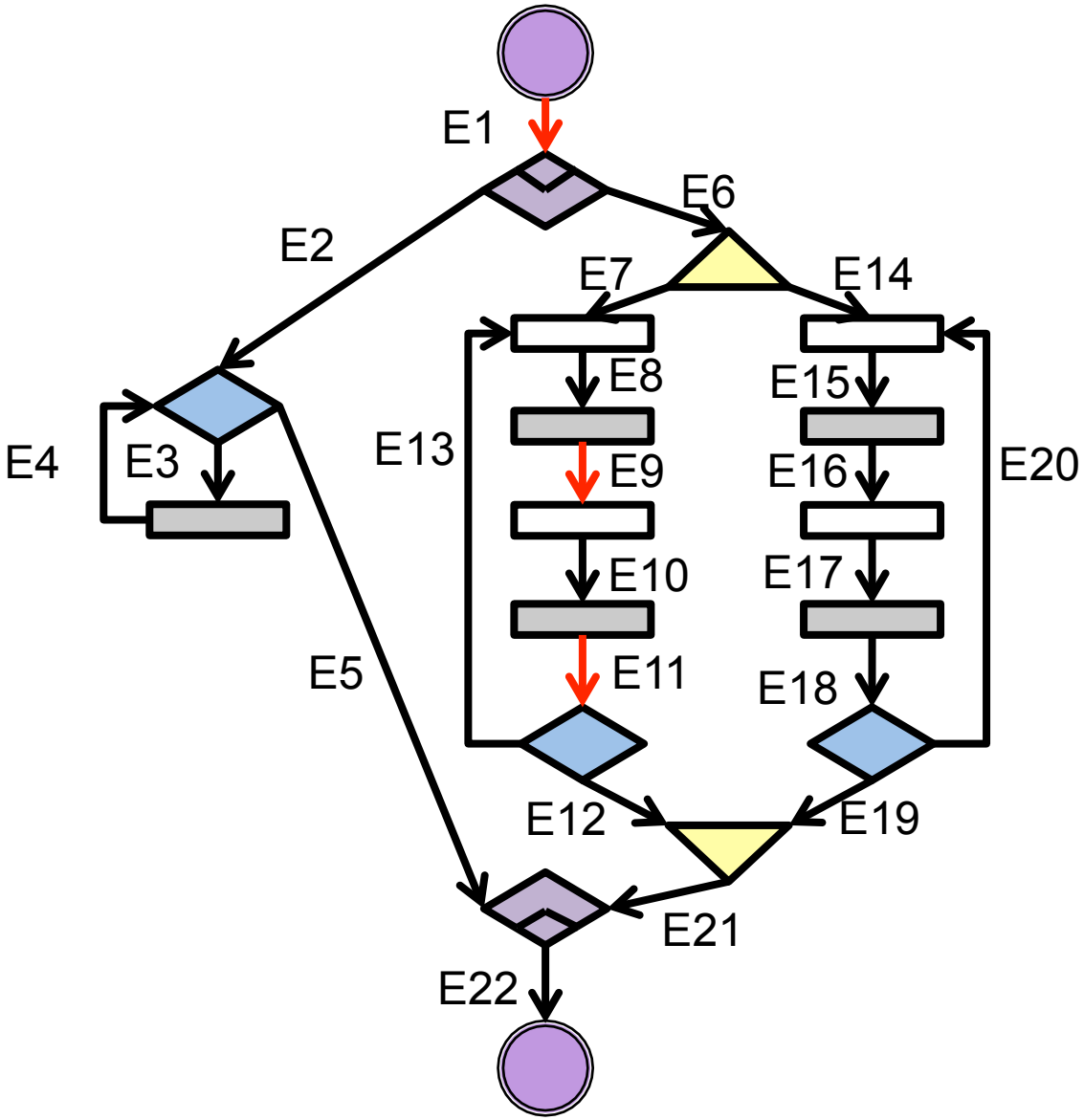
ILP model



EOT:

E1 + E4 ≤ 1

ILP model

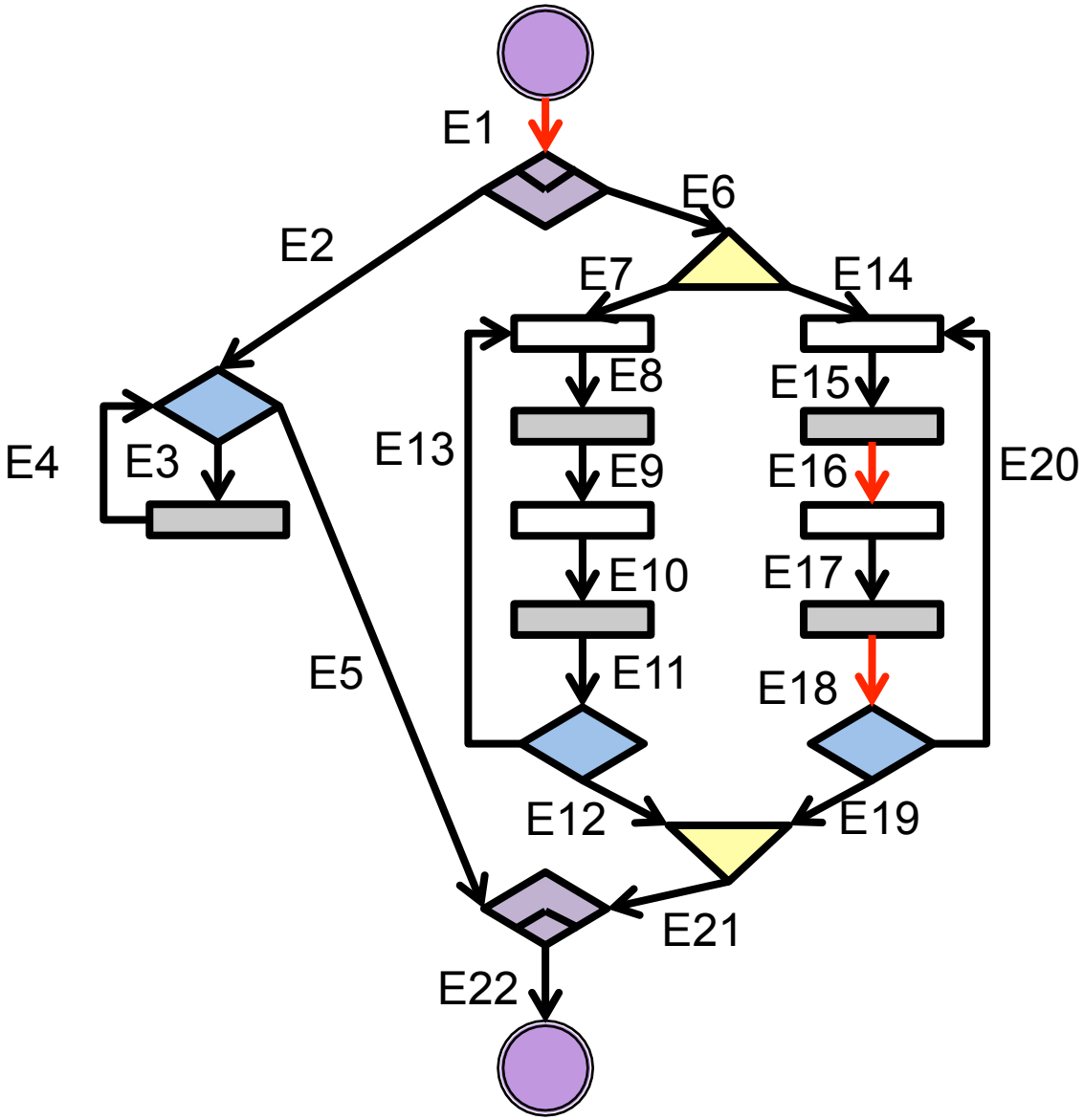


EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

ILP model



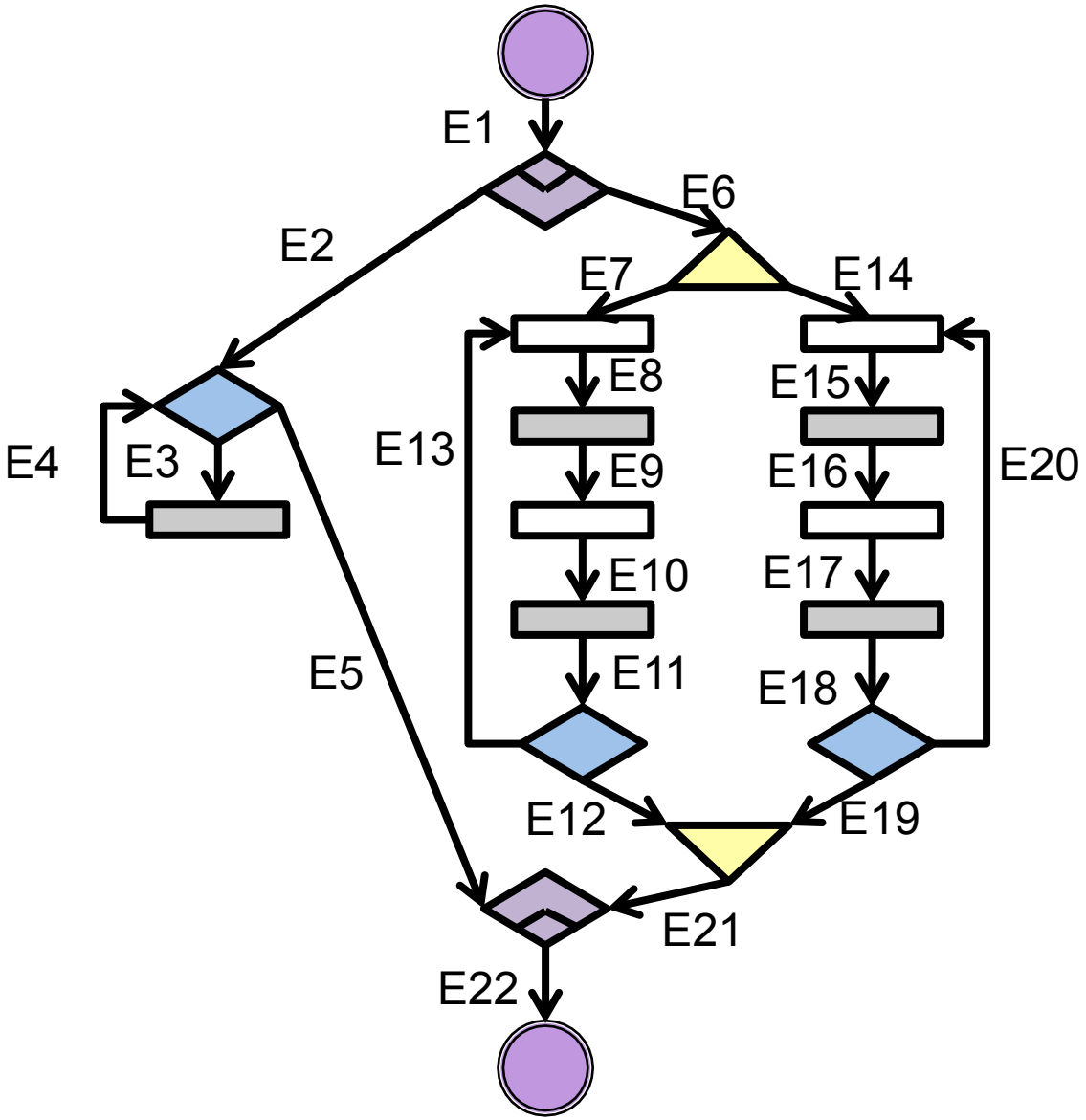
EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

ILP model



EOT:

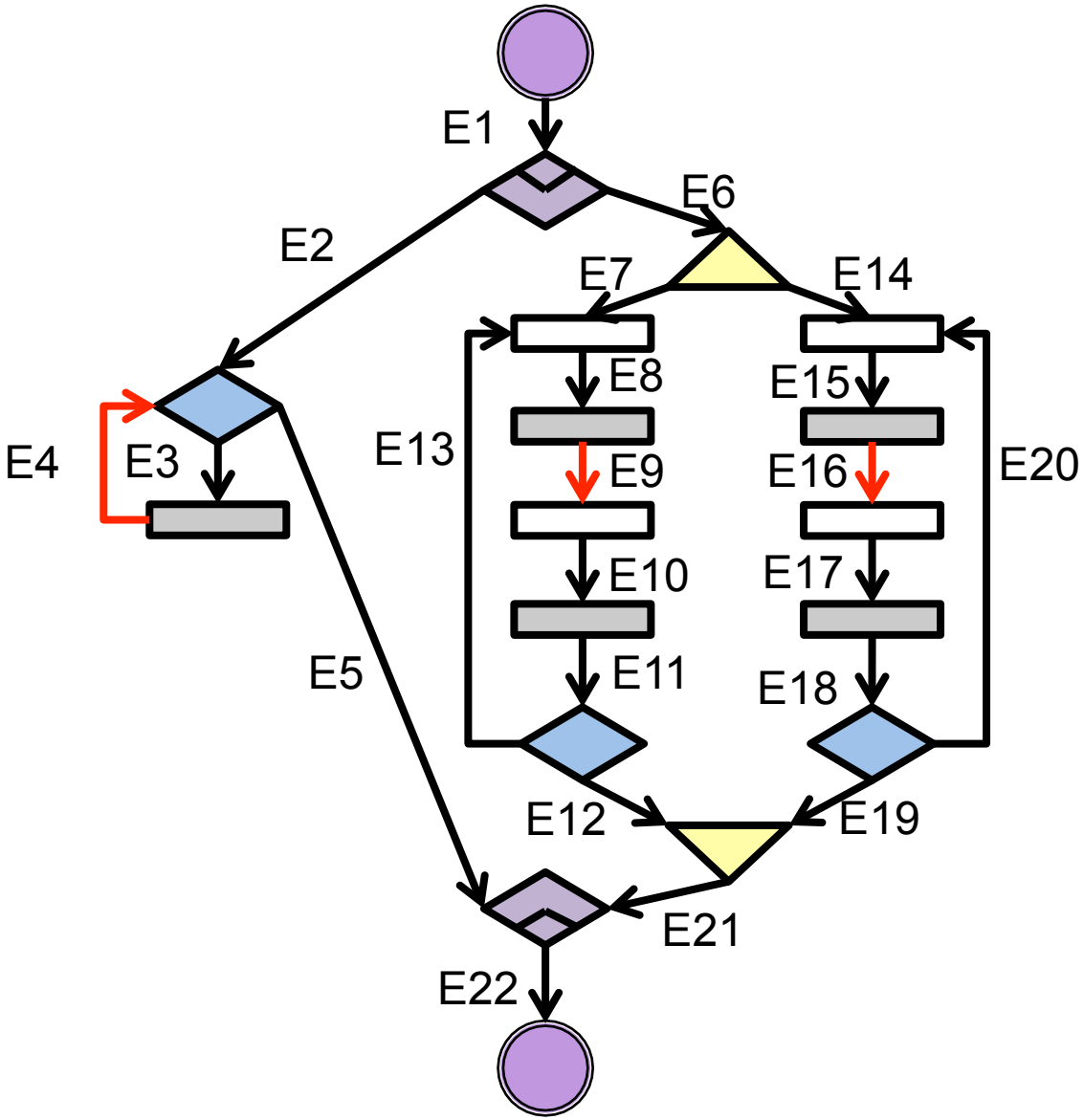
E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

- At most one EOT edge can be active per thread
- Parent / child relationship needs to be considered.

ILP model



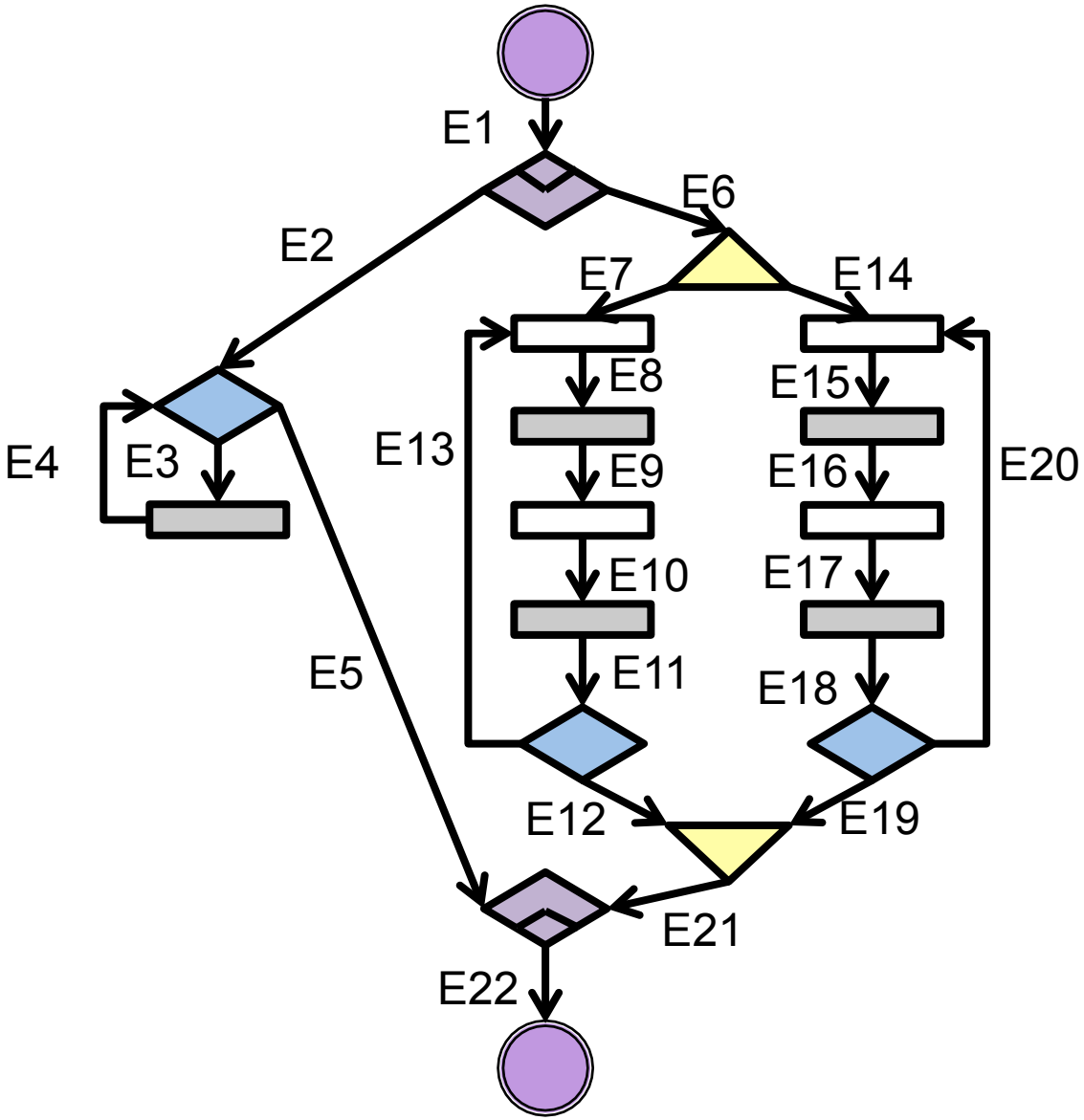
EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

ILP model



EOT:

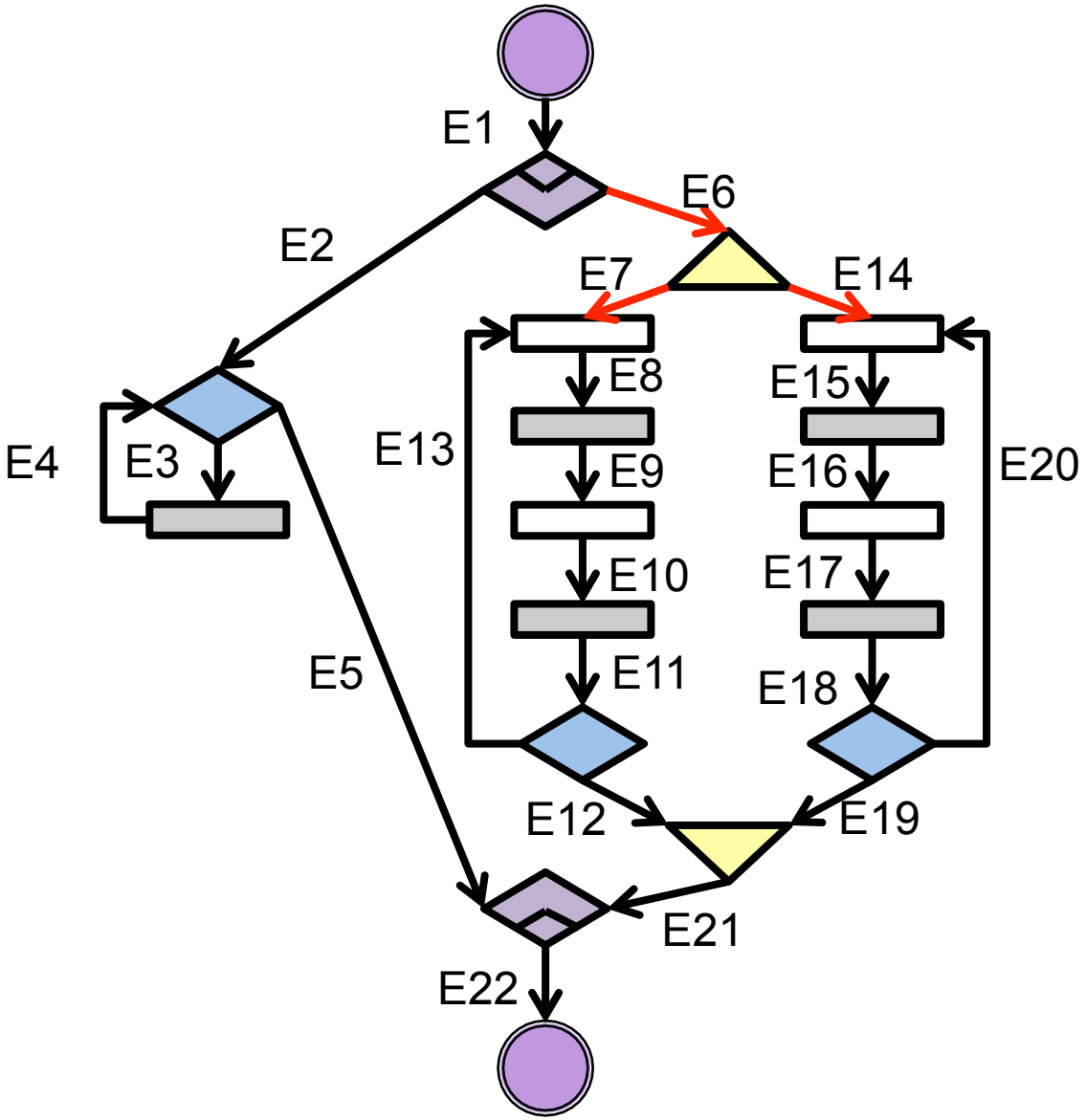
E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

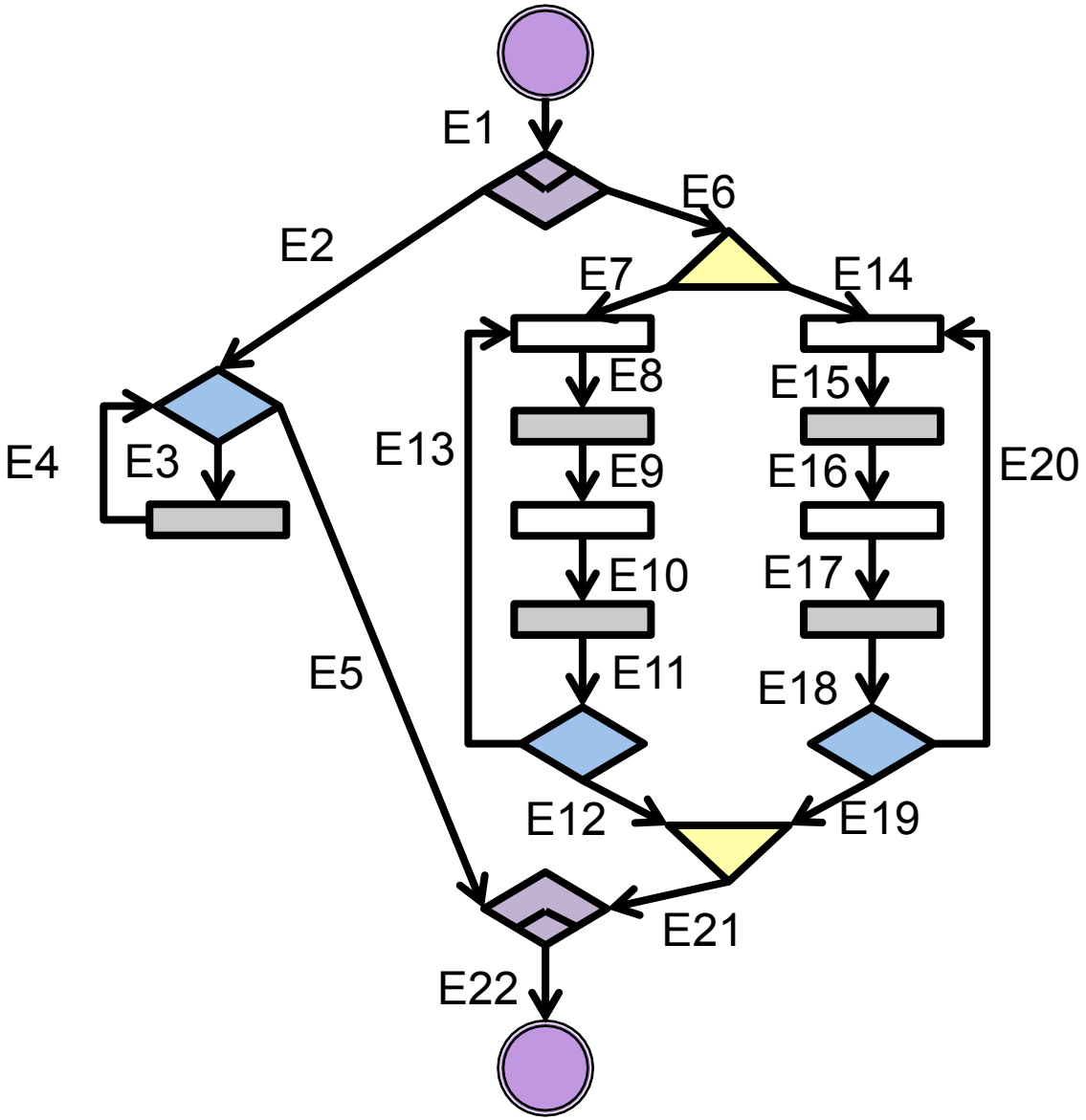
E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

E14 = E6

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

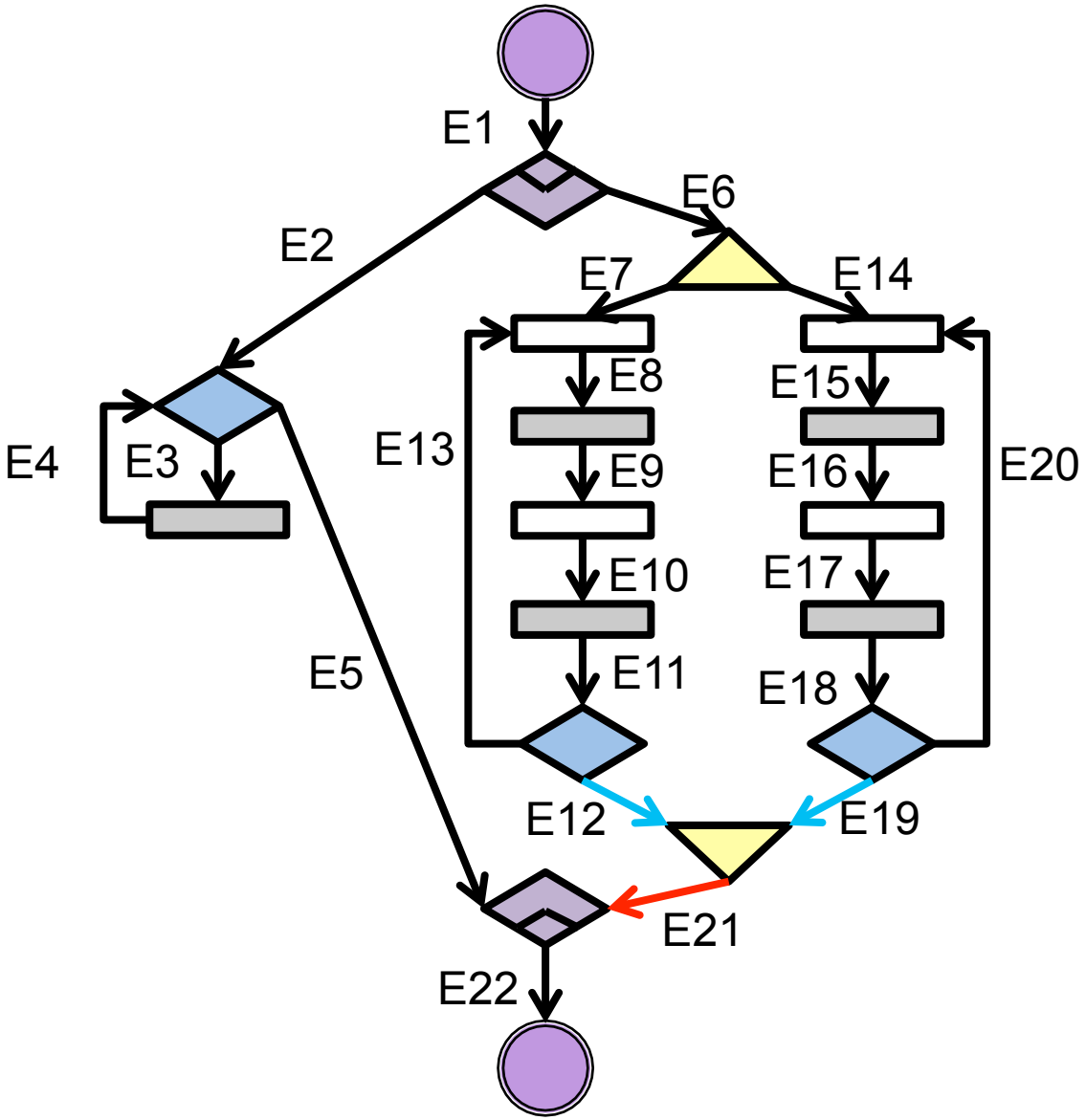
Fork:

E7 = E6

E14 = E6

Join:

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

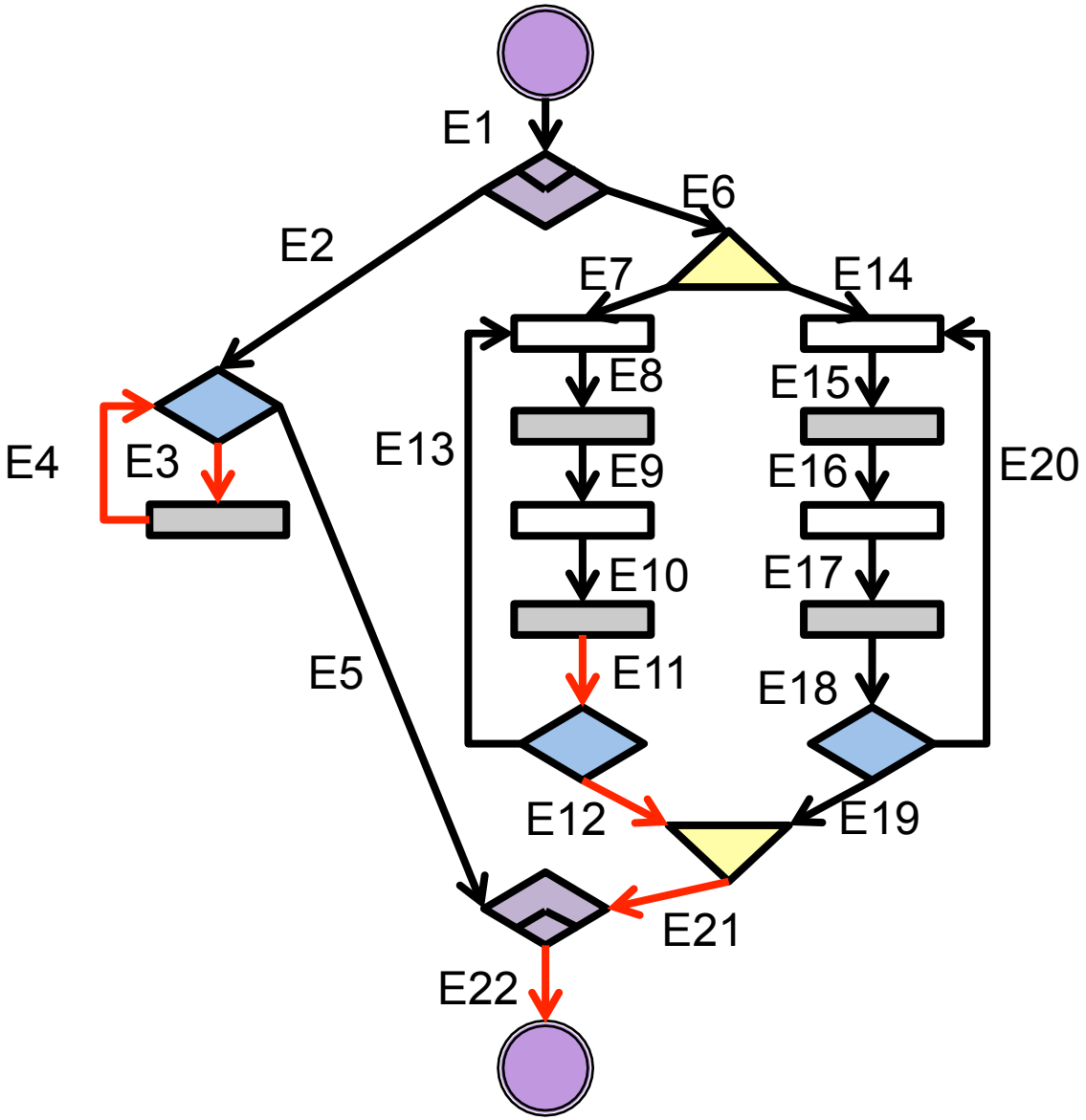
E7 = E6

E14 = E6

Join:

E21 ≤ E12 + E19

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

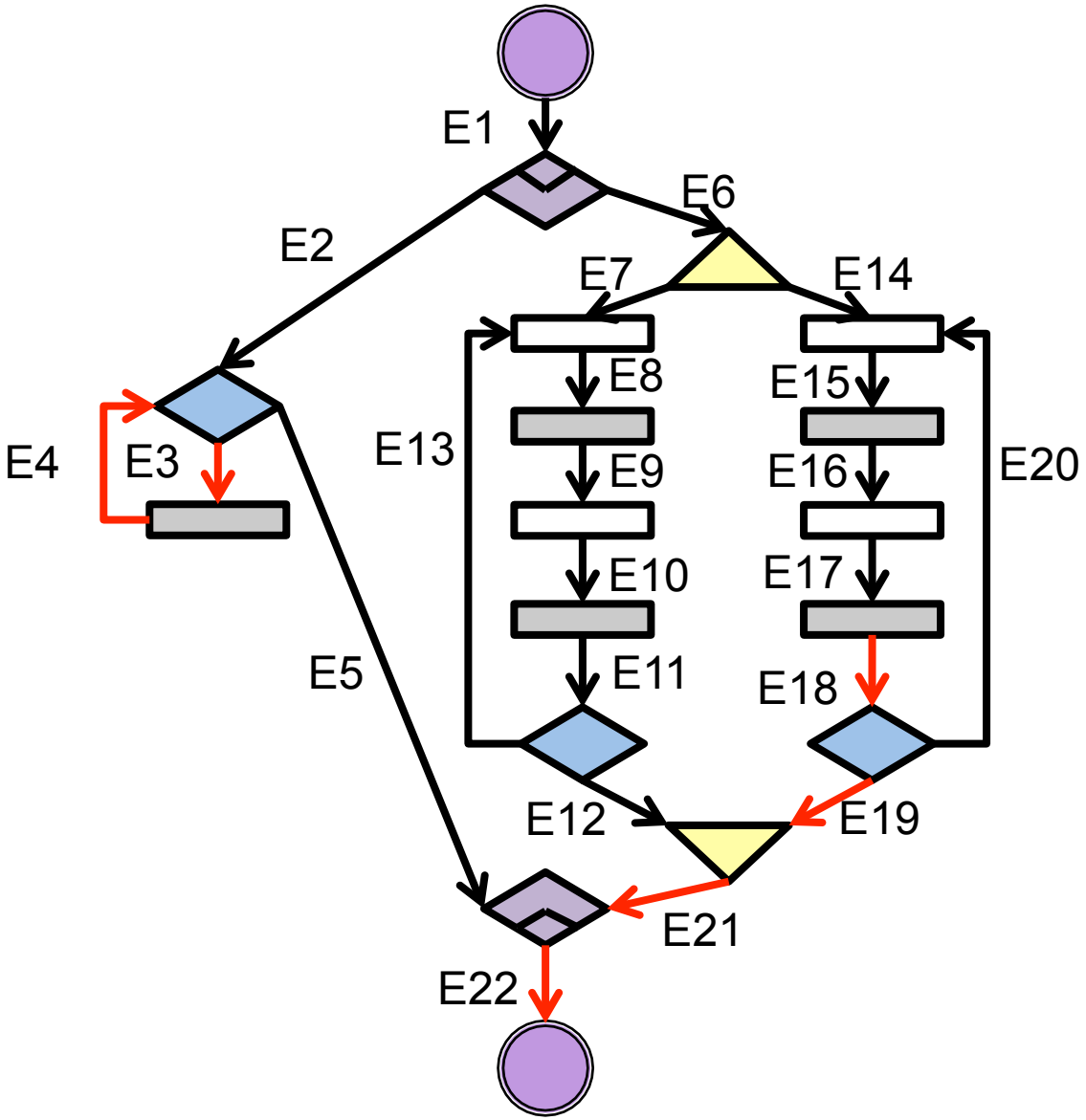
E7 = E6

E14 = E6

Join:

E21 ≤ E12 + E19

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

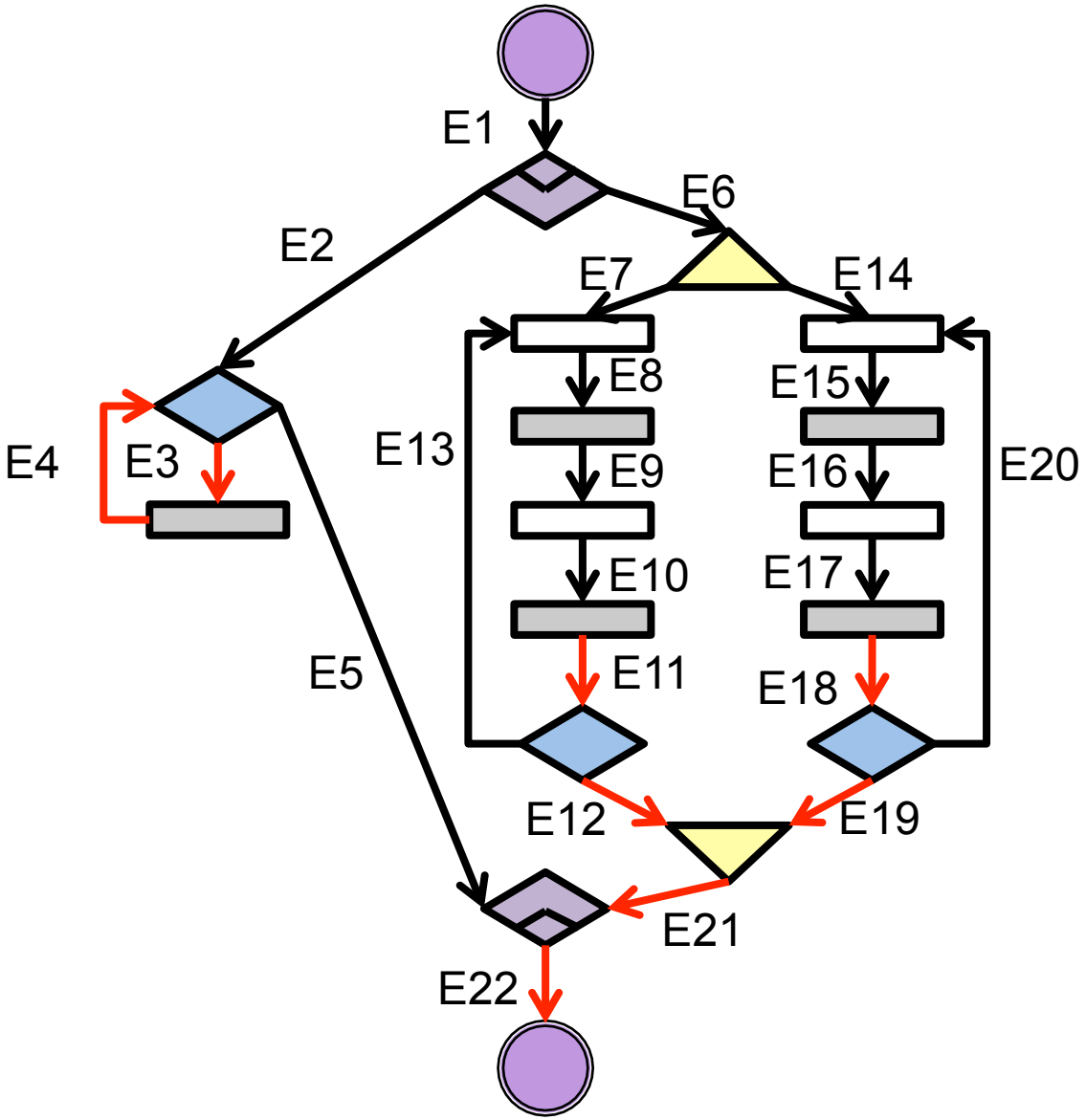
E7 = E6

E14 = E6

Join:

E21 ≤ E12 + E19

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

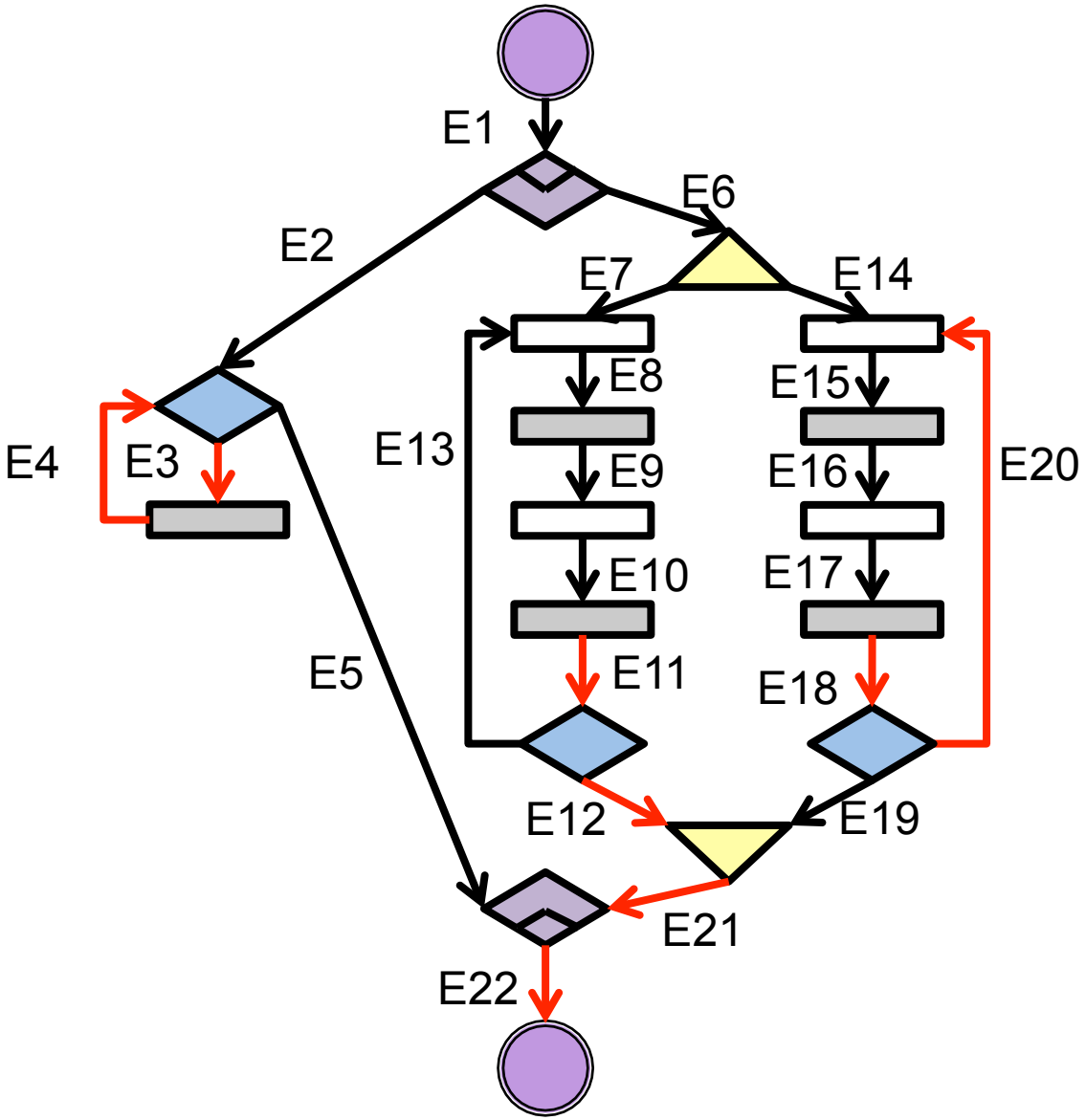
E7 = E6

E14 = E6

Join:

E21 ≤ E12 + E19

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

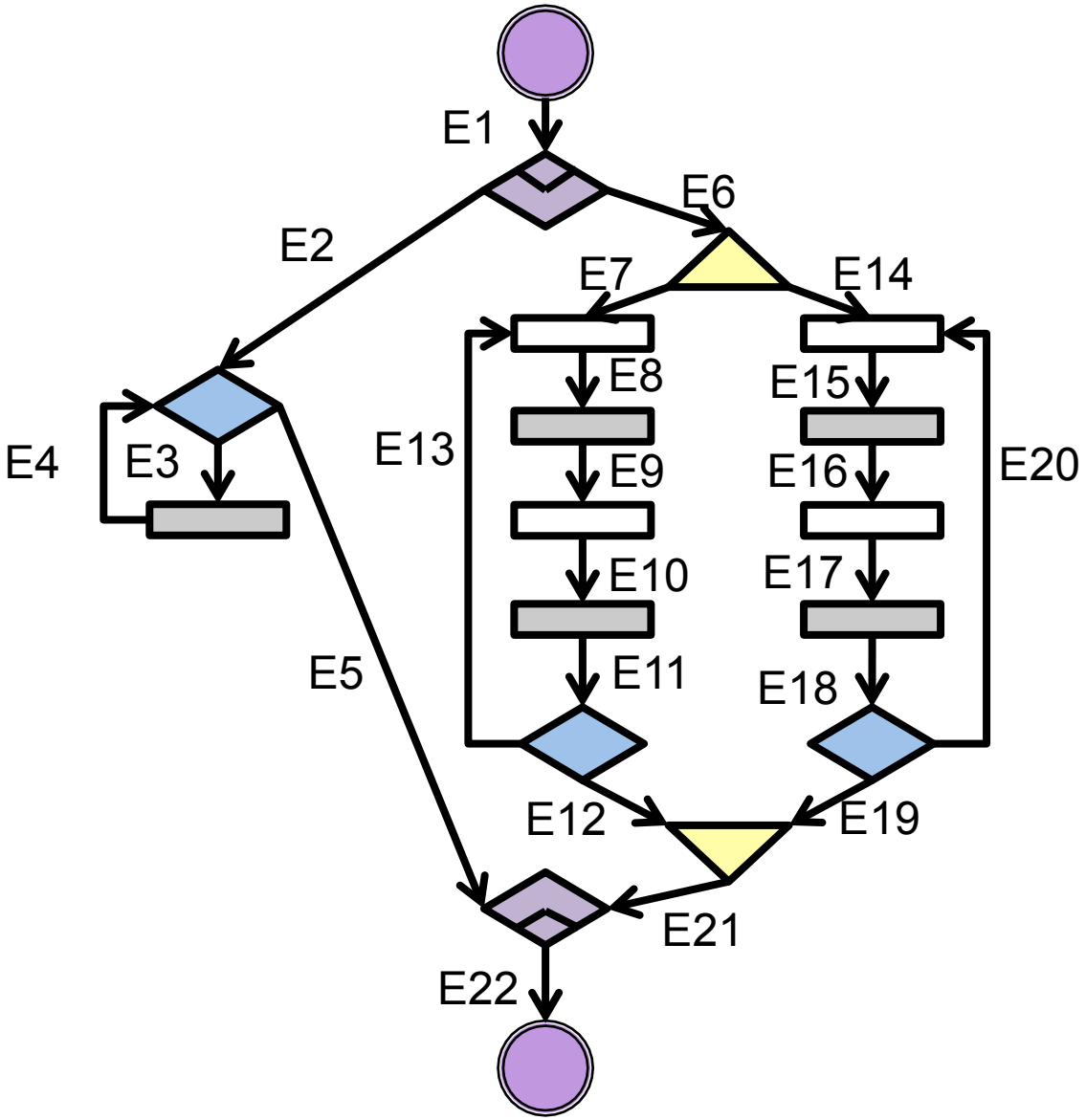
E14 = E6

Join:

E21 ≤ E12 + E19

Exception

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

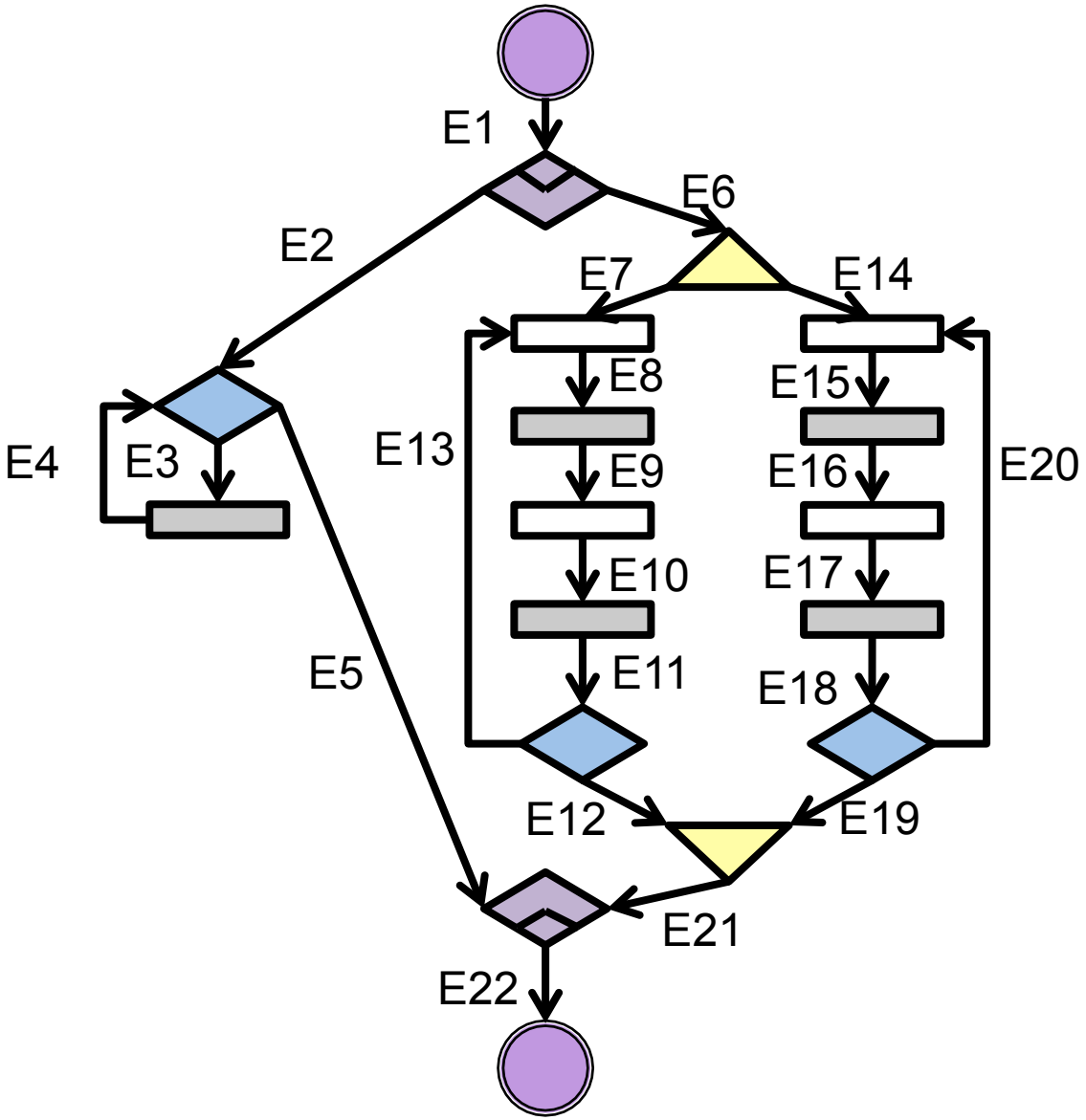
E14 = E6

Join:

E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

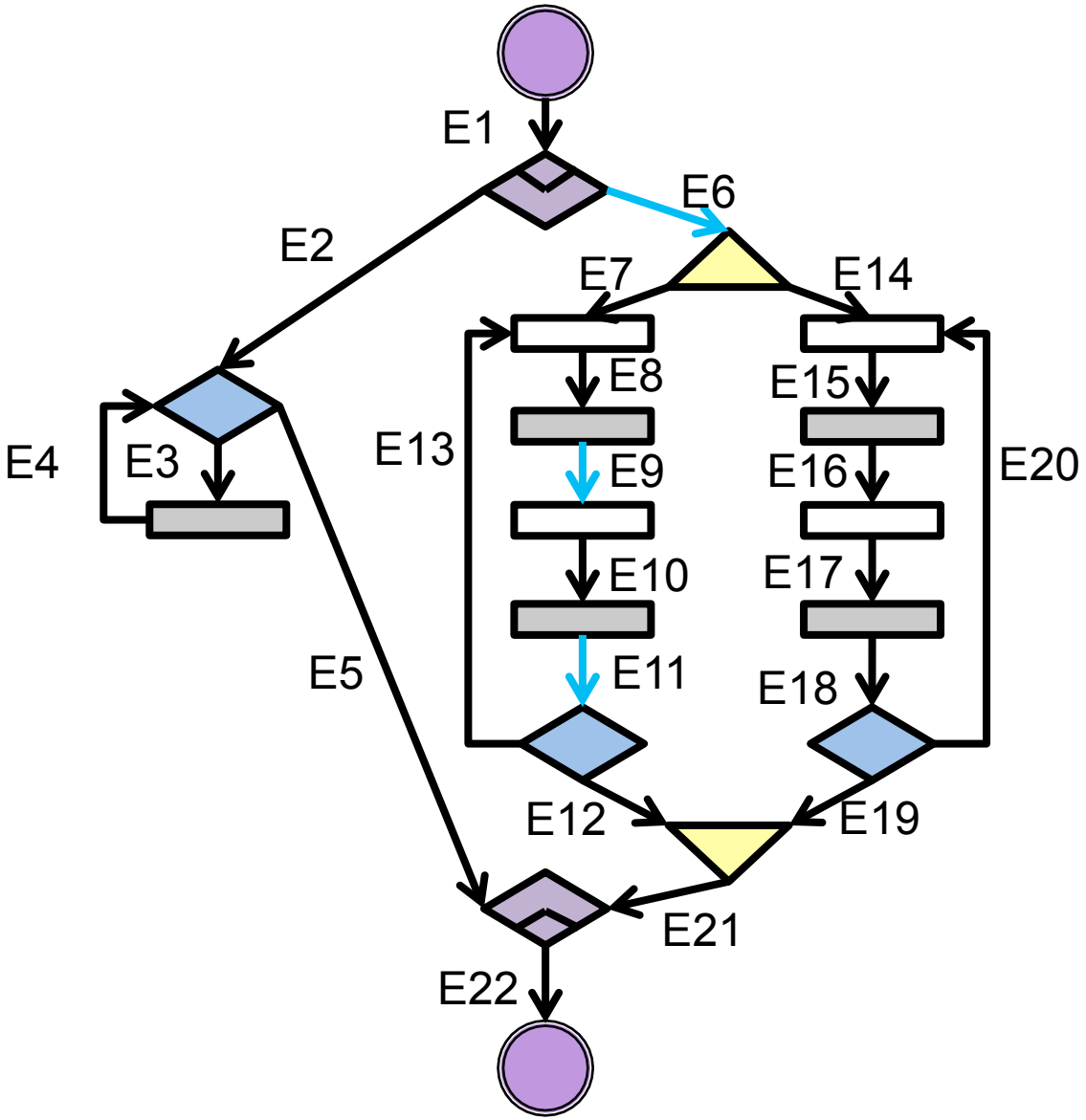
E14 = E6

Join:

E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

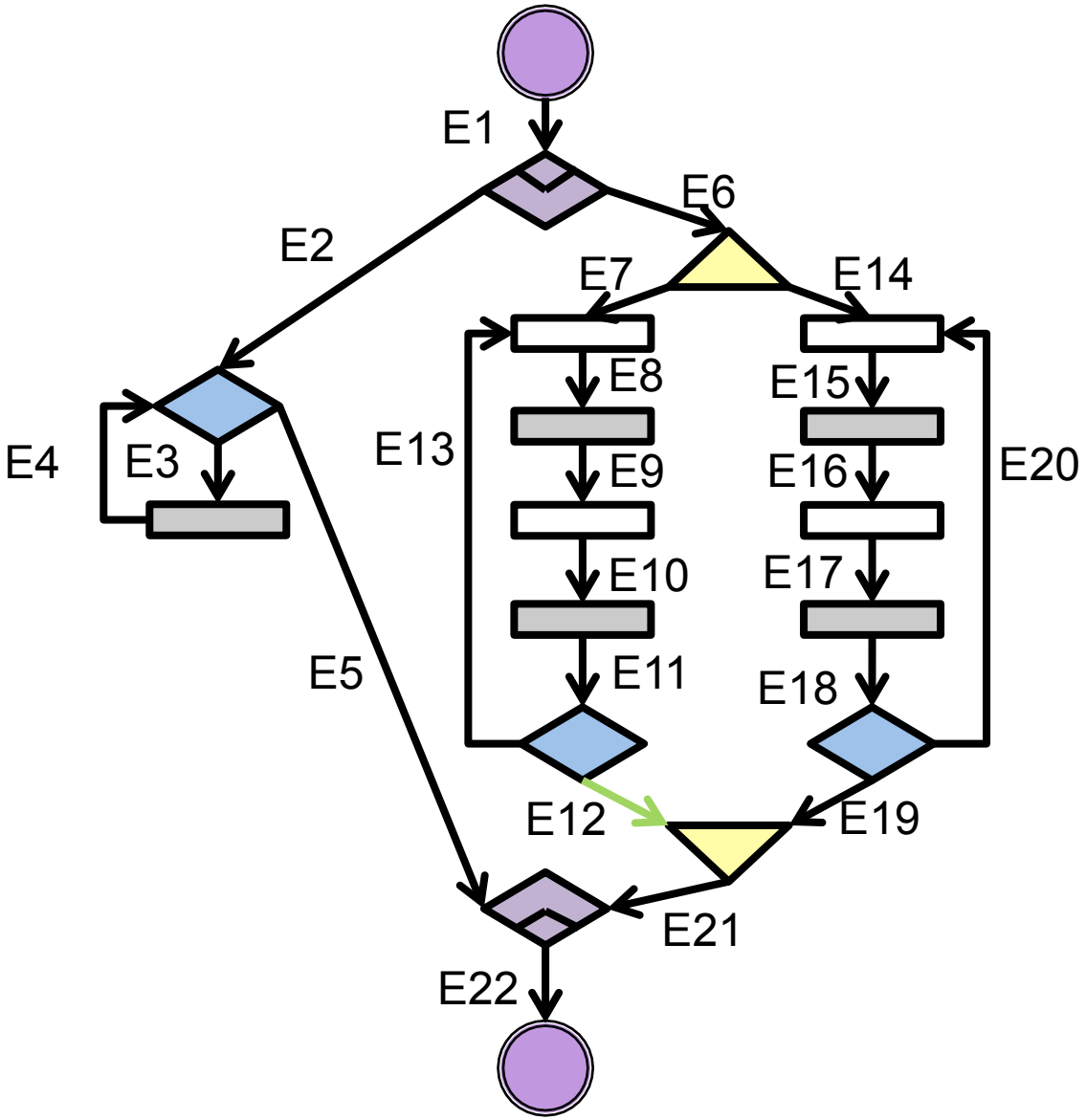
E14 = E6

Join:

E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

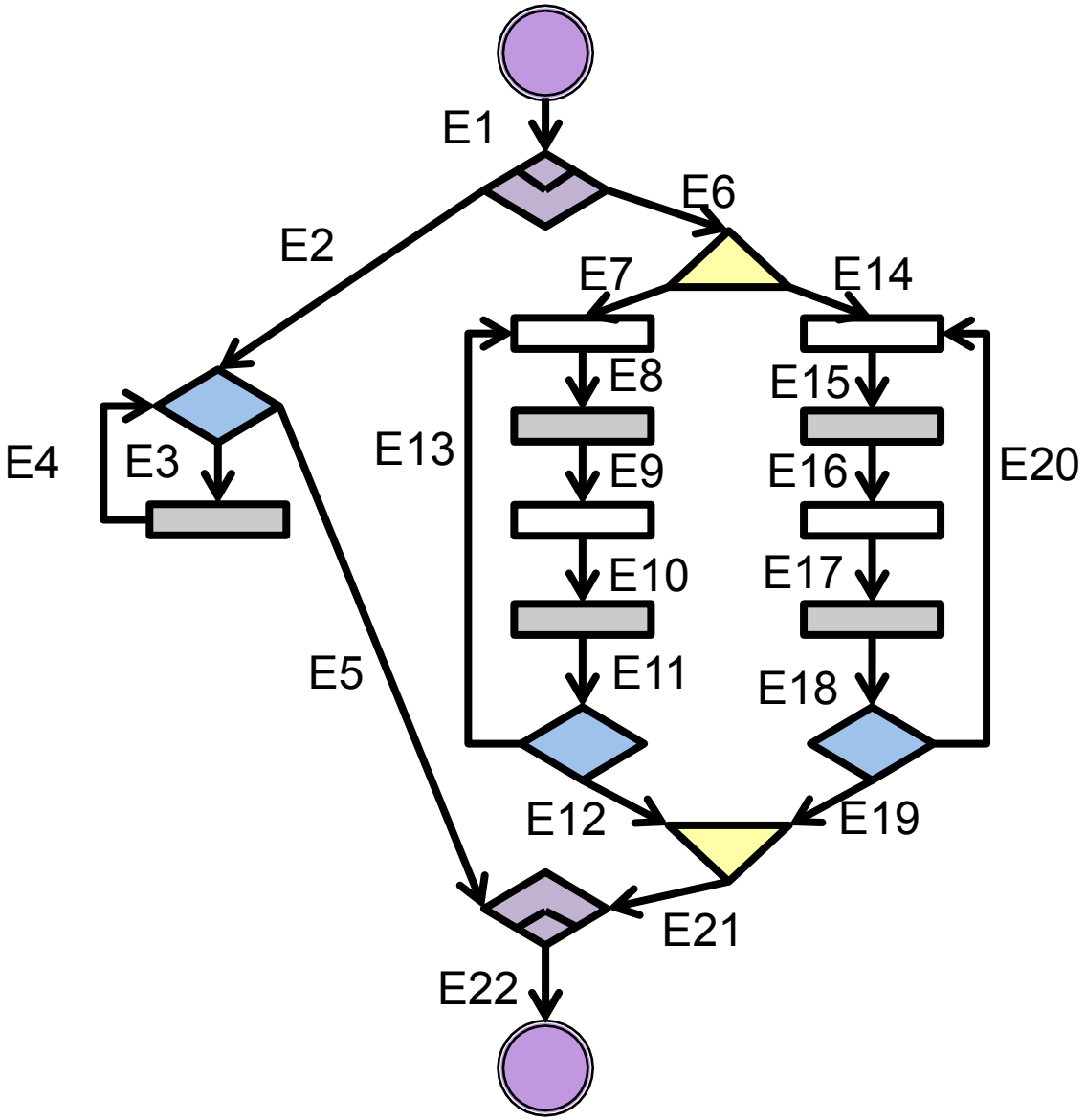
E14 = E6

Join:

E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

E14 = E6

Join:

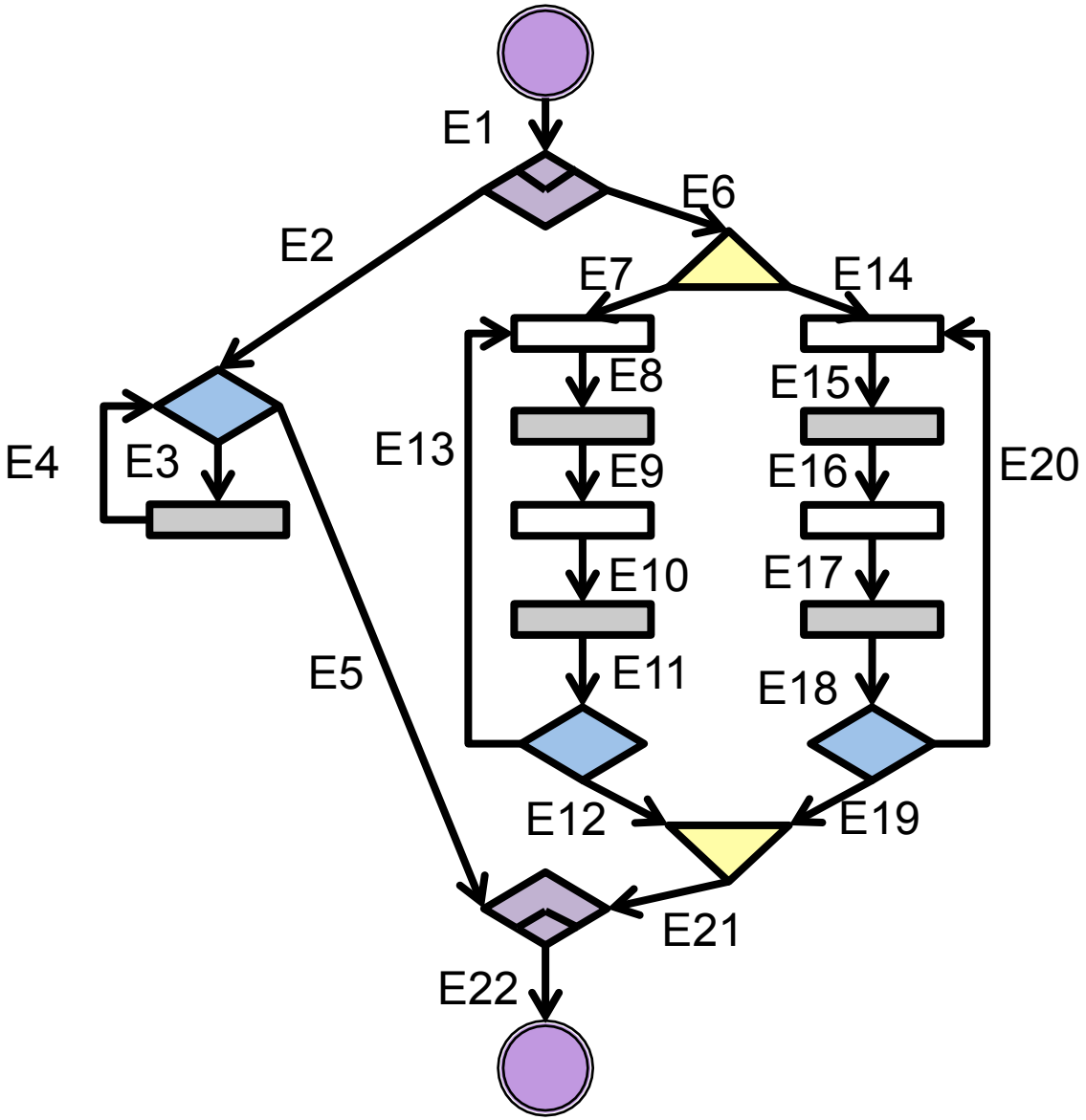
E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

Using EOT edges:

- Enough to represent all edges in a thread

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

E14 = E6

Join:

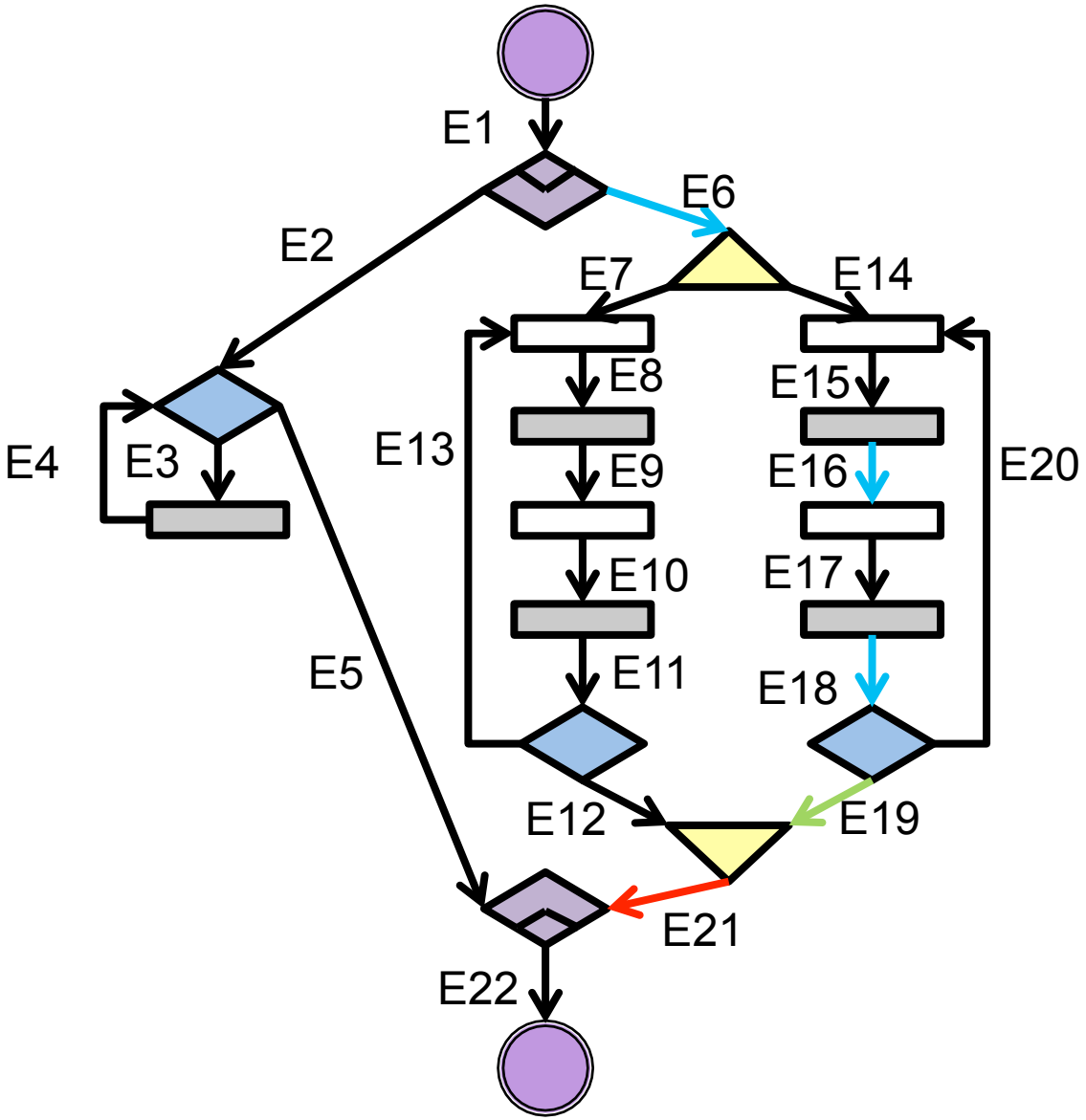
E21 ≤ E12 + E19

E21≤(1-E6-E9-E11)+E12

Using EOT edges:

- Enough to represent all edges in a thread
- Re-use EOT constraints

ILP model



EOT:

$E1 + E4 \leq 1$

$E1 + E9 + E11 \leq 1$

$E1 + E16 + E18 \leq 1$

Fork:

$E7 = E6$

$E14 = E6$

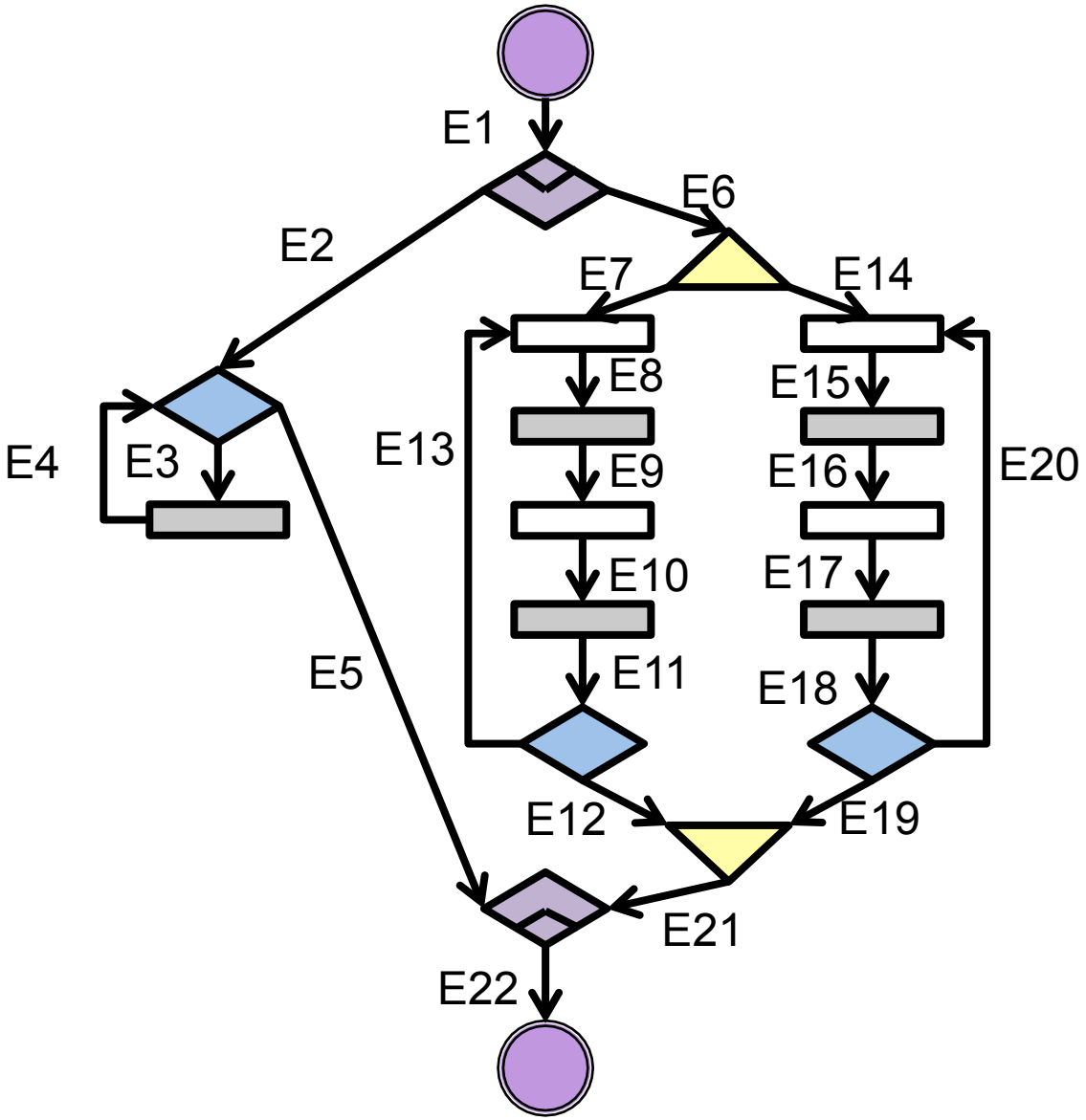
Join:

$E21 \leq E12 + E19$

$E21 \leq (1 - E6 - E9 - E11) + E12$

$E21 \leq (1 - E6 - E16 - E18) + E19$

ILP model



EOT:

E1 + E4 ≤ 1

E1 + E9 + E11 ≤ 1

E1 + E16 + E18 ≤ 1

Fork:

E7 = E6

E14 = E6

Join:

E21 ≤ E12 + E19

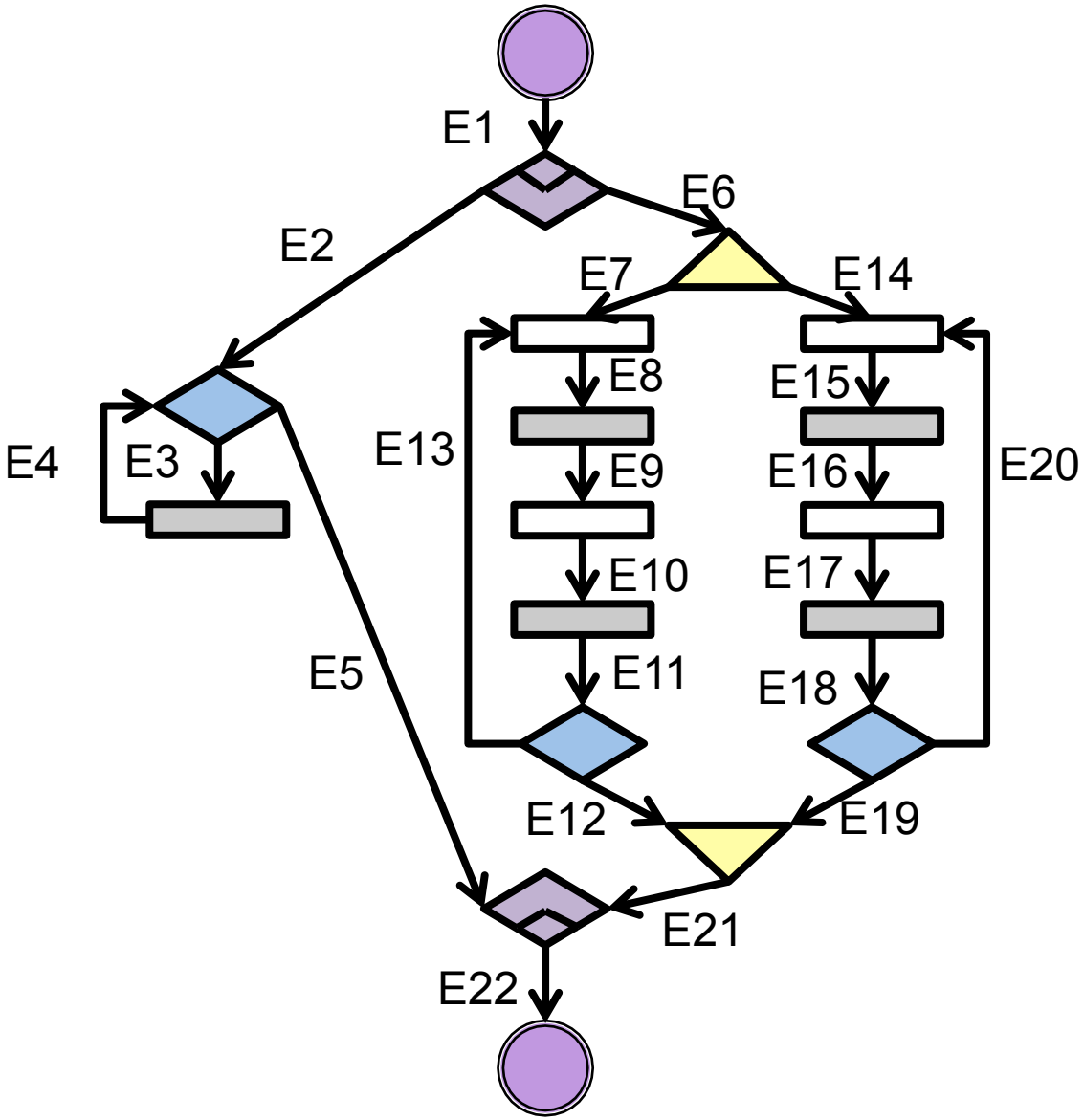
E21≤(1-E6-E9-E11)+E12

E21≤(1-E6-E16-E18)+E19

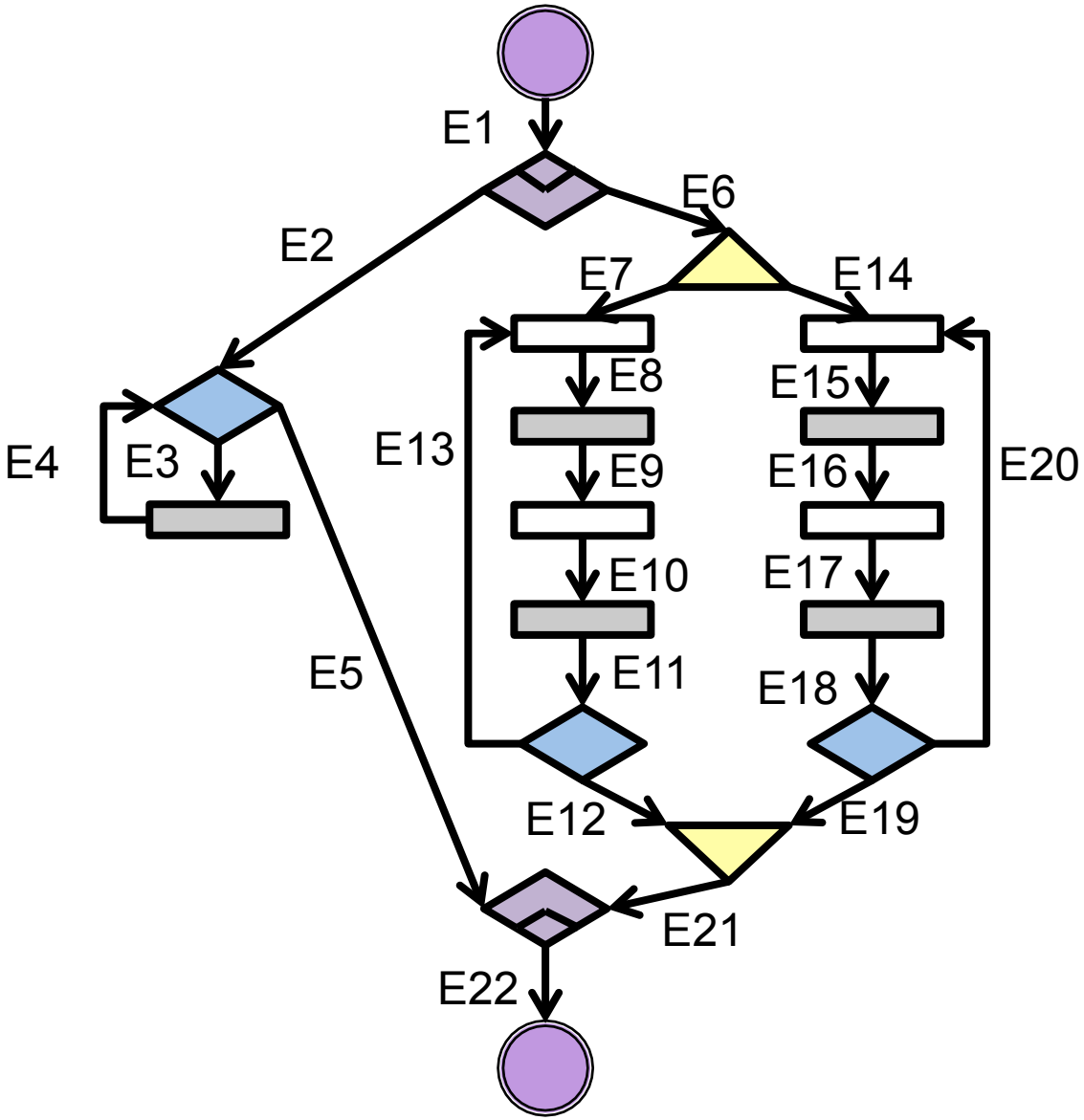
E21 is ‘1’ if the right hand side of all the constraints are greater than ‘0’

ILP model

Abort start:



ILP model

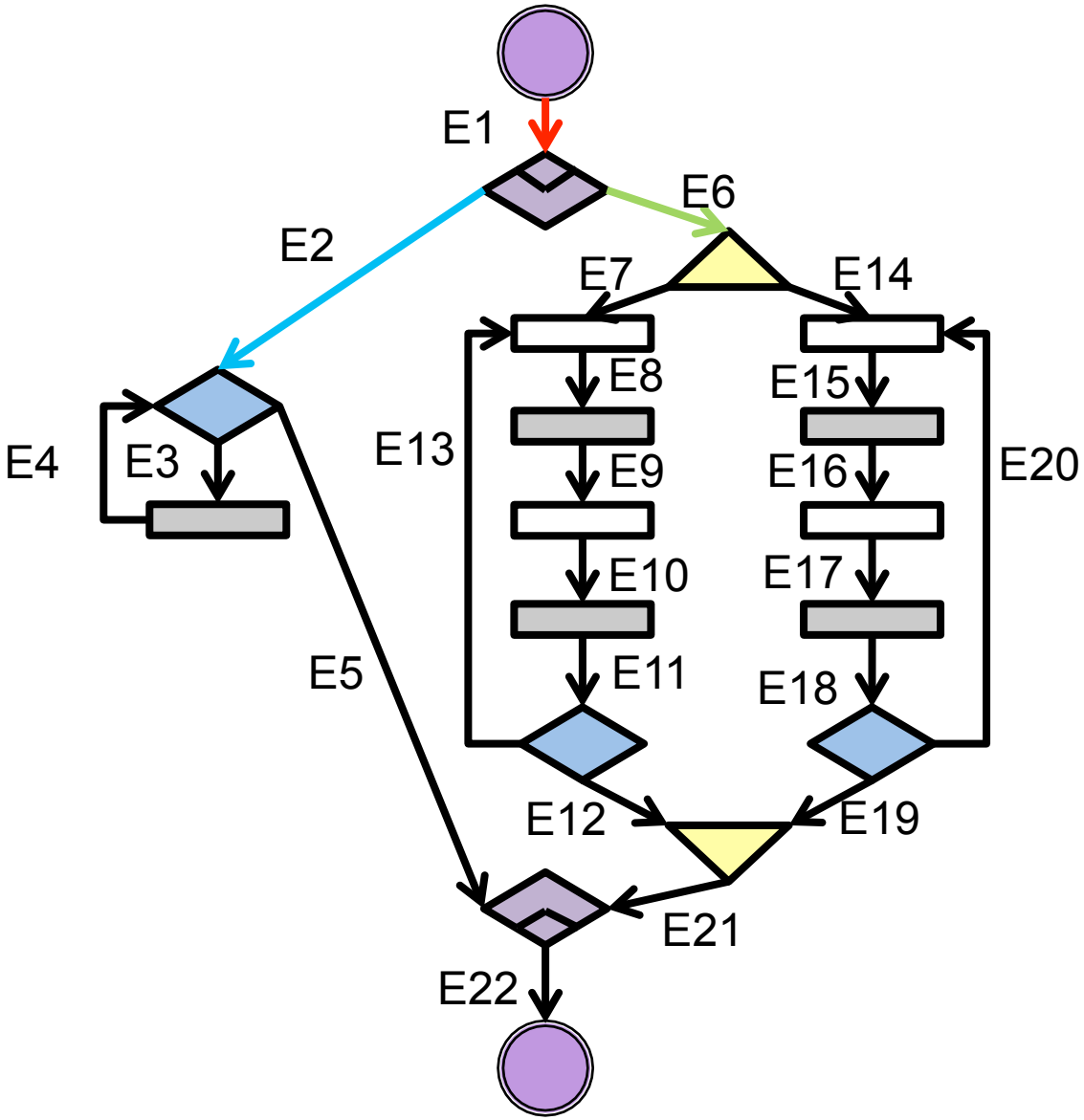


Abort start:

E1 = E2

E1 ≥ E6

ILP model

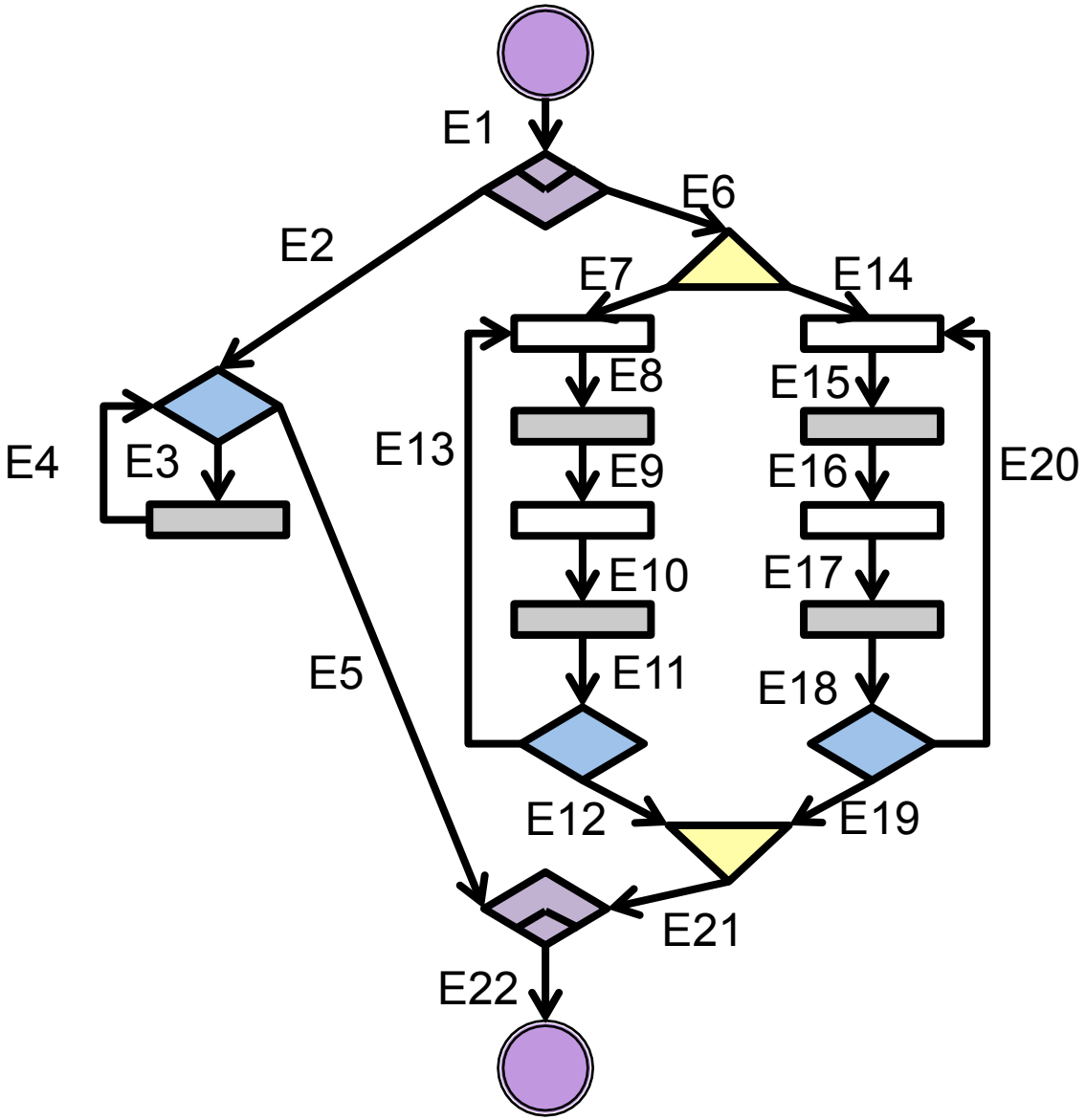


Abort start:

E1 = ***E2***

E1 ≥ ***E6***

ILP model



Abort start:

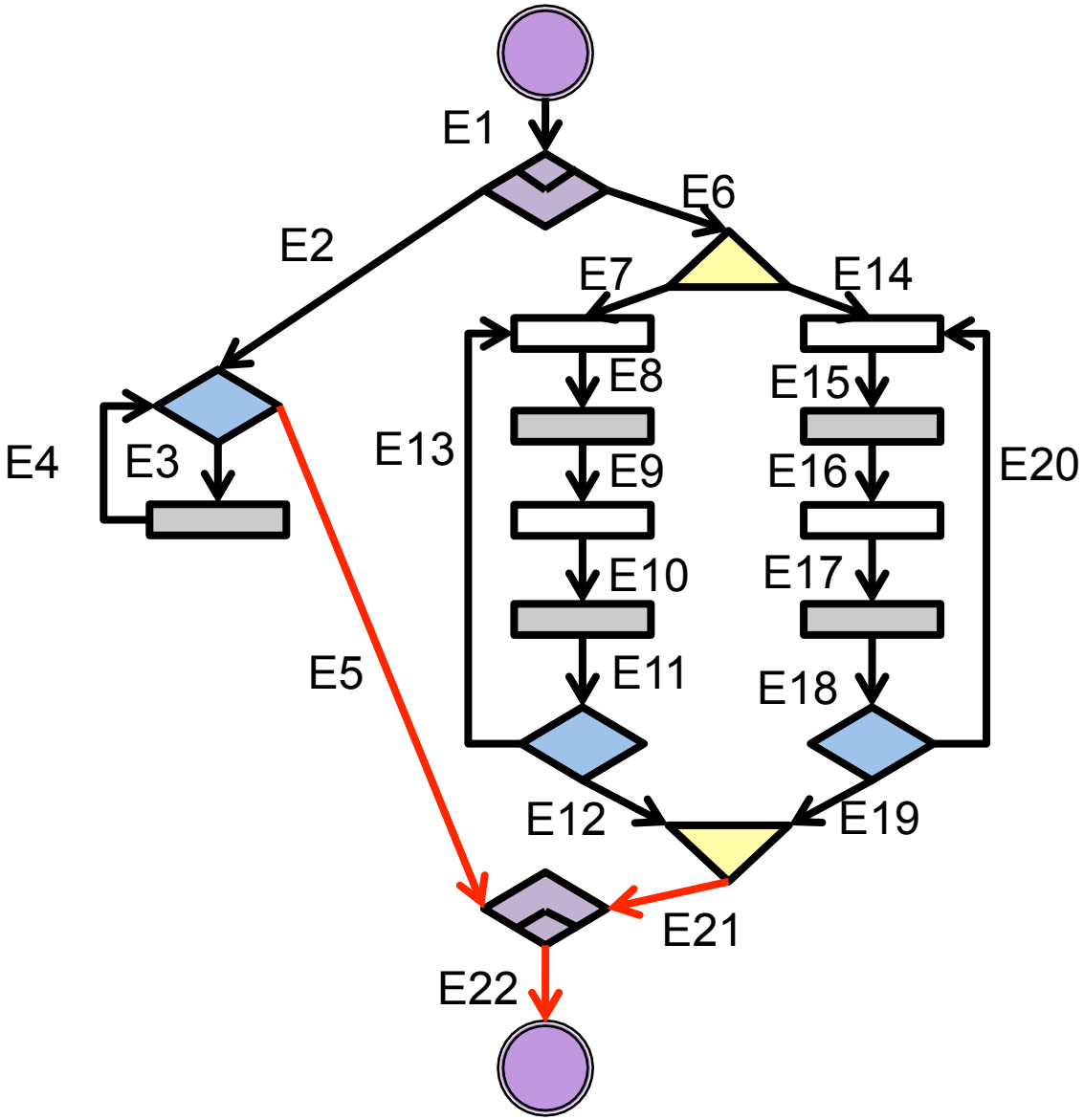
E1 = E2

E1 ≥ E6

Abort End:

E22 = E5 + E21

ILP model



Abort start:

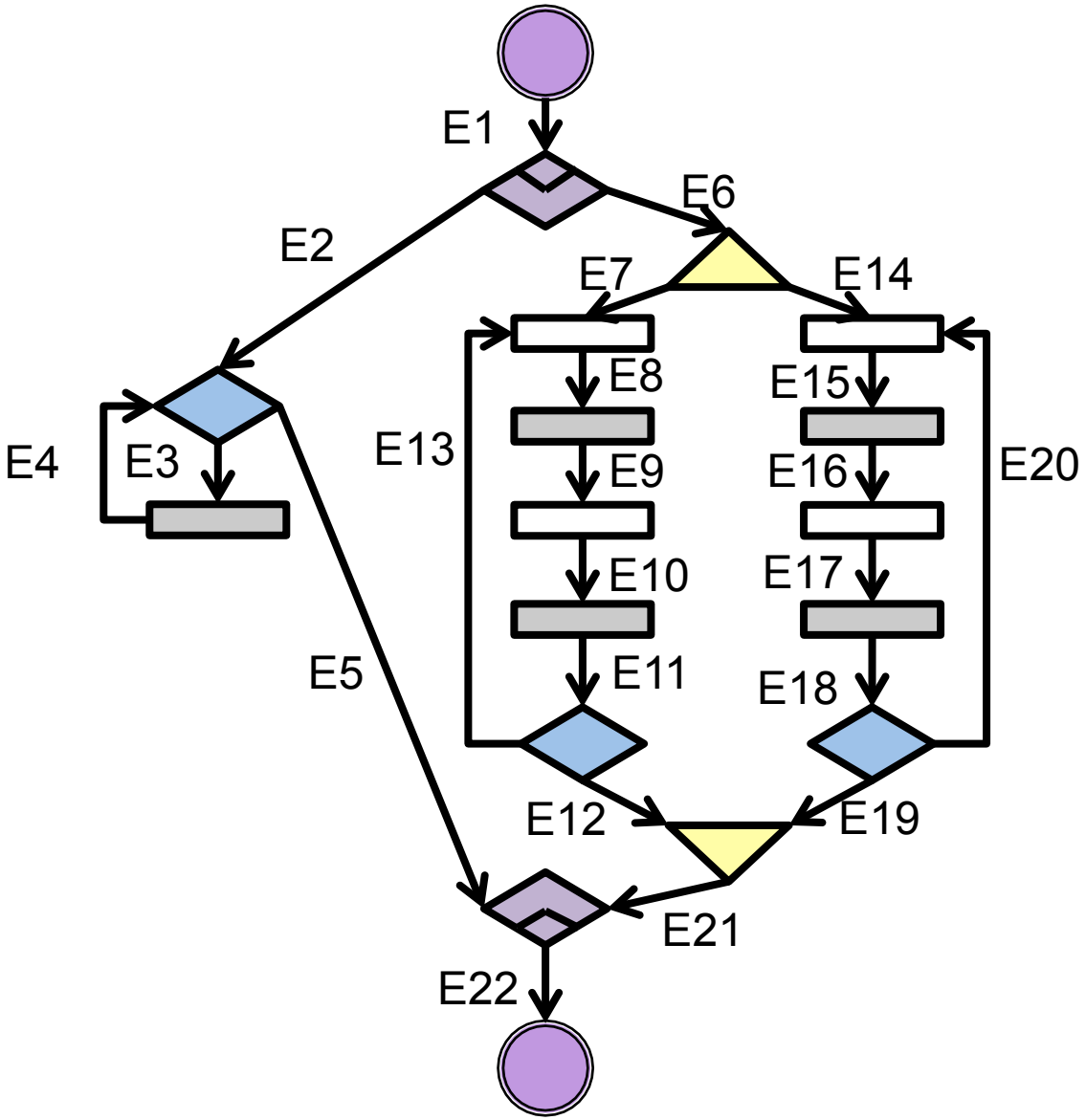
E1 = E2

E1 ≥ E6

Abort End:

E22 = E5 + E21

ILP model



Abort start:

E1 = E2

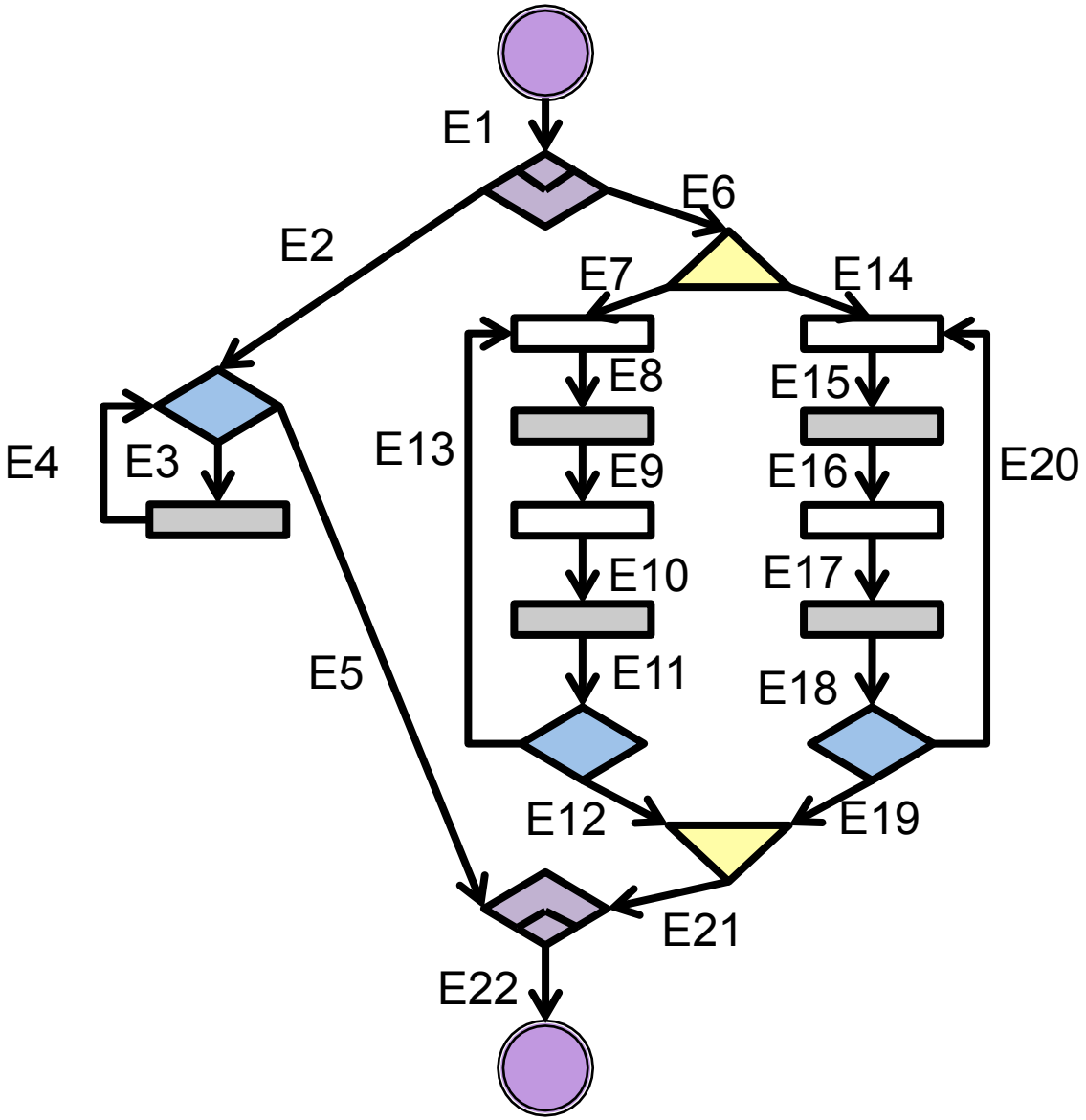
E1 ≥ E6

Abort End:

E22 = E5 + E21

Preemption:

ILP model



Abort start:

E1 = E2

E1 ≥ E6

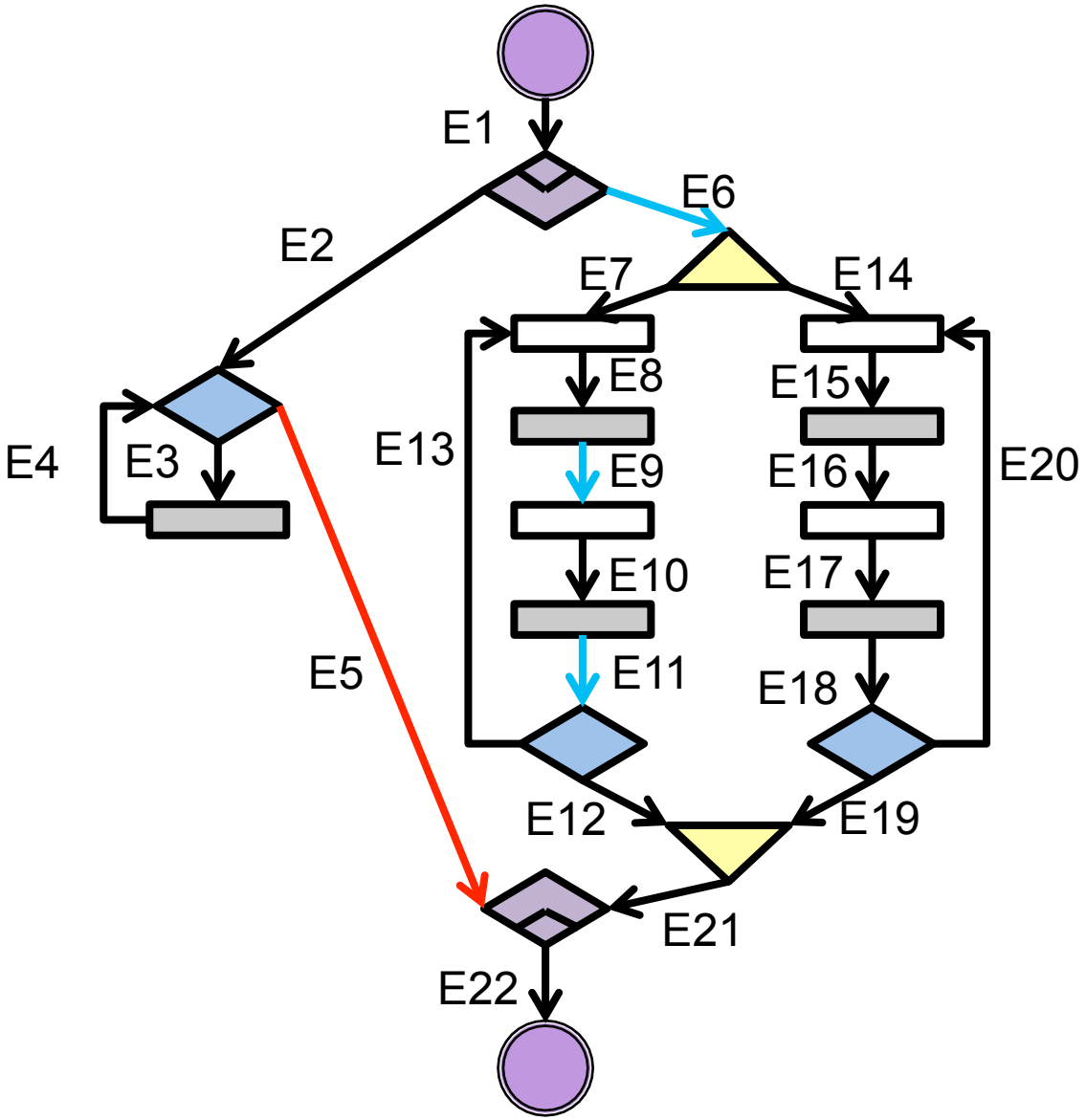
Abort End:

E22 = E5 + E21

Preemption:

E6+E9+E11 ≤ 1-E5

ILP model



Abort start:

E1 = E2

E1 ≥ E6

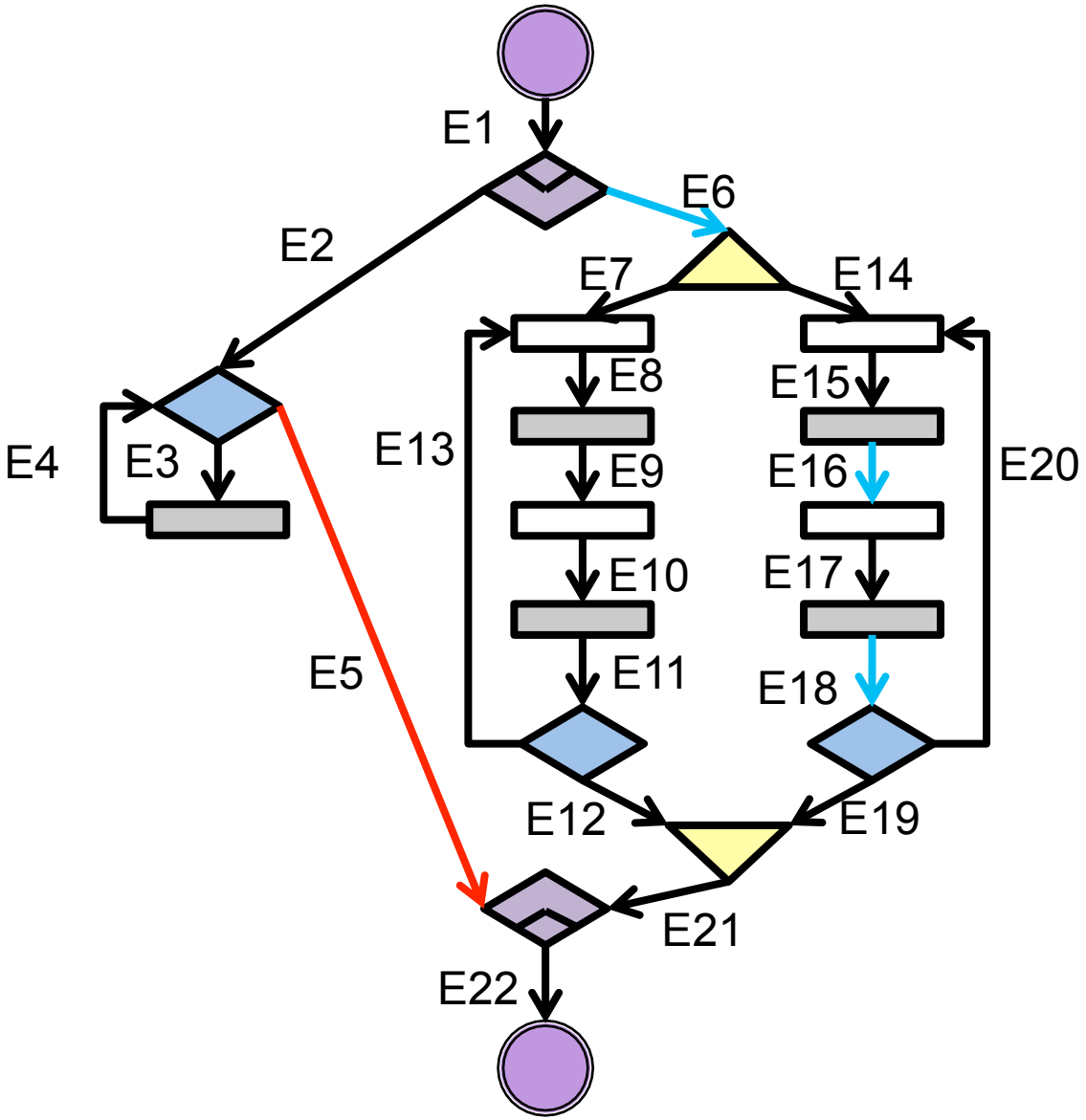
Abort End:

E22 = E5 + E21

Preemption:

E6+E9+E11 ≤ 1-E5

ILP model



Abort start:

E1 = E2

E1 ≥ E6

Abort End:

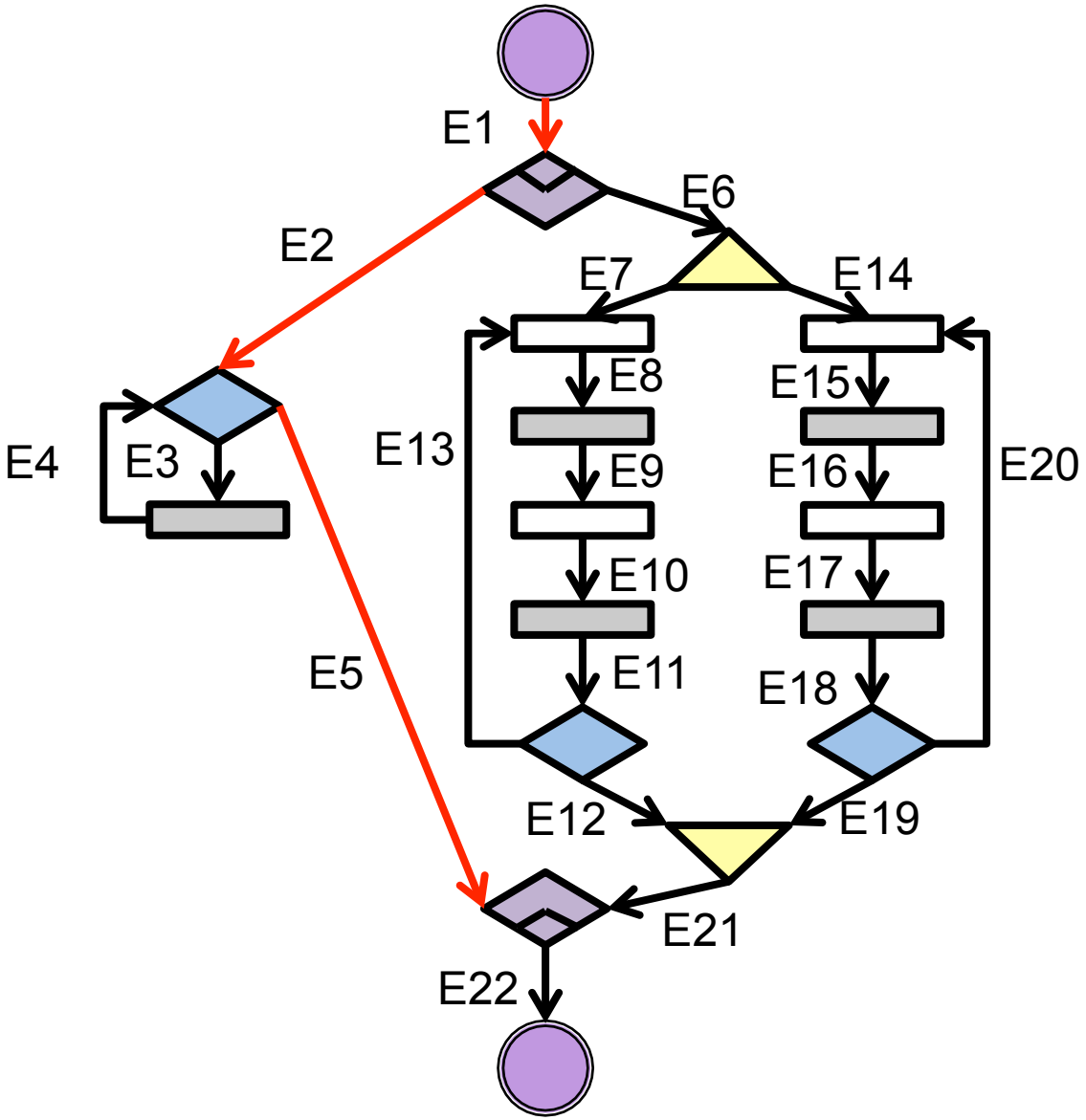
E22 = E5 + E21

Preemption:

E6+E9+E11 ≤ 1-E5

E6+E16+E18 ≤ 1-E5

ILP model



Abort start:

E1 = E2

E1 ≥ E6

Abort End:

E22 = E5 + E21

Preemption:

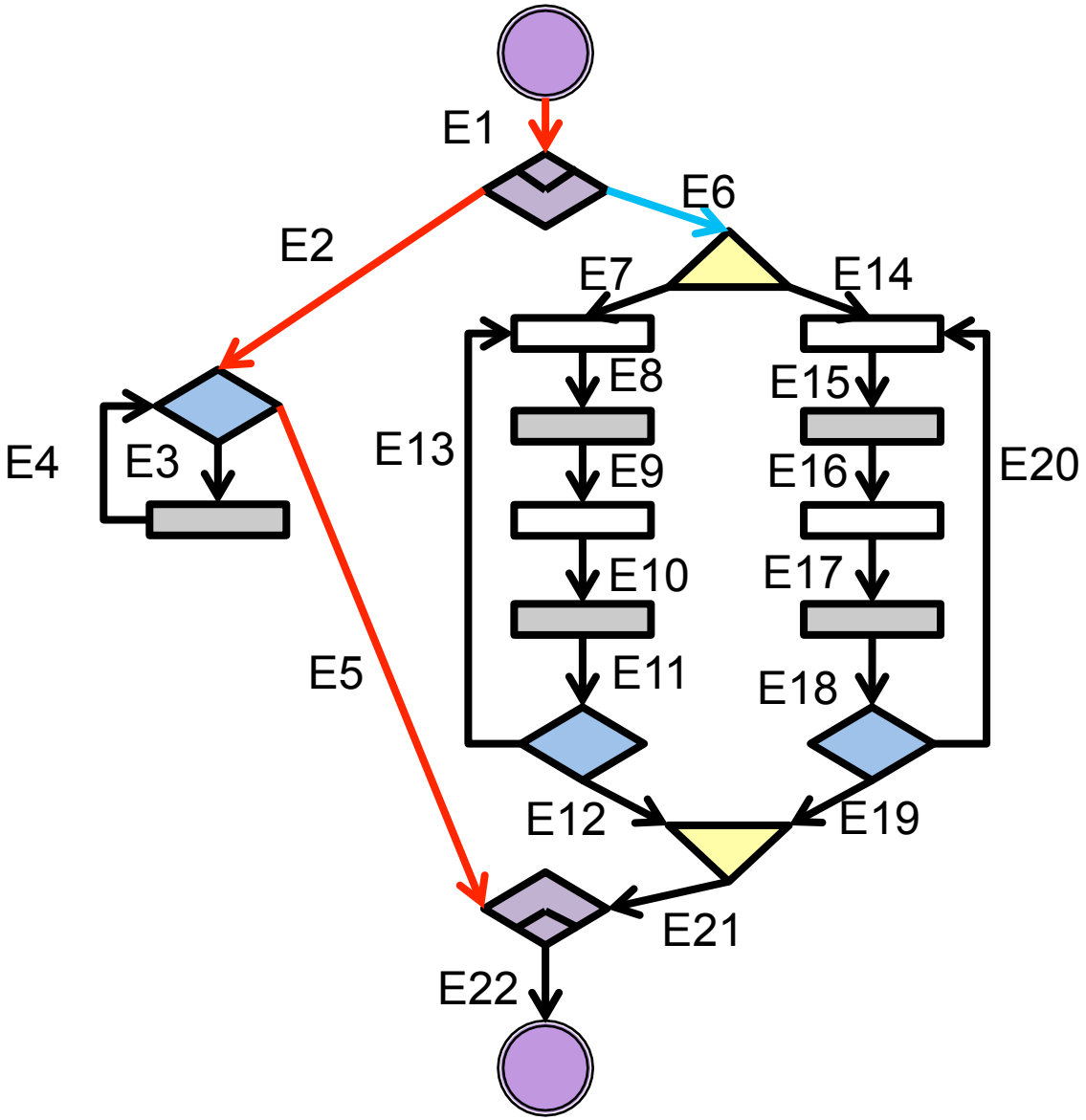
E6+E9+E11 ≤ 1-E5

E6+E16+E18 ≤ 1-E5

0 0 0 0

Preemption

ILP model



Abort start:

E1 = E2

E1 ≥ E6

Abort End:

E22 = E5 + E21

Preemption:

E6+E9+E11 ≤ 1-E5

E6+E16+E18 ≤ 1-E5

0 0 0 0

Preemption

Benchmarking

- Compared with 3 existing approaches [1,2,3]
- Conducted in 2 phases
  - Phase 1: Theoretical performance
  - Phase 2: Real-world applications
- Benchmark computer
  - Windows based
  - Quad-core 1.6 GHz CPU
  - 8 GB memory

[1] L. Ju, B. K. Huynh, S. Chakraborty, and A. Roychoudhury. *Context-sensitive timing analysis of Esterel programs, DAC, 2009.*
[2] S. Andalam, P. S. Roop, and A. Girault. *Pruning infeasible paths for tight WCRT analysis of synchronous programs, DATE 2011.*
[3] M. Kuo, R. Sinha, and P. S. Roop. *Efficient WCRT analysis of synchronous programs using reachability, DAC, 2011.*

Scalability

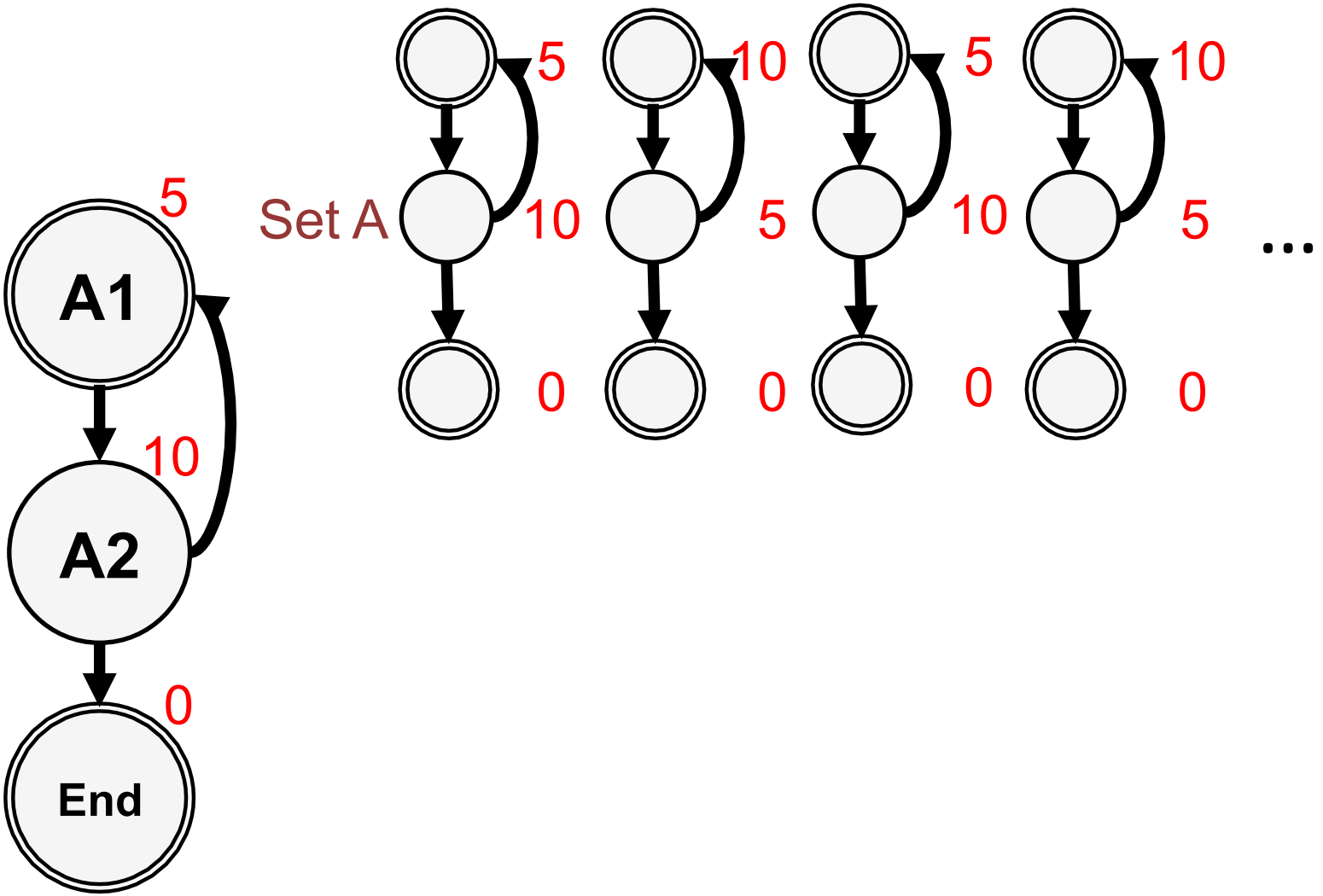
- Analysis time V.S. program states.

- Scalability of ILPc
  - Time taken for each iteration.
    - Depend on number of program states.
  - The number of iteration.
    - Depend on structure and cost distribution.

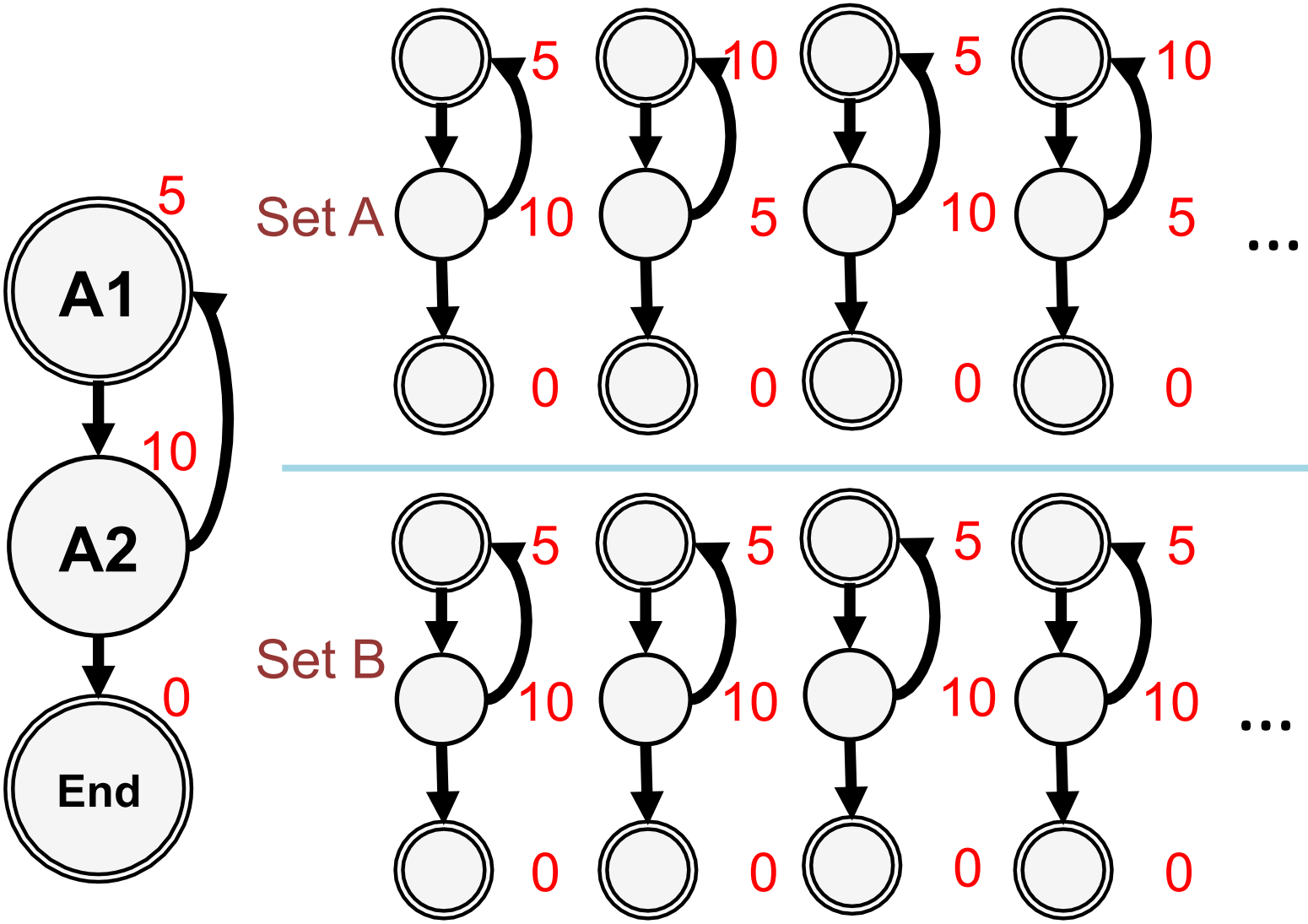
Benchmarking: Phase 1

- Two sets of benchmarks
  - Set A - Maximize the number of required iterations for a given number of program states.
  - Set B - Minimize the number of required iterations for a given number of program states.
    - Iteration = 1

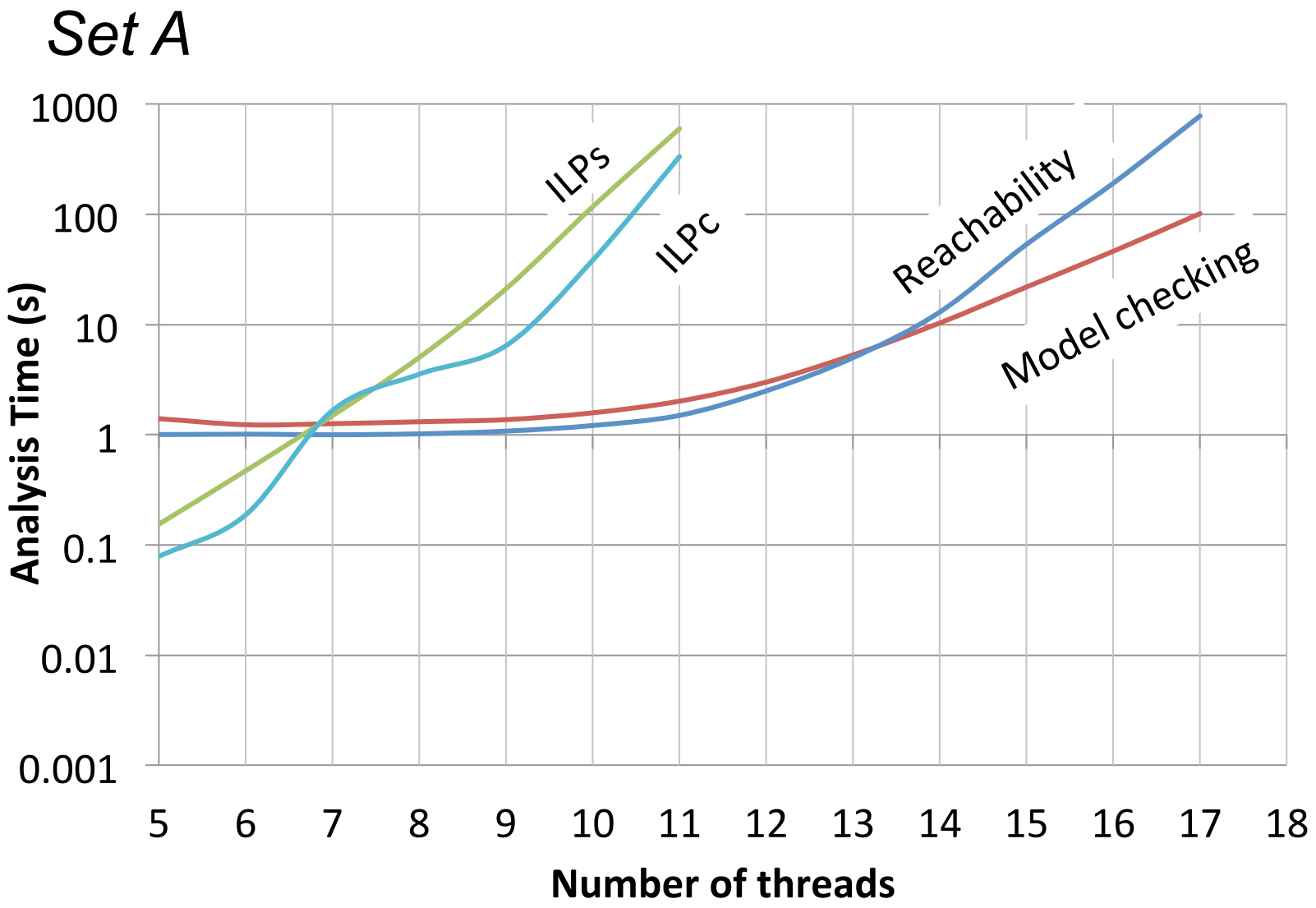
Benchmarking: Phase 1



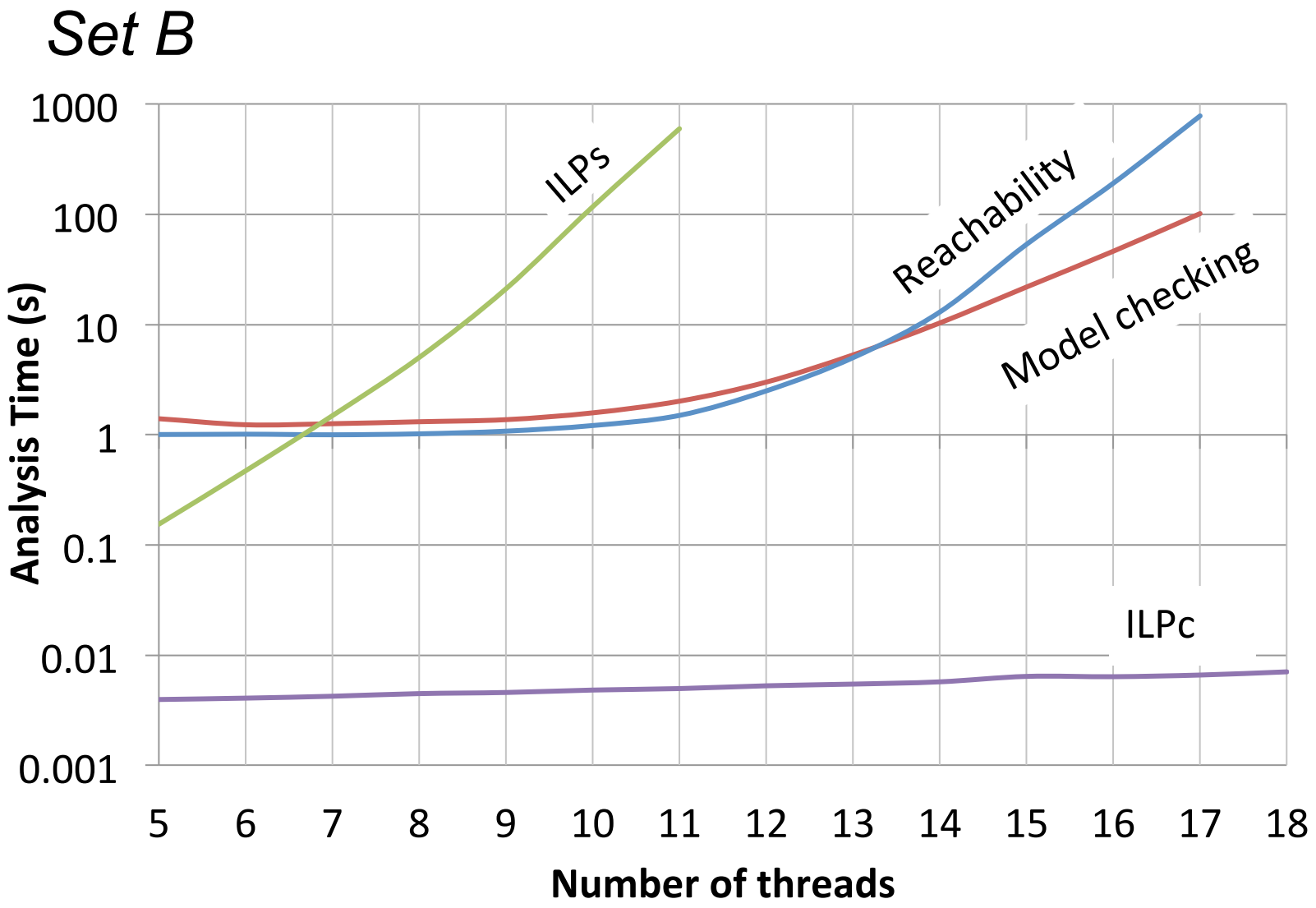
Benchmarking: Phase 1



Benchmarking: Phase 1



Benchmarking: Phase 1



Benchmarking: Phase 1

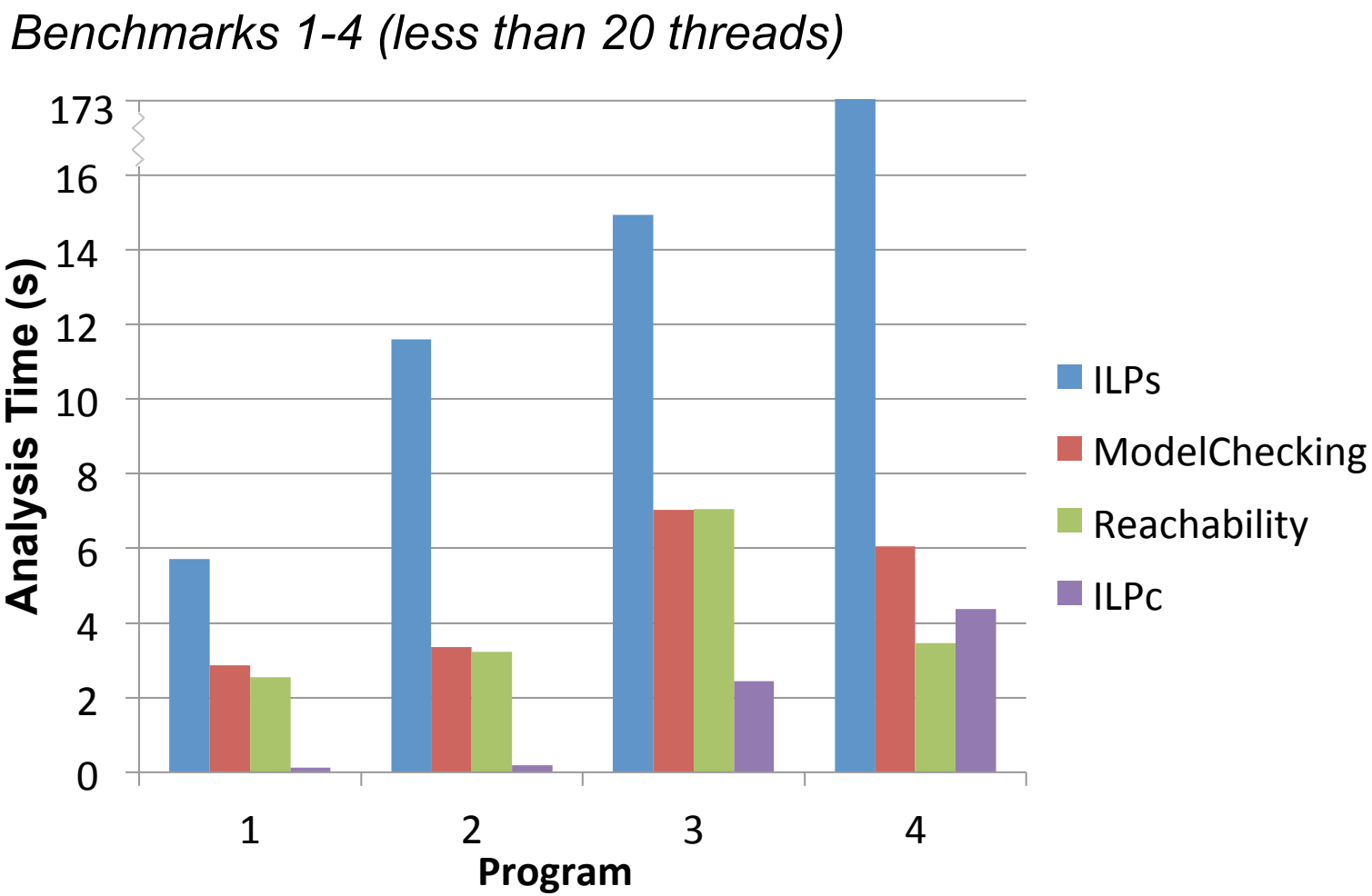
- Same precision.
- Analysis time of ILPc heavily depends on number of iterations, but not number of program states.
- On average, analysis time of ILPc should be between the worst case and best case scenarios.

Benchmarking: Phase2

	Name	WCRT	Threads	
1	ChannelProtocol	997	7	Small
2	Flasher	617	7	
3	RobotSonar	1874	7	
4	DrillStation	2751	15	
5	CruiseControl	1931	25	Large
6	RailroadCrossing	4472	30	
7	WaterMonitor	4631	40	

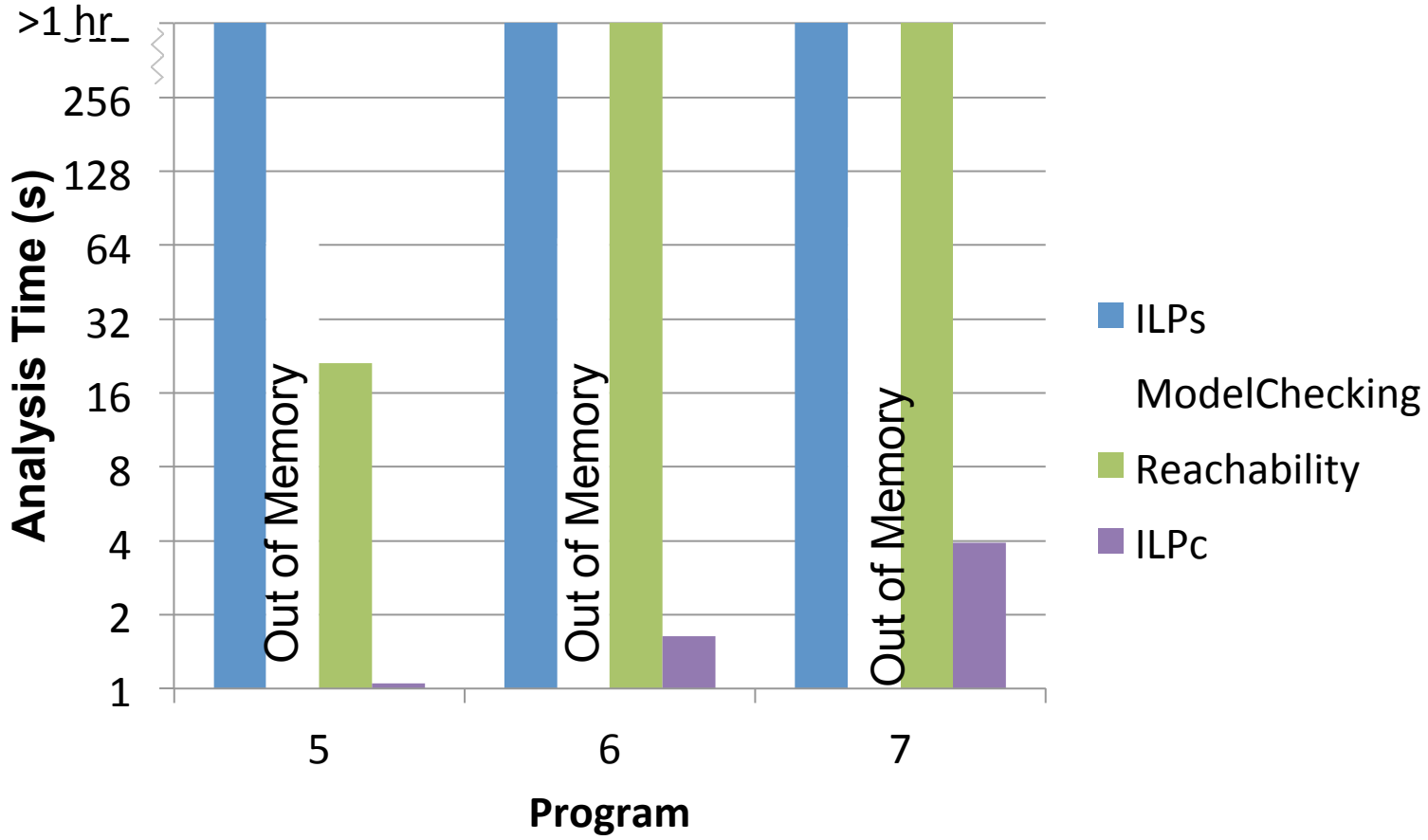
L. H. Yoong and G. D. Shaw. *Auckland function block benchmark. University of Auckland*, 2010.
www.ece.auckland.ac.nz/~pretzel/Auckland_FB_Benchmark.zip

Benchmarking: Phase2

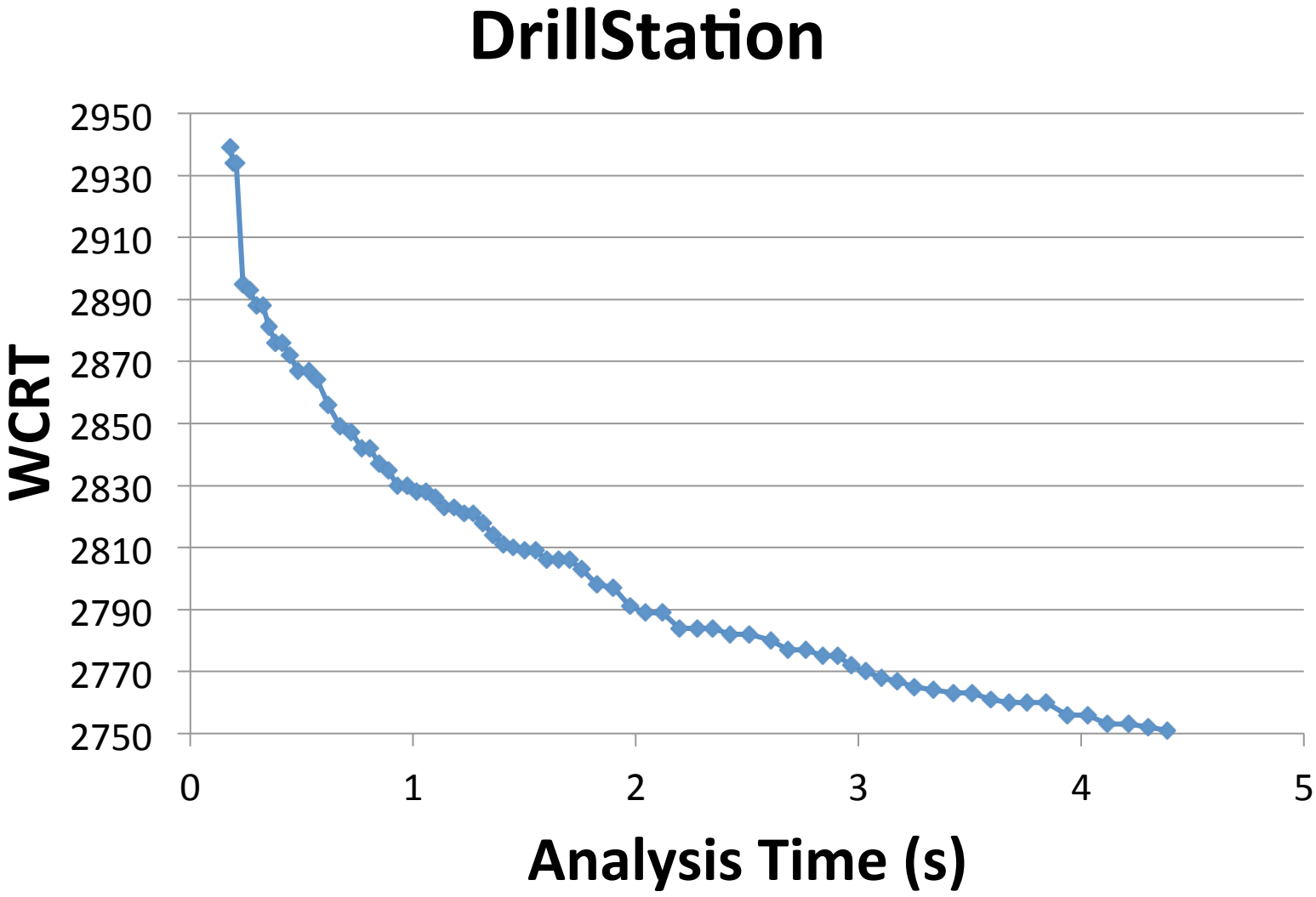


Benchmarking: Phase2

Benchmarks 5-7 (more than 20 threads)



Benchmarking: Phase2



Results summary

- Same precision as state exploration approaches
- Orders of magnitude faster compared to state exploration approaches for synchronous benchmarks for industrial automation applications

Conclusions

- We proposed ILPc that is tailored for timing analysis of concurrent programs with scalable performance.
- Precision improves with the number of iterations.
- Safe over-approximation in any iteration. If the user's precision requirement is met, analysis can be stopped.
- Compositional: adding further analysis components such as DVFS module / caches etc easier.
- Outlook:
  - User guided path infeasibility constraints through model checking.
  - Bicriteria optimization techniques for (WCRT, Energy)

Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux, and Fabienne Carrier. *Timing analysis enhancement for synchronous program.* RTNS 2013