Precision Timed Computing in the Synchronous Setting

Partha S Roop

INRIA
RHÔNE-ALPES

Synchron 2010

29 November - 4 December 2010

Acknowledgements

The following people actively contributed to the PRETzel research.
`www.ece.auckland.ac.nz/~pretzel`

- Sidharta Andalam
- Alain Girault
- Reinhard von Hanxleden
- Claus Traulsen

Layout

Research

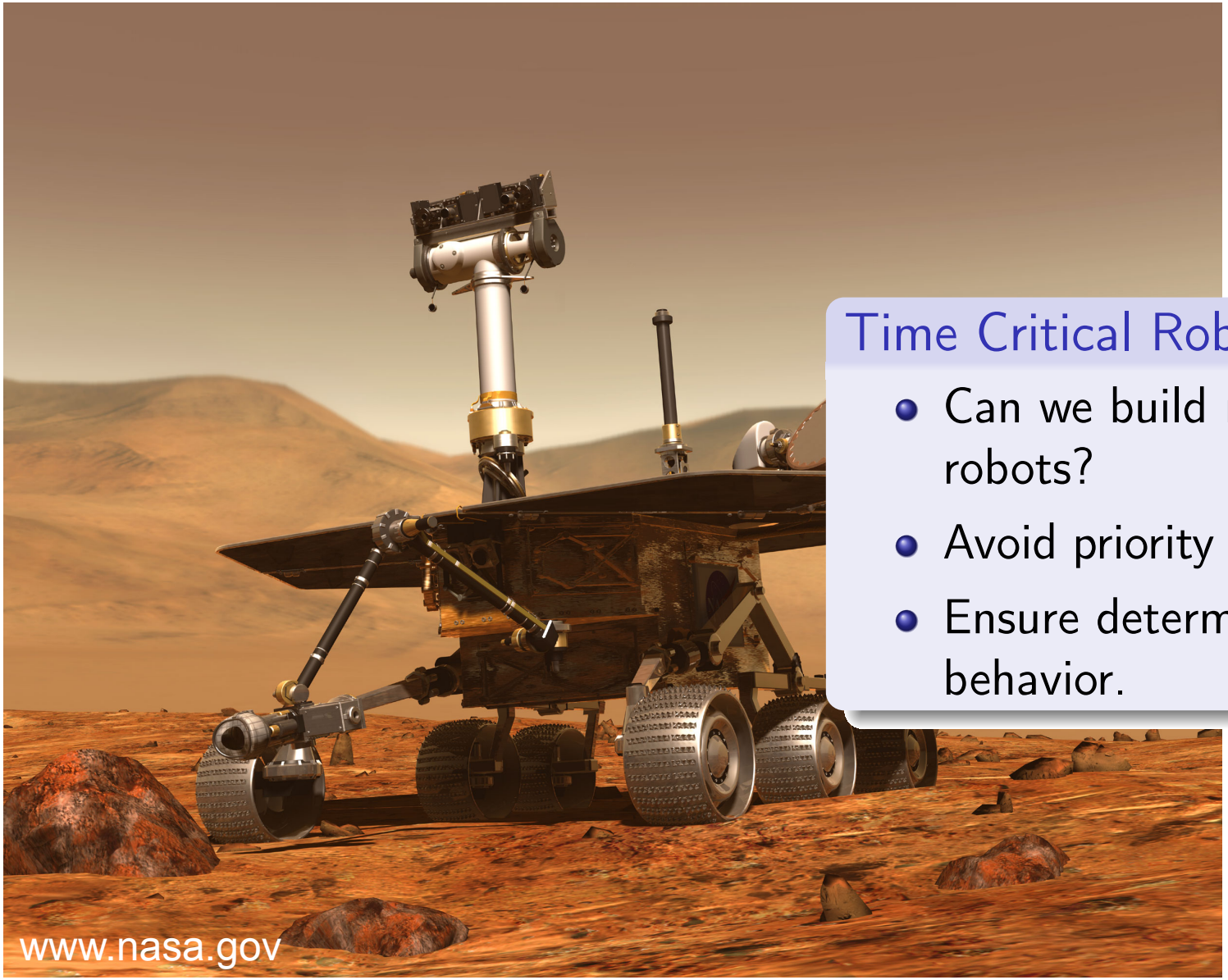
- Model checking: SoC bus protocols, web services composition, synchronous observers, real-time systems.
- Synchronous languages: Reactive and precision timed architectures, compilation, distribution, timing analysis.
- Industrial Informatics: PLC control, industrial buses, industrial PCs, standards and semantics, compilation, applications.

Teaching

- A year-4 course on embedded system design.
- We teach both the RTOS approach and the synchronous approach.

Layout

Space Exploration



Time Critical Robotics

- Can we build reliable robots?
- Avoid priority inversion.
- Ensure deterministic behavior.

Automotive Industry

Toyota

- Feb 2010: Global recall of more than 400,000 of the auto-maker's 2010 hybrid models, for problems in their anti-lock braking systems.
- Overall, more than 8 million Toyota cars have been recalled globally due to accelerator problems.

http://edition.cnn.com/2010/BUSINESS/02/09/japan.prius.recall/index.html

Real-Time System Design

Current design approaches of Time Critical Systems

- Rely on the well known theory of real-time scheduling.
- A set of tasks with timing parameters, which execute on RTOS.
- WCET derived through static analysis.
- Major issue with WCET analysis: "modern processors render WCET virtually unknowable; even simple problems demand heroic efforts" [Edwards and Lee, DAC 2007].

Motivation

Motivation

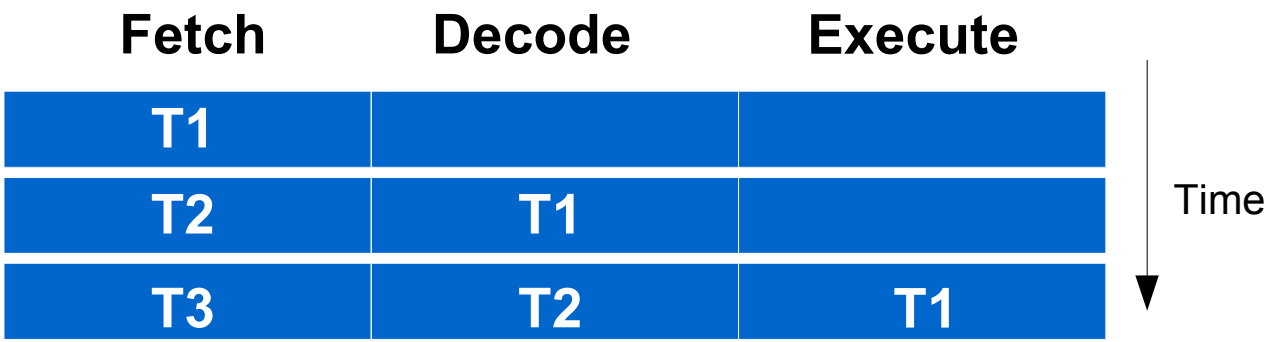
Can we rethink the link between computation and timing so that we can:

- Design Precision Timed Architectures (PRET) [Edwards and Lee, DAC 2007], where processors elicit time as a property of computation.
- The goals of PRET would be to simplify static timing analysis.
- To achieve predictability without sacrificing throughput.

Multiple Active Context System (MACS)

B. Cogswell and Z. Segall. MACS: A predictable architecture for real time systems. In Real-Time Systems Symposium. IEEE CS Press, 1991.

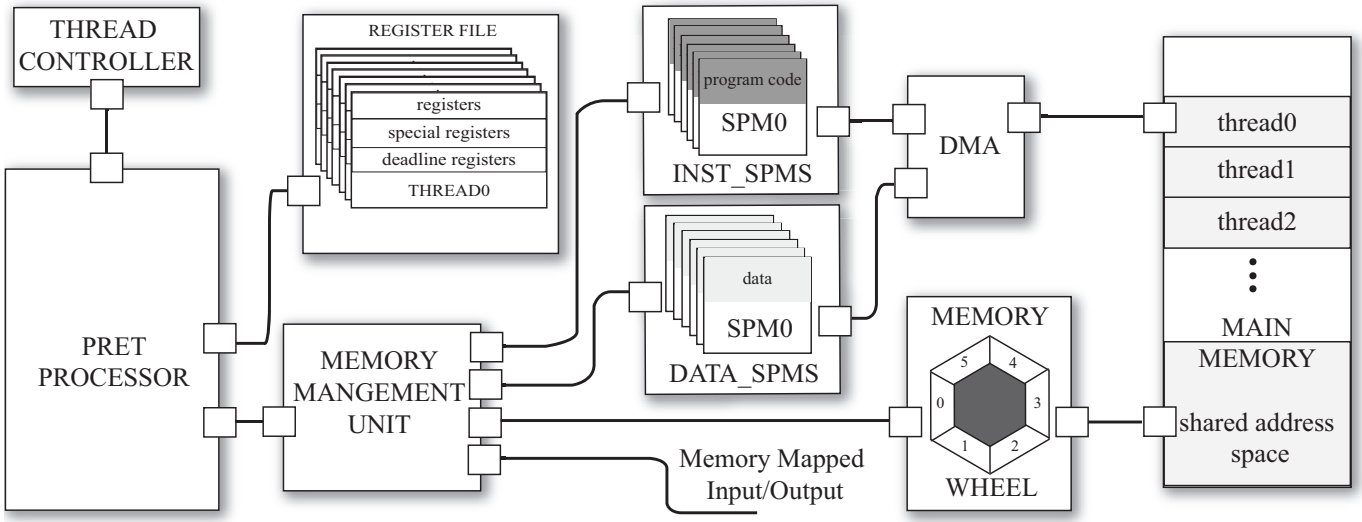
- The use of a shared pipelined processor as a possible solution to the problem of predictability and high performance in real time systems.
- Task-level parallelism is used to maintain high processor throughput while individual threads execute at a relatively slow, but very predictable rate.



Layout

The PRET Machine Design [Lickly et al., CASES'08]

- PRET machine designed based on the SPARC's ISA.
- Uses multiple active contexts and latency hiding similar to the MACS architecture.
- Memory hierarchy is replaced by statically allocated scratch-pad memories.



The PRET Programming Model

- Concurrent C programs with shared memory communication.
- Precise timing of threads using the deadi (deadline) instruction.
- Thread-safe programming by time interleaving the shared memory access.

Producer

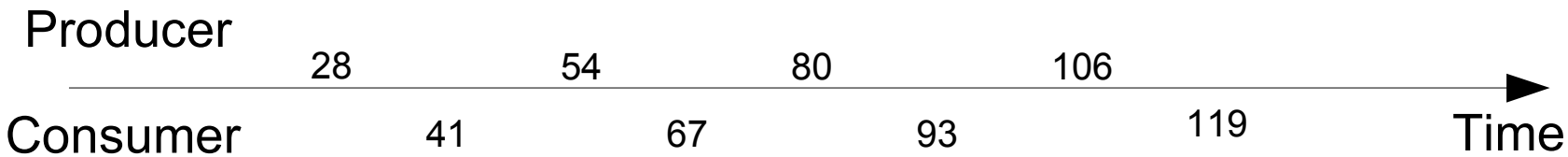
```
int main() {
  DEAD(28);
  volatile unsigned int * buf =
      (unsigned int*)(0x3F800200);
  unsigned int i = 0;
  for (i = 0; ; i++ ) {
    DEAD(26);
    *buf = i;
  }
  return 0;
}
```

Consumer

```
int main() {
  DEAD(41);
  volatile unsigned int * buf =
      (unsigned int*)(0x3F800200);
  unsigned int i = 0;
  int arr[8];
  for (i =0; i<8; i++)
    arr[i] = 0;
  for (i = 0; ; i++) {
    DEAD(26);
    register int tmp = *buf;
    arr[i%8] = tmp;
  }
  return 0;
}
```

Observer

```
int main() {
  DEAD(41);
  volatile unsigned int * buf =
      (unsigned int*)(0x3F800200);
  volatile unsigned int * fd =
      (unsigned int*)(0x80000600);
  unsigned int i = 0;
  for (i = 0; ; i++ ) {
    DEAD(26);
    *fd = *buf;
  }
  return 0;
}
```



Layout

Reactive Processors

- Alternative platforms for reactive embedded systems.
- Use ISA support for environment interaction instead of interrupts.
- Have been used for direct execution of Esterel [www.ece.auckland.ac.nz/~proo003/ReactiveProcessors.php].

Features	***Reactive Processors***	***Conventional Processors***
Execution progression	Evolves in discrete instants separated by "tick delimiting instructions"	Evolves continuously
Preemption	Accomplished through event reaction block with implicit priority resolution and context switching in hardware	Accomplished through interrupt mechanism requiring explicit priority resolution, context saving and restoration in software
Concurrency	Synchronous parallel execution and broadcast communication between threads	Asynchronous execution requiring explicit message passing/rendezvous for communication between threads
View of the environment	Changes at discrete instants. Inputs are latched at the beginning and outputs are sustained till the end of a "tick".	Changes continuously Inputs can be read at any time, and outputs can be sustained for any duration.

Reactive Processors

- Alternative platforms for reactive embedded systems.
- Use ISA support for environment interaction instead of interrupts.
- Have been used for direct execution of Esterel [www.ece.auckland.ac.nz/~proo003/ReactiveProcessors.php].

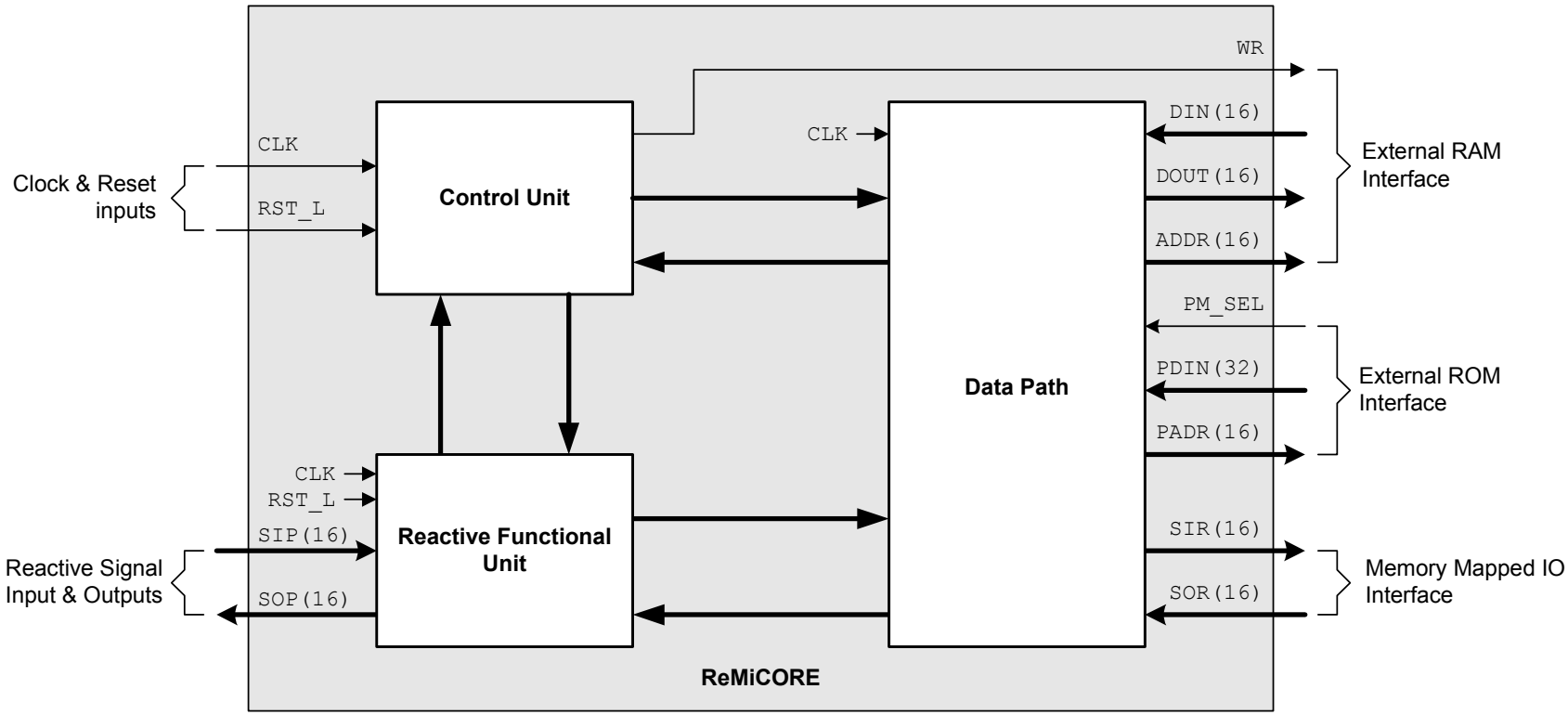
Features	*Reactive Processors*	*Conventional Processors*
Execution progression	Evolves in discrete instants separated by "tick delimiting instructions"	Evolves continuously
Preemption	Accomplished through event reaction block with implicit priority resolution and context switching in hardware	Accomplished through interrupt mechanism requiring explicit priority resolution, context saving and restoration in software
Concurrency	Synchronous parallel execution and broadcast communication between threads	Asynchronous execution requiring explicit message passing/rendezvous for communication between threads
View of the environment	Changes at discrete instants. Inputs are latched at the beginning and outputs are sustained till the end of the tick.	Changes continuously Inputs can be read at any time, and outputs can be sustained for any duration.

Time triggered

Event triggered

Example: The ReMIC Processor

Example: The ReMIC Processor



Motivation for our approach

Motivation for our approach

- Berkeley-Columbia approach is based on tailored processors that is resource intensive.

Motivation for our approach

- Berkeley-Columbia approach is based on tailored processors that is resource intensive.
- This approach also mixes physical and logical time and is not portable.

Motivation for our approach

- Berkeley-Columbia approach is based on tailored processors that is resource intensive.
- This approach also mixes physical and logical time and is not portable.
- Hard to derive the values of deadlines for thread-interleaved access.

Motivation for our approach

- Berkeley-Columbia approach is based on tailored processors that is resource intensive.
- This approach also mixes physical and logical time and is not portable.
- Hard to derive the values of deadlines for thread-interleaved access.
- Reactive processors, while being predictable, can only execute pure Esterel.

Philosophy

Philosophy

- Notion of concurrency: concurrency is logical but execution is sequential very similar to synchronous languages [Benveniste'03].

Philosophy

- Notion of concurrency: concurrency is logical but execution is sequential very similar to synchronous languages [Benveniste'03].
- Notion of time: time is logical and the mapping of logical to physical time is done using static analysis of code.

Philosophy

- Notion of concurrency: concurrency is logical but execution is sequential very similar to synchronous languages [Benveniste'03].
- Notion of time: time is logical and the mapping of logical to physical time is done using static analysis of code.
- Design approach: Auckland Reactive PRET (ARPRET) architectures are designed by simple customization of soft-core processors.

Layout

Motivation for PRET-C

Precision Timed C (PRET-C)

Simple set of synchronous extensions to C for:

- light-weight multithreading in C.
- all extensions implemented as C macros.
- provides thread-safe shared memory access.
- supports predictable programming by mapping logical time to physical time through static analysis.

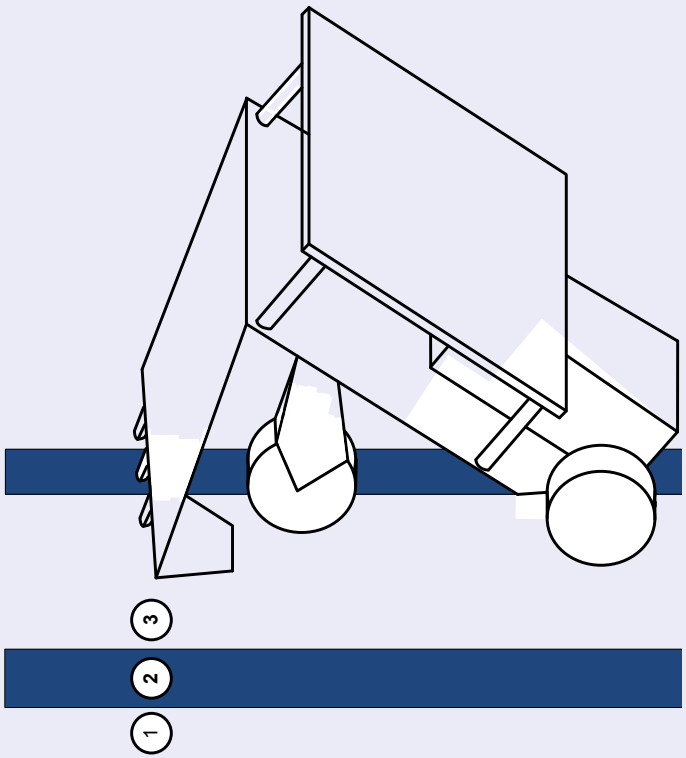
Synchronous extensions to C

Statement	Meaning
`ReactiveInput I`	declares `I` as a reactive input coming from the environment
`ReactiveOutput O`	declares `O` as a reactive output emitted to the environment
`PAR(T1,...,Tn)`	synchronously executes in parallel the `n` threads `Ti`, with higher priority of `Ti` over `Ti+1`
`EOT`	marks the end of a tick (local or global depending on its position)
`[weak] abort P when pre C`	immediately kills `P` when `C` is true in the previous instant

Table: PRET-C extensions to C.

Example: Line Following Robot

Application

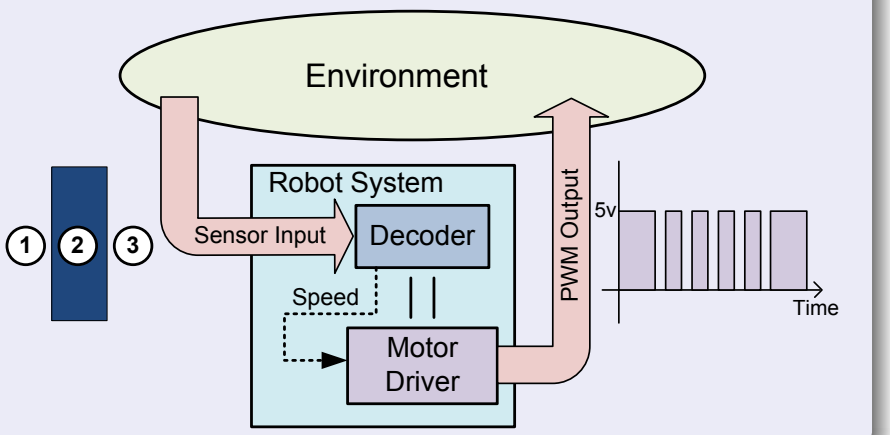


Description

- Robot has to follow the line.
- Data from 3 sensors as an input from the environment.
- Two PWM outputs controlling two motors.

Example: Line Following Robot

Overview



Description

- Three reactive inputs (sensors) from the environment.
- Two reactive outputs (PWM) controlling the motors.
- Two threads (Decoder, Motor Driver) execute concurrently.
- Speed information is shared between threads.

Example: Line Following Robot

Environment

```
ReactiveInput int reset;
ReactiveInput int sensor1,sensor2,sensor3;
...
ReactiveOutput int leftMotor; //PWM
ReactiveOutput int rightMotor; //PWM
```

Example: Line Following Robot

Environment

```
ReactiveInput int reset;
ReactiveInput int sensor1,sensor2,sensor3;
...
ReactiveOutput int leftMotor; //PWM
ReactiveOutput int rightMotor; //PWM
```

main

```
int main(void){
   while(1){
      abort
         PAR(Decoder,MotorDriver);
         if (error1) set LED 1;
         if (error2) set LED 2;
         else set both LED;
         EOT;
      when(reset==1);
} }
```

Example: Line Following Robot

```
thread Decoder(void) {
  //initialize variables;
   EOT;
   while(1) {
      process sensor data;
      store previous motor instructions;
      /*set motor control
      according to sensor data*/
      if (sensor==some value) {
         /*set goLeft,goRight
           according to the LUT*/
      }
      EOT;
      if(robot is turning){
         stuckCounter++;
      }
      else{reset stuckCounter;}
      if(stuckCounter over threshold){
         set error 2; stop robot;
      }
      if(error > 0){break;}
      else{EOT;}
} }
```

Sensor			Motor		Error
Left	Middle	Right	Left	Right	One
0	0	0	0	0	0
1	0	0	0	2	0
0	1	0	1	1	0
1	1	0	0	1	0
0	0	1	2	0	0
1	0	1	0	0	1
0	1	1	1	0	0
1	1	1	0	0	0

Table: Look-up table for tracking

Example: Line Following Robot

```
thread Decoder(void) {
  //initialize variables;
   EOT;
   while(1) {
      process sensor data;
      store previous motor instructions;
      /*set motor control
      according to sensor data*/
      if (sensor==some value) {
         /*set goLeft,goRight
           according to the LUT*/
      }
      EOT;
      if(robot is turning){
         stuckCounter++;
      }
      }else{reset stuckCounter;}
      if(stuckCounter over threshold){
         set error 2; stop robot;
      }
      if(error > 0){break;}
      else{EOT;}
} }
```

```
thread MotorDriver(void){
  int PWMCounter = 0;
  EOT;
  while(1){
    if(error > 0){
       stop robot; break;
    }else{
       if(PWMCounter < goLeft){
         leftMotor = 1; //left high
       }else{
         leftMotor = 0; //left low
       }
       if(PWMCounter < goRight){
         rightMotor = 1; //right high
       }else{
         rightMotor = 0; //right low
       }
       PWMCounter++;
       if(PWMCounter == maxSpeed){
         PWMCounter = 0;
       }
       EOT;
} } }
```

Comparison with Esterel

Esterel

```
[
  emit A(0);
  pause;
  emit A(?B+1)
||
  emit B(?A);
  pause;
  emit B(7)
]
```

PRET-C

```
PAR(T1, T2);
...
void T1()
  A=0;
  EOT;
  A=B+1;


void T2()
  B=A;
  EOT;
  B=7;
```

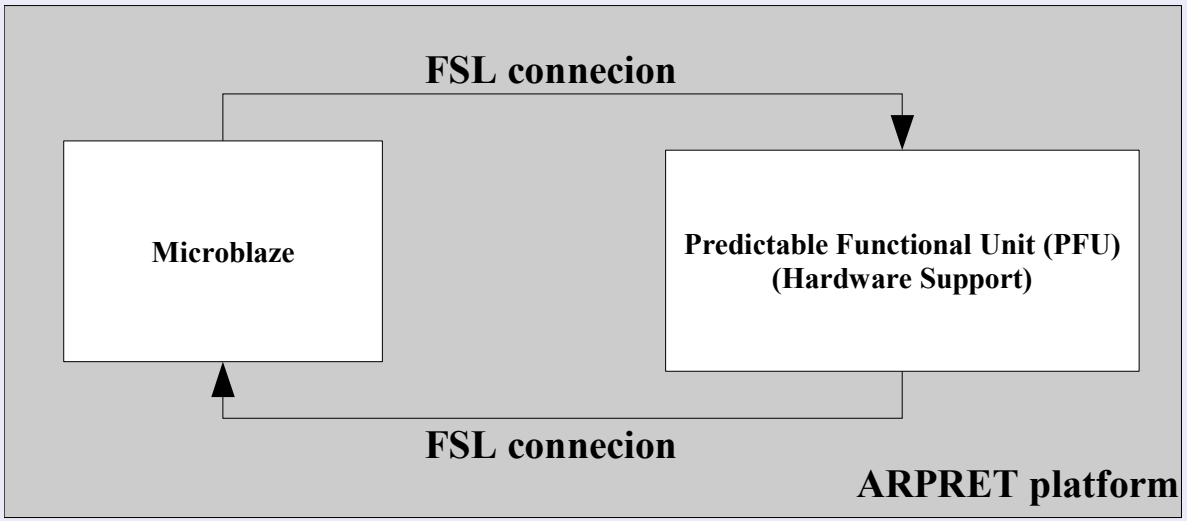
I/O

Esterel	A	0	8
	B	0	7
PRET-C	A	0	1
	B	0	7

Layout

Hardware support

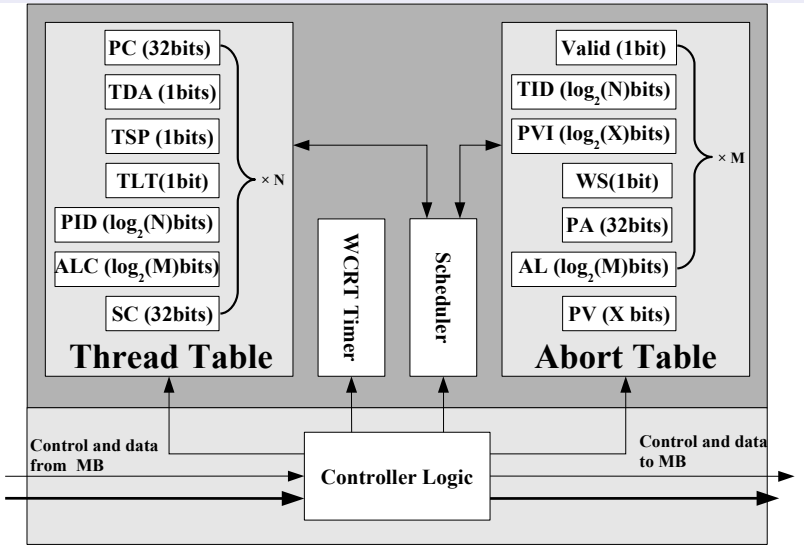
ARPRET



Hardware extension (PFU) to the Microblaze (GPP) in order to achieve better throughput while simplifying WCRT analysis. The FastSimplex Link (FSL) provides a predictable communication.

Hardware support

Predictable Functional Unit (PFU)



Thread Table stores:

- priority, local tick.
- alive, suspension.
- spawn count.
- parent ID.

Abort Table stores:

- type of preemption (Weak/Strong)
- nesting of preemptions.
- monitoring signal.
- preemption address.

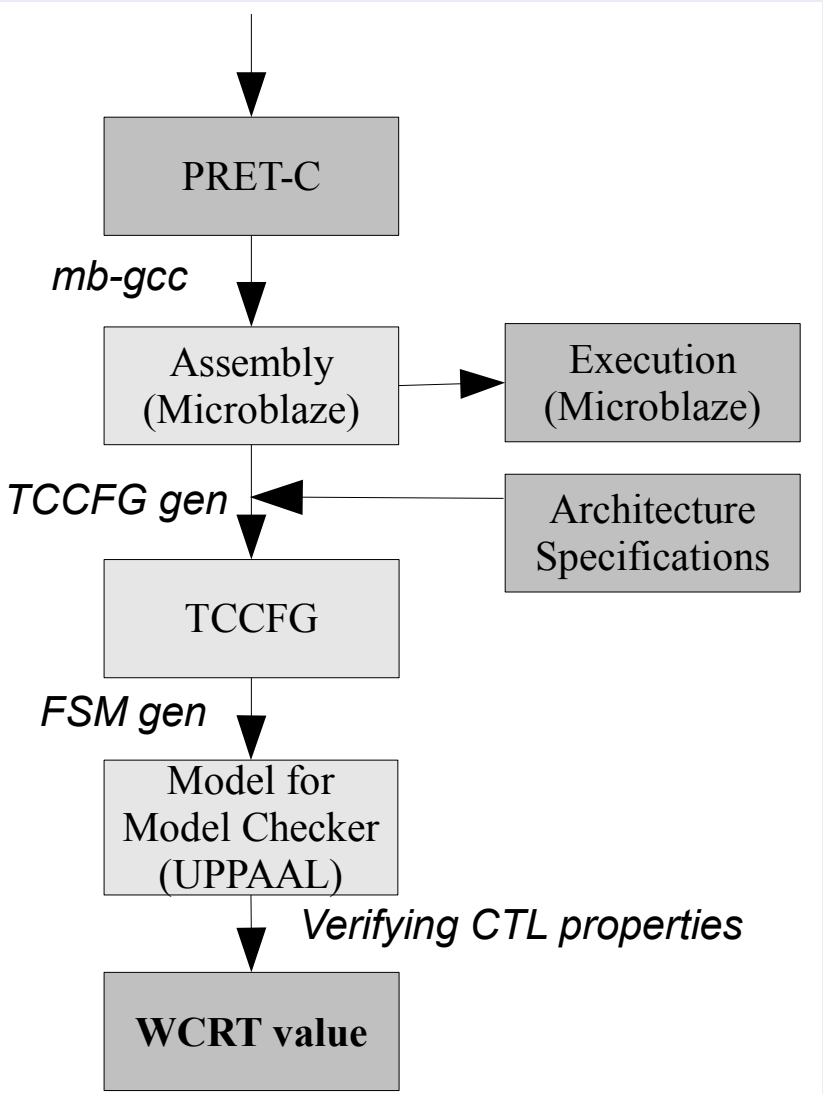
Layout

Design Flow: PRET-C to WCRT

Stages

1. PRET-C to Assembly: standard gcc based compilers can be used.
2. Assembly to TCCFG : our code analyser.
3. TCCFG to Model Checker : our FSM generator (XML).
4. CTL temporal logic property checking: bounded integer checking.

Overview



Mapping PRET-C to TCCFG

PRET-C

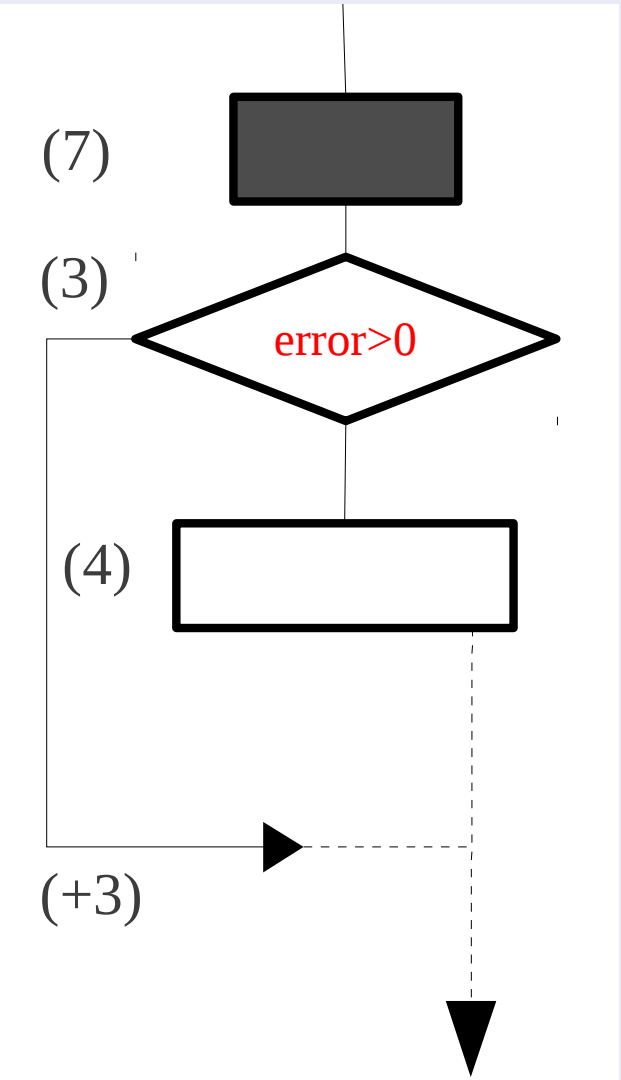
```
thread Decoder() {
  ......
     EOT;
     if(error>0){
        ...
     }
  ......
}
```

Execution cost 3

Assembly

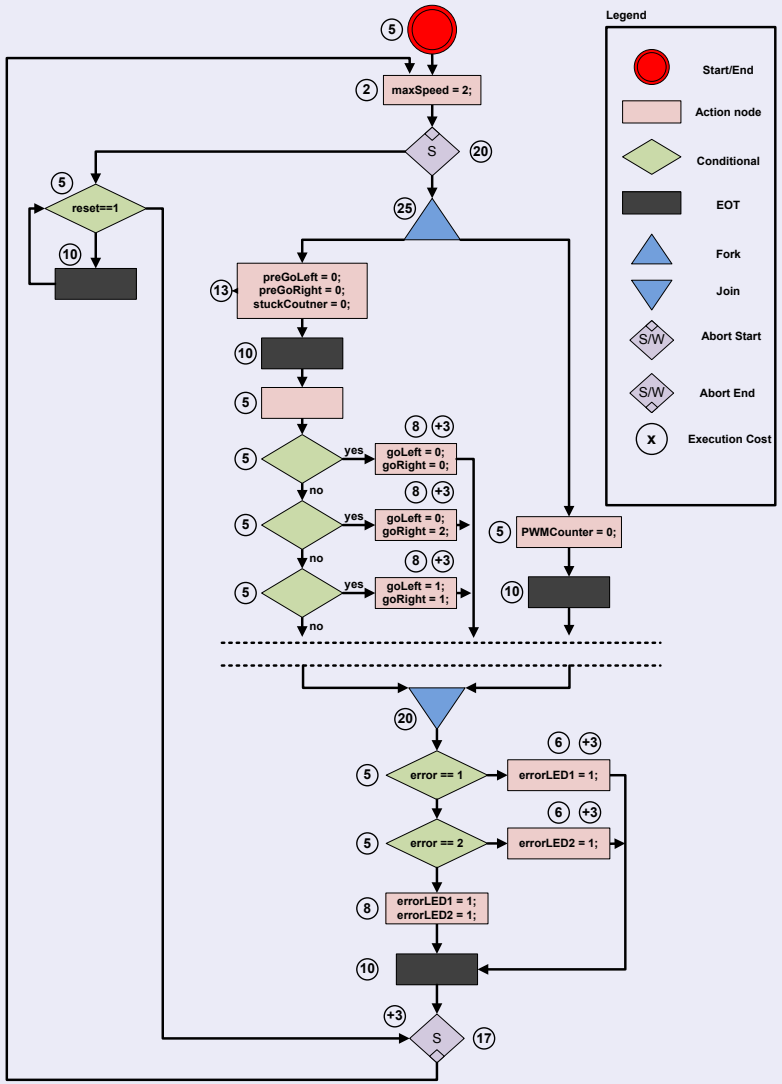
```
lwi r3,r0,error
ble r3,$L50
##...
....
$L50:
##...
```

TCCFG



Timed Concurrent Control Flow Graph

TCCFG



Intermediate format captures:

- concurrent control flow.
- preemption using checkaborts.
- execution cost of every node.

Overview of the solution

Stages

1. PRET-C: simple synchronous extension to C (using macros).

Code

```
void main() {
  while(1) {
    abort
      PAR(sampler,display);
    when pre (reset);
    EOT;
  }
}
```

Overview of the solution

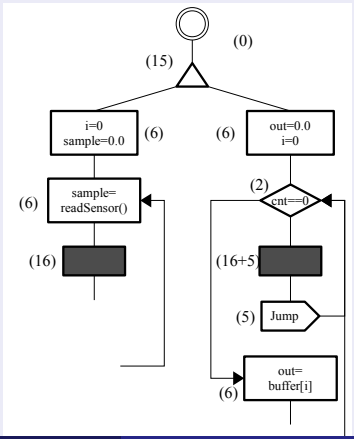
Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG : intermediate format.

Code

```
void main() {
  while(1) {
    abort
      PAR(samp
    when pre (
    EOT;
  }
}
```

TCCFG



Overview of the solution

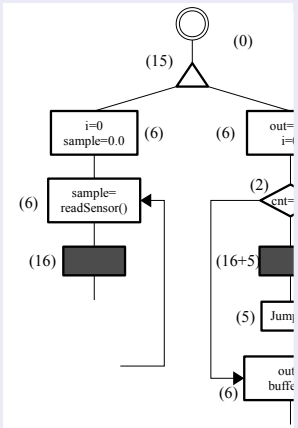
Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG : intermediate format.
3. TFSM : FSM denoted with execution costs.

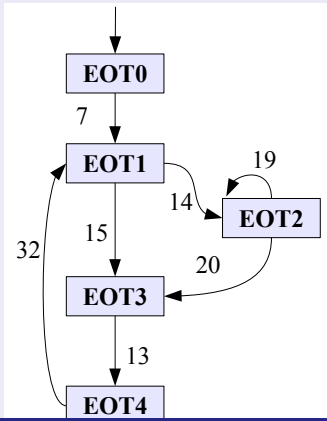
Code

```
void main() {
  while(1) {
    abort
      PAR(samp
    when pre (
    EOT;
  }
}
```

TCCFG



TFSM



Overview of the solution

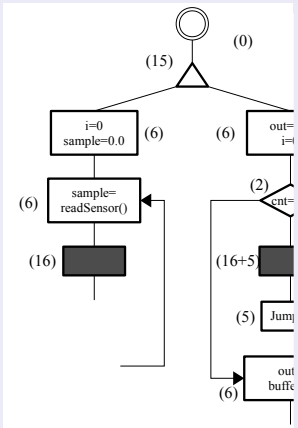
Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG : intermediate format.
3. TFSM : FSM denoted with execution costs.
4. Model Checking : calculates the WCRT based on a set of TFSMs and a safety property.

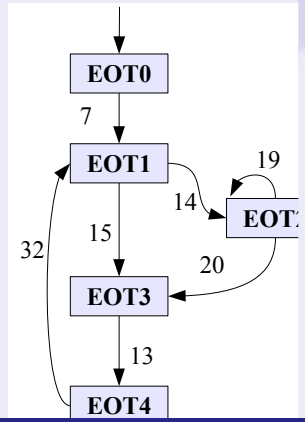
Code

```
void main() {
  while(1) {
    abort
      PAR(samp
    when pre (
    EOT;
  }
}
```

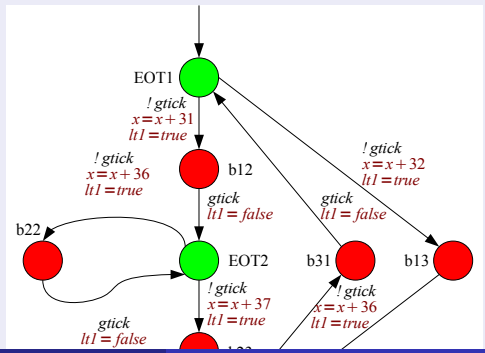
TCCFG



TFSM



Model Checker (UPPAAL)



Overview of the solution

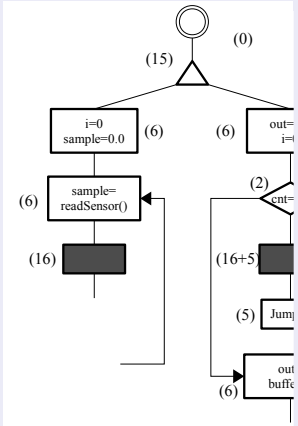
Stages

1. PRET-C: simple synchronous extension to C (using macros).
2. TCCFG : intermediate format.
3. TFSM : FSM denoted with execution costs.
4. Model Checking : calculates the WCRT based on a set of TFSMs and a safety property.

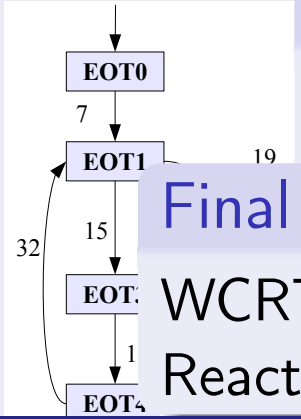
Code

```
void main() {
  while(1) {
    abort
      PAR(samp
    when pre (
    EOT;
  }
}
```

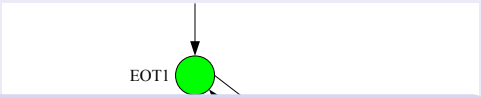
TCCFG



TFSM



Model Checker (UPPAAL)



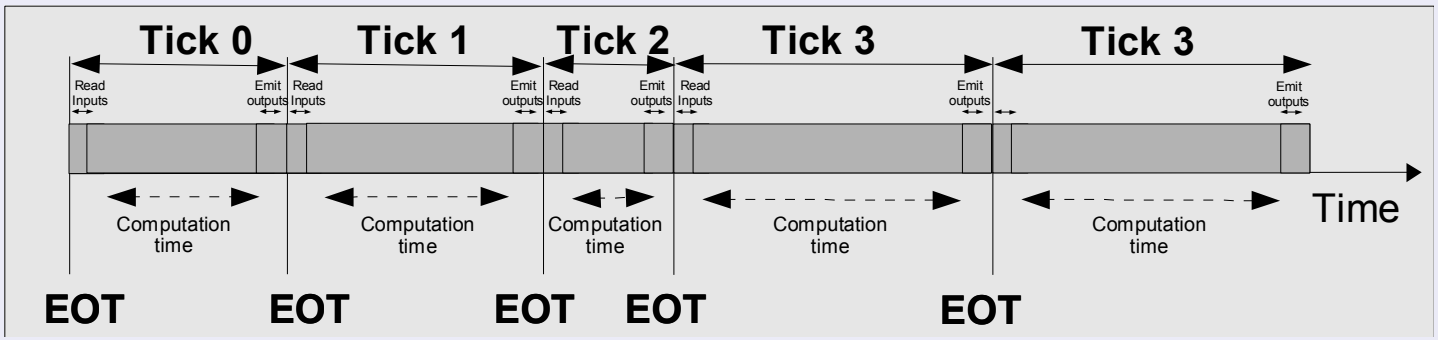
Final Output

WCRT analysis of the Reactive function.

Problem and motivation

main

```
loop each tick{
  ReadInptuts();
  ReactiveFunction();
  EmitOutputs();
}
```



How to determine the "tight/optimal" worst case tick length of the Reactive Function?

Related Work

Current approaches

Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P'07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.

Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P'07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- WCRT algebra [Mendler et al., DATE'09]: max-plus algebra with support for infeasible path pruning.

Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P'07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- WCRT algebra [Mendler et al., DATE'09]: max-plus algebra with support for infeasible path pruning.
- Timed KS [Logothetis et al., RTSS'03]: Synchronous program compiled into a timed Kripke structure.

Related Work

Current approaches

- MaxPlus [Boldt et al., SLA++P'07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- WCRT algebra [Mendler et al., DATE'09]: max-plus algebra with support for infeasible path pruning.
- Timed KS [Logothetis et al., RTSS'03]: Synchronous program compiled into a timed Kripke structure.
- ILP formulation [Ju et al., CODES+ISSS'08]: Esterel program mapped to C using CEC and then the C code is analysed.

Related Work

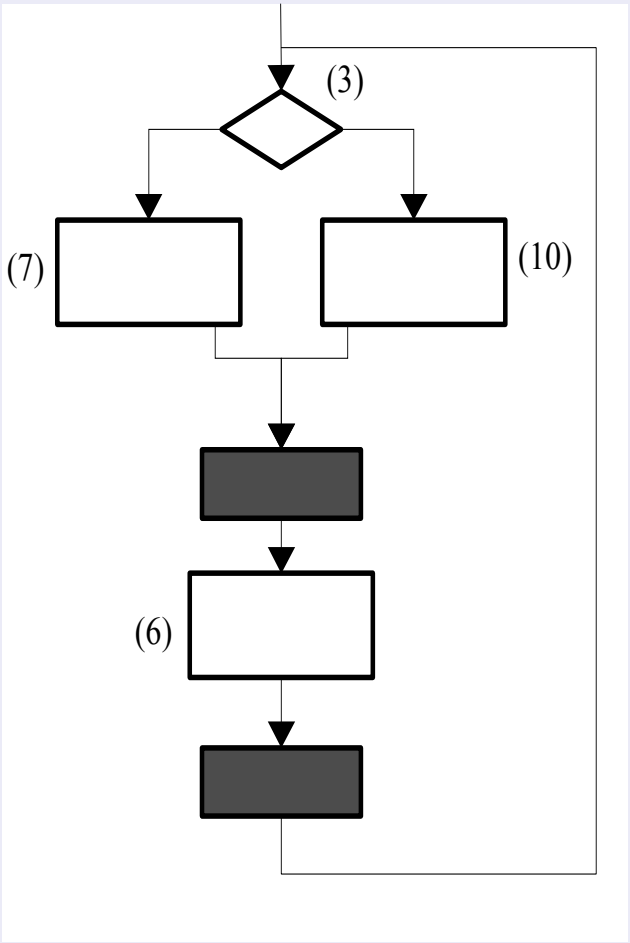
Current approaches

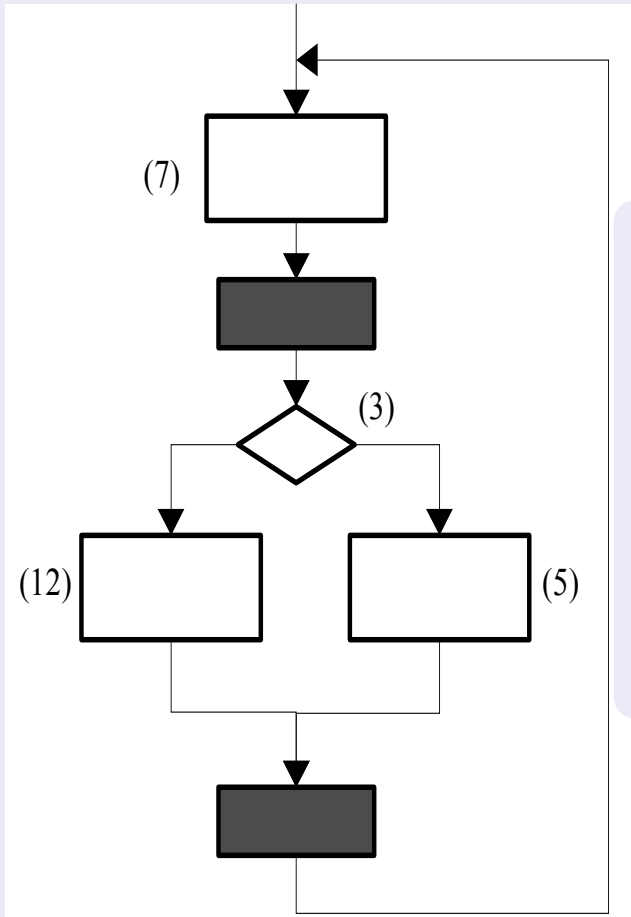
- MaxPlus [Boldt et al., SLA++P'07]: CKAG based intermediate format of reactive processor KEP that is analysed. Close to 40% overestimation.
- WCRT algebra [Mendler et al., DATE'09]: max-plus algebra with support for infeasible path pruning.
- Timed KS [Logothetis et al., RTSS'03]: Synchronous program compiled into a timed Kripke structure.
- ILP formulation [Ju et al., CODES+ISSS'08]: Esterel program mapped to C using CEC and then the C code is analysed.

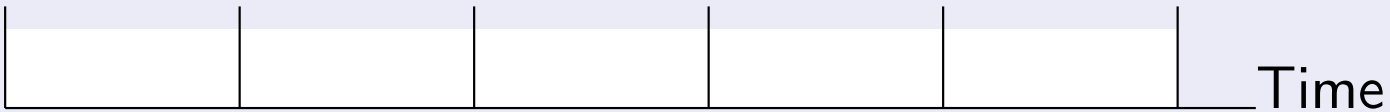
Limitation

Most approaches ignore state-dependencies while determining infeasible paths. A model checking based formulation may be used to compute the reachable state-space and this reachability analysis automatically prunes unreachable paths.

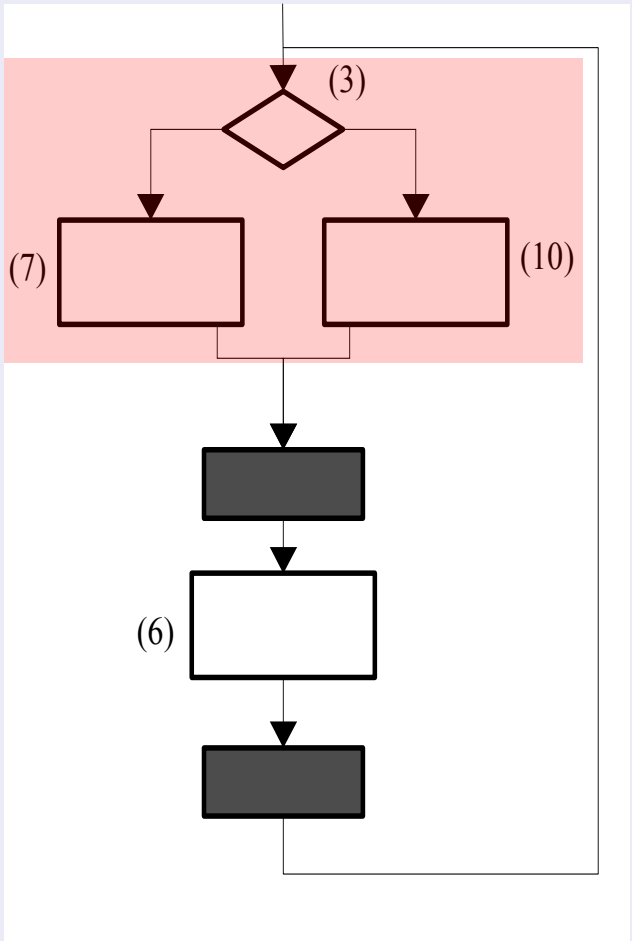
Example 1: Two simple threads

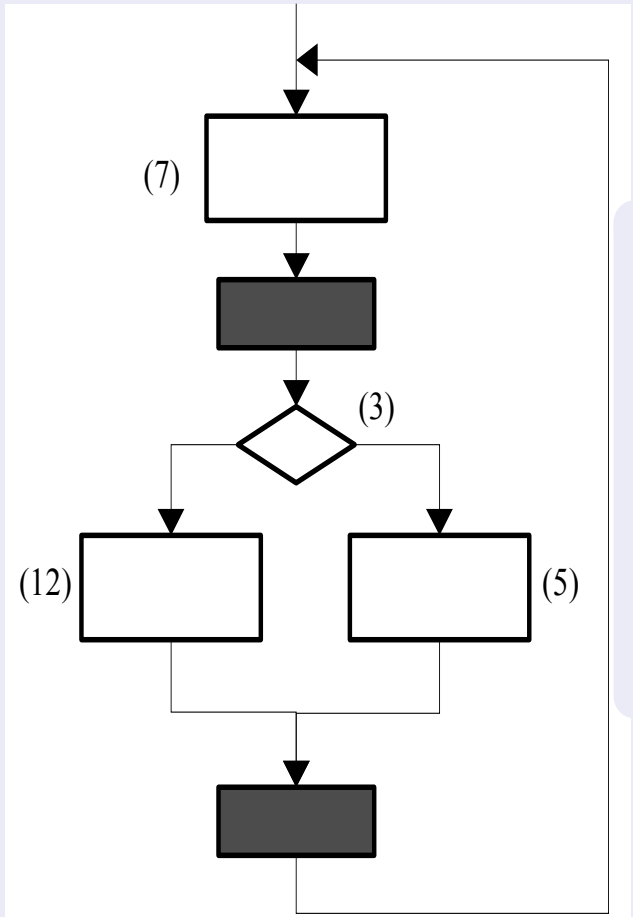


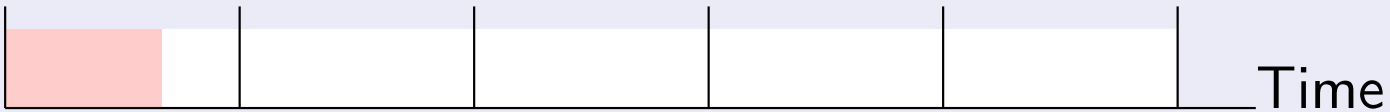




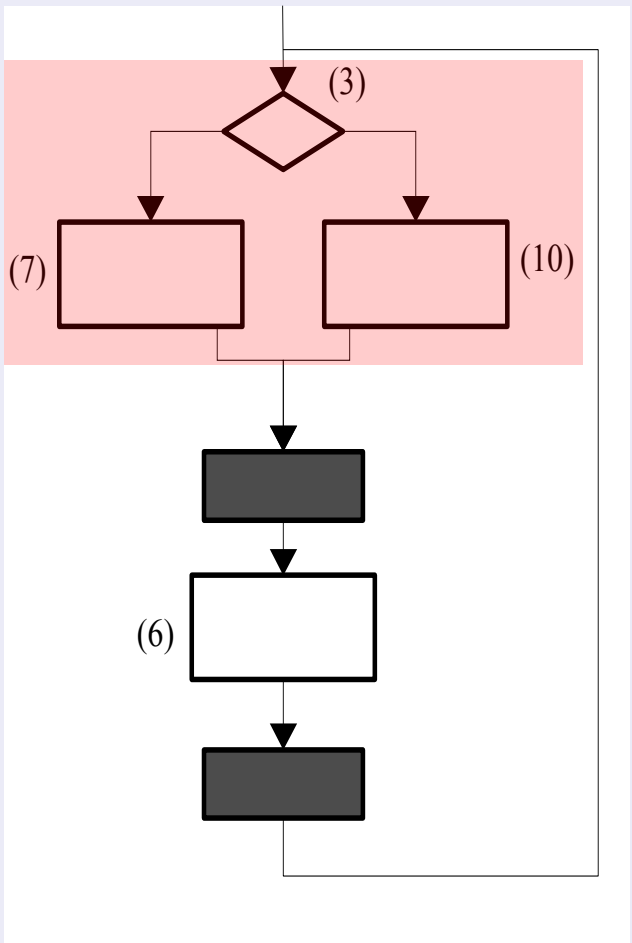
Example 1: Two simple threads

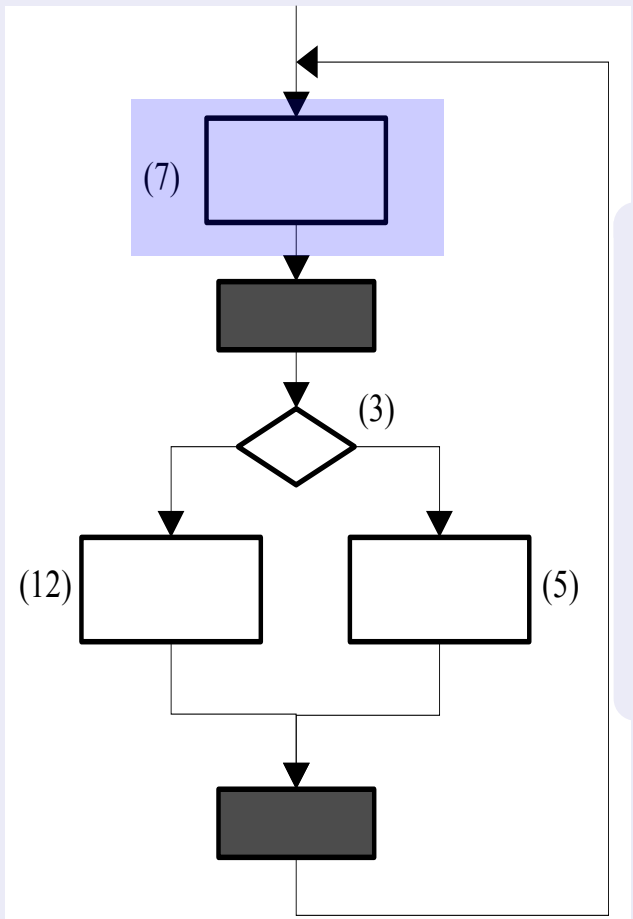


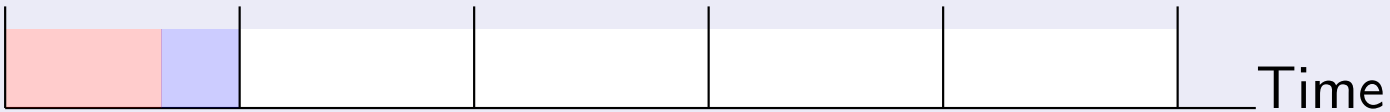




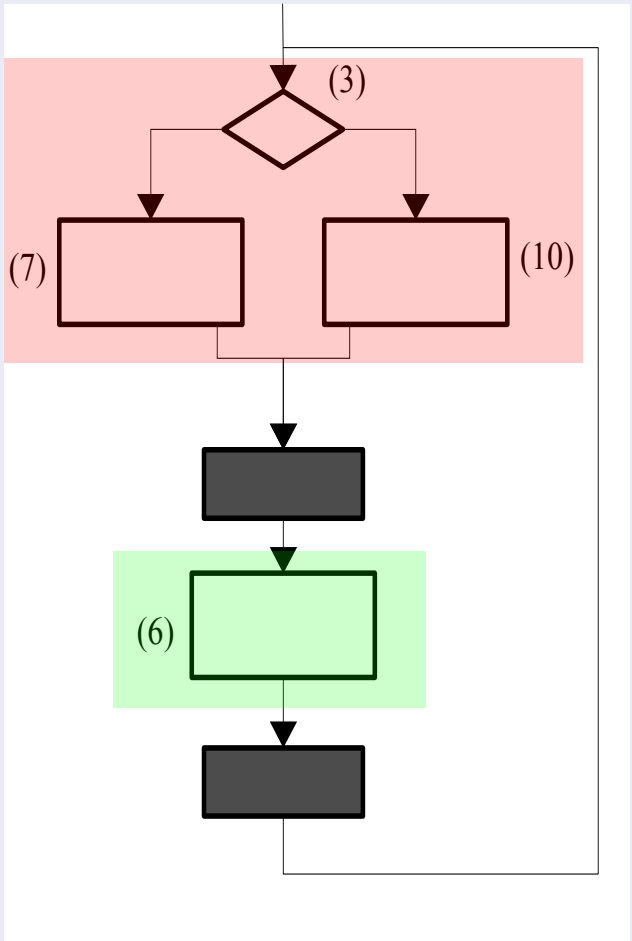
Example 1: Two simple threads

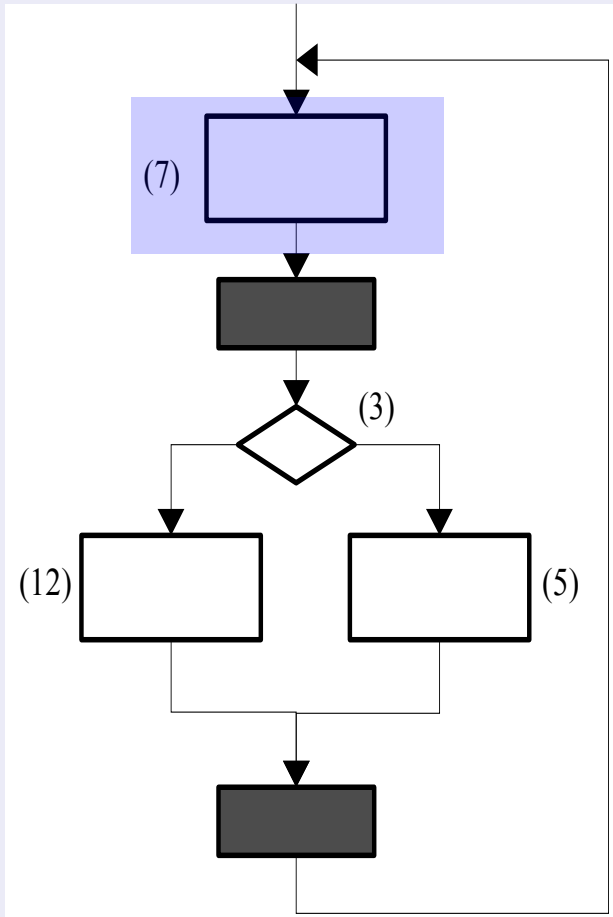


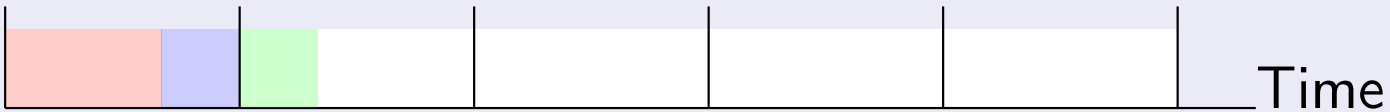




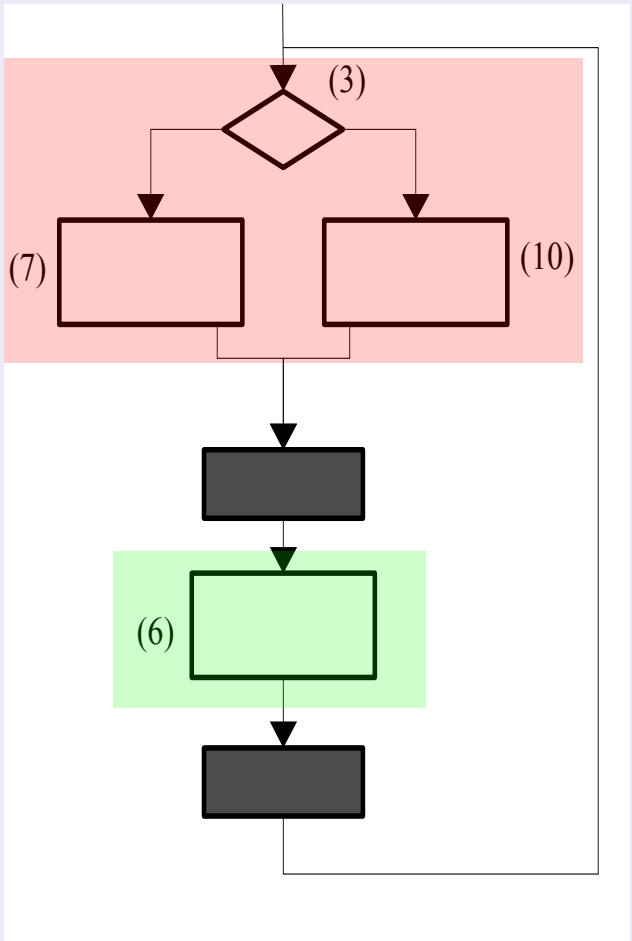
Example 1: Two simple threads

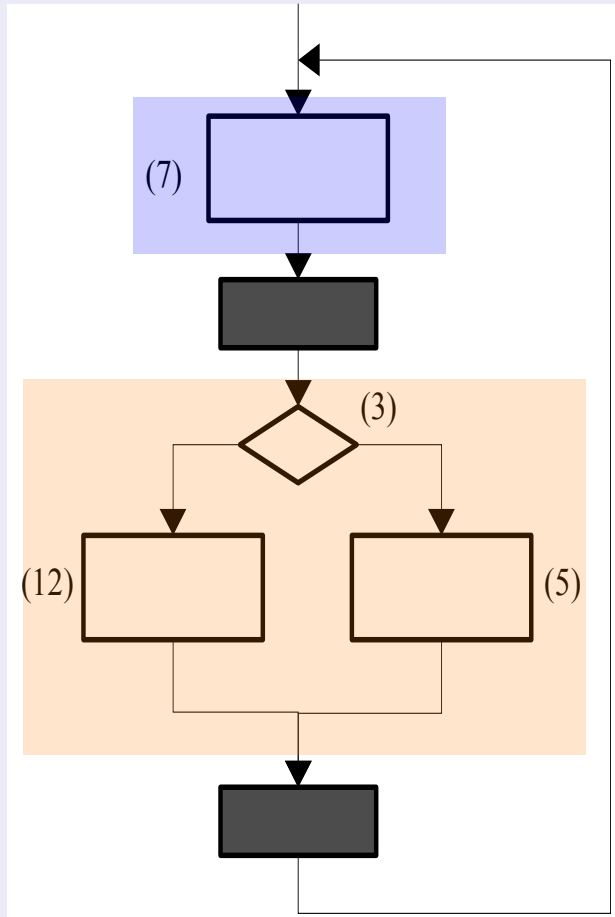


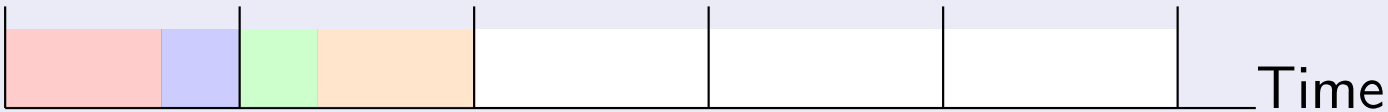




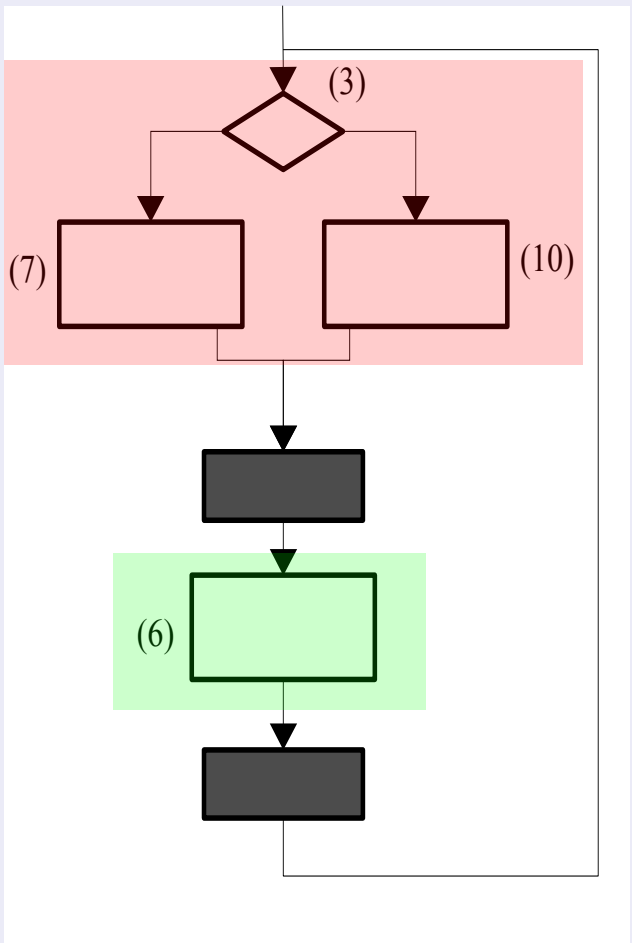
Example 1: Two simple threads

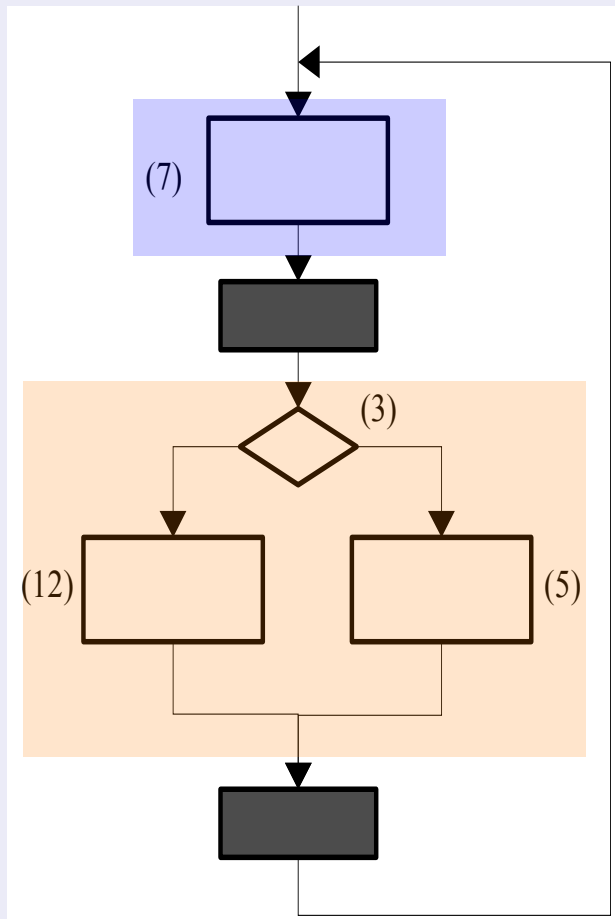


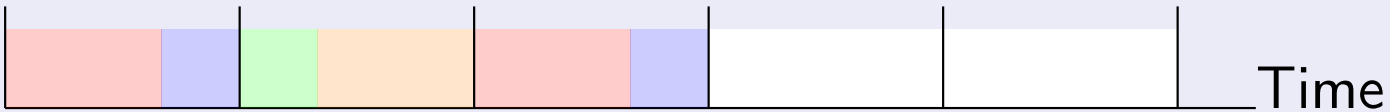




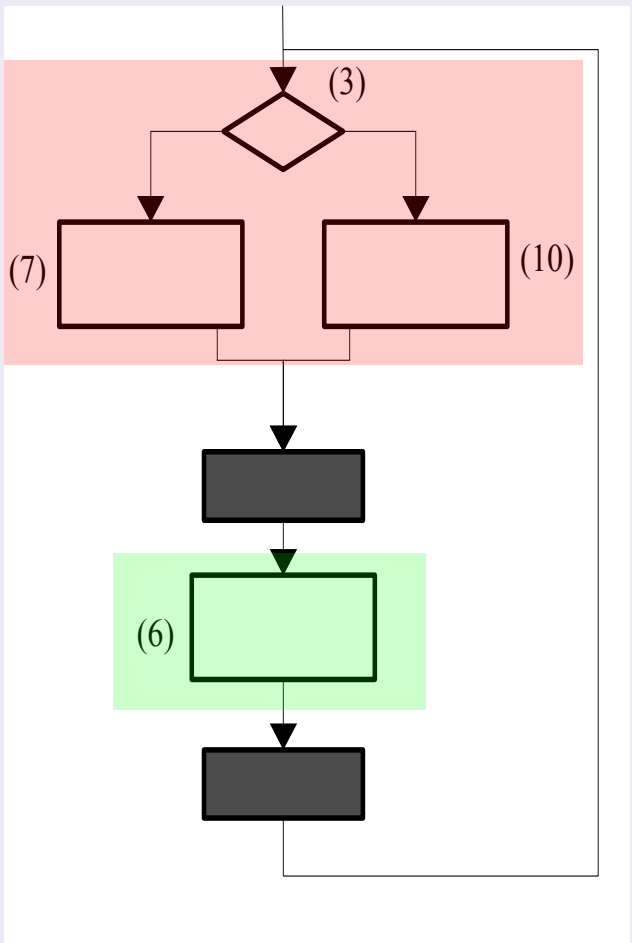
Example 1: Two simple threads

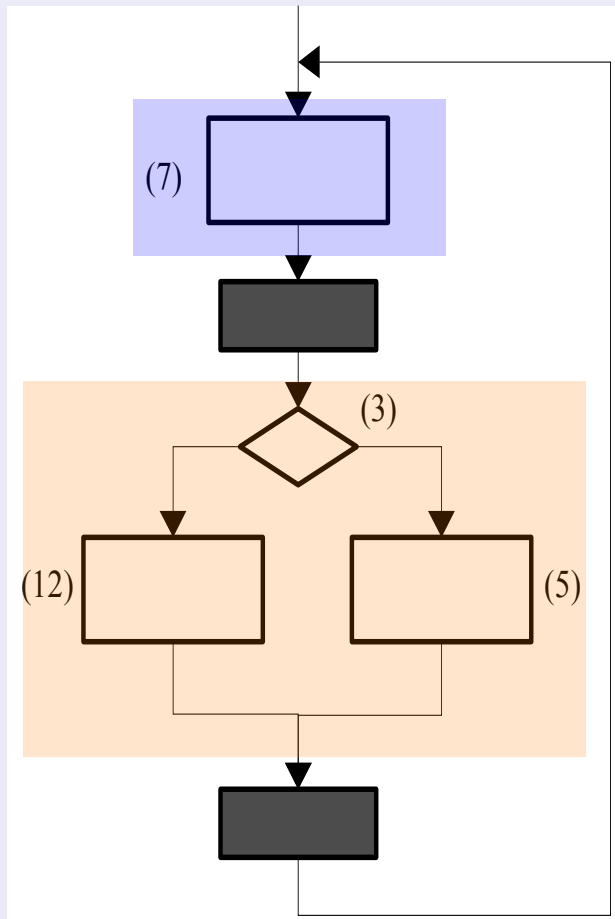


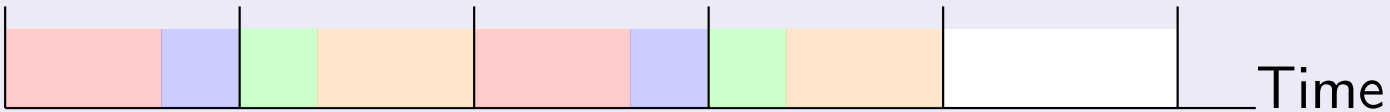




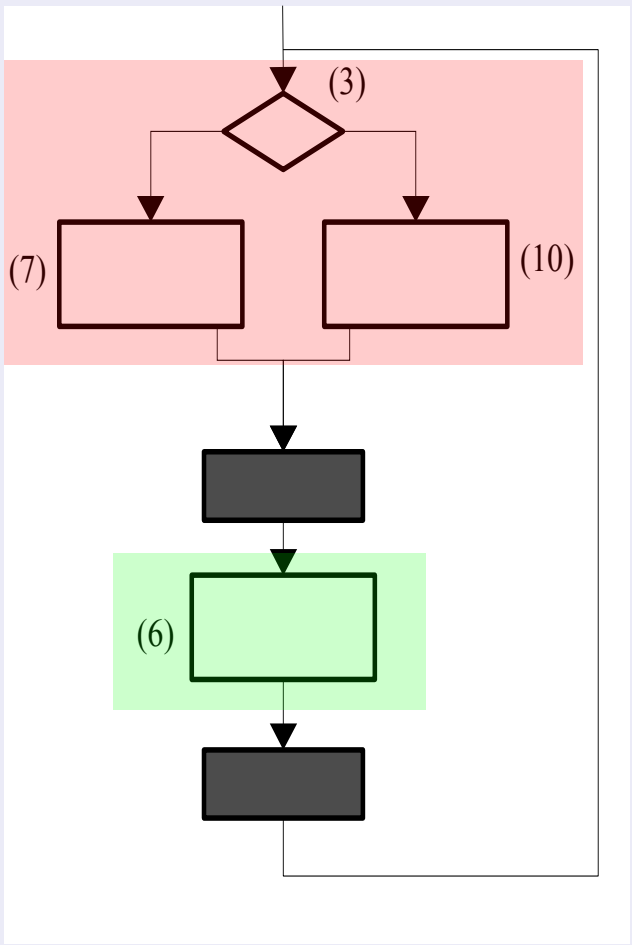
Example 1: Two simple threads

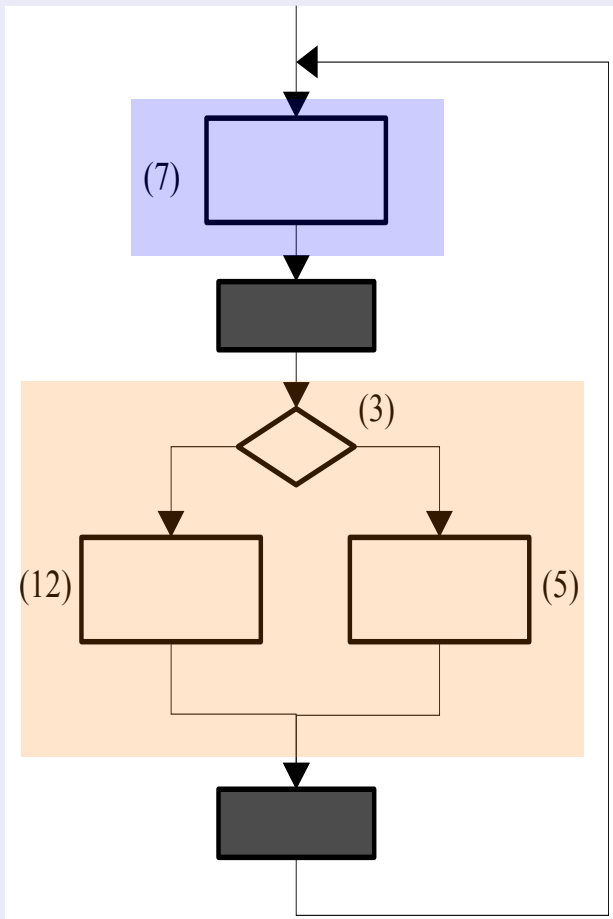


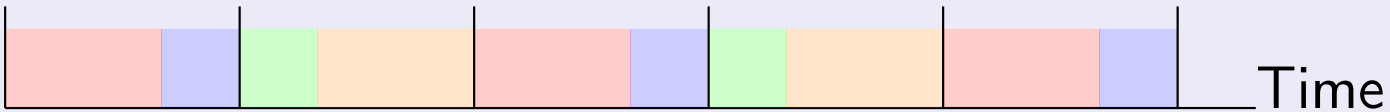




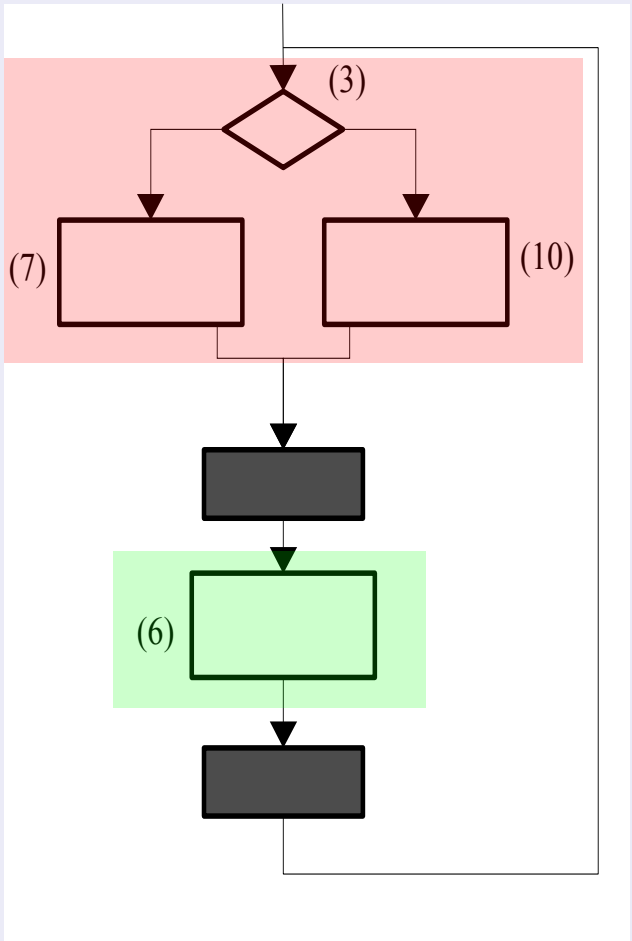
Example 1: Two simple threads

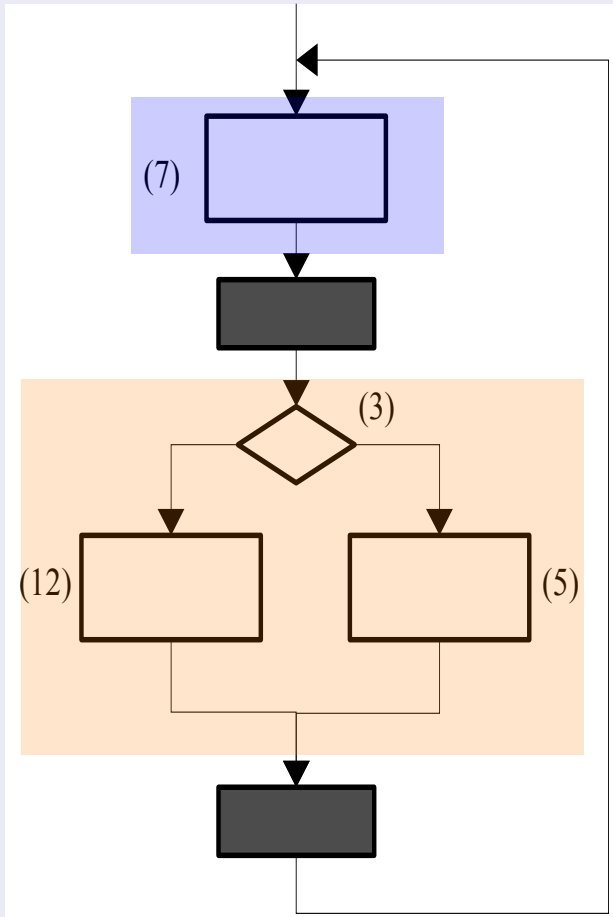




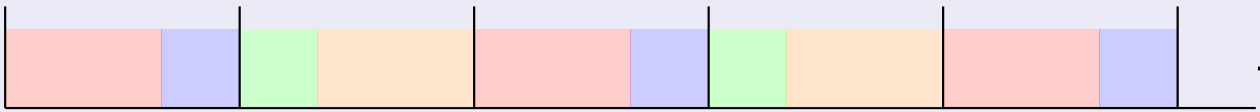


Example 1: Two simple threads

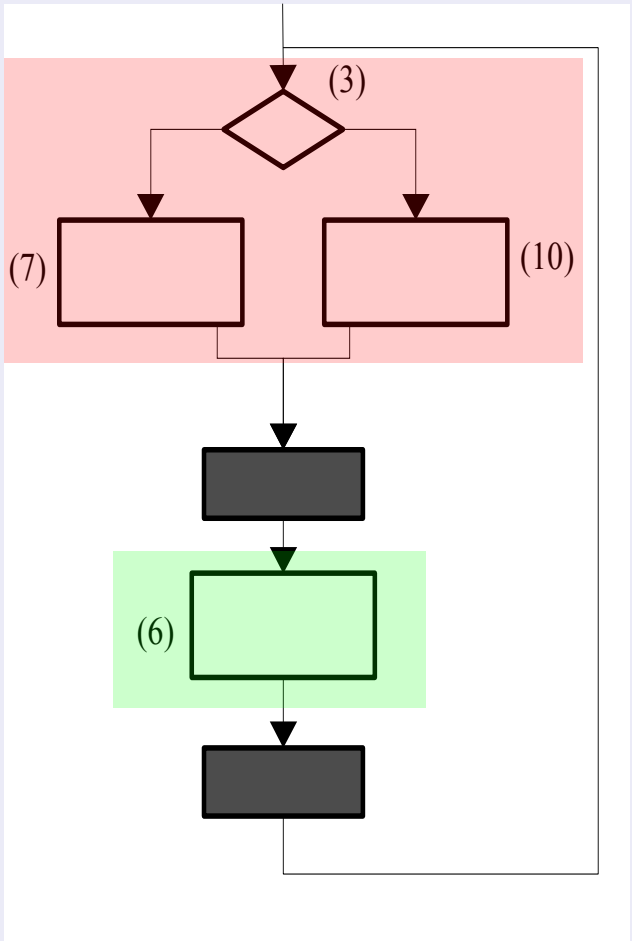


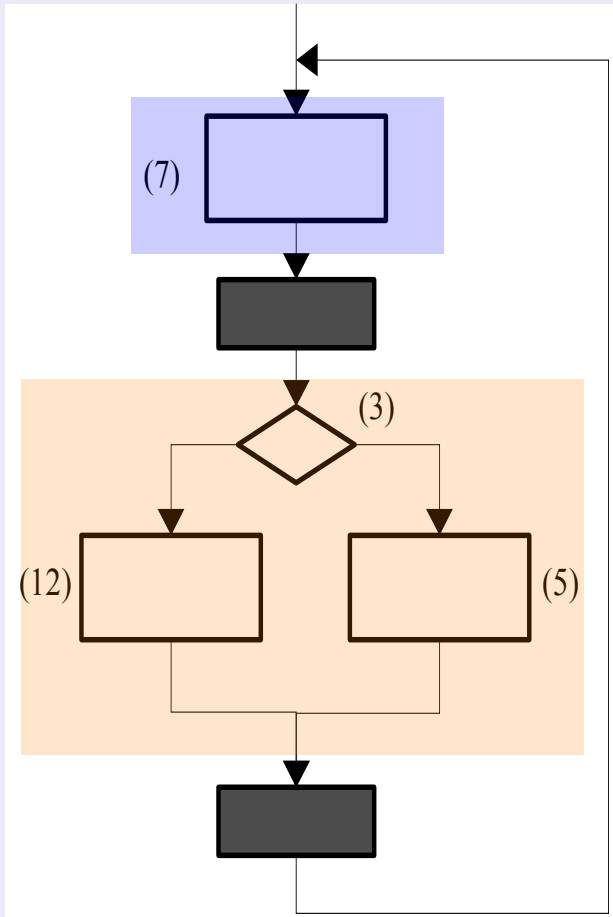


What is the WCRT of this program?

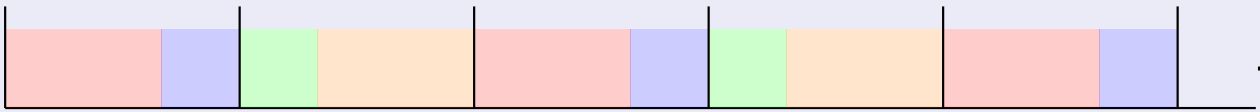


Example 1: Two simple threads





WCRT
=max(T1)+max(T2)
= 13 + 15
= 28

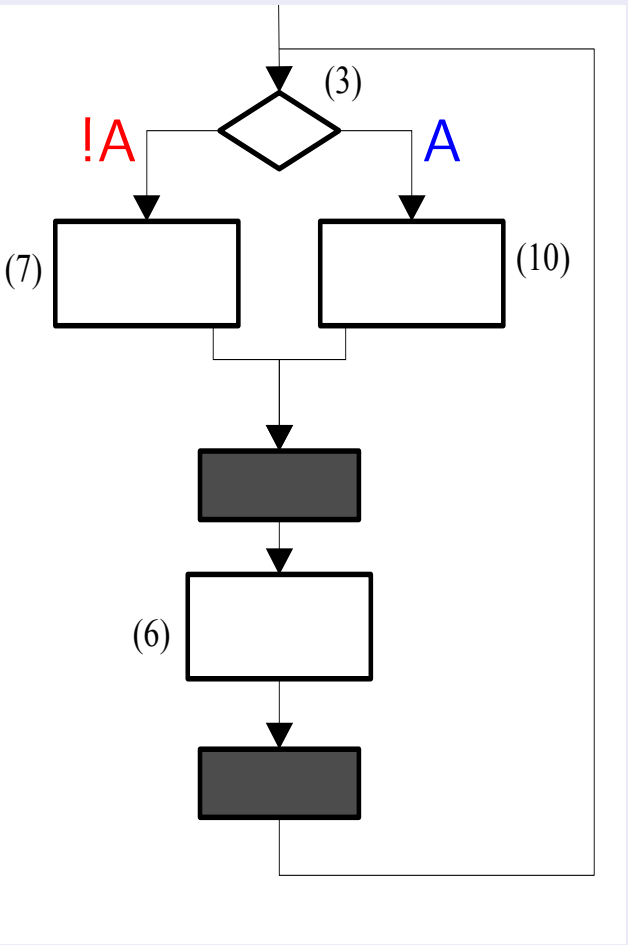


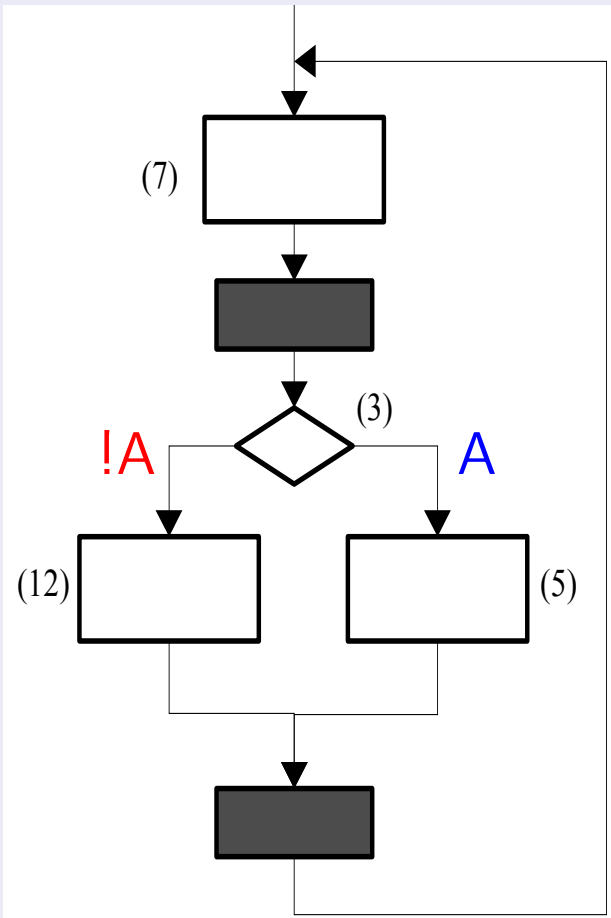
Motivation for the proposed approach

How?

Tight analysis depends on:
a) Data/Control dependency.
b) Tick/State alignment across threads.

Motivation for the proposed approach

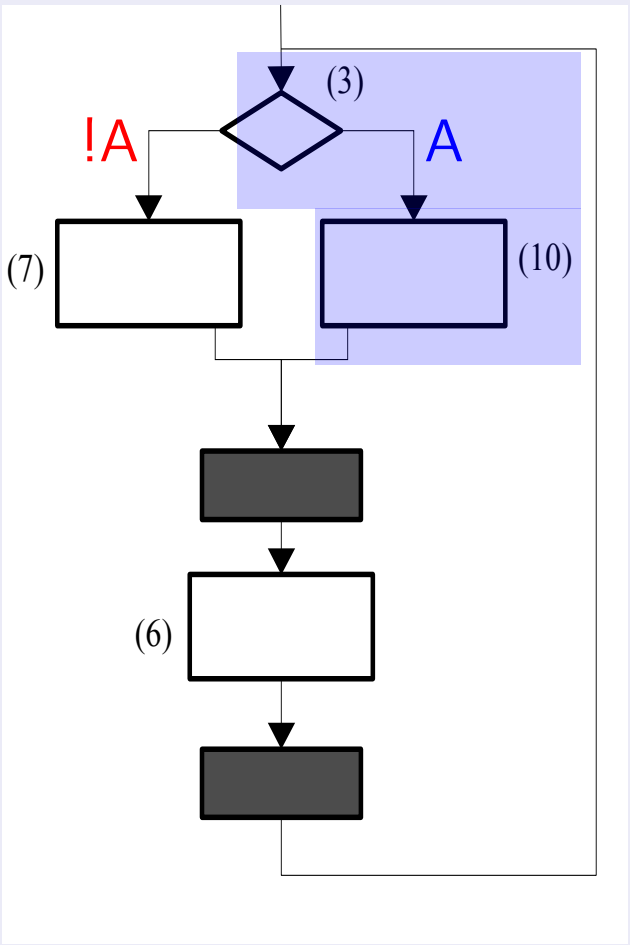


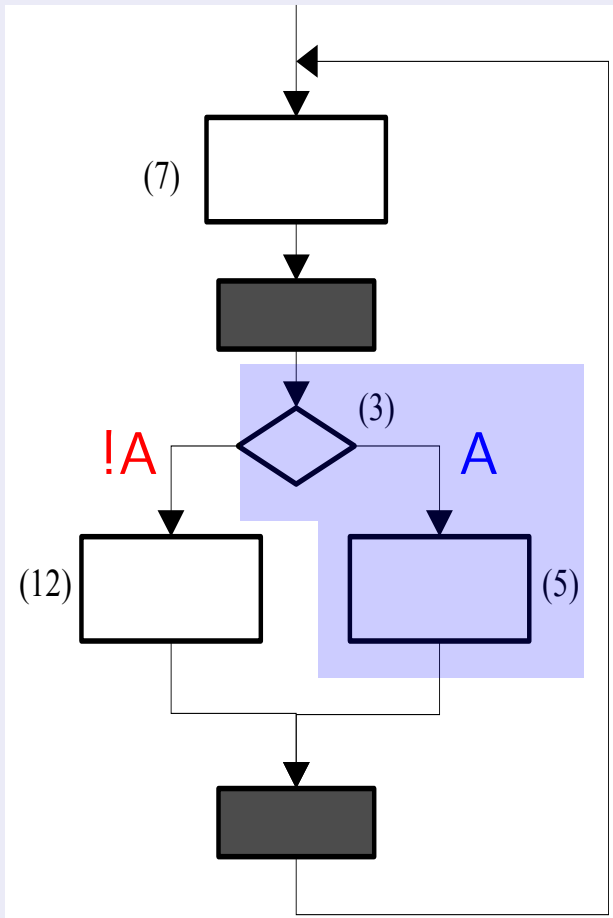


A can be
True or False

a) By considering Data/Control dependencies.

Motivation for the proposed approach

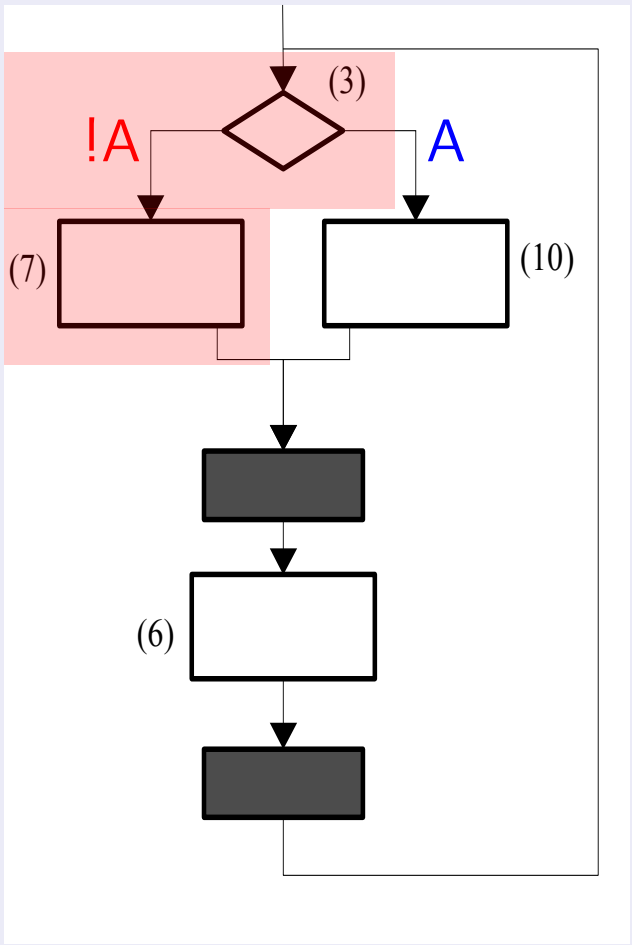


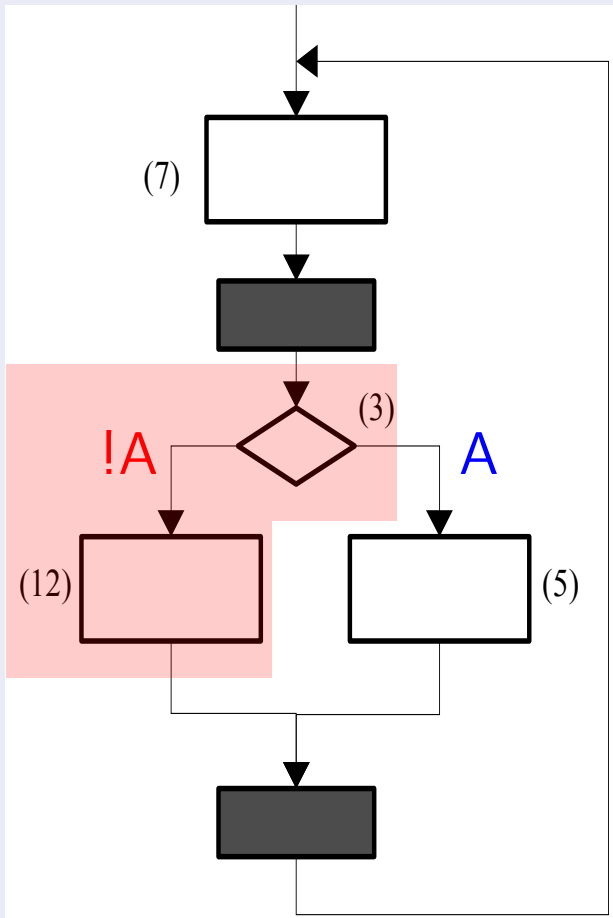


When True :
WCRT = 13+8 = 21

a) By considering Data/Control dependencies.

Motivation for the proposed approach



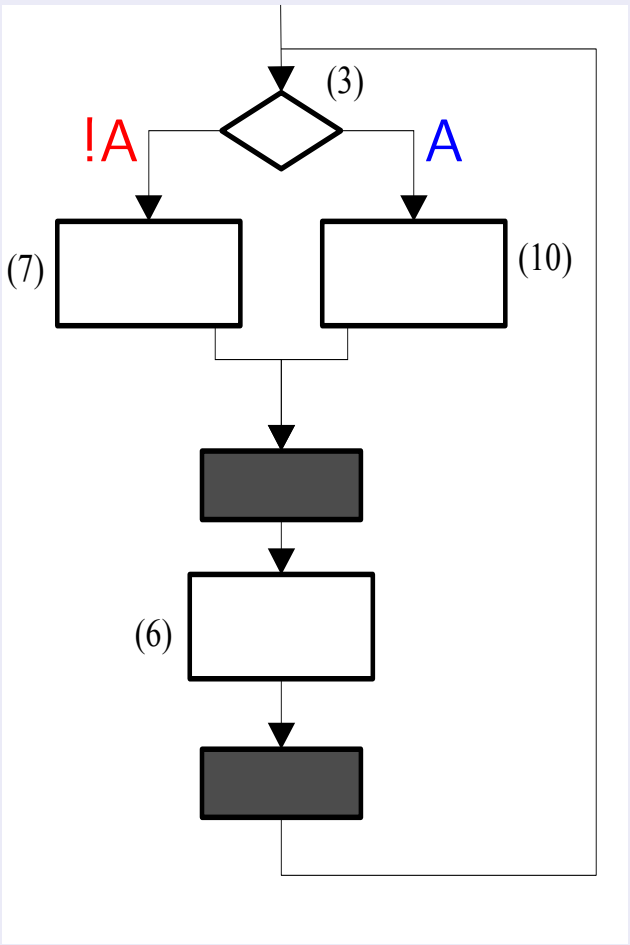


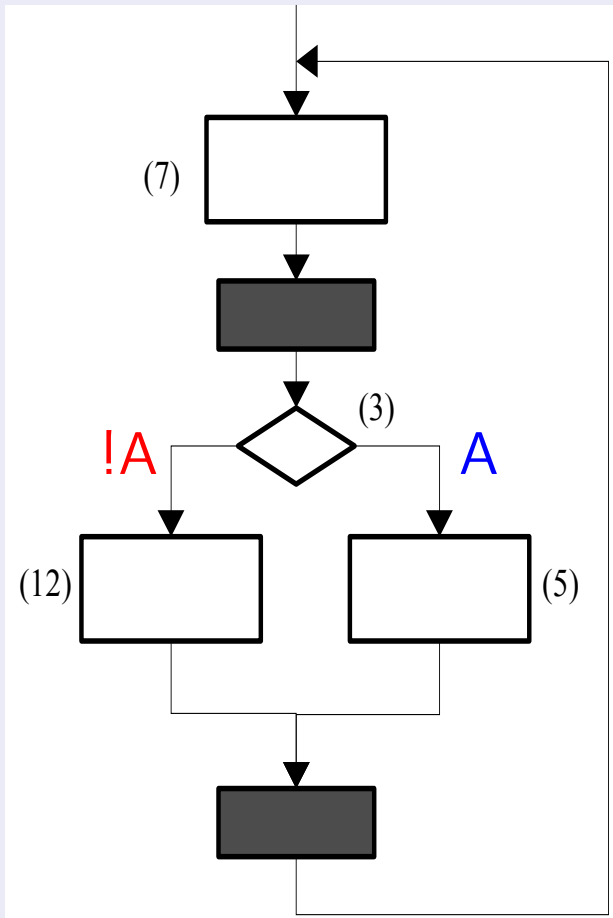
When True :
WCRT = 13+8 = 21

When False:
WCRT = 10+15
=25

a) By considering Data/Control dependencies.

Motivation for the proposed approach





When True :
WCRT = 13+8 = 21

When False:
WCRT = 10+15
=25

WCRT = max(21,25)
=25

a) By considering Data/Control dependencies.

Our Solution

Tighter

Tighter analysis by taking state and data dependencies into account. We can also keep track of our variables in every tick for further infeasible path pruning.

Our Solution

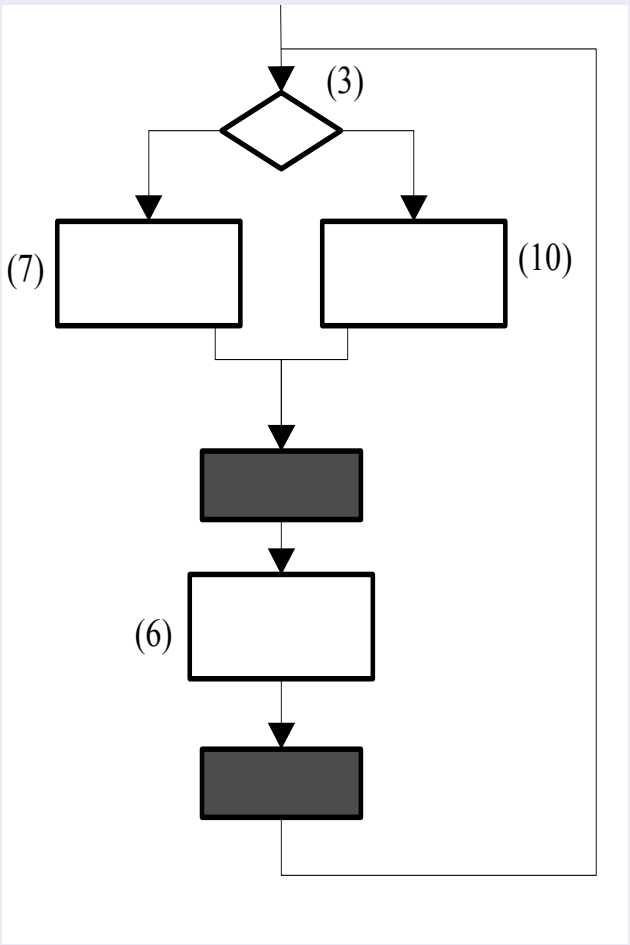
Tighter

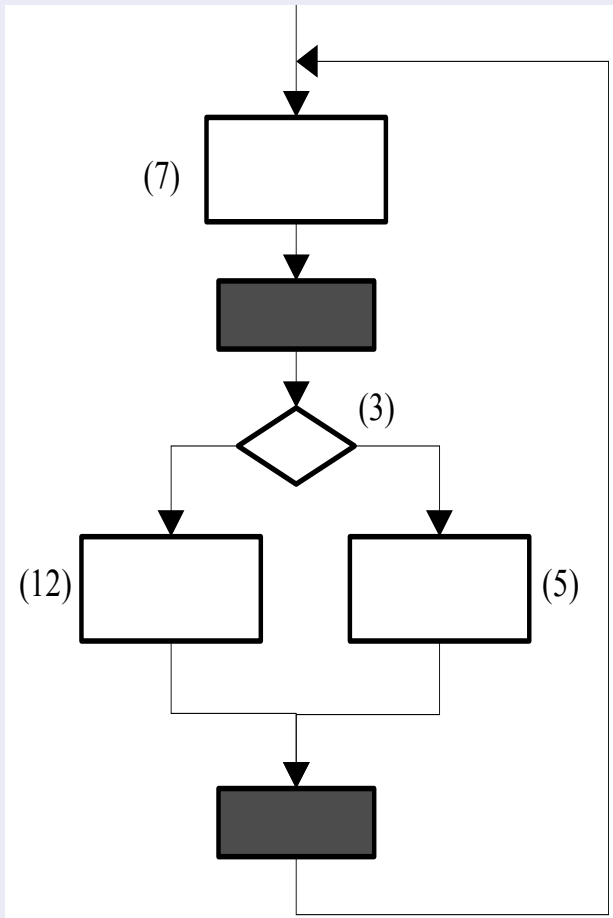
Tighter analysis by taking state and data dependencies into account. We can also keep track of our variables in every tick for further infeasible path pruning.

Model Checking

Synchronous C programs may be represented as a set of concurrent FSMs with transition guards that represent the *execution cost*. This can be exploited by a model checker to determine the reachable state-space and the maximum tick length.

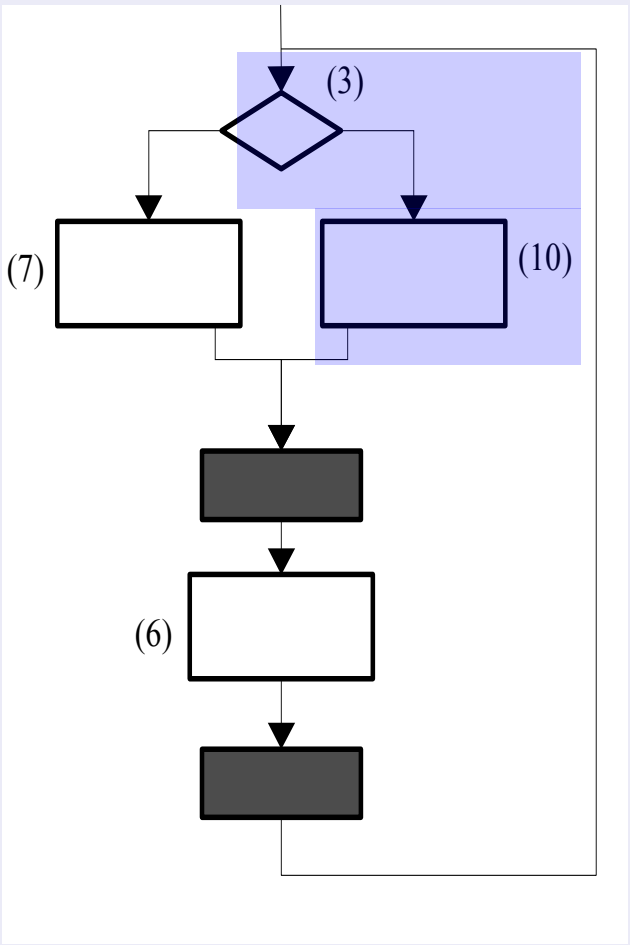
Motivation for the proposed approach

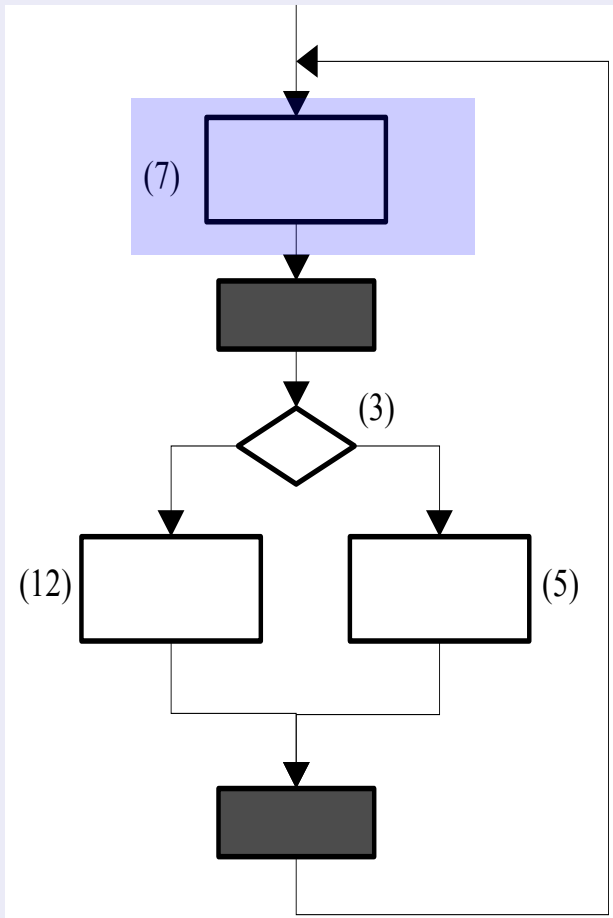




b) By considering only Tick/State alignment.

Motivation for the proposed approach

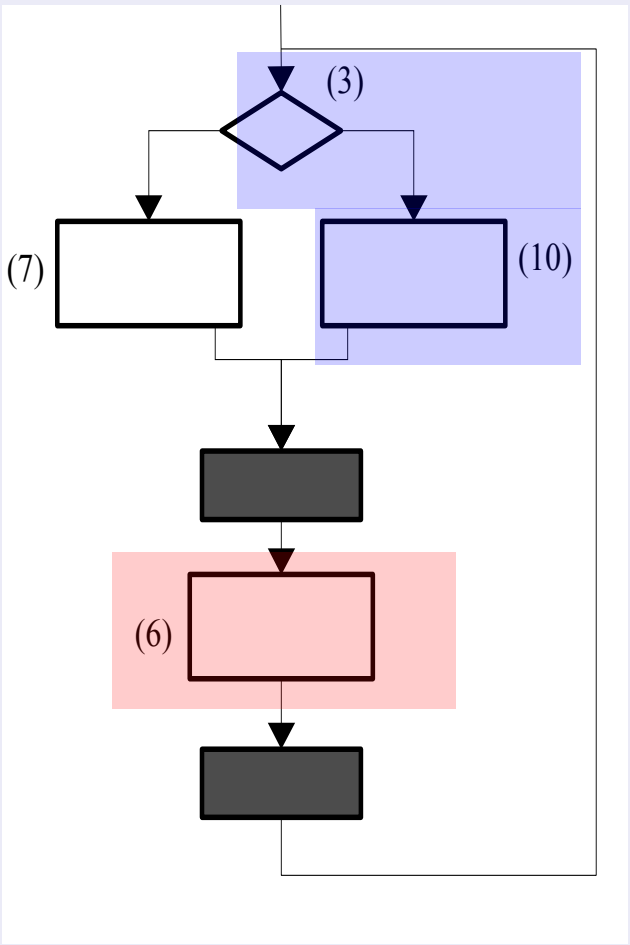


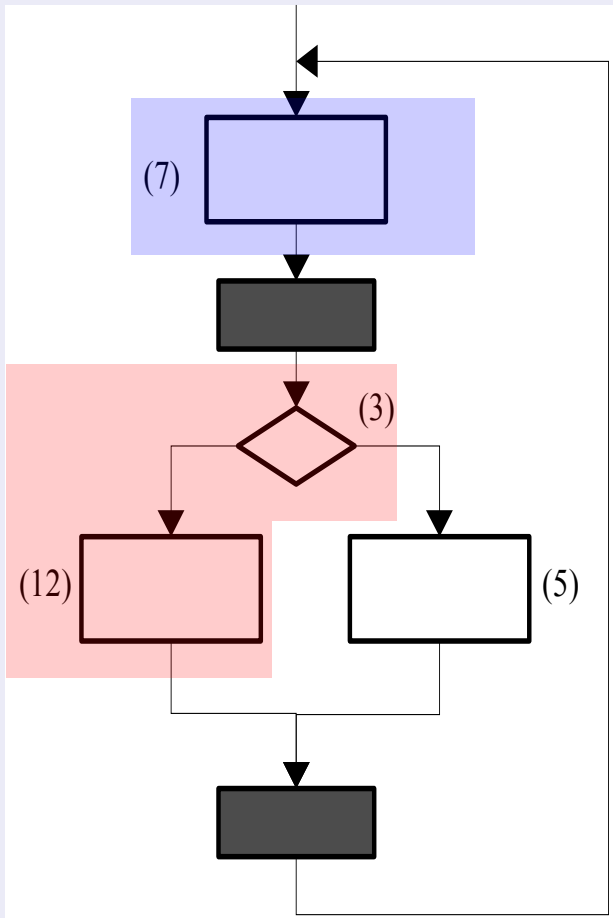


Tick 0: 13+7 = 20

b) By considering only Tick/State alignment.

Motivation for the proposed approach



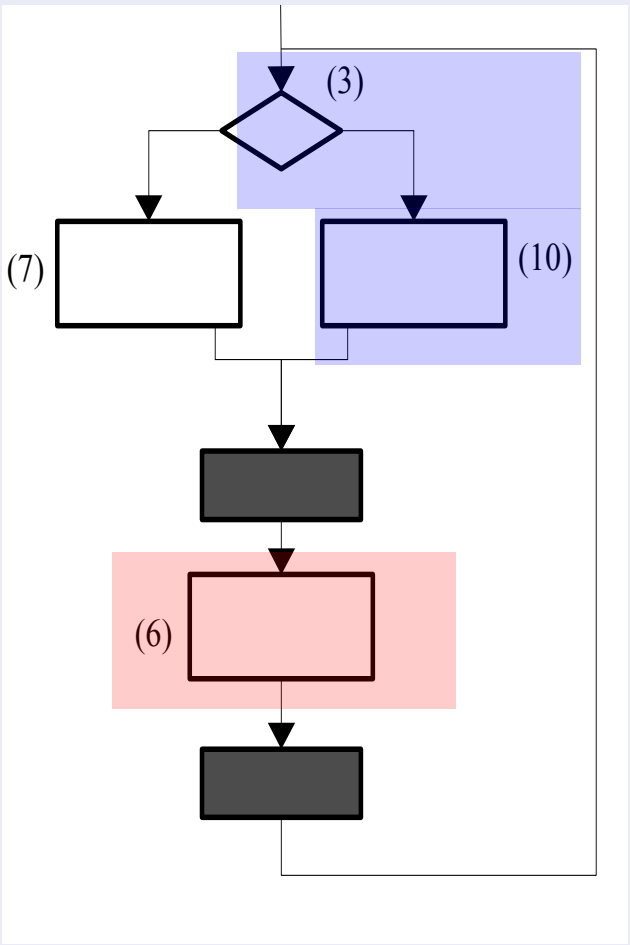


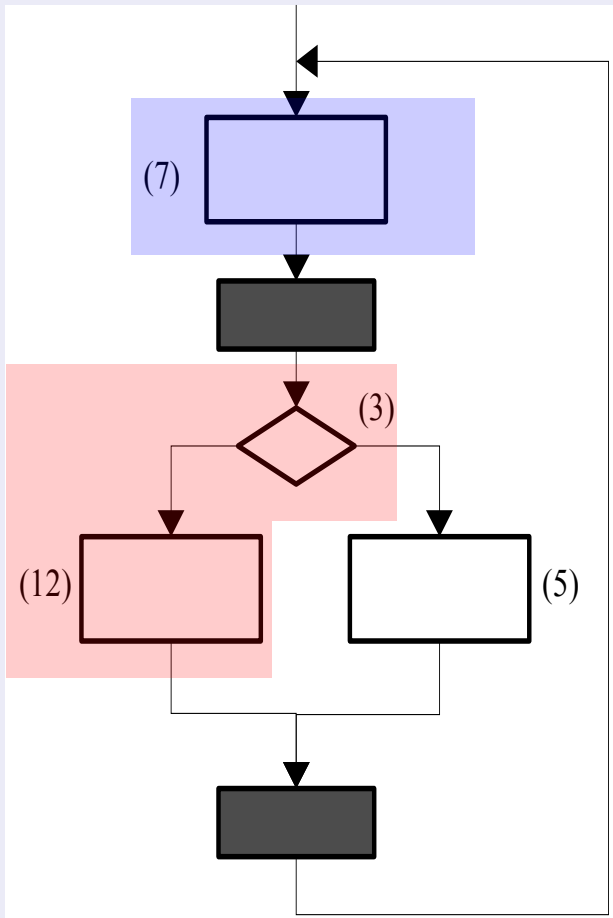
Tick 0: 13+7 = 20

Tick 1: 6+15 = 21

b) By considering only Tick/State alignment.

Motivation for the proposed approach





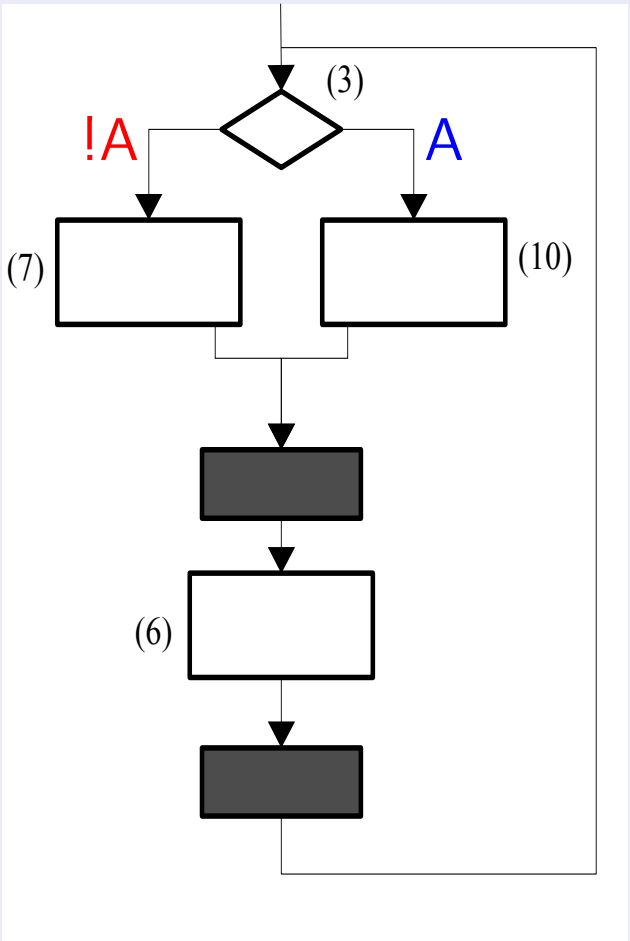
Tick 0: 13+7 = 20

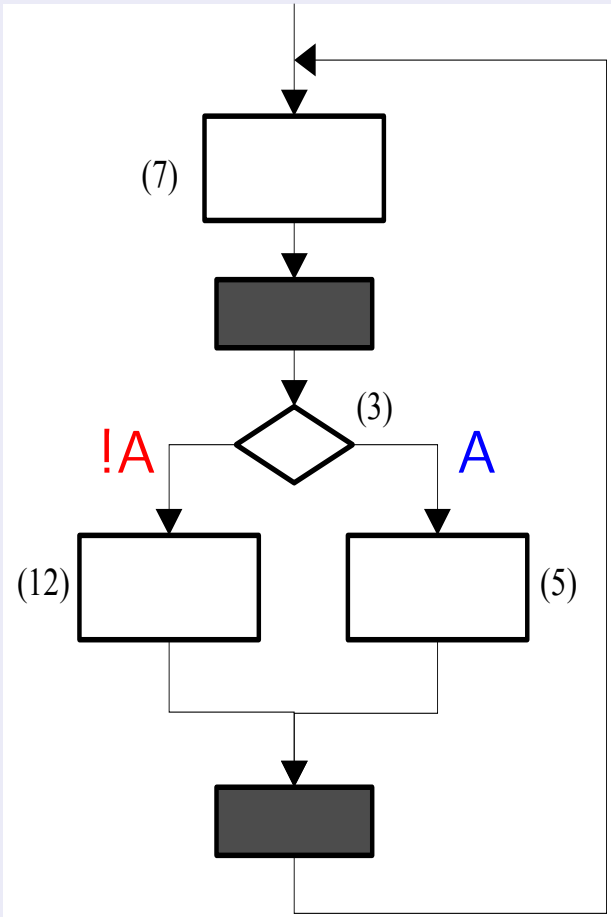
Tick 1: 6+15 = 21

WCRT = max(20,21) =21

b) By considering only Tick/State alignment.

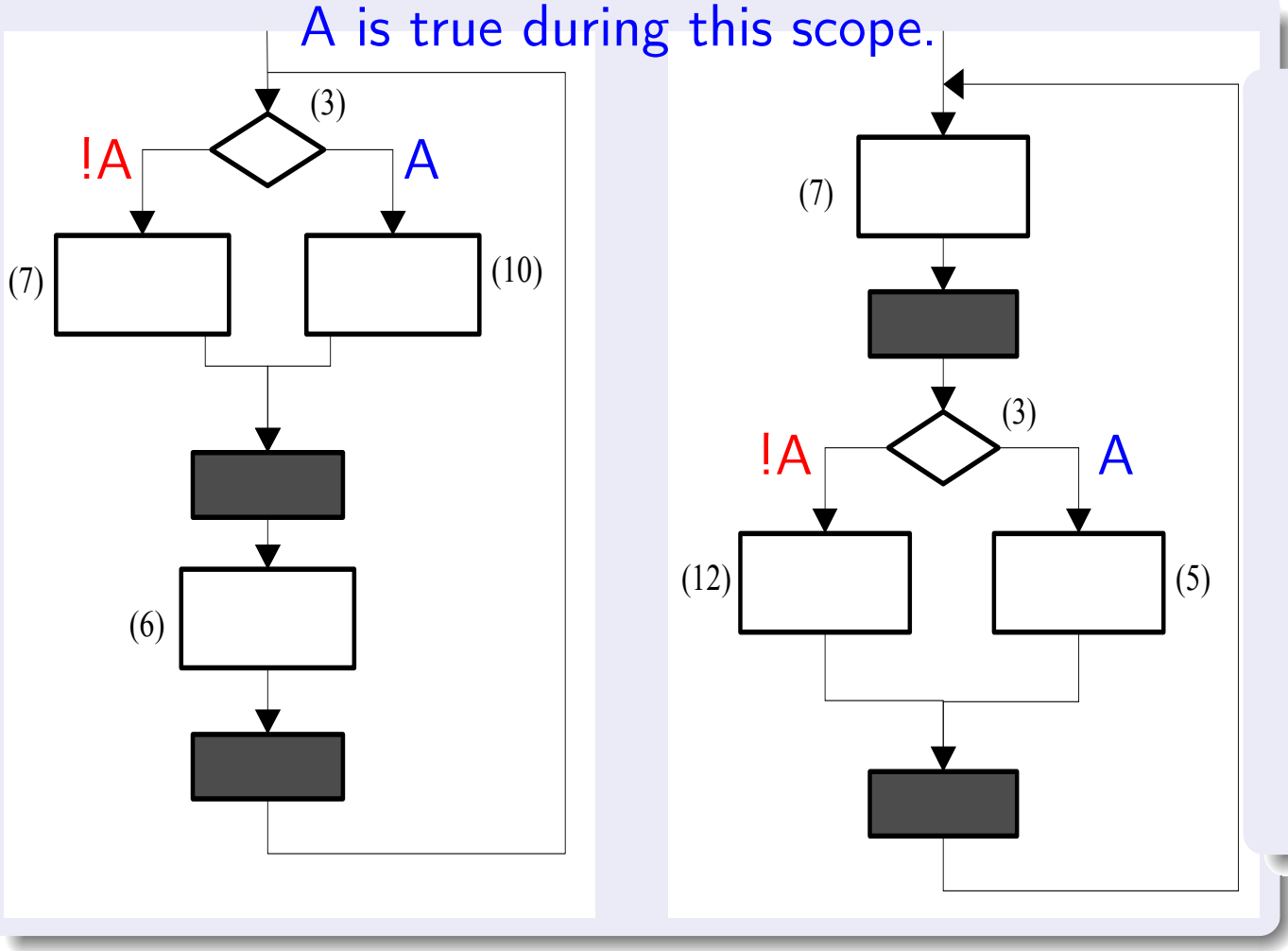
Motivation for the proposed approach





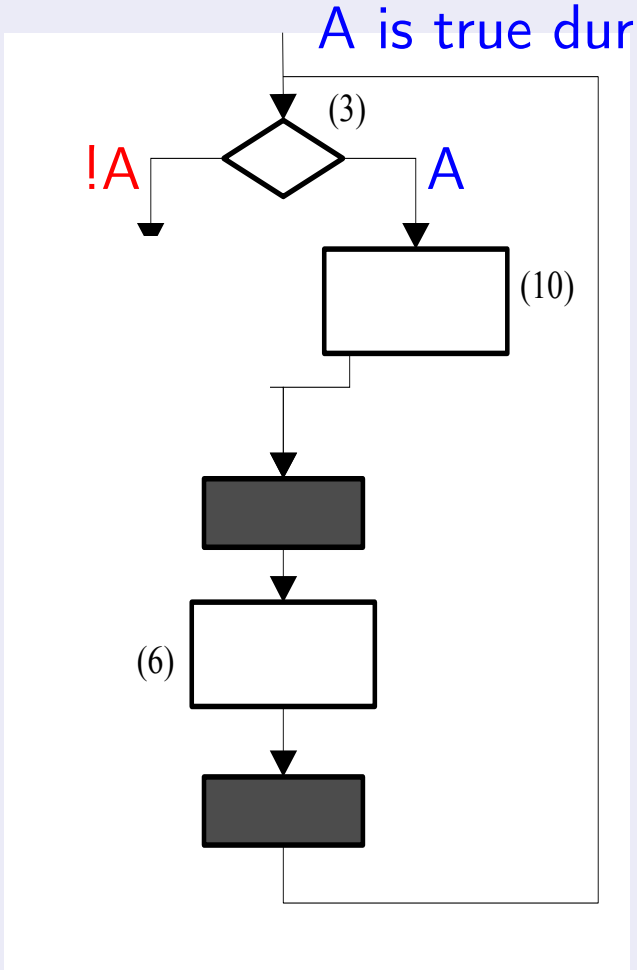
c) By considering data dependencies , tick alignment and by tracking the value of the variables.

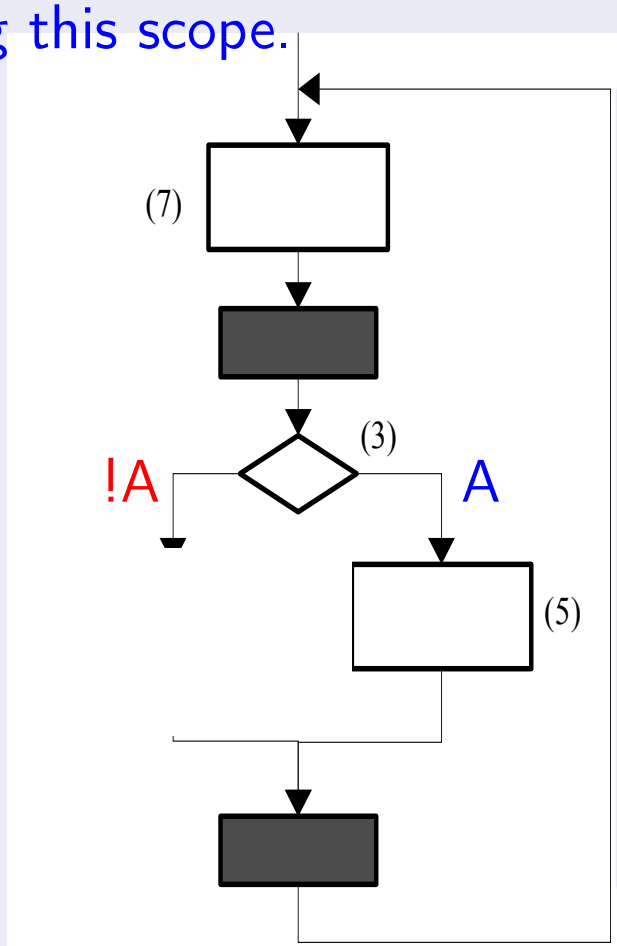
Motivation for the proposed approach



c) By considering data dependencies , tick alignment and by tracking the value of the variables.

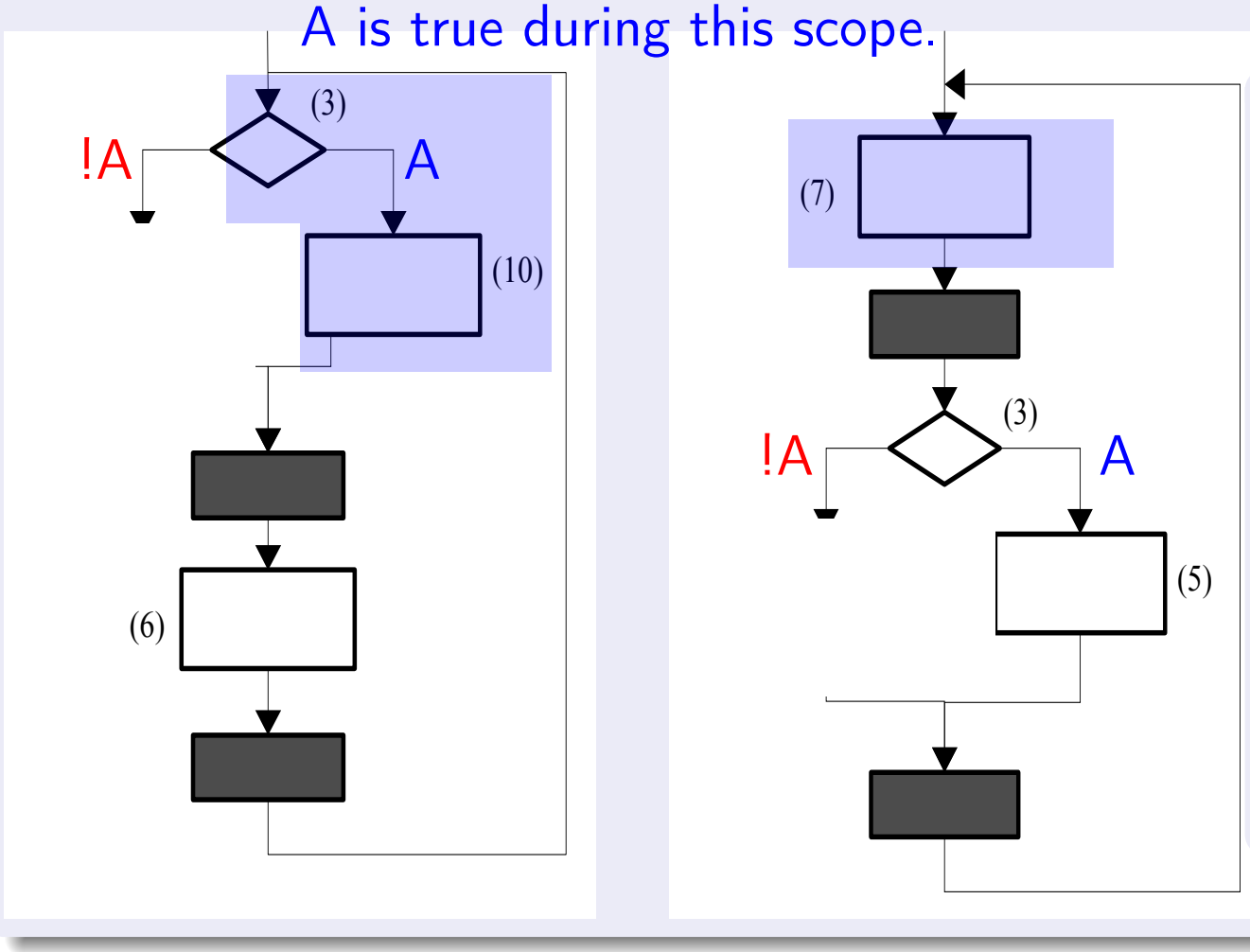
Motivation for the proposed approach





c) By considering data dependencies , tick alignment and by tracking the value of the variables.

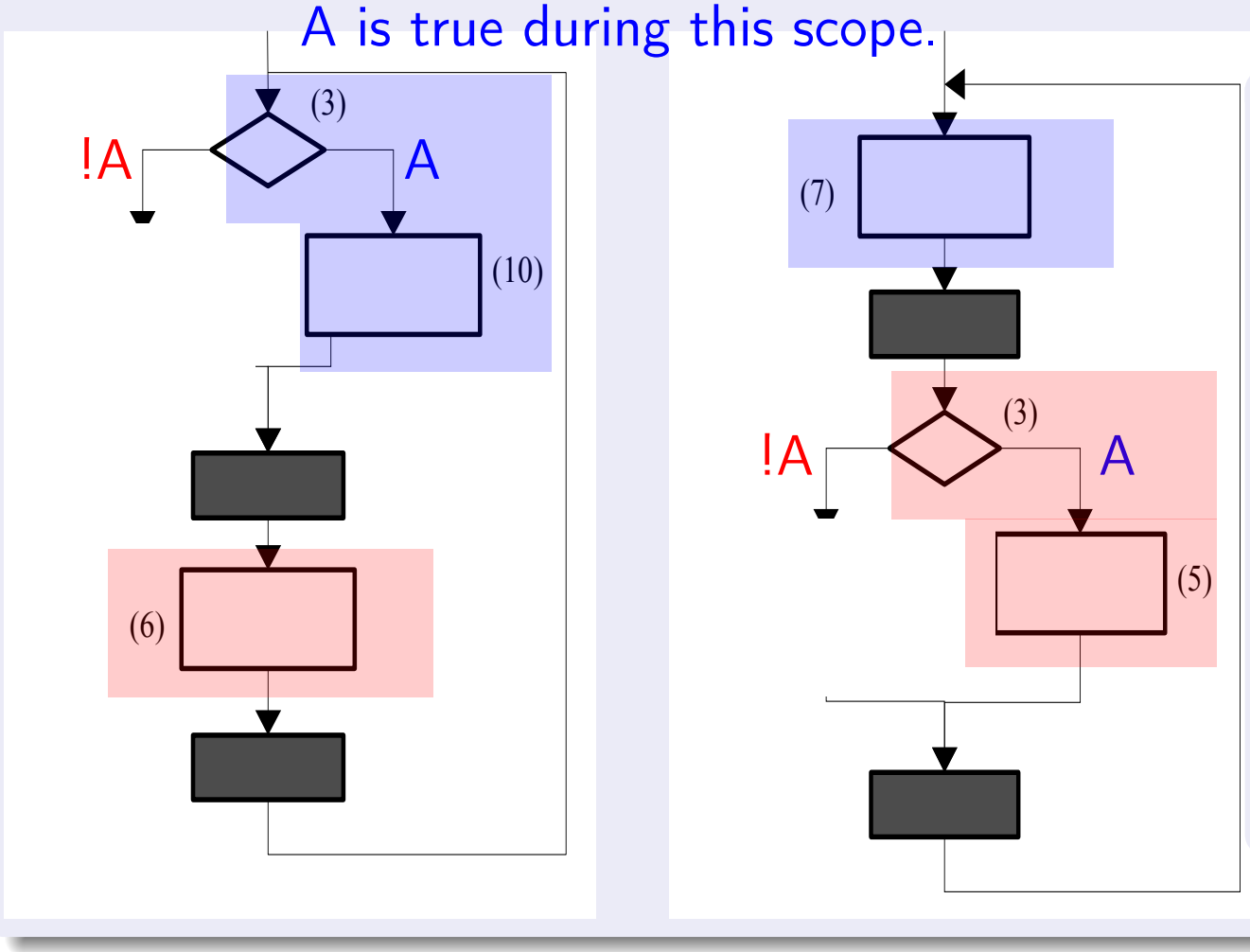
Motivation for the proposed approach



Tick 0: 13+7 = 20

c) By considering data dependencies , tick alignment and by tracking the value of the variables.

Motivation for the proposed approach

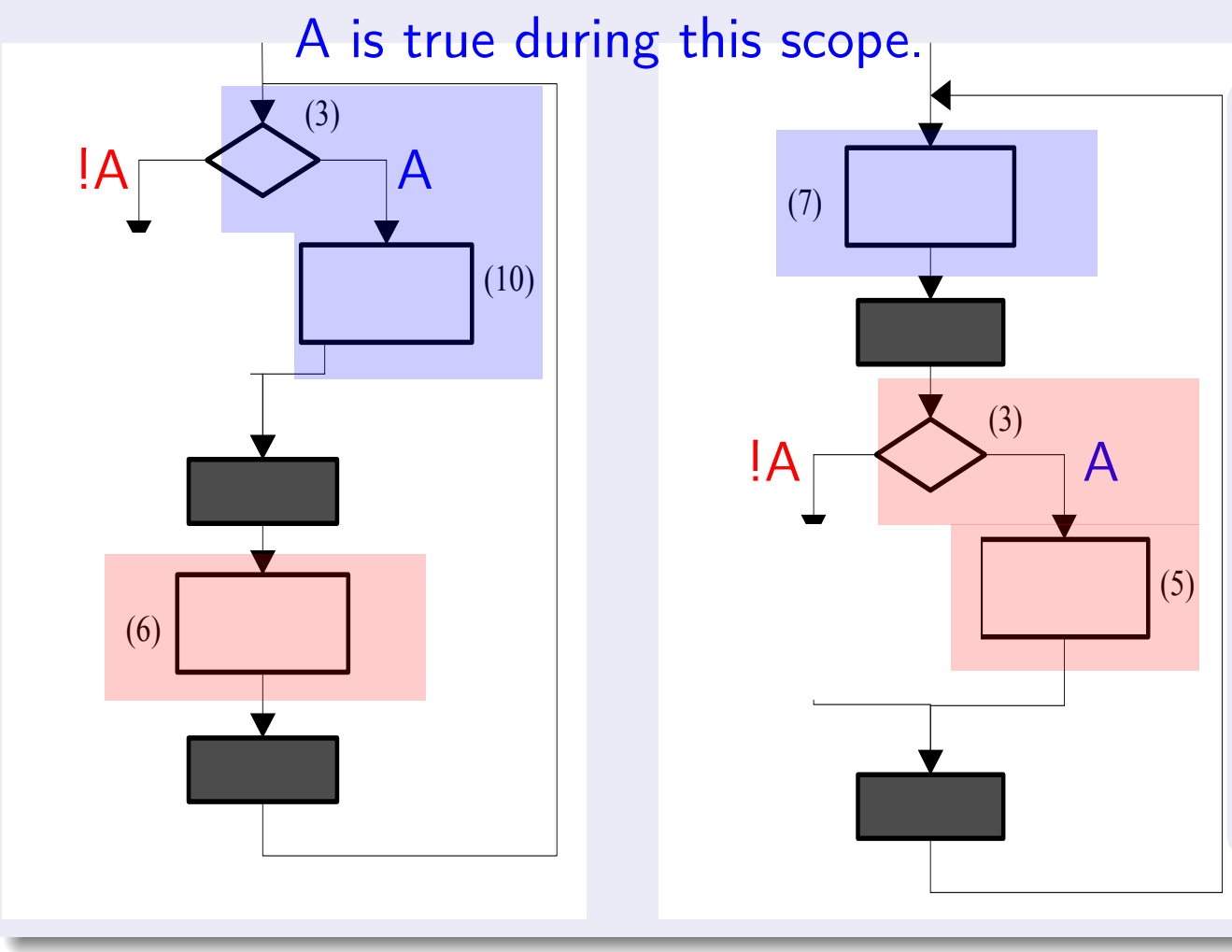


Tick 0: 13+7 = 20

Tick 1: 6+8 = 14

c) By considering data dependencies , tick alignment and by tracking the value of the variables.

Motivation for the proposed approach



Tick 0: 13+7 = 20

Tick 1: 6+8 = 14

WCRT = max(20,14) =20

c) By considering data dependencies , tick alignment and by tracking the value of the variables.

Comparison

Methods

Comparison

Methods

- MaxPlus: 28

Comparison

Methods

- MaxPlus: 28
- MaxPlus + Data/Control : 25

Comparison

Methods

- MaxPlus: 28
- MaxPlus + Data/Control : 25
- MaxPlus + Tick alignment: 21

Comparison

Methods

- MaxPlus: 28
- MaxPlus + Data/Control : 25
- MaxPlus + Tick alignment: 21
- MaxPlus + Data/Control + Tick alignment: 21

Comparison

Methods

- MaxPlus: 28
- MaxPlus + Data/Control : 25
- MaxPlus + Tick alignment: 21
- MaxPlus + Data/Control + Tick alignment: 21
- Data/Control + Tick alignment + Track Variable: 20

Finding the WCRT

Model checking formulation

- We generate an input format of the model checker UPPAAL called timed automata (TA). Our timed automaton do not use any clocks and instead have only bounded integers.
- We compose all TAs in parallel and introduce a barrier TA to emulate the synchronous composition.
- We then evaluate a set of queries of the form $AG(gtick \Rightarrow x \leq val)$.
- We perform binary search in the interval $[WCRT_{min}, WCRT_{max}]$ by successively trying different val until the tight value is found.

Layout

Benchmarking

- PRET-C execution:
  - Hardware support (ARPRET).
  - We have also developed a software model (CEC-like linked-list based scheduler).

Benchmarking

- PRET-C execution:
  - Hardware support (ARPRET).
  - We have also developed a software model (CEC-like linked-list based scheduler).

- We have compared with other light-weight C extensions such as:
  - SyncCharts in C.
  - Protothreads.
  - Esterel.

Benchmarking

Hardware vs Software

Example	Hardware			Software		Gain%	
	A	W	U	A	W	A	W
ABRO	29	58	64	36	94	19.45	38.29
Channel Protocol	57	88	90	91	122	37.36	27.86
Reactor Control	64	82	86	98	114	34.69	28.07
Producer Consumer	42	50	53	43	62	2.32	19.35
Smokers	224	409	413	328	412	31.70	0.73
Robot Sonar	73	92	96	130	175	43.85	47.43
Average						28.23	26.96

- On average, the throughput on the hardware is about 26% greater than the software implementation.

Benchmarking

Quantitative Comparison

Example	PRET-C A	PRET-C W	SC A	SC W	PT A	PT W	Esterel A	Esterel W	AC Gain% SC	AC Gain% PT	AC Gain% ES	WC Gain% SC	WC Gain% PT	WC Gain% ES
ABRO	36	94	261	493	53	138	78	109	86	32	62	80	31	13
Channel Protocol	91	122	684	757	139	162	232	313	86	34	75	83	24	61
Reactor Control	98	114	444	520	93	106	112	144	77	-5	42	78	-7	20
Producer Consumer	43	62	355	422	74	86	408	417	87	41	89	85	27	85
Smokers	328	412	589	671	268	520	552	1063	44	-22	59	38	20	61
Robot Sonar	130	175	720	770	194	236	408	417	81	32	82	77	25	58
Average									77	18	68	74	20	50

Quantitative Comparison of the PRET-C software approach with SyncCharts in C (SC), Protothreads and Esterel.

- Memory usage of PRET-C is better than (2.5%)Prothreads and (26%) Esterel, while slightly worse (3.7%) than SC.

Execution Time

Example	**WCRT (Model Checker)**	**WCRT (Actual Execution)**	**Gain %**
ABRO	89	87	97.75
Channel Protocol	152	149	98.03
Reactor Control	118	114	96.61
Producer-Consumer	92	88	95.65
Smokers	449	430	95.77
Robot Sonar	365	339	97.40
Average			96.87

- On an average, the actual value is approximately 96% of the value obtained from UPPAAL.

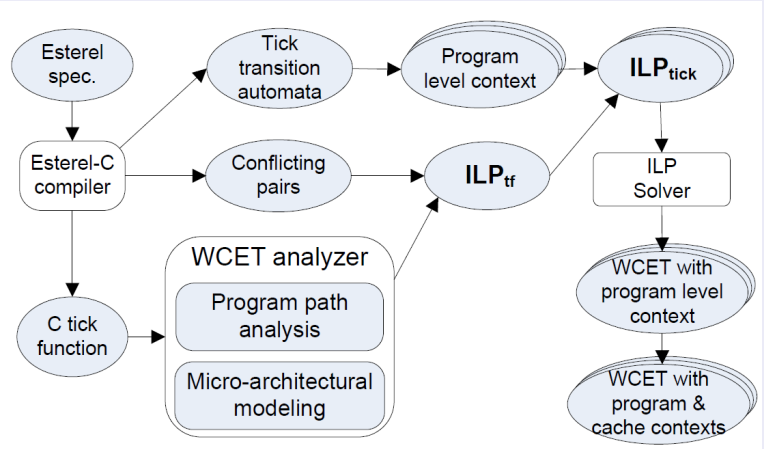
NUS approach for timing analysis of Esterel [Suhendra et al., DAC'06]

Contributions

- Timing analysis over Esterel language.
- Information from intermediate representations of the compiler (CEC) are used to analyse data flow.
- Explicit Tick Transition Automaton is extracted to capture tick alignment between synchronous threads.
- Models Caches, Pipelines and other architectural features for tighter analysis.
- Timing analysis over multi-core platforms.
- Formulation is based on Integer linear Programming (ILP).
- Currently can only handle simple data flow analysis (SDFA) for pruning infeasible paths.

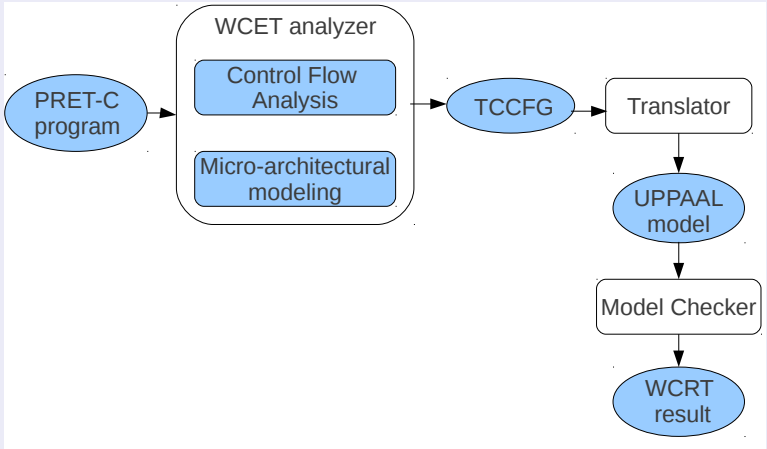
Comparison with NUS approach. [Andalam et al., DATE'11]

NUS framework



- Timing analysis of Esterel.
- ILP based solution.

UoA framework



- Timing analysis of PRET-C.
- MC based solution.

Comparison with NUS approach. [Andalam et al., DATE'11]

code snippet

```
## analysed by NUS
x=3; if(x){..}      //simple conflicting pair

## not analysed by NUS
x=3; z=3; if(z){..} // not a pair
x=3; z=3; if(x<=z){..} // RHS is not a constant
x=3; EOT; if(x){..} //data across ticks
```

- NUS approach can only handle conflicting pairs. Were simple set and test pairs are analysed. [NUS'DAC06]
- While testing RHS of the expression has to be a constant.
- Cannot analyse data flow across ticks..
- Always, user guided loop bounds.

Comparison with NUS approach. [Andalam et al., DATE'11]

NUS's ILP approach can handle simple data flow analysis.
Our MC approach can handle more expressive data flow analysis.

Type	Infeasible paths	NUS	UoA
(1)	Set and Test	Yes	Yes
(2)	Encoding tick transition	Yes	Yes
(3)	Loops with fixed bounds	Yes	Yes
(4)	Set and Test with expressive data-flow analysis	No	Yes
(5)	Encoding tick transition with expressive data-flow analysis	No	Yes
(6)	Loops bounds requiring expressive data-flow analysis	No	Yes

NUS vs UoA

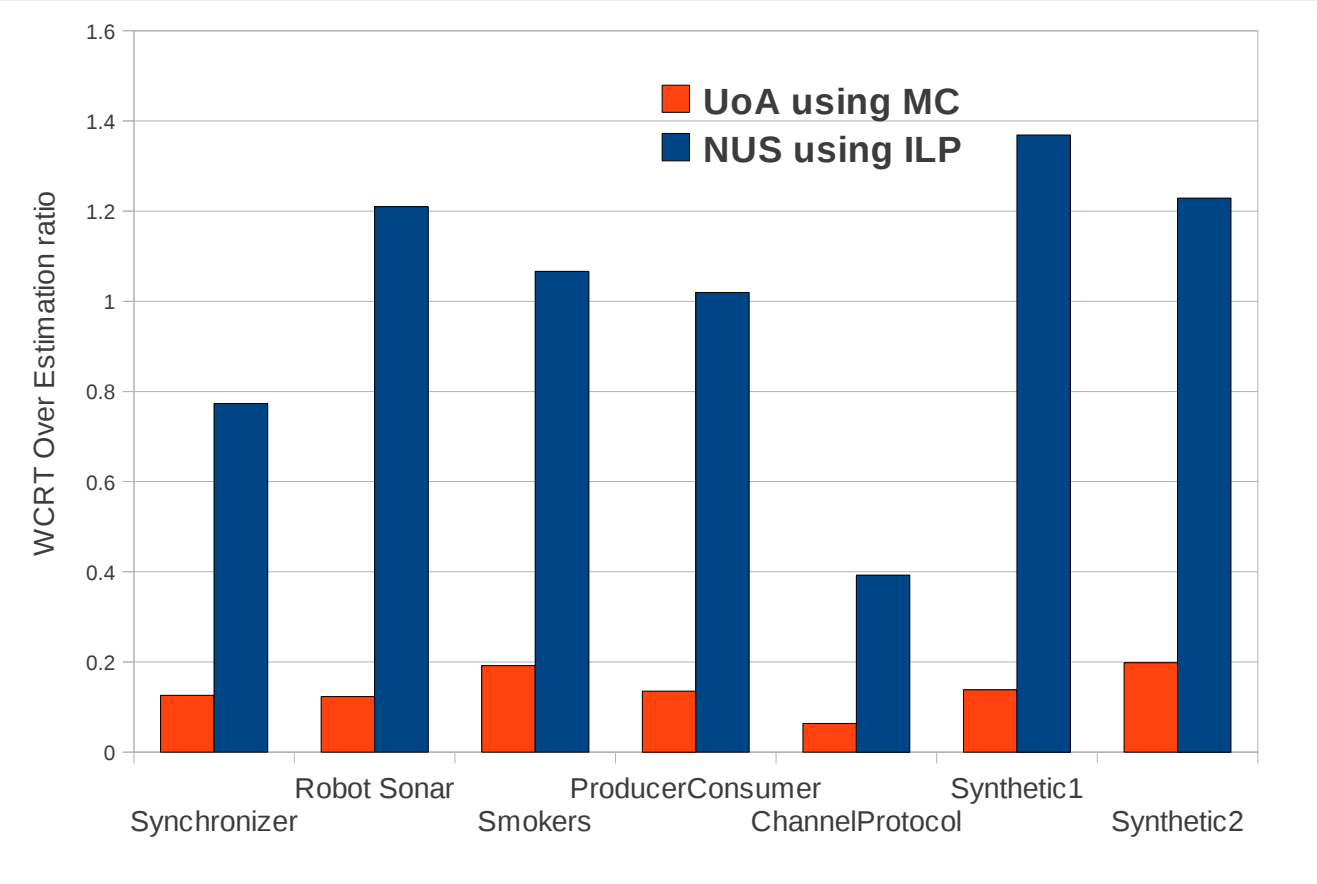
Results

Example	LOC	Observed WCRT	(MAX+) Est WCRT	(NUS) est. WCRT (1)(2)(3)	(UoA) est. WCRT (1)to(6)	NUS/ UoA
Synchronizer	455	238	608	422	268	1.57
ProducerConsumer	567	259	808	523	294	1.78
Smokers	648	437	1309	903	521	1.73
Channel Protocol	727	644	1426	897	685	1.31
Robot Sonar	1081	764	2028	1688	858	1.97
Synthetic1	1569	898	3593	2127	1022	2.08
Synthetic2	1630	786	3617	1752	942	1.86
Average						1.67

- MicroBlaze is the target platform.
- Entire program fits on on-chip memory.

NUS vs UoA

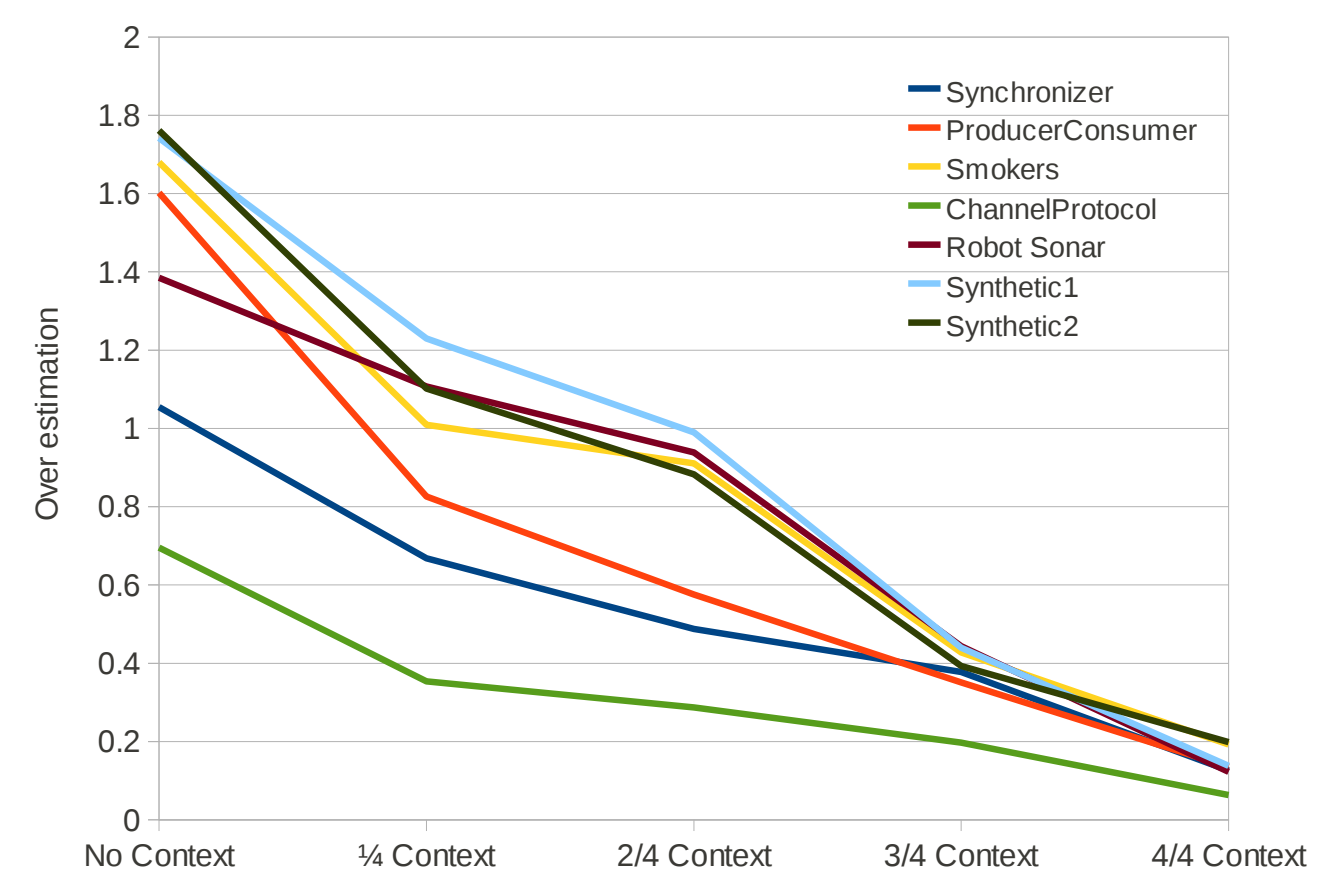
Over Estimation



- On an average, the NUS approach overestimates by 89% while UoA approach only overestimates by about 13%.

Context Sensitive Pruning

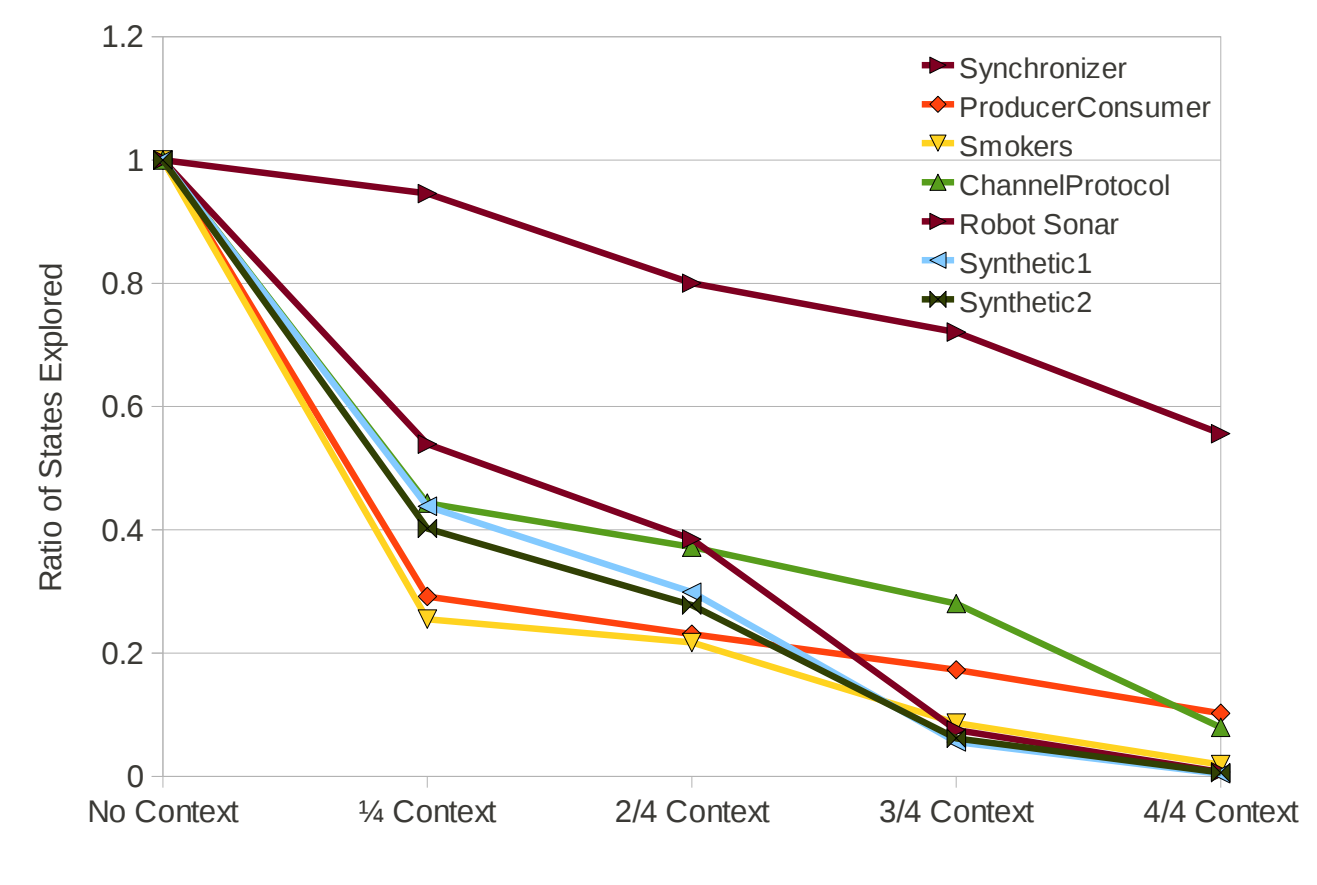
Tighness



- Increase in context information prunes infeasible paths. Thus reducing the WCRT overestimation.

Context Sensitive Pruning

States Explored



- Increase in Context information also reduces the number of states explored. This reduces the time to analyse the model.

Conclusions

Contributions

- New synchronous language for predictable programming.
- Thread-safe shared memory access via simple semantics.
- Predictable preemption support.
- Hardware accelerator to improve the worst case behaviour.
- Mapping of logical time to physical time through static analysis.
- Side effect: PRET-C excels both in the worst case and average case execution over other light-weight threading libraries.

Future work

- To explore the trade-offs of scratchpads versus caches.
- Support for parallel execution of PRET-C via new semantics.
- Exploring the link between our research and the Berkeley-Columbia research.