TORO: A TOol to compute RObustness margins for cause-effect chains.

Leonie Köhler, M.Sc.
Nikolas Brendes, B.Sc.
Prof. Dr.-Ing. Rolf Ernst

Version 1.0
(typeset November 13, 2019)

Abschlussbericht AP3

iTUBS
Braunschweig

Contents

1 Introduction

Control applications in automotive software systems often contain time-critical cause-effect chains. A time-critical chain typically includes the reading of sensor data, their processing and finally the control of actuators. Satisfying specified end-to-end deadlines of cause-effect chains serves to ensure correct system behavior and can also increase driving comfort. From an implementation point of view, a cause-effect chain consists of communicating tasks that are distributed among various system components.

Designing a software application in such a way that all end-to-end deadlines of cause-effect chains are satisfied, even in the worst case, is challenging. Since software applications are regularly extended, it is another central concern to guarantee end-to-end deadlines in case of software updates without having to carry out a completely new software design.
The tool TORO (TOol to compute RObustness margins)

- processes the model of a given software application with time-critical effect chains,
- calculates the maximum end-to-end latencies of the given effect chains,
- makes statements about the robustness of applications during software updates with regard to compliance with end-to-end deadlines.

The results can be used

- to evaluate the robustness of a given software application regarding software updates,
- to identify critical (less robust) segments of a cause-effect chain,
- to predict whether a given software update will change the timing of effect chains such that end-to-end deadlines are missed,
- to determine where and why a given software update changes the time behavior such that end-to-end deadlines are missed.

The tool TORO is based on a novel robustness analysis that was developed in 2018/19 in a collaboration between Daimler and iTUBS. The analysis is summarized in a scientific article which forms the appendix of this documentation. The submission of the article to ECRTS 2020 is planned.

1.1 Structure of the Document

todo

2 Functionality

This chapter describes the terms and methods that are used by the tool **Toro**.

2.1 Input: System Specification

This section explains what type of systems are accepted by **Toro** as input. In section 2.1.1, a general system model is introduced first. All systems that fulfill this system model can be analyzed by **Toro**. Section 2.1.2 lists some practical examples to which the system model applies.

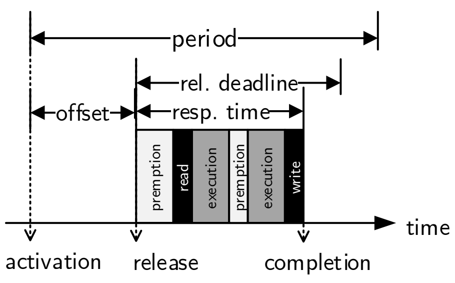
2.1.1 System Model

To analyze the timing of one or more cause-effect chains, a model of the hardware-software system is required. The system model represents the hardware platform and the application software neglecting all aspects that do not determine the timing of the system. In the following, the system model for **Toro** is specified.

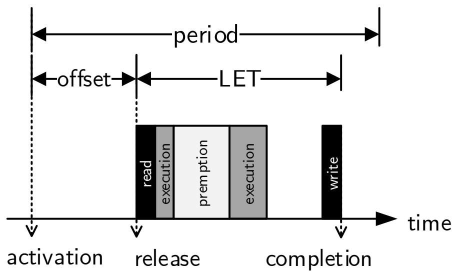
A *hardware platform* $\mathcal{P} = (\mathcal{R}, \mathcal{E})$ is a directed graph. A vertex of the graph is an element from the set of resources $\mathcal{R}$. The edges $\mathcal{E}$ correspond to the directed physical connections between resources. A *resource* $R \in \mathcal{R}$ is characterized by the property that it provides processing or communication service to tasks according to a given scheduling policy. A resource can thus be a processor core, a data bus, etc. We assume that the clocks of all resources are perfectly synchronized.
A *task* $\tau_i \in \mathcal{T}$ is a service-consuming entity. This definition is very general and corresponds to the concept of a task in the real-time community. Since consumed service can be processing service or communication service; we do not need to distinguish between notions like 'software tasks' and 'messages'. The jth instance of a task τ_i is called job $\tau_i(j)$. Two variants of a task exist:

Definition 1 (Bounded execution time task)**.** *The bounded execution time (BET) task is characterized by the following attributes*

- Periodic activation model. *A task τ_i is periodically activated with period P_{τ_i}.*
- Release offset. *A constant release offset Φ_{τ_i} applies after each activation of task τ_i.*
- Worst-case response time. *The response time of a job $\tau_i(j)$ is the duration between its release and completion. It is bounded from above by the worst-case response time $WCRT_{\tau_i}$.*
- Relative deadline. *The relative deadline d_{τ_i} is added to the release instant of a job $\tau_i(j)$ to determine the latest acceptable moment of completion of job $\tau_i(j)$.*
- Read-execute-write semantics. *A task with read-process-write semantics reads all required inputs before it starts to process. All outputs are written to registers directly after the data processing is finished.*
- Allocation. *A task is associated with the resource from which it receives service.*



(a) Periodic BET task with release offset.



(b) LET task with release offset.

Figure 1: Task model.

Definition 2 (Logical execution time task). *The logical execution time (LET) task is characterized by*

- Periodic activation model. *A task τ_i is periodically activated with period P_{τ_i}.*
- Release offset. *A constant release offset Φ_{τ_i} may applies after each activation of task τ_i.*
- Logical execution time. *A task with logical execution time (LET) semantics reads all required inputs at its release time. All outputs are written to registers with the elapse of the LET_{τ_i}.*
- Allocation. *A task is associated with the resource from which it receives service.*

An *application* is a directed graph $\mathcal{A} = (\mathcal{T}, \mathcal{D})$, where the set of vertices $\mathcal{T}$ represents the tasks in the application and the set of directed edges $\mathcal{D}$ represents the data-flow dependencies among the tasks.

Definition 3 (Cause-effect chain). *A cause-effect chain $C = (\tau_1^c, \tau_2^c, \dots, \tau_n^c)$ is a finite directed walk in an application $\mathcal{A}$ which is executed on a hardware platform $\mathcal{P}$.*

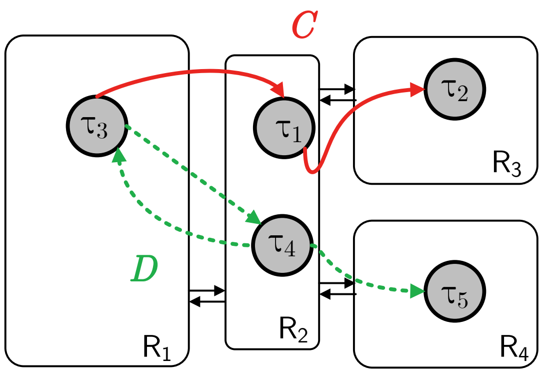
Definition 4 (Instance of a cause-effect chain). *An instance of a cause-effect chain $C(j_1, j_2, \dots, j_n)$ is a sequence of jobs $(\tau_1^c(j_1), \tau_2^c(j_2), \dots, \tau_n^c(j_n))$, where each job $\tau_{k+1}^c(j_{k+1})$ reads data from the predecessor job $\tau_k^c(j_k)$.*

The concepts of a cause-effect chain and an instance of a cause-effect chain are illustrated in Figure 2. The possible instances of a cause-effect chain can be determined by a data flow analysis, which checks whether a data flow through the sequence of jobs $\tau_1^c(j_1)\tau_2^c(j_2), \dots, \tau_n^c(j_n)$ is at all possible.

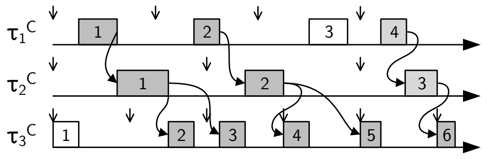
The tool **TORO** supports time-triggered and LET-triggered cause-effect chains:

Definition 5 (Time-triggered cause-effect chain). *A time-triggered cause-effect chain C^{tt} is a cause-effect chain which has the following properties:*

- *Each task τ_i^c in C^{tt} is a periodically activated BET task with period $P_{\tau_i^c}$ and release offset $\Phi_{\tau_i^c}$.*



(a) Exemplary cause-effect chains $C = (\tau_1^c, \tau_2^c, \tau_3^c) = (\tau_3, \tau_1, \tau_2)$ and $D = (\tau_1^D, \tau_2^D, \tau_3^D, \tau_4^D, \tau_5^D) = (\tau_3, \tau_4, \tau_3, \tau_4, \tau_5)$ in an application.



(b) Exemplary instances of a cause-effect chain $C = (\tau_{C_1}^c, \tau_2^c, \tau_3^c)$:
$C(1,1,2) = (\tau_1^c(1), \tau_2^c(1), \tau_3^c(2))$
$C(1,1,3) = (\tau_1^c(1), \tau_2^c(1), \tau_3^c(3))$
$C(2,2,4) = (\tau_1^c(2), \tau_2^c(2), \tau_3^c(4))$
$C(2,2,5) = (\tau_1^c(2), \tau_2^c(2), \tau_3^c(5))$
$C(4,3,6) = (\tau_1^c(4), \tau_2^c(3), \tau_3^c(6))$

Figure 2: Modeling cause-effect chains.

- *The deadline of each task τ_i^c in C^{tt} is arbitrary.*
- *At the system start t_0, all tasks in C^{let} are activated with release offsets $\Phi_{\tau_i^c}$.*

Definition 6 (LET-triggered cause-effect chain)**.** *A LET-triggered cause-effect chain C^{let} is a cause-effect chain which has the following properties:*

- *Each task τ_i^c in C^{let} is a LET task.*
- *The deadline of each task τ_i^c in C^{let} is arbitrary.*
- *At the system start t_0, all tasks in C^{let} are activated with release offsets $\Phi_{\tau_i^c}$.*

TORO assumes that each task can satisfy its deadline or LET.

2.1.2 Exemplary Analyzable Systems

Application on a single-core ECU under the following assumptions

- Only time-triggered and LET-triggered cause-effect chains.
- Each task meets its deadline or LET.

Application on a multi-core ECU under the following assumptions

- Only time-triggered and LET-triggered cause-effect chains.
- Each task meets its deadline or LET.
- All cores have a common clock, all schedules start simulateneously.

Distributed application on a systems with multiple ECUS connected by data buses and networks under the following assumptions

- Only time-triggered and LET-triggered cause-effect chains. *Note that messages can also be modeled as tasks.*
- Each task meets its deadline or LET.
- All clocks are synchronized, all schedules start simulateneously.

2.2 Outputs: Latencies and Robustness Margins

This section describes the expected outputs of the analysis, which are both numerical and graphical.

2.2.1 Numerical Outputs

Maximum End-to-End Latencies

Feiertag et al. [1] list different definitions for end-to-end latencies of cause-effect chains. Each definition is relevant in specific contexts, but the most important one is the following:

Definition 7 (End-to-end latency, also: data age)**.** *The end-to-end latency of an instance of a cause-effect chain, denoted as $Lat(C(j_1, j_2, \ldots, j_n))$, is the maximum amount of time that may elapse from the release of the first job $\tau_1^c(j_1)$ to the completion of the last job $\tau_n^c(j_n)$ in $C(j_1, j_2, \ldots, j_n)$.*

Definition 8 (Maximum end-to-end latency)**.** *The maximum end-to-end latency $\overline{Lat}(C)$ of a cause-effect chain C is an upper bound on the end-to-end latency of any of its instances*

$$\overline{Lat}(C) = \max_{C(j_1, j_2, \ldots, j_n) \in \mathcal{C}} \left\{ Lat(C(j_1, j_2, \ldots, j_n)) \right\}.$$

The maximum end-to-end latency describes how long it can take until fresh input data at the beginning of the cause-effect chain (e.g. a sensor sample) impacts the output of a cause-effect chain (e.g. actuator control value). This property is important for the reactivity of the system and practical timing requirements often do not relate to the deadline of tasks but in particular to end-to-end deadlines. Tasks deadlines are auxiliary values which emerge in the course of design and implementation.

Definition 9 (End-to-end deadline)**.** *An end-to-end deadline d_C^{e2e} is the maximum tolerable end-to-end latency of a cause-effect chain C.*

Robustness Margins For a Single Cause-Effect Chain

While the maximum end-to-end latency of a cause-effect chain is an important property of the present design, robustness margins indicate to what degree the tasks in a cause-effect chain can be extended in terms of their worst-case response time, resp. LET, without violating the end-to-end deadline.

Theorem: Robustness test for a single time-triggered cause-effect chain

Let $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ be a time-triggered cause-effect chain with the property that each task τ_k^c in C has a worst-case response time $WCRT_{\tau_k^c}$ satisfying the task deadline $d_{\tau_k^c}$. Moreover, the cause-effect chain C also satisfies its end-to-end deadline d_C^{e2e}. Let $\Delta WCRT_{\tau_k^c} \geq 0$ be the increase in the worst-case response time of each task τ_k^c in C that is caused by a software update.

The time-triggered cause-effect chain C still satisfies its end-to-end deadline d_C^{e2e} under the increase of the worst-case response times of its tasks τ_k^c by $\Delta WCRT_{\tau_k^c}$, if

$$\forall \tau_k^c \in C: \ \Delta WCRT_{\tau_k^c} < RM^C(\tau_k^c).$$

Theorem: Robustness test for a single LET-triggered cause-effect chain

Let $C = (\tau_1^c, \tau_2^c, \dots, \tau_n^c)$ be a LET-triggered cause-effect chain with the property that each task τ_k^c in C satisfies its LET $LET_{\tau_k^c}$. Moreover, the cause-effect chain C also satisfies its end-to-end deadline d_C^{e2e}. Let $\Delta LET_{\tau_k^c} \geq 0$ be a planned increase of the $LET_{\tau_k^c}$ of a task τ_k^c.

The LET-triggered cause-effect chain C still satisfies its end-to-end deadline d_C^{e2e} under the increase of the LETs of its tasks τ_k^c by $\Delta LET_{\tau_k^c}$, if

$$\forall \tau_k^c \in C: \ \Delta LET_{\tau_k^c} < RM^C(\tau_k^c).$$

Robustness Margins For Multiple Cause-Effect Chain

A task of an application may be part of several cause-effect chains. Consequently, increasing the worst-case response time (or LET) of this particular task may have an impact on all cause-effect chains that share this task. We can find new robustness margins $RM(\tau_k^c)$ (instead of: $RM^C(\tau_k^c)$) which guarantee that *all* cause-effect chains in the application still satisfy their deadlines under a disturbance $\Delta WCRT_{\tau_k^c}$ resp. $\Delta LET_{\tau_k^c}$.

Theorem (Robustness test for time-triggered cause-effect chains)

Let $\mathcal{S}$ be a set of time-triggered cause-effect chains. Each cause-effect chain $C = (\tau_1^c, \tau_2^c, \dots, \tau_n^c)$ in $\mathcal{S}$ has the property that the worst-case response time $WCRT_{\tau_k^c}$ of every task τ_k^c in C satisfies the task deadline $d_{\tau_k^c}$. Moreover, each cause-effect chain C in $\mathcal{S}$ also satisfies its end-to-end deadline d_C^{e2e}. Let $\Delta WCRT_{\tau_k^c} \geq 0$ be the increase in the worst-case response time of a task τ_k^c in $\mathcal{T}$ that is caused by a software update.

All time-triggered cause-effect chain C in $\mathcal{S}$ still satisfy their end-to-end deadlines d_C^{e2e} under the increase of the worst-case response times of its tasks τ_k^c by $\Delta WCRT_{\tau_k^c}$, if

$$\forall C: \forall \tau_k^c \in C: \ \Delta WCRT_{\tau_k^c} < RM(\tau_k^c).$$

Theorem (Robustness test for LET-triggered cause-effect chains)

Let $\mathcal{S}$ be a set of LET-triggered cause-effect chains. Each cause-effect chain $C = (\tau_1^c, \tau_2^c, \dots, \tau_n^c)$ in $\mathcal{S}$ has the property that every task τ_k^c in C satisfies its LET

$LET_{\tau_k^c}$. Moreover, each cause-effect chain C in $\mathcal{S}$ also satisfies its end-to-end deadline d_C^{e2e}. Let $\Delta LET_{\tau_k^c} \geq 0$ be a planned increase of the $LET_{\tau_k^c}$ of a task τ_k^c. All LET-triggered cause-effect chain C in $\mathcal{S}$ still satisfy their end-to-end deadlines d_C^{e2e} under the increase of the LETs of its tasks τ_k^c by $\Delta LET_{\tau_k^c}$, if

$$\forall C : \forall \tau_k^c \in C : \; \Delta LET_{\tau_k^c} < RM(\tau_k^c).$$

2.2.2 Graphical Outputs

The tool **TORO** provides several graphical outputs. Graph 1 summarizes the results, graph 2 and graph 3 offer detailed insights into the analysis as presented in the following chapter 2.3.

Graph 1: Summary

Graph 2: Reachability Graph

Graph 3: Interval Graph.

2.3 Computation of Latencies and Robustness Margins

Please refer to the paper in the appendix for details.

3 Usability

This chapter explains how to install and use the tool **TORO**.

3.1 Installation

This section explains how to install the tool **TORO**.

- Install a Python Interpreter (version 2.7 or 3). You may want to install a distribution (e.g. Python(x,y)) which contains already many useful packages.
- The tool **TORO** will be delivered in a zipped folder `Toro.zip`. This folder has the following content:
  - **Main script `toro_main.py`**,
  - **Libraries (pycpa, toro) in the folder `libs`**. The libraries under `./libs` are dynamically included. They do not need to be installed unless you want to change their location.
  - **Input folder `data`.** Systems to be analyzed can be stored here.
- Unzip the zip folder to your preferred destination.

3.2 Use

This section gives an overview how to use the tool **TORO**.

- Open a terminal and change to the `/Toro` folder.
- The tool **TORO** is called via the script `toro_main.py`. A path is passed to the script as an argument to specify the location of the system to be analyzed. It is recommended to store the system to be analyzed in the provided `data` folder. If several systems are to be analyzed, a subfolder can be created in `data` for each system to be analyzed. The folder structure is illustrated in Figure 3.
- An exemplary call of the tool would then look as follows:

```
user@computer:~/Toro$ python toro_main.py ./data
```

- The tool **TORO** then executes, searching for specified systems. From the set of found systems, one ore more can interactively be selected for analysis. Then the same procedure is repeated for each system to be analyzed:
  - User query to identify the type of system and verify assumptions about the system.
  - Parsing the csv-files which specify the system (see Section 3.3.
  - Calculation of maximum end-to-end latencies and robustness margins.
  - Generation of diagrams.
  - Creation of a file containing numerical results. The file is written to the folder in which the analyzed system is specified.

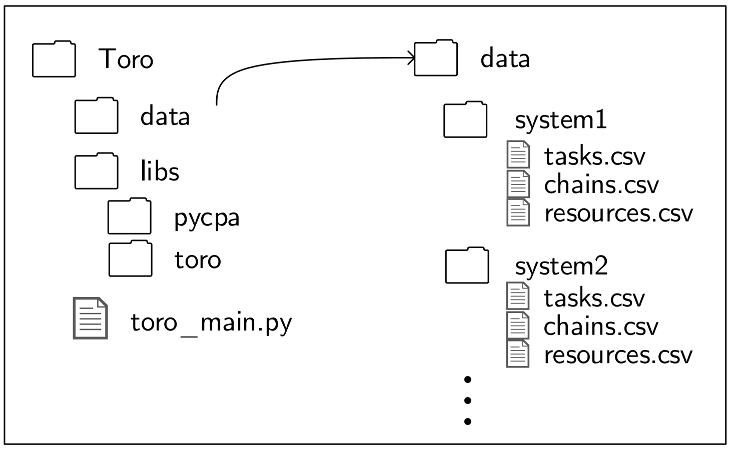


Figure 3: Folder structure

3.3 Inputs

This section explains how to specify systems that the tool **Toro** should analyze.

Firstly, **Toro** calls the method `get_system_dirs(dir)` and searches the specified folder for systems to be analyzed. Each individual system should be encapsulated in a dedicated folder as illustrated in Figure 3. This system folder should contain three semicolon-separated csv-files, namely

- `chains.csv`,
- `resources.csv`,
- `tasks.csv`.

These files can be created in Microsoft Excel, for example, and can then be exported as CSV files. If **Toro** finds more than one system in the specified top-level folder (e.g. `./data`), the user can choose from a list which system(s) should be examined. An example output would look like this:

```
The following systems have been found:
       1:  system1
       2:  system2
       2:  system3
       0:  all
To select systems enter their ID or a comma-separated list of
IDs.  For instance, enter 1 to select the first of the listed
systems, enter 1,3 to select the first and the third of the
listed systems, or enter '0' to select all systems:
```

The following sections 3.3.1-3.3.3 explain the structure of the CSV files. Then sections 3.3.4-3.3.6 demonstrate the three possible types of use cases and which information needs to be filled in.

3.3.1 Input File ’resources.csv’

The `resources.csv` file describes the execution platform of the system by listing the different resources (CPUs, CAN bus etc.) and the scheduling policy that is applied on each resource. The `resources.csv` file should have the following column headers:

name Type: String
: Each computing core and each data bus, on which at least one chain task is executed, is represented by a so-called *resource*. This field specifies the name of the resource.

scheduler Type: String
: e.g., `SPPScheduler` or `SPNPScheduler`

3.3.2 Input File ’tasks.csv’

The `tasks.csv` file specifies tasks that are executed on the platform, it should have the following column headers:

task_name Type: String
: Unique task ID

period Type: Integer
: Activation period

offset Type: Integer
: Release offset

priority Type: Integer
: Note that 0 is the highest priority.

wcet Type: Integer
: Worst Case Execution Time

resource Type: String
: Resource that services the task. The name should match a resource from file `pycpa_resources.csv`.

bcrt Type: Integer
: Best Case Response Time

wcrt Type: Integer
: Worst Case Response Time

let Typ: Integer
: Logical Execution Time

3.3.3 Input File 'chains.csv'

The `chains.csv` file specifies which tasks are part of each cause-effect chain; it should have the following column headers:

chain_name Type: String
Unique cause-effect chain ID

e2e_deadline Type: Integer
End-to-end deadline

members Type: String
The member tasks must be listed in correct order and the task names must match those in the task definitions. The list of member tasks comprises as many cells in a row as needed.

3.3.4 Use Case 1:Cause-effect chains with BET tasks, WCRT for each chain task known

- applicable to many systems
- the name, period, offset, WCRT of every task in a cause-effect chain must be known

For *Use Case 1* given that

- all clocks in the system are synchronized,
- all tasks in the cause-effect chains are BET tasks,
- task deadlines are implicit,

(which is checked in an interactive user query), it is sufficient to specify the following details of the entire systems. An example system of type *'Use Case 1'* is depicted in Figure 4.
For the example system, the file `resources.csv` would contain:

name	**scheduler**
unknown	unknown

With regard to the tasks that are part of the cause-effect chains to be analyzed, we have in `tasks.csv`

task_name	**period**	**offset**	**priority**	**wcet**	**resource**	**bcrt**	**wcrt**	**let**

Note that a number of fields in `resources.csv` and `tasks.csv` can be left empty or declared as 'unknown' because the WCRTs are already available and do not need to be computed from these parameters.

The cause-effect chains in `chains.csv` are specified by

chain_name	**e2e_deadline**	**members**		
chain_1		T1	T3	T2
chain_2		T3	T4	

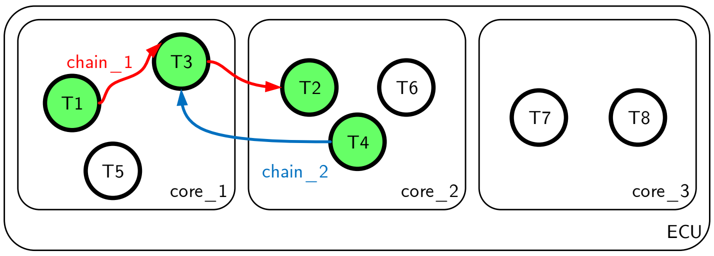


Figure 4: Use case 1; relevant parts of the system for the analysis are marked in color.

3.3.5 Use Case 2: Cause-effect chains with BET tasks, WCRT of each chain task computable

- only applicable to systems with static priority preemptive (SPP) und static priority non-preemptive (SPNP) scheduling
- not only tasks that are part of cause-effect chains must be specified but also the entire background load with all parameters

For *Use Case 2* given that

- all clocks in the system are synchronized,
- ALL tasks are BET tasks (not only those in the listed cause-effect chains),
- task deadlines are implicit,
- ALL tasks in the system must be known with their parameters,
- ALL resources must be known with their scheduling algorithms (SPP or SPNP) and the task-to-resource mapping,

(which is checked in an interactive user query), it is sufficient to specify the following details of the entire systems. An example system of type *'Use Case 2'* is depicted in Figure 5. For the example system, the file `resources.csv` would contain:

Name	**Scheduler**
core_1	SPPScheduler
core_2	SPPScheduler
core_3	SPNPScheduler

With regard to the tasks that are part of the cause-effect chains to be analyzed, we have not only the chain tasks but also the tasks of the background load with all parameters required to compute the WCRTs.

task_name	**period**	**offset**	**priority**	**wcet**	**resource**	**bcrt**	**wcrt**	**let**

The cause-effect chains are specified again by

chain_name	**e2e_deadline**	**members**		
chain_1		T1	T3	T2
chain_2		T3	T4	

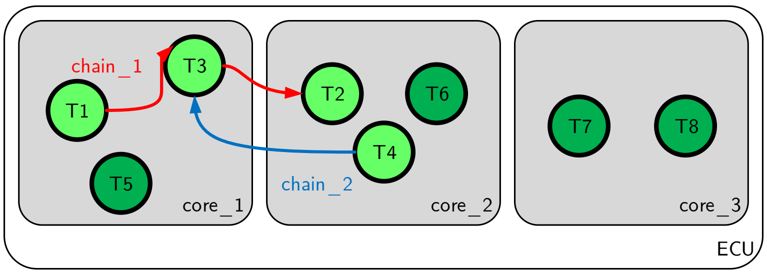


Figure 5: Use case 2; relevant parts of the system for the analysis are marked in color

3.3.6 Use Case 3: Cause-effect chains with LET tasks, LET for each chain task known

- applicable to LET systems

For *Use Case 3* given that

- all clocks in the system are synchronized,
- all tasks in the cause-effect chains are LET tasks,

(which is checked in an interactive user query), it is sufficient to specify the following details of the entire systems. An example system of type *'Use Case 3'* is depicted in Figure 6.
For the example system, the file `resources.csv` would contain:

name	**scheduler**
unknown	unknown

With regard to the tasks that are part of the cause-effect chains to be analyzed, we have in `tasks.csv`

task_name	**period**	**offset**	**priority**	**wcet**	**resource**	**bcrt**	**wcrt**	**let**

The cause-effect chains are specified by

chain_name	**e2e_deadline**	**members**		
chain_1		T1	T3	T2
chain_2		T3	T4	

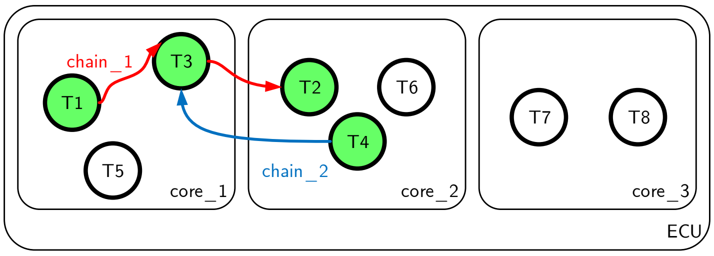


Figure 6: Use case 3; relevant parts of the system for the analysis are marked in color.

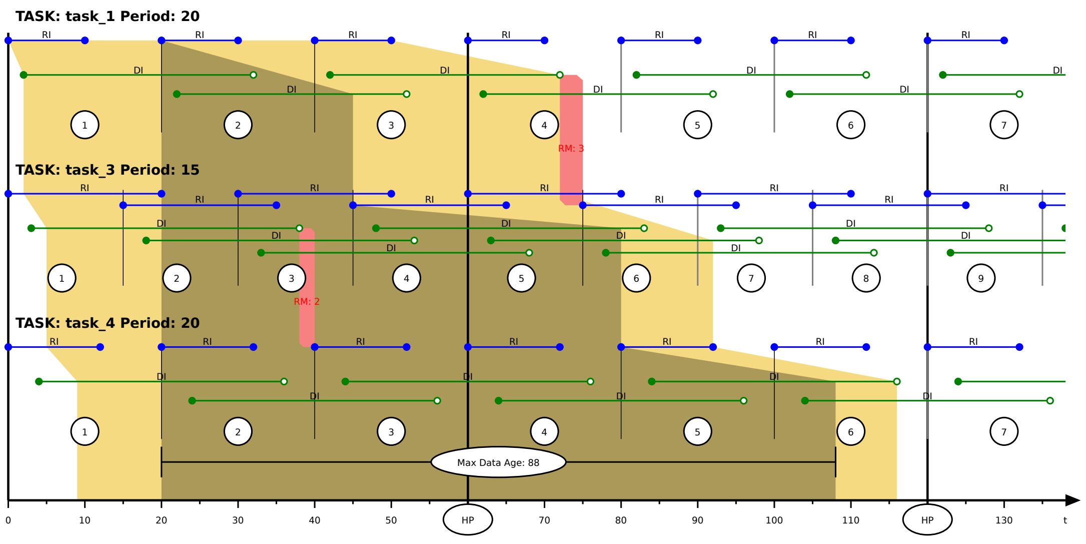


Figure 7: Interval diagram.

3.4 Outputs

Outputs are written into the folder of the specified system.

3.4.1 Log File

To Do.

3.4.2 Interval Diagram

In the interval diagram, the individual jobs are represented as numbered circles. The blue-marked read intervals represent the earliest possible to latest possible time, when a job can read data. The green-marked data intervals bound the period of time in which the output data of a job is available to other jobs. The short vertical lines stand for the period of a task. The long vertical lines show the hyper period (HP).

The yellow area covers all instances of the cause-effect chain which are relevant for the computation of the maximum end-to-end latency and the robustness margins.
The dark area highlights one instance of a cause-effect chains which actually has the maximum end-to-end latency.
The red areas illustrate some selected robustness margins, which are valid for the isolated cause-effect chain (not for the set of specified cause-effect chains in `chains.csv`.

3.4.3 Reachability Graph

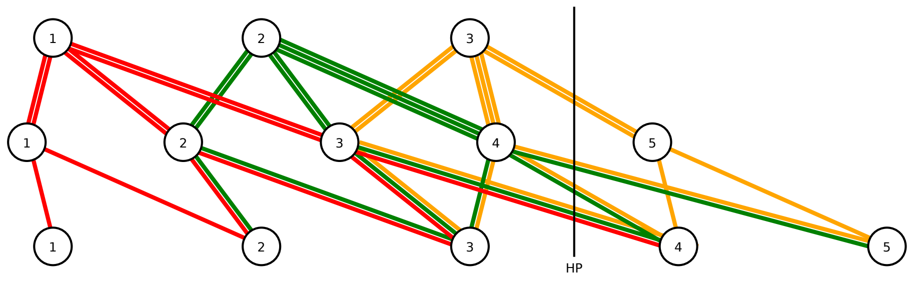


Figure 8: Reachability graph

The reachability graph results from the overlap of read intervals and data intervals of jobs (data flow analysis). In the reachability graph, all possible paths of starting in from the first hyperperiod are represented. These paths are required for the computation of the maximum end-to-end latency and the robustness margins. The colored marking of the individual paths makes it easy to recognize where a path begins and ends. The numbers stand for the respective job number.

3.4.4 Overview Graph

The overview graph represents the specified cause-effect chains as a tasks with data flow dependencies. Moreover, it lists the computed maximum end-to-end latencies for each cause-effect chain. Finally, it indicates the robustness margins for the isolated chains and in the presence of all chains.

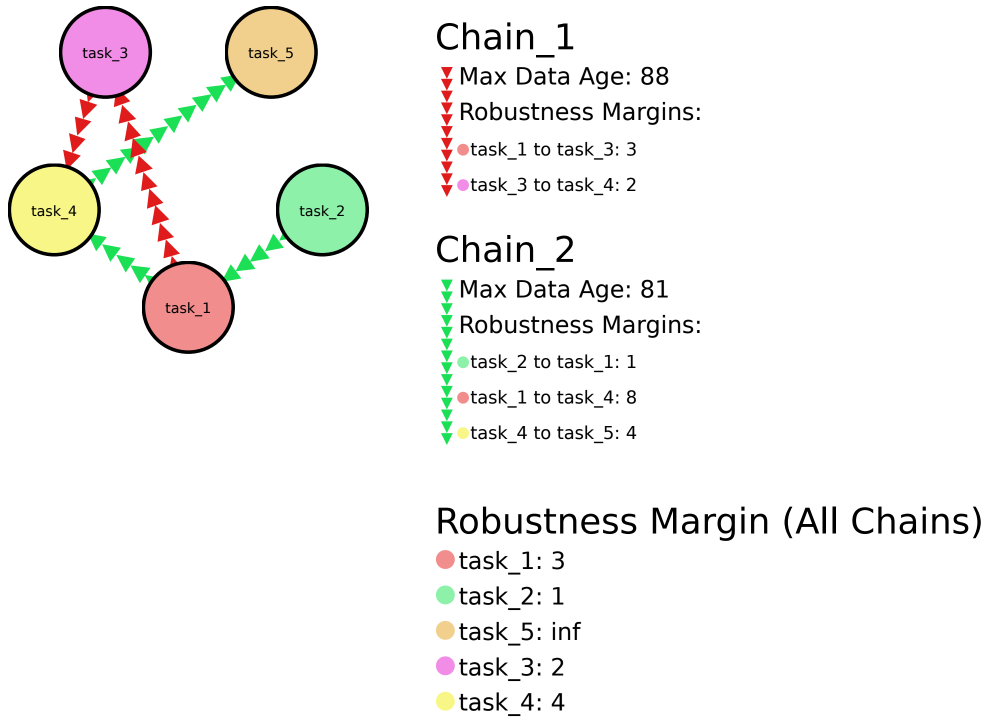


Figure 9: Summarizing diagram.

3.5 Internals

The tool **Toro** relies on the Python packages

- `toro`
- `pycpa`

and the main script

- `toro_main.py`

Details about the `pycpa` package can be found in the online documentation at `https://pycpa.readthedocs.io`. The other two components are discussed in more detail in the following sections 3.5.1 and 3.5.2.

3.5.1 User Script

The main script `toro_main.py` consists of two methods (cf. Figure 10):

get_system_dirs(dir)

The script `toro_main.py` starts with the main method and passes the call parameter (folder path) to the `get_system_dirs(dir)` method.
The method `get_system_dirs(dir)` searches for existing systems under the specified path. If more than one system is found, the user can choose from a list which systems should to be analyzed.

perform_analysis(dir)

The method `perform_analysis(dir)`

- starts a user query to identify the type of system and verify assumptions about the system,
- parses the csv-files which specify the system,
- calculates the maximum end-to-end latencies and robustness margins,
- generates of diagrams,
- logs the numerical results (`RESULTS_LOG.txt`),

for each of the selected systems.

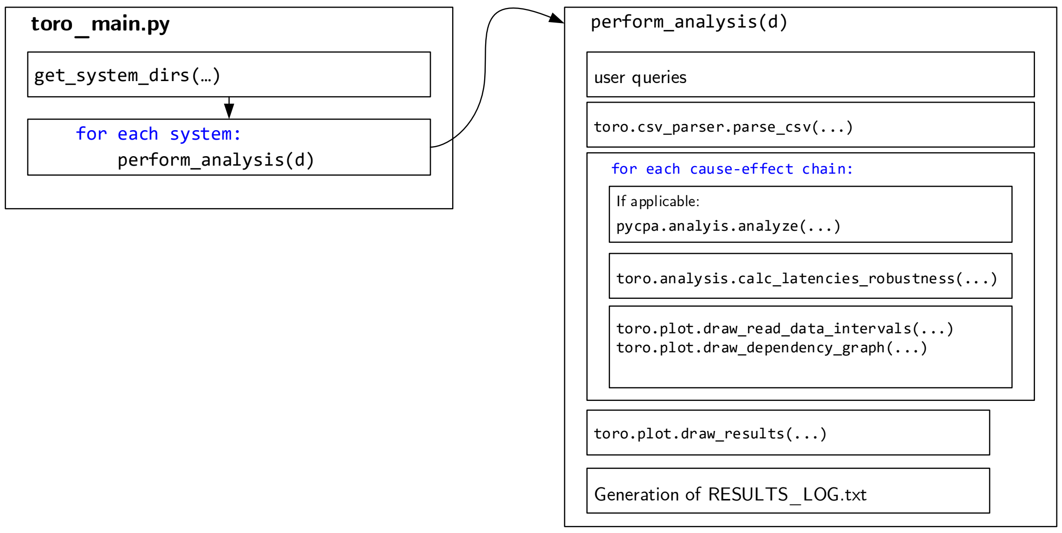

toro_main.py
get_system_dirs(…)
for each system:
perform_analysis(d)
perform_analysis(d)
user queries
toro.csv_parser.parse_csv(...)
for each cause-effect chain:
If applicable:
pycpa.analysis.analyze(...)
toro.analysis.calc_latencies_robustness(...)
toro.plot.draw_read_data_intervals(...)
toro.plot.draw_dependency_graph(...)
toro.plot.draw_results(...)
Generation of RESULTS_LOG.txt

Figure 10: Internals of the main script `toro_main.py`

3.5.2 Toro Package

This section describes the modules in the package `toro`.

model

This module defines the data structures required within the software. It is an extension of the module of the same name from the `pycpa` package.

toro.analysis

This module contains the `calc_latencies_robustness` class. When an object of this class is initialized and a chain is passed as an argument, then the maximum latency and the robustness margins for each task are computed.

toro.csv_parser

This module reads the CSV-files from the specified folder and transfers them into the software's internal data structure. If files are not available or data is not properly defined, an error message is issued. The parser is called via the `prase_csv` class, which requires the folder path of the system to be analyzed as argument.

toro.plot

The module plots creates graphical representations of the analysis results; the three different types of graphs have been discussed in the section 3.4. The module consists of several classes, each is designed for one of the graphical output formats.

`image` The `image` class provides the basic functions to draw circles, lines, text and other elements in an SVG image.

`draw_read_data_intervals` This class draws the read and data intervals of jobs in a representative time window on the basis of the analysis results. In addition, the computed properties – maxmimum end-to-end latency and the robustness margins for the isolated chain – can be drawn using the options "all", "none", "first" and "last". A further option is to include the "Dependency Polygon":

- max_data_age = all | first | last | none
- robustness_margin = all | first | last | none
- dependency_polygon = True | False.

`draw_dependency_graph` The reachability graph illustrates possible instances of a cause-effect chain. As the call parameter are the analysis results required.

`draw_results` The relevant results of all tasks and cause-effect chains of a system are displayed in an overview graph. Chains and tasks are used as call parameters.

References

[1] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics,” in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*, IEEE Communications Society, 2009.