UNIVERSITY OF TWENTE.

Analysing gCSP Models Using Runtime and Model Analysis Algorithms

Communicating Process Architectures 2009

Maarten Bezemer, Marcel Groothuis and Jan Broenink

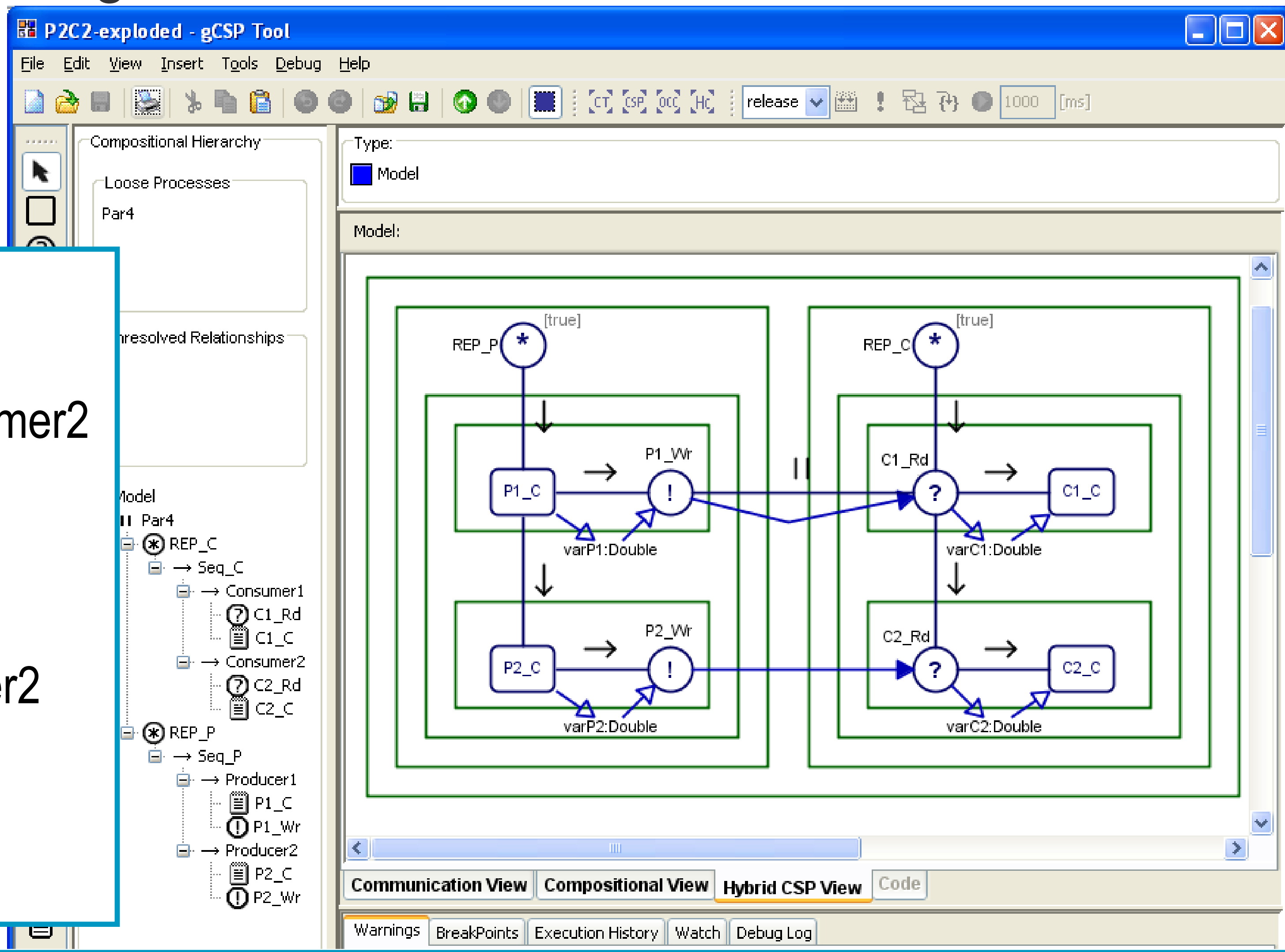
Control Engineering, University of Twente, The Netherlands

Contents

- Introduction
- Runtime Analysis Algorithm
- Model Analysis Algorithm
- Conclusions

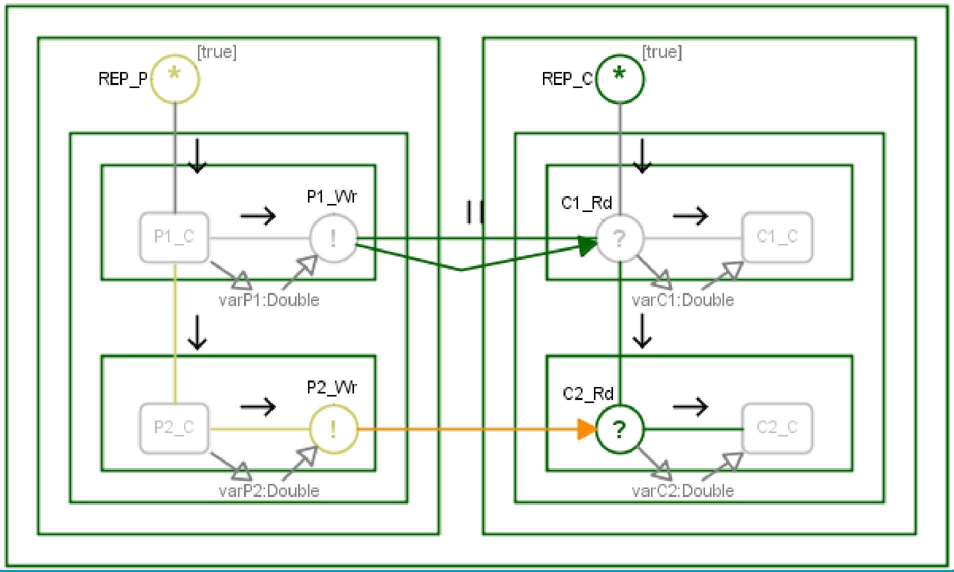
Introduction

- CSP usage at Control Engineering
  - Modelling tool → gCSP

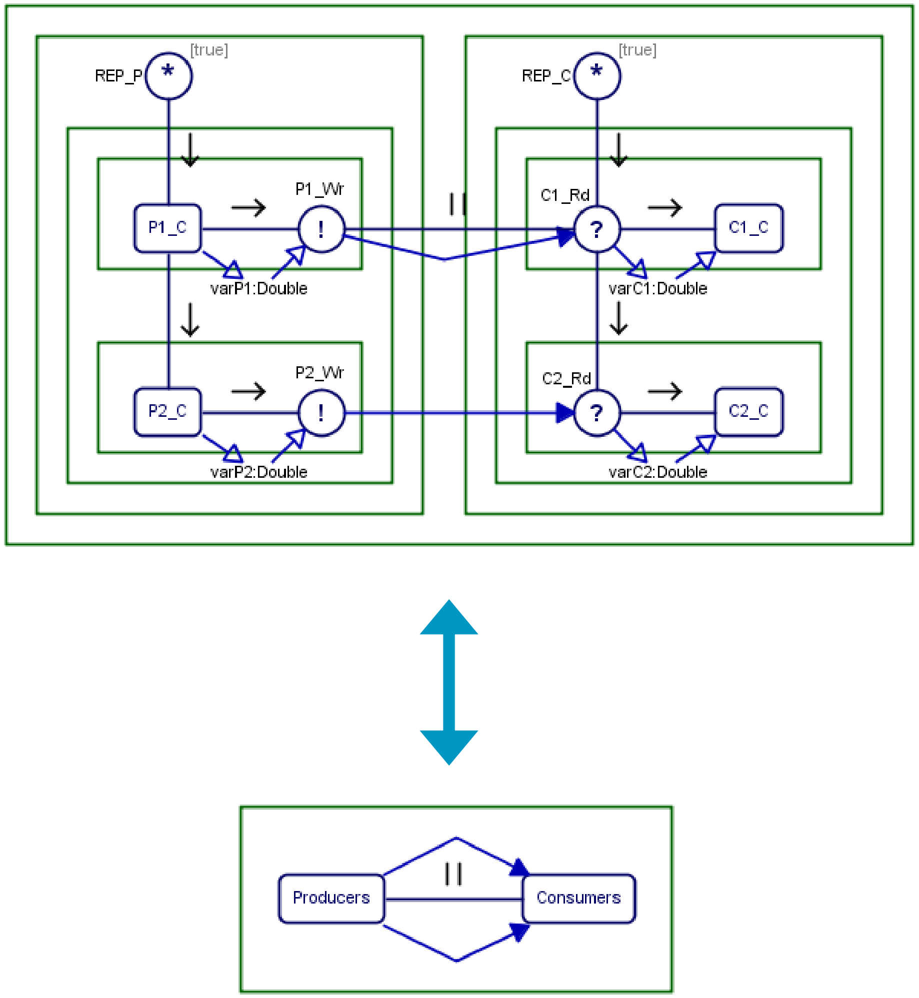


Par4 = REP_P || REP_C
REP_C = Seq_C; REP_C
Seq_C = Consumer1; Consumer2
Consumer1 = C1_Rd; C1_C
Consumer2 = C2_Rd; C2_C
REP_P = Seq_P; REP_P
Seq_P = Producer1; Producer2
Producer1 = P1_C; P1_Wr
Producer2 = P2_C; P2_Wr

- CSP usage at Control Engineering
  - Modelling tool → gCSP
    - Code generation for (robotic) controllers
      - Using Communicating Threads (CT) library
    - Debugging possibilities while running the code
      - Animating the model (processes and channels)
      - Stepping through model, while showing channel values



- Designer Point of View
  - Detailed modelling
  - Lots of small processes
- Executing Point of View
  - Fast code
  - A few bigger processes
- Both Points of View conflict!

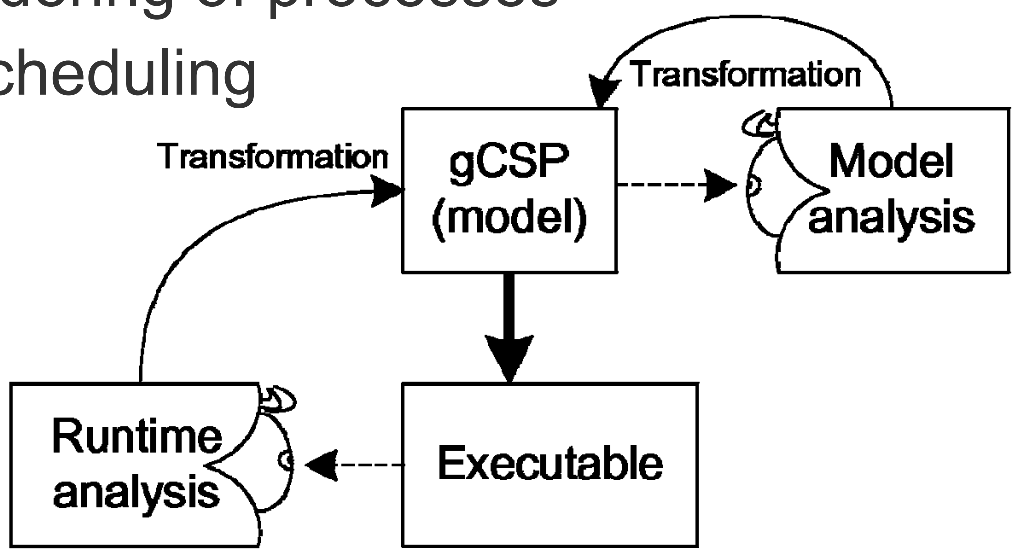


Introduction Solution

- Translate Designer PoV to Executing PoV

- Requires
  - Analysis of the gCSP model
  - Model transformation

- Solution: two analysis algorithms
  - Runtime analysis for static ordering of processes
  - Model analysis for process scheduling



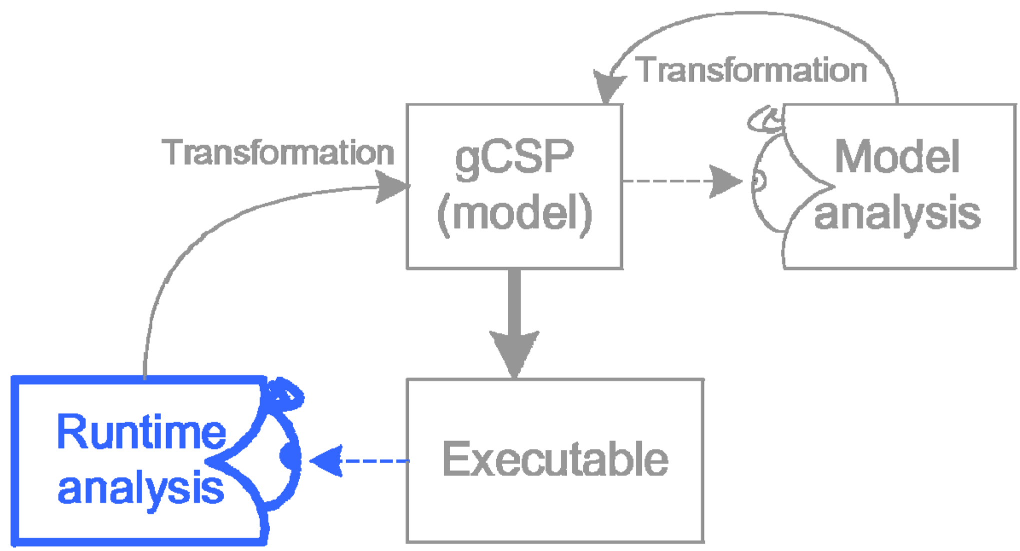
Contents

UNIVERSITY OF TWENTE.

- Introduction
- Runtime Analysis Algorithm
  - Introduction
  - Algorithms
  - Results
- Model Analysis Algorithm
- Conclusions

Runtime Analysis Algorithm

- Why static ordering of processes
  - No complex scheduler required
  - Possibility for grouping of processes
- Goal of Runtime Analysis Algorithm
  - Find a static running order for the processes



Runtime Analysis Algorithm Processes

- Process states
  - New — Process is created
  - Ready — Process is ready to be started
  - Running — Process is started and still running
  - Blocked — Process is blocked
  - Finished — Process is ended

- Algorithm mainly uses Finished state to determine the static running order

Runtime Analysis Algorithm Notations

- Set of chains
  - Clear view of groups of processes
  - Cross-Reference types
    - To other chain
    - To start of same chain
  - Comparable with a CSP Trace

```
D → C → F → B → (B,D*)
B → E → A → C → (D)
[start] → A → B → C → D → (B)
```

- Traces
  - Finished processes of running model
  - For demonstration purposes

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow B \rightarrow E \rightarrow A \rightarrow C \rightarrow E \rightarrow A \rightarrow C \rightarrow D \rightarrow C \rightarrow F \rightarrow \ldots$

Runtime Analysis Algorithm Algorithm

- Process Ordering Rules
  - Chains with no cross-refs (the active chain is not finished yet)
    - Add processes to chain

- Rules
  - If the state of a process changes to Finished add it to the end of the active chain.
  - If a process is Finished and is already present in the active chain, it will become a cross-reference of this chain pointing to a chain starting with this process.

A → B → C → B → D → E → F → B

B → D → E → F → (B*)

[start] → A → B → C → (B)

Runtime Analysis Algorithm Algorithm

UNIVERSITY OF TWENTE.

- Process Ordering Rules
  - Chains with cross-refs (the active chain got finished already)
    - Perform validation on chains

- Rules
  - If the active process does not match the Finished process the chain must be split.

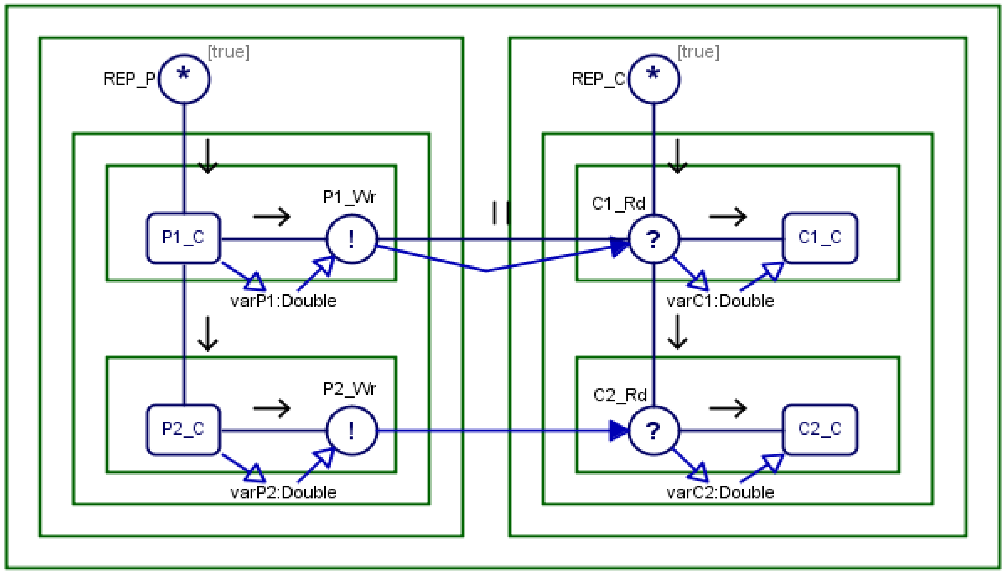
Rules for 'chains with no cross-references' applied

… → *D* → *E* → *F* → *B* → *D* → *G* → *H*

```
G → H
E → F → (B)
B → D → (E,G)
[start] → A → B → C → (B)
```

Runtime Analysis Algorithm Results

- Set of chains as expected
  - All processes could be placed in one big process
    - Writer-Reader combinations can be removed
    - Channels become internal variables

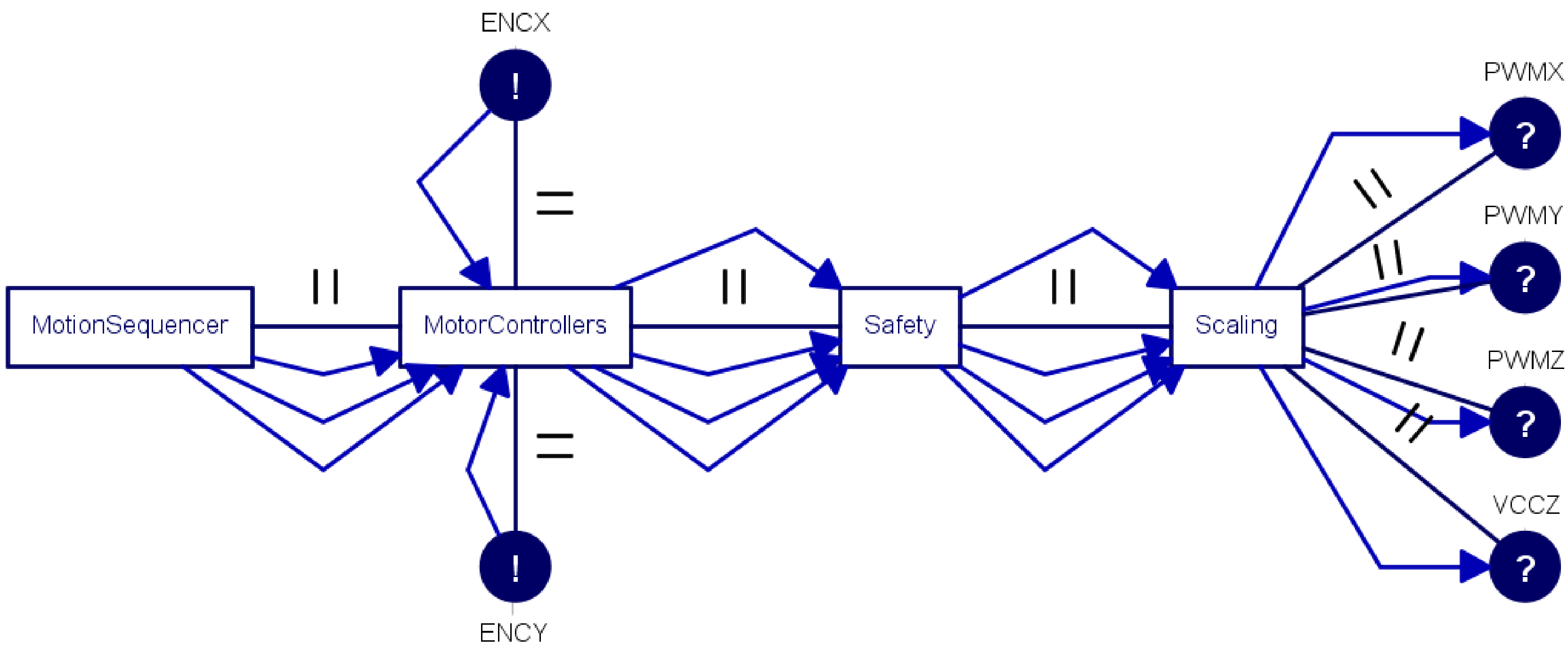



```
P1_C →C1_Rd →C1_C →P1_Wr →P2_C →
      P2_Wr →C2_Rd →C2_C →(P1_C*)
[start]→(P1_C)
```

Runtime Analysis Algorithm Results

- Scalability of the algorithm
  - Complex traces
    - Hard to verify results
  - Static ordering available

- Working Cartesian plotter model



- Working Cartesian plotter model

1 Sc_Rd8 → DoubletoBooleanConversion → Sc_Wr17 → Sa_Wr8 → Sa_Rd4 → MC_Wr4 → Sa_Rd7 → MC_Wr7 → Sa_Rd6 → MC_Wr6 → Sa_Rd5 → MC_Wr5 → Sa_Rd_ESX2_2 → Sa_Rd_ESX2_1 → Sa_Rd_ESX1_2 → Sa_Rd_ESX1_1 → MC_Rd12 → MC_Rd13 → Safety_X → Sa_Rd_ESY1 → Sa_Rd_ESY2 → Safety_Y → Safety_Z → MC_Rd1 → MS_Wr1 → MC_Rd2 → MS_Wr2 → Sa_Wr9 → Sc_Rd9 → MC_Rd3 → LongtoDoubleConversion → Controller → MS_Wr3 → Sc_Rd10 → Sa_Wr10 → (Sc_Rd11)

2 Sc_Rd11 → DoubletoShortConversion → Sc_Wr14 → Sc_Wr15 → Sc_Wr16 → Sa_Wr11 → (Sc_Rd8, HPGLParser)

3 MC_Rd12 → MC_Rd13 → Sa_Rd_ESX2_2 → Sa_Rd_ESX2_1 → Sa_Rd_ESX1_2 → Sa_Rd_ESX1_1 → MC_Rd1 → MS_Wr1 → Safety_X → Sa_Rd_ESY1 → Sa_Rd_ESY2 → Safety_Y → Safety_Z → MC_Rd2 → MS_Wr2 → MC_Rd3 → LongtoDoubleConversion → Controller → MS_Wr3 → Sa_Wr9 → Sc_Rd9 → Sc_Rd10 → Sa_Wr10 → (HPGLParser)

4 HPGLParser → (MC_Rd12, Sc_Rd11)

→ [start] → MC_Rd12 → MC_Rd13 → HPGLParser → MS_Wr1 → MC_Rd1 → MC_Rd2 → MS_Wr2 → MC_Rd3 → LongtoDoubleConversion → Controller → MS_Wr3 → MC_Wr5 → Sa_Rd5 → MC_Wr6 → Sa_Rd6 → MC_Wr7 → Sa_Rd7 → MC_Wr4 → Sa_Rd4 → (HPGLParser)

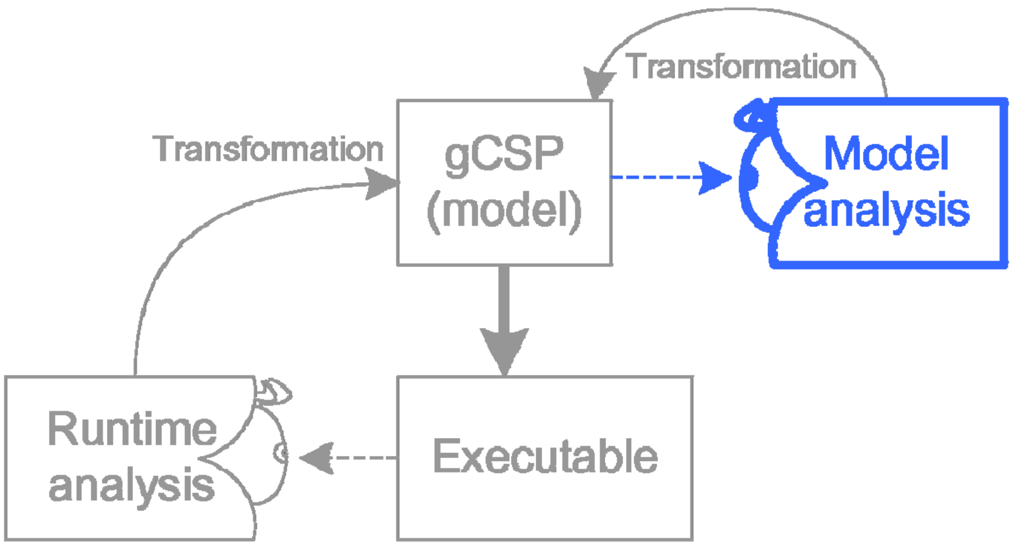
Contents

Model Analysis Algorithm

UNIVERSITY OF TWENTE.

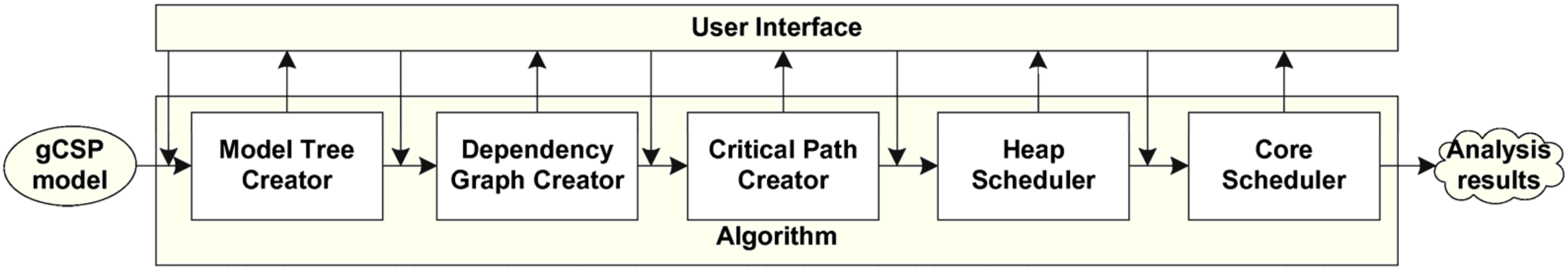
- More towards model analysis/ model transformation
  - What processes are related?
  - How to schedule large models onto a target system?

- Goal
  - Schedule processes on cores/ networked nodes

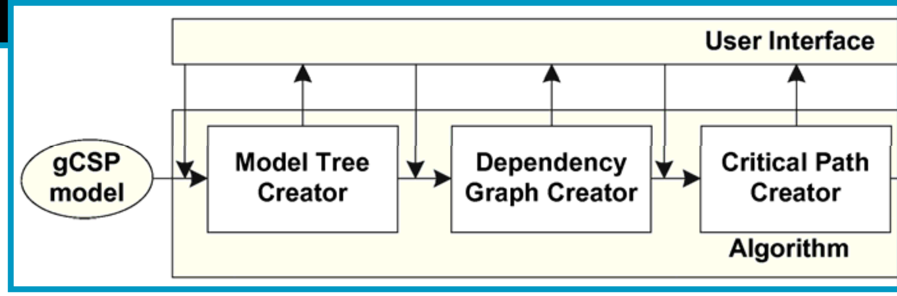


Model Analysis Algorithm Architecture

- Algorithm Architecture
  - Build modular
- gCSP model & User Interface feed the algorithm with data
  - Process weights (or execution times)
  - Available cores or networked nodes
  - Communication (setup) time

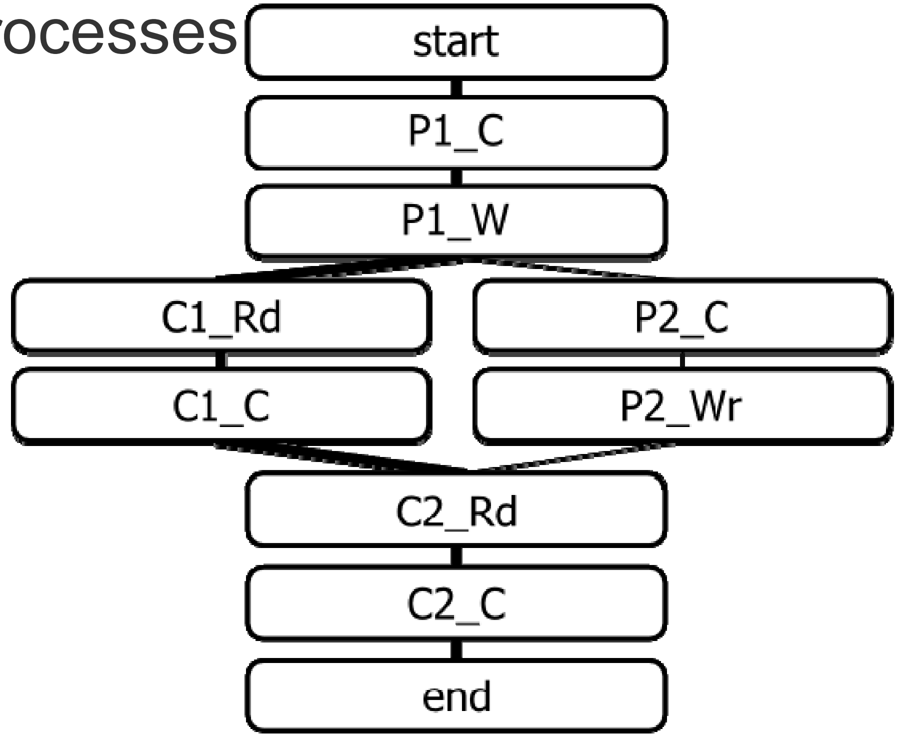


Model Analysis Algorithm Algorithms



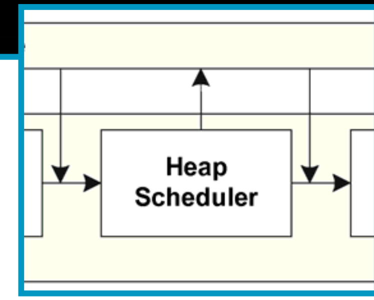
- Model Tree Creator
  - Recreates the model tree
  - Only for displaying purposes for the user interface

- Dependency Graph Creator
  - Finds dependencies between processes
    - Sequential relations
    - Channels

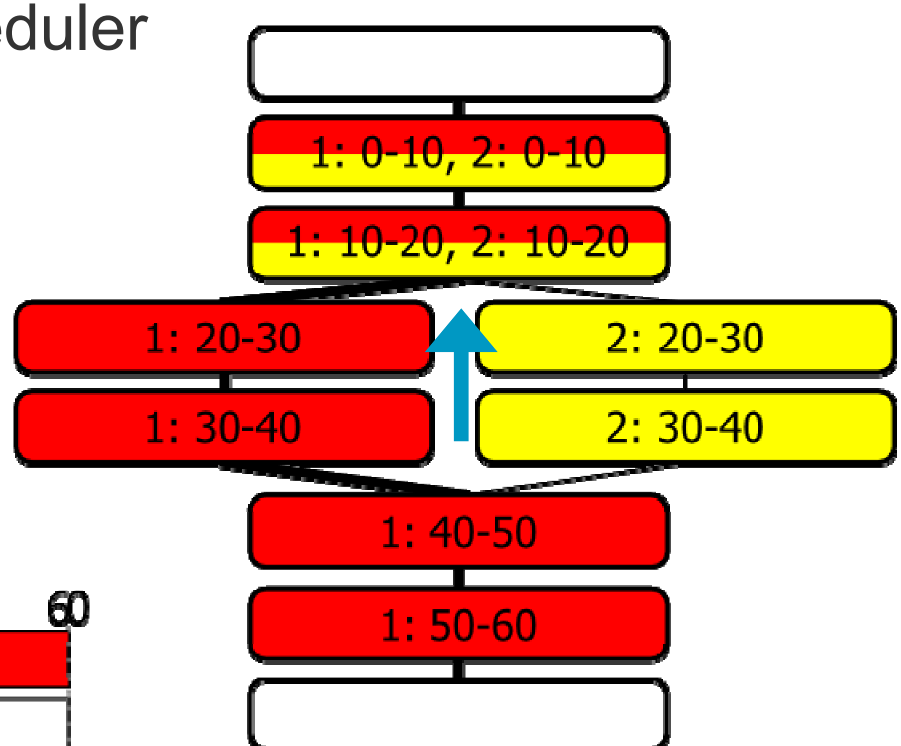


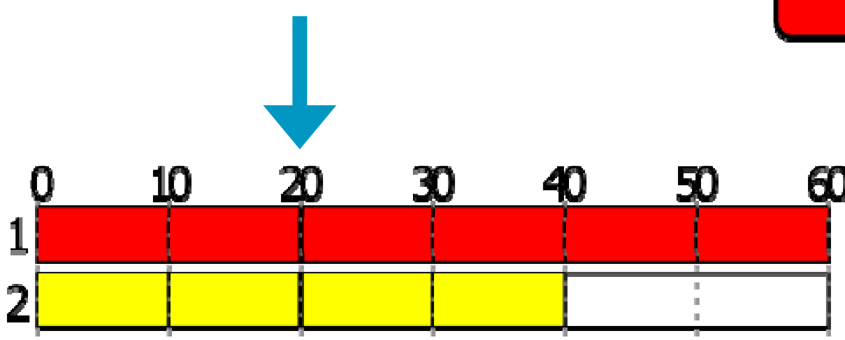
- Critical Path Creator
  - Finds the critical path using the dependencies

Model Analysis Algorithm Heaps

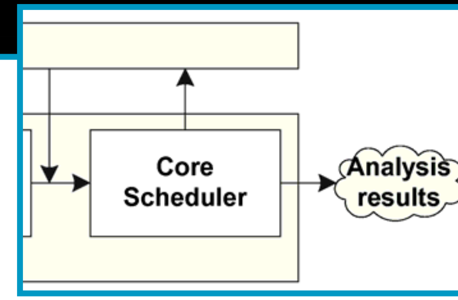


- Heaps
  - Groups of ‘related’ processes
  - Influenced by
    - Process weight
    - Communication (setup) time
  - Reduce complexity of core scheduler
- Index blocks
  - Subdivision of heaps
  - When multiple outgoing dependencies are available

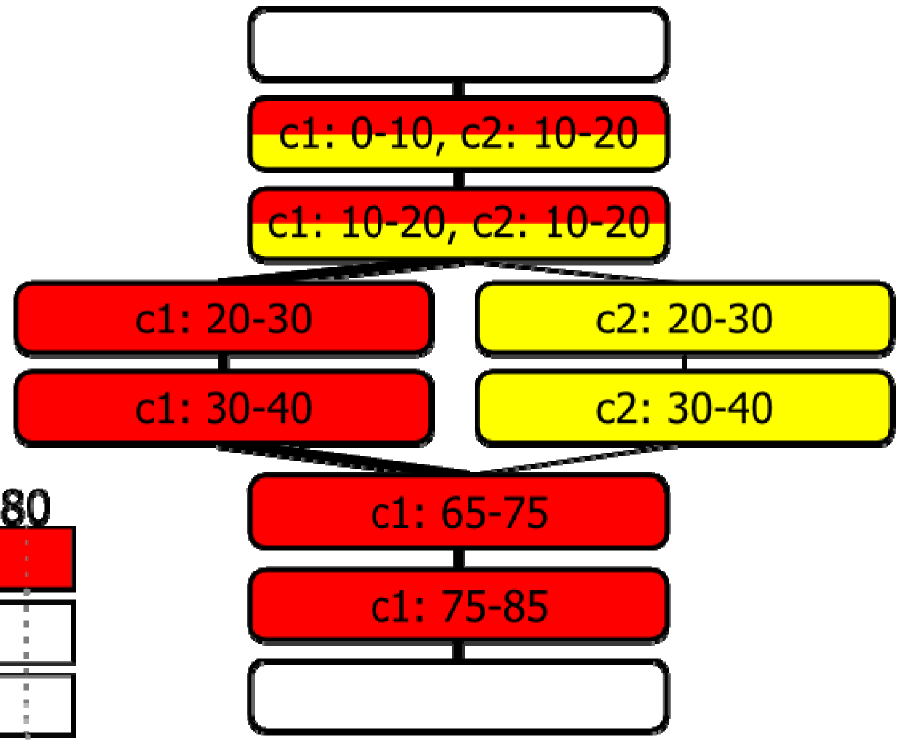


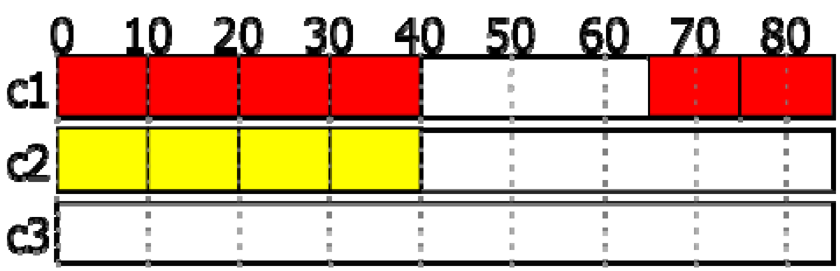


Model Analysis Algorithm Cores



- Cores
  - Groups of processes to be scheduled on the same core/ networked node
  - Find optimum for end time
    - Amount of cores
    - Relative speed of cores





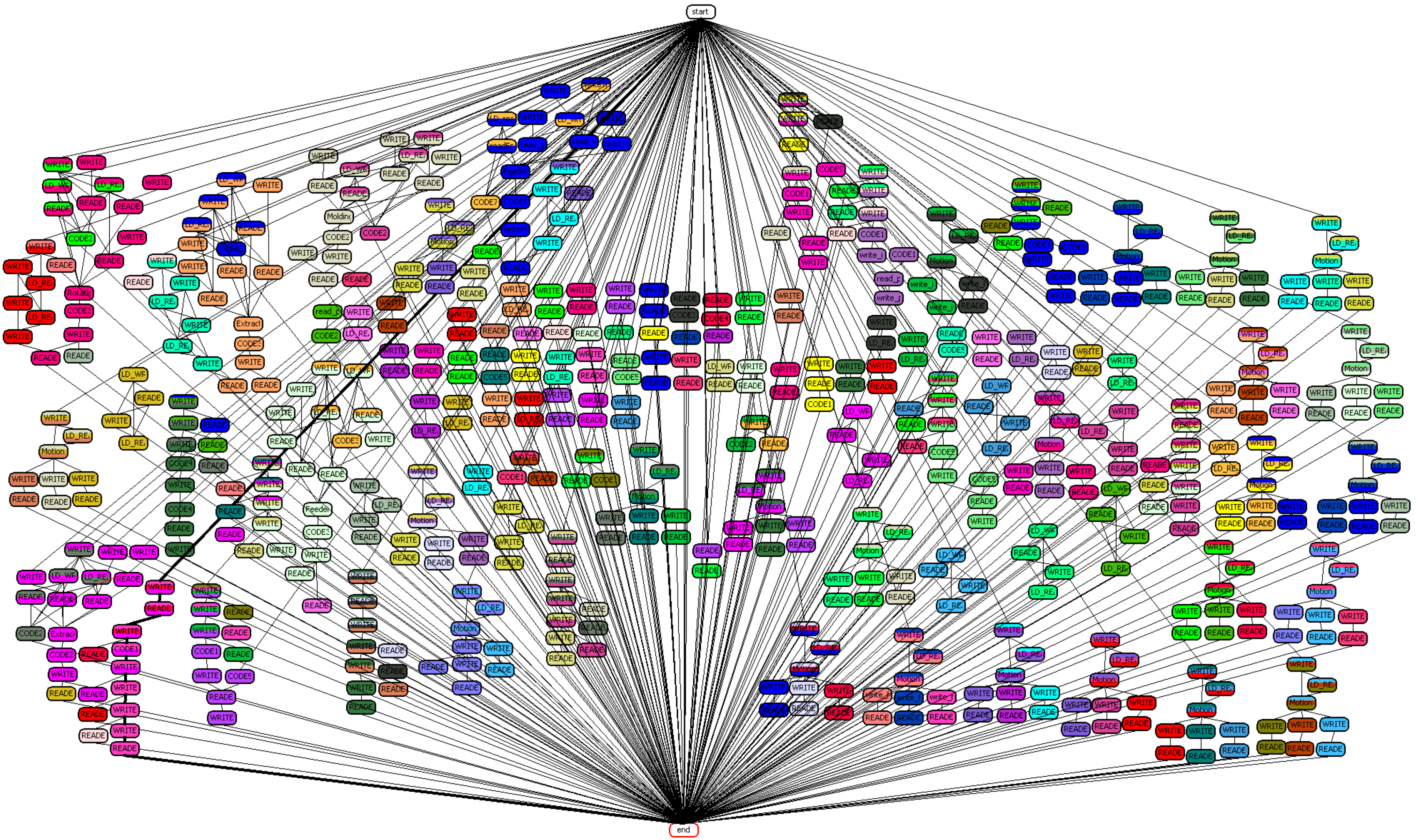
Model Analysis Algorithm Results

- The processes are optimally scheduled
  - For the given process weights
  - For the given communication times
  - For the given target systems (mostly)

- Scalable for models of real-life setups
  - Cartesian plotter model
  - Production cell model
    - 597 Processes
    - 210 Heaps
    - ~50 Cores for optimal ending time

Model Analysis Algorithm Results

UNIVERSITY OF TWENTE.



Contents

- Introduction
- Runtime Analysis Algorithm
- Model Analysis Algorithm
- Conclusions

Conclusions

- Both analysis algorithms work as expected
  - Functional
  - Scale well
- Both analysis algorithms complement each other
  - Runtime analysis algorithm
    - Groups processes into bigger processes
    - Reduces the amount of context switches
  - Model analysis algorithm
    - Schedules processes onto multiple cores
    - Reduces the amount of network communication
  - Both reduce execution time
    - Without losing concurrency aspects

Recommendations

- Refinements are necessary
  - Better representation of the results
  - Include more CSP constructs
  - Support for allocating specific processes on a core
  - Better support for networked nodes

- Next steps
  - Include model transformations after the analysis phase
  - Implement the algorithms in the gCSP2 tool

Runtime Analysis Algorithm Algorithm

UNIVERSITY OF TWENTE.

- The active chain should be split at process **B** when process p is unexpected, but a chain starting with process **B** is present.
  - Compare the processes after **B** with the chain starting with process **B** → equal!
  - Remove the remaining process (**E**) in the active chain starting at process **B**.
  - Add the cross-references (**G**) to the chain starting with **B** if they are not present at this chain.
  - Create a new chain starting with **I** and make it the active chain.

```
G → H → B → D → (G*)
E → F → (B)
B → D → (E,G)
A → B → C → (B)
```

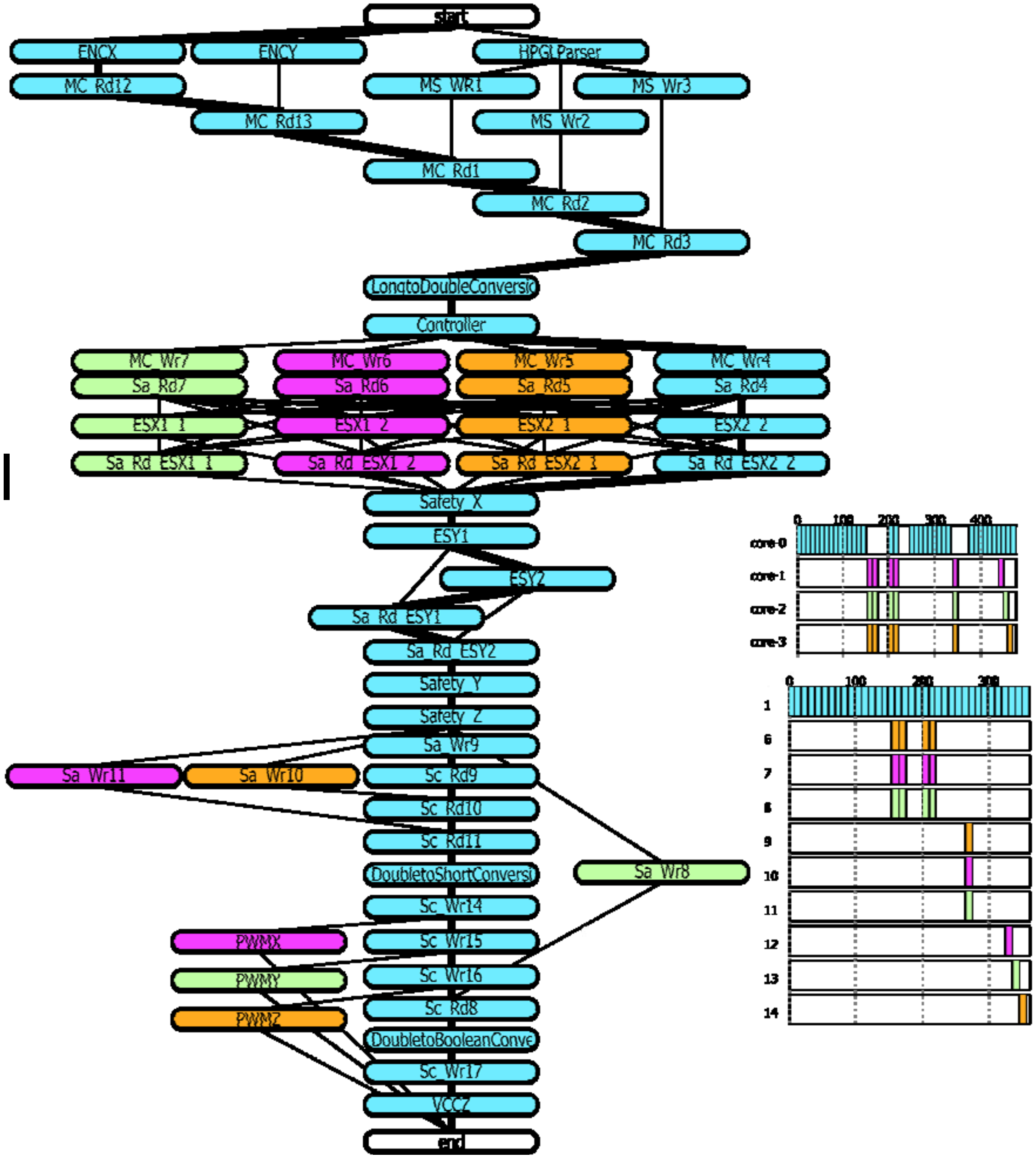
```
I
G → H → (B, I)
E → F → (B)
B → D → (E,G)
A → B → C → (B)
```

A → B → C → B → D → E → F → B → D → G → H → B → D → G → H → ***I***

Cartesian plotter

- 56 processes
- 10 heaps
- ~4 cores

- 1 core is almost optimal



Ongoing Work

- gCSP2
  - Eclipse based
  - Much more stable compared to gCSP