# TACoS: A tool for MTL controller synthesis

## Tool presentation SEFM'21

Till Hofmann  Stefan Schupp



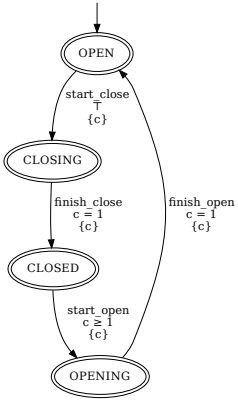
2021-12-10

# Motivation



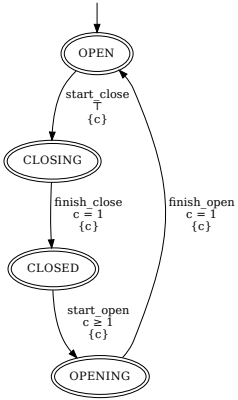
# Motivation

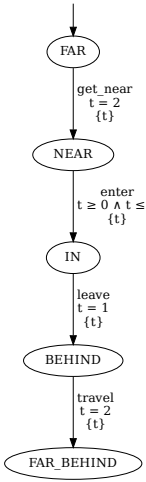




# Motivation





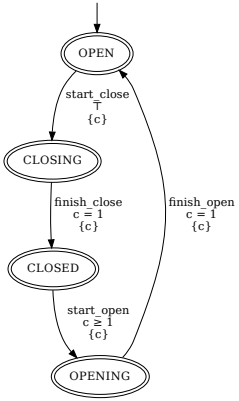
# Motivation

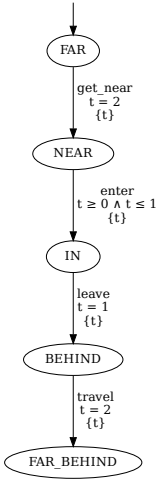
The gate should be closed before the train enters it:

$$\mathit{enter}\ \mathsf{R}\ \neg \mathit{finish\_close}$$

The gate should open after 5 time units:

$$\mathsf{G}(\mathit{finish\_close} \rightarrow \mathsf{F}_{\leq 5}\mathit{start\_open})$$



# Motivation: Controller Synthesis

Given:

- The plant, a timed automaton to be controlled
- A partitioning of the actions into *controller* and *environment* actions
- A specification in MTL, expressing the *undesired behavior*

**Goal**

Select controller actions such that for all possible environment actions, the resulting execution traces satisfy the specification

# Motivation

TACoS:

- A tool for *Timed Automata Controller Synthesis* for MTL specifications
- C++ tool library to synthesize controllers
- Implements an approach described by [Bouyer et al., 2006]

In this talk:

1. An intuitive introduction to the approach
2. Several techniques to improve the performance
3. Evaluation of those techniques in several scenarios

# Approach

Based on [Bouyer et al., 2006]:

- Convert MTL specification into an Alternating Timed Automaton
  *intuitively*: tracks unsatisfied parts of the specification
- Regionalize clock values of the TA/ATA configurations
  *intuitively:* allows finite abstraction of infinite state space
- Partition configurations according to fractional parts

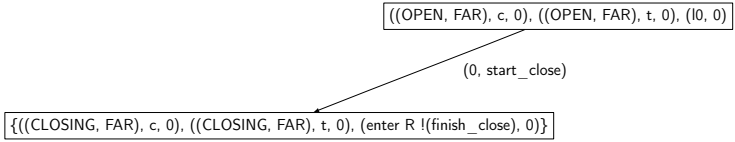
→ Canonical representation of a TA/ATA configuration, e.g.,

```
{((CLOSING, FAR), c, 0)},
{((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
```

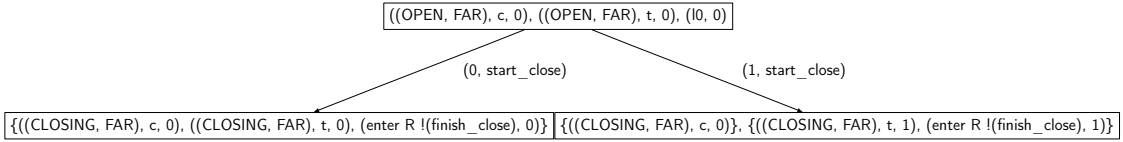
# Approach

((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)

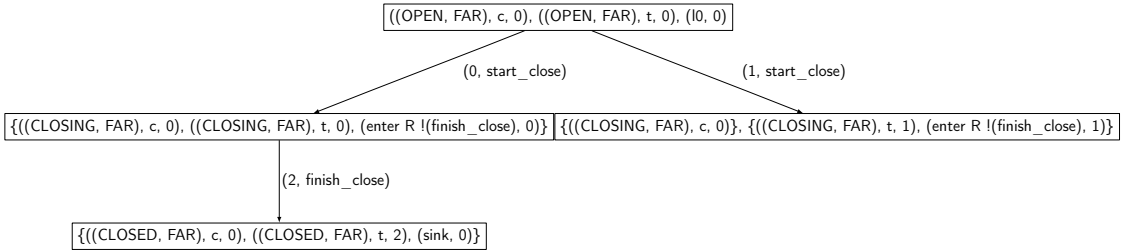
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}

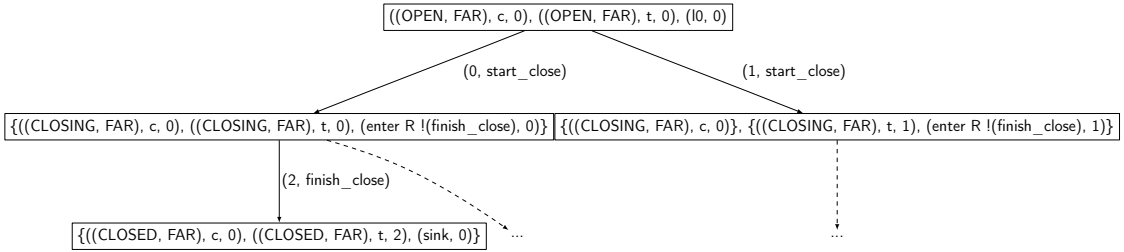
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{{((CLOSING, FAR), c, 0)}, {{((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}

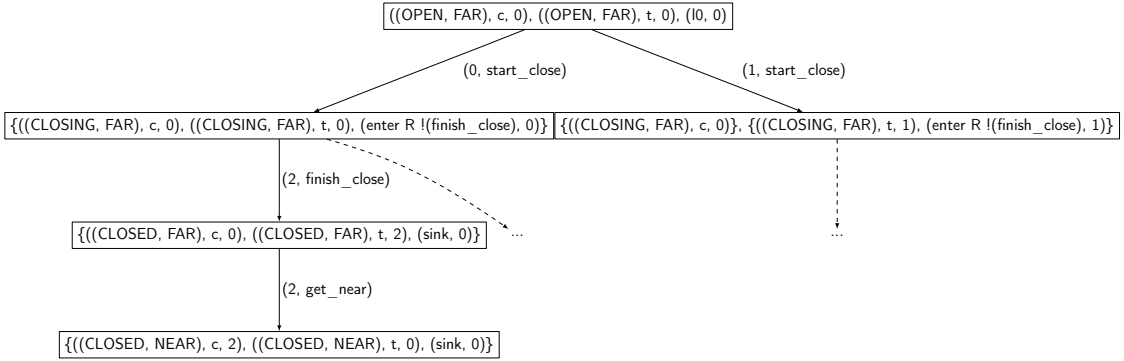
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}

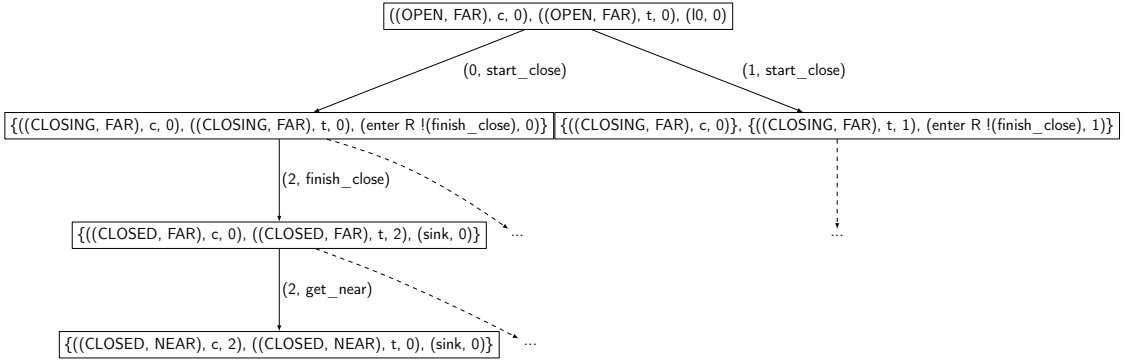
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
...
...

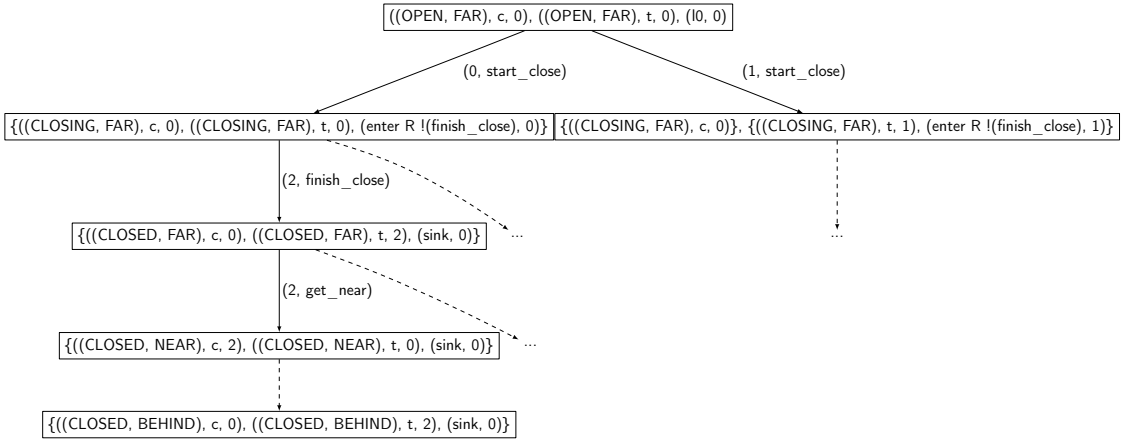
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}

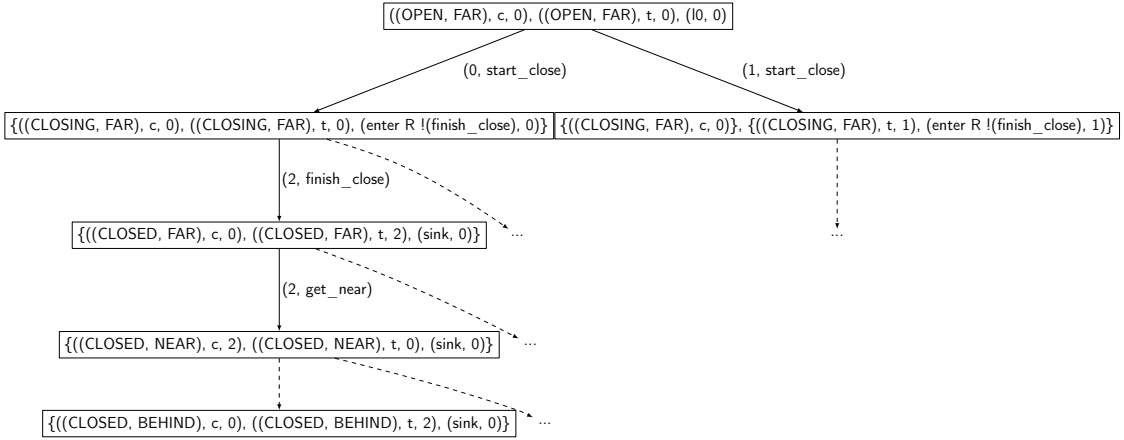
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}

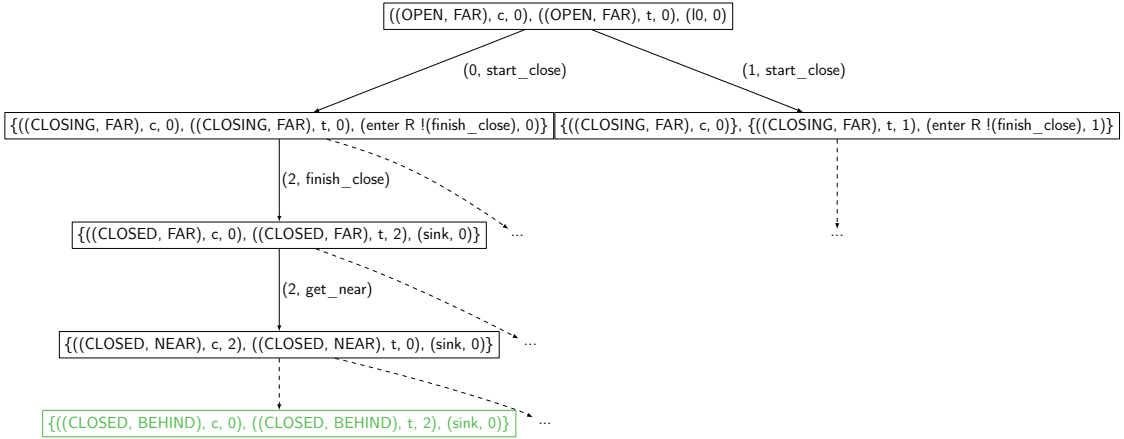
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

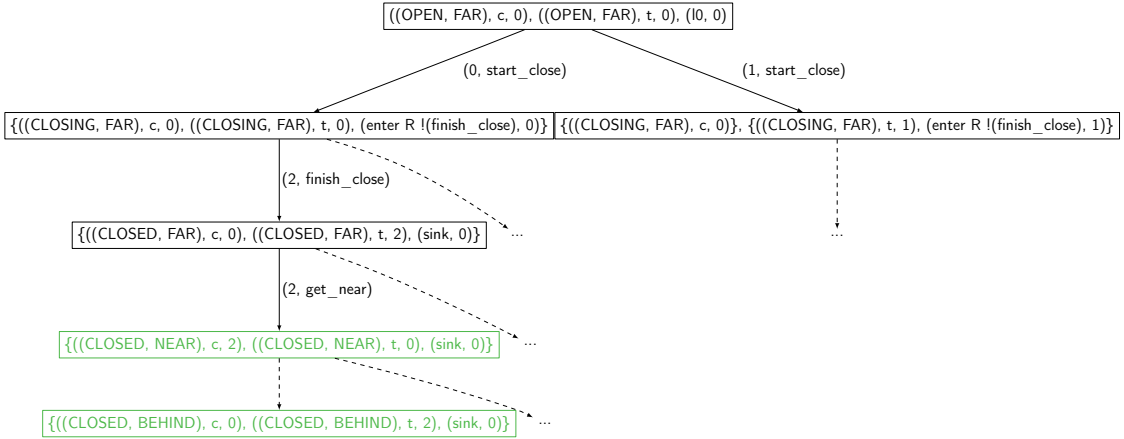
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

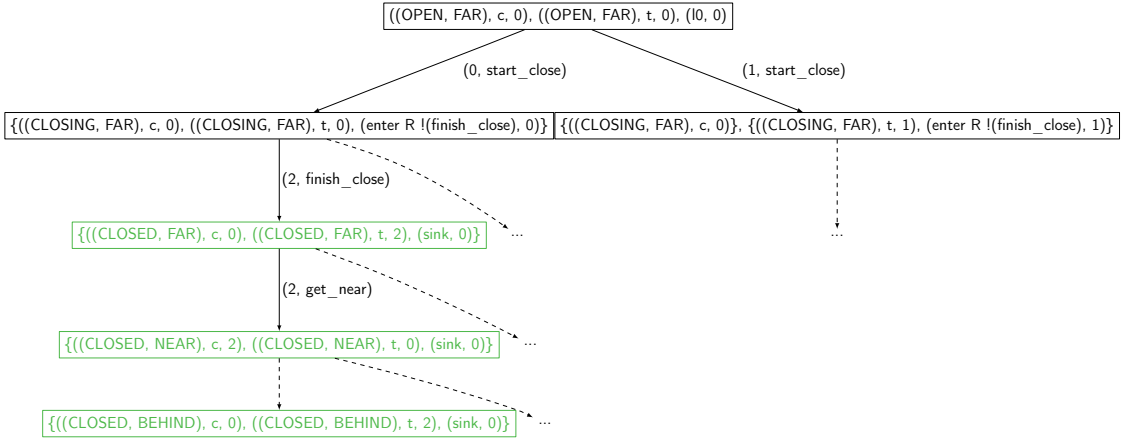
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

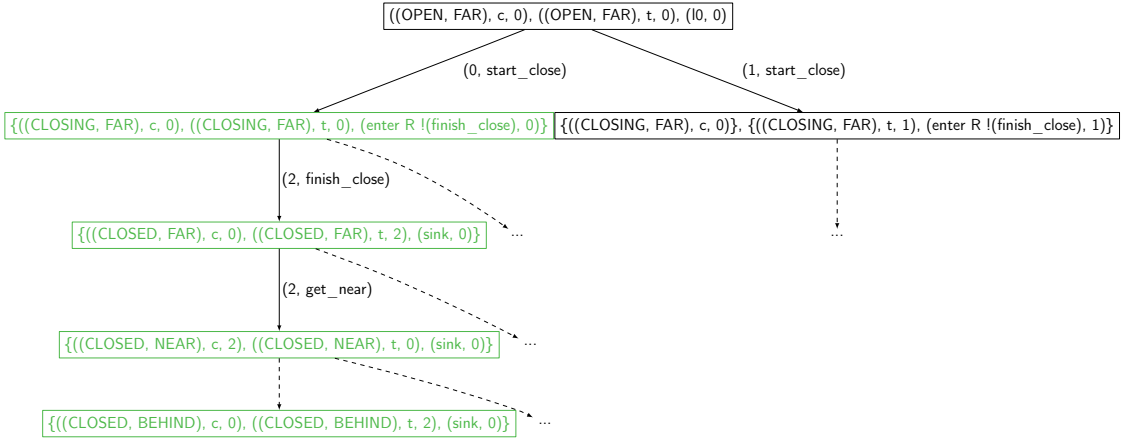
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

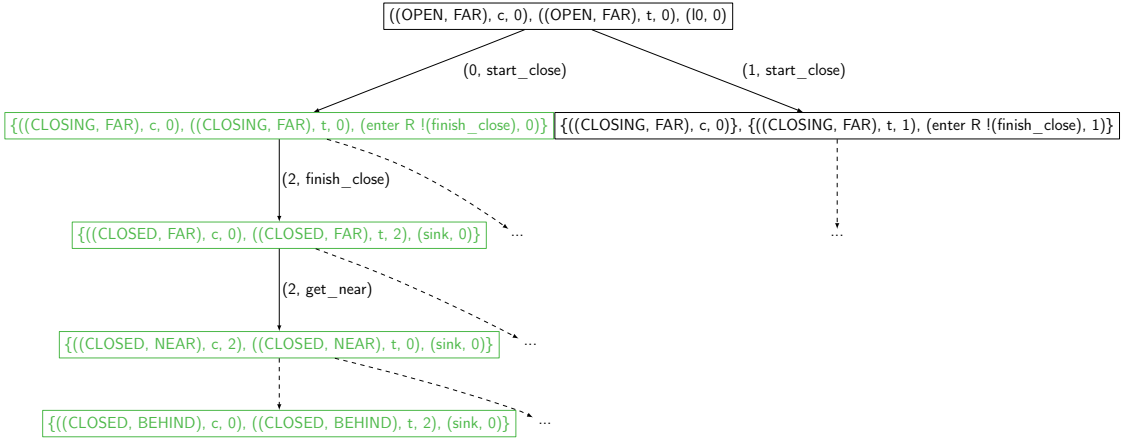
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

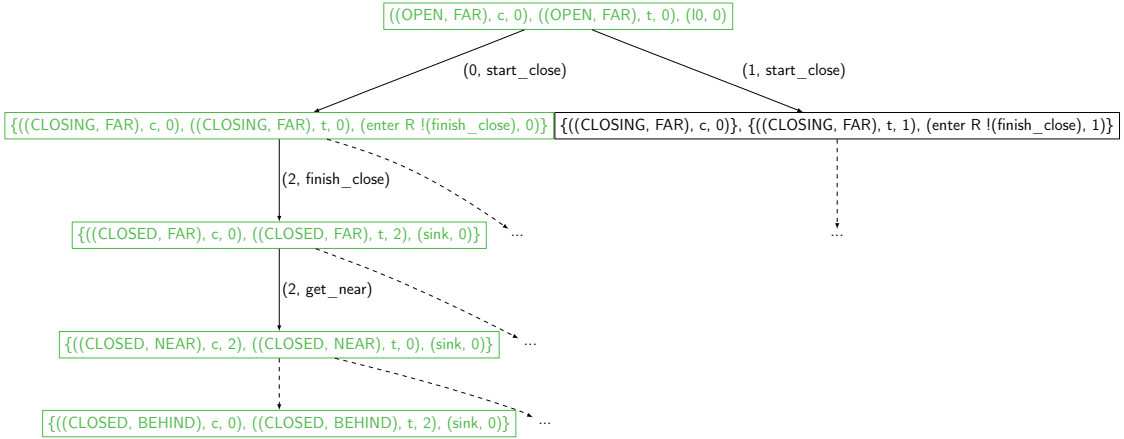
# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

# Approach


((OPEN, FAR), c, 0), ((OPEN, FAR), t, 0), (l0, 0)
(0, start_close)
(1, start_close)
{((CLOSING, FAR), c, 0), ((CLOSING, FAR), t, 0), (enter R !(finish_close), 0)}
{((CLOSING, FAR), c, 0)}, {((CLOSING, FAR), t, 1), (enter R !(finish_close), 1)}
(2, finish_close)
{((CLOSED, FAR), c, 0), ((CLOSED, FAR), t, 2), (sink, 0)}
(2, get_near)
{((CLOSED, NEAR), c, 2), ((CLOSED, NEAR), t, 0), (sink, 0)}
{((CLOSED, BEHIND), c, 0), ((CLOSED, BEHIND), t, 2), (sink, 0)}

# Features

Observation: The size of the search tree grows quickly.

Counter measures:

- Multithreading
  Idea: traverse the search tree in parallel
- Node reusing
  Idea: same configurations are reachable via different paths
- Incremental Labeling and Pruning
  Idea: a single node's label may determine the label of its parent
- Search heuristics
  Idea: find the pruning node fast

# Features

Counter measures:

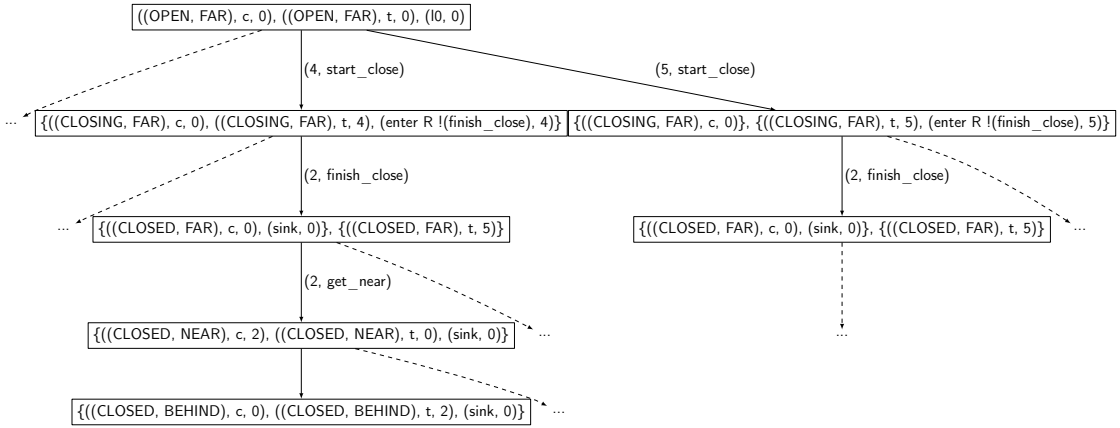
- Multithreading
  Idea: traverse the search tree in parallel

# Features

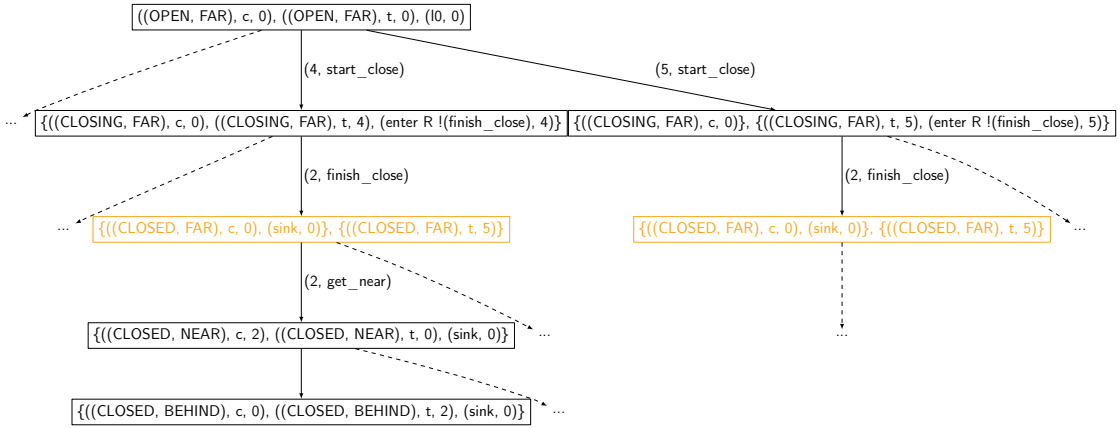
Counter measures:

- Node reusing
  Idea: same configurations are reachable via different paths

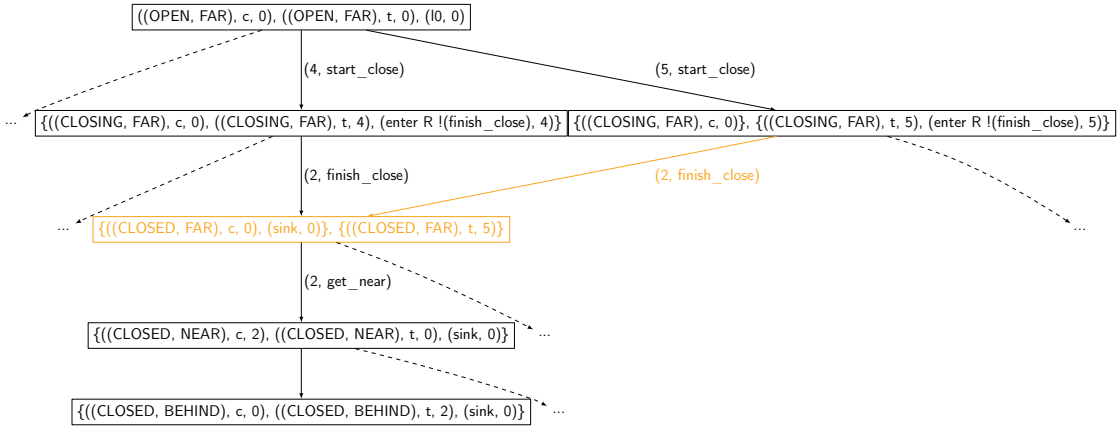
# Node Reusing



# Node Reusing



# Node Reusing

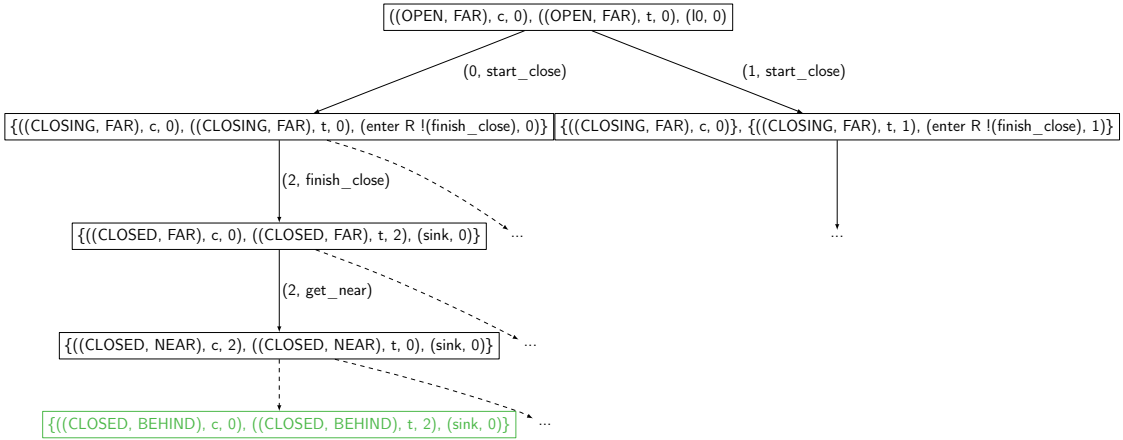


# Features

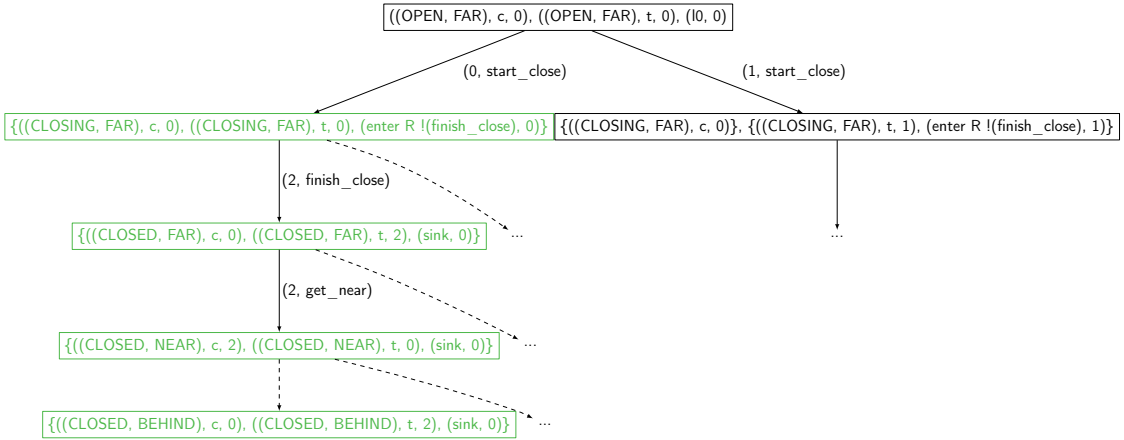
Counter measures:

- Incremental Labeling and Pruning
  Idea: a single node's label may determine the label of its parent

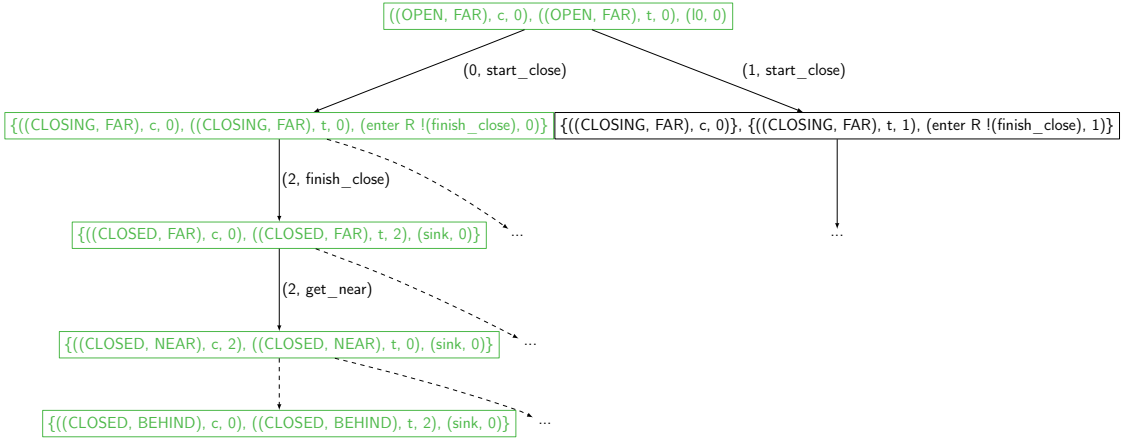
# Incremental Labeling and Pruning



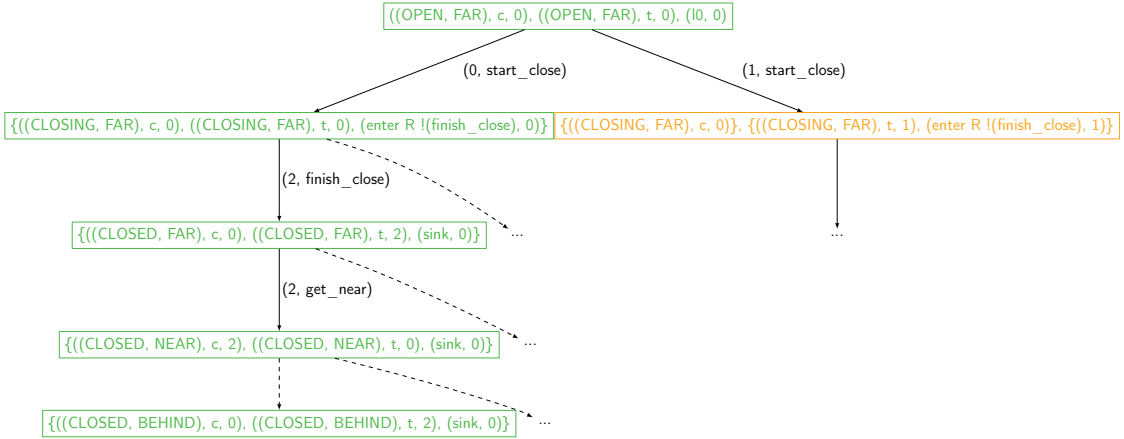
# Incremental Labeling and Pruning



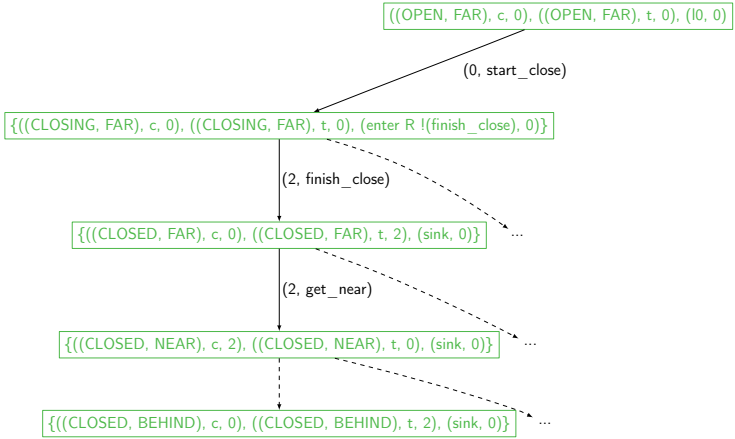
# Incremental Labeling and Pruning



# Incremental Labeling and Pruning



# Incremental Labeling and Pruning



# Features

Counter measures:

- Search heuristics
  Idea: find the pruning node fast

# Heuristics

Search heuristics:

- **bfs** breadth-first search
- **dfs** depth-first search
- **cw** # of canonical words
  → *prefer states with many unsatisfied parts of the spec*
- **time** # of region increments to reach the node
  → *prefer early actions*
- **env** prefer environment actions → *lazy controller*
- **random** assign a random value to the node → useful for comparison
- **comp** weighted portfolio: $16 \times \texttt{cw} + 4 \times \texttt{env} + 1 \times \texttt{time}$

# Experimental results

3 case studies

- Railroad-gate system (as in [Alur et al., 1993]), scalable (# gates)
- Transport robot with a camera (as in [Viehmann et al., 2021])
- Airport conveyor belt (as in [van Hulst et al., 2017])

# Experimental results

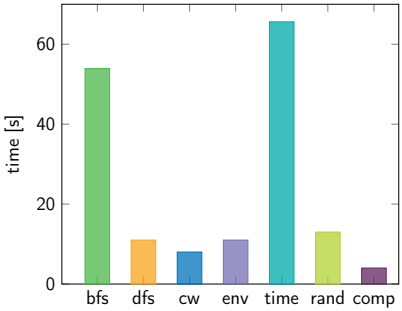
3 case studies

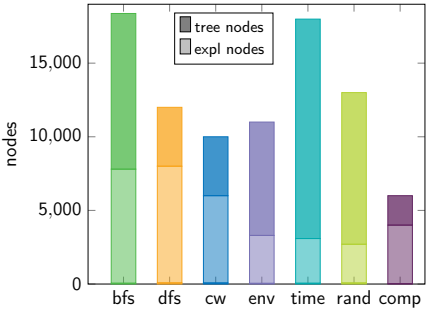
- Railroad-gate system (as in [Alur et al., 1993]), scalable (# gates)
- Transport robot with a camera (as in [Viehmann et al., 2021])
- Airport conveyor belt (as in [van Hulst et al., 2017])

Evaluate

- Effect of search heuristics
- Effect of multithreading
- Scalability [see paper]

# Evaluation: Heuristics

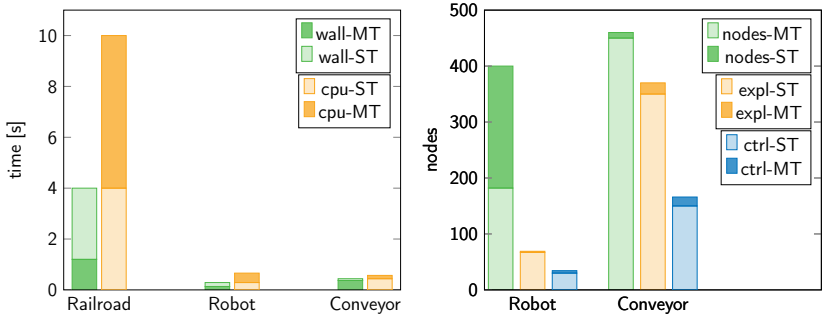



## Observations

- Heuristics reduce the number of nodes and total search time
- Computational effort may pay off
- Weighted "portfolio" `comp` heuristics performs best

# Evaluation: Multithreading




## Observations

- General speed-up, higher CPU-usage
- More nodes discovered and explored
- Larger controller size

# Conclusion & Future work

We have presented TACoS:

- Controller synthesis for timed automata and MTL specifications
- Several improvements to increase performance and scalability
- Free and open-source tool
- Available at `https://github.com/morxa/tacos/`

## Outlook

- Better heuristics
- Synthesis for programs, e.g., on robots

# References I


Alur, R., Henzinger, T., and Vardi, M. (1993).
Parametric real-time reasoning.
In *STOC*.

Bouyer, P., Bozzelli, L., and Chevalier, F. (2006).
Controller Synthesis for MTL Specifications.
In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR)*, pages 450–464. Springer Berlin Heidelberg.

van Hulst, A., Reniers, M., and Fokkink, W. (2017).
Maximally permissive controlled system synthesis for non-determinism and modal logic.
*DEDS*, 27(1).

Viehmann, T., Hofmann, T., and Lakemeyer, G. (2021).
Transforming robotic plans with timed automata to solve temporal platform constraints.
In *IJCAI*.

# Heuristics Comparison

Scenario: railroad, 2 gates, plant size: 144

| heu | wall (s) | CPU (s) | nodes (k) | expl (k) | ctrl |
|---|---|---|---|---|---|
| bfs | 5.39(9) | 5.38(9) | 1.832(2) | 0.78(2) | 53(7) |
| dfs | 1.1(4) | 1.1(4) | 1.2(2) | 0.8(1) | 79(29) |
| cw | 0.8(3) | 0.8(3) | 1.0(2) | 0.60(9) | 71(8) |
| env | 1.1(3) | 1.1(3) | 1.1(2) | 0.33(8) | 46(3) |
| time | 6.56(9) | 6.55(9) | 1.799(8) | 0.309(7) | 52(10) |
| rand | 1.3(5) | 1.3(5) | 1.3(2) | 0.27(6) | 71(20) |
| comp | 0.4(3) | 0.4(3) | 0.6(3) | 0.4(2) | 32(10) |

# Heuristics Comparison

Scenario: railroad, 2 gates, plant size: 144

| heu | wall (s) | CPU (s) | nodes (k) | expl (k) | ctrl |
|---|---|---|---|---|---|
| bfs | 5.39(9) | 5.38(9) | 1.832(2) | 0.78(2) | 53(7) |
| dfs | 1.1(4) | 1.1(4) | 1.2(2) | 0.8(1) | 79(29) |
| cw | 0.8(3) | 0.8(3) | 1.0(2) | 0.60(9) | 71(8) |
| env | 1.1(3) | 1.1(3) | 1.1(2) | 0.33(8) | 46(3) |
| time | 6.56(9) | 6.55(9) | 1.799(8) | 0.309(7) | 52(10) |
| rand | 1.3(5) | 1.3(5) | 1.3(2) | 0.27(6) | 71(20) |
| comp | 0.4(3) | 0.4(3) | 0.6(3) | 0.4(2) | 32(10) |

# Heuristics Comparison

Scenario: railroad, 2 gates, plant size: 144

| heu | wall (s) | CPU (s) | nodes (k) | expl (k) | ctrl |
|---|---|---|---|---|---|
| bfs | 5.39(9) | 5.38(9) | 1.832(2) | 0.78(2) | 53(7) |
| dfs | 1.1(4) | 1.1(4) | 1.2(2) | 0.8(1) | 79(29) |
| cw | 0.8(3) | 0.8(3) | 1.0(2) | 0.60(9) | 71(8) |
| env | 1.1(3) | 1.1(3) | 1.1(2) | 0.33(8) | 46(3) |
| time | 6.56(9) | 6.55(9) | 1.799(8) | 0.309(7) | 52(10) |
| rand | 1.3(5) | 1.3(5) | 1.3(2) | 0.27(6) | 71(20) |
| comp | 0.4(3) | 0.4(3) | 0.6(3) | 0.4(2) | 32(10) |

# Heuristics Comparison

Scenario: railroad, 2 gates, plant size: 144

| heu | wall (s) | CPU (s) | nodes (k) | expl (k) | ctrl |
|---|---|---|---|---|---|
| bfs | 5.39(9) | 5.38(9) | 1.832(2) | 0.78(2) | 53(7) |
| dfs | 1.1(4) | 1.1(4) | 1.2(2) | 0.8(1) | 79(29) |
| cw | 0.8(3) | 0.8(3) | 1.0(2) | 0.60(9) | 71(8) |
| env | 1.1(3) | 1.1(3) | 1.1(2) | 0.33(8) | 46(3) |
| time | 6.56(9) | 6.55(9) | 1.799(8) | 0.309(7) | 52(10) |
| rand | 1.3(5) | 1.3(5) | 1.3(2) | 0.27(6) | 71(20) |
| comp | 0.4(3) | 0.4(3) | 0.6(3) | 0.4(2) | 32(10) |

# Backup - Scalability Results

## Scalability

- Heuristic: compositional
- Model: Railroad (RR)

| Scenario | size | wall (s) | CPU (s) | nodes (k) | expl (k) | ctrl |
|---|---|---|---|---|---|---|
| RR(2,2) | 144 | 0.12(1) | 1.0(1) | 0.93(7) | 0.58(5) | 43(5) |
| RR(2,4) | 144 | 0.42(4) | 4.1(8) | 2.26(7) | 1.4(2) | 49(9) |
| RR(2,8) | 144 | 2.0(7) | 21(10) | 5.8(7) | 3.1(8) | 47(10) |
| RR(4,4) | 144 | 1.14(7) | 15(1) | 3.24(1) | 2.238(8) | 48(2) |
| RR(4,8) | 144 | 6(1) | 91(19) | 8.3(6) | 5.1(5) | 64(19) |
| RR(8,8) | 144 | 28(9) | 431(151) | 11.1(3) | 7.5(1) | 45(10) |
| RR(1,1,1) | 832 | 4(1) | 38(14) | 13(4) | 6(1) | 74(82) |
| RR(2,1,1) | 832 | 1877(287) | 29 858(4582) | 45(3) | 33(2) | 101(31) |
| RR(2,2,2) | 832 | 3654(1243) | 58 228(19820) | 49(13) | 37(7) | 103(45) |