SINTEF

Improving Online Railway Deadlock Detection using a Partial Order Reduction

Bjørnar Luteberget

FMAS 2021, October 22, 2021

Acknowledgments

Project context: **GoTo – Greater Oslo Area Train Optimization**, with Norwegian railway infrastructure manager Bane NOR.

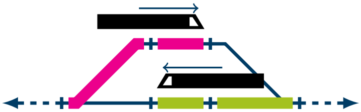
Thanks to:

- Koen Claessen, Chalmers Univ., Gothenburg
- Christian Johansen, NTNU, Gjøvik
- Martin Steffen, Univ. Oslo
- Veronica Dal Sasso, Optrail, Rome
- Carlo Mannino, SINTEF Digital, Oslo

Automation and autonomy in railway

- Tight schedules are often disrupted by unforseen events.
- Manual dispatching: operators try to re-schedule.
- Autonomous dispatching: automatically compute optimal schedules.
  - ✓ Sensors are connected to online computer system.
  - ✓ Optimization tools can compute good or optimal schedules in real time.
  - ✗ ... for large infrastructures
  - ✗ ... with direct control of trains

Schedules and deadlocks

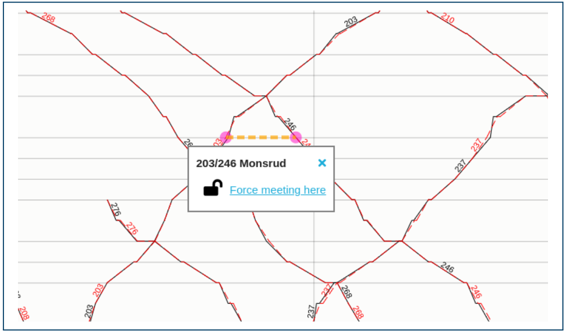


- Even finding a feasible schedule in a real-time dispatching situation is NP-hard[1].
- Situations change in minutes.
- Large scale re-scheduling systems are likely to use heuristics and/or limited scheduling horizons.
- no feasible schedule $\Leftrightarrow$ bound for deadlock
- Check for deadlocks as a separate procedure?
- Found to be hard in a 2021 study[2].

[2] Lu, Dessouky, and Leachman, “Modeling train movements through complex rail networks”.
[2] Sasso et al., “The Tick Formulation for deadlock detection and avoidance in railways traffic control”.

SINTEF

Manual overrides



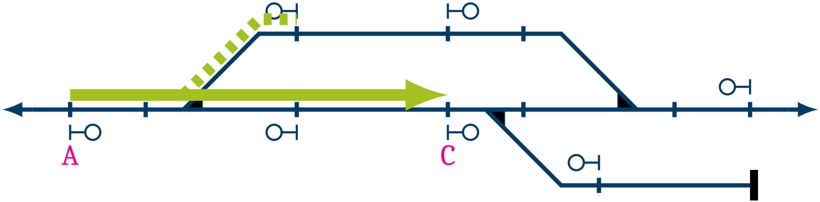
- Even with a working autonomous re-scheduling and dispatching, operators might want to override decisions.
- This may cause deadlocks.

Deadlocks in manual dispatching

Photo by Nate Beal (CC-BY-2.0)

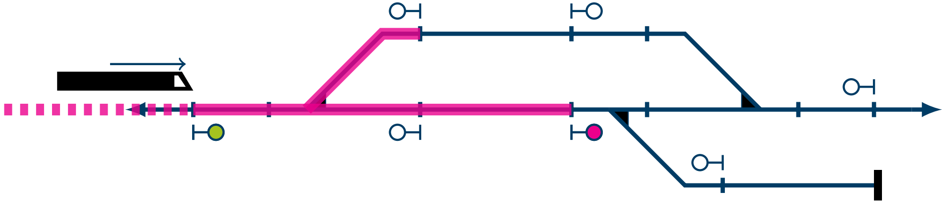
- With manual dispatching of long trains, actual deadlocks happen in practice.
- These require costly recovery operations.
- Beneficial to know about deadlocks as early as possible.

Train movements

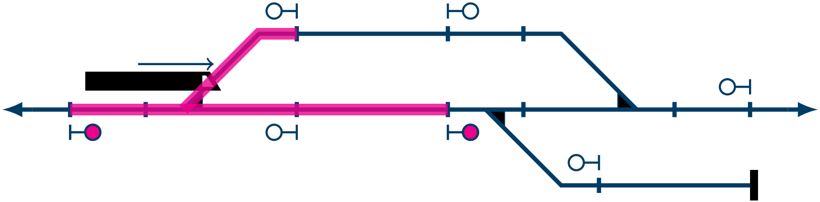


SINTEF

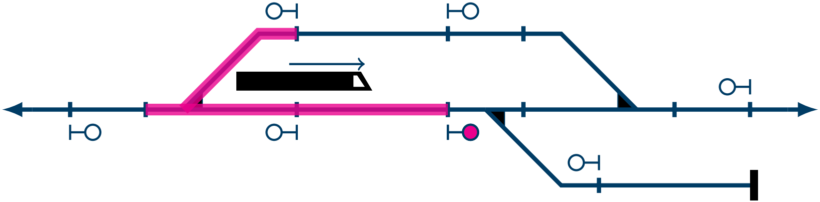
Train movements



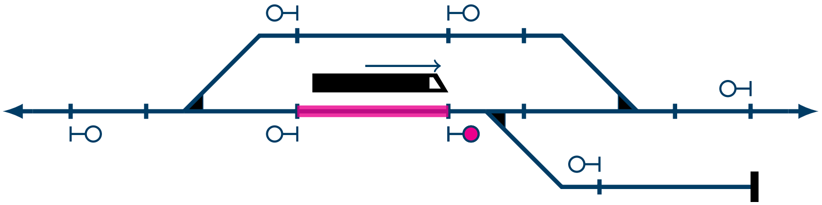
Train movements



Train movements



Train movements



Approach to the deadlock detection problem

The route system already supplies a discretization for us!

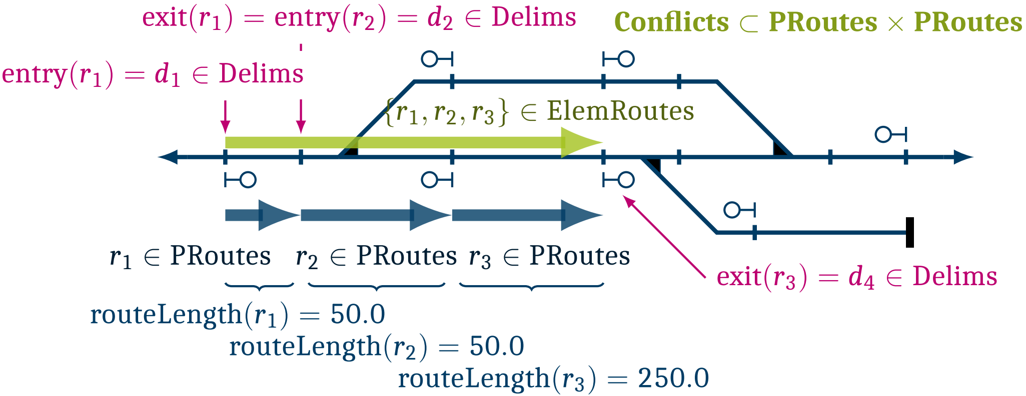
For deadlock detection, we can ignore:

- velocities and exact locations
- acceleration/braking power
- sight distances

Solution idea:

1. Define problem using route infrastructure model.
2. Model as a (discrete) transition system.
3. Solve with a SAT solver.

Infrastructure model I



Trains model T

In addition to infrastructure, we need the following train data:

- A set of trains, Trains.
- The length of each train, $\text{trainLength} : \text{Trains} \to \mathbb{R}$.
- Each train's initial position, $\text{initialRoutes} : \text{Trains} \to 2^{\text{PRoutes}}$.
- Each train's final position alternatives, $\text{finalRoutes} : \text{Trains} \to 2^{\text{PRoutes}}$.

Online railway deadlock detection problem

Definition

- The online railway deadlock detection problem $D = (I, T)$ ...
- ... is solved by a deadlock detection algorithm $d : D \rightarrow \{\text{Live}, \text{Dead}\}$, which returns Live if all trains can travel to one of their final positions, and Dead otherwise.

SINTEF

Transition system model[3]

- Propositional logic: k-step unrolling of transition relation:

$$\Phi_k = \bigwedge_{i=0}^{k} \phi_i$$

- Variables for state i:
  - For each partial route r: One-hot encoding of

$$o_r^i \in \text{Trains} \cup \{\text{Free}\}$$

  - Does the train reach its destination in or before state i?

$$f_t^i$$

[3] Based on Luteberget et al., "SAT modulo discrete event simulation applied to railway design capacity analysis".

SINTEF

Specifying the transition relation

- First attempts at planning with SAT used classical frame axioms[4]: one action in each step and consecutive states equal except for action effects.

$$\text{atMostOne}(\{\alpha\}), \qquad \alpha \Rightarrow (v^{i-1} \Rightarrow v^i), \qquad \text{for all } v \text{ not in effects}(\alpha)$$

- Usually, better encodings from explanatory frame axioms[5]: a changed value must be explained as an action effect. Thinking backwards!

$$(\neg v^{i-1} \wedge v^i) \Rightarrow \bigvee \alpha, \qquad \text{(any } \alpha \text{ with effect v)}$$

- Non-interfering actions can happen in the same step!

[4] Kautz and Selman, "Planning as Satisfiability".
[5] Kautz, McAllester, and Selman, "Encoding Plans in Propositional Logic".

Constraints (1)

- Mutual exclusion between conflicting routes:

$$\bigwedge_{(a,b)\in \text{Conflicts}} \left((o_a^i = \text{Free}) \vee (o_b^i = \text{Free}) \right)$$

SINTEF

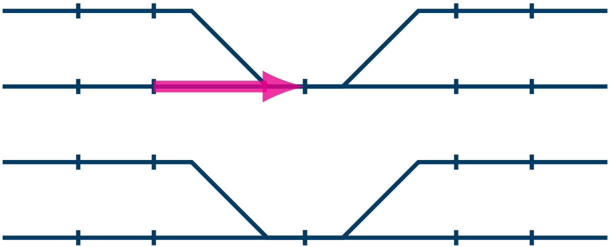
Constraints (2)

Path consistency:

$$\left(o_r^{i-1} \neq t \wedge o_r^{i} = t\right) \Rightarrow \ldots$$

State $i-1$

State i



SINTEF

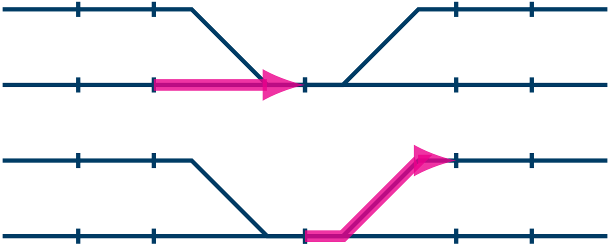
Constraints (2)

Path consistency:

$$\left(o_r^{i-1} \neq t \wedge o_r^{i} = t\right) \Rightarrow \bigvee_{\substack{x \in \text{PRoutes} \\ \text{entry}(r)=\text{exit}(x)}} o_x^{i-1} = t$$

State $i-1$

State i

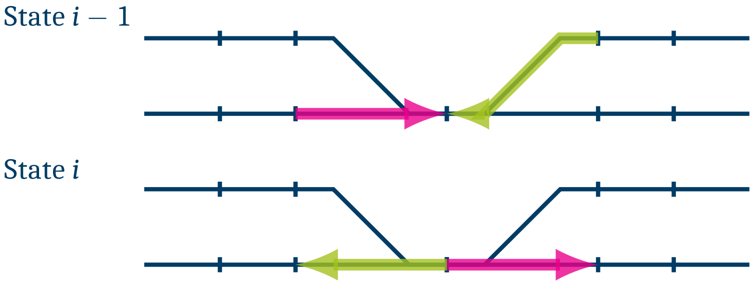


SINTEF

Constraints (2)

Path consistency:

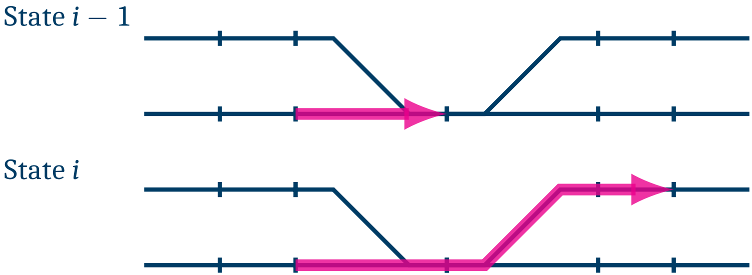
$$\left(o_r^{i-1} \neq t \wedge o_r^{i} = t\right) \Rightarrow \bigvee_{\substack{x \in \text{PRoutes} \\ \text{entry}(r) = \text{exit}(x)}} \left(o_x^{i-1} = t \wedge o_x^{i} = t\right)$$



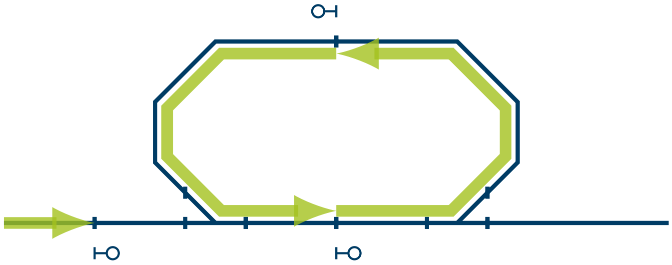
Constraints (2)

Path consistency:

$$\left(o_r^{i-1} \neq t \wedge o_r^i = t\right) \Rightarrow \bigvee_{\substack{x \in \text{PRoutes} \\ \text{entry}(r) = \text{exit}(x)}} o_x^i = t$$



Path consistency in cyclic infrastructure



- Can use cycle elimination constraints[6].
- This a general issue with logical encoding of problems involving graphs[7].

[6] Luteberget et al., "SAT modulo discrete event simulation applied to railway design capacity analysis".
[7] Gebser, Janhunen, and Rintanen, "SAT Modulo Graphs: Acyclicity".

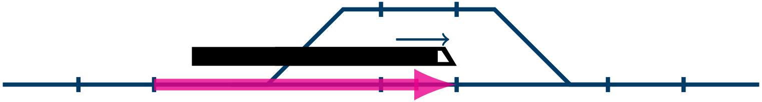
SINTEF

Constraints (3)

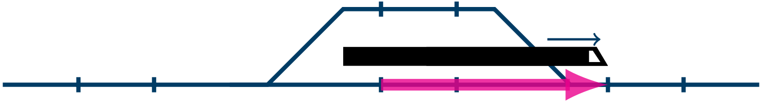
Train persistence:

$$\left(o_r^{i-1} = t \land o_r^i \neq t\right) \Rightarrow \ldots$$

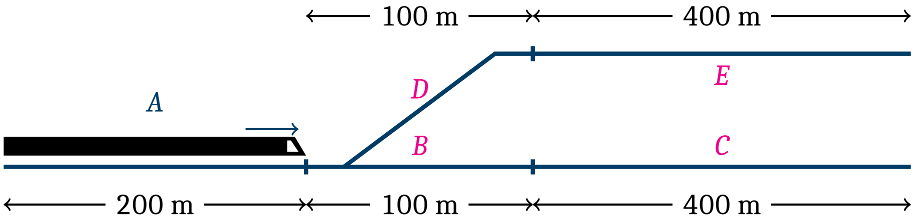
State $i-1$



State i



Freeing combinations



If A holds a train t of length 200.0 m, freeing A is constrained by:

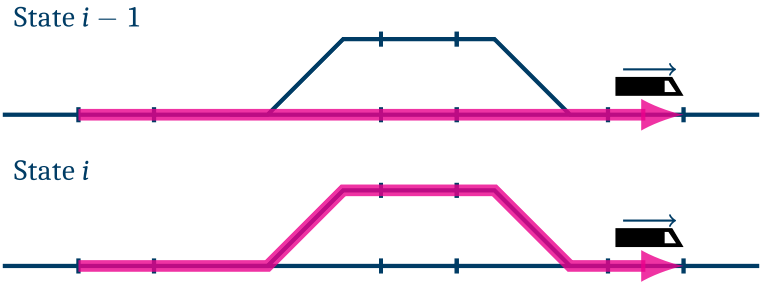
$$A^{i-1} \Rightarrow \left(A^i \vee (B^i \wedge C^i) \vee (D^i \wedge E^i)\right).$$

SINTEF

Constraints (3)

Train persistence:

$$\left(o_r^{i-1} = t \wedge o_r^i \neq t\right) \Rightarrow \text{freeable}_t^i(r, \text{trainLength}(t))$$



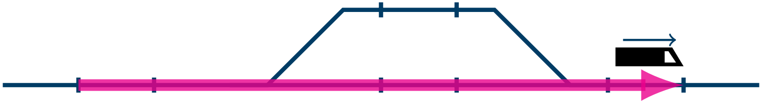
SINTEF

Constraints (3)

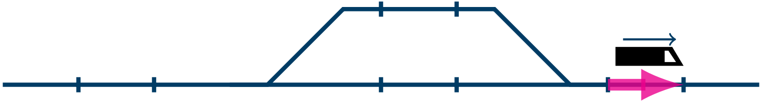
Train persistence:

$$(o_r^{i-1} = t) \Rightarrow (o_r^i \neq t \Leftrightarrow \text{freeable}_t^i(r, \text{trainLength}(t)))$$

State $i - 1$



State i



SINTEF

The transition system

Putting the constraints together, ϕ_i is the conjunction of:

- **Mutual exclusion:** $\bigwedge_{(a,b)\in\text{Conflicts}} \left((o_a^i = \text{Free}) \vee (o_b^i = \text{Free})\right)$
- **Path consistency:** $\left(o_r^{i-1} \neq t \wedge o_r^i = t\right) \Rightarrow \bigvee_{\substack{x\in\text{PRoutes} \\ \text{entry}(r)=\text{exit}(x)}} o_x^i = t$
- **Train persistence:** $(o_r^{i-1} = t) \Rightarrow (o_r^i \neq t \Leftrightarrow \text{freeable}_t^i(r, \text{trainLength}(t)))$
- **Elementary routes:** $\bigwedge_{e\in\text{ElemRoutes}} \bigwedge_{r\in e} \left(\left(o_r^{i-1} \neq t \wedge o_r^i = t\right) \Rightarrow \bigwedge_{r\in e}(o_r^i = t)\right)$

We also have a known **initial state** ϕ_0, and a **goal condition** $G_i = \bigwedge_{t\in\text{Trains}} f_t^i$, where:

$$(\neg f_t^{i-1} \wedge f_t^i) \Rightarrow \bigvee_{r\in\text{finalRoutes}(t)} o_r^i = t$$

SINTEF

Complete bounded model checking

- Bounded model checking[8] is not complete unless the number of transitions exceeds the completeness threshold.
- Completeness threshold for acyclic route-based railway model[9]: longest possible path, summed over trains.

Algorithm 1: Deadlock detection using incremental k-bounded model checking

---

Input : A problem instance $D = (I, T)$ and a bound k.

Output: Dead if the system is bound for deadlock, Live otherwise.

1 **let** $i = 1$.

2 **if** $\Phi_i \land G_i$ is Sat, **return** Live

3 **if** $i < k$, increment i and go to 2, **else return** Dead

---

[8] Biere et al., "Bounded model checking".

[9] Sasso et al., "The Tick Formulation for deadlock detection and avoidance in railways traffic control".

Instance				Ticks MIP alg. (reported in [21])		Algorithm 1	
#	Result	n_r	n_t	Steps	Time (s)	Steps	Time (s)
01	LIVE	14	3	8	1.08	5	0.00
02	DEAD	14	3	8	0.98	10	0.00
03	LIVE	14	3	8	0.93	5	0.00
04	LIVE	30	2	15	1.20	4	0.00
05	LIVE	30	3	20	1.31	5	0.00
06	DEAD	30	3	20	2.78	19	0.03
07	DEAD	38	5	34	37.31	34	0.17
08	LIVE	46	5	33	1.78	5	0.00
09	DEAD	38	6	37	>60.00	37	0.26
10	DEAD	38	7	42	4.23	42	0.02
11	DEAD	62	2	27	17.60	26	3.30
12	DEAD	62	4	39	>60.00	40	1.70
13	DEAD	62	4	39	>60.00	40	1.30
14	LIVE	62	4	39	3.27	6	0.00
15	DEAD	46	4	42	>60.00	42	0.22
16	LIVE	62	5	50	5.33	5	0.00
17	LIVE	62	4	50	43.11	6	0.00
18	DEAD	62	4	50	>60.00	49	2.10
19	DEAD	62	5	51	>60.00	50	0.83
20	DEAD	70	5	57	>60.00	56	1.20

Zig-zag algorithm[10]

Idea: at least one route must be allocated in each transition:

$$z_i = \bigvee_{t \in \text{Trains}} \bigvee_{r \in \text{PRoutes}} ((o_r^{i-1} \neq t) \wedge (o_r^i = t)), \qquad Z_i = \bigwedge_{j=1}^{i} z_j$$

Algorithm 2: Online railway deadlock detection with global progress constraint

---

Input : A problem instance $D = (I, T)$.

Output: Dead if the system is bound for deadlock, Live otherwise.

1 **let** $i = 1$.

2 **if** $\Phi_i \wedge Z_i$ is Unsat, **return** Dead

3 **if** $\Phi_i \wedge Z_i \wedge G_i$ is Sat, **return** Live

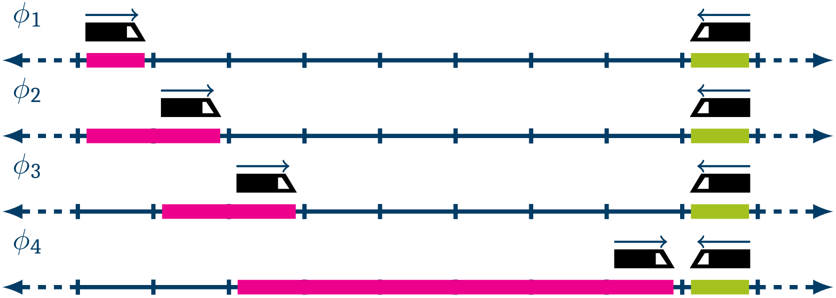
4 increment i and go to 2.

---

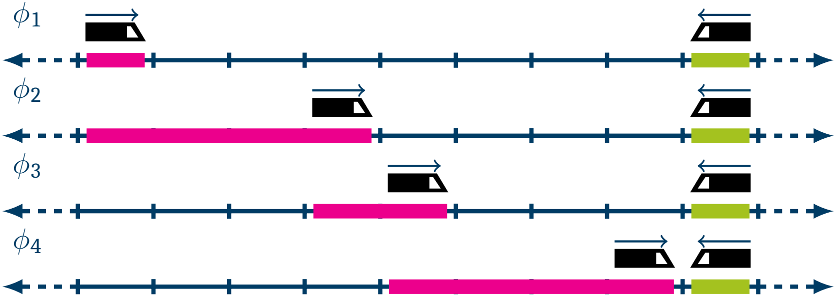
[10] Eén and Sörensson, "Temporal induction by incremental SAT solving".

Instance				Ticks MIP alg. (reported in [21])		Algorithm 1		Algorithm 2	
#	Result	n_r	n_t	Steps	Time (s)	Steps	Time (s)	Steps	Time (s)
01	LIVE	14	3	8	1.08	5	0.00	5	0.00
02	DEAD	14	3	8	0.98	10	0.00	7	0.00
03	LIVE	14	3	8	0.93	5	0.00	5	0.00
04	LIVE	30	2	15	1.20	4	0.00	4	0.00
05	LIVE	30	3	20	1.31	5	0.00	5	0.00
06	DEAD	30	3	20	2.78	19	0.03	9	0.04
07	DEAD	38	5	34	37.31	34	0.17	7	0.00
08	LIVE	46	5	33	1.78	5	0.00	5	0.00
09	DEAD	38	6	37	>60.00	37	0.26	14	0.25
10	DEAD	38	7	42	4.23	42	0.02	2	0.00
11	DEAD	62	2	27	17.60	26	3.30	15	3.30
12	DEAD	62	4	39	>60.00	40	1.70	20	9.60
13	DEAD	62	4	39	>60.00	40	1.30	20	11.00
14	LIVE	62	4	39	3.27	6	0.00	6	0.01
15	DEAD	46	4	42	>60.00	42	0.22	15	1.30
16	LIVE	62	5	50	5.33	5	0.00	5	0.00
17	LIVE	62	4	50	43.11	6	0.00	6	0.01
18	DEAD	62	4	50	>60.00	49	2.10	15	13.10
19	DEAD	62	5	51	>60.00	50	0.83	16	36.00
20	DEAD	70	5	57	>60.00	56	1.20	-	>60.00

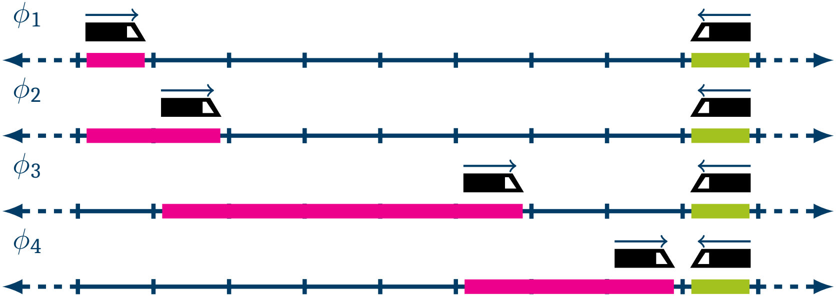
Symmetries



Symmetries



Symmetries



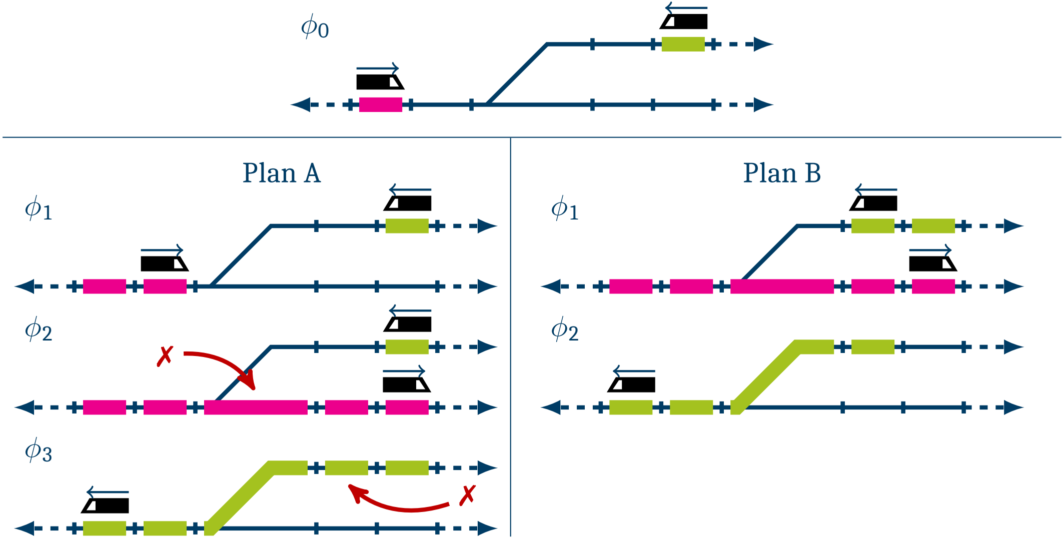
SINTEF

Maximal progress

- For any allocation that is extending the path from the previous state,
- ... a route conflicting with r must be occupied in the previous state.
- Same idea as process semantics for planning as SAT[11]: all actions happen as early as possible.

$$\left(o_r^{i-1} \neq t \wedge o_r^{i} = t \wedge \bigvee_{\substack{x \in \text{PRoutes} \\ \text{entry}(r)=\text{exit}(x)}} (o_x^{i-1} = t) \right) \Rightarrow \bigvee_{\substack{(r,y) \in \text{Conflicts} \\ \text{...or } y=r}} (o_y^{i-1} \neq t \wedge o_y^{i-1} \neq \text{Free})$$

[11] Rintanen, Heljanko, and Niemelä, "Planning as satisfiability: parallel plans and algorithms for plan search".



ϕ_0
Plan A
ϕ_1
ϕ_2
ϕ_3
Plan B
ϕ_1
ϕ_2

Maximal progress

We call this constraint a partial order reduction because it creates a unique representation of all solutions that represent the same partial order containing the trains' paths and the order in which they use conflicting routes.

Algorithm 3: Online railway deadlock detection with partial order reduction

---

Input : A problem instance $D = (I, T)$.
Output: Dead if the system is bound for deadlock, Live otherwise.

1 **let** $i = 1$.
2 **if** $\Phi_i \land Z_i \land P_i$ is Unsat, **return** Dead
3 **if** $\Phi_i \land Z_i \land P_i \land G_i$ is Sat, **return** Live
4 increment i and go to 2.

Instance				Ticks MIP alg. (reported in [21])		Algorithm 1		Algorithm 2		Algorithm 3	
#	Result	n_r	n_t	Steps	Time (s)	Steps	Time (s)	Steps	Time (s)	Steps	Time (s)
01	LIVE	14	3	8	1.08	5	0.00	5	0.00	5	0.00
02	DEAD	14	3	8	0.98	10	0.00	7	0.00	5	0.00
03	LIVE	14	3	8	0.93	5	0.00	5	0.00	5	0.00
04	LIVE	30	2	15	1.20	4	0.00	4	0.00	4	0.00
05	LIVE	30	3	20	1.31	5	0.00	5	0.00	5	0.00
06	DEAD	30	3	20	2.78	19	0.03	9	0.04	5	0.00
07	DEAD	38	5	34	37.31	34	0.17	7	0.00	5	0.00
08	LIVE	46	5	33	1.78	5	0.00	5	0.00	5	0.00
09	DEAD	38	6	37	>60.00	37	0.26	14	0.25	7	0.01
10	DEAD	38	7	42	4.23	42	0.02	2	0.00	2	0.00
11	DEAD	62	2	27	17.60	26	3.30	15	3.30	3	0.00
12	DEAD	62	4	39	>60.00	40	1.70	20	9.60	8	0.19
13	DEAD	62	4	39	>60.00	40	1.30	20	11.00	8	0.12
14	LIVE	62	4	39	3.27	6	0.00	6	0.01	6	0.02
15	DEAD	46	4	42	>60.00	42	0.22	15	1.30	6	0.01
16	LIVE	62	5	50	5.33	5	0.00	5	0.00	5	0.01
17	LIVE	62	4	50	43.11	6	0.00	6	0.01	6	0.02
18	DEAD	62	4	50	>60.00	49	2.10	15	13.10	6	0.05
19	DEAD	62	5	51	>60.00	50	0.83	16	36.00	6	0.03
20	DEAD	70	5	57	>60.00	56	1.20	-	>60.00	6	0.03

Conclusion

Adapted a transition system model for railway planning from[12] to the online railway deadlock detection problem.

- Shows improved performance over earlier work[13] because of:
  - Specifying the transition relation with more parallelism, decreasing the minimum number of transitions to find a feasible schedule.
  - Partial order reduction, decreasing the maximum number of transitions to find a bound-for-deadlock situation.
  - Using a SAT solver (instead of MIP)

Future work:

- Abstraction refinement to find deadlocks in larger networks
- Integration with a scheduling algorithm for producing deadlock-free schedules

[12] Luteberget et al., "SAT modulo discrete event simulation applied to railway design capacity analysis".
[13] Sasso et al., "The Tick Formulation for deadlock detection and avoidance in railways traffic control".