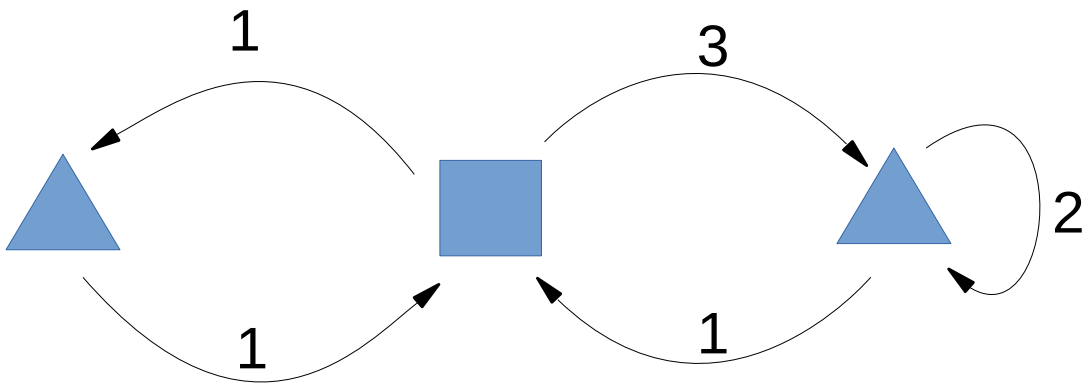
Parity Games:
Another View on Lehtinen's Algorithm

Paweł Parys
University of Warsaw

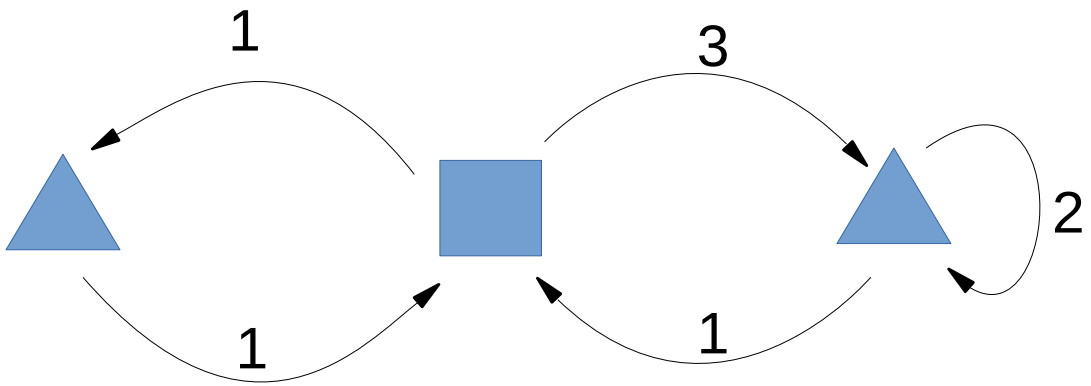
CSL 2020

Parity games



- Priorities on edges
- Player owning the current vertex chooses the next vertex
- Player □ wins if the biggest priority seen infinitely often is even

Parity games



- Priorities on edges
- Player owning the current vertex chooses the next vertex
- Player □ wins if the biggest priority seen infinitely often is even

Long standing open problem:

Decide in PTIME which player has a winning strategy.

Recent results

Long standing open problem:
Decide in PTIME which player has a winning strategy.

Recent result:
This can be decided in quasi-polynomial time, i.e. $n^{O(\log n)}$

A few algorithms achieving this:

- play summaries - Calude, Jain, Khoussainov, Li, Stephan 2017
- antagonistic play summaries - Fearnley, Jain, Schewe, Stephan, Wojtczak 2017
- succinct progress measures - Jurdziński, Lazić 2018
- register games - Lehtinen 2018
- recursive à la Zielonka - Parys 2019
- improved recursive à la Zielonka - Lehtinen, Schewe, Wojtczak 2019

This paper:

1) We explain the Lehtinen's algorithm in the separation approach
2) Small improvement in the complexity analysis of this algorithm
3) We give more details in the correctness proof

The separation approach

Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić, Parys 2019:

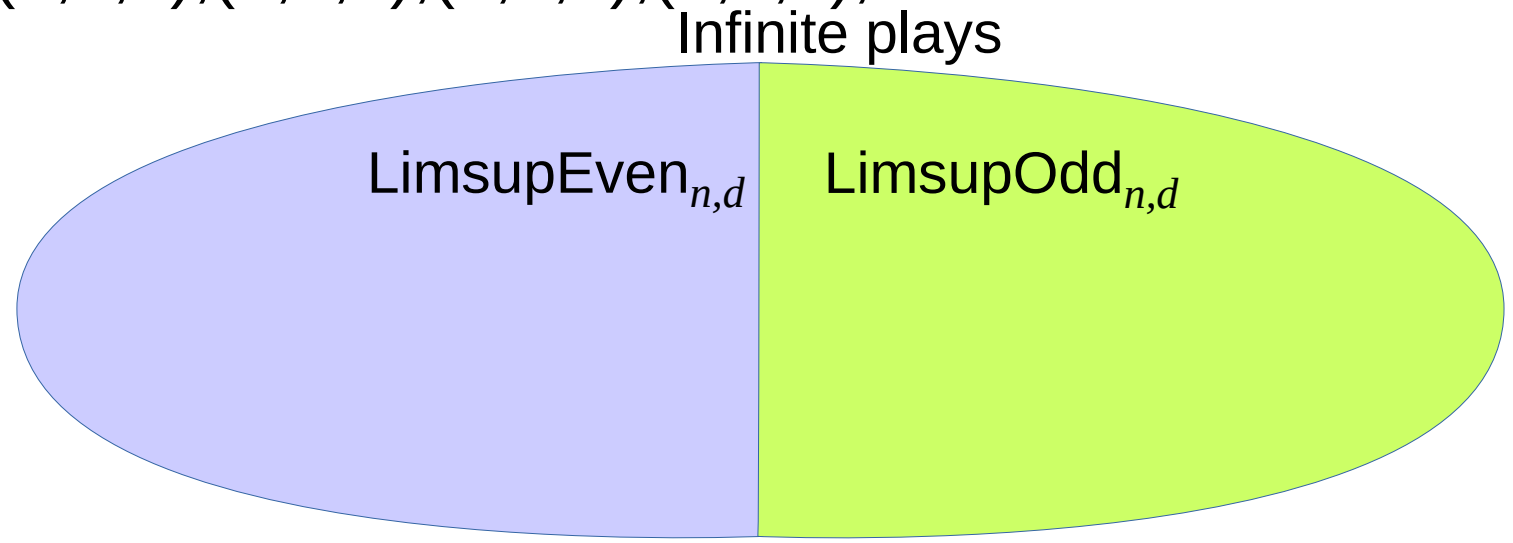
- the first four algorithms from this list (i.e., play summaries, antagonistic play summaries, succinct progress measures, register games) follow so-called separation approach
- (partial) quasipolynomial lower bound for the separation approach

The separation approach

Encoding of infinite plays – a sequence of triples:

- source vertex
- the priority read from this vertex
- target vertex

➔(1,1,2),(2,2,2),(2,3,3),(3,1,2),(2,3,1),...

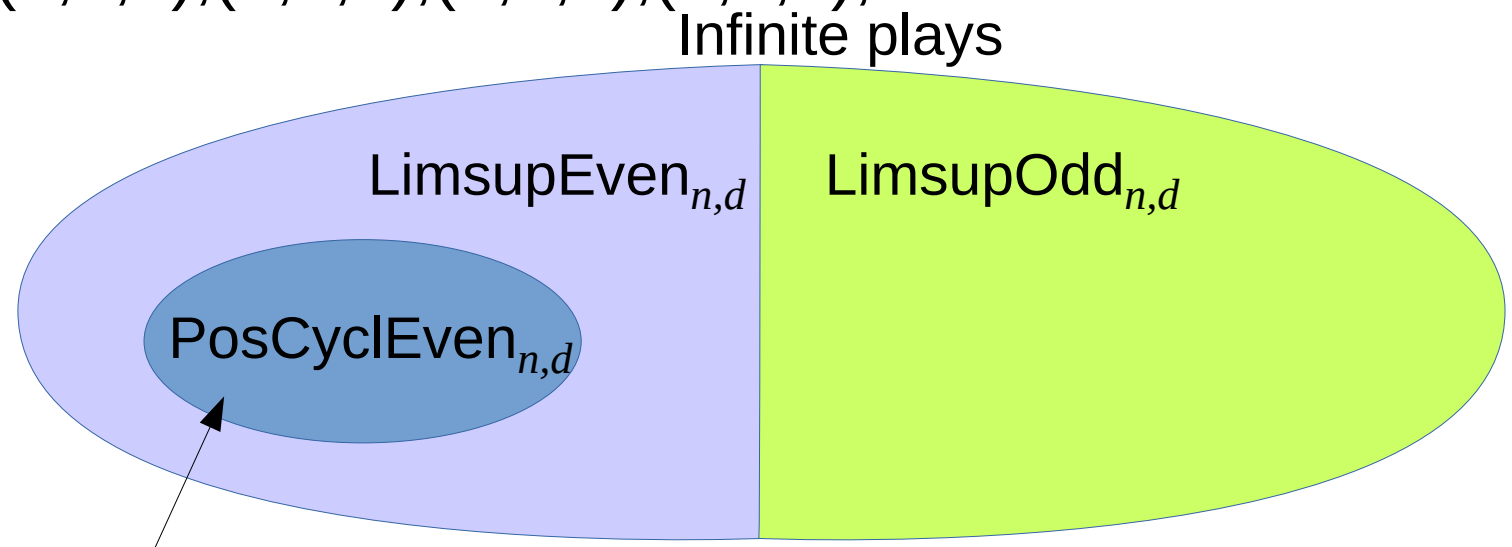


The separation approach

Encoding of infinite plays – a sequence of triples:

- source vertex
- the priority read from this vertex
- target vertex

➔(1,1,2),(2,2,2),(2,3,3),(3,1,2),(2,3,1),...

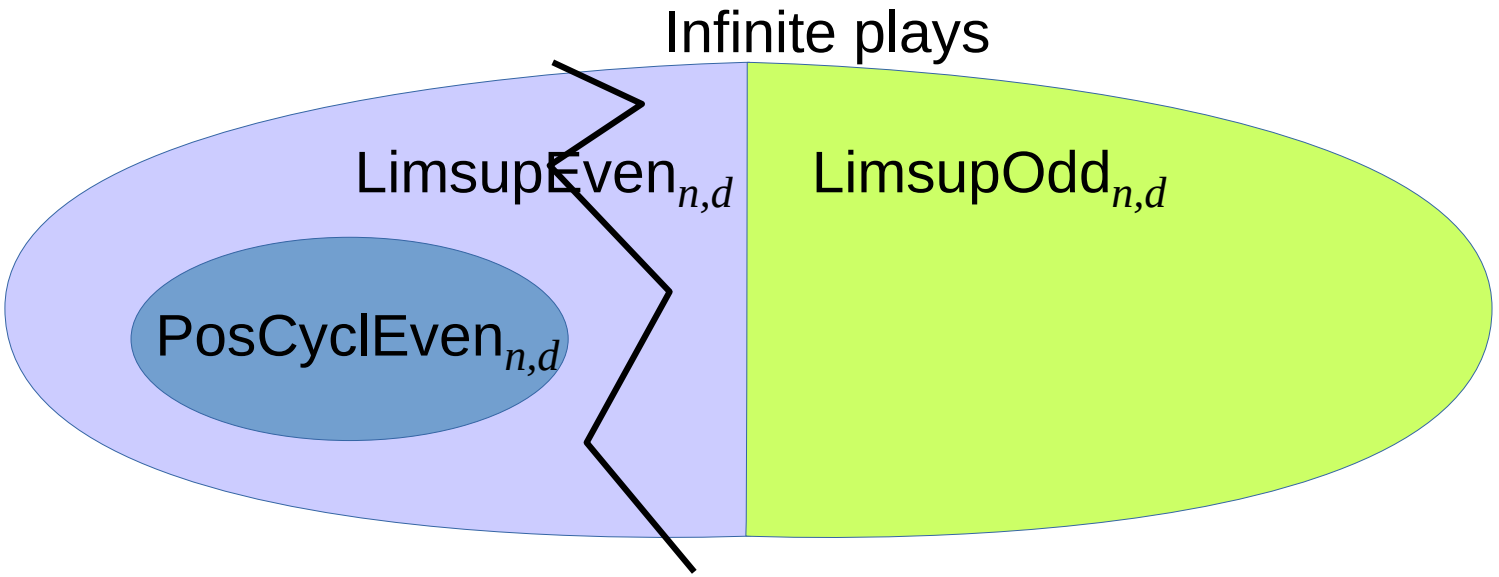


plays consistent with a positional winning strategy (in some game graph)

Theorem

If a player has a winning strategy, then it has a positional winning strategy (a move does not depend on the history, only on the current vertex)

The separation approach

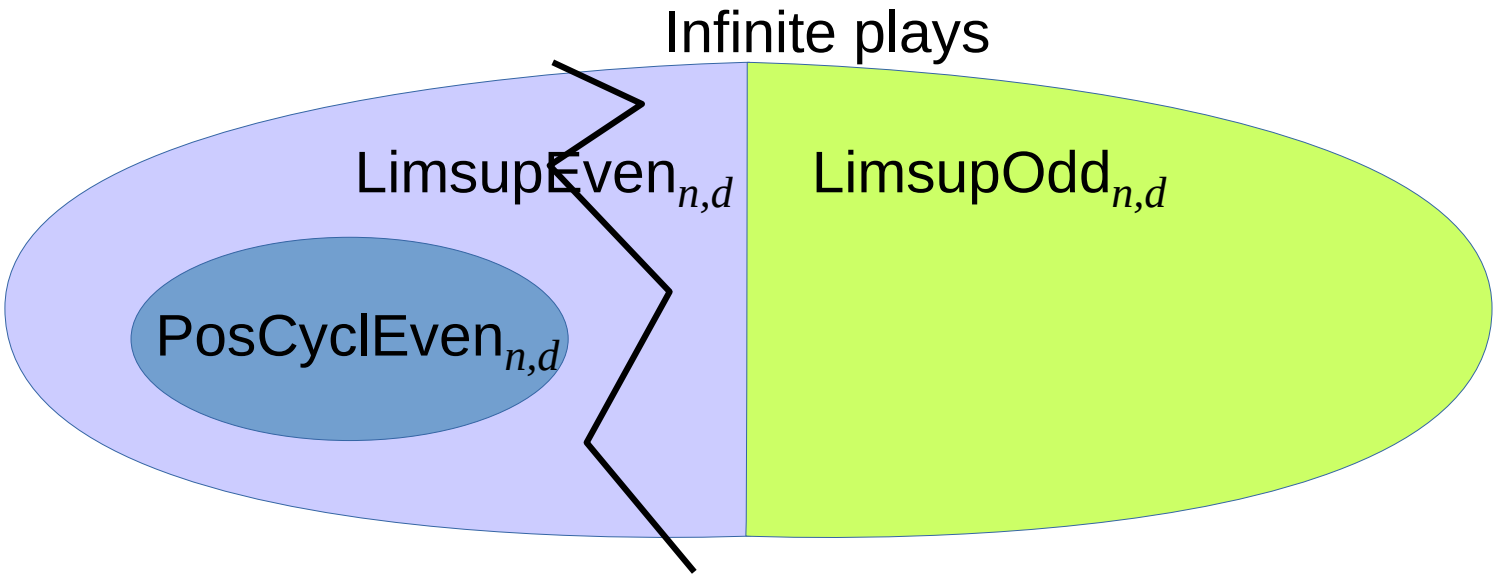


1) Construct an automaton A which
   - accepts plays compatible with a positional strategy for Even
   - rejects plays lost by Even
2) Consider the product game $G \times A$
3) Solve this game (larger but with a simpler winning condition)

   (running time $\approx$ size of A)

Remark: A does not depend on G, only on n and d

The separation approach



1) Construct an automaton A which
   - accepts plays compatible with a positional strategy for Even
   - rejects plays lost by Even

play summaries
antagonistic play summaries
succinct progress measures
} deterministic safety automaton

register games [Lehtinen] → nondet. parity automaton with log(n) priorities
(nondet. safety automaton)

2) Consider the product game $G \times A$
3) Solve this game (larger but with a simpler winning condition)

The separation approach – if A is **deterministic**

1) Construct an automaton A which
   - accepts plays compatible with a positional strategy for Even
   - rejects plays lost by Even
2) Consider the product game $G \times A$ – what does it mean?
   - players play in the G part
   - automaton A reads a play of G
3) Solve this game (larger but with a simpler winning condition)

Why does it work?

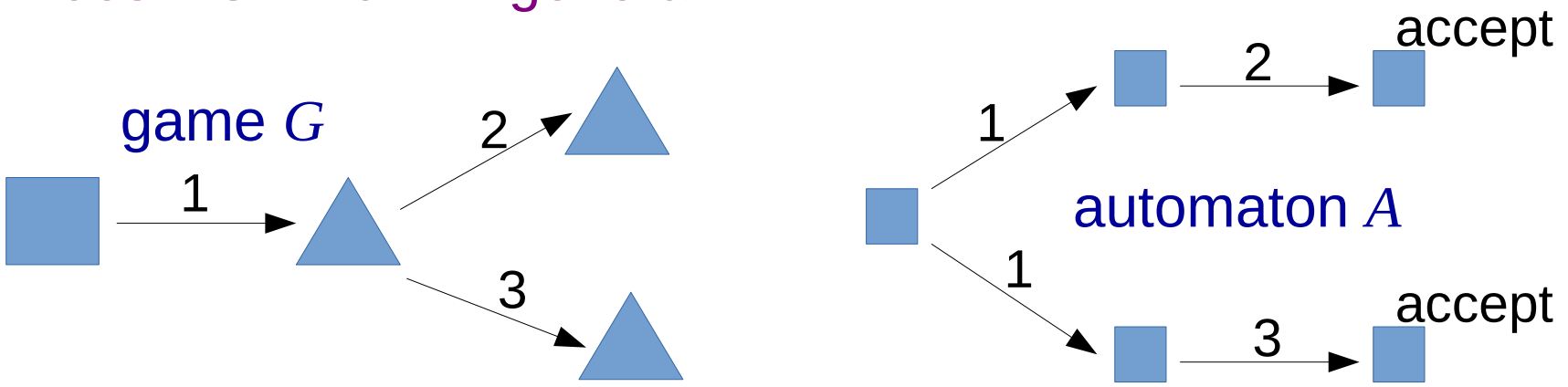
Even wins in $G \times A \Leftrightarrow$ Even wins in G
(the same winning strategy)

The separation approach – if A is **nondeterministic**

1) Construct an automaton A which
   - accepts plays compatible with a positional strategy for Even
   - rejects plays lost by Even
2) Consider the product game $G \times A$ – what does it mean?
   - players play in the G part
   - automaton A reads a play of G, player Even resolves nondeterminism
3) Solve this game (larger but with a simpler winning condition)

Why does it work?

Does NOT work in general



But works for the Lehtinen's "register automaton" - WHY?

Good-for-games automata

A (nondeterministic) automaton is good-for-games if there exists an acceptance strategy that depends only on the prefix of the word read so far (such that every word that can be somehow accepted, can also be accepted by following this strategy).

This is exactly what Even can do in $G \times A$.

So: for every good-for-games automaton A, and for every game G with winning condition $L(A)$:

Even wins in $G \times A \Leftrightarrow$ Even wins in G

Good-for-games automata

A (nondeterministic) automaton is good-for-games if there exists an acceptance strategy that depends only on the prefix of the word read so far (such that every word that can be somehow accepted, can also be accepted by following this strategy).

This is exactly what Even can do in $G \times A$.

So: for every good-for-games automaton A, and for every game G with winning condition $L(A)$:

Even wins in $G \times A \Leftrightarrow$ Even wins in G

But the Lehtinen's "register automaton" is not good-for-games.

So WHY does her algorithm work?

Suitable-for-parity-games automata

But the Lehtinen's "register automaton" is not good-for-games.
So WHY does her algorithm work?

We define a similar, but more admissive, notion of automata "**suitable-for-parity-games**" - there should exist an acceptance strategy (for plays following a positional winning strategy) that may depends on:

- a game G
- a positional winning strategy used by Even in G
- the prefix of the word read so far

Lemma 1: for every suitable-for-parity-games automaton A,
and for every parity game G:

Even wins in $G \times A \Leftrightarrow$ Even wins in G

Lemma 2: Lehtinen's register automaton is suitable-for-parity-games.

(for a similar notion of "good-for-small-games" automata,
see Colcombet & Fijalkow 2019)

Improvement in the complexity

Let: n – size of the original game
d – number of priorities in the original game

Lehtinen's "register automaton" A is a parity automaton with:

- size $n^{O(\log d)}$ a
- $O(\log n)$ priorities

After transforming to a safety automaton, it is of

- size $n^{O(\log d * \log n)}$ – and this is the complexity of the algorithm

This paper:

We prove that this parity automaton is of a special form, so it can be transformed to a safety automaton of

- size $n^{O(\log n)}$ – and this is the complexity of the algorithm

More precisely: we prove that in the product parity game $G \times A$, it is possible to win for Even without seeing n opponent's priorities in a row

Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$

Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.

Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.

$(1,1,1) \xrightarrow{4} (4,4,4)$

Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.

$(1,1,1) \xrightarrow[6]{4} \begin{matrix} \sout{(4,4,4)} \\ (4,4,1) \end{matrix}$

Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.

$$(1,1,1) \xrightarrow[6]{4} \begin{matrix} \cancel{(4,4,4)} \\ (4,4,1) \end{matrix} \xrightarrow{2} (4,4,2)$$

Details: how the “register automaton” works?

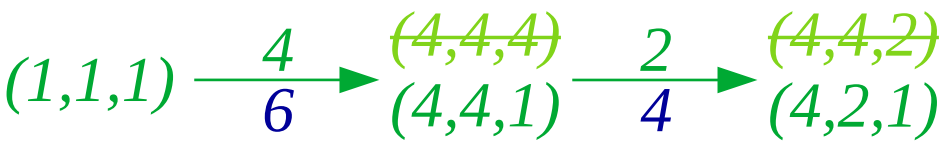
The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.



Details: how the “register automaton” works?

The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},...,r_1)$ of registers with values in $\{1,...,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.

$$(1,1,1) \xrightarrow[6]{4} \begin{matrix}\cancel{(4,4,4)}\\(4,4,1)\end{matrix} \xrightarrow[4]{2} \begin{matrix}\cancel{(4,4,2)}\\(4,2,1)\end{matrix} \xrightarrow[1]{3} (4,3,3)$$

Details: how the “register automaton” works?

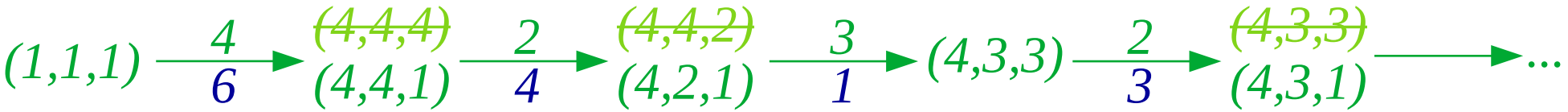
The automaton reads sequences of priorities, it should accept sequences coming from a positional winning strategy of Even.

Let: n – size of the original game

d – number of priorities in the original game

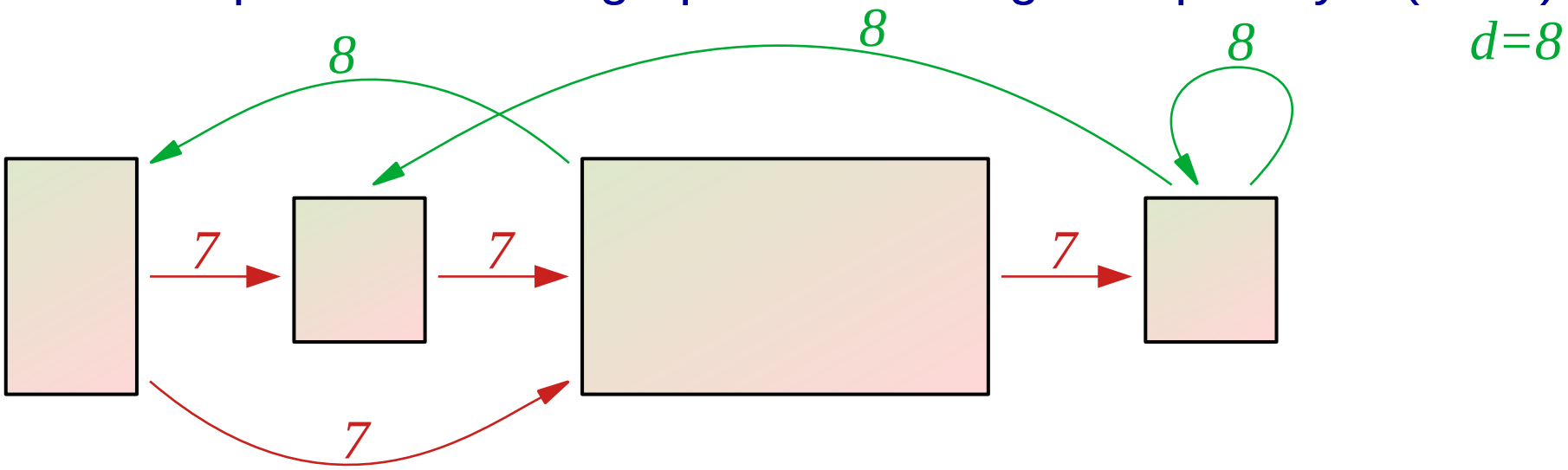
$rn(n)=1+\lfloor \log n \rfloor$ – number of registers needed

- States of A = non-increasing sequences $(r_{rn(n)},\ldots,r_1)$ of registers with values in $\{1,\ldots,d\}$
- While reading priority p, change all register values smaller than p to p.
- Then, possibly reset some register r_k to 1 and shift it to the end.
- Resetting r_k with even / odd value emits priority $2k$ / $2k+1$.



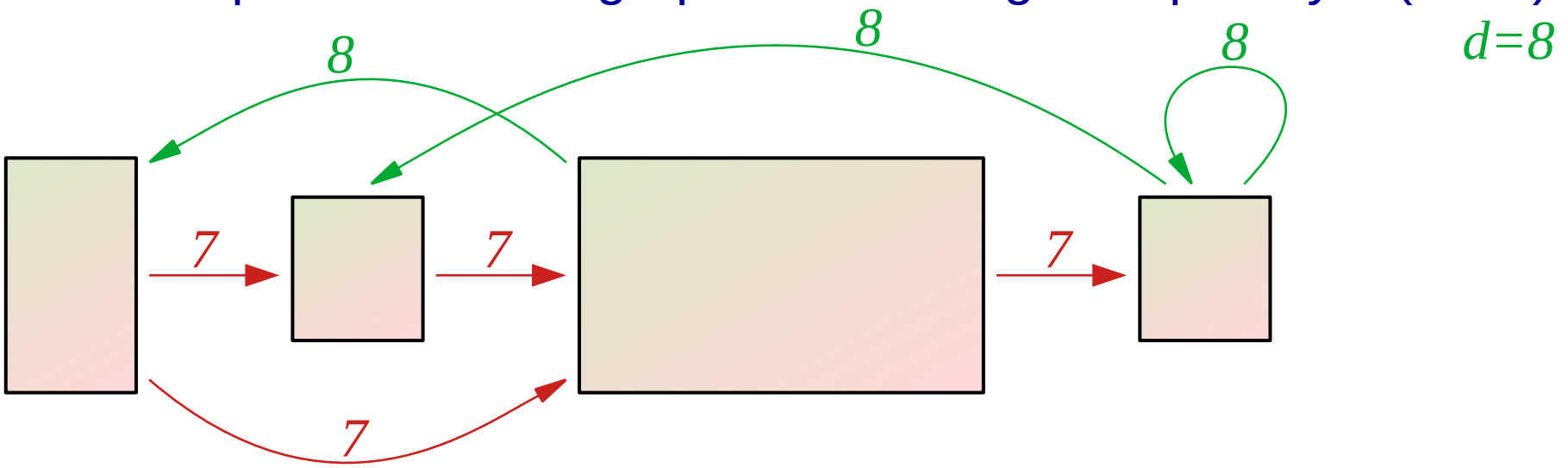
Details: why the “register automaton” works?

Fix a positional winning strategy of Even
Remove edges not appearing in this strategy
Consider SC components of the graph without edges of priority d (even)



Details: why the “register automaton” works?

Fix a positional winning strategy of Even
Remove edges not appearing in this strategy
Consider SC components of the graph without edges of priority d (even)



Correctness proof ideas:

- induction assumption: at the beginning no odd numbers in registers
- after entering a zone with n' nodes we remove odd numbers from registers $r_1,\ldots,r_{rn(n')}$ and we continue in the zone by the induction assumption
- if the play stays in a zone forever, it is accepted by the ind. ass.
- after reading d, we reset register $r_{rn(n)}$ (value d remains there)
- key point: at most one zone with $rn(n')=rn(n)$ (while entering this zone we have d in register $r_{rn(n)}$ – no need of odd reset)

Conclusion:

This paper makes a small step in the direction of understanding parity games:

1) We explain the Lehtinen's algorithm in the separation approach

2) Small improvement in the complexity analysis of this algorithm

3) We give more details in the correctness proof

Thank you