Register Games on Infinite Ordered Domains
(N,>) or (Q,>)

Léo Exibard, Emmanuel Filiot, Ayrat Khalimov
Register games extend two-player zero-sum turn-based games to infinite alphabets.

Play starts in the configuration \((q_0, 0^R)\), with all registers 0. Adam provides a data. It is compared with the register values. Based on the result, the game dictates a register assignment action and a successor state (unique). We update the registers and move into the new configuration. Repeat the same for Eve.

The parity acceptance is defined on vertices.

Strategies don’t see the data: Adam \(Tst^* \rightarrow \mathbb{N}\), Eve \(Tst^+ \rightarrow \mathbb{N}\). Solving such games is undecidable: they can simulate 2CMs.
In one-sided register games, only Adam provides a data, while Eve gives a label from a finite alphabet.

Strategies become: Adam $\Sigma^* \rightarrow \mathbb{N}$, Eve $\text{Tst}^+ \rightarrow \Sigma$.

Our results:
- They are **decidable** for $(\mathbb{N}, >)$ (ExpTime in $\#$ registers).
- **Finite memory** suffices for Eve.
- Either Eve wins or Adam wins (determinacy).
Example. Registers $R = \{r_l, r_M\}$, Eve finite alphabet is $\{a, b\}$.

In $\mathbb{N}$, Eve wins by always outputting $a$. Adam loses because:

But Adam wins in the domain $\mathbb{Q}$. 
Reduction to finite-arena games
Treat tests-assignments as labels, modify a winning condition:

\[ \text{EveWinPlays} = \{ \pi \mid \text{tst}_1 \text{asgn}_1 \ldots \text{of} \ \pi \ \text{is feasible} \Rightarrow \pi \models \alpha \} \].

In the example, the tests-assignments of the play \(1, 2, (3, 4)\omega\) are not feasible in \(\mathbb{N}\) (⇒ winning for Eve).

Sequence \(\text{tst}_1 \text{asgn}_1 \ldots\) is feasible \(\equiv\) generated by some data values:

\( (r_1 < * < r_2)(\downarrow r_1 r_2)(r_1 = r_2 = *)\) is feasible but

\( (r_1 < * < r_2)(\downarrow r_1 r_2)(r_1 < * < r_2)\) is not.

Detecting unfeasible sequences in \(\mathcal{Q}\) is easy: spot inconsistencies like above.

In \(\mathbb{N}\) it is hard. Studying constraint sequences (page 5) gives us det-max automaton that can detect unfeasible sequences.

Finally, with a help of a pigeon\(^\dagger\), we get rid of det-max automata and get games with standard \(\omega\)-regular winning conditions.

\(^\dagger\): the proof relies on determinacy and finite-mem of \(\omega\)-regular games, and the pigeon hole
Satisfiability of constraint sequences in $\mathbb{N}$

A constraint defines the relation between the registers. A constraint sequence describes how these relations evolve:

We prove that a constraint sequence is unfeasible in $\mathbb{N}$ iff it

has an infinite decreasing chain

or

has unbounded decreasing or increasing trespassing chains

We construct a det-max automaton that reads constraint sequences and spots such chains.
Finally, we apply these results to **transducer synthesis**: input: deterministic “transducer-like” register automaton $S$ output: transducer $T$ such that $L(T) \subseteq L(S)$, or ‘unrealisable’. And show it is decidable.