TECHNISCHE UNIVERSITÄT WIEN

On the computational complexity of unfriendly partitions

Vittorio Cipriani (TU Wien)

j.w.w. Belanger, Goh, Jain, Richter, Stephan

CCA (Swansea)

July 18, 2024

Notations, definitions

In this talk, all graphs $G = (V, E)$ we consider are

- *countable*,
- *undirected*,
- and *no self-loops*.

Notations, definitions

In this talk, all graphs $G = (V, E)$ we consider are

- *countable*,
- *undirected*,
- and *no self-loops*.

For $v \in V(G)$, $N(v)$ denotes the set of *neighbors* of v in G, i.e.

$$N(v) := \{w \in V(G) : (v, w) \in E(G)\}.$$

Notations, definitions

In this talk, all graphs $G = (V, E)$ we consider are

- *countable*,
- *undirected*,
- and *no self-loops*.

For $v \in V(G)$, $N(v)$ denotes the set of *neighbors* of v in G, i.e.

$$N(v) := \{w \in V(G) : (v, w) \in E(G)\}.$$

A *partition* $V(G) = V_0 \cup V_1$ is *unfriendly* if

$$(\forall i < 2)\ (\forall v \in V_i)\ (|V_i \cap N(v)| \leq |V_{1-i} \cap N(v)|).$$

Notations, definitions

In this talk, all graphs $G = (V, E)$ we consider are

- *countable*,
- *undirected*,
- and *no self-loops*.

For $v \in V(G)$, $N(v)$ denotes the set of *neighbors* of v in G, i.e.

$$N(v) := \{w \in V(G) : (v, w) \in E(G)\}.$$

A *partition* $V(G) = V_0 \cup V_1$ is *unfriendly* if

$$(\forall i < 2)\ (\forall v \in V_i)\ (|V_i \cap N(v)| \leq |V_{1-i} \cap N(v)|).$$

Informally, for a partition to be unfriendly, we require that any vertex has at most as many neighbors in the opposite partition than in its own one.

Known results

Finite graphs always admit an unfriendly partition.

Known results

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Known results

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Question (Cowan, Emerson): Does any graph G admit an unfriendly partition?

Known results

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Question (Cowan, Emerson): Does any graph G admit an unfriendly partition?
(Milner, Shelah): No (their G is uncountable).

Known results

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Question (Cowan, Emerson): Does any graph G admit an unfriendly partition?
(Milner, Shelah): No (their G is uncountable).

Question, rephrased: Does any countable graph G admit an unfriendly partition?

Known results

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Question (Cowan, Emerson): Does any graph G admit an unfriendly partition?
(Milner, Shelah): No (their G is uncountable).

Question, rephrased: Does any countable graph G admit an unfriendly partition?
The question is still open.

Known results

TECHNISCHE UNIVERSITÄT WIEN

Finite graphs always admit an unfriendly partition.
By ∼compactness, the same holds for *locally finite graphs* (graph in which every vertex has finitely many neighbors).

Question (Cowan, Emerson): Does any graph G admit an unfriendly partition?
(Milner, Shelah): No (their G is uncountable).

Question, rephrased: Does any countable graph G admit an unfriendly partition?
The question is still open.

Remark: Milner, Shelah also showed that any graph has an *unfriendly 3-partition*.

When unfriendly partitions exist

TECHNISCHE UNIVERSITÄT WIEN

(Aharoni, Milner, Prikry)

- graphs with only *finitely many vertices with infinite degrees*;

When unfriendly partitions exist

TECHNISCHE UNIVERSITÄT WIEN

(Aharoni, Milner, Prikry)

- graphs with only *finitely many vertices with infinite degrees*;

(Bruhn, Diestel)

- graphs *not* containing *infinite rays*;

When unfriendly partitions exist

TECHNISCHE UNIVERSITÄT WIEN

(Aharoni, Milner, Prikry)

- graphs with only *finitely many vertices with infinite degrees*;

(Bruhn, Diestel)

- graphs *not* containing *infinite rays*;

(Berger)

- graphs *not* containing a *subdivision of the infinite clique*;

When unfriendly partitions exist

TECHNISCHE UNIVERSITÄT WIEN

(Aharoni, Milner, Prikry)

- graphs with only *finitely many vertices with infinite degrees*;

(Bruhn, Diestel)

- graphs *not* containing *infinite rays*;

(Berger)

- graphs *not* containing a *subdivision of the infinite clique*;

...

When unfriendly partitions exist

TECHNISCHE UNIVERSITÄT WIEN

(Aharoni, Milner, Prikry)

- graphs with only *finitely many vertices with infinite degrees*;

(Bruhn, Diestel)

- graphs *not* containing *infinite rays*;

(Berger)

- graphs *not* containing a *subdivision of the infinite clique*;

...

In this talk, we are interested in understanding

how hard is to find an unfriendly partition of a graph?

Weihrauch reducibility

Problems

TECHNISCHE UNIVERSITÄT WIEN

Definition

A represented space $\mathbf{X}$ is a pair $(X, \delta_{\mathbf{X}})$ where $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$.
We say that $p \in \mathbb{N}^{\mathbb{N}}$ is a *name* for x if $\delta_X(p) = x$.

Problems

TECHNISCHE UNIVERSITÄT WIEN

Definition

A represented space $\mathbf{X}$ is a pair $(X, \delta_{\mathbf{X}})$ where $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$.
We say that $p \in \mathbb{N}^{\mathbb{N}}$ is a *name* for x if $\delta_X(p) = x$.

Here graphs are represented via their characteristic functions.

Problems

TECHNISCHE UNIVERSITÄT WIEN

Definition

A represented space $\mathbf{X}$ is a pair $(X, \delta_{\mathbf{X}})$ where $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$. We say that $p \in \mathbb{N}^{\mathbb{N}}$ is a *name* for x if $\delta_X(p) = x$.

Here graphs are represented via their characteristic functions. A *problem* $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ is a "function" assuming > 1 distinct values in the range for at least one point in the domain.

Problems

TECHNISCHE UNIVERSITÄT WIEN

Definition

A represented space $\mathbf{X}$ is a pair $(X, \delta_{\mathbf{X}})$ where $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$.
We say that $p \in \mathbb{N}^{\mathbb{N}}$ is a *name* for x if $\delta_X(p) = x$.

Here graphs are represented via their characteristic functions.
A *problem* $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ is a "function" assuming > 1 distinct values in the range for at least one point in the domain.

Let $\mathcal{F}$ be a class of graphs for which an unfriendly partition always exists.

Problems

TECHNISCHE UNIVERSITÄT WIEN

Definition

A represented space $\mathbf{X}$ is a pair $(X, \delta_{\mathbf{X}})$ where $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$.
We say that $p \in \mathbb{N}^{\mathbb{N}}$ is a *name* for x if $\delta_X(p) = x$.

Here graphs are represented via their characteristic functions.
A *problem* $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ is a "function" assuming > 1 distinct values in the range for at least one point in the domain.

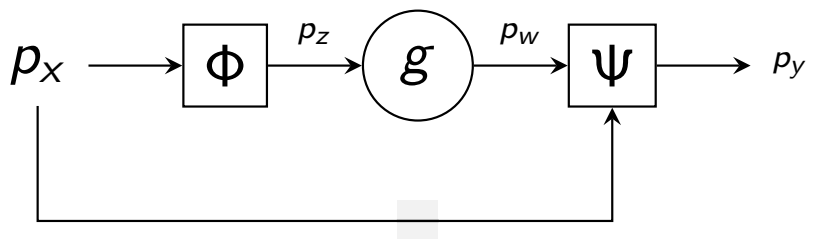
Let $\mathcal{F}$ be a class of graphs for which an unfriendly partition always exists.

$\mathrm{UP}_{\mathcal{F}}$

Input: $G \in \mathcal{F}$.
Output: an unfriendly partition of G (an element of $2^{\mathbb{N}}$).

Let f and g be problems on represented spaces. Then f is *Weihrauch reducible* to a problem g ($f \leq_{\mathrm{W}} g$), if there are computable maps $\Phi, \Psi : \mathbb{N}^{\mathbb{N}} \rightrightarrows \mathbb{N}^{\mathbb{N}}$ s.t.

- for every name p_x for some $x \in dom(f)$, $\Phi(p_x) = p_z$, where p_z is a name for some $z \in dom(g)$ and,
- for every name p_w for some $w \in g(z)$, $\Psi(p_x \oplus p_w) = p_y$ where p_y is a name for $y \in f(x)$.



Some problems in the W. lattice

TECHNISCHE UNIVERSITÄT WIEN

$\mathsf{UP}_{\mathcal{F}}$

Input: $G \in \mathcal{F}$.
Output: an unfriendly partition of G (an element of $2^{\mathbb{N}}$).

$\mathsf{UP}_{\mathcal{F}}$

Input: $G \in \mathcal{F}$.
Output: an unfriendly partition of G (an element of $2^{\mathbb{N}}$).

In general, for a problem f, the *parallelization* of f, denoted by $\hat{f}$ is the problem that solves infinitely many instances of f in parallel.

$\mathrm{UP}_{\mathcal{F}}$

Input: $G \in \mathcal{F}$.
Output: an unfriendly partition of G (an element of $2^{\mathbb{N}}$).

In general, for a problem f, the *parallelization* of f, denoted by $\hat{f}$ is the problem that solves infinitely many instances of f in parallel.
It is easy to notice that $\mathrm{UP}_{\mathcal{F}}$ is *parallelizable*, i.e., $\mathrm{UP}_{\mathcal{F}} \equiv_{\mathrm{W}} \widehat{\mathrm{UP}_{\mathcal{F}}}$.

Some problems in the W. lattice

TECHNISCHE UNIVERSITÄT WIEN

$\mathrm{UP}_{\mathcal{F}}$

Input: $G \in \mathcal{F}$.
Output: an unfriendly partition of G (an element of $2^{\mathbb{N}}$).

In general, for a problem f, the *parallelization* of f, denoted by $\hat{f}$ is the problem that solves infinitely many instances of f in parallel. It is easy to notice that $\mathrm{UP}_{\mathcal{F}}$ is *parallelizable*, i.e., $\mathrm{UP}_{\mathcal{F}} \equiv_{\mathrm{W}} \widehat{\mathrm{UP}_{\mathcal{F}}}$.

LPO

Input: $p \in 2^{\mathbb{N}}$.
Output: 0 if $p = 0^{\mathbb{N}}$, 1 otherwise.

Notice that $\mathrm{UP}_{\mathcal{F}}$ for $\mathcal{F}$ being the class of graphs in which every connected component has size at most 4 is computable.

Notice that $\mathsf{UP}_{\mathcal{F}}$ for $\mathcal{F}$ being the class of graphs in which every connected component has size at most 4 is computable.

Theorem

Let $\mathcal{F}$ be the class of graphs in which every connected component is finite (even size at most 5): then $\mathsf{UP}_{\mathcal{F}} \equiv_{\mathrm{W}} \widehat{\mathsf{LPO}}$.

Notice that $\mathsf{UP}_{\mathcal{F}}$ for $\mathcal{F}$ being the class of graphs in which every connected component has size at most 4 is computable.

Theorem

Let $\mathcal{F}$ be the class of graphs in which every connected component is finite (even size at most 5): then $\mathsf{UP}_{\mathcal{F}} \equiv_{\mathrm{W}} \widehat{\mathsf{LPO}}$.

Sketch of $\mathsf{UP}_{\mathcal{F}} \leq_{\mathrm{W}} \widehat{\mathsf{LPO}}$

Notice that $\mathrm{UP}_{\mathcal{F}}$ for $\mathcal{F}$ being the class of graphs in which every connected component has size at most 4 is computable.

Theorem

Let $\mathcal{F}$ be the class of graphs in which every connected component is finite (even size at most 5): then $\mathrm{UP}_{\mathcal{F}} \equiv_{\mathrm{W}} \widehat{\mathrm{LPO}}$.

Sketch of $\mathrm{UP}_{\mathcal{F}} \leq_{\mathrm{W}} \widehat{\mathrm{LPO}}$
Gura, Hirst and Mummert showed that $\widehat{\mathrm{LPO}}$ is enough to compute all disconnected components of a graph. Then, we can computably find an unfriendly partition of a finite graph.

Sketch $\widehat{\mathsf{LPO}} \leq_{\mathrm{W}} \mathsf{UP}_{\mathcal{F}}$ (Shafer)

Sketch $\widehat{\mathrm{LPO}} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$ (Shafer)

Since $\mathrm{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\mathrm{LPO} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$.
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.

Sketch $\widehat{\mathrm{LPO}} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$ (Shafer)

Since $\mathrm{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\mathrm{LPO} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$.
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.
Reduction: We compute an input G for $\mathrm{UP}_{\mathcal{F}}$ as follows.

Sketch $\widehat{\mathrm{LPO}} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$ (Shafer)

Since $\mathrm{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\mathrm{LPO} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$.
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.
Reduction: We compute an input G for $\mathrm{UP}_{\mathcal{F}}$ as follows.
At stage 0, let G be $0 \text{——} 1$.

Sketch $\widehat{\mathrm{LPO}} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$ (Shafer)

Since $\mathrm{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\mathrm{LPO} \leq_{\mathrm{W}} \mathrm{UP}_{\mathcal{F}}$.
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.
Reduction: We compute an input G for $\mathrm{UP}_{\mathcal{F}}$ as follows.
At stage 0, let G be $0 \text{——} 1$.
If $p = 0^{\mathbb{N}}$, no further vertex will be added.

Sketch $\widehat{\text{LPO}} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$ (Shafer)

Since $\text{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\text{LPO} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$.

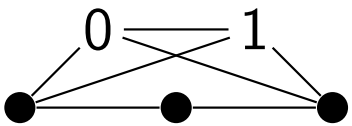
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.

Reduction: We compute an input G for $\text{UP}_{\mathcal{F}}$ as follows.

At stage 0, let G be $0 \text{——} 1$.

If $p = 0^{\mathbb{N}}$, no further vertex will be added.

If at some stage we witness $p \neq 0^{\mathbb{N}}$, we do the following:



Sketch $\widehat{\text{LPO}} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$ (Shafer)

Since $\text{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\text{LPO} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$.

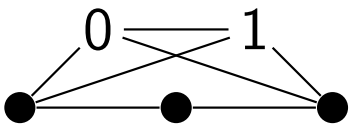
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.

Reduction: We compute an input G for $\text{UP}_{\mathcal{F}}$ as follows.

At stage 0, let G be $0 \text{——} 1$.

If $p = 0^{\mathbb{N}}$, no further vertex will be added.

If at some stage we witness $p \neq 0^{\mathbb{N}}$, we do the following:



Verification: Any u.p. of:

Sketch $\widehat{\text{LPO}} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$ (Shafer)

Since $\text{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\text{LPO} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$.

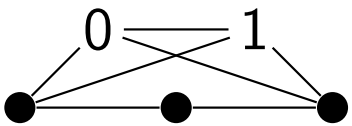
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.

Reduction: We compute an input G for $\text{UP}_{\mathcal{F}}$ as follows.

At stage 0, let G be $0 \text{——} 1$.

If $p = 0^{\mathbb{N}}$, no further vertex will be added.

If at some stage we witness $p \neq 0^{\mathbb{N}}$, we do the following:



Verification: Any u.p. of:

$0 \text{——} 1$ forces 0 and 1 to be in <u>different</u> partitions;

Sketch $\widehat{\text{LPO}} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$ (Shafer)

Since $\text{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\text{LPO} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$.

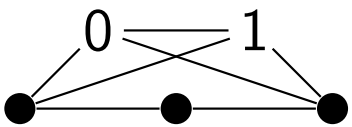
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.

Reduction: We compute an input G for $\text{UP}_{\mathcal{F}}$ as follows.

At stage 0, let G be $0 \text{——} 1$.

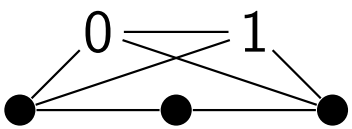
If $p = 0^{\mathbb{N}}$, no further vertex will be added.

If at some stage we witness $p \neq 0^{\mathbb{N}}$, we do the following:



Verification: Any u.p. of:

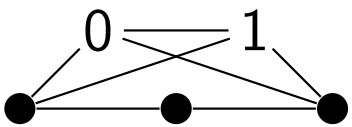
$0 \text{——} 1$ forces 0 and 1 to be in <u>different</u> partitions;



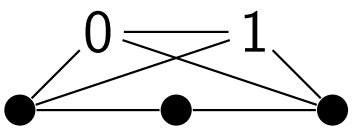
forces 0 and 1 to be in <u>the same</u> partition;

Sketch $\widehat{\text{LPO}} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$ (Shafer)

Since $\text{UP}_{\mathcal{F}}$ is parallelizable, it suffices to show that $\text{LPO} \leq_{\text{W}} \text{UP}_{\mathcal{F}}$.
Let $p \in 2^{\mathbb{N}}$ be the input for LPO.
Reduction: We compute an input G for $\text{UP}_{\mathcal{F}}$ as follows.
At stage 0, let G be $0 \text{——} 1$.
If $p = 0^{\mathbb{N}}$, no further vertex will be added.
If at some stage we witness $p \neq 0^{\mathbb{N}}$, we do the following:



Verification: Any u.p. of:
$0 \text{——} 1$ forces 0 and 1 to be in <u>different</u> partitions;



forces 0 and 1 to be in <u>the same</u> partition;

Hence, given any u.p. of G, we have that 0 and 1 are in different partitions iff $p = 0^{\mathbb{N}}$ iff $\text{LPO}(p) = 0$.

WKL

Input: an ill-founded binary tree T.
Output: a path in $[T]$.

WKL

Input: an ill-founded binary tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}} \widehat{\mathrm{LPO}}$.

WKL

Input: an ill-founded binary tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}} \widehat{\mathrm{LPO}}$.
A graph G is

- a *forest* if it is the disconnected union of trees.
- *bipartite* if it does not contain any odd-length cycles.

WKL

Input: an ill-founded binary tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}} \widehat{\mathsf{LPO}}$.
A graph G is

- a *forest* if it is the disconnected union of trees.
- *bipartite* if it does not contain any odd-length cycles.

Theorem

$\mathsf{WKL} \equiv_{\mathrm{W}} \mathsf{UP}_{\mathrm{Forests}} \equiv_{\mathrm{W}} \mathsf{UP}_{\mathrm{Bipartite}}$.

Sketch WKL $\leq_W$ $\mathsf{UP}_{\text{Forests}}$

Let T be an ill-founded[1] binary tree.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_W$ $\mathsf{UP}_{\text{Forests}}$

Let T be an ill-founded[1] binary tree.
Reduction: We compute G as follows.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_W$ $\mathrm{UP}_{\mathrm{Forests}}$

Let T be an ill-founded[1] binary tree.

Reduction: We compute G as follows.

For any $\sigma \in T$ we add 5 isolated vertices $a_\sigma, b_\sigma, c_\sigma, d_\sigma, e_\sigma$.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_{\mathrm{W}}$ $\mathsf{UP}_{\mathrm{Forests}}$

Let T be an ill-founded[1] binary tree.
Reduction: We compute G as follows.
For any $\sigma \in T$ we add 5 isolated vertices $a_\sigma, b_\sigma, c_\sigma, d_\sigma, e_\sigma$.
We start to check if $\sigma^\frown 0 \vee \sigma^\frown 1$ is well-founded.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_W$ $UP_{Forests}$

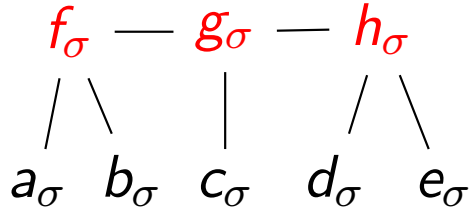
Let T be an ill-founded[1] binary tree.

Reduction: We compute G as follows.

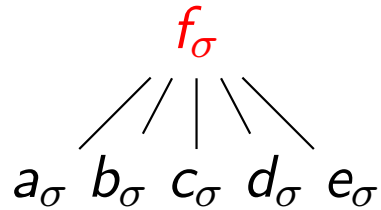
For any $\sigma \in T$ we add 5 isolated vertices $a_\sigma, b_\sigma, c_\sigma, d_\sigma, e_\sigma$.

We start to check if $\sigma^\frown 0 \vee \sigma^\frown 1$ is well-founded.

if $\sigma^\frown 0$ is well-founded



if $\sigma^\frown 1$ is well-founded



[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_W$ $\mathrm{UP}_{\mathrm{Forests}}$

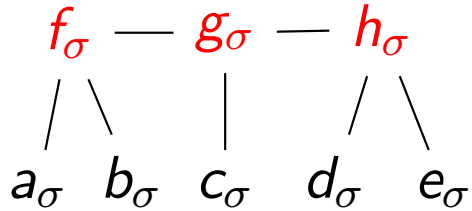
Let T be an ill-founded[1] binary tree.

Reduction: We compute G as follows.

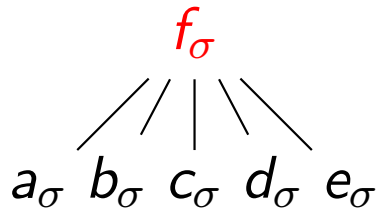
For any $\sigma \in T$ we add 5 isolated vertices $a_\sigma, b_\sigma, c_\sigma, d_\sigma, e_\sigma$.

We start to check if $\sigma^\frown 0 \vee \sigma^\frown 1$ is well-founded.

if $\sigma^\frown 0$ is well-founded



if $\sigma^\frown 1$ is well-founded



Verification Any u.p. of

- the lhs forces $a_\sigma - e_\sigma$ to be in different partitions,
- the rhs forces $a_\sigma - e_\sigma$ to be in the same partition.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

Sketch WKL $\leq_W$ $UP_{Forests}$

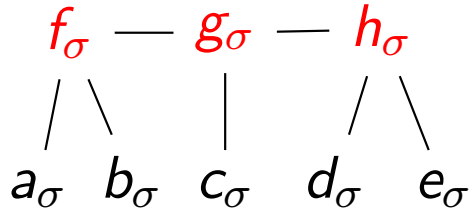
Let T be an ill-founded[1] binary tree.

Reduction: We compute G as follows.

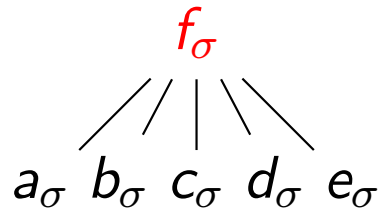
For any $\sigma \in T$ we add 5 isolated vertices $a_\sigma, b_\sigma, c_\sigma, d_\sigma, e_\sigma$.

We start to check if $\sigma^\frown 0 \vee \sigma^\frown 1$ is well-founded.

if $\sigma^\frown 0$ is well-founded



if $\sigma^\frown 1$ is well-founded



Verification Any u.p. of

- the lhs forces $a_\sigma - e_\sigma$ to be in different partitions,
- the rhs forces $a_\sigma - e_\sigma$ to be in the same partition.

Given an u.p. of G we compute $p \in [T]$ letting $p(n+1) = 0$ if the u.p. does not have $a_{p[n]} - e_{p[n]}$ in the same partition and 1 otherwise.

[1] being ill-founded is a Π^0_1 condition and hence well-founded is Σ^0_1

KL

Input: an ill-founded finitely branching tree T.
Output: a path in $[T]$.

KL

Input: an ill-founded finitely branching tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}}$ $\widehat{\mathsf{LPO}}$ $<_{\mathrm{W}}$ KL.

KL

Input: an ill-founded finitely branching tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}}$ $\widehat{\mathsf{LPO}}$ $<_{\mathrm{W}}$ KL.
Let $\mathcal{LF}$ be the class of locally finite graphs.

KL

Input: an ill-founded finitely branching tree T.
Output: a path in $[T]$.

Notice that WKL $<_{\mathrm{W}}$ $\widehat{\mathsf{LPO}}$ $<_{\mathrm{W}}$ KL.
Let $\mathcal{LF}$ be the class of locally finite graphs.

Theorem

KL $\equiv_{\mathrm{W}}$ $\mathsf{UP}_{\mathcal{LF}}$.

"Stratifying" $\mathrm{UP}_{\mathcal{F}}$

Notice that:

- not having a subdivision of a 3-clique is the same as bipartite;
- we proved that $\mathrm{UP}_{\mathcal{F}_4} \equiv_{\mathrm{W}} \mathrm{lim}$ for $\mathcal{F}_4$ being the class of graphs with only finite components and no subdivision of the infinite 4-clique.

"Stratifying" $\mathrm{UP}_{\mathcal{F}}$

Notice that:

- not having a subdivision of a 3-clique is the same as bipartite;
- we proved that $\mathrm{UP}_{\mathcal{F}_4} \equiv_{\mathrm{W}} \lim$ for $\mathcal{F}_4$ being the class of graphs with only finite components and no subdivision of the infinite 4-clique.

Question: is there a nice characterization of graph (not) having a subdivision of an n-clique?

"Stratifying" $\mathrm{UP}_{\mathcal{F}}$

Notice that:

- not having a subdivision of a 3-clique is the same as bipartite;
- we proved that $\mathrm{UP}_{\mathcal{F}_4} \equiv_{\mathrm{W}} \mathrm{lim}$ for $\mathcal{F}_4$ being the class of graphs with only finite components and no subdivision of the infinite 4-clique.

Question: is there a nice characterization of graph (not) having a subdivision of an n-clique?

Question: are there other natural classes of graphs that can be characterized in terms of well-known problems?

Above arithmetic

These (and other) results that we have can also be rephrased as

"There is a computable graph G s.t. any u.p. of G computes

- $\emptyset'$
- PA
- $\mathrm{PA}(\emptyset')$
- a DNR_n function

and vice-versa".

These (and other) results that we have can also be rephrased as

"There is a computable graph G s.t. any u.p. of G computes

- $\emptyset'$
- PA
- $\mathrm{PA}(\emptyset')$
- a DNR_n function

and vice-versa".

Something more?

Theorem

There is a computable graph which has unfriendly partitions such that each such partition computes all hyperarithmetic sets.
No unfriendly partition of such a graph can be Σ^1_1 *or* Π^1_1.

Theorem

There is a computable graph which has unfriendly partitions such that each such partition computes all hyperarithmetic sets.
No unfriendly partition of such a graph can be Σ^1_1 *or* Π^1_1. $\Longleftarrow$

Let $V(G) = V_0 \cup V_1$ be an u.p. of a graph G and consider $A = \{(v, c) : v \in V_c\}$ for $c \in \{0, 1\}$.

Theorem

There is a computable graph which has unfriendly partitions such that each such partition computes all hyperarithmetic sets.
No unfriendly partition of such a graph can be Σ^1_1 *or* Π^1_1. $\Longleftarrow$

Let $V(G) = V_0 \cup V_1$ be an u.p. of a graph G and consider $A = \{(v, c) : v \in V_c\}$ for $c \in \{0, 1\}$.
The complement of A is $\overline{A} := \{(v, c) : v \notin V_c\}$: clearly if A is Σ^1_1 (Π^1_1) $\overline{A}$ is a Π^1_1 (Σ^1_1) set.

Theorem

There is a computable graph which has unfriendly partitions such that each such partition computes all hyperarithmetic sets.
No unfriendly partition of such a graph can be Σ^1_1 *or* Π^1_1. $\Longleftarrow$

Let $V(G) = V_0 \cup V_1$ be an u.p. of a graph G and consider $A = \{(v, c) : v \in V_c\}$ for $c \in \{0, 1\}$.
The complement of A is $\overline{A} := \{(v, c) : v \notin V_c\}$: clearly if A is Σ^1_1 (Π^1_1) $\overline{A}$ is a Π^1_1 (Σ^1_1) set. But now we have that $(v, c) \in A \iff (v, 1 - c) \in \overline{A}$, and hence A cannot be Σ^1_1 or Π^1_1.

Theorem

There is a computable graph which has unfriendly partitions such that each such partition computes all hyperarithmetic sets. $\Longleftarrow$
No unfriendly partition of such a graph can be Σ^1_1 *or* Π^1_1.

Let $V(G) = V_0 \cup V_1$ be an u.p. of a graph G and consider $A = \{(v, c) : v \in V_c\}$ for $c \in \{0, 1\}$.
The complement of A is $\overline{A} := \{(v, c) : v \notin V_c\}$: clearly if A is Σ^1_1 (Π^1_1) $\overline{A}$ is a Π^1_1 (Σ^1_1) set. But now we have that $(v, c) \in A \iff (v, 1 - c) \in \overline{A}$, and hence A cannot be Σ^1_1 or Π^1_1.

Before giving a (rough) description of the graph mentioned in the theorem, we describe some of its parts: namely the *NOT* and *NAND* gates.

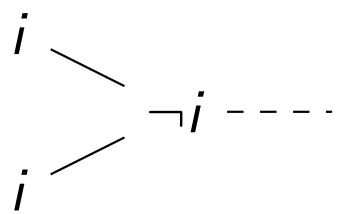
NOT gate

Warning: Gates may not "work" as desired: That is, we rely on the output of some gate only under some special conditions (to be defined later).

NOT gate

Warning: Gates may not "work" as desired: That is, we rely on the output of some gate only under some special conditions (to be defined later).

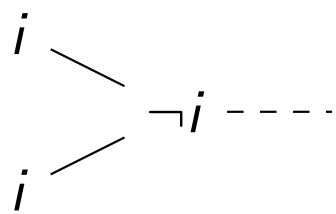
NOT gate: if both inputs vertices of the gate (labelled by i) are in the same partition, then the output vertex, labelled by $\neg i$ must be in the opposite one (provided that $\neg a$ has at most 3 neighbors).



NOT gate

Warning: Gates may not "work" as desired: That is, we rely on the output of some gate only under some special conditions (to be defined later).

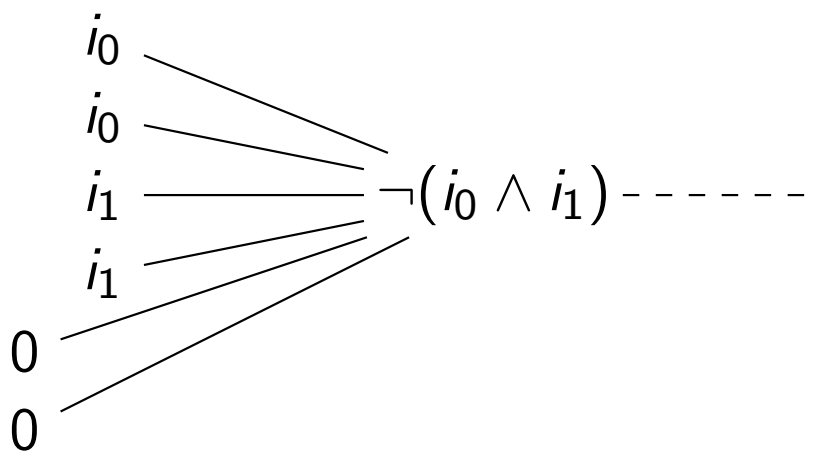
NOT gate: if both inputs vertices of the gate (labelled by i) are in the same partition, then the output vertex, labelled by $\neg i$ must be in the opposite one (provided that $\neg a$ has at most 3 neighbors).



The gate does not "work" if the i's are in different partitions.

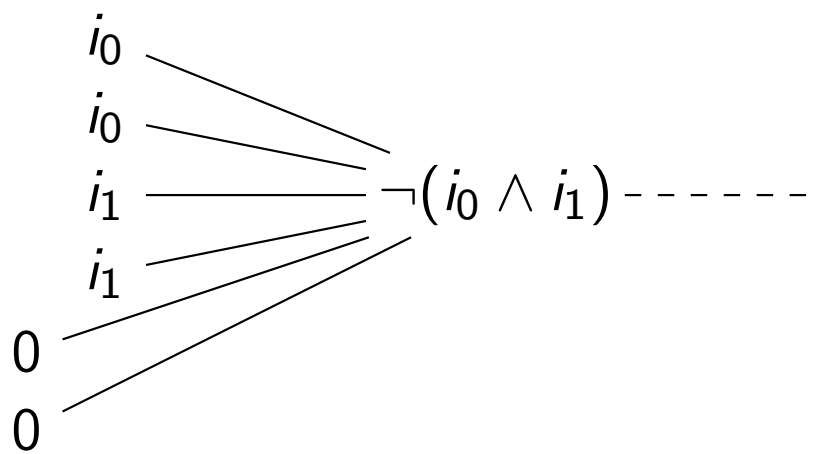
NAND gate

If all input vertices (labelled by i_0, i_1) are in the same partition (say 1) and the vertices labelled 0 are in the opposite one then the vertex labelled $\neg(i_0 \wedge i_1)$ must be in partition 0 (provided $\neg(i_0 \wedge i_1)$ has at most seven neighbors).



NAND gate

If all input vertices (labelled by i_0,i_1) are in the same partition (say 1) and the vertices labelled 0 are in the opposite one then the vertex labelled $\neg(i_0 \wedge i_1)$ must be in partition 0 (provided $\neg(i_0 \wedge i_1)$ has at most seven neighbors).



Differently from the NOT gate, such gate needs the "gadget" inputs (labelled 0).

Combinations of NAND and NOT

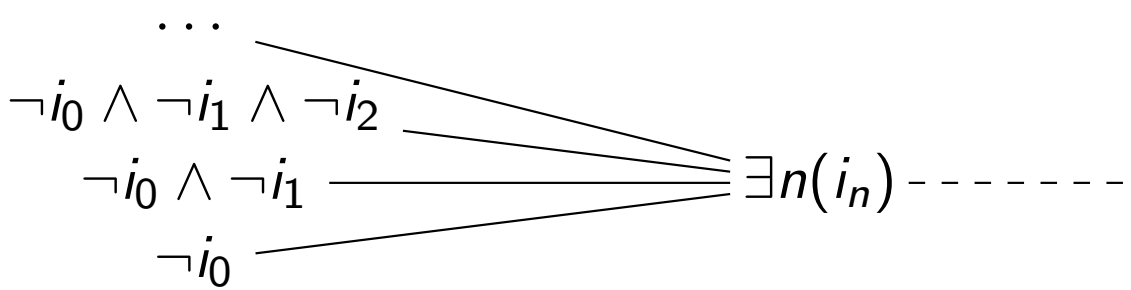
TECHNISCHE UNIVERSITÄT WIEN

E.g., the application of a NOT gate after a NAND gate gives us finite conjunctions (of each fixed length).

Combinations of NAND and NOT

E.g., the application of a NOT gate after a NAND gate gives us finite conjunctions (of each fixed length).
This may in turn be used to form a gate for $\exists$: Indeed:

- if i_n fails for all n, then $\neg i_0 \wedge \cdots \wedge \neg i_n$ holds for all n,
- if i_n holds for some n, then $\neg i_0 \wedge \cdots \wedge \neg i_n$ fails for cofinitely many n.

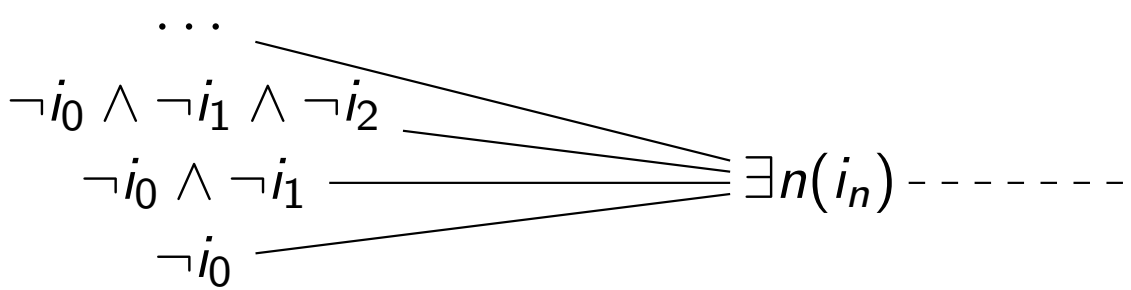


Combinations of NAND and NOT

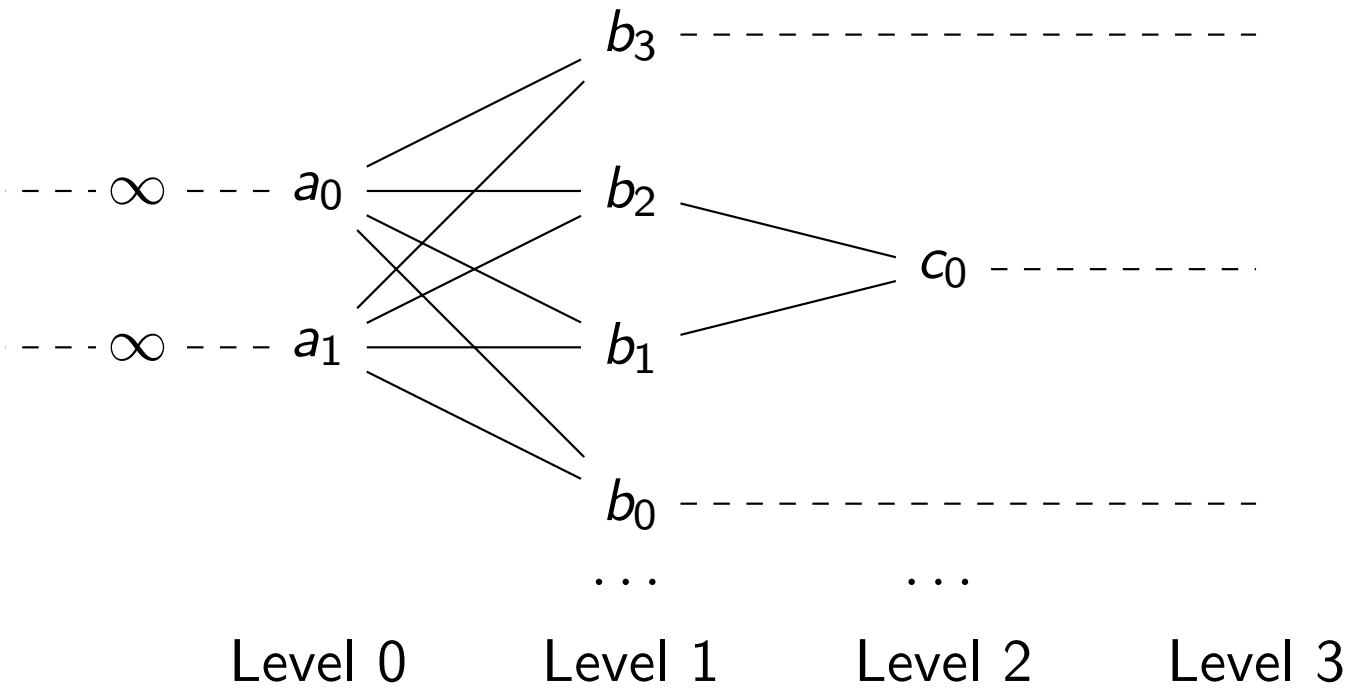
TECHNISCHE UNIVERSITÄT WIEN

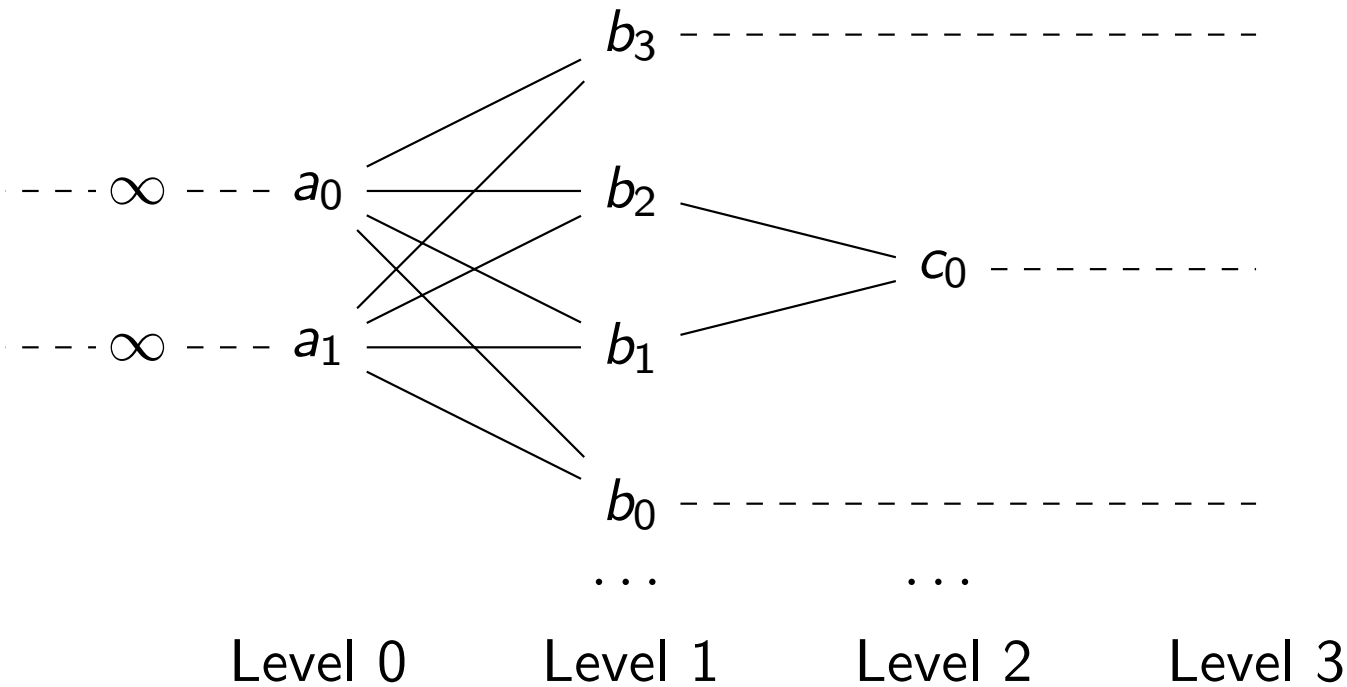
E.g., the application of a NOT gate after a NAND gate gives us finite conjunctions (of each fixed length).
This may in turn be used to form a gate for $\exists$: Indeed:

- if i_n fails for all n, then $\neg i_0 \wedge \cdots \wedge \neg i_n$ holds for all n,
- if i_n holds for some n, then $\neg i_0 \wedge \cdots \wedge \neg i_n$ fails for cofinitely many n.

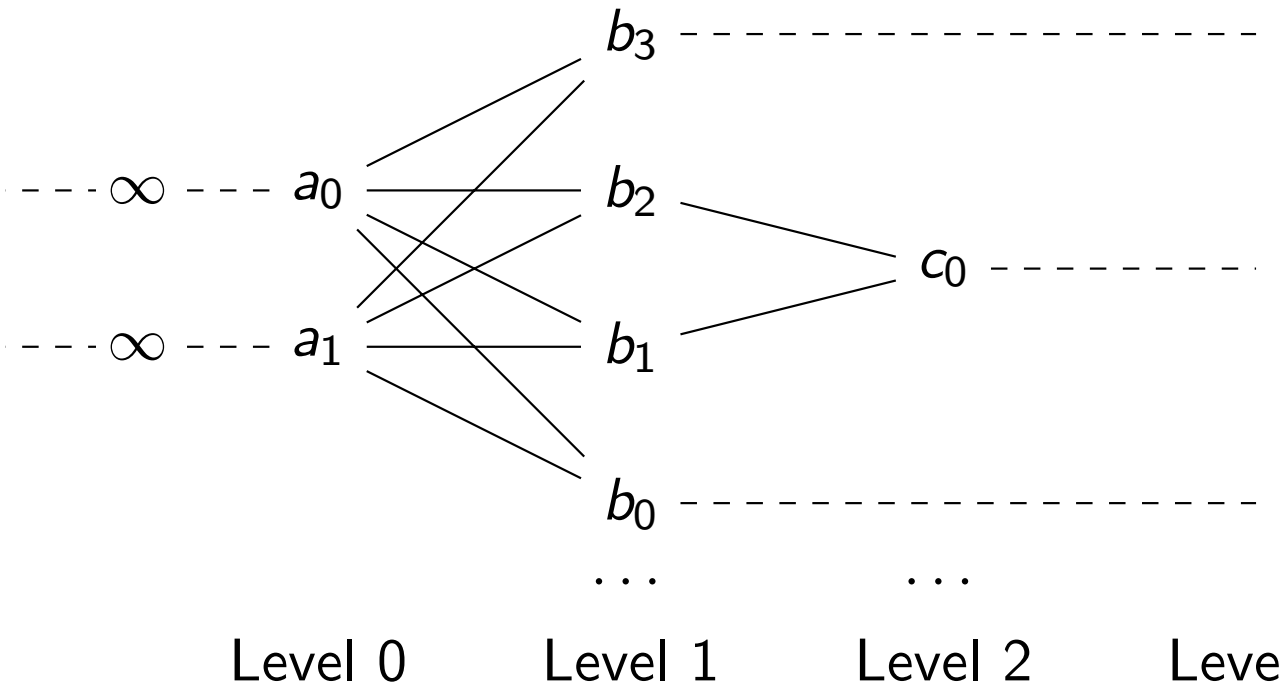


Now let's start with the construction of the graph.


b_3
∞
a_0
b_2
c_0
∞
a_1
b_1
b_0
...
...
Level 0
Level 1
Level 2
Level 3

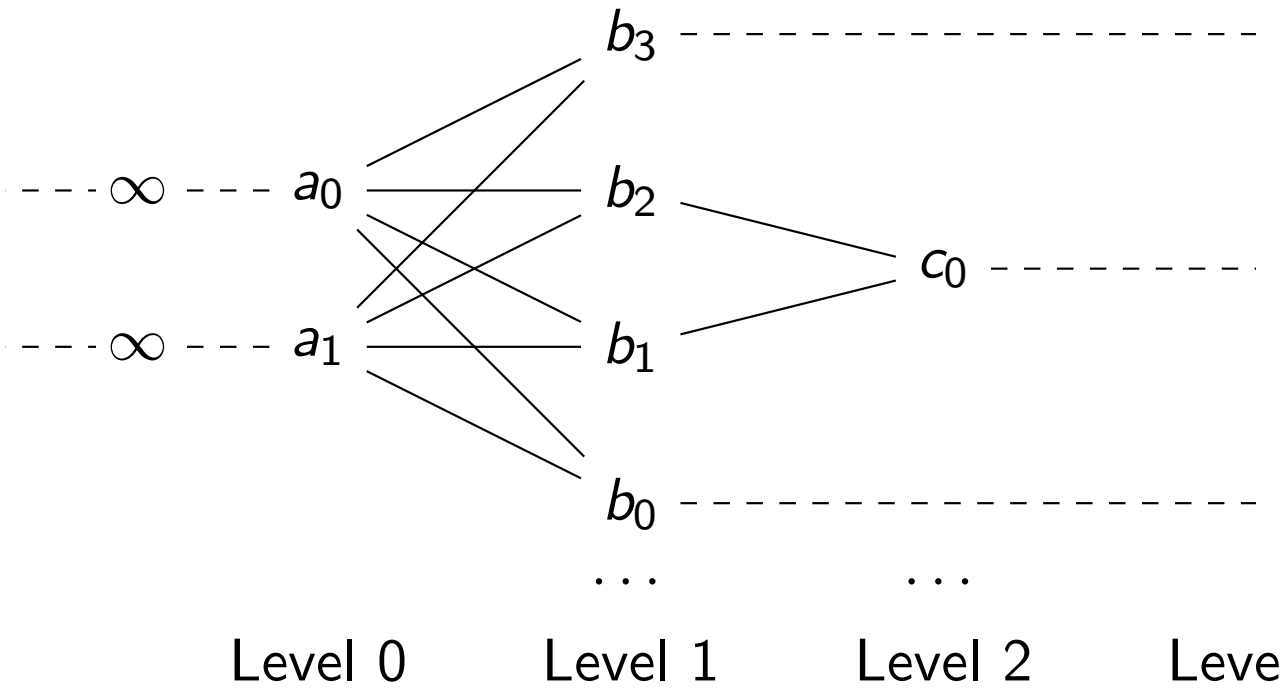


Assume a_0 and a_1 are in the same partition, say 0 (we postpone the discussion on how to ensure this for later).



Assume a_0 and a_1 are in the same partition, say 0 (we postpone the discussion on how to ensure this for later).

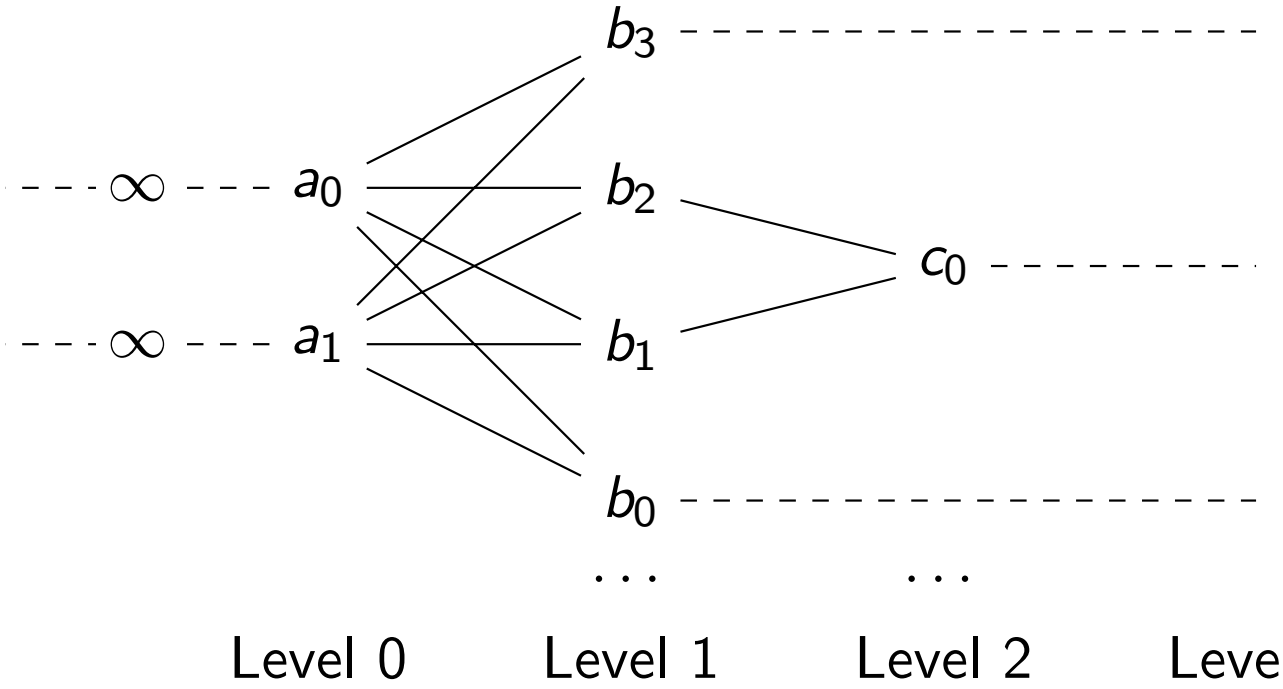
N.B.: the b_i's must be in partition 1 and the c_i's in partition 0.



Assume a_0 and a_1 are in the same partition, say 0 (we postpone the discussion on how to ensure this for later).

- c_i's are *cousins* of each other;
- b_{3n}'s are *cousins* of each other;
- c_i and b_{3n} are not *cousins*.

N.B. cousins are always on the same level and b_{3n+1} and b_{3n+2} are not cousins: they only force the c_i's to be in partition 0.



Assume a_0 and a_1 are in the same partition, say 0 (we postpone the discussion on how to ensure this for later).

Adding only one neighbor to c_i does not interfere with the partition in which c_i is.

$\Longrightarrow$ we will add to c_i a new neighbor, namely an output vertex of some logic gate.

The same reasoning can be applied to the b_{3n}'s.

To ensure that a_0 and a_1 are in the same partition, we consider instead a_0, a_1 and a_2: such vertices are at the level 0 of three graphs G_{01}, G_{02} and G_{12}.

To ensure that a_0 and a_1 are in the same partition, we consider instead a_0, a_1 and a_2: such vertices are at the level 0 of three graphs G_{01}, G_{02} and G_{12}.

N.B.:

- $G_{01} \cong G_{02} \cong G_{12}$ (and they contain a copy of the graph described above).

To ensure that a_0 and a_1 are in the same partition, we consider instead a_0, a_1 and a_2: such vertices are at the level 0 of three graphs G_{01}, G_{02} and G_{12}.

N.B.:

- $G_{01} \cong G_{02} \cong G_{12}$ (and they contain a copy of the graph described above).
- any two of these graphs share only one a_i for $i < 3$. So the only neighbors of a_0 are the b_i's of G_{01} and the $b'_i s$ of G_{02} (similarly for a_1 and a_2).

To ensure that a_0 and a_1 are in the same partition, we consider instead a_0, a_1 and a_2: such vertices are at the level 0 of three graphs G_{01}, G_{02} and G_{12}.

N.B.:

- $G_{01} \cong G_{02} \cong G_{12}$ (and they contain a copy of the graph described above).
- any two of these graphs share only one a_i for $i < 3$. So the only neighbors of a_0 are the b_i's of G_{01} and the $b'_i s$ of G_{02} (similarly for a_1 and a_2).

Regardless of how the construction continues (again the construction is identical for the 3 graphs), a_i, a_j for $i \neq j < 3$ must be in the same partition.

To ensure that a_0 and a_1 are in the same partition, we consider instead a_0, a_1 and a_2: such vertices are at the level 0 of three graphs G_{01}, G_{02} and G_{12}.

N.B.:

- $G_{01} \cong G_{02} \cong G_{12}$ (and they contain a copy of the graph described above).
- any two of these graphs share only one a_i for $i < 3$. So the only neighbors of a_0 are the b_i's of G_{01} and the $b'_i s$ of G_{02} (similarly for a_1 and a_2).

Regardless of how the construction continues (again the construction is identical for the 3 graphs), a_i, a_j for $i \neq j < 3$ must be in the same partition.

To compute the desired hyperarithemtic set we will only consider the u.p. of G_{ij} and forget about the other two.

We stop the description of the graph (for a slide) to recall what is a *Harrison order*[2].

[2] The proof that such an order exists is due to Harrison

We stop the description of the graph (for a slide) to recall what is a *Harrison order*[2].

Definition

A Harrison order is a computable linear order that is not well-founded but has no hyperarithemtic descending chain.

[2] The proof that such an order exists is due to Harrison

We stop the description of the graph (for a slide) to recall what is a *Harrison order*[2].

Definition

A Harrison order is a computable linear order that is not well-founded but has no hyperarithemtic descending chain.

N.B.: *all computable well-orderings are isomorphic to a proper initial segment of a Harrison ordering.*

[2] The proof that such an order exists is due to Harrison

We stop the description of the graph (for a slide) to recall what is a *Harrison order*[2].

Definition

A Harrison order is a computable linear order that is not well-founded but has no hyperarithemtic descending chain.

N.B.: *all computable well-orderings are isomorphic to a proper initial segment of a Harrison ordering.*
The order type of such an order is $\omega_1^{\mathsf{ck}}(1+\eta)+\alpha$ where:

- ω_1^{ck} is the first non computable ordinal;
- η is the order type of the rational numbers;
- $\alpha < \omega_1^{\mathsf{ck}}$ is a well-order.

[2] The proof that such an order exists is due to Harrison

We stop the description of the graph (for a slide) to recall what is a *Harrison order*[2].

Definition

A Harrison order is a computable linear order that is not well-founded but has no hyperarithemtic descending chain.

N.B.: *all computable well-orderings are isomorphic to a proper initial segment of a Harrison ordering.*
The order type of such an order is $\omega_1^{\mathsf{ck}}(1+\eta)+\alpha$ where:

- ω_1^{ck} is the first non computable ordinal;
- η is the order type of the rational numbers;
- $\alpha < \omega_1^{\mathsf{ck}}$ is a well-order.

Starting from the setup described before, our graph will consist of levels G_α indexed by elements α of a fixed Harrison order.

[2] The proof that such an order exists is due to Harrison

So far we have just described levels 0, 1, and 2. Notice that nodes at higher levels will be inputs and outputs of different gates.

So far we have just described levels 0, 1, and 2. Notice that nodes at higher levels will be inputs and outputs of different gates. Hence, every node v will have:

- a neighbor at a higher level (i.e., the output of some gate in which v is involved).
  - if v is at a *successor* level it will have 2 or 6 neighbors from lower levels (i.e., v is the output of some gate);
  - if v is at a *limit* level it will have infinitely many neighbors from lower levels;

So far we have just described levels 0, 1, and 2. Notice that nodes at higher levels will be inputs and outputs of different gates.
Hence, every node v will have:

- a neighbor at a higher level (i.e., the output of some gate in which v is involved).
  - if v is at a *successor* level it will have 2 or 6 neighbors from lower levels (i.e., v is the output of some gate);
  - if v is at a *limit* level it will have infinitely many neighbors from lower levels;

No node has neighbors on the same level.

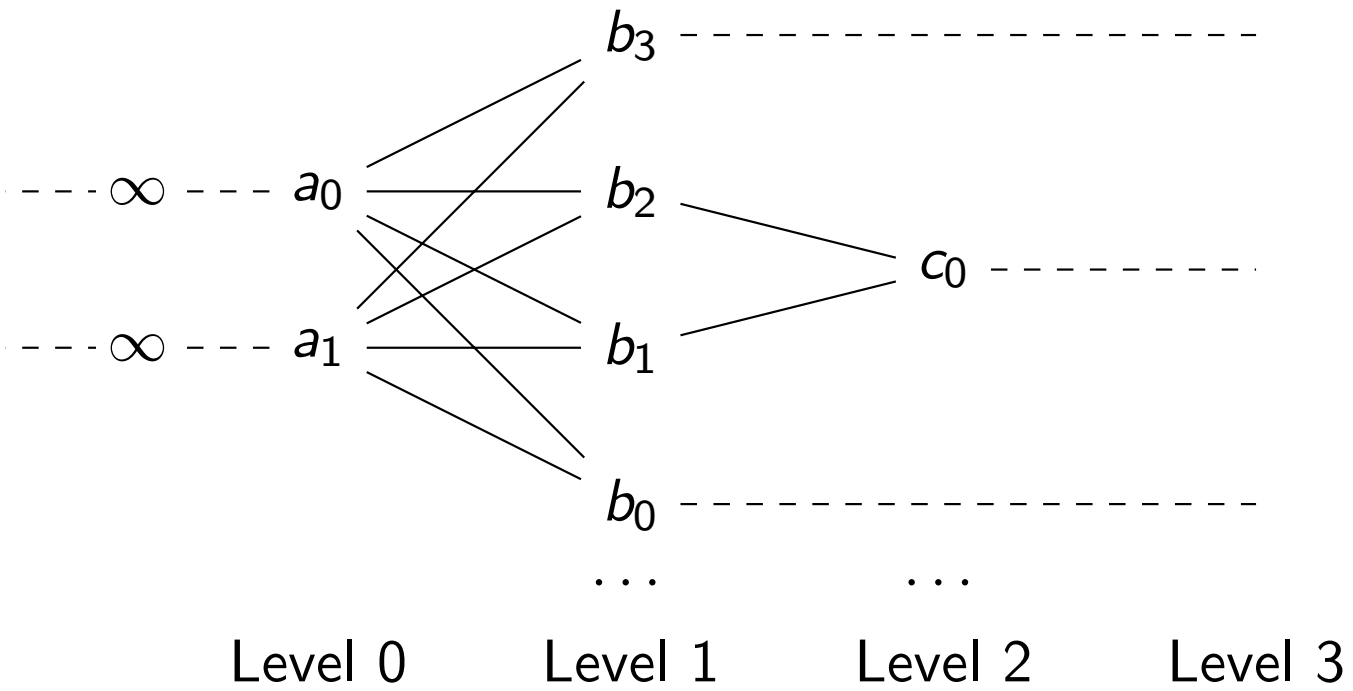
Recap

Wlog, assume that G_{01} is such that a_0 and a_1 are in the same partition.

Recap

Wlog, assume that G_{01} is such that a_0 and a_1 are in the same partition.

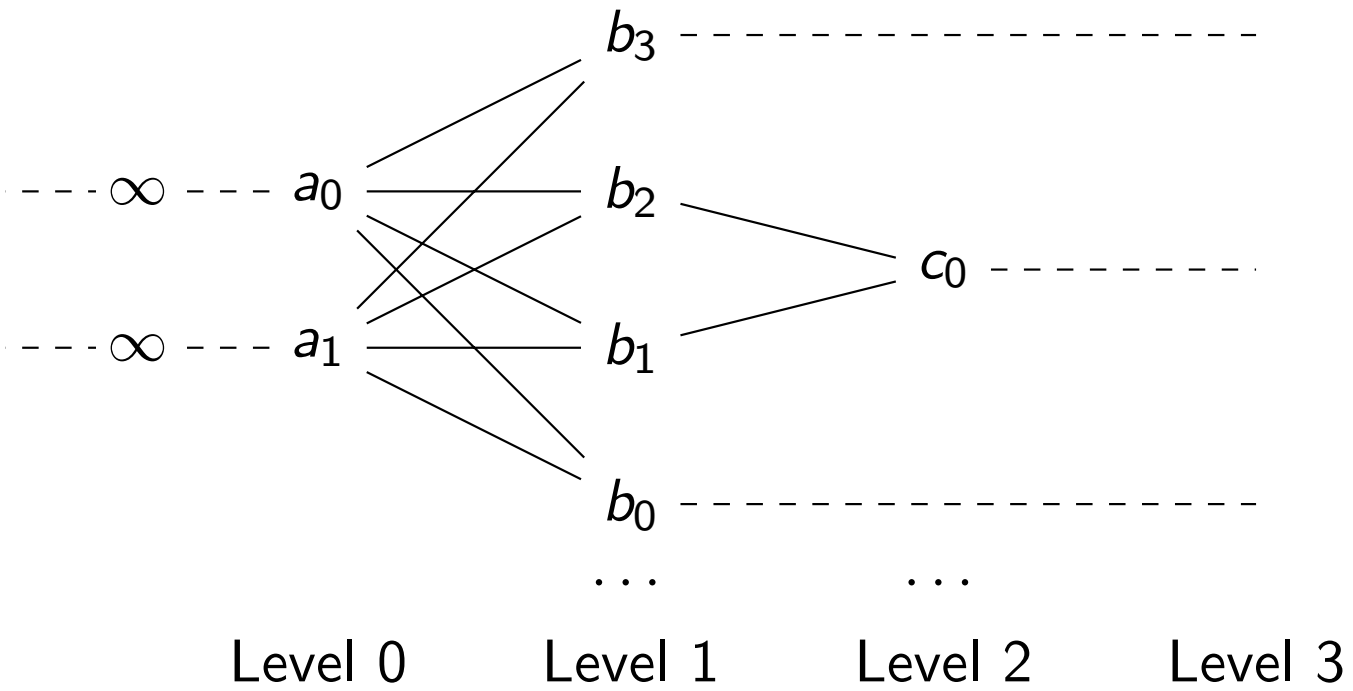
We have defined the first three levels



Recap

Wlog, assume that G_{01} is such that a_0 and a_1 are in the same partition.

We have defined the first three levels



We know what gates are.

Key properties of cousins: $\alpha + 1$

Wlog assume a_0 and a_1 are in partition 0.

Key properties of cousins: $\alpha + 1$

Wlog assume a_0 and a_1 are in partition 0.
What happens at successor stages:

Key properties of cousins: $\alpha + 1$

Wlog assume a_0 and a_1 are in partition 0.
What happens at successor stages:

- cousins represent the same formula (notice that there are infinitely many cousins);

Key properties of cousins: $\alpha + 1$

Wlog assume a_0 and a_1 are in partition 0.
What happens at successor stages:

- cousins represent the same formula (notice that there are infinitely many cousins);
- Such formulae "approximate" the members in the desired hyperarithmetic set (recall how the existential gate worked: wait for the next slide).

Key properties of cousins: $\alpha + 1$

Wlog assume a_0 and a_1 are in partition 0.
What happens at successor stages:

- cousins represent the same formula (notice that there are infinitely many cousins);
- Such formulae "approximate" the members in the desired hyperarithmetic set (recall how the existential gate worked: wait for the next slide).
- We need infinitely many because we may want to use that node/formula as an input for different gates. Indeed, remember that any node is both an input and an output for different gates. Using the same node as an input for different gates may "break" the gate of which it was an output.

Key properties of cousins: α limit

TECHNISCHE UNIVERSITÄT WIEN

- We have infinitely many sets of cousins. The crucial part of this construction is that cousins must be in the same partition.

Key properties of cousins: α limit

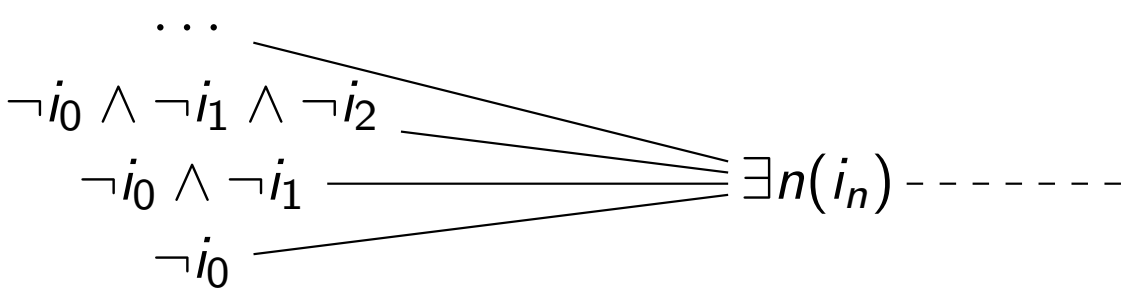
- We have infinitely many sets of cousins. The crucial part of this construction is that cousins must be in the same partition.
- Any vertex in a set of cousins represents an element for which we want to know whether it is in the desired set.

Key properties of cousins: α limit

- We have infinitely many sets of cousins. The crucial part of this construction is that cousins must be in the same partition.
- Any vertex in a set of cousins represents an element for which we want to know whether it is in the desired set.
- The construction will ensure that i is in partition 1 if $\varphi_i^A(i)\downarrow$ where A is the partition defined so far.

Key properties of cousins: α limit

- We have infinitely many sets of cousins. The crucial part of this construction is that cousins must be in the same partition.
- Any vertex in a set of cousins represents an element for which we want to know whether it is in the desired set.
- The construction will ensure that i is in partition 1 if $\varphi_i^A(i)\downarrow$ where A is the partition defined so far.
  How? Recall that vertices at limit levels have infinitely many neighbors coming from lower levels. These infinitely many nodes represent formulae approximating the fact that i is in the set or not: it will be in the set iff cofinitely many of its neighbors from lower levels are in partition 0.



To "conclude" one takes the unfriendly partition A of G_{01} (the one for which a_0 and a_1 are in the same partition).

To "conclude" one takes the unfriendly partition A of G_{01} (the one for which a_0 and a_1 are in the same partition).
This requires some computation but, given a computable ordinal α, let $\beta = \omega(\alpha + 1)$, and let A_β be the restriction of A assigning $A(x)$ for every node at level $< \beta$ and undefined otherwise.

To "conclude" one takes the unfriendly partition A of G_{01} (the one for which a_0 and a_1 are in the same partition).
This requires some computation but, given a computable ordinal α, let $\beta = \omega(\alpha + 1)$, and let A_β be the restriction of A assigning $A(x)$ for every node at level $< \beta$ and undefined otherwise.
We then show that A_β (and hence A) computes the α-th jump of a computable set.

To "conclude" one takes the unfriendly partition A of G_{01} (the one for which a_0 and a_1 are in the same partition).
This requires some computation but, given a computable ordinal α, let $\beta = \omega(\alpha + 1)$, and let A_β be the restriction of A assigning $A(x)$ for every node at level $< \beta$ and undefined otherwise.
We then show that A_β (and hence A) computes the α-th jump of a computable set.
The key step for proving this is to show that:

- for each level the inputs depend only on strictly lower levels;
- cousins have their neighbors below also being cousins.

To "conclude" one takes the unfriendly partition A of G_{01} (the one for which a_0 and a_1 are in the same partition).
This requires some computation but, given a computable ordinal α, let $\beta = \omega(\alpha + 1)$, and let A_β be the restriction of A assigning $A(x)$ for every node at level $< \beta$ and undefined otherwise.
We then show that A_β (and hence A) computes the α-th jump of a computable set.
The key step for proving this is to show that:

- for each level the inputs depend only on strictly lower levels;
- cousins have their neighbors below also being cousins.

Done?

Remember: it is not known whether every countable graph admits an unfriendly partition.

Remember: it is not known whether every countable graph admits an unfriendly partition.
Does our graph admit an unfriendly partition?

Remember: it is not known whether every countable graph admits an unfriendly partition.
Does our graph admit an unfriendly partition?

Theorem

No graph like the one described above has a subdivision of a 5*-clique as a subgraph. In particular, by Berger's theorem, it has an unfriendly partition.*

An adaptation of the graph above allows us to prove the following (recall that by Bruhn and Diestel's theorem, any rayless graph admits an unfriendly partition).

An adaptation of the graph above allows us to prove the following (recall that by Bruhn and Diestel's theorem, any rayless graph admits an unfriendly partition).

Theorem

For every hyperarithmetic set A there is a rayless graph such that every unfriendly partition B of that graph is Turing above A.

An adaptation of the graph above allows us to prove the following (recall that by Bruhn and Diestel's theorem, any rayless graph admits an unfriendly partition).

Theorem

For every hyperarithmetic set A there is a rayless graph such that every unfriendly partition B of that graph is Turing above A.

To do so instead of a Harrison ordering, we repeat the "same" construction but on some computable ordinal α.

An adaptation of the graph above allows us to prove the following (recall that by Bruhn and Diestel's theorem, any rayless graph admits an unfriendly partition).

Theorem

For every hyperarithmetic set A there is a rayless graph such that every unfriendly partition B of that graph is Turing above A.

To do so instead of a Harrison ordering, we repeat the "same" construction but on some computable ordinal α.

Theorem

There is an "easiest" u.p. of this graph which is Turing reducible to all other u.p.: its Turing degree is between the α-th and $\alpha + 2$-nd jump of some computable set.

A universal graph

A "universal graph" for u.p.

Let $(\varphi_i)_{i\in\mathbb{N}}$ be a computable enumeration of all partial computable functions computing graphs.

A "universal graph" for u.p.

Let $(\varphi_i)_{i\in\mathbb{N}}$ be a computable enumeration of all partial computable functions computing graphs.
Let

$$V(G) = \{(e,k) : (\forall i,j \leq k)(\varphi_e(i,j)\downarrow \in \{0,1\})\}$$

and let

$$\big((e_i,k_i),(e_j,k_j)\big) \in E(G) \iff e_i = e_j \wedge k_i \neq k_j \wedge \varphi_{e_i}(k_i,k_j) = 1.$$

Notice that G is computable.

Theorem

Every computable graph is isomorphic to a component of G via a computable isomorphism. Furthermore, every u.p. of G is also a u.p. of every computable graph.

Theorem

Every computable graph is isomorphic to a component of G via a computable isomorphism. Furthermore, every u.p. of G is also a u.p. of every computable graph.

To prove the theorem, given a graph H_e identified by φ_e, map any $v \in V(H_e)$ to the unique $f(v)$ s.t. $e_{f(v)} = e$ and $k_{f(v)} = v$.

Known results

TECHNISCHE UNIVERSITÄT WIEN

Theorem (Gandy basis theorem)

Every nonempty Σ^1_1 class has a member f such that $\omega_1^f = \omega_1^{\mathsf{ck}}$.

Known results

TECHNISCHE UNIVERSITÄT WIEN

Theorem (Gandy basis theorem)

Every nonempty Σ^1_1 class has a member f such that $\omega_1^f = \omega_1^{\mathsf{ck}}$.

The following definitions are due to Calvert, Franklin and Turetsky.

Theorem (High for isomorphism/paths)

We call a degree $\mathbf{d}$ *high for isomorphism if for any two computable structures $\mathcal{M}$ and $\mathcal{N}$ with $\mathcal{M} \cong \mathcal{N}$, there is a* $\mathbf{d}$*-computable isomorphism from $\mathcal{M}$ to $\mathcal{N}$.*
We call a degree $\mathbf{d}$ *high for paths if for any ill-founded tree, the degree* $\mathbf{d}$ *computes a path in it.*

Known results

TECHNISCHE UNIVERSITÄT WIEN

Theorem (Gandy basis theorem)

Every nonempty Σ^1_1 class has a member f such that $\omega_1^f = \omega_1^{\mathsf{ck}}$.

The following definitions are due to Calvert, Franklin and Turetsky.

Theorem (High for isomorphism/paths)

We call a degree $\mathbf{d}$ *high for isomorphism if for any two computable structures $\mathcal{M}$ and $\mathcal{N}$ with $\mathcal{M} \cong \mathcal{N}$, there is a* $\mathbf{d}$*-computable isomorphism from $\mathcal{M}$ to $\mathcal{N}$.*
We call a degree $\mathbf{d}$ *high for paths if for any ill-founded tree, the degree* $\mathbf{d}$ *computes a path in it.*

The two notions coincide.

Theorem

Suppose that every computable graph has an u.p.: then there is no function f mapping a computable tree T to a computable graph G such that every u.p. of G computes some $p \in [T]$.

Theorem

Suppose that every computable graph has an u.p.: then there is no function f mapping a computable tree T to a computable graph G such that every u.p. of G computes some $p \in [T]$.

Sketch.

Suppose such an f exists: then any u.p. A of the universal graph computes a path in every computable tree and hence A is high for isomorphism/paths.

Theorem

Suppose that every computable graph has an u.p.: then there is no function f mapping a computable tree T to a computable graph G such that every u.p. of G computes some $p \in [T]$.

Sketch.

Suppose such an f exists: then any u.p. A of the universal graph computes a path in every computable tree and hence A is high for isomorphism/paths.
According to results of Calvert, Franklin and Turetsky $A''' \geq \mathcal{O}$ (where $\mathcal{O}$ is Kleene's $\mathcal{O}$).

Theorem

Suppose that every computable graph has an u.p.: then there is no function f mapping a computable tree T to a computable graph G such that every u.p. of G computes some $p \in [T]$.

Sketch.

Suppose such an f exists: then any u.p. A of the universal graph computes a path in every computable tree and hence A is high for isomorphism/paths.
According to results of Calvert, Franklin and Turetsky $A''' \geq \mathcal{O}$ (where $\mathcal{O}$ is Kleene's $\mathcal{O}$).
So A should compute a well-ordered isomorphic to ω_1^{ck}. But, by Gandy basis theorem, one can choose A being low for ω_1^{ck} (i.e., A computes only computable ordinals). □

We also considered the problem of *strict unfriendly partitions*.

We also considered the problem of *strict unfriendly partitions*. They behave quite differently: e.g., C_3 does not admit a strong unfriendly partition.

We also considered the problem of *strict unfriendly partitions*.
They behave quite differently: e.g., C_3 does not admit a strong unfriendly partition.
We have some results in this direction (e.g., the set of indices of computable graphs with an unfriendly partition is as complicated as possible), but we stop here.

We also considered the problem of *strict unfriendly partitions*.
They behave quite differently: e.g., C_3 does not admit a strong unfriendly partition.
We have some results in this direction (e.g., the set of indices of computable graphs with an unfriendly partition is as complicated as possible), but we stop here.

Thanks!