On the coinductive nature of centralizers

Charles Grellois

LIAFA & PPS — Université Paris 7

January 14th, 2015

Commutation of words

Consider the word equation

$$x \cdot w \; = \; w \cdot x$$

A solution is a word x which is a prefix and a suffix of w.

A well-known fact: if

$$w \; = \; u^n$$

(with u minimal in some sense) then the set of solutions is

$$\{ \, u^m \mid m \in \mathbb{N} \, \}$$

Commutation of words

Consider the word equation

$$x \cdot w \;=\; w \cdot x$$

A solution is a word x which is a prefix and a suffix of w.

A well-known fact: if

$$w \;=\; u^n$$

(with u minimal in some sense) then the set of solutions is

$$\{\, u^m \mid m \in \mathbb{N} \,\}$$

Commutation of words

Example:

$$x \cdot abab \;=\; abab \cdot x$$

The set of solutions is $\{\,(ab)^m \mid m \in \mathbb{N}\,\}$.

There are generalizations of this result to polynomials or formal series (whose variables do not commute).

But what about commutation of languages ?

Commutation of words

Example:

$$x \cdot abab \;=\; abab \cdot x$$

The set of solutions is $\{\, (ab)^m \mid m \in \mathbb{N} \,\}$.

There are generalizations of this result to polynomials or formal series (whose variables do not commute).

But what about commutation of languages ?

Centralizers

Consider the language commutation equation

$$X \cdot L \;=\; L \cdot X \qquad (1)$$

What does it mean ?

If X is a solution, then for every word $w \in L$ and every word $x \in X$ the concatenation

$$w \cdot x$$

can be factored as

$$x' \cdot w'$$

with $x' \in X$ and $w' \in L$.

Centralizers

Consider the language commutation equation

$$X \cdot L \; = \; L \cdot X \tag{1}$$

What does it mean ?

If X is a solution, then for every word $w \in L$ and every word $x \in X$ the concatenation

$$w \cdot x$$

can be factored as

$$x' \cdot w'$$

with $x' \in X$ and $w' \in L$.

Centralizers

Consider the language commutation equation

$$X \cdot L \;=\; L \cdot X \qquad (1)$$

What does it mean ?

If X is a solution, then for every word $w \in L$ and every word $x \in X$ the concatenation

$$w \cdot x$$

can be factored as

$$x' \cdot w'$$

with $x' \in X$ and $w' \in L$.

Centralizers: an example

Consider (from Choffrut-Karhumäki-Ollinger):

$$L \quad = \quad \{\, a,\ a^3,\ b,\ ba,\ ab,\ aba \,\}$$

then a solution of the commutation equation is

$$X \quad = \quad L \cup \{\, a^2 \,\}$$

A few verifications:

$$a^2 \cdot b \quad = \quad a \cdot ab$$

$$a^2 \cdot aba \quad = \quad a^3 \cdot ba$$

$$aba \cdot a^2 \quad = \quad ab \cdot a^3$$

Centralizers: an example

Consider (from Choffrut-Karhumäki-Ollinger):

$$L \quad = \quad \{\, a,\ a^3,\ b,\ ba,\ ab,\ aba \,\}$$

then a solution of the commutation equation is

$$X \quad = \quad L \cup \{\, a^2 \,\}$$

A few verifications:

$$a^2 \cdot b \quad = \quad a \cdot ab$$

$$a^2 \cdot aba \quad = \quad a^3 \cdot ba$$

$$aba \cdot a^2 \quad = \quad ab \cdot a^3$$

Centralizers: an example

Consider (from Choffrut-Karhumäki-Ollinger):

$$L \quad = \quad \{\, a,\ a^3,\ b,\ ba,\ ab,\ aba \,\}$$

then a solution of the commutation equation is

$$X \quad = \quad L \cup \{\, a^2 \,\}$$

A few verifications:

$$a^2 \cdot b \quad = \quad a \cdot ab$$

$$a^2 \cdot aba \quad = \quad a^3 \cdot ba$$

$$aba \cdot a^2 \quad = \quad ab \cdot a^3$$

Centralizers: an example

Consider (from Choffrut-Karhumäki-Ollinger):

$$L \quad = \quad \{\, a,\ a^3,\ b,\ ba,\ ab,\ aba \,\}$$

then a solution of the commutation equation is

$$X \quad = \quad L \cup \{\, a^2 \,\}$$

A few verifications:

$$a^2 \cdot b \quad = \quad a \cdot ab$$

$$a^2 \cdot aba \quad = \quad a^3 \cdot ba$$

$$aba \cdot a^2 \quad = \quad ab \cdot a^3$$

Centralizers

$$X \cdot L = L \cdot X$$

Note that this equation always have solutions, among which obviously:

$$\emptyset \quad \text{and} \quad \{\epsilon\} \quad \text{and} \quad L \quad \text{and} \quad L^* \quad \text{and} \quad L^+$$

Note that the union of two solutions is a solution as well: if

$$X_1 \cdot L = L \cdot X_1 \quad \text{and} \quad X_2 \cdot L = L \cdot X_2$$

then

$$(X_1 \cup X_2) \cdot L = L \cdot (X_1 \cup X_2)$$

The same holds for the intersection of solutions.

Centralizers

$$X \cdot L \; = \; L \cdot X$$

Note that this equation always have solutions, among which obviously:

$$\emptyset \quad \text{and} \quad \{\epsilon\} \quad \text{and} \quad L \quad \text{and} \quad L^* \quad \text{and} \quad L^+$$

Note that the union of two solutions is a solution as well: if

$$X_1 \cdot L \; = \; L \cdot X_1 \quad \text{and} \quad X_2 \cdot L \; = \; L \cdot X_2$$

then

$$(X_1 \cup X_2) \cdot L \; = \; L \cdot (X_1 \cup X_2)$$

The same holds for the intersection of solutions.

Centralizers

$$X \cdot L \; = \; L \cdot X$$

This equation has a greatest solution, defined as the union of all solutions.

We get back to this later.

This greatest solution is defined as the centralizer of the language L, and is denoted $\mathcal{C}(L)$.

Note that it contains ϵ. We denote $\mathcal{C}_+(L)$ the largest solution not containing ϵ.

Centralizers

Note the following approximation results:

$$L^* \quad \subseteq \quad \mathcal{C}(L) \quad \subseteq \quad Pref(L^*) \cap Suff(L^*)$$

$$L^+ \quad \subseteq \quad \mathcal{C}_+(L) \quad \subseteq \quad Pref_+(L^+) \cap Suff_+(L^+)$$

Centralizers, game-theoretically

There is a natural interactive interpretation of centralizers. Consider a two-player game, starting on some word

$$u \in A^*$$

which is a position owned by Adam (the "attacker").

Adam then adds a word $w \in L$ to a side of the word, and reaches Eve's position:

$$u \cdot w \ \in \ A^* \cdot L$$

Eve answers by removing a word $v \in L$ at the other side of the word and reaches

$$v^{-1} u\, w \ \in \ L^{-1} \cdot A^* \cdot L \ \subseteq \ A^*$$

which belongs to Adam.

Centralizers, game-theoretically

There is a natural interactive interpretation of centralizers. Consider a two-player game, starting on some word

$$u \in A^*$$

which is a position owned by Adam (the "attacker").

Adam then adds a word $w \in L$ to a side of the word, and reaches Eve's position:

$$u \cdot w \; \in \; A^* \cdot L$$

Eve answers by removing a word $v \in L$ at the other side of the word and reaches

$$v^{-1} \, u \, w \; \in \; L^{-1} \cdot A^* \cdot L \quad \subseteq \quad A^*$$

which belongs to Adam.

Centralizers, game-theoretically

There is a natural interactive interpretation of centralizers. Consider a two-player game, starting on some word

$$u \in A^*$$

which is a position owned by Adam (the “attacker”).

Adam then adds a word $w \in L$ to a side of the word, and reaches Eve’s position:

$$u \cdot w \ \in \ A^* \cdot L$$

Eve answers by removing a word $v \in L$ at the other side of the word and reaches

$$v^{-1}\, u\, w \ \in \ L^{-1} \cdot A^* \cdot L \ \subseteq \ A^*$$

which belongs to Adam.

Centralizers, game-theoretically

If Eve can not answer at some point, she looses. If she can play forever, Adam looses.

We have that

$$u \in \mathcal{C}(L) \quad \text{iff Eve can win every play starting from it}$$

A more elaborate game-theoretic account of centralizers was given by Jeandel and Ollinger (2008).

Centralizers, game-theoretically

If Eve can not answer at some point, she looses. If she can play forever, Adam looses.

We have that

$$u \in \mathcal{C}(L) \quad \text{iff Eve can win every play starting from it}$$

A more elaborate game-theoretic account of centralizers was given by Jeandel and Ollinger (2008).

Centralizers

Conway's problem: if L is regular, what can be said of $\mathcal{C}(L)$?

Open problem for a long time; it seems that people expected some regularity. Until:

Theorem (Kunc 2006)

- *There exists a regular, star-free language L such that $\mathcal{C}(L)$, $\mathcal{C}_+(L)$ and $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$ are not recursively enumerable.*
- *There exists a finite language L such that $\mathcal{C}(L)$ and $\mathcal{C}_+(L)$ are not recursively enumerable.*

In the second point, nothing is said about $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$.

Centralizers

Conway's problem: if L is regular, what can be said of $\mathcal{C}(L)$?

Open problem for a long time; it seems that people expected some regularity. Until:

Theorem (Kunc 2006)

- *There exists a regular, star-free language L such that* $\mathcal{C}(L)$, $\mathcal{C}_+(L)$ *and* $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$ *are not recursively enumerable.*
- *There exists a finite language L such that* $\mathcal{C}(L)$ *and* $\mathcal{C}_+(L)$ *are not recursively enumerable.*

In the second point, nothing is said about $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$.

Centralizers

Theorem (Kunc 2006)

- *There exists a regular, star-free language* L *such that* $\mathcal{C}(L)$, $\mathcal{C}_+(L)$ *and* $\mathcal{C}(L) \setminus \mathcal{C}_+(L)$ *are not recursively enumerable.*
- *There exists a finite language* L *such that* $\mathcal{C}(L)$ *and* $\mathcal{C}_+(L)$ *are not recursively enumerable.*

In this talk:

- we describe the main elements of Kunc's proof,
- we rephrase it in an alternate model of computation,
- and we reveal an important key for understanding this Theorem: centralizers are coinductive.

Elements of Kunc's proof

The first step is to encode the behaviour of a Turing-complete machine in a centralizer.

However, we can only build L...

The point is to design L satisfying two dual purposes:

- by adding words for simulating the machine's transitions,
- and by adding words for restricting the centralizer (in particular, it should only "simulate" the transitions defined in the machine)

Elements of Kunc's proof

The first step is to encode the behaviour of a Turing-complete machine in a centralizer.

However, we can only build L...

The point is to design L satisfying two dual purposes:

- by adding words for simulating the machine's transitions,
- and by adding words for restricting the centralizer (in particular, it should only "simulate" the transitions defined in the machine)

Encoding Minsky machines

In his original proof, Kunc encodes Minsky machines, that is machines with:

- two counters storing integers,
- a finite number of states,
- increase/decrease operations over counters,
- and a conditional operation (does a counter store 0?)

and which are Turing-complete.

Encoding Minsky machines

A typical configuration is

$$(q,\ i,\ j) \quad \in \quad Q \times \mathbb{N} \times \mathbb{N}$$

Each state affects the counters of the machine.

If q is a state increasing the first counter and going to q',
then a transition of the machine corresponds to

$$(q,\ i,\ j) \quad \longrightarrow \quad (q',\ i+1,\ j)$$

Encoding Minsky machines

A typical configuration is

$$(q,\, i,\, j) \quad \in \quad Q \times \mathbb{N} \times \mathbb{N}$$

Each state affects the counters of the machine.

If q is a state increasing the first counter and going to q', then a transition of the machine corresponds to

$$(q,\, i,\, j) \quad \longrightarrow \quad (q',\, i+1,\, j)$$

Encoding Minsky machines

Kunc designs L such that $\mathcal{C}(L)$ contains the word

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2$$

which is the encoding of the configuration

$$(q,\ n,\ m)$$

How do the encodings of configurations relate ?

Encoding Minsky machines

Kunc designs L such that $\mathcal{C}(L)$ contains the word

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2$$

which is the encoding of the configuration

$$(q,\ n,\ m)$$

How do the encodings of configurations relate ?

A key proposition on centralizers

Proposition

Given u, $v \in L$, suppose that

$$u \cdot x = y \cdot v \tag{2}$$

Then

$$x \in \mathcal{C}(L) \iff y \in \mathcal{C}(L)$$

Note that (2) means that x and y can commute with one word of L, and that this assumption is one-sided.

Remark that this proposition is the core of the game-theoretic interpretation of centralizers.

A key proposition on centralizers

Proposition

Given u, $v \in L$, suppose that

$$u \cdot x \;=\; y \cdot v \tag{2}$$

Then

$$x \in \mathcal{C}(L) \quad \Longleftrightarrow \quad y \in \mathcal{C}(L)$$

Note that (2) means that x and y can commute with one word of L, and that this assumption is one-sided.

Remark that this proposition is the core of the game-theoretic interpretation of centralizers.

A key proposition on centralizers

Proposition

Given u, $v \in L$, suppose that

$$u \cdot x = y \cdot v \tag{2}$$

Then

$$x \in \mathcal{C}(L) \iff y \in \mathcal{C}(L)$$

Note that (2) means that x and y can commute with one word of L, and that this assumption is one-sided.

Remark that this proposition is the core of the game-theoretic interpretation of centralizers.

Encoding Minsky machines

To simulate the increasing transition

$$(q,\, i,\, j) \quad \longrightarrow \quad (q',\, i+1,\, j)$$

in $\mathcal{C}(L)$, Kunc proceeds by showing that their encodings relate as

$$\begin{array}{rll} & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\ \iff & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L) \end{array}$$

using the previous Proposition.

Encoding Minsky machines

Indeed, start from

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in A^*$$

Then

$$g_q \cdot a \cdot a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in L \cdot A^*$$

And

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \cdot \widehat{d}_q \in A^* \cdot L$$

So that, by the Proposition,

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in \mathcal{C}(L)$$

Encoding Minsky machines

Indeed, start from

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in A^*$$

Then

$$g_q \cdot a \cdot a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in L \cdot A^*$$

And

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \cdot \widehat{d}_q \in A^* \cdot L$$

So that, by the Proposition,

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in \mathcal{C}(L)$$

Encoding Minsky machines

Indeed, start from

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in A^*$$

Then

$$g_q \cdot a \cdot a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in L \cdot A^*$$

And

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \cdot \widehat{d}_q \in A^* \cdot L$$

So that, by the Proposition,

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in \mathcal{C}(L)$$

Encoding Minsky machines

Indeed, start from

$$a^{n+1} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q^2 \in A^*$$

Then

$$g_q \cdot a \cdot a^{n+1} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q^2 \in L \cdot A^*$$

And

$$g_q \cdot a^{n+2} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q \cdot \widehat{d}_q \in A^* \cdot L$$

So that, by the Proposition,

$$a^{n+1} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q^2 \in \mathcal{C}(L) \iff g_q a^{n+2} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^*$$

we obtain

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in L \cdot A^*$$

and

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^* \cdot L$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^*$$

we obtain

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in L \cdot A^*$$

and

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^* \cdot L$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^*$$

we obtain

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in L \cdot A^*$$

and

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^* \cdot L$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^*$$

we obtain

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in L \cdot A^*$$

and

$$e_q \cdot f_q \cdot g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q \in A^* \cdot L$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \in A^*$$

we obtain

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in A^* \cdot L$$

and

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in L \cdot A^*$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}\ \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$e_q f_q g_q a^{n+2} \; b \; \widehat{a}^{m+1} \; \in A^*$$

we obtain

$$e_q f_q g_q a^{n+2} \; b \; \widehat{a}^{m+1} \widehat{d}_{q'} \in A^* \cdot L$$

and

$$e_q f_q g_q a^{n+2} \; b \; \widehat{a}^{m+1} \widehat{d}_{q'} \in L \cdot A^*$$

So that

$$a^{n+1} \; b \; \widehat{a}^{m+1} \; \widehat{d}_q^{\,2} \in \mathcal{C}(L) \iff f_q g_q a^{n+2} \; b \; \widehat{a}^{m+1} \widehat{d}_{q'} \; \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \in A^*$$

we obtain

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in A^* \cdot L$$

and

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in L \cdot A^*$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}\ \in \mathcal{C}(L)$$

Encoding Minsky machines

Then, from

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1}\ \in A^*$$

we obtain

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in A^* \cdot L$$

and

$$e_q f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \in L \cdot A^*$$

So that

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}\ \in \mathcal{C}(L)$$

Encoding Minsky machines

Finally, from

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}\ \in A^*$$

we obtain

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'}\ \in A^* \cdot L$$

and

$$f_q g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'}\ \in L \cdot A^*$$

So that we related two configurations of the machine:

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}^2\ \in \mathcal{C}(L)$$

Note that L is designed such that only valid transitions could be simulated in $\mathcal{C}(L)$.

Encoding Minsky machines

Finally, from

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \ \in A^*$$

we obtain

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in A^* \cdot L$$

and

$$f_q g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in L \cdot A^*$$

So that we related two configurations of the machine:

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}^2 \ \in \mathcal{C}(L)$$

Note that L is designed such that only valid transitions could be simulated in $\mathcal{C}(L)$.

Encoding Minsky machines

Finally, from

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \ \in A^*$$

we obtain

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in A^* \cdot L$$

and

$$f_q g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in L \cdot A^*$$

So that we related two configurations of the machine:

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}^2 \ \in \mathcal{C}(L)$$

Note that L is designed such that only valid transitions could be simulated in $\mathcal{C}(L)$.

Encoding Minsky machines

Finally, from

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \ \in A^*$$

we obtain

$$f_q g_q a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in A^* \cdot L$$

and

$$f_q g_q \cdot a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in L \cdot A^*$$

So that we related two configurations of the machine:

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 \in \mathcal{C}(L) \iff a^{n+2}\ b\ \widehat{a}^{m+1} \widehat{d}_{q'}^2 \ \in \mathcal{C}(L)$$

Note that L is designed such that only valid transitions could be simulated in $\mathcal{C}(L)$.

Encoding Minsky machines

Finally, from

$$f_q g_q a^{n+2} \ b \ \widehat{a}^{m+1} \widehat{d}_{q'} \ \in A^*$$

we obtain

$$f_q g_q a^{n+2} \ b \ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in A^* \cdot L$$

and

$$f_q g_q \cdot a^{n+2} \ b \ \widehat{a}^{m+1} \widehat{d}_{q'} \cdot \widehat{d}_{q'} \ \in L \cdot A^*$$

So that we related two configurations of the machine:

$$a^{n+1} \ b \ \widehat{a}^{m+1} \ \widehat{d}_q^2 \in \mathcal{C}(L) \iff a^{n+2} \ b \ \widehat{a}^{m+1} \widehat{d}_{q'}^2 \ \in \mathcal{C}(L)$$

Note that L is designed such that only valid transitions could be simulated in $\mathcal{C}(L)$.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
 & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, ... is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
& a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, ... is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
 & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, ... is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
 & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, ... is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
 & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, ... is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

$$
\begin{array}{rrl}
 & a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'} & \in \mathcal{C}(L) \\
\Longleftrightarrow & a^{n+2}\ b\ \widehat{a}^{m+1}\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}
$$

Note that the purpose of the words e_q, $f_q g_q$, $\ldots$ is to manipulate data.

They allow to add (or remove, by "reading bottom-up" the equivalences) letters on each side of a word in the centralizer, depending on the state.

By contrast, the words $\widehat{d}_q$ do not manipulate data, but carry the state information.

Encoding Minsky machines: a summary

Kunc encodes similarly the other operations of Minsky machines: decreasing counters, testing, for each counter.

Another useful construction he uses provides

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d_q^2} \in \mathcal{C}(L) \quad \Longleftrightarrow \quad d_q^2\ a^{n+1}\ b\ \widehat{a}^{m+1} \in \mathcal{C}(L)$$

Before giving the last ingredients of Kunc's proof, we find useful to mention that an arguably simpler kind of machine can be used for the proof.

Encoding Minsky machines: a summary

Kunc encodes similarly the other operations of Minsky machines: decreasing counters, testing, for each counter.

Another useful construction he uses provides

$$a^{n+1}\ b\ \widehat{a}^{m+1}\ \widehat{d_q^2} \in \mathcal{C}(L) \quad \Longleftrightarrow \quad d_q^2\ a^{n+1}\ b\ \widehat{a}^{m+1} \in \mathcal{C}(L)$$

Before giving the last ingredients of Kunc's proof, we find useful to mention that an arguably simpler kind of machine can be used for the proof.

Clockwise Turing machines

We found it easier to adapt the proof to encode clockwise Turing machines, which only have one possible transition.

They are due to Neary and Woods. Informally, they are a variant of Turing machines with

- one circular tape,
- a clockwise-moving head,
- and the possibility to output two symbols at once to extend the tape.

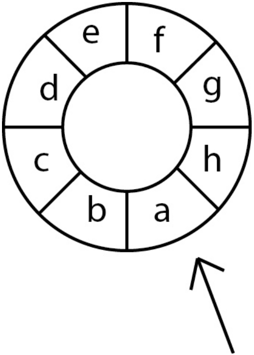
Clockwise Turing machines

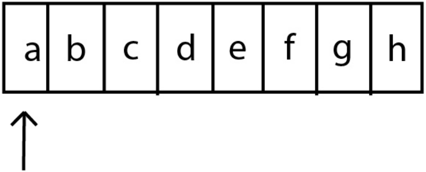
We found it easier to adapt the proof to encode clockwise Turing machines, which only have one possible transition.

They are due to Neary and Woods. Informally, they are a variant of Turing machines with

- one circular tape,
- a clockwise-moving head,
- and the possibility to output two symbols at once to extend the tape.

Clockwise Turing machines vs. Turing machines

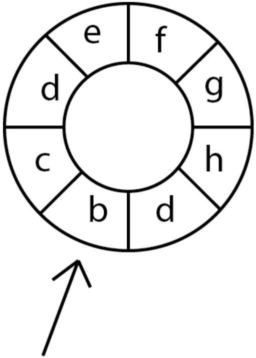


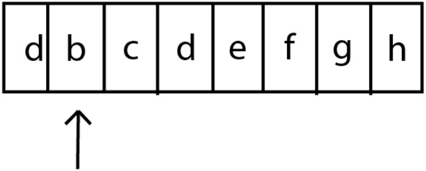


Suppose both machines are in state q, and the Turing machine reads a, writes d and moves tape to the right.

Clockwise Turing machines vs. Turing machines

We obtain:





both in the new state q'.

Clockwise Turing machines

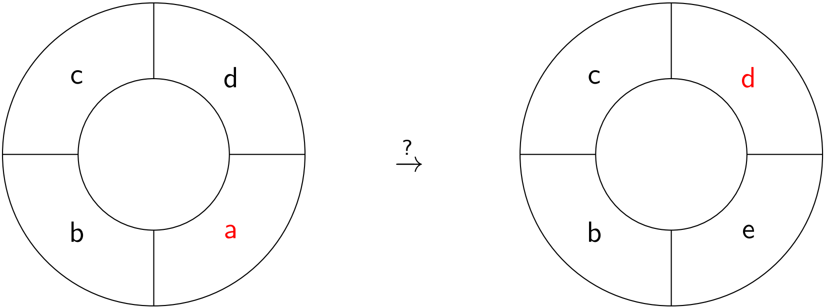
Clockwise Turing machines simulate Turing machines.

It should be noted that they do not need to store the size of the tape to simulate a counter-clockwise transition.

This is a crucial point, since encoding such a register would have required an infinite alphabet.

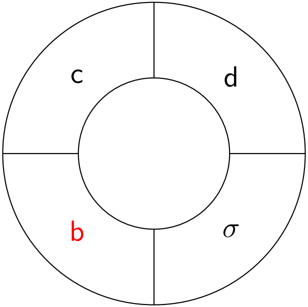
But this is not the case. Let us sketch how a counter-clockwise move can be simulated with clockwise transitions and without infinite memory (it needs some more states and one more tape symbol, though).

Clockwise Turing machines



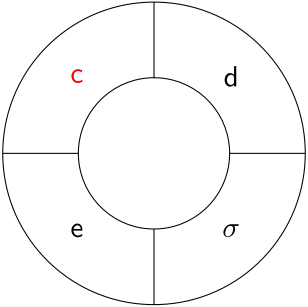
Suppose that the machine's head in on *a*, and that it wants to move counter-clockwise after replacing *a* with *e*.

Clockwise Turing machines



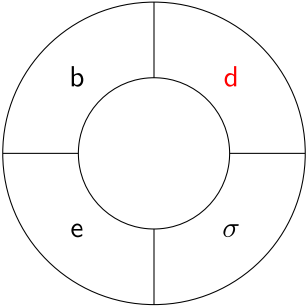
First, it replaces *a* with a special symbol σ, and moves clockwise.

Clockwise Turing machines



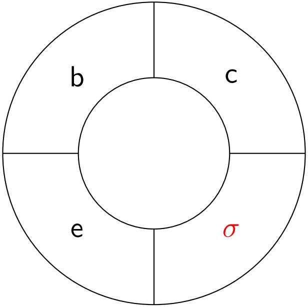
Then b is replaced with e, and the state remembers the fact that b needs to be translated. The head moves clockwise.

Clockwise Turing machines



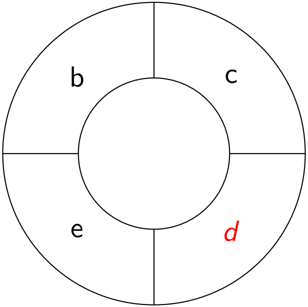
Then c is replaced with b, and the state remembers the fact that c needs to be translated. The head moves clockwise.

Clockwise Turing machines



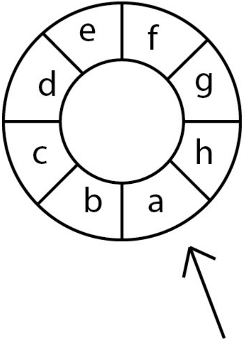
Then d is replaced with c, and the state remembers the fact that d needs to be translated. The head moves clockwise.

Clockwise Turing machines



On the special symbol σ, the machine outputs d, and proceeds to the simulation of the next move of the circular Turing machine.

Clockwise Turing machines: encoding configurations

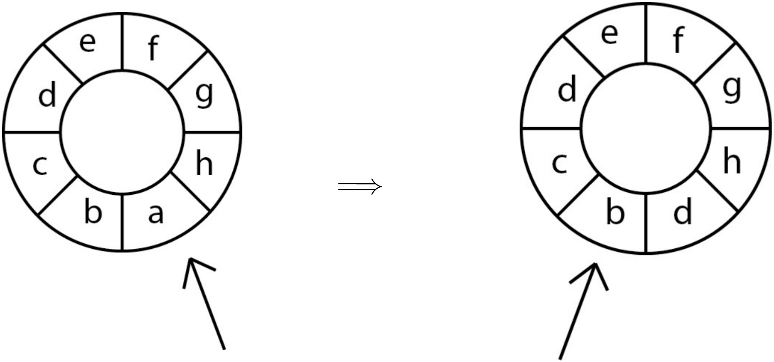


in state q is encoded as the word

$$a\ b\ c\ d\ e\ f\ g\ h\ \widehat{d}_q^2$$

Clockwise Turing machines: encoding configurations

The transition



from state q to state q' should be, following Kunc, encoded as

$$a\ b\ c\ d\ e\ f\ g\ h\ \widehat{d}^2_q \ \in\ \mathcal{C}(L) \iff b\ c\ d\ e\ f\ g\ h\ d\ \widehat{d}^2_{q'} \ \in\ \mathcal{C}(L)$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \; = \; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$
\begin{array}{lcr}
 & u_1 \; u_2 \; \cdots \; u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\iff & f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; d_q & \in \mathcal{C}(L) \\
\iff & e_{q,u_1} \; f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n & \in \mathcal{C}(L) \\
\iff & g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\iff & u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} \; \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\iff & d_{q'} \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\iff & d_{q'}^2 \; u_2 \; \cdots \; u_n \; v & \in \mathcal{C}(L) \\
\iff & u_2 \; \cdots \; u_n \; v \; \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}
$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \;=\; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rll}
 & u_1\ u_2\ \cdots\ u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ \widehat{d_q} & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1}\ f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ v\ \widehat{g_{q',v}} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{g_{q',v}}\ \widehat{f_{q',v}}\ \widehat{e_{q',v}} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}\ u_2\ \cdots\ u_n\ v\ \widehat{g_{q',v}}\ \widehat{f_{q',v}} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2\ u_2\ \cdots\ u_n\ v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \ = \ (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$
\begin{array}{llr}
 & u_1 \ u_2 \ \cdots \ u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1} \ g_{q,u_1} \ u_1 \ u_2 \ \cdots \ u_n \ d_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1} \ f_{q,u_1} \ g_{q,u_1} \ u_1 \ u_2 \ \cdots \ u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1} \ u_1 \ u_2 \ \cdots \ u_n \ v \ \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \ \cdots \ u_n \ v \ \widehat{g}_{q',v} \ \widehat{f}_{q',v} \ \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'} \ u_2 \ \cdots \ u_n \ v \ \widehat{g}_{q',v} \ \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2 \ u_2 \ \cdots \ u_n \ v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \ \cdots \ u_n \ v \ \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}
$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) = (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rrr}
 & u_1\ u_2\ \cdots\ u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ \widehat{d_q} & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1}\ f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v}\ \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2\ u_2\ \cdots\ u_n\ v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \;=\; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rcr}
 & u_1\ u_2\ \cdots\ u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ d_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1}\ f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v}\ \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2\ u_2\ \cdots\ u_n\ v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) = (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rcr}
 & u_1\ u_2\ \cdots\ u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\iff & f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ \widehat{d_q} & \in \mathcal{C}(L) \\
\iff & e_{q,u_1}\ f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n & \in \mathcal{C}(L) \\
\iff & g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\iff & u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v}\ \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\iff & d_{q'}\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\iff & d_{q'}^2\ u_2\ \cdots\ u_n\ v & \in \mathcal{C}(L) \\
\iff & u_2\ \cdots\ u_n\ v\ \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \;=\; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rrl}
 & u_1 \; u_2 \; \cdots \; u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1} \; f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} \; \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'} \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2 \; u_2 \; \cdots \; u_n \; v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \; \cdots \; u_n \; v \; \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \;=\; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{lcr}
 & u_1 \; u_2 \; \cdots \; u_n \widehat{d_q^2} & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; \widehat{d_q} & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1} \; f_{q,u_1} \; g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1} \; u_1 \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} \; \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'} \; u_2 \; \cdots \; u_n \; v \; \widehat{g}_{q',v} \; \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2 \; u_2 \; \cdots \; u_n \; v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2 \; \cdots \; u_n \; v \; \widehat{d_{q'}^2} & \in \mathcal{C}(L)
\end{array}$$

Clockwise Turing machines

We can use Kunc's ideas to define L such that a transition

$$\delta(q, u_1) \;=\; (v, q')$$

when the circular tape contains $u_1 \cdots u_n$ corresponds to:

$$\begin{array}{rcr}
 & u_1\ u_2\ \cdots\ u_n \widehat{d}_q^2 & \in \mathcal{C}(L) \\
\Longleftrightarrow & f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ \widehat{d}_q & \in \mathcal{C}(L) \\
\Longleftrightarrow & e_{q,u_1}\ f_{q,u_1}\ g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n & \in \mathcal{C}(L) \\
\Longleftrightarrow & g_{q,u_1}\ u_1\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v}\ \widehat{e}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}\ u_2\ \cdots\ u_n\ v\ \widehat{g}_{q',v}\ \widehat{f}_{q',v} & \in \mathcal{C}(L) \\
\Longleftrightarrow & d_{q'}^2\ u_2\ \cdots\ u_n\ v & \in \mathcal{C}(L) \\
\Longleftrightarrow & u_2\ \cdots\ u_n\ v\ \widehat{d}_{q'}^2 & \in \mathcal{C}(L)
\end{array}$$

Elements of the centralizer

Using ideas of Kunc's proof, we can prove that the encoding of every configuration is in $\mathcal{C}(L)$.

The way of proving this is by checking by hand that the set of words used to relate configurations is a solution of the commutation equation.

Recursive enumerability

With this encoding, we intuitively get that centralizers can encode recursively enumerable languages, as they simulate the behaviour of Turing machines.

But where does the non-r.e. comes from ?

The answer is that the intuition is somehow misleading, because centralizers are coinductive.

In other terms, they compute the whole configuration graph of the machine.

Another key ingredient of Kunc's proof is to remove the initial configurations of $\mathcal{C}(L)$, so that what remains in the centralizer corresponds to the complementary of the language of the machine.

Recursive enumerability

With this encoding, we intuitively get that centralizers can encode recursively enumerable languages, as they simulate the behaviour of Turing machines.

But where does the non-r.e. comes from ?

The answer is that the intuition is somehow misleading, because centralizers are coinductive.

In other terms, they compute the whole configuration graph of the machine.

Another key ingredient of Kunc's proof is to remove the initial configurations of $\mathcal{C}(L)$, so that what remains in the centralizer corresponds to the complementary of the language of the machine.

Recursive enumerability

With this encoding, we intuitively get that centralizers can encode recursively enumerable languages, as they simulate the behaviour of Turing machines.

But where does the non-r.e. comes from ?

The answer is that the intuition is somehow misleading, because centralizers are coinductive.

In other terms, they compute the whole configuration graph of the machine.

Another key ingredient of Kunc's proof is to remove the initial configurations of $\mathcal{C}(L)$, so that what remains in the centralizer corresponds to the complementary of the language of the machine.

Induction vs. coinduction

In an inductive construction, one starts from some initial element and iterates a construction over it.

This is the point of view of calculus: a machine starts on an initial configuration and iterates its transition function over it.

Inductive interpretations only build finitary objects.

In the situation of calculus, this corresponds to only considering terminating computations – which is the standard point of view in calculability theory.

Induction vs. coinduction

In an inductive construction, one starts from some initial element and iterates a construction over it.

This is the point of view of calculus: a machine starts on an initial configuration and iterates its transition function over it.

Inductive interpretations only build finitary objects.

In the situation of calculus, this corresponds to only considering terminating computations – which is the standard point of view in calculability theory.

Induction vs. coinduction

In a coinductive construction, one starts from all elements and iteratively removes the ones which contradict some construction (or deduction) rule.

In the example of calculus, this corresponds to the graph of configurations of a machine:

1. Start with the (countable) complete graph whose vertices are the configurations:
$$V = A^* \times Q$$
2. Iteratively remove the edges which do not correspond to a transition of the machine.

As we will see now, this is precisely how coinduction works.

Induction vs. coinduction

In a coinductive construction, one starts from all elements and iteratively removes the ones which contradict some construction (or deduction) rule.

In the example of calculus, this corresponds to the graph of configurations of a machine:

1. Start with the (countable) complete graph whose vertices are the configurations:
$$V \quad = \quad A^* \times Q$$
2. Iteratively remove the edges which do not correspond to a transition of the machine.

As we will see now, this is precisely how coinduction works.

Induction vs. coinduction

In a coinductive construction, one starts from all elements and iteratively removes the ones which contradict some construction (or deduction) rule.

In the example of calculus, this corresponds to the graph of configurations of a machine:

1. Start with the (countable) complete graph whose vertices are the configurations:
$$V \quad = \quad A^* \times Q$$
2. Iteratively remove the edges which do not correspond to a transition of the machine.

As we will see now, this is precisely how coinduction works.

Induction vs. coinduction: lattices and fixed points

Theorem (Tarski-Knaster)

Let $\mathcal{L}$ be a complete lattice and let

$$f \; : \; \mathcal{L} \longrightarrow \mathcal{L}$$

be an order-preserving function.
Then the set of fixed points of f in $\mathcal{L}$ is also a complete lattice.

In other terms: if you define a function f on an ordered structure with supremum, infimum, least and greatest element, then it has fixed points.

Moreover, there is a least and a greatest fixed points of f.

And the greatest is the supremum of all fixed points.

Induction vs. coinduction: lattices and fixed points

Theorem (Tarski-Knaster)

Let $\mathcal{L}$ be a complete lattice and let

$$f \;:\; \mathcal{L} \longrightarrow \mathcal{L}$$

be an order-preserving function.
Then the set of fixed points of f in $\mathcal{L}$ is also a complete lattice.

In other terms: if you define a function f on an ordered structure with supremum, infimum, least and greatest element, then it has fixed points.

Moreover, there is a least and a greatest fixed points of f.

And the greatest is the supremum of all fixed points.

Induction vs. coinduction: lattices and fixed points

Inductive constructions correspond to least fixpoints

$$\text{lfp}(f) = \bigvee_i f^i(\bot)$$

and coinductive ones to greatest fixpoints

$$\text{gfp}(f) = \bigwedge_i f^i(\top)$$

(note that in some cases i may have to take ordinal values. . . but not in this talk)

Induction vs. coinduction: lattices and fixed points

$$\mathrm{lfp}(f) \;=\; \bigvee_i \; f^i(\bot)$$

precisely means that inductive constructions start over some element (in a lattice $\mathcal{P}(S)$, it is the empty set), and construct iteratively a solution.

This is the spirit of the calculus of a machine.

Induction vs. coinduction: lattices and fixed points

$$\mathrm{gfp}(f) \;=\; \bigwedge_i \; f^i(\top)$$

precisely means that coinductive constructions start from all elements (in a lattice $\mathcal{P}(S)$, it is S), and "destruct it" iteratively until obtaining a solution.

This is the spirit of the "computation" of the configuration graph of the machine.

Induction vs. coinduction: intuitions

A useful intuition is the fact that induction and coinduction provide have two different understandings of the word infinity:

- Induction generates infinite structure, in the sense that they are unbounded.
- Coinduction generates infinite structures, in the sense that they can contain infinite (countable or more, depending on the framework) sequences.

Induction vs. coinduction: examples

Inductive	Coinductive
Languages of words	Languages of ω-words
Finite trees	Infinite trees
Lists	Streams
Computation	Configuration graph

Induction vs. coinduction: examples

Inductive	Coinductive
Languages of words	Languages of ω-words
Finite trees	Infinite trees
Lists	Streams
Computation	Configuration graph

Induction vs. coinduction: examples

Inductive	Coinductive
Languages of words	Languages of ω-words
Finite trees	Infinite trees
Lists	Streams
Computation	Configuration graph

Induction vs. coinduction: examples

Inductive	Coinductive
Languages of words	Languages of ω-words
Finite trees	Infinite trees
Lists	Streams
Computation	Configuration graph

Other applications of coinduction

Coinduction is used to:

- Study the behavourial equivalence of (potentially infinite) processes: this is the notion of bisimulation, which is an equivalence relation between processes
- Define infinite data types (infinite trees, streams. . .)
- In μ-calculus, to specify properties about infinite behaviour of programs (it also hides in LTL and CTL)
- More generally, it hides in every "relation refinment" algorithm, as in the computation of the minimal automaton for example.

Other applications of coinduction

Coinduction is used to:

- Study the behavourial equivalence of (potentially infinite) processes: this is the notion of bisimulation, which is an equivalence relation between processes
- Define infinite data types (infinite trees, streams. . .)
- In μ-calculus, to specify properties about infinite behaviour of programs (it also hides in LTL and CTL)
- More generally, it hides in every "relation refinment" algorithm, as in the computation of the minimal automaton for example.

Other applications of coinduction

Coinduction is used to:

- Study the behavourial equivalence of (potentially infinite) processes: this is the notion of bisimulation, which is an equivalence relation between processes
- Define infinite data types (infinite trees, streams. . .)
- In μ-calculus, to specify properties about infinite behaviour of programs (it also hides in LTL and CTL)
- More generally, it hides in every "relation refinment" algorithm, as in the computation of the minimal automaton for example.

Other applications of coinduction

Coinduction is used to:

- Study the behavourial equivalence of (potentially infinite) processes: this is the notion of bisimulation, which is an equivalence relation between processes
- Define infinite data types (infinite trees, streams. . .)
- In μ-calculus, to specify properties about infinite behaviour of programs (it also hides in LTL and CTL)
- More generally, it hides in every "relation refinment" algorithm, as in the computation of the minimal automaton for example.

Centralizers are coinductive

Over the lattice $\mathcal{L} = \mathcal{P}(A^*)$, the function

$$\phi \ : \ X \ \mapsto \ (L^{-1} X) \cdot L \cap L \cdot (X L^{-1})$$

is order-preserving. As a consequence, it has fixed points, which form a complete lattice.

Notice that X is a fixed point of ϕ if and only if

$$X \ = \ (L^{-1} X) \cdot L \quad \text{and} \quad X \ = \ L \cdot (X L^{-1})$$

if and only if

$$X \cdot L \ = \ L \cdot X$$

The greatest fixpoint is $\mathcal{C}(L)$. It can be defined as the union (supremum) of all solutions.

Centralizers are coinductive

Over the lattice $\mathcal{L} = \mathcal{P}(A^*)$, the function

$$\phi \; : \; X \; \mapsto \; (L^{-1}\,X) \cdot L \cap L \cdot (X\,L^{-1})$$

is order-preserving. As a consequence, it has fixed points, which form a complete lattice.

Notice that X is a fixed point of ϕ if and only if

$$X \;=\; (L^{-1}\,X) \cdot L \quad \text{and} \quad X \;=\; L \cdot (X\,L^{-1})$$

if and only if

$$X \cdot L \;=\; L \cdot X$$

The greatest fixpoint is $\mathcal{C}(L)$. It can be defined as the union (supremum) of all solutions.

Centralizers are coinductive

Over the lattice $\mathcal{L} = \mathcal{P}(A^*)$, the function

$$\phi \ : \ X \ \mapsto \ (L^{-1}\,X) \cdot L \cap L \cdot (X\,L^{-1})$$

is order-preserving. As a consequence, it has fixed points, which form a complete lattice.

Notice that X is a fixed point of ϕ if and only if

$$X \ = \ (L^{-1}\,X) \cdot L \quad \text{and} \quad X \ = \ L \cdot (X\,L^{-1})$$

if and only if

$$X \cdot L \ = \ L \cdot X$$

The greatest fixpoint is $\mathcal{C}(L)$. It can be defined as the union (supremum) of all solutions.

Coinduction and game interpretation

Game-theoretically, we can understand coinductive constructions as games where Eve can prove during "long-enough plays" (here, countably infinite ones) that she has a justification for her moves iff she starts from an element of the coinductive object.

Recall the situation for centralizers: starting from some word, Eve had a winning strategy iff the word was in $\mathcal{C}(L)$.

Note that in this commutation game, Eve's winning strategies correspond to commutation orbits of the language.

These orbits correspond to the notion of self-justifying sets, which is typical of coinduction.

Coinduction and game interpretation

Game-theoretically, we can understand coinductive constructions as games where Eve can prove during “long-enough plays” (here, countably infinite ones) that she has a justification for her moves iff she starts from an element of the coinductive object.

Recall the situation for centralizers: starting from some word, Eve had a winning strategy iff the word was in $\mathcal{C}(L)$.

Note that in this commutation game, Eve's winning strategies correspond to commutation orbits of the language.

These orbits correspond to the notion of self-justifying sets, which is typical of coinduction.

Coinduction and game interpretation

Game-theoretically, we can understand coinductive constructions as games where Eve can prove during "long-enough plays" (here, countably infinite ones) that she has a justification for her moves iff she starts from an element of the coinductive object.

Recall the situation for centralizers: starting from some word, Eve had a winning strategy iff the word was in $\mathcal{C}(L)$.

Note that in this commutation game, Eve's winning strategies correspond to commutation orbits of the language.

These orbits correspond to the notion of self-justifying sets, which is typical of coinduction.

Coinduction and centralizers

Recall the elements of Kunc's proof: we can design a language L such that

1. It contains the encoding of the configurations of a circular Turing machine
2. Actually, the coinductive interpretation says that it encodes the configuration graph of the machine
3. Two encodings are in the same commutation orbit if and only if they are in the same connected component of the configuration graph of the machine
4. We have a way to exclude some configurations of $\mathcal{C}(L)$

Coinduction and centralizers

Recall the elements of Kunc's proof: we can design a language L such that

1. It contains the encoding of the configurations of a circular Turing machine
2. Actually, the coinductive interpretation says that it encodes the configuration graph of the machine
3. Two encodings are in the same commutation orbit if and only if they are in the same connected component of the configuration graph of the machine
4. We have a way to exclude some configurations of $\mathcal{C}(L)$

Coinduction and centralizers

Recall the elements of Kunc's proof: we can design a language L such that

1. It contains the encoding of the configurations of a circular Turing machine
2. Actually, the coinductive interpretation says that it encodes the configuration graph of the machine
3. Two encodings are in the same commutation orbit if and only if they are in the same connected component of the configuration graph of the machine
4. We have a way to exclude some configurations of $\mathcal{C}(L)$

Coinduction and centralizers

Recall the elements of Kunc's proof: we can design a language L such that

1. It contains the encoding of the configurations of a circular Turing machine
2. Actually, the coinductive interpretation says that it encodes the configuration graph of the machine
3. Two encodings are in the same commutation orbit if and only if they are in the same connected component of the configuration graph of the machine
4. We have a way to exclude some configurations of $\mathcal{C}(L)$

Coinduction and centralizers

Now, adapting Kunc's proof, we modify L so that precisely every initial configuration of the machine is removed from $\mathcal{C}(L)$.

It has the effect of removing their commutation orbits as well, that is, the corresponding connected components of the configuration graph of the machine.

But these components are precisely the computations of the machine !

Coinduction and centralizers

Now, adapting Kunc's proof, we modify L so that precisely every initial configuration of the machine is removed from $\mathcal{C}(L)$.

It has the effect of removing their commutation orbits as well, that is, the corresponding connected components of the configuration graph of the machine.

But these components are precisely the computations of the machine !

Coinduction and centralizers

As a consequence, at this stage, $\mathcal{C}(L)$ contains only

- commutation orbits corresponding to infinite computations (non-terminating ones), which do not compute elements of the language of the machine,
- and commutation orbits which may reach a final configuration but not accessible from an initial configuration: that is, elements of the complementary of the machine's language.

Coinduction and centralizers

In other terms:

$\mathcal{C}(L)$ contains the encoding of the complementary of the language of a (circular) Turing machine.

Taking a universal machine gives Kunc's theorem:

$\mathcal{C}(L)$ is not recursively enumerable (but it is co-r.e.).

Centralizers of finite languages

Recall the second part of the Theorem: L can be finite.

So far, the language we built is star-free – yet defined with stars.

It consists on a finite amount of interaction words: $f_{u,q}\, g_{u,q}$, $\widehat{d}_q$, ... used for simulating transitions, and of an infinite amount of restriction words, designed to restrict the centralizer to actual simulations of transitions.

Informally, they ensure that if you remove more than you should, then you have to remove so much that you will eventually "loose the game".

$$e_q \ f_q \ g_q \ a^{n+2} \ b \ \widehat{a}^{m+1}$$

Kunc gives a manner to "cut the stars" into finite words, while "forcing the players to respect them in their plays".

Centralizers of finite languages

Recall the second part of the Theorem: L can be finite.

So far, the language we built is star-free – yet defined with stars.

It consists on a finite amount of interaction words: $f_{u,q}\, g_{u,q}$, $\widehat{d}_q$, ... used for simulating transitions, and of an infinite amount of restriction words, designed to restrict the centralizer to actual simulations of transitions.

Informally, they ensure that if you remove more than you should, then you have to remove so much that you will eventually "loose the game".

$$e_q \;\; f_q \;\; g_q \;\; a^{n+2} \;\; b \;\; \widehat{a}^{m+1}$$

Kunc gives a manner to "cut the stars" into finite words, while "forcing the players to respect them in their plays".

Centralizers of finite languages

Recall the second part of the Theorem: L can be finite.

So far, the language we built is star-free – yet defined with stars.

It consists on a finite amount of interaction words: $f_{u,q}\, g_{u,q}$, $\widehat{d}_q$, ...
used for simulating transitions, and of an infinite amount of restriction words, designed to restrict the centralizer to actual simulations of transitions.

Informally, they ensure that if you remove more than you should, then you have to remove so much that you will eventually "loose the game".

$$e_q \ f_q \ g_q \ a^{n+2} \ b \ \widehat{a}^{m+1}$$

Kunc gives a manner to "cut the stars" into finite words, while "forcing the players to respect them in their plays".

Centralizers of finite languages

Recall the second part of the Theorem: L can be finite.

So far, the language we built is star-free – yet defined with stars.

It consists on a finite amount of interaction words: $f_{u,q}\, g_{u,q}$, $\widehat{d}_q$, ... used for simulating transitions, and of an infinite amount of restriction words, designed to restrict the centralizer to actual simulations of transitions.

Informally, they ensure that if you remove more than you should, then you have to remove so much that you will eventually "loose the game".

$$e_q\ f_q\ g_q\ a^{n+2}\ b\ \widehat{a}^{m+1}$$

Kunc gives a manner to "cut the stars" into finite words, while "forcing the players to respect them in their plays".

Centralizers of finite languages

This gives a finite language L, obtained from the star-free language one. However, it requires a huge number of impossibility words.

The main reason for us to use a circular Turing machine – and not a Minsky machine – was in fact to estimate the cardinality of this finite language.

For the smallest universal Turing machine we know (4 states over a 4-symbol alphabet), it is about 10^{21} words; almost all of them are restriction words.

Centralizers of finite languages

This gives a finite language L, obtained from the star-free language one. However, it requires a huge number of impossibility words.

The main reason for us to use a circular Turing machine – and not a Minsky machine – was in fact to estimate the cardinality of this finite language.

For the smallest universal Turing machine we know (4 states over a 4-symbol alphabet), it is about 10^{21} words; almost all of them are restriction words.

Conclusion

We sketched a variant of Kunc's proof, which has three strengths:

- Only one kind of transition has to be considered, unlike for Minsky machines (or usual Turing ones)
- The coinductive nature of centralizers helps the understanding of the result and the presentation of the proof
- Cardinality of L can be estimated more accurately in the finite case.

Thank you for your attention !

Conclusion

We sketched a variant of Kunc's proof, which has three strengths:

- Only one kind of transition has to be considered, unlike for Minsky machines (or usual Turing ones)
- The coinductive nature of centralizers helps the understanding of the result and the presentation of the proof
- Cardinality of L can be estimated more accurately in the finite case.

Thank you for your attention !