On the coinductive nature of centralizers

Charles Grellois

PPS & LIAFA — Université Paris 7

September 25th, 2014

Commutation of words

In language theory, the solution of the word equation

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

is well-know: it is $\{w^n \mid n \in \mathbb{N}\}$.

But what about commutation of languages ?

Commutation of words

In language theory, the solution of the word equation

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

is well-know: it is $\{w^n \mid n \in \mathbb{N}\}$.

But what about commutation of languages ?

Centralizers

Consider the language commutation equation

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

It has solutions : since

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

is order-preserving, it has fixpoints (Knaster-Tarski), which form a complete lattice.

The greatest fixpoint is called the centralizer of L, and is denoted $\mathcal{C}(L)$.

It is the greatest language satisfying (1), and can be defined as the union (supremum) of all solutions.

Centralizers

Consider the language commutation equation

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

It has solutions : since

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

is order-preserving, it has fixpoints (Knaster-Tarski), which form a complete lattice.

The greatest fixpoint is called the centralizer of L, and is denoted $\mathcal{C}(L)$.

It is the greatest language satisfying (1), and can be defined as the union (supremum) of all solutions.

Centralizers

Consider the language commutation equation

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

It has solutions : since

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

is order-preserving, it has fixpoints (Knaster-Tarski), which form a complete lattice.

The greatest fixpoint is called the centralizer of L, and is denoted $\mathcal{C}(L)$.

It is the greatest language satisfying (1), and can be defined as the union (supremum) of all solutions.

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 whose centralizer is not recursively enumerable.*
- *There exists a finite language whose centralizer is not recursively enumerable.*

In this talk, we describe a simpler variant of Kunc's proof, and reveal an important key for understanding this theorem: centralizers are coinductive.

I would like to thank Prof. Karhumäki for introducing me to this problem.

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 whose centralizer is not recursively enumerable.*
- *There exists a finite language whose centralizer is not recursively enumerable.*

In this talk, we describe a simpler variant of Kunc's proof, and reveal an important key for understanding this theorem: centralizers are coinductive.

I would like to thank Prof. Karhumäki for introducing me to this problem.

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 whose centralizer is not recursively enumerable.*
- *There exists a finite language whose centralizer is not recursively enumerable.*

In this talk, we describe a simpler variant of Kunc's proof, and reveal an important key for understanding this theorem: centralizers are coinductive.

I would like to thank Prof. Karhumäki for introducing me to this problem.

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 whose centralizer is not recursively enumerable.*
- *There exists a finite language whose centralizer is not recursively enumerable.*

In this talk, we describe a simpler variant of Kunc's proof, and reveal an important key for understanding this theorem: centralizers are coinductive.

I would like to thank Prof. Karhumäki for introducing me to this problem.

Centralizers, game-theoretically

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

$$u \in L$$

The first player appends a word $x \in A^*$ as a prefix or a suffix, and reaches:

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

Then the second player removes a prefix $y \in A^*$ such that

$$y^{-1} u x \;\in\; L$$

and this word becomes the new position of the game, on which the first player appends again a prefix or a suffix, and so on.

Centralizers, game-theoretically

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

$$u \in L$$

The first player appends a word $x \in A^*$ as a prefix or a suffix, and reaches:

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

Then the second player removes a prefix $y \in A^*$ such that

$$y^{-1} u x \ \in \ L$$

and this word becomes the new position of the game, on which the first player appends again a prefix or a suffix, and so on.

Centralizers, game-theoretically

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

$$u \in L$$

The first player appends a word $x \in A^*$ as a prefix or a suffix, and reaches:

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

Then the second player removes a prefix $y \in A^*$ such that

$$y^{-1}\, u\, x \ \in \ L$$

and this word becomes the new position of the game, on which the first player appends again a prefix or a suffix, and so on.

Centralizers, game-theoretically

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

$$u \in L$$

The first player appends a word $x \in A^*$ as a prefix or a suffix, and reaches:

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

Then the second player removes a prefix $y \in A^*$ such that

$$y^{-1} u x \ \in \ L$$

and this word becomes the new position of the game, on which the first player appends again a prefix or a suffix, and so on.

Centralizers, game-theoretically

If the second player can always move, the set of words x, y, $\cdots$ picked by both players form a solution of the commutation equation

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

since:

Proposition

Given $u, v \in L$, suppose that $u \cdot x \;=\; y \cdot v$. Then $x \in \mathcal{C}(L) \iff y \in \mathcal{C}(L)$.

So $\mathcal{C}(L)$ is the maximal set of words which can be picked in successful (that is, infinite) plays of this game.

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

Centralizers, game-theoretically

If the second player can always move, the set of words x, y, $\cdots$ picked by both players form a solution of the commutation equation

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

since:

Proposition

Given u, $v \in L$, suppose that $u \cdot x \ = \ y \cdot v$. Then $x \in \mathcal{C}(L) \iff y \in \mathcal{C}(L)$.

So $\mathcal{C}(L)$ is the maximal set of words which can be picked in successful (that is, infinite) plays of this game.

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

Centralizers, game-theoretically

If the second player can always move, the set of words x, y, $\cdots$ picked by both players form a solution of the commutation equation

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

since:

Proposition

Given u, $v \in L$, suppose that $u \cdot x \; = \; y \cdot v$. Then $x \in \mathcal{C}(L) \iff y \in \mathcal{C}(L)$.

So $\mathcal{C}(L)$ is the maximal set of words which can be picked in successful (that is, infinite) plays of this game.

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

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 not simulate more transitions)

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 not simulate more transitions)

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.

A configuration is "encoded" in $\mathcal{C}(L)$ as the word

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

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.

A configuration is "encoded" in $\mathcal{C}(L)$ as the word

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

Encoding Minsky machines

Increasing the first counter is simulated in $\mathcal{C}(L)$ as follows :

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

thanks to 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

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

Clockwise Turing machines

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

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

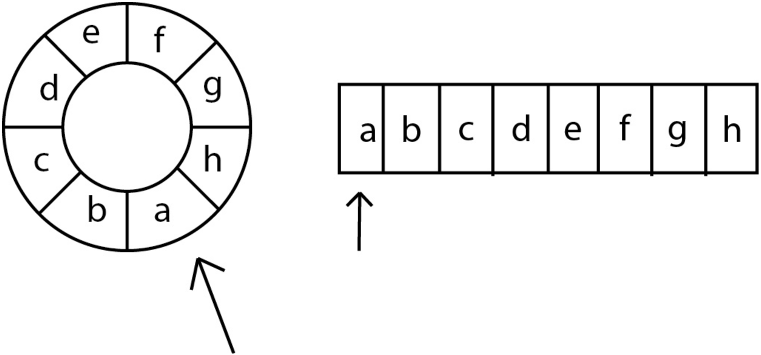
Kunc encodes similarly the other operations of Minsky machines: decreasing counters, testing, for each counter.

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.

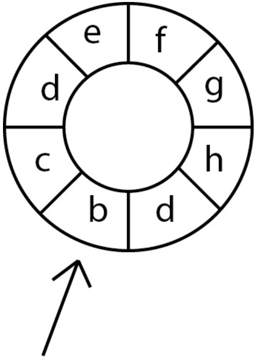
Clockwise Turing machines vs. Turing machines

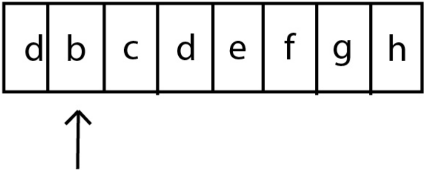


Suppose both machines are in state q, and the Turing machine reads a, writes d and moves tape to the right.

Clockwise Turing machines vs. Turing machines

We obtain:





both in the new state q'.

Clockwise Turing machines

Clockwise Turing machines simulate Turing machines.

And 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:

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

Clockwise Turing machines

Clockwise Turing machines simulate Turing machines.
And 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:

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

Clockwise Turing machines

Clockwise Turing machines simulate Turing machines.
And 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:

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

Recursive enumerability

With this encoding, we intuitively get that centralizers can encode recursively enumerable languages.

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

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

Recursive enumerability

With this encoding, we intuitively get that centralizers can encode recursively enumerable languages.

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

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

Induction vs. coinduction

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

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

Note that these are dual notions, in the sense of category theory.

Also, the game interpretation we gave of centralizers is strongly related to coinduction and self-justifying sets.

Intuitively, if you can play forever (never be contradicted), you construct an element of a coinductive structure. Even if you play periodically the same finite play – this corresponds to self-justifying sets.

Induction vs. coinduction

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

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

Note that these are dual notions, in the sense of category theory.

Also, the game interpretation we gave of centralizers is strongly related to coinduction and self-justifying sets.

Intuitively, if you can play forever (never be contradicted), you construct an element of a coinductive structure. Even if you play periodically the same finite play – this corresponds to self-justifying sets.

Induction vs. coinduction

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

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

Note that these are dual notions, in the sense of category theory.

Also, the game interpretation we gave of centralizers is strongly related to coinduction and self-justifying sets.

Intuitively, if you can play forever (never be contradicted), you construct an element of a coinductive structure. Even if you play periodically the same finite play – this corresponds to self-justifying sets.

Induction vs. coinduction

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

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

Note that these are dual notions, in the sense of category theory.

Also, the game interpretation we gave of centralizers is strongly related to coinduction and self-justifying sets.

Intuitively, if you can play forever (never be contradicted), you construct an element of a coinductive structure. Even if you play periodically the same finite play – this corresponds to self-justifying sets.

Induction vs. coinduction

Inductive constructions correspond to least fixpoints

$$\mathrm{lfp}(f) \;=\; \bigvee_{i \in \mathbb{N}} \; f^i(\bot)$$

and coinductive ones to greatest fixpoints

$$\mathrm{gfp}(f) \;=\; \bigwedge_{i \in \mathbb{N}} \; f^i(\top)$$

So that centralizers are coinductive. But what does it imply ?

Induction vs. coinduction, a first example

Consider a function $\phi : \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathbb{N})$ such that

$$\phi(X) \;=\; X \cup \{n+2 \mid n \in X\} \cup \{0\}$$

Its least fixpoint (inductive) is given by iteration over $\emptyset$:

$$\phi(\emptyset) \;=\; \{0\}$$

$$\phi^2(\emptyset) \;=\; \{0,\, 2\}$$

$$\phi^3(\emptyset) \;=\; \{0,\, 2,\, 4\}$$

so that $\mathrm{lfp}(\phi) \;=\; 2\mathbb{N}$.

Induction vs. coinduction, a first example

Consider a function $\phi : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ such that

$$\phi(X) \;=\; X \cup \{n+2 \mid n \in X\} \cup \{0\}$$

Its least fixpoint (inductive) is given by iteration over $\emptyset$:

$$\phi(\emptyset) \;=\; \{0\}$$

$$\phi^2(\emptyset) \;=\; \{0, 2\}$$

$$\phi^3(\emptyset) \;=\; \{0, 2, 4\}$$

so that $\mathrm{lfp}(\phi) \;=\; 2\mathbb{N}$.

Induction vs. coinduction, a first example

Consider a function $\phi : \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathbb{N})$ such that

$$\phi(X) \;=\; X \cup \{n+2 \mid n \in X\} \cup \{0\}$$

Its least fixpoint (inductive) is given by iteration over $\emptyset$:

$$\phi(\emptyset) \;=\; \{0\}$$

$$\phi^2(\emptyset) \;=\; \{0,\, 2\}$$

$$\phi^3(\emptyset) \;=\; \{0,\, 2,\, 4\}$$

so that $\mathrm{lfp}(\phi) \;=\; 2\mathbb{N}$.

Induction vs. coinduction, a first example

Consider a function $\phi : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ such that

$$\phi(X) \ = \ X \cup \{n+2 \mid n \in X\} \cup \{0\}$$

Its least fixpoint (inductive) is given by iteration over $\emptyset$:

$$\phi(\emptyset) \ = \ \{0\}$$

$$\phi^2(\emptyset) \ = \ \{0,\, 2\}$$

$$\phi^3(\emptyset) \ = \ \{0,\, 2,\, 4\}$$

so that $\text{lfp}(\phi) \ = \ 2\mathbb{N}$.

Induction vs. coinduction, a first example

Consider a function $\phi : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ such that

$$\phi(X) \;=\; X \cup \{n + 2 \mid n \in X\} \cup \{0\}$$

Dually, coinduction proceeds by considering all sets of $\mathcal{P}(\mathbb{N})$, and removing the ones which could not have been obtained from ϕ.

So gfp(ϕ) still contains $2\mathbb{N}$, but it also contains $\mathbb{N}$:

$$\text{gfp}(\phi) \;=\; \{\mathbb{N},\, 2\mathbb{N}\}$$

Induction vs. coinduction, a second example

Consider now configuration graphs of (circular) Turing machine.

Inductively, such a graph would be built from an initial configuration, iterating the transitions of the machine.

The result is the graph of all configurations of a given calculus – if it terminates.

In this case, the final configuration it contains encodes a word of the language of the machine.

Induction vs. coinduction, a second example

But coinductively, the graph is built starting from a graph

- whose vertices are all configurations of the machine (thus in infinite amount),
- and whose edge set is maximal.

Coinduction proceeds by "iteratively" (this is a transfinite process) removing the edges inconsistent with the machine's transitions.

The result is the whole configurations graph of the (circular) Turing machine.

It may have many connected components, some of which correspond to actual computations (they contain an initial configuration).

Induction vs. coinduction, a second example

But coinductively, the graph is built starting from a graph

- whose vertices are all configurations of the machine (thus in infinite amount),
- and whose edge set is maximal.

Coinduction proceeds by "iteratively" (this is a transfinite process) removing the edges inconsistent with the machine's transitions.

The result is the whole configurations graph of the (circular) Turing machine.

It may have many connected components, some of which correspond to actual computations (they contain an initial configuration).

The coinductive nature of centralizers

So, for the language L we designed from Kunc's ideas and constructions, we obtain $\mathcal{C}(L)$ representing the whole configurations graph of a circular Turing machine.

Note that all words simulating transitions belong to $\mathcal{C}(L)$, because they form self-justifying sets.

In other terms, both players can play chains of equivalences infinitely, by going back and forth.

The coinductive nature of centralizers

The next step is to extend the restricting power of L in order to exclude the encodings of initial configurations from it.

Coinductively, this removes all the connected components containing an initial configurations, that is, all valid computations of the machine from the graph.

Every encoding of a final configuration in the centralizer is thus unreachable by a machine's calculus.

In other terms, $\mathcal{C}(L)$ represents the complement of the machine's language.

The coinductive nature of centralizers

The next step is to extend the restricting power of L in order to exclude the encodings of initial configurations from it.

Coinductively, this removes all the connected components containing an initial configurations, that is, all valid computations of the machine from the graph.

Every encoding of a final configuration in the centralizer is thus unreachable by a machine's calculus.

In other terms, $\mathcal{C}(L)$ represents the complement of the machine's language.

The coinductive nature of centralizers

Considering a machine whose language is r.e. but not co-r.e., as a universal machine for instance, we obtain the first part of Kunc's theorem.

Note that there are universal circular Turing machines.

(in fact, they were introduced in order to obtain optimal ones)

The coinductive nature of centralizers

Considering a machine whose language is r.e. but not co-r.e., as a universal machine for instance, we obtain the first part of Kunc's theorem.

Note that there are universal circular Turing machines.

(in fact, they were introduced in order to obtain optimal ones)

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 notion of self-justifying set avoids a huge part of the proof (checking that the words encoding configurations where a transition is defined effectively belong to $\mathcal{C}(L)$)
- Cardinality of L can be estimated more accurately in the finite case.

Moreover, we revealed the coinductive nature of centralizers, which explains how they can compute "more than a machine" while simulating its transitions.

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 notion of self-justifying set avoids a huge part of the proof (checking that the words encoding configurations where a transition is defined effectively belong to $\mathcal{C}(L)$)
- Cardinality of L can be estimated more accurately in the finite case.

Moreover, we revealed the coinductive nature of centralizers, which explains how they can compute "more than a machine" while simulating its transitions.

Thank you for your attention !