The Expressive Power of Linear-time Temporal Logic

K Narayan Kumar

Chennai Mathematical Institute
email:kumar@cmi.ac.in

Chennai, Sept 2010

Linear-time Temporal Logic

LTL — convenient specification language

- Atomic propositions, boolean connectives, temporal modalities.

Linear-time Temporal Logic

LTL — convenient specification language

- Atomic propositions, boolean connectives, temporal modalities.
- Models are words.

Linear-time Temporal Logic

LTL — convenient specification language

- Atomic propositions, boolean connectives, temporal modalities.
- Models are words.

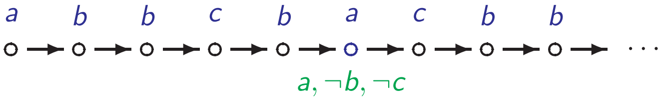
  Formulas are interpreted at positions of a word.

$$w = w_1 w_2 w_3 \ldots \qquad \text{with } w_i \in \Sigma$$

$$w, i \models \varphi \; ?$$

Atomic propositions: elements of Σ.

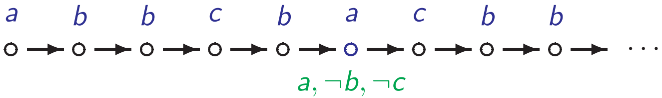
$$w, i \models a \iff w_i = a$$



Syntax and Semantics

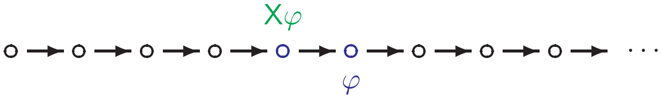
Atomic propositions: elements of Σ.

$$w, i \models a \iff w_i = a$$



The Next state operator:

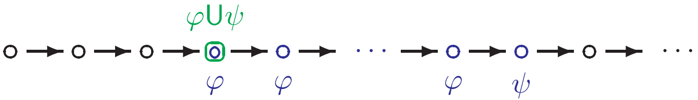
$$w, i \models \mathrm{X}\varphi \iff w, i+1 \models \varphi$$



Syntax and Semantics

The Until operator:

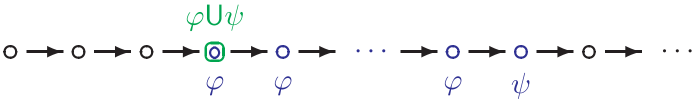
$$w, i \models \varphi \mathsf{U} \psi \iff \exists j \geq i.\ w, j \models \psi \text{ and } \forall i \leq k < j.\ w, k \models \varphi$$



Syntax and Semantics

The Until operator:

$$w, i \models \varphi \mathsf{U} \psi \quad \iff \quad \exists j \geq i.\ w, j \models \psi \text{ and } \forall i \leq k < j.\ w, k \models \varphi$$



Boolean Connectives:

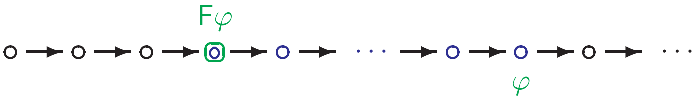
$$\varphi \wedge \psi, \quad \neg\varphi, \quad \ldots$$

with the usual interpretation.

Other Modalities

The Future modality

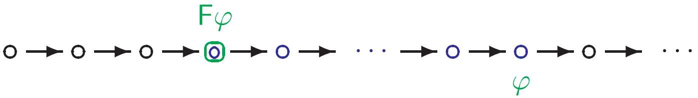
$$w, i \models F\varphi \iff \exists j \geq i.\ w, j \models \varphi$$



Other Modalities

The Future modality

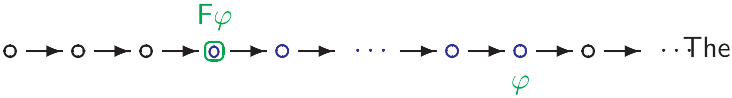
$$F\varphi \quad = \quad \top U \varphi$$



Other Modalities

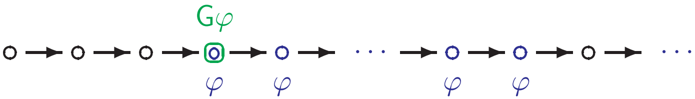
The Future modality

$$F\varphi \quad = \quad \top U \varphi$$



Henceforth modality:

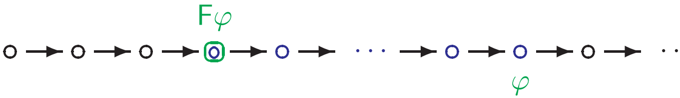
$$w, i \models G\varphi \quad \Longleftrightarrow \quad \forall j \geq i.\ w, j \models \varphi$$



Other Modalities

The Future modality

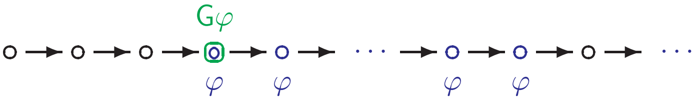
$$F\varphi \quad = \quad \top U \varphi$$



The

Henceforth modality:

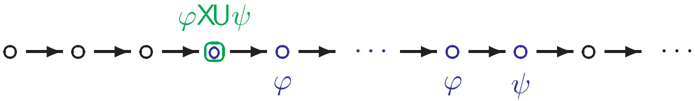
$$G\varphi \quad = \quad \neg F \neg \varphi$$



The Universal Modality

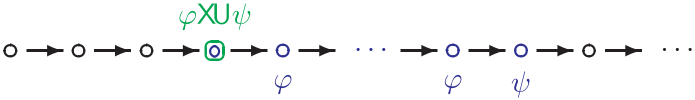
The Next-Until modality:

$$w, i \models \varphi \mathsf{XU} \psi \quad \equiv \quad \exists j > i.\ w, j \models \psi \text{ and } \forall i < k \leq j.\ w, k \models \varphi$$



The Universal Modality

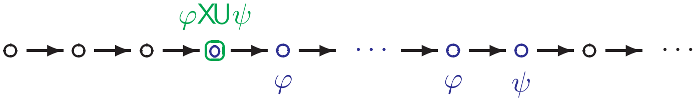
The Next-Until modality:



$$\varphi \mathsf{XU} \psi \quad = \quad \mathsf{X}(\varphi \mathsf{U} \psi)$$

The Universal Modality

The Next-Until modality:



$$\varphi XU \psi \quad = \quad X(\varphi U \psi)$$

Next-Until can express everthing else

$$\begin{aligned} X\varphi &= \bot XU \varphi \\ \varphi U \psi &= \psi \vee (\varphi \wedge \varphi XU \psi) \end{aligned}$$

LTL definable languages

A word satisfies φ if the initial position satisfies φ

$$w \models \varphi \iff w, 1 \models \varphi$$

LTL definable languages

A word satisfies φ if the initial position satisfies φ

$$w \models \varphi \iff w, 1 \models \varphi$$

Formulas define languages. For example,

$$\mathsf{G}(\mathsf{a} \implies \mathsf{F}\mathsf{b})$$

describes words in which there is a b somewhere to the right of every a.

$$b^*(aa^*bb^*)^*$$

Finite/Infinite Words

- LTL formulas are interpreted over both finite and infinite words.

Finite/Infinite Words

- LTL formulas are interpreted over both finite and infinite words.
- Satisfiability of a formula may depend on the class of models.

Finite/Infinite Words

- LTL formulas are interpreted over both finite and infinite words.
- Satisfiability of a formula may depend on the class of models.

$$\mathsf{G}\mathsf{X}\top$$

is satisfied only over infinite words.

$$\mathsf{F}\neg\mathsf{X}\top$$

is satisfied only by finite words.
- The empty word is not a model.

Finite/Infinite Words

- LTL formulas are interpreted over both finite and infinite words.
- Satisfiability of a formula may depend on the class of models.

  $$\mathsf{G}\mathsf{X}\top$$

  is satisfied only over infinite words.

  $$\mathsf{F}\neg\mathsf{X}\top$$

  is satisfied only by finite words.
- The empty word is not a model.

We restrict ourselves to finite word models (for now!).

LTL to FO over Words

- LTL formulas are interpreted at a pair w, i.

LTL to FO over Words

- LTL formulas are interpreted at a pair w, i.
- Translated to FO formulas with a single free variable x.

LTL to FO over Words

- LTL formulas are interpreted at a pair w, i.
- Translated to FO formulas with a single free variable x.

$$
\begin{aligned}
\mathcal{T}(a) &= a(x) \\
\mathcal{T}(X\alpha) &= \exists y.\ (y = x + 1) \wedge \mathcal{T}(\alpha)[y/x] \\
\mathcal{T}(\varphi \mathsf{U} \psi) &= \exists y.\ (y \geq x) \wedge \mathcal{T}(\psi)[y/x] \wedge \\
&\qquad \forall z.(x \leq z < y) \implies \mathcal{T}(\varphi)[z/x]
\end{aligned}
$$

LTL to FO over Words

- LTL formulas are interpreted at a pair w, i.
- Translated to FO formulas with a single free variable x.

$$
\begin{array}{rcl}
\mathcal{T}(a) & = & a(x) \\
\mathcal{T}(X\alpha) & = & \exists y.\ (y = x + 1) \wedge \mathcal{T}(\alpha)[y/x] \\
\mathcal{T}(\varphi \mathsf{U} \psi) & = & \exists y.\ (y \geq x) \wedge \mathcal{T}(\psi)[y/x] \wedge \\
 & & \forall z.(x \leq z < y) \implies \mathcal{T}(\varphi)[z/x]
\end{array}
$$

- $w, i \models \mathcal{T}(\varphi) \quad \iff \quad w, i \models \varphi.$

LTL to FO over Words

- LTL formulas are interpreted at a pair w, i.
- Translated to FO formulas with a single free variable x.

$$\begin{array}{rcl} \mathcal{T}(a) & = & a(x) \\ \mathcal{T}(X\alpha) & = & \exists y.\ (y = x+1) \wedge \mathcal{T}(\alpha)[y/x] \\ \mathcal{T}(\varphi \mathsf{U} \psi) & = & \exists y.\ (y \geq x) \wedge \mathcal{T}(\psi)[y/x] \wedge \\ & & \forall z.(x \leq z < y) \implies \mathcal{T}(\varphi)[z/x] \end{array}$$

- $w, i \models \mathcal{T}(\varphi) \quad \iff \quad w, i \models \varphi$.
- $\mathcal{T}(\varphi)$ uses at the most 3 variables (x, y and z). So, LTL is expressible in FO(3).

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

In particular, there is a satisfiability checking algorithm that runs in time $2^{|\varphi|}$.

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

In particular, there is a satisfiability checking algorithm that runs in time $2^{|\varphi|}$.

Not very different from the best known for propositional formulas.

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

In particular, there is a satisfiability checking algorithm that runs in time $2^{|\varphi|}$.

Not very different from the best known for propositional formulas.

What about FO?

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

In particular, there is a satisfiability checking algorithm that runs in time $2^{|\varphi|}$.

Not very different from the best known for propositional formulas.

Theorem: (Albert Meyer) Satisfiability checking for FO over words is non-elementary.

Complexity of LTL and FO

Satisfiability: Given a formula φ determine whether there is some word w such tha $w \models \varphi$.

Theorem: (Clarke-Sistla) Satisfiability problem for LTL formulas is PSPACE complete.

In particular, there is a satisfiability checking algorithm that runs in time $2^{|\varphi|}$.

Not very different from the best known for propositional formulas.

Theorem: (Albert Meyer) Satisfiability checking for FO over words is non-elementary.

Conclusion: FO seems to be a stronger logic than LTL.

Model Checking

Given a FA A and a formula φ check if every word accepted by the automaton A satisfies the formula φ.

Model Checking

Given a FA A and a formula φ check if every word accepted by the automaton A satisfies the formula φ.

Theorem:(Clarke/Sistla) The Model checking problem for LTL over words is PSPACE-complete.

Model Checking

Given a FA A and a formula φ check if every word accepted by the automaton A satisfies the formula φ.

Theorem:(Clarke/Sistla) The Model checking problem for LTL over words is PSPACE-complete.

In particular

Model Checking

Given a FA A and a formula φ check if every word accepted by the automaton A satisfies the formula φ.

Theorem:(Clarke/Sistla) The Model checking problem for LTL over words is PSPACE-complete.

In particular

Theorem:(Vardi/Wolper) The model-checking problem for LTL is solvable in time $O(|A|.2^{O(|\varphi|)})$.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.
- Gabbay, Pnueli, Shelah and Stavi: Expressive completeness for the future fragment.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.
- Gabbay, Pnueli, Shelah and Stavi: Expressive completeness for the future fragment.
- Other proofs: Cohen, Perrin and Pin, Thomas Wilke.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.
- Gabbay, Pnueli, Shelah and Stavi: Expressive completeness for the future fragment.
- Other proofs: Cohen, Perrin and Pin, Thomas Wilke.

Wilke's proof uses a simple double induction. Has been generalized to Mazurkiewicz traces.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.
- Gabbay, Pnueli, Shelah and Stavi: Expressive completeness for the future fragment.
- Other proofs: Cohen, Perrin and Pin, Thomas Wilke.

Wilke's proof uses a simple double induction. Has been generalized to Mazurkiewicz traces.

Our presentation shall follow a variation of Wilke's proof due to Volker Diekert and Paul Gastin.

Expressive Completeness of LTL

Theorem: (Kamp) LTL is as expressive as FO over words.

- Kamp's logic uses "future" and "past" modalities.
- Gabbay, Pnueli, Shelah and Stavi: Expressive completeness for the future fragment.
- Other proofs: Cohen, Perrin and Pin, Thomas Wilke.

Wilke's proof uses a simple double induction. Has been generalized to Mazurkiewicz traces.

Our presentation shall follow a variation of Wilke's proof due to Volker Diekert and Paul Gastin.

The rest of this talk and the next would be devoted to proving this result.

Star-free Regular Languages

Regular expressions constructed without the $*$ operator:

$$e \quad ::= a \mid e_1 + e_2 \mid \neg e_1 \mid e_1.e_2$$

Star-free Regular Languages

Regular expressions constructed without the $*$ operator:

$$e \quad ::= a \mid e_1 + e_2 \mid \neg e_1 \mid e_1.e_2$$

Theorem:(Schutzenberger) L is aperiodic if and only if it is star-free.

Theorem:(McNaughton and Papert) L is star-free if and only if it is FO expressible.

Star-free Regular Languages

Regular expressions constructed without the $*$ operator:

$$e \quad ::= a \mid e_1 + e_2 \mid \neg e_1 \mid e_1.e_2$$

Theorem:(Schutzenberger) L is aperiodic if and only if it is star-free.

Theorem:(McNaughton and Papert) L is star-free if and only if it is FO expressible.

Question: Can we translate star-free expressions into LTL?

Star-free Regular Languages

Regular expressions constructed without the $*$ operator:

$$e \quad ::= a \mid e_1 + e_2 \mid \neg e_1 \mid e_1.e_2$$

Theorem:(Schutzenberger) L is aperiodic if and only if it is star-free.

Theorem:(McNaughton and Papert) L is star-free if and only if it is FO expressible.

Question: Can we translate star-free expressions into LTL?

How do we put together LTL formulas φ_1 and φ_2 to describe the language $L(\varphi_1).L(\varphi_2)$?

Star-free Regular Languages

Regular expressions constructed without the $*$ operator:

$$e \quad ::= a \mid e_1 + e_2 \mid \neg e_1 \mid e_1.e_2$$

Theorem:(Schutzenberger) L is aperiodic if and only if it is star-free.

Theorem:(McNaughton and Papert) L is star-free if and only if it is FO expressible.

Question: Can we translate star-free expressions into LTL?

How do we put together LTL formulas φ_1 and φ_2 to describe the language $L(\varphi_1).L(\varphi_2)$?

Easy if the decomposition is unambiguous. (eg.) $L_1.c.L_2$ where either L_1 or L_2 is c-free.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.
    - L is Σ^+. Use $\top$.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.
  - L is Σ^+. Use $\top$.
  - L is $\emptyset$. Use $\bot$.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.
  - L is Σ^+. Use $\top$.
  - L is $\emptyset$. Use $\bot$.
- Σ is singleton.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.
  - L is Σ^+. Use $\top$.
  - L is $\emptyset$. Use $\bot$.
- Σ is singleton.
  - L is finite. Easy.

The Proof: Base cases

The proof proceeds via a double induction: On the size of the monoid recognizing L and the size of the alphabet.

The Base Cases:

- M is the trivial monoid.
  - L is Σ^+. Use $\top$.
  - L is $\emptyset$. Use $\bot$.
- Σ is singleton.
  - L is finite. Easy.
  - L is $\{a^i \mid i \geq N\}$. Easy.

The Proof:

Induction Step: Given L over an alphabet Σ recognized by a monoid M such that:

The Proof:

Induction Step: Given L over an alphabet Σ recognized by a monoid M such that:

- if $|M'| < |M|$ then any language recognized by M' is expressible in LTL.

The Proof:

Induction Step: Given L over an alphabet Σ recognized by a monoid M such that:

- if $|M'| < |M|$ then any language recognized by M' is expressible in LTL.
- if L' is a language over an alphabet A with $|A| < |\Sigma|$ recognized by M then L' is expressible in LTL_A.

show that L is expressible in LTL_Σ.

The Proof:

Induction Step: Given L over an alphabet Σ recognized by a monoid M such that:

- if $|M'| < |M|$ then any language recognized by M' is expressible in LTL.
- if L' is a language over an alphabet A with $|A| < |\Sigma|$ recognized by M then L' is expressible in LTL_A.

show that L is expressible in LTL_Σ.

Observation 1: If φ is a LTL_A formula describing the language L and $A \subseteq \Sigma$ then

$$\varphi \wedge \bigwedge_{a \in \Sigma \setminus A} \mathrm{G}\neg \mathrm{a}$$

is a LTL_Σ formula that describes L.

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Pick a letter c such that $h(c) \neq 1$.

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Pick a letter c such that $h(c) \neq 1$.

> *Such a c must exist. Otherwise, L is recognized by the trivial monoid.*

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Pick a letter c such that $h(c) \neq 1$.

> *Such a c must exist. Otherwise, L is recognized by the trivial monoid.*

Decompose L into three disjoint sets:

- L_0 consisting of words of L with no cs.
- L_1 consisting of words of L with exactly one c.
- L_2 consisting of words of L with at least two cs.

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Pick a letter c such that $h(c) \neq 1$.

> *Such a c must exist. Otherwise, L is recognized by the trivial monoid.*

Decompose L into three disjoint sets:

- L_0 consisting of words of L with no cs.
- L_1 consisting of words of L with exactly one c.
- L_2 consisting of words of L with at least two cs.

"No cs", "Exactly 1 c" and "Atleast 2 cs" are expressible in LTL.

Splitting by a letter

Let L be recognized by M via the morphism h as $h^{-1}(X)$.

Pick a letter c such that $h(c) \neq 1$.

> *Such a c must exist. Otherwise, L is recognized by the trivial monoid.*

Decompose L into three disjoint sets:

- L_0 consisting of words of L with no cs.
- L_1 consisting of words of L with exactly one c.
- L_2 consisting of words of L with at least two cs.

"No cs", "Exactly 1 c" and "Atleast 2 cs" are expressible in LTL.

It suffices to show that each of these three languages is LTL expressible.

The Trivial Case: L_0

Let $A = \Sigma \setminus \{c\}$.

The Trivial Case: L_0

Let $A = \Sigma \setminus \{c\}$.

- L_0 is language over a smaller alphabet A, recognized by M via h.

The Trivial Case: L_0

Let $A = \Sigma \setminus \{c\}$.

- L_0 is language over a smaller alphabet A, recognized by M via h.
- So, L_0 is defined by an LTL_A formula φ_0 over A.

The Trivial Case: L_0

Let $A = \Sigma \setminus \{c\}$.

- L_0 is language over a smaller alphabet A, recognized by M via h.
- So, L_0 is defined by an LTL_A formula φ_0 over A.
- By Observation 1, it is expressible in LTL_Σ.

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Why?

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Why?

- If xcy is in the RHS then $h(xcy) = \alpha.h(c).\beta \in X$. Thus $xcy \in L$.

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Why?

- If xcy is in the RHS then $h(xcy) = \alpha.h(c).\beta \in X$. Thus $xcy \in L$.
- Let $w \in L_1$. Therefore, $w = xcy$. Take $\alpha = h(x)$ and $\beta = h(y)$.

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Let $L_\alpha = h^{-1}(\alpha) \cap A^*$ and $L_\beta = h^{-1}(\beta) \cap A^*$.

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Let $L_\alpha = h^{-1}(\alpha) \cap A^*$ and $L_\beta = h^{-1}(\beta) \cap A^*$.

L_1 is a union of languages of the form $L_\alpha.c.L_\beta$ where $L_\alpha, L_\beta \subseteq A^*$ are recognized by M and hence LTL_A (and therefore LTL_Σ) expressible.

The Easy Case: L_1

$$L_1 = \bigcup_{\alpha.h(c).\beta \in X} (h^{-1}(\alpha) \cap A^*).c.(h^{-1}(\beta) \cap A^*)$$

Let $L_\alpha = h^{-1}(\alpha) \cap A^*$ and $L_\beta = h^{-1}(\beta) \cap A^*$.

L_1 is a union of languages of the form $L_\alpha.c.L_\beta$ where $L_\alpha, L_\beta \subseteq A^*$ are recognized by M and hence LTL_A (and therefore LTL_Σ) expressible.
Well, almost! $L_\alpha \cap A^+$ and $L_\beta \cap A^+$ are LTL expressible. We have to deal with ϵ separately

Dealing with Unambiguous Concatenations

We may rewrite $L_\alpha.c.L_\beta$ as

$$A^*.c.L_\beta \;\cap\; L_\alpha.c.\Sigma^*$$

Dealing with Unambiguous Concatenations

We may rewrite $L_\alpha.c.L_\beta$ as

$$A^*.c.L_\beta \ \cap \ L_\alpha.c.\Sigma^*$$

If φ_β is the LTL_Σ formula expressing $L_\beta \cap A^+$ then
$\varphi_1 \ = \ \top \mathsf{U}(\mathsf{c} \wedge \mathsf{X}\varphi_\beta)$ describes $A^*.c.(L_\beta \cap A^+)$.

Dealing with Unambiguous Concatenations

We may rewrite $L_\alpha.c.L_\beta$ as

$$A^*.c.L_\beta \;\cap\; L_\alpha.c.\Sigma^*$$

If φ_β is the LTL_Σ formula expressing $L_\beta \cap A^+$ then $\varphi_1 \;=\; \top \mathsf{U}(\mathsf{c} \wedge \mathsf{X}\varphi_\beta)$ describes $A^*.c.(L_\beta \cap A^+)$.

If $\epsilon \notin L_\beta$ then φ_1 also describes the language $A^*.c.L_\beta$.

Dealing with Unambiguous Concatenations

We may rewrite $L_\alpha.c.L_\beta$ as

$$A^*.c.L_\beta \ \cap \ L_\alpha.c.\Sigma^*$$

If φ_β is the LTL_Σ formula expressing $L_\beta \cap A^+$ then $\varphi_1 \ = \ \top \mathsf{U}(\mathsf{c} \wedge \mathsf{X}\varphi_\beta)$ describes $A^*.c.(L_\beta \cap A^+)$.

If $\epsilon \notin L_\beta$ then φ_1 also describes the language $A^*.c.L_\beta$.

Otherwise, $\varphi_1 \vee \top \mathsf{U}(\mathsf{c} \wedge \neg \mathsf{X}\top)$ describes the language $A^*.c.L_\beta$.

Dealing with Unambiguous Concatenations

We may rewrite $L_\alpha.c.L_\beta$ as

$$A^*.c.L_\beta \;\cap\; L_\alpha.c.\Sigma^*$$

If φ_β is the LTL_Σ formula expressing $L_\beta \cap A^+$ then $\varphi_1 \;=\; \top \mathrm{U}(\mathrm{c} \wedge \mathrm{X}\varphi_\beta)$ describes $A^*.c.(L_\beta \cap A^+)$.

If $\epsilon \notin L_\beta$ then φ_1 also describes the language $A^*.c.L_\beta$.

Otherwise, $\varphi_1 \vee \top \mathrm{U}(\mathrm{c} \wedge \neg \mathrm{X}\top)$ describes the language $A^*.c.L_\beta$.

This case was easy because our modalities walk only to the right and so cannot "stray" to the left. Dealing with $L_\alpha.c.\Sigma^*$ will need a little more work.

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

We cannot use φ_α to describe $L_\alpha.c.\Sigma^*$ since the modalities may walk to the right and cross the c boundary.

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

We "relativize" φ_α to a formula φ'_α which examines the part to the left of the first c and checks if it satisfies φ_α.

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

We “relativize” φ_α to a formula φ'_α which examines the part to the left of the first c and checks if it satisfies φ_α.

Formally, $w \models \varphi'_\alpha$ iff $w = xcy$, $x \in A^+$ and $x \models \varphi_\alpha$.

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

We "relativize" φ_α to a formula φ'_α which examines the part to the left of the first c and checks if it satisfies φ_α.

Formally, $w \models \varphi'_\alpha$ iff $w = xcy$, $x \in A^+$ and $x \models \varphi_\alpha$.

This relativization is defined via structural recursion as follows:

$$
\begin{array}{lcl}
a' & = & a \wedge \mathrm{XF}c \\
(\varphi \wedge \psi)' & = & \varphi' \wedge \psi' \\
(\neg\varphi)' & = & (\neg\varphi') \wedge \neg c \wedge \mathrm{F}c \\
(\varphi \mathrm{XU} \psi)' & = & (\varphi' \wedge \neg c)\mathrm{XU}(\psi' \wedge \neg c)
\end{array}
$$

Unambiguous Concatenation: $L_\alpha.c.\Sigma^*$

Let φ_α be a LTL_A formula describing $L_\alpha \cap A^+$.

We "relativize" φ_α to a formula φ'_α which examines the part to the left of the first c and checks if it satisfies φ_α.

Formally, $w \models \varphi'_\alpha$ iff $w = xcy$, $x \in A^+$ and $x \models \varphi_\alpha$.

This relativization is defined via structural recursion as follows:

$$\begin{aligned} a' &= a \wedge \mathrm{XFc} \\ (\varphi \wedge \psi)' &= \varphi' \wedge \psi' \\ (\neg\varphi)' &= (\neg\varphi') \wedge \neg c \wedge \mathrm{F}c \\ (\varphi \mathrm{XU} \psi)' &= (\varphi' \wedge \neg c)\mathrm{XU}(\psi' \wedge \neg \mathrm{c}) \end{aligned}$$

$\varphi_2 = \varphi'_\alpha$ describes $(L_\alpha \cap A^+).c.\Sigma^*$. If $\epsilon \notin L_\alpha$ then φ_2 also describes $L_\alpha.c.\Sigma^*$. Otherwise, use $\varphi_2 \vee c$.

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

A word w in L_2 is of the form $t_0ct_1ct_2c \ldots t_{k-1}ct_k$ for some $k > 1$, $t_i \in A^*$.

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

A word w in L_2 is of the form $t_0 c t_1 c t_2 c \ldots t_{k-1} c t_k$ for some $k > 1$, $t_i \in A^*$.

Further, $h(w) = h(t_0) h(c t_1 c t_2 c t_3 \ldots t_{k-1} c) h(t_k) \in X$.

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

A word w in L_2 is of the form $t_0 c t_1 c t_2 c \ldots t_{k-1} c t_k$ for some $k > 1$, $t_i \in A^*$.

Further, $h(w) = h(t_0)h(ct_1ct_2ct_3 \ldots t_{k-1}c)h(t_k) \in X$.

Let $\Delta = (cA^*)^+ c$. Then, $L_2 \subseteq A^*.\Delta.A^*$.

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

A word w in L_2 is of the form $t_0 c t_1 c t_2 c \ldots t_{k-1} c t_k$ for some $k > 1$, $t_i \in A^*$.

Further, $h(w) = h(t_0) h(c t_1 c t_2 c t_3 \ldots t_{k-1} c) h(t_k) \in X$.

Let $\Delta = (cA^*)^+ c$. Then, $L_2 \subseteq A^*.\Delta.A^*$.

$$L_2 = \bigcup_{\alpha\beta\gamma \in X} (h^{-1}(\alpha) \cap A^*).(h^{-1}(\beta) \cap \Delta).(h^{-1}(\gamma) \cap A^*)$$

The Interesting Case: L_2

So far, we got away by examining the alphabet. Here we need to examine M and induct on its size.

A word w in L_2 is of the form $t_0 c t_1 c t_2 c \dots t_{k-1} c t_k$ for some $k > 1$, $t_i \in A^*$.

Further, $h(w) = h(t_0) h(c t_1 c t_2 c t_3 \dots t_{k-1} c) h(t_k) \in X$.

Let $\Delta = (cA^*)^+ c$. Then, $L_2 \subseteq A^*.\Delta.A^*$.

$$L_2 = \bigcup_{\alpha\beta\gamma \in X} (h^{-1}(\alpha) \cap A^*).(h^{-1}(\beta) \cap \Delta).(h^{-1}(\gamma) \cap A^*)$$

The first and third components are LTL definable. What about the middle component?

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:
   1. $\sigma^{-1}(K) = L_\beta \cap \Delta$

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:
   1. $\sigma^{-1}(K) = L_\beta \cap \Delta$
   2. K is recognized by a aperiodic monoid smaller than M.

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:
   1. $\sigma^{-1}(K) = L_\beta \cap \Delta$
   2. K is recognized by a aperiodic monoid smaller than M.
   3. the LTL_M formula describing K can be lifted to a formula in LTL_Σ describing $L_\beta \cap \Delta$.

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:
    1. $\sigma^{-1}(K) = L_\beta \cap \Delta$
    2. K is recognized by a aperiodic monoid smaller than M.
    3. the LTL_M formula describing K can be lifted to a formula in LTL_Σ describing $L_\beta \cap \Delta$.

An Outline of the proof

We show that the language $L_\beta \cap \Delta$ is LTL definable as follows:

1. Translate each word in Δ to a word over the alphabet M (actually $h(A^*) \subseteq M$) via a map σ.
2. Construct a language K over M such that:
   1. $\sigma^{-1}(K) = L_\beta \cap \Delta$
   2. K is recognized by a aperiodic monoid smaller than M.
   3. the LTL_M formula describing K can be lifted to a formula in LTL_Σ describing $L_\beta \cap \Delta$.

We use m to denote elements of M when treated as letters and m when they are treated as elements of the monoid M.

The map σ and Language K

The map σ is the obvious one:

$$\sigma(ct_1ct_2\ldots t_{k-2}ct_{k-1}c) \quad = \quad \mathrm{h}(\mathrm{t}_1)\mathrm{h}(\mathrm{t}_2)\ldots\mathrm{h}(\mathrm{t}_{\mathrm{k}-1})$$

The map σ and Language K

The map σ is the obvious one:

$$\sigma(ct_1ct_2\dots t_{k-2}ct_{k-1}c) \quad = \quad \mathrm{h(t_1)h(t_2)\dots h(t_{k-1})}$$

Given the map σ and requirement 2.1, the definition of K is also quite obvious:

$$K \quad = \quad \{\mathrm{m_1m_2\dots m_k} \mid h(c)m_1h(c)m_2\dots h(c)m_kh(c) = \beta\}$$

The map σ and Language K

The map σ is the obvious one:

$$\sigma(ct_1ct_2\ldots t_{k-2}ct_{k-1}c) \quad = \quad \mathrm{h}(\mathrm{t}_1)\mathrm{h}(\mathrm{t}_2)\ldots\mathrm{h}(\mathrm{t}_{\mathrm{k}-1})$$

Given the map σ and requirement 2.1, the definition of K is also quite obvious:

$$K \quad = \quad \{\mathrm{m}_1\mathrm{m}_2\ldots\mathrm{m}_\mathrm{k} \mid h(c)m_1h(c)m_2\ldots h(c)m_kh(c) = \beta\}$$

With these definitions:

$$\sigma^{-1}(K) \quad = \quad \{ct_1ct_2\ldots ct_kc \mid \mathrm{h}(\mathrm{t}_1)\mathrm{h}(\mathrm{t}_2)\ldots\mathrm{h}(\mathrm{t}_\mathrm{k}) \in K\}$$

The map σ and Language K

The map σ is the obvious one:

$$\sigma(ct_1ct_2\dots t_{k-2}ct_{k-1}c) \quad = \quad \mathrm{h(t_1)h(t_2)\dots h(t_{k-1})}$$

Given the map σ and requirement 2.1, the definition of K is also quite obvious:

$$K \quad = \quad \{\mathrm{m_1m_2\dots m_k} \mid h(c)m_1h(c)m_2\dots h(c)m_kh(c) = \beta\}$$

With these definitions:

$$\begin{aligned}\sigma^{-1}(K) &= \{ct_1ct_2\dots ct_kc \mid \mathrm{h(t_1)h(t_2)\dots h(t_k)} \in K\} \\ &= \{ct_1ct_2\dots ct_kc \mid h(c)h(t_1)h(c)h(t_2)\dots h(c)h(t_k)h(c) = \beta\end{aligned}$$

The map σ and Language K

The map σ is the obvious one:

$$\sigma(ct_1ct_2\ldots t_{k-2}ct_{k-1}c) \quad = \quad \mathrm{h(t_1)h(t_2)\ldots h(t_{k-1})}$$

Given the map σ and requirement 2.1, the definition of K is also quite obvious:

$$K \quad = \quad \{\mathrm{m_1m_2\ldots m_k} \mid h(c)m_1h(c)m_2\ldots h(c)m_kh(c) = \beta\}$$

With these definitions:

$$\begin{aligned}\sigma^{-1}(K) &= \{ct_1ct_2\ldots ct_kc \mid \mathrm{h(t_1)h(t_2)\ldots h(t_k)} \in K\}\\ &= \{ct_1ct_2\ldots ct_kc \mid h(c)h(t_1)h(c)h(t_2)\ldots h(c)h(t_k)h(c) = \beta\\ &= L_\beta \cap \Delta \text{ as required by 2.1}\end{aligned}$$

Localizing a Monoid at an element

The following construction is due to Diekert and Gastin.

The Monoid $\mathrm{Loc}_m(M)$: Let M be a monoid and $m \in M$. Then

$$\mathrm{Loc}_m(M) \quad = \quad (mM \cap Mm, \circ, m)$$

where $(xm) \circ (my) \stackrel{\triangle}{=} xmy$.

Localizing a Monoid at an element

The following construction is due to Diekert and Gastin.

The Monoid $\mathrm{Loc}_m(M)$: Let M be a monoid and $m \in M$. Then

$$\mathrm{Loc}_m(M) \quad = \quad (mM \cap Mm, \circ, m)$$

where $(xm) \circ (my) \stackrel{\triangle}{=} xmy$.

- Observe that $xm \circ ym = xm \circ my' = xmy' = xym$. Thus $\circ$ is associative and $m = 1.m$ is the identity w.r.t. $\circ$.

Localizing a Monoid at an element

The following construction is due to Diekert and Gastin.

The Monoid $\mathrm{Loc}_m(M)$: Let M be a monoid and $m \in M$. Then

$$\mathrm{Loc}_m(M) \quad = \quad (mM \cap Mm, \circ, m)$$

where $(xm) \circ (my) \stackrel{\triangle}{=} xmy$.

- Observe that $xm \circ ym = xm \circ my' = xmy' = xym$. Thus $\circ$ is associative and $m = 1.m$ is the identity w.r.t. $\circ$.
- $xm \circ xm \circ \ldots xm \;=\; x^N m$. Thus, $\mathrm{Loc}_m(M)$ is aperiodic whenever M is aperiodic.

Localizing a Monoid at an element

The following construction is due to Diekert and Gastin.

The Monoid $\mathrm{Loc}_m(M)$: Let M be a monoid and $m \in M$. Then

$$\mathrm{Loc}_m(M) \quad = \quad (mM \cap Mm, \circ, m)$$

where $(xm) \circ (my) \triangleq xmy$.

- Observe that $xm \circ ym = xm \circ my' = xmy' = xym$. Thus $\circ$ is associative and $m = 1.m$ is the identity w.r.t. $\circ$.
- $xm \circ xm \circ \ldots xm \; = \; x^N m$. Thus, $\mathrm{Loc}_m(M)$ is aperiodic whenever M is aperiodic.
- $1 \notin \mathrm{Loc}_m(M)$ if $m \neq 1$. This follows from the fact that $1 \neq m'm$ for any $m, m' \neq 1$.

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

Claim: $K \ = \ g^{-1}(\beta)$

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

Claim: $K = g^{-1}(\beta)$

Proof:

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

Claim: $K = g^{-1}(\beta)$

Proof:

- Note that $\beta \in \mathrm{Loc}_{h(c)}(M)$ whenever $h^{-1}(\beta) \cap \Delta \neq \emptyset$.

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

Claim: $K = g^{-1}(\beta)$

Proof:

- Note that $\beta \in \mathrm{Loc}_{h(c)}(M)$ whenever $h^{-1}(\beta) \cap \Delta \neq \emptyset$.
- $g(\mathrm{m_1 m_2 \ldots m_k}) = \beta$ if and only if $h(c)m_1h(c) \circ h(c)m_2h(c) \circ \ldots h(c)m_kh(c) = \beta$ if and only if $h(c)m_1h(c)m_2h(c) \ldots h(c)m_kh(c) = \beta$ if and only if $\mathrm{m_1 m_2 \ldots m_k} \in K$.

A Monoid for K

We now show that the monoid $\mathrm{Loc}_{h(c)}(M)$ accepts the language K.

Let $g : M^* \longrightarrow \mathrm{Loc}_{h(c)}(M)$ be given by $g(\mathrm{m}) = h(c)mh(c)$.

Claim: $K \; = \; g^{-1}(\beta)$

Proof:

- Note that $\beta \in \mathrm{Loc}_{h(c)}(M)$ whenever $h^{-1}(\beta) \cap \Delta \neq \emptyset$.
- $g(\mathrm{m_1 m_2 \ldots m_k}) = \beta$ if and only if $h(c)m_1 h(c) \circ h(c)m_2 h(c) \circ \ldots h(c)m_k h(c) = \beta$ if and only if $h(c)m_1 h(c)m_2 h(c) \ldots h(c)m_k h(c) = \beta$ if and only if $\mathrm{m_1 m_2 \ldots m_k} \in K$.

K is recognized by a smaller monoid and hence there is an LTL_M formula that describes K

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff w = ct_1ct_2c\dots t_{k-1}ct_k, \text{ with } t_i \in A^* \text{ and } \sigma(ct_1ct_2\dots t_{k-1}c) \models \varphi$$

The formula $\varphi^{\#}$ is defined recursively on the structure as follows:

$$\mathrm{m}^{\#} = (c \wedge \mathrm{XFc}) \wedge (\mathrm{X}\psi'_{\mathrm{m}})$$

where ψ_m is the formula in LTL_A describing $h^{-1}(m) \cap A^+$ and ψ'_m is its relativization

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

The formula $\varphi^{\#}$ is defined recursively on the structure as follows:

$$\begin{array}{lcl} \mathrm{m}^{\#} & = & (c \wedge \mathsf{XFc}) \wedge (\mathsf{X}\psi'_{\mathrm{m}}) \\ & & \text{where } \psi_m \text{ is the formula in } LTL_A \text{ describing} \\ & & h^{-1}(m) \cap A^+ \text{ and } \psi'_m \text{ is its relativization} \\ (\varphi_1 \wedge \varphi_2)^{\#} & = & \varphi_1^{\#} \wedge \varphi_2^{\#} \end{array}$$

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^\#$ in LTL_Σ such that

$$w \models \varphi^\# \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

The formula $\varphi^\#$ is defined recursively on the structure as follows:

$$\begin{array}{rcl} \mathrm{m}^\# & = & (c \wedge \mathsf{XF}c) \wedge (\mathsf{X}\psi'_{\mathrm{m}}) \\ & & \text{where } \psi_m \text{ is the formula in } LTL_A \text{ describing} \\ & & h^{-1}(m) \cap A^+ \text{ and } \psi'_m \text{ is its relativization} \\ (\varphi_1 \wedge \varphi_2)^\# & = & \varphi_1^\# \wedge \varphi_2^\# \\ (\neg\varphi)^\# & = & \neg(\varphi^\#) \wedge (c \wedge \mathsf{XF}c) \end{array}$$

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

The formula $\varphi^{\#}$ is defined recursively on the structure as follows:

$$\begin{array}{lcl} \mathrm{m}^{\#} & = & (c \wedge \mathsf{XFc}) \wedge (\mathsf{X}\psi'_{\mathrm{m}}) \\ & & \text{where } \psi_m \text{ is the formula in } LTL_A \text{ describing} \\ & & h^{-1}(m) \cap A^+ \text{ and } \psi'_m \text{ is its relativization} \\ (\varphi_1 \wedge \varphi_2)^{\#} & = & \varphi_1^{\#} \wedge \varphi_2^{\#} \\ (\neg\varphi)^{\#} & = & \neg(\varphi^{\#}) \wedge (c \wedge \mathsf{XFc}) \\ (\mathsf{X}\varphi)^{\#} & = & \mathsf{X}(\neg c\mathsf{U}(\mathsf{c} \wedge \varphi^{\#})) \end{array}$$

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

The formula $\varphi^{\#}$ is defined recursively on the structure as follows:

$$\begin{array}{lcl} \mathrm{m}^{\#} & = & (c \wedge \mathrm{XF}c) \wedge (\mathrm{X}\psi'_{\mathrm{m}}) \\ & & \text{where } \psi_m \text{ is the formula in } LTL_A \text{ describing} \\ & & h^{-1}(m) \cap A^+ \text{ and } \psi'_m \text{ is its relativization} \\ (\varphi_1 \wedge \varphi_2)^{\#} & = & \varphi_1^{\#} \wedge \varphi_2^{\#} \\ (\neg\varphi)^{\#} & = & \neg(\varphi^{\#}) \wedge (c \wedge \mathrm{XF}c) \\ (\mathrm{X}\varphi)^{\#} & = & \mathrm{X}(\neg c \mathrm{U}(c \wedge \varphi^{\#})) \\ (\varphi_1 \mathrm{U} \varphi_2)^{\#} & = & (c \implies \varphi_1^{\#})\mathrm{U}(c \wedge \varphi_2^{\#}) \end{array}$$

Lifting the formula for K

We show that for any formula φ in LTL_M, there is a formula $\varphi^{\#}$ in LTL_Σ such that

$$w \models \varphi^{\#} \iff \begin{array}{l} w = ct_1ct_2c\ldots t_{k-1}ct_k, \text{ with } t_i \in A^* \\ \text{and } \sigma(ct_1ct_2\ldots t_{k-1}c) \models \varphi \end{array}$$

The formula $\varphi^{\#}$ is defined recursively on the structure as follows:

$$\begin{array}{lcl} \mathrm{m}^{\#} & = & (c \wedge \mathsf{XFc}) \wedge (\mathsf{Xc} \vee \mathsf{X}\psi'_{\mathrm{m}}) \\ & & \text{ where } \psi_m \text{ is the formula in } LTL_A \text{ describing} \\ & & h^{-1}(m) \cap A^+ \text{ and } \psi'_m \text{ is its relativization} \\ (\varphi_1 \wedge \varphi_2)^{\#} & = & \varphi_1^{\#} \wedge \varphi_2^{\#} \\ (\neg\varphi)^{\#} & = & \neg(\varphi^{\#}) \wedge (c \wedge \mathsf{XFc}) \\ (\mathsf{X}\varphi)^{\#} & = & \mathsf{X}(\neg c \mathsf{U}(\mathsf{c} \wedge \varphi^{\#})) \\ (\varphi_1 \mathsf{U} \varphi_2)^{\#} & = & (c \implies \varphi_1^{\#})\mathsf{U}(\mathsf{c} \wedge \varphi_2^{\#}) \end{array}$$

Combining the Three parts

The formula describing $(L_\alpha \cap A^*).(L_\beta \cap \Delta).(L_\gamma \cap A^*)$ is the conjunction of the formulas describing the following languages.

1. $(L_\alpha \cap A^*).(cA^*)^+.c.A^*$.

2. $A^*.(cA^*)^+.c.(L_\gamma \cap A^*)$.

3. $A^*.((L_\beta \cap \Delta).A^*)$.

Combining the Three parts

The formula describing $(L_\alpha \cap A^*).(L_\beta \cap \Delta).(L_\gamma \cap A^*)$ is the conjunction of the formulas describing the following languages.

1. $(L_\alpha \cap A^*).(cA^*)^+.c.A^*$.

2. $A^*.(cA^*)^+.c.(L_\gamma \cap A^*)$.

3. $A^*.((L_\beta \cap \Delta).A^*)$.

Combining the Three parts

The formula describing $(L_\alpha \cap A^*).(L_\beta \cap \Delta).(L_\gamma \cap A^*)$ is the conjunction of the formulas describing the following languages.

1. $(L_\alpha \cap A^*).(cA^*)^+.c.A^*$.

$$\varphi' \wedge (\mathsf{F}(\mathsf{c} \wedge \mathsf{XFc}))$$

2. $A^*.(cA^*)^+.c.(L_\gamma \cap A^*)$.

3. $A^*.((L_\beta \cap \Delta).A^*)$.

Combining the Three parts

The formula describing $(L_\alpha \cap A^*).(L_\beta \cap \Delta).(L_\gamma \cap A^*)$ is the conjunction of the formulas describing the following languages.

1. $(L_\alpha \cap A^*).(cA^*)^+.c.A^*$.

$$\varphi' \wedge (\mathsf{F}(\mathsf{c} \wedge \mathsf{XFc}))$$

2. $A^*.(cA^*)^+.c.(L_\gamma \cap A^*)$.

$$\mathsf{F}(\mathsf{c} \wedge \mathsf{XF}(\mathsf{c} \wedge \mathsf{F}(\mathsf{c} \wedge \neg(\mathsf{XFc}) \wedge \mathsf{X}\varphi)))$$

3. $A^*.((L_\beta \cap \Delta).A^*)$.

Combining the Three parts

The formula describing $(L_\alpha \cap A^*).(L_\beta \cap \Delta).(L_\gamma \cap A^*)$ is the conjunction of the formulas describing the following languages.

1. $(L_\alpha \cap A^*).(cA^*)^+.c.A^*$.

$$\varphi' \wedge (\mathsf{F}(\mathsf{c} \wedge \mathsf{XFc}))$$

2. $A^*.(cA^*)^+.c.(L_\gamma \cap A^*)$.

$$\mathsf{F}(\mathsf{c} \wedge \mathsf{XF}(\mathsf{c} \wedge \mathsf{F}(\mathsf{c} \wedge \neg(\mathsf{XFc}) \wedge \mathsf{X}\varphi)))$$

3. $A^*.((L_\beta \cap \Delta).A^*)$.

$$\neg c\mathsf{U}(\mathsf{c} \wedge \varphi^{\#})$$