A Lean Mechanization of Reversible Occurrence Nets

Hernán Melgratti
CONICET & Universidad de Buenos Aires
Director

Daniel Dávalos
Universidad de Buenos Aires
Doctorando

Seminarios FormAL

28 de Octubre, 2026







Introducción

- Goal: **formalizar las redes de Petri reversibles y ocurrentes** en un asistente de pruebas.
  - Nos enfocamos principalmente en las redes reversibles del paper Reversing Place Transition Nets (Melgratti, Mezzina, Ulidowski).
  - Probaremos varias propiedades importantes en teoría concurrente, incluyendo el lema parabólico.

Introducción

- Goal: **formalizar las redes de Petri reversibles y ocurrentes** en un asistente de pruebas.
  - Nos enfocamos principalmente en las redes reversibles del paper Reversing Place Transition Nets (Melgratti, Mezzina, Ulidowski).
  - Probaremos varias propiedades importantes en teoría concurrente, incluyendo el lema parabólico.
- En particular, elegimos **Lean Prover** como asistente de pruebas:
  - full programming language
  - interactivo theorem prover

Definiciones preliminares

Net

Definiciones preliminares

Una red (**net**) N es una tupla $(\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$ donde

- α es el conjunto de lugares (**places**),
- β es el conjunto de transiciones (**transitions**), y
- ${}^\bullet{-}_N, {-}^\bullet_N : \beta \to \mathbb{N}^\alpha$ son funciones que asignan origen y destino para cada transición, siempre que ${}^\bullet\mathtt{t} \neq \emptyset$ y $\mathtt{t}^\bullet \neq \emptyset$ para cada $\mathtt{t} \in \beta$.

Net

Definiciones preliminares

Una red (**net**) N es una tupla $(\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$ donde

- α es el conjunto de lugares (**places**),
- β es el conjunto de transiciones (**transitions**), y
- ${}^\bullet{-}_N, {-}^\bullet_N : \beta \to \mathbb{N}^\alpha$ son funciones que asignan origen y destino para cada transición, siempre que ${}^\bullet\mathtt{t} \neq \emptyset$ y $\mathtt{t}^\bullet \neq \emptyset$ para cada $\mathtt{t} \in \beta$.

Net.lean

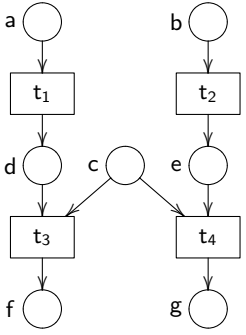
```
structure Net (α : Type) (β : Type) where
 pre: β → Multiset α
 post: β → Multiset α
 cons_prod: ∀ t, pre t ≠ ∅ ∧ post t ≠ ∅
```

Ejemplo

Net: $N = (\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$

$$\alpha = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}\}$$

$$\beta = \{\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, \mathsf{t}_4\}$$

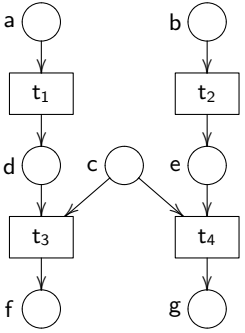


Ejemplo

Net: $N = (\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$

$$\alpha = \{a, b, c, d, e, f, g\}$$

$$\beta = \{t_1, t_2, t_3, t_4\}$$



Net.lean

```
inductive Pl | a | b | c | d | e | f | g
inductive Tr | t₁ | t₂ | t₃ | t₄

def pre₁ : Tr → Multiset Pl
 | t₁ => {a}
 | t₂ => {b}
 | t₃ => {d, c}
 | t₄ => {c, e}

def post₁ : Tr → Multiset Pl
 | t₁ => {d}
 | t₂ => {e}
 | t₃ => {f}
 | t₄ => {g}

def cons_prod₁ : ∀ t : Tr, pre ≠ ∅ ∧ post ≠ ∅ :=
  ...

def N1 : Net Pl Tr := ⟨pre₁, post₁, cons_prod₁⟩
```

Petri Nets

Definición

- Un *marking* m de una red N es un multiconjunto sobre el conjunto de places, i.e., $m \in \mathbb{N}^{\alpha}$.
- Una **red de Petri** o **marked net** es un par (N, m) donde N es una red y m es un marking de N.

Petri Nets

Definición

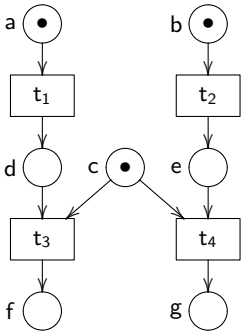
- Un *marking* m de una red N es un multiconjunto sobre el conjunto de places, i.e., $m \in \mathbb{N}^{\alpha}$.
- Una **red de Petri** o **marked net** es un par (N, m) donde N es una red y m es un marking de N.

Net.lean

```
1 structure MarkedNet α β extends Net α β where
2  m₀ : Multiset α
```

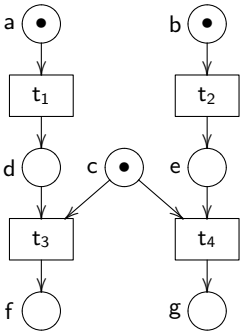
Ejemplo

Marked net: $(N, \mathsf{a} + \mathsf{b} + \mathsf{c})$



Ejemplo

Marked net: $(N, \mathsf{a} + \mathsf{b} + \mathsf{c})$



Net.lean

```
1 def M1 : MarkedNet Pl Tr := ⟨N1, {a, b, c}⟩
```

Nets

Preset y postset

Pre- and postsets para places

- **Preset:** ${}^\bullet a := \{t \mid a \in t^\bullet\}$
- **Postset:** $a^\bullet := \{t \mid a \in {}^\bullet t\}$

Nets

Preset y postset

Pre- and postsets para places

- **Preset:** ${}^\bullet \mathsf{a} := \{\mathsf{t} \mid \mathsf{a} \in \mathsf{t}^\bullet\}$
- **Postset:** $\mathsf{a}^\bullet := \{\mathsf{t} \mid \mathsf{a} \in {}^\bullet\mathsf{t}\}$

Net.lean

```
1  def preset_p (N : Net α β) (p : α) : Set β :=
2    {t | p ∈ N.post t}
3
4  def postset_p (N : Net α β) (p : α) : Set β :=
5    {t | p ∈ N.pre t}
6
7  def preset_t (N : Net α β) (t : β) :=
8    N.pre t
9
10 def postset_t (N : Net α β) (t : β):=
11   N.post t
```

Introducimos las notaciones $\bullet\{\!\{\mathtt{N}\}\!\}\mathtt{x}$ y $\mathtt{x}\bullet\{\!\{\mathtt{N}\}\!\}$ para unificar la sintaxis de presets y postsets sobre lugares y transiciones en Lean.

Semántica

Ejecuciones

- Una transición $\mathtt{t}$ está habilitada (*enabled*) en un marking m, denotado por $m[\![\mathtt{t}]\!]$, si ${}^\bullet\mathtt{t} \leq m$.
- La ejecución (*firing*) de una transición habilitada $\mathtt{t}$ en un marking m es denotada por $m[\![\mathtt{t}\rangle m'$ si $m' = m - {}^\bullet\mathtt{t} + \mathtt{t}^\bullet$.

Semántica

Ejecuciones

- Una transición $\mathtt{t}$ está habilitada (*enabled*) en un marking m, denotado por $m[\![\mathtt{t}]\!]$, si ${}^\bullet\mathtt{t} \leq m$.
- La ejecución (*firing*) de una transición habilitada $\mathtt{t}$ en un marking m es denotada por $m[\![\mathtt{t}\rangle m'$ si $m' = m - {}^\bullet\mathtt{t} + \mathtt{t}^\bullet$.

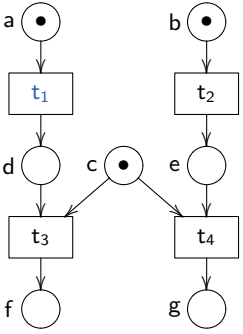
Net.lean

```
1  def is_enabled (N : Net α β)
2      (m : Multiset α) (t : β) : Prop :=
3    •{{N}} t ≤ m
4
5  notation m "[[" t "]]{{" N "}}" =>
6                          is_enabled ↑N m t
7
8  def marking_after_firing (N: Net α β)
9      (m : Multiset α) (_: m[[t]]{{N}}) :=
10   m - •{{N}} t + t•{{N}}
```

`m[[e⟩{{N}}m'` si el firing `t` habilitada por `e :m[[t]]{{N}}` produce `m'`

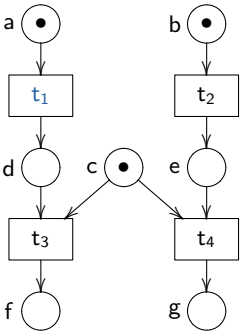
Ejemplo

Firing: $(a + b + c)[\![t_1\rangle_{\{\!\{N\}\!\}}(d + b + c)$



Ejemplo

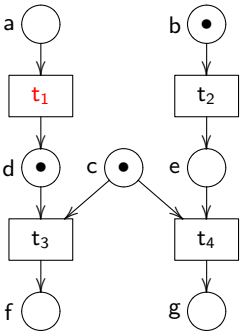
Firing: $(a + b + c)[\![t_1\rangle_{\{\!\{N\}\!\}}(d + b + c)$




```
1  --Net N1 fue definida previamente
2
3  lemma et1 : {a, b, c}⟦t1⟧{{N2}} := by
4    unfold is_enabled preset_t N2 Net.pre pre2
5    simp
6
7
8  .
```

Ejemplo

Firing: $(a + b + c)[\![t_1\rangle\!\rangle_{\{\!\{N\}\!\}}(d + b + c)$




```
1  --Net N1 fue definida previamente
2
3  lemma et1 : {a, b, c}⟦t1⟧{{N2}} := by
4    unfold is_enabled preset_t N2 Net.pre pre2
5    simp
6
7  example : {a,b,c} ⟦ et1⟩{{N1}} {d,b,c} := by
8    exact Multiset.add_comm ({b} + {c}) {d}
```

Semántica

Firing sequence

Definimos por $s = \mathsf{t}_1; \mathsf{t}_2; \ldots; \mathsf{t}_n$ una secuencia de transiciones. El *firing sequence* $m[\![s\rangle\!\rangle m_n$ se define inductivamente por

- $m[\![\varepsilon\rangle\!\rangle m$ if $n = 0$
- $m[\![\mathsf{t}_1\rangle\!\rangle m_1 \wedge m_1[\![\mathsf{t}_2; \cdots; \mathsf{t}_n\rangle\!\rangle m_n$ si $n > 0$

$s = \varepsilon$ es la secuencia vacía.

Semántica

Firing sequence

Definimos por $s = \mathtt{t}_1; \mathtt{t}_2; \ldots; \mathtt{t}_n$ una secuencia de transiciones. El *firing sequence* $m[\![s\rangle\!\rangle m_n$ se define inductivamente por

- $m[\![\varepsilon\rangle m$ if $n = 0$
- $m[\![\mathtt{t}_1\rangle m_1 \wedge m_1[\![\mathtt{t}_2; \cdots; \mathtt{t}_n\rangle\!\rangle m_n$ si $n > 0$

$s = \varepsilon$ es la secuencia vacía.

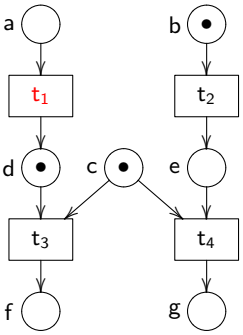
Nets.lean

```
1 inductive firing_sequence : Net α β → Multiset α →
      List β → Multiset α → Prop
2 | empty m :
3     firing_sequence _ m [] m
4 | step {N : Net α β} {t} {ts}
5     (et : m [[t]⟩{N})
6     (f : m [[et⟩{N} m')
7     (fs : firing_sequence N m' ts m''):
8     firing_sequence N m (t :: ts) m''
```

`m[[s⟩⟩{N}m'` si la secuencia `s` se ejecuta desde `m` y produce `m'`.

Ejemplo

Firing sequence: $(a + b + c)[\![t_1; t_2; t_3]\!\rangle_{\{\!\{N\}\!\}}(e + f)$

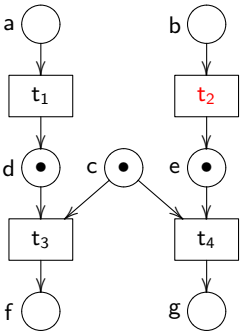


Nets.lean

```
example : {a,b,c} ⟦ [t₁, t₂, t₃]⟩⟩ {{N₂}} {e,f} := by
  obtain et₁ : {a, b, c}⟦t₁⟧{{N₂}}:= ...
  obtain fir₁ : {a, b, c} ⟦et₁⟩{{N₂}} {d, b, c} := ...
  sorry
.
```

Ejemplo

Firing sequence: $(a + b + c)[\![t_1; t_2; t_3\rangle\!\rangle_{\{\!\{N\}\!\}}(e + f)$

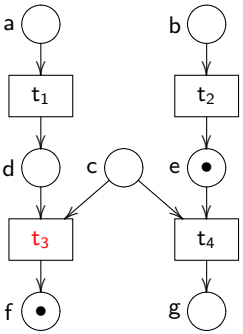


Nets.lean

```
example : {a,b,c} ⟦ [t₁, t₂, t₃]⟩⟩ {{N₂}} {e,f} := by
  obtain et₁ : {a, b, c}⟦t₁⟧{{N₂}}:= ...
  obtain fir₁ : {a, b, c} ⟦et₁⟩{{N₂}} {d, b, c} := ...
  obtain et₂ : {d,b,c} ⟦t₂⟧{{N₂}}:= ...
  obtain fir₂ : {d,b,c} ⟦et₂⟩{{N₂}} {d,c,e} := ...
  sorry
```

Ejemplo

Firing sequence: $(a + b + c)[\![t_1; t_2; t_3]\!\rangle_{\{\!\{N\}\!\}}(e + f)$

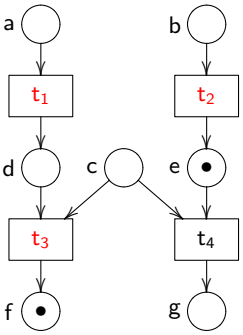


Nets.lean

```
example : {a,b,c} ⟦ [t₁, t₂, t₃]⟩⟩ {{N₂}} {e,f} := by
  obtain et₁ : {a, b, c}⟦t₁⟧{{N₂}}:= ...
  obtain fir₁ : {a, b, c} ⟦et₁⟩{{N₂}} {d, b, c} := ...
  obtain et₂ : {d,b,c} ⟦t₂⟧{{N₂}}:= ...
  obtain fir₂ : {d,b,c} ⟦et₂⟩{{N₂}} {d,c,e} := ...
  obtain et₃ : {d,c,e}⟦t₃⟧{{N₂}} := ...
  obtain fir₃ : {d,c,e}⟦et₃⟩{{N₂}} {e,f} := ...
.
```

Ejemplo

Firing sequence: $(a + b + c)[\![t_1 ; t_2 ; t_3]\!\rangle_{\{\!\{N\}\!\}}(e + f)$



Nets.lean

```
example : {a,b,c} ⟦ [t₁, t₂, t₃]⟩⟩ {{N₂}} {e,f} := by
  obtain et₁ : {a, b, c}⟦t₁⟧{N₂}:= ...
  obtain fir₁ : {a, b, c} ⟦et₁⟩{N₂} {d, b, c} := ...
  obtain et₂ : {d,b,c} ⟦t₂⟧{N₂}:= ...
  obtain fir₂ : {d,b,c} ⟦et₂⟩{N₂} {d,c,e} := ...
  obtain et₃ : {d,c,e}⟦t₃⟧{N₂} := ...
  obtain fir₃ : {d,c,e}⟦et₃⟩{N₂} {e,f} := ...
  apply firing_sequence.step et₁ fir₁ <|
    .step et₂ fir₂ <|
    .step et₃ fir₃ <|
    .empty {e,f}
```

Concurrencia

Redes de ocurrencia

Dependencia causal

Causal Dependency

La *relación de dependencia inmediata* $<_{\{\!\{N\}\!\}} \subseteq (\alpha \cup \beta) \times (\alpha \cup \beta)$ se define por el conjunto

$$\{(x, y) \mid (y \in \beta \wedge x \in {}^\bullet y) \vee (x \in \beta \wedge y \in x^\bullet)\}$$

- **Causal relation:** clausura transitiva

$$\prec_{\{\!\{N\}\!\}} := <^*_{\{\!\{N\}\!\}}$$

- **Causal preorder:** clausura reflexo-transitiva

$$\preceq_{\{\!\{N\}\!\}} := \prec_{\{\!\{N\}\!\}} \cup \{(x, x) \mid x \in \alpha \cup \beta\}$$

Redes de ocurrencia

Dependencia causal

Causal Dependency

La *relación de dependencia inmediata* $<_{\{\!|N|\!\}} \subseteq (\alpha \cup \beta) \times (\alpha \cup \beta)$ se define por el conjunto

$$\{(x, y) \mid (y \in \beta \wedge x \in {}^\bullet y) \vee (x \in \beta \wedge y \in x^\bullet)\}$$

- **Causal relation:** clausura transitiva

$$\prec_{\{\!|N|\!\}} := <^*_{\{\!|N|\!\}}$$

- **Causal preorder:** clausura reflexo-transitiva

$$\preceq_{\{\!|N|\!\}} := \prec_{\{\!|N|\!\}} \cup \{(x, x) \mid x \in \alpha \cup \beta\}$$

Occurrence.lean

```
1  def immediate (N: Net α β) : (α ⊕ β) → (α ⊕ β) →
       Prop
2    | inl p, inr t  =>  p ∈ •{|N|} t
3    | inr t, inl p  =>  p ∈ t •{|N|}
4    | _, _          =>  false
5
6  notation l "<{|"N"|}" r => immediate N l r
7
8  def causal (N : Net α β) :
9      (α ⊕ β) → (α ⊕ β) → Prop :=
10   TransGen (immediate N)
11
12 notation l "≺{|"N"|}" r => causal ↑N l r
13
14 def preorder (N : Net α β) :
15     (α ⊕ β) → (α ⊕ β) → Prop :=
16   ReflTransGen (immediate N)
17
18 notation l "≼{|"N"|}" r => preorder ↑N l r
```

Redes de ocurrencia

Conflicto

- Dos transiciones diferentes t y t' están en *conflicto inmediato* si ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Lo denotamos por $t \#_0 t$.
- $x, y \in \alpha \cup \beta$ están en *conflicto* si existen transiciones t, t' tales que: $t \preceq x$, $t' \preceq y$ y $t \#_0 t'$.

Redes de ocurrencia

Conflicto

- Dos transiciones diferentes t y t' están en *conflicto inmediato* si ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Lo denotamos por $t \#_0 t$.
- $x, y \in \alpha \cup \beta$ están en *conflicto* si existen transiciones t, t' tales que: $t \preceq x$, $t' \preceq y$ y $t \#_0 t'$.

Occurrence.lean

```
1  def immediate_conflict (N : Net α β) (t t' : β) : Prop :=
2    t ≠ t' ∧ ¬ (Disjoint (•{{N}} t) (•{{N}} t'))
3
4  notation l "#₀{{" N "}}" r => immediate_conflict ↑N l r
5
6  def conflict (N : Net α β) (x y : α ⊕ β) : Prop :=
7    ∃ t t' : β, inr t ≼{{N}} x ∧ inr t' ≼{{N}} y ∧ t #₀{{N}} t'
8
9  notation l "#{{" N "}}" r => conflict ↑N l r
```

Redes de ocurrencia

Definamos las **redes de ocurrencia**:

Occurrence net

Una red marcada (N, m_0) es de *ocurrencia* si satisface las siguientes condiciones:

1. N es acíclica;
2. no hay autoconflicto, i.e., $\neg(\mathtt{t} \#_{\{\!\{N}\!\}} \mathtt{t})$ para cada transición $\mathtt{t}$;
3. no hay backward conflict;
4. N es 1-safe, i.e., cada marking es un conjunto;
5. el marking inicial es el conjunto de lugares iniciales, i.e., m_0 es minimal.

Redes de ocurrencia

Definamos las **redes de ocurrencia**:

Occurrence net

Una red marcada (N, m_0) es de *ocurrencia* si satisface las siguientes condiciones:

1. N es acíclica;
2. no hay autoconflicto, i.e., $\neg(\mathtt{t} \#_{\{\!\{N}\!\}} \mathtt{t})$ para cada transición $\mathtt{t}$;
3. no hay backward conflict;
4. N es 1-safe, i.e., cada marking es un conjunto;
5. el marking inicial es el conjunto de lugares iniciales, i.e., m_0 es minimal.

Redes de ocurrencia

Definamos las **redes de ocurrencia**:

Occurrence net

Una red marcada (N, m_0) es de *ocurrencia* si satisface las siguientes condiciones:

1. N es acíclica;
2. no hay autoconflicto, i.e., $\neg(\mathtt{t} \#_{\{\!\{N}\!\}} \mathtt{t})$ para cada transición $\mathtt{t}$;
3. no hay backward conflict;
4. para cada transición $\mathtt{t}$, $\bullet\mathtt{t}$ y $\mathtt{t}\bullet$ son conjuntos.
5. el marcado inicial es el conjunto de lugares iniciales, i.e., m_0 es minimal.

Redes de ocurrencia

Definamos las **redes de ocurrencia**:

Occurrence net

Una red marcada (N, m_0) es de *ocurrencia* si satisface las siguientes condiciones:

1. N es acíclica;
2. no hay autoconflicto, i.e., $\neg(\mathtt{t} \#_{\{\!\{N}\!\}} \mathtt{t})$ para cada transición $\mathtt{t}$;
3. no hay backward conflict;
4. para cada transición $\mathtt{t}$, $\bullet\mathtt{t}$ y $\mathtt{t}\bullet$ son conjuntos.
5. el marcado inicial es el conjunto de lugares iniciales, i.e., m_0 es minimal.

Occurrence.lean

```
1  structure is_occurrence (N : Net α β) where
2    acyclicity : acyclic N
3    no_self_conflict :
4      ∀ t : β, ¬inr t #{{N}} inr t
5    no_backward_conflict :
6      ∀ a : α, ¬backward_conflict N a
7    set_preset :
8      ∀ t : β,  Nodup (•{{N}} t)
9    set_postset :
10     ∀ t : β, Nodup (t •{{N}})
11
12 structure is_marked_occurrence (M : MarkedNet α β)
       : Prop where
13   occurrence : is_occurrence M.toNet
14   initial_set : Nodup M.m₀
15   inital_minimal : ∀ {x}, x ∈ M.m₀ → x ∈ minimal
         M.toNet
```

Occurrence

Una red marcada de ocurrencia es safe

Occurrence.lean

```
/-- La definicion de safe se encuentra en Nets.lean -/
def safe (M : MarkedNet α β) : Prop :=
  ∀ m, M ⇝ m → Nodup m

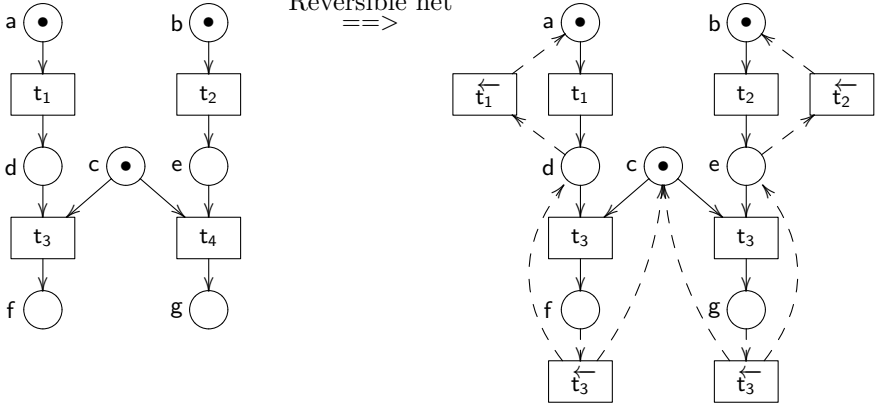
theorem occ_net_safe (MO : is_marked_occurrence M) : safe M :=
/-- La prueba es omitida -/
```

`M ⇝ m` es verdadera si `m ∈ reach(M)`.

Redes reversibles

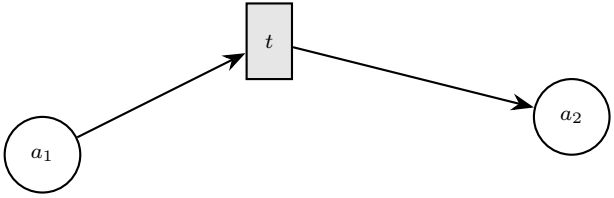
Redes reversibles

¿Cómo obtener una red reversible?



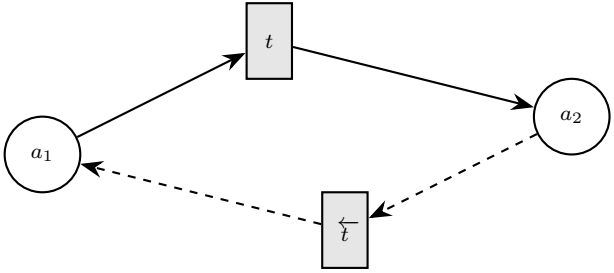
Redes reversibles

Transition Type



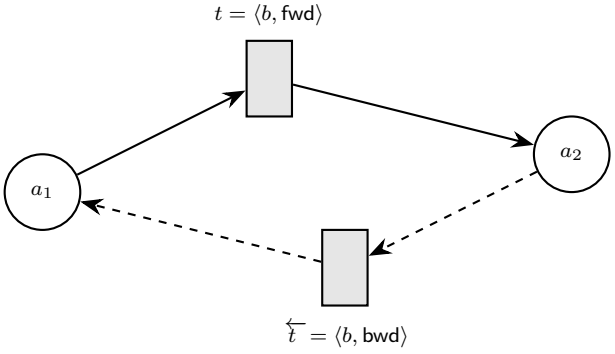
Redes reversibles

Transition Type



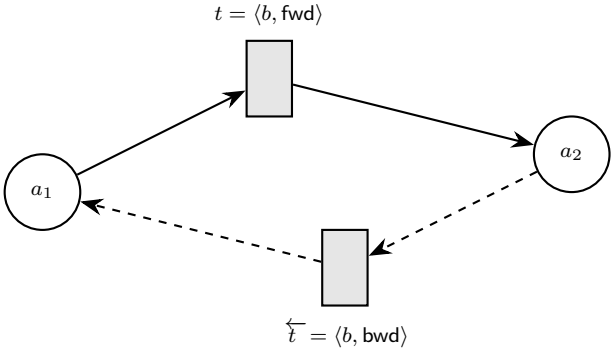
Redes reversibles

Transition Type



Redes reversibles

Transition Type



ReversibleOccurrence.lean

```
1  inductive TransitionType
2  | fwd : TransitionType
3  | bwd : TransitionType
4
5  abbrev Transition (β : Type) :=
       β×TransitionType
```

Redes reversibles

Definición

Una *red de ocurrencia reversible* es una red $N = (\alpha, \beta \uplus \{ \overleftarrow{t} \mid t \in \beta\}, {}^\bullet_, _^\bullet)$ definida por las siguientes condiciones:

1. $(\alpha, \beta, ({}^\bullet_)_{|\beta}, (_^\bullet)_{|\beta})$ es una red de ocurrencia, donde $f_{|\beta}$ denota la función restricción f del dominio β.
2. $\forall t,\ \exists t',\ {}^\bullet t = t'^\bullet \wedge t^\bullet = {}^\bullet t'$.

Redes reversibles

Definición

Una *red de ocurrencia reversible* es una red $N = (\alpha, \beta \uplus \{ \overleftarrow{t} \mid t \in \beta\}, {}^\bullet_, _^\bullet)$ definida por las siguientes condiciones:

1. $(\alpha, \beta, ({}^\bullet_)_{|\beta}, (_^\bullet)_{|\beta})$ es una red de ocurrencia, donde $f_{|\beta}$ denota la función restricción f del dominio β.
2. $\forall t,\ \exists t',\ {}^\bullet t = t'^\bullet \wedge t^\bullet = {}^\bullet t'$.

ReversibleOccurrence.lean

```
1  structure Reversible α β
2      extends Net α (Transition β) where
3    welldef (t t' : Transition β) :
4      t ⟝ t' → •{{toNet}}t = t'•{{toNet}}
5
6  structure ReversibleOccurrence α β extends Reversible
       α β where
7    isOccurrence :
8      is_occurrence (fwd_subnet toNet)
9
10 structure MarkedReversibleOccurrence α β extends
       Reversible α β where
11   m₀ : Multiset α
12   isFwdMarkedOccurrence :
13     is_marked_occurrence ⟨fwd_subnet toNet, m₀⟩
```

- `fwd_subnet`: es la función que toma una red de tipo `N : α (β×TransitionType)` y retorna su red forward.

Trace Equivalence

Equivalencia en firing sequences:
es la menor congruencia $\asymp$ en secuencias de trasiciones cerradas bajo las siguientes reglas:

(i) $t; t' \asymp t'; t$ if $t \; co \; t'$,

(ii) $t; \overleftarrow{t} \asymp \varepsilon$, y

(iii) $\overleftarrow{t}; t \asymp \varepsilon$.

Trace Equivalence

Equivalencia en firing sequences: es la menor congruencia $\asymp$ en secuencias de trasiciones cerradas bajo las siguientes reglas:

(i) $t; t' \asymp t'; t$ if t *co* t',

(ii) $t; \overleftarrow{t} \asymp \varepsilon$, y

(iii) $\overleftarrow{t}; t \asymp \varepsilon$.

Equivalence.lean

```
1  inductive TrEq (R : Reversible α β) :
2    List (Transition β) → List (Transition β) → Prop
3  | swap : (t t' : Transition β) →
4    Disjoint (t•{|R|}) (•{|R|} t') → Disjoint (t'•{|R|}) (•{|R|} t) →
5    TrEq R [t, t'] [t', t]
6  | canc_l : (t t' : Transition β) → t ↚ t' →
7    TrEq R [t, t'] []
8  | canc_r : (t t' : Transition β) → t ↛ t' →
9    TrEq R [] [t, t']
10 ...
11 notation:50 s₁:51 "≍{|" R "|} " s₂:51 => TrEq R s₁ s₂
12
13 /--We omit the proofs which show that ≍ is an equivalence
       relation.-/
14
15 lemma TrEq_congruence (h : s₁ ≍{|R|} s₂) (h' : s₃ ≍{|R|} s₄) :
16     (s₁ ++ s₃) ≍{|R|} (s₂ ++ s₄)
```

Parabolic lemma

Enunciado formal

Lemma (Parabolic)

Let R be a reversible occurrence net, m a reachable marking, and $m[\![\omega\rangle\!\rangle_{\{\!\{R\}\!\}} m'$ a firing sequence. Then there exist two sequences of transitions, ω' and ω'', such that ω' consists only of backward transitions, ω'' consists only of forward transitions, $\omega \asymp_{\{\!\{R\}\!\}} \omega';\omega''$, and $m[\![\omega';\omega''\rangle\!\rangle_{\{\!\{R\}\!\}} m'$.

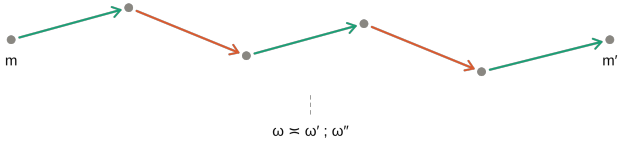
Parabolic lemma

Enunciado formal

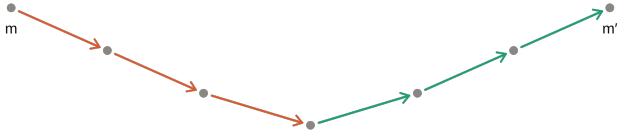
Lemma (Parabolic)

Let R be a reversible occurrence net, m a reachable marking, and $m[\![\omega\rangle\!\rangle_{\{\!\{R\}\!\}} m'$ a firing sequence. Then there exist two sequences of transitions, ω' and ω'', such that ω' consists only of backward transitions, ω'' consists only of forward transitions, $\omega \asymp_{\{\!\{R\}\!\}} \omega';\omega''$, and $m[\![\omega';\omega''\rangle\!\rangle_{\{\!\{R\}\!\}} m'$.

ω: secuencia arbitraria (hacia adelante y hacia atrás mezcladas)



ω′ ; ω″: primero solo hacia atrás, luego solo hacia adelante



Parabolic lemma

Formalización en Lean

Parabolic.lean

```
variable {RO : ReversibleOccurrence α β}

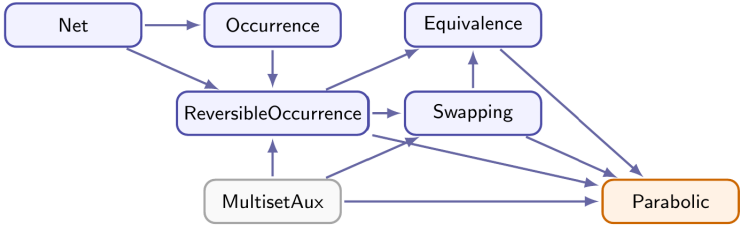
def parabolic_decomp (s s' s'' : List (Transition β)) (m m' : Multiset α) :=
  ⋪s' ∧ ⋫s'' ∧ m ⟦s' ++ s''⟩⦃RO⦄ m' ∧ s ≍ ⦃RO.toReversible⦄ (s' ++ s'')

theorem parabolic (MO : MarkedReversibleOccurrence α β)
    (rr : ⟨MO.toNet, MO.m₀⟩ ⇝ m) (fs : m ⟦s⟩⦃MO⦄ m') :
    ∃ s' s'', parabolic_decomp (RO := marked_RO_to_RO MO) s s' s'' m m'
  induction fs with
  | @empty m =>
    exists [], []; simp_all [parabolic_decomp, all_backward, all_forward]
    exact ⟨.empty m, TrEq_refl []⟩
  | @step t fs' m m₁ _ et f _ IH =>
    ...
    by_cases h : isBwd t
    · ... /-- Case h : ◁t -/
    · ... /-- Case h : ¬◁t -/
```

Overview de la formalización

Librería

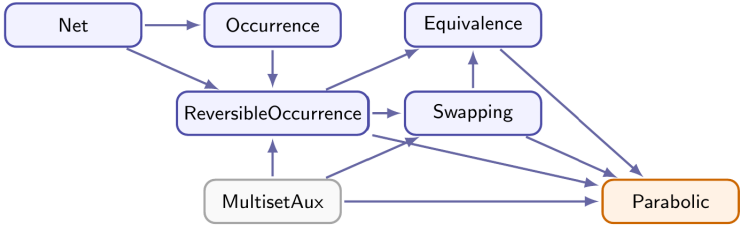
Esta librería contiene la formalización completa de la teoría de redes de Petri con reversibilidad causal consistente. La carpeta principal del proyecto (`PetriNet`) está organizado según los archivos `.lean` del siguiente diagrama:



Overview de la formalización

Librería

Esta librería contiene la formalización completa de la teoría de redes de Petri con reversibilidad causal consistente. La carpeta principal del proyecto (`PetriNet`) está organizado según los archivos `.lean` del siguiente diagrama:



Lineas de código	defs.	lemas	teoremas
>2.8 k	62	136	14

Overview of the formalization

Blueprint y documentación

- Repositorio abierto disponible en: `github.com/danieldavalos93/reversible-occ-nets`.
  - **Lean version:** 4.30.0
  - **Mathlib version:** v4.30.0
- Blueprint:
  - website
  - pdf
- Documentación:
  - website (estilo Lean Library)
- Dependency graph

Conclusión

Conclusión y futuros trabajos

Nuestra contribución:

- Primera mecanización de redes de Petri reversibles en Lean Prover.
- Formalización de resultados claves, incluyendo: backtracking, loop lemma y parabolic lemma.

Conclusión y futuros trabajos

Nuestra contribución:

- Primera mecanización de redes de Petri reversibles en Lean Prover.
- Formalización de resultados claves, incluyendo: backtracking, loop lemma y parabolic lemma.

Escenario para futuros trabajos:

- Colored Petri net.
- Modelos probabilísticos para redes reversibles.
- Modelos reversibles y concurrentes para la computación cuántica.

Muchas gracias! :)

λx:Question $\rightsquigarrow$ trying to answer x...

Consultas

Trace Equivalence

Disjoint implies concurrence

$$
\begin{aligned}
t^\bullet \cap {}^\bullet t' = \emptyset &\Rightarrow {}^\bullet \overleftarrow{t} \cap {}^\bullet t' = \emptyset \\
&\Rightarrow \exists\, \overleftarrow{t}, t'.\ t \preceq \overleftarrow{t} \wedge t' \preceq t' \wedge \neg t \#_0 t' \\
&\Rightarrow \neg(t \# t') \\
&\Rightarrow \neg(t \preceq t') \wedge \neg(t' \preceq t) \wedge \neg(t \# t') \equiv t \; co \; t'
\end{aligned}
$$

Reversing nets

Reverse and Inverse definitions

- $\mathsf{opposite}(tt) = \begin{cases} \mathsf{bwd} \text{ if } tt = \mathsf{fwd} \\ \mathsf{fwd} \text{ if } t_t = \mathsf{bwd} \end{cases}$
- Reverse: $\overleftarrow{t} = (\pi_1(t), \mathsf{opposite}(\pi_2(t)))$
- Inverse: $t \leftarrow\!\!\rightarrow t'$ if and only if $\overleftarrow{t} = t'$

ReversibleOccurrence.lean

```
1  def opposite (tt : TransitionType)
2      : TransitionType :=
3    match tt with
4    | fwd  => bwd
5    | bwd  => fwd
6
7  def reverse (t : Transition β) : Transition β :=
8    (t.fst, opposite (t.snd))
9
10 prefix:300 "←- " => reverse
11
12 def inverse (t t' : Transition β) : Prop :=
13   t = ←- t'
14
15 notation:300 l:300 "←-→ " r:301 => inverse l r
```

Backward conflict

$a \in \alpha$ is in backward conflict over a net N if there are two different transitions t, t' such that $t < a$ and $t' < a$.

Parabolic lemma

Swapping lemma instead square lemma

Swapping.lean

```
def no_rev_in_list (t : Transition β) (ls : List (Transition β)) :=
  ∀ t' ∈ ls, ¬(t ↜⇀ t')

infix:50 " ∉ʳ " => no_rev_in_list

lemma swapping [DecidableEq α] (MO : MarkedReversibleOccurrence α β)
    (rr : ⟨MO.toNet, MO.m₀⟩ ⇝ m)
    (bw : ◁◁s) (ft : ▷t) (nrev : t ∉ʳ s)
    (fs : m ⟦[t] ++ s⟩⟪MO⟫ m') :  m ⟦s ++ [t]⟩⟪MO⟫ m' := ...
```

- ▷t: `t` is a forward transition
- ◁◁ `s`: `s` is a list containing only backward transitions