A Lean Mechanization of Reversible Occurrence Nets

Hernán Melgratti
CONICET & Universidad de Buenos Aires
Supervisor

Daniel Dávalos
Universidad de Buenos Aires
PhD student

18th International Conference on
Reversible Computation
Torino, Italy
July 9–10, 2026

CONICET

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNIVERSIDAD DE BUENOS AIRES

Introduction

- Goal: **formalizing reversible occurrence Petri nets**, and their properties in a proof assistant.
  - We focus on the reversible nets of Reversing Place Transition Nets (Melgratti, Mezzina, Ulidowski).
  - We prove several properties including the parabolic lemma.

Introduction

- Goal: **formalizing reversible occurrence Petri nets**, and their properties in a proof assistant.
  - We focus on the reversible nets of Reversing Place Transition Nets (Melgratti, Mezzina, Ulidowski).
  - We prove several properties including the parabolic lemma.
- We chose **Lean Prover** as proof assistant.
  - full programming language
  - interactive theorem prover

Net

Definitions

A **net** N is a tuple $(\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$ where

- α is the set of **places**,
- β is the set of **transitions**, and
- ${}^\bullet{-}_N, {-}^\bullet_N : \beta \to \mathbb{N}^\alpha$ are functions that assign source and target to each transitions such that ${}^\bullet t \neq \emptyset$ and $t^\bullet \neq \emptyset$ for all $t \in \beta$.

Net

Definitions

A **net** N is a tuple $(\alpha, \beta, {}^\bullet{-}_N, {-}^\bullet_N)$ where

- α is the set of **places**,
- β is the set of **transitions**, and
- ${}^\bullet{-}_N, {-}^\bullet_N : \beta \to \mathbb{N}^\alpha$ are functions that assign source and target to each transitions such that ${}^\bullet t \neq \emptyset$ and $t^\bullet \neq \emptyset$ for all $t \in \beta$.

Net.lean

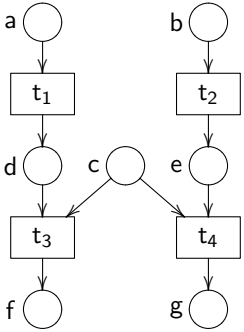
```
structure Net (α : Type) (β : Type) where
 pre: β → Multiset α
 post: β → Multiset α
 cons_prod: ∀ t, pre t ≠ ∅ ∧ post t ≠ ∅
```

Example

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

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

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

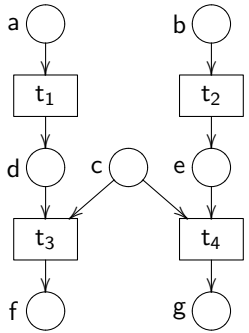


Example

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

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

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



Net.lean

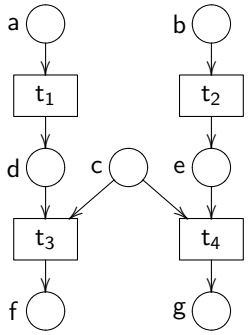
```
inductive Pl | a | b | c | d | e | f | g
inductive Tr | t₁ | t₂ | t₃ | t₄




.
```

Example

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

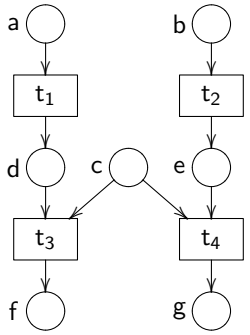
```
inductive Pl | a | b | c | d | e | f | g
inductive Tr | t₁ | t₂ | t₃ | t₄

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









.
```

Example

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

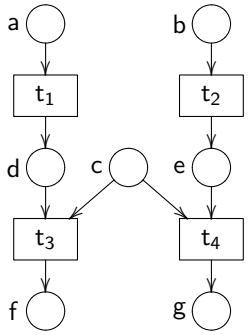
```
inductive Pl | a | b | c | d | e | f | g
inductive Tr | t₁ | t₂ | t₃ | t₄

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}




.
```

Example

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

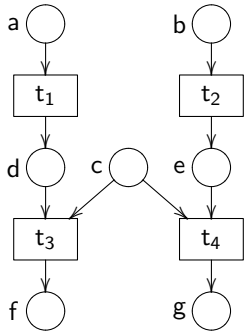
```
inductive Pl | a | b | c | d | e | f | g
inductive Tr | t₁ | t₂ | t₃ | t₄

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 ≠ ∅ :=
  ...

.
```

Example

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

Definition

- A *marking* m of a net N is a multiset over the set of places, i.e., $m \in \mathbb{N}^{\alpha}$.
- A **Petri net** or **marked net** is a pair (N, m) where N is a net and m is a marking of N.

Petri Nets

Definition

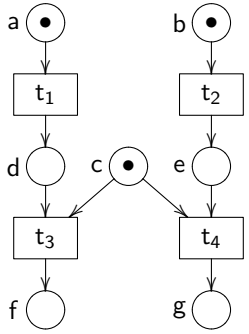
- A *marking* m of a net N is a multiset over the set of places, i.e., $m \in \mathbb{N}^{\alpha}$.
- A **Petri net** or **marked net** is a pair (N, m) where N is a net and m is a marking of N.

Net.lean

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

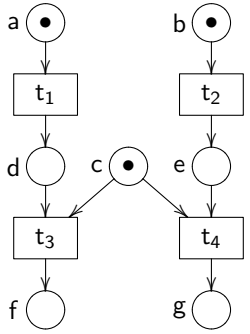
Example

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



Example

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



Net.lean

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

Nets

Preset and postset

Pre- and postsets for places

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

Nets

Preset and postset

Pre- and postsets for places

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

Net.lean

```
def preset_p (N : Net α β) (p : α) : Set β :=
  {t | p ∈ N.post t}

def postset_p (N : Net α β) (p : α) : Set β :=
  {t | p ∈ N.pre t}

def preset_t (N : Net α β) (t : β) :=
  N.pre t

def postset_t (N : Net α β) (t : β):=
  N.post t
```

We introduce the shortland notation $\bullet\{\!\{N\}\!\}x$ and $x\bullet\{\!\{N\}\!\}$ for unified preset and postset syntax in Lean.

Firing

Definition

- A transition t is *enabled* at a marking m, denoted $m[\![\mathsf{t}]\!]$, if ${}^\bullet\mathsf{t} \leq m$.
- The *firing* of a enabled transition t at m is denoted by $m[\![\mathsf{t}\rangle m'$ if $m' = m - {}^\bullet\mathsf{t} + \mathsf{t}^\bullet$.

Firing

Definition

- A transition t is *enabled* at a marking m, denoted $m[\![\mathrm{t}]\!]$, if ${}^\bullet\mathrm{t} \leq m$.
- The *firing* of a enabled transition t at m is denoted by $m[\![\mathrm{t}\rangle m'$ if $m' = m - {}^\bullet\mathrm{t} + \mathrm{t}^\bullet$.

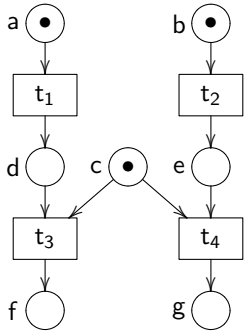
Net.lean

```
def is_enabled (N : Net α β)
    (m : Multiset α) (t : β) : Prop :=
  •{N} t ≤ m

notation m "[[" t "]]{" N "}" =>
                        is_enabled ↑N m t

def marking_after_firing (N: Net α β)
    (m : Multiset α) (_: m[[t]]{N}) :=
  m - •{N} t + t•{N}
```

m[[e⟩{N}m' if firing t (enabled by e : m[[t]]{N}) produces m'.

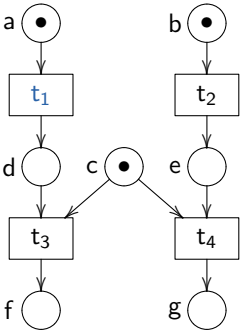
Example

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



Example

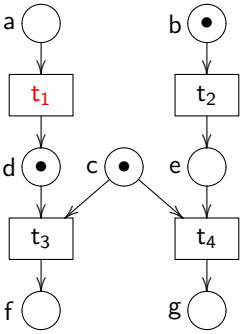
Firing: $(a + b + c)[\![t_1\rangle\!\rangle_{\{\!\{N\}\!\}}(d + b + c)$




```
1  --Net N₁ was building previously
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  .
```

Example

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




```
1  --Net N1 was building previously
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}
```

Semantics

Firing sequence

$s = \mathsf{t}_1; \mathsf{t}_2; \ldots; \mathsf{t}_n$ with *firing sequence*
$m[\![s\rangle\!\rangle m_n$ defined inductively by

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

$s = \varepsilon$ is the empty sequence.

Semantics

Firing sequence

$s = \mathsf{t}_1; \mathsf{t}_2; \ldots; \mathsf{t}_n$ with *firing sequence* $m[\![s\rangle\!\rangle m_n$, defined inductively by

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

$s = \varepsilon$ is the empty sequence.

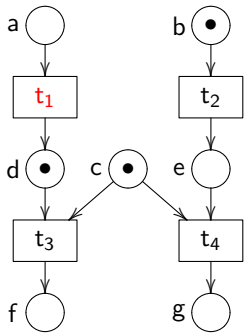
Nets.lean

```
inductive firing_sequence : Net α β →
    Multiset α → List β → Multiset α → Prop
| empty m :
    firing_sequence _ m [] m
| step {N : Net α β} {t} {ts}
    (et : m [[t]]{{N}})
    (f : m [[et⟩{{N}} m')
    (fs : firing_sequence N m' ts m''):
    firing_sequence N m (t :: ts) m''
```

`m[[s⟩⟩{{N}}m'` if the firing sequence from `m` through the sequence `s` is produces `m'`.

Example

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

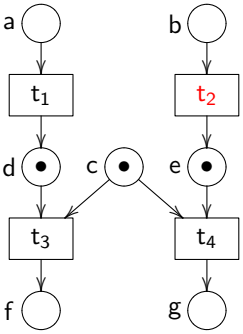


Nets.lean

```
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} := ...
  sorry

.
```

Example

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

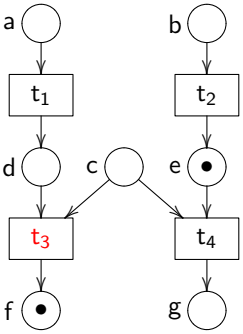


Nets.lean

```
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
.
```

Example

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

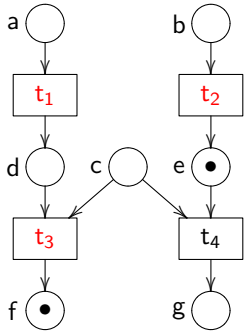


Nets.lean

```
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} := ...
.
```

Example

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}
```

Occurrence nets

Causal dependency

Causal Dependency

The *immediate dependency relation* $<_{\{\!|N|\!\}} \subseteq (\alpha \cup \beta) \times (\alpha \cup \beta)$ is defined by

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

- Causal relation: transitive closure

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

- Causal preorder: reflexive-transitive closure

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

Occurrence nets

Causal dependency

Causal Dependency

The *immediate dependency relation* $<_{\{\!\{N\}\!\}} \subseteq (\alpha \cup \beta) \times (\alpha \cup \beta)$ is defined by

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

- Causal relation:transitive closure

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

- Causal preorder: reflexo-transitive closure

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

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
9
10
11
12
13
14
15
16
17
18 .
```

Occurrence nets

Causal dependency

Causal Dependency

The *immediate dependency relation* $<_{\{\!\{N\}\!\}} \subseteq (\alpha \cup \beta) \times (\alpha \cup \beta)$ is defined by

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

- Causal relation:transitive closure

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

- Causal preorder: reflexo-transitive closure

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

Occurrence.lean

```
def immediate (N: Net α β) :
    (α ⊕ β) → (α ⊕ β) → Prop
  | inl p, inr t  =>  p ∈ •{{N}} t
  | inr t, inl p  =>  p ∈ t •{{N}}
  | _, _          =>  false

notation l "<{{" N "}}" r => immediate N l r

def causal (N : Net α β) :
    (α ⊕ β) → (α ⊕ β) → Prop :=
  TransGen (immediate N)

notation l "≺{{" N "}}" r => causal ↑N l r

def preorder (N : Net α β) :
    (α ⊕ β) → (α ⊕ β) → Prop :=
  ReflTransGen (immediate N)

notation l "≼{{" N "}}" r => preorder ↑N l r
```

Occurrence net

Conflict

- Two different transitions t and t' are in *immediate conflict* if ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, we denote $t \#_0 t$.
- $x, y \in \alpha \cup \beta$ are in *conflict* if there are transitions t, t' such that: $t \preceq x$, $t' \preceq y$ and $t \#_0 t'$.

Occurrence net

Conflict

- Two different transitions t and t' are in *immediate conflict* if ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, we denote $t \#_0 t$.
- $x, y \in \alpha \cup \beta$ are in *conflict* if there are transitions t, t' such that: $t \preceq x$, $t' \preceq y$ and $t \#_0 t'$.

Occurrence.lean

```
def immediate_conflict (N : Net α β) (t t' : β) : Prop :=
  t ≠ t' ∧ ¬ (Disjoint (•{N} t) (•{N} t'))

notation l "#₀{" N "}" r => immediate_conflict ↑N l r

def conflict (N : Net α β) (x y : α ⊕ β) : Prop :=
  ∃ t t' : β, inr t ≼{N} x ∧ inr t' ≼{N} y ∧ t #₀{N} t'

notation l "#{" N "}" r => conflict ↑N l r
```

Occurrence net

We recall the definition of occurrence nets

Occurrence net

A marked net (N, m_0) is an *occurrence net* if it satisfies the following statements:

1. N is acyclic;
2. there is no self-conflict, i.e., $\neg(\mathrm{t} \#_{\{\!\{N}\!\}} \mathrm{t})$ for all transition t;
3. there is no backward conflict;
4. N is 1-safe, i.e., marking is a set;
5. the initial marking is the set of initial places, i.e., m_0 is minimal.

Occurrence net

We recall the definition of occurrence nets

Occurrence net

A marked net (N, m_0) is an *occurrence net* if it satisfies the following statements:

1. N is acyclic;
2. there is no self-conflict, i.e., $\neg(\mathrm{t} \#_{\{\!|N|\!\}} \mathrm{t})$ for all transition t;
3. there is no backward conflict;
4. N is 1-safe, i.e., marking is a set;
5. the initial marking is the set of initial places, i.e., m_0 is minimal.

Occurrence

Occurrence net (redefined)

A marked net (N, m_0) is an *occurrence net* if it satisfies the following statements:

1. N is acyclic;
2. there is no self-conflict, i.e., $\neg(\mathrm{t}\#\mathrm{t})$ for all transition t;
3. there is no backward conflict;
4. for every transition t, $\bullet$t and t$\bullet$ are sets.
5. the initial marking is the set of initial places, i.e., m_0 is minimal.

Occurrence

Occurrence net (redefined)

A marked net (N, m_0) is an *occurrence net* if it satisfies the following statements:

1. N is acyclic;
2. there is no self-conflict, i.e., $\neg(t\#t)$ for all transition t;
3. there is no backward conflict;
4. for every transition t, •t and t• are sets.
5. the initial marking is the set of initial places, i.e., m_0 is minimal.

Occurrence.lean

```
structure is_occurrence (N : Net α β) where
  acyclicity : acyclic N
  no_self_conflict :
    ∀ t : β, ¬inr t #{|N|} inr t
  no_backward_conflict :
    ∀ a : α, ¬backward_conflict N a
  set_preset :
    ∀ t : β,  Nodup (•{|N|} t)
  set_postset :
    ∀ t : β, Nodup (t •{|N|})

structure is_marked_occurrence (M :
    MarkedNet α β) : Prop where
  occurrence : is_occurrence M.toNet
  initial_set : Nodup M.m₀
  inital_minimal : ∀ {x}, x ∈ M.m₀ → x ∈
      minimal M.toNet
```

Occurrence

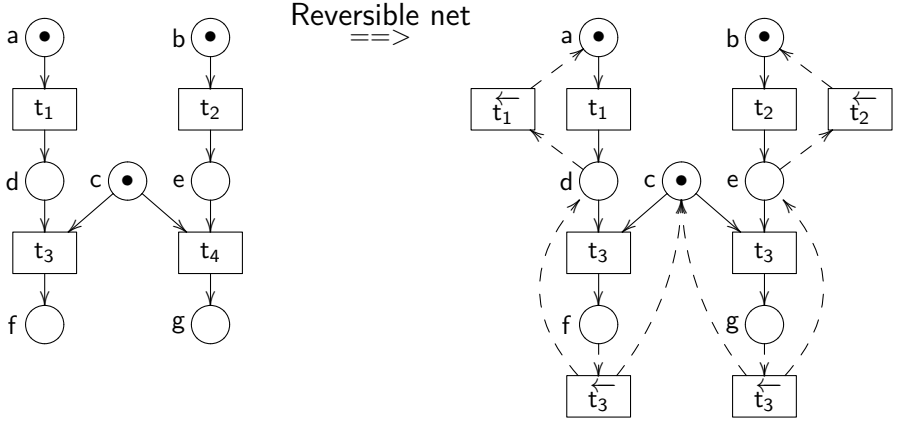
Occurrence.lean

```
/-- This def. is in Nets.lean file -/
def safe (M : MarkedNet α β) : Prop :=
  ∀ m, M ⇝ m → Nodup m

theorem occ_net_safe (MO : is_marked_occurrence M) : safe M
```

M ⇝ m is true if m ∈ reach(M).

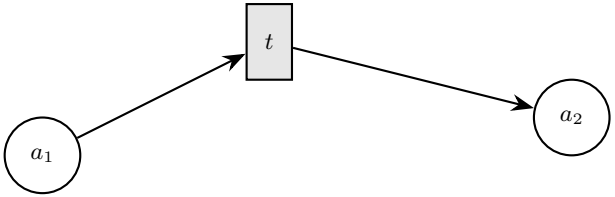
Reversing nets

How do you obtain a reversible net?



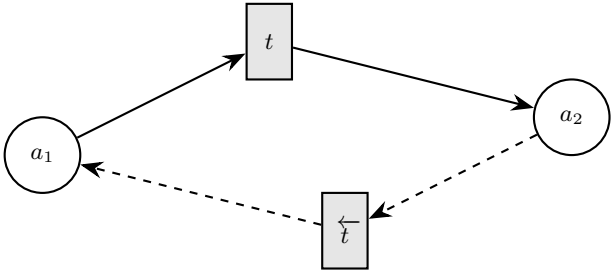
Reversing nets

Transition Type



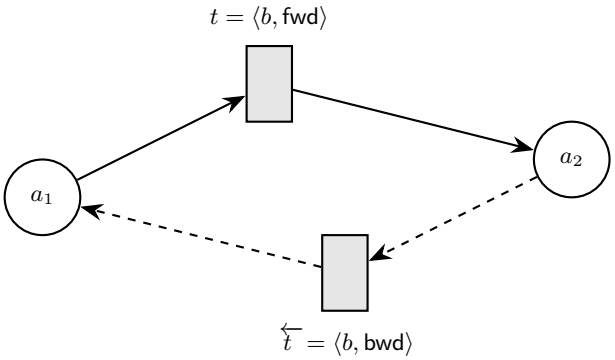
Reversing nets

Transition Type



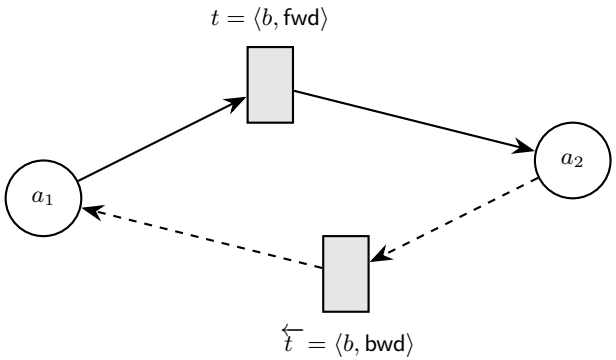
Reversing nets

Transition Type



Reversing nets

Transition Type



ReversibleOccurrence.lean

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

Reversing nets

Definition

A *reversible occurrence net* is a net $N = (\alpha, \beta \uplus \{ \overleftarrow{t} \mid t \in \beta \}, {}^\bullet\_, \_^\bullet)$ defined by the following conditions:

1. $(\alpha, \beta, ({}^\bullet\_)_{|\beta}, (\_^\bullet)_{|\beta})$ is an occurrence net, where $f_{|\beta}$ denotes the restriction of the function f to the domain β.
2. $\forall t\ \exists t',\ {}^\bullet t = t'^\bullet \wedge t^\bullet = {}^\bullet t'$.

Reversing nets

Definition

A *reversible occurrence net* is a net $N = (\alpha, \beta \uplus \{ \overleftarrow{t} \mid t \in \beta\}, {}^\bullet\_, \_^\bullet)$ defined by the following conditions:

1. $(\alpha, \beta, ({}^\bullet\_)_{|\beta}, (\_^\bullet)_{|\beta})$ is an occurrence net, where $f_{|\beta}$ denotes the restriction of the function f to the domain β.
2. $\forall t\ \exists t',\ {}^\bullet t = t'^\bullet \wedge t^\bullet = {}^\bullet t'$.

ReversibleOccurrence.lean

```
structure Reversible α β
    extends Net α (Transition β) where
  welldef (t t' : Transition β) :
    t ⤚ t' → •{{toNet}}t = t'•{{toNet}}

structure ReversibleOccurrence α β extends
    Reversible α β where
  isOccurrence :
    is_occurrence (fwd_subnet toNet)

structure MarkedReversibleOccurrence α β
    extends Reversible α β where
  m₀ : Multiset α
  isFwdMarkedOccurrence :
    is_marked_occurrence ⟨fwd_subnet toNet, m₀⟩
```

- `fwd_subnet`: is the function that take a net of type `N:α (β×TransitionType)` and return their forward net.

Trace Equivalence

Equivalence on firing sequences:
the smallest congruence $\asymp$ on transition sequences closed under the following rules:

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

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

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

Trace Equivalence

The *equivalent firing sequences*, is the smallest congruence $\asymp$ on transition sequences closed under the following rules:

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

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

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

Equivalence.lean

```
inductive TrEq (R : Reversible α β) :
  List (Transition β) → List (Transition β) → Prop
| swap : (t t' : Transition β) →
  Disjoint (t•{|R|}) (•{|R|} t') → Disjoint (t'•{|R|}) (•{|R|} t) →
  TrEq R [t, t'] [t', t]
| canc_l : (t t' : Transition β) → t ↚ t' →
  TrEq R [t, t'] []
| canc_r : (t t' : Transition β) → t ↚ t' →
  TrEq R [] [t, t']
...
notation:50 s₁:51 "≍{|" R "|} " s₂:51 => TrEq R s₁ s₂

/--We omit the proofs that ≍ is an equivalence relation.-/

lemma TrEq_congruence (h : s₁ ≍{|R|} s₂) (h' : s₃ ≍{|R|} s₄) :
    (s₁ ++ s₃) ≍{|R|} (s₂ ++ s₄)
```

Parabolic lemma

Formal statement

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 $m[\![\omega';\omega''\rangle\!\rangle_{\{\!\{R\}\!\}} m'$.

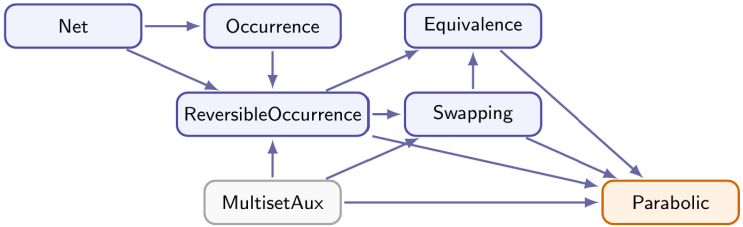
Parabolic lemma

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 of the formalization

This library contains the fully formalized of the theory of reversible Petri nets with causally consistent reversibility. The main folder (`PetriNet`) is organized as follow files `.lean`:



Lines of code	def	lemma	theorem
>2.8 k	62	136	14

Overview of the formalization

Blueprint and documentation

- Open repository available at: `github.com/danieldavalos93/reversible-occ-nets`.
  - **Lean version:** 4.30.0
  - **Mathlib version:** v4.30.0
- Blueprint:
  - website
  - pdf
- Documentation:
  - website
- Dependency graph

Conclusion and future works

Our contribution:

- First mechanization of reversible Petri nets on Lean Prover.
- We mechanize key results, including the backtracking, loop and parabolic lemmas.

Scenarios for future extensions:

- Colored Petri net.
- Probabilistic models for reversible nets.
- Reversible and concurrent models of quantum computing.

Thank you!

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

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

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