Un *Transition-based Nondeterministic Finite Automaton* (TNFA) è una tupla $\mathcal{A} = (S, \Sigma, S_0, R, Acc)$ con accettazione transition-based.

Da un TNFA $\mathcal{A}$ può essere derivato (per i nostri scopi) un *State-based Nondeterministic Finite Automaton* (SNBA) $\mathcal{A}'$ con accettazione state-based ed etichettatura sugli stati nel seguente modo:

- $\mathcal{A}' = (S', S'_0, R', L, Acc')$ dove:
  - $S' = \{s_t \,|\, t \in R\}$,
  - $S'_0 = \{s_t \,|\, t = (s, a, s') \text{ e } s \in S_0\}$,
  - $R' \subseteq S' \times S'$ è definita dalla seguente regola:

$$\frac{t = (s, a, s') \wedge t' = (s', b, s'')}{s_t \longrightarrow s'_t}$$

  - $L(s_t) = a$, se $t = (s, a, s')$,
  - $Acc' = \{s_t \,|\, t \in Acc\} \subseteq S'$.

Algorithm 1: `backwardSNBA`($\mathcal{A}$): conversione in un SNFA trasposto dato un TNFA $\mathcal{A}$ in input

Data: $\mathcal{A} = (S, \Sigma, S_0, R, Acc)$: TNFA $\mathcal{A}$ in input
Data: $outTransitionStases \leftarrow \emptyset$: mappa che associa a ogni stato $s \in S$ i suoi transition states $s_t \in S'$ in uscita
Data: $inTransitionStases \leftarrow \emptyset$: mappa che associa a ogni stato $s \in S$ i suoi transition states $s'_{t'} \in S'$ in entrata
Output: SNFA $\mathcal{A}' = (S', S'_0, R', L, Acc')$ trasposto

```
1  A' ← (S' = ∅, S'_0 = ∅, R' = ∅, L = ∅, Acc' = ∅)
2  for s ∈ S do
3      for (s, t, s') ∈ R(s) do
4          for stateDenotation ∈ extractStateDenotations(A, t.cond)
           do
5              s_t = newTransitionState(stateDenotation)
6              S' = S' ∪ s_t
7              if s ∈ S_0 then
8                  S'_0 = S'_0 ∪ s_t
9              else
10                 outTransitionStates[s].pushBack(s_t)
11             inTransitionStates[s'].pushBack(s_t)
12             if t ∈ Acc then
13                 Acc' = Acc' ∪ s_t
14             for s'_{t'} ∈ outTransitionStates[s'] do
15                 if s'_{t'} ∈ Acc' then
16                     R' = R' ∪ markItAsAccepting({s'_{t'}, s_t})
17                 else
18                     R' = R' ∪ ({s'_{t'}, s_t})
19     for s'_{t'} ∈ inTransitionStates[s] do
20         for s_t ∈ outTransitionStates[s] do
21             if s_t ∈ Acc' then
22                 R' = R' ∪ markItAsAccepting({s_t, s'_{t'}})
23             else
24                 R' = R' ∪ ({s_t, s'_{t'}})
25 return A'
```

L'Algoritmo 1 prende in input un TNFA $\mathcal{A}$ risultato della traduzione di una formula LTL in un automa e lo converte in un SNBA $\mathcal{A}'$ trasposto. L'implementazione esegue la conversione visitando l'automa $\mathcal{A}$ una sola volta.

Chiamiamo uno stato $s_t \in S'$ dell'automa SNBA $\mathcal{A}'$ *transition state* in quanto viene creato a partire da una transizione appartenente all'automa $\mathcal{A}$. Siccome la libreria Spot associa agli archi dell'automa delle condizioni, l'Algoritmo 1 interpreta una transizione $(s, t, s') \in R$ come un insieme di transizioni. Tale insieme di transizioni deve essere estrapolato da t. Da t vengono estrapolati i mintermini associati alla sua guardia t.`cond`, e per ognuno di essi viene creato un nuovo transition state s_t. Questo lavoro viene fatto dalla funzione `extractStateDenotations` in Linea 4. Nella implementazione C++ viene calcolato anche l'interpretazione dello stato.

L'algoritmo mantiene traccia di alcune informazioni:

- i transition states $s_t \in S'$ in uscita da uno stato $s \in S$ (cioè quelli creati a partire dagli archi uscenti di s). Tale informazione viene mantenuta in una mappa associativa *outTransitionStates*.
- i transition states $s'_{t'} \in S'$ in entrata in uno stato $s \in S$ (cioè quelli creati a partire dagli archi entranti di s). Tale informazione viene mantenuta in una mappa associativa *inTransitionStates*.

Inoltre, la libreria Spot, indipendentemente dal fatto che l'automa usi una accettazione basata sugli stati o sulle transizioni, l'appartenenza di uno stato a un insieme di accettazione viene sempre memorizzato sulle transizioni (come un bit vector). La convenzione di Spot per implementare la state-base acceptance è quella di marcare ogni arco uscente da uno stato come accettante. Per questo motivo, a Linea 16 e a Linea 22, se un transition state s_t è accettante, tutte le sue transizioni in uscita devono essere marcate come accettanti.

Infine, si noti che a Linea 16 e a Linea 22 gli archi vengono creati in ordine inverso in quanto vogliamo il trasposto di $\mathcal{A}'$.

Di seguito vengono riportati due esempi di output. Le formule subiscono le solite trasformazioni (discretizzazione e LTL).

- Formula: $\mathtt{G}(i) \wedge t_0 \wedge \mathtt{G}(t_1) \wedge \mathtt{F}(p \wedge \mathtt{F}\, q)$
  Total states: 52
  Total final states: 8
  Total transitions: 416

- Formula: $p_1 \wedge q_1 \wedge \mathtt{X}(p_1) \wedge \mathtt{X}(q_1) \wedge (v_1\, \mathtt{U}\, (r_1\, \mathtt{R}\, z_1)) \wedge \mathtt{G}(x_1) \wedge \mathtt{F}(u_1 \wedge \mathtt{F}(p2 \wedge (\mathtt{F}p_3 \vee (u_2\, \mathtt{W}\, p_4)))) \wedge (t \vee \mathtt{G}(\mathtt{X}(w)))$
  Total states: 69857
  Total final states: 0
  Total transitions: 141222368