Notes on Petri Nets and Attack Trees

Aubrey Bryant and Harley Eades III

November 5, 2018

Abstract

Cybersecurity professionals often use attack trees to visualize the various possible paths of attack on a resource. Such visualizations provide important tools for communication and threat analysis. However, they can quickly become complex and unwieldy, and must rely on humans for error-checking and the tracing of possible paths. We are working to translate attack trees into Petri nets, mathematical graphical models that can be analyzed and verified much more quickly and accurately. In their original form, Petri nets are constructed of places and transitions between places, with a flow relation indicating how resources can flow through the net, or what sequence of steps is needed to get from one point to another in the net. We have begun development of a chainable Petri net, which facilitates the construction of complex nets using simple logical operations already found in attack trees. Once constructed, these nets can be analyzed using the powerful tools native to the Petri net model. This will enable cybersecurity professionals to check two nets for equivalence or hierarchy, incorporate one net into another, compute possibilities under given circumstances, and complete many other tasks with greater speed and accuracy than is possible under the current model.

1 Introduction

Attack trees are an important resource for cybersecurity researchers. However, as purely graphical models, they are laborious to build and analyze. To address this limitation, researchers have explored ways to represent them as Petri nets. Building and analyzing attack trees in the form of Petri nets remain complex tasks. To simplify the translation of attack trees into Petri nets and facilitate their analysis, we have developed a chainable Petri net, or CNET, which tracks both its initial and final places (the beginning and end points of the net) to make it easy to chain these nets together to build a large, complex net out of simple pieces. To join the simple nets together, we have developed connectives based on logical operators frequently found in attack tree models, such as sequential composition, disjunction, and conjunction. In addition, we have defined two new varieties of conjunction not currently used in attack trees: stepwise conjunction and synchronous conjunction.

Attack trees provide a useful visual model with a number of advantages. Because they are fairly easy to understand, attack trees facilitate communication between professionals with different areas of expertise; such collaboration among specialists improves the quality of the security analysis. In addition, attack trees help defenders

communicate with both detail and clarity, improving the quality of their end product. Attack tree models also encourage the people building them to decompose attacks into smaller steps; by helping to draw out lines of attack, these models lead to a more comprehensive view of both vulnerabilities and methods of defense. The attack tree method encourages defenders to take the attacker's view by drawing attention to the attacker's goals and the available means to achieve them, thus helping them to develop a more complete defense. Attack trees also provide good visual documentation of a team's collaborative efforts to discover and mitigate security vulnerabilities, facilitating their efforts to secure a system or resource.

Given their many strengths, it is no surprise that attack trees have been a key tool for cybersecurity professionals since their development. However, as cybersecurity vulnerabilities and defenses become more complex, it becomes more difficult to use an attack tree to model them. One problem is that attack trees are generally built by hand; though there are tools to assist with the process, it remains labor-intensive, and the work required increases with the complexity of the system and its vulnerabilities. Beyond this, even in relatively simple scenarios, it is often possible to construct two attack trees that look very different, but in fact accurately capture the same vulnerabilities of the system. This problem of variation in equivalent attack trees also naturally increases as the complexity of the attack tree increases. As purely visual models, attack trees have very little formal structure to support them, and few resources to deal with the problems sketched above. However, work in the field has established a link between attack trees and Petri nets, which are both graphical and mathematical models.

Petri nets, first developed by Carl Adam Petri in 1962, are composed of places and transitions, with arcs connecting the two. The places model the state of the system under attack, and/or the progress of an attacker toward the goal. The transitions between the places represent actions that the attacker or the system can take to progress or impede the attack. The arcs link places to transitions, and transitions to places to show the paths through the net. Petri nets can be represented mathematically as well as graphically, allowing us to reason about them at a more abstract level while still maintaining an important common ground with attack trees.

Researchers have applied category theory to Petri nets, allowing us to abstract even further, so that we can observe the most essential features of the objects under study. In stripping away the nonessential, category theory also helps us to notice important relationships between different objects that were not obvious before. By viewing Petri nets as a category, we can use abstraction and comparison to discover and prove important mathematical features that a purely graphical model like attack trees cannot reveal. In particular, we view Petri nets as monoids, a structure with objects and a binary operation that is associative and has an identity element. The places of a Petri net are a monoid with set union as the operator, and the transitions are a monoid with concatenation as the operator. Viewing Petri nets as monoids has allowed us to focus on the structure of each logical operator, so that we can build complex nets from simple parts. In sum, our contributions include:

- A chainable Petri net, or CNET, that tracks initial and final places, and start and end transitions, facilitating chaining multiple nets together.
- A definition of sequential composition for CNETs, with proof that it is associa-

tive but not symmetric, and keeps repetitions distinct.
- A definition of disjunction for CNETs, with proof that it is a coproduct.
- A definition of simple conjunction for CNETs that allows for TRUE CONCURRENCY without added complexity due to the chainable feature of the CNET, with proof that it is a tensor product.
- A definition of synchronous conjunction for CNETs, a new type of connective that is not currently used in attack trees but would likely prove very useful in capturing steps that need to take place simultaneously, like denial-of-service attacks. Also, we include a proof that this is a product.
- A definition of stepwise conjunction for CNETs, where two nets must complete one at a time, in any order. This is also not currently used in attack trees, but may prove useful.

Ultimately, by joining in the effort to translate attack trees into mathematical/categorical models instead of simply visual models, we hope to give cybersecurity professionals a model that is just as useful as attack trees, but that additionally has properties provable with mathematical certainty, such as equivalence or hierarchy between two nets.

2 Petri Net Basics

Petri nets are powerful models for processes in many different areas, because they can capture key elements of processes and facilitate reasoning about them. Petri nets help to visualize the dynamics of a system, and support that visualization with mathematical definitions, analysis, and proof. There are many varieties of Petri nets to support many different uses, but the basic Petri net is a structure composed of places and transitions, with arcs between them that capture possible flows or paths through the net.

Definition 1. *A **Petri net**, (P, T, F), consists of the following structure:*

- *A finite set of places P,*
- *A finite set of transitions T that is disjoint from P,*
- *A causal dependency relation $F : (P \times T) + (T \times P) \longrightarrow \mathbb{N}^+$,*

The places track progress through the net by means of markers called tokens, while the transitions enable progress to be made. If a transition has two input places, both inputs must be active (contain tokens) in order for the transition to fire. Similarly, if a transition has two output places, both places will become active once the transition has fired. The transitions consume tokens from their input places and produce tokens in their output places each time their input conditions are satisfied. The input/output relation is captured in F, the causal dependency or flow relation.

As established in [6], Petri nets are monoids. Monoids are structures with objects and an associative binary relation with an identity element. In the case of Petri nets, they are monoids in a twofold way because they are composed of two kinds of things - places and transitions. Multiset union forms a monoid over the places of a Petri net, while composition forms a monoid over the transitions, with the empty set as the identity element for both.

Definition 2. *A **finite multiset** over a set S can be defined as a formal sum denoted by $n_1a_1 \oplus \cdots \oplus n_ia_i$, where $n_1, \ldots, n_i \in \mathbb{N}$, $a_1, \ldots, a_i \in S$, and na denotes that a occurs n times in the multiset, such that the following properties hold:*

- $na_i \oplus ma_j = ma_j \oplus na_i$,
- $na \oplus ma = (m + n)a$,
- $na \oplus 0 = na = 0 \oplus na$.

Definition 3. *The **free commutative monoid** generated by the set A is the set of all finite multisets drawn from A, where the monoidal operation is the formal sum of multisets, and the monoidal unit is $\emptyset$.*

Petri nets fit more generally into the category of graphs.

Definition 4. *A **generalized graph**, $(V, T, \mathsf{src}, \mathsf{tar}, \_^*, I, \otimes)$, consists of the following structure:*

- *A set of nodes V,*
- *A set of edges T,*
- *A functor $\_^* : \mathsf{Set} \longrightarrow \mathsf{Set}$, where $(V^*, I, \otimes)$ is a monoid,*
- *A source function $\mathsf{src} : T \longrightarrow V^*$,*
- *A target function $\mathsf{tar} : T \longrightarrow V^*$.*

Definition 5. *A **graph morphism**, $h : G \longrightarrow G'$, between graphs is a pair (f, g) where:*

- *$f : T \longrightarrow T'$ is a function,*
- *$g : V \longrightarrow V'$ is a function.*

Definition 6. *A **monoid homomorphism** between two monoids $(M_1, \oplus_1, 0_1)$ and $(M_2, \oplus_2, 0_2)$ is a function $f : M_1 \longrightarrow M_2$ such that:*

- $f(a_1 \oplus_1 a_2) = f(a_1) \oplus f(a_2)$
- $f(0_1) = 0_2$

Definition 7. *A **generalized graph morphism** between two generalized graphs $(V_1, T_1, \mathsf{src}_1, \mathsf{tar}_1, \_^{*_1}, I_1, \otimes_1)$ and $(V_2, T_2, \mathsf{src}_2, \mathsf{tar}_2, \_^{*_2}, I_2, \otimes_2)$ is a graph morphism (f, g) where $f : T_1 \longrightarrow T_2$ is a function and $g : V_1^{*_1} \longrightarrow V_2^{*_2}$ is a monoid homomorphism.*

Definition 8. *A **Petri net**, $(S, T, \mathsf{src}, \mathsf{tar}, \_^{\oplus}, \emptyset, \oplus)$, is a generalized graph. We call the elements of the set S places and the elements of T transitions. Furthermore, we call the generalized graph homomorphisms between Petri nets **Petri net homomorphisms**.*

We will denote a Petri net $(S, T, \mathsf{src}, \mathsf{tar}, \_^{\oplus}, \emptyset, \oplus)$ by the pair $(S^{\oplus}, T)$ for readability. Also, for $t \in T \, such that \mathsf{src(t)} = s_A \, and \mathsf{tar(t)} = s_B$, we will use the notation $t : s_A \to s_B$ for clarity and conciseness. Thus, take as objects Petri nets $(S^{\oplus}, T)$ and as morphisms Petri net morphisms. This defines a category Petri.

Lemma 9. *Suppose we have two Petri nets, $(S_1^{\oplus}, T_1)$ and $(S_2^{\oplus}, T_2)$.*

$$(S_1^{\oplus} \times S_2^{\oplus}) \cong (S_1 + S_2)^{\oplus}$$

Proof. To show this, we need to define two homomorphisms between them, which we will call F and G.

It is important to note that the $\oplus$ operation is commutative. This commutativity allows the elements of the resulting free commutative monoid (FCM) to be rearranged, which will help us in our proof by allowing FCMs to be sorted in the following way:

Consider an arbitrary $s \in (S_1 + S_2)^{\oplus}$. Since the generator is the disjoint union of S_1 and S_2, the elements of s are drawn from those two sets, but are not grouped according to their origin set. Using commutativity, we can move all $y_i \in S_1$ to the front, and all $z_j \in S_2$ to the end. Then the form of s is $(\oplus_i y_i) \oplus (\oplus_j z_j)$, where the elements are grouped according to their origin set. Any element of $(S_1 + S_2)^{\oplus}$ can be similarly sorted. For the purposes of this proof, assume that all such elements are so sorted.

Now, let us define the function $F : (S_1 + S_2)^{\oplus} \longrightarrow (S_1^{\oplus} \times S_2^{\oplus})$.
Let k be an arbitrary element of $(S_1 + S_2)^{\oplus}$. We know that $k \in (S_1 + S_2)^{\oplus}$ must have the form $(\oplus_i y_i) \oplus (\oplus_j z_j)$, where $y_i \in S_1$ and $z_i \in S_2$ as described above. Furthermore, since the + operator is disjoint union, each element includes a marker indicating its origin set, which we can use to locate the boundary between elements of S_1 and elements of S_2. Therefore we can define F as follows:

$$F((\oplus_i y_i \in S_1) \oplus (\oplus_j z_j \in S_2)) = ((\oplus_i y_i \in S_1), (\oplus_j z_j \in S_2))$$

$$F((S_1, \oplus) \oplus (S_2, \oplus)) = ((S_1, \oplus), (S_2, \oplus))$$

This allows an element of $(S_1 + S_2)^{\oplus}$ to be matched with an element of $(S_1^{\oplus} \times S_2^{\oplus})$, since the generator of $(S_1 + S_2)^{\oplus}$ contains the generator for both $S_1^{\oplus}$ and $S_2^{\oplus}$.

Another important feature of FCMs arises from lifting the coproducts of sets to FCMs. For the coproduct of sets, we know that there exist the following functions:

$$inj_1 : S_1 \longrightarrow (S_1 + S_2)$$

$$inj_2 : S_2 \longrightarrow (S_1 + S_2)$$

Likewise, for the coproduct of FCMs, there are the functions:

$$inj_1^{\oplus} : S_1^{\oplus} \longrightarrow (S_1 + S_2)^{\oplus}$$

$$inj_1^{\oplus}(\oplus_i n_i x_i) = \oplus_i n_i (inj_1 x_i)$$

$$inj_2^{\oplus} : S_2^{\oplus} \longrightarrow (S_1 + S_2)^{\oplus}$$

$$inj_2^{\oplus}(\oplus_i n_i x_i) = \oplus_i n_i (inj_2 x_i)$$

Using this property, let us now define $G : (S_1^{\oplus} \times S_2^{\oplus}) \longrightarrow (S_1 + S_2)^{\oplus}$.
Let k be an arbitrary element of $(S_1^{\oplus} \times S_2^{\oplus})$. We know k must have the form (M_1, M_2) where M_i is a multiset of the form $(n_1 z_1 \oplus ... \oplus n_b z_b)$, and all $z \in S_i$ for $i \in \{1, 2\}$ (all n

being natural number counters for the multiset).
Therefore we define the function G as follows:

$$G(M_1, M_2) \quad = \quad (inj_1^{\oplus} M_1) \oplus (inj_2^{\oplus} M_2)$$

By this transformation, we can match an element of $(S_1^{\oplus} \times S_2^{\oplus})$ with an element of $(S_1 + S_2)^{\oplus}$, since by the nature of coproducts, any s $\in S_1^{\oplus} \in (S_1 + S_2)^{\oplus}$, and similarly any s $\in S_2^{\oplus} \in (S_1 + S_2)^{\oplus}$.

Now that we have defined a homomorphism in both directions, we must prove that $F; G = Id$ and $G; F = Id$. Take an arbitrary multiset M of the form $(n_1 z_1 \oplus ... \oplus n_b z_b)$, with some elements drawn from a set S_a and some drawn from S_b and each $n_i \in \mathbb{N}$. Sorting this multiset will yield the form $(\oplus_a n_a y_a) \oplus (\oplus_b n_b z_b)$, where $y_a \in S_a$ and $z_b \in S_b$. Putting this into the functions, we get:

$$\begin{array}{rcl} G(F((\oplus_a n_a y_a) \oplus (\oplus_b n_b z_b))) & = & G((\oplus_a n_a y_a), (\oplus_b n_b z_b)) \\ & = & (\oplus_a n_a y_a) \oplus (\oplus_b n_b z_b) \end{array}$$

Thus, $F; G = Id$.

Now let us check for identity in the opposite direction, proving G;F = Id.
Take arbitrary sets S_a and S_b. Generate the FCM of each and then their product: $(S_a^{\oplus} \times S_b^{\oplus})$. The result will have the form $((\oplus_a n_a y_a), (\oplus_b n_b z_b))$, where $y_a \in S_a$ and $z_b \in S_b$. Putting this into the functions, we get:

$$\begin{array}{rcl} G((\oplus_a n_a y_a), (\oplus_b n_b z_b)) & = & ((\oplus_a n_a y_a) \oplus (\oplus_b n_b z_b)) \\ F((\oplus_a n_a y_a) \oplus (\oplus_b n_b z_b)) & = & ((\oplus_a n_a y_a), (\oplus_b n_b z_b)) \end{array}$$

Thus, G;F = Id. Since $(S_1^{\oplus} \times S_2^{\oplus}) \leftrightarrow (S_1 + S_2)^{\oplus}$, we can conclude that $(S_1^{\oplus} \times S_2^{\oplus}) \cong (S_1 + S_2)^{\oplus}$.

□

Definition 10. *A **chainable Petri net**, $(S^{\oplus}, T, i, f, start, end)$ is a Petri net with the following additional features:*

- *A set drawn from $S, \{i_1, i_2, ...i_n\} \in S$, called the initial places. This is the starting point of the net.*
- *A set drawn from $S, \{f_1, f_2, ...f_n\} \in S$, called the final places. This marks the completion of the net.*
- *A set drawn from $T, \{start_1, start_2...start_n\} \in T$, called the starting transitions of the net. If the initial marking is $(s_1 \oplus s_2... \oplus s_n)$, then* src$\{start\} = \{i\}$, *and* tar$\{start\} = (s_1 \oplus s_2... \oplus s_n)$, *with elements not duplicated: $s_j \neq s_k$ when $j \neq k$.*
- *A set drawn from $T, \{end\} \in T$, called the end transitions of the net. If the final marking is $(s_1 \oplus s_2... \oplus s_n)$,* src$(end) = (s_1 \oplus s_2... \oplus s_n)$, *and* tar$(end) = f$.

- *Morphisms for chainable nets must preserve both start and end, and their associated sets of places i and f. So, a morphism* $< a, b >: (N, start, end) \longrightarrow (N', start', end')$ *is an ordinary net morphism that preserves the markings* $b(start) = start'$ *and* $b(end) = end'$.

BA: does the preservation of i and f need to be explicit or is it implied already? And, do the morphisms make sense now that these things are sets instead of individual things?

The chainable petri net, or CNET, is a petri net that tracks both its initial and its final places, to facilitate chaining the petri net together with other petri nets at either end. This enables the use of operators such as OR, SEQ, & AND.

Example 11. *First let us look at a simple operation, the sequential ordering (SEQ) of two Petri nets. This is a common operation that requires one task to be completed before another. This means that the operation cannot be commutative, since the order matters. In a simple Petri net, this is modeled graphically as two (or more) places arranged along a non-branching line. Clearly in such an arrangement the first place must be activated in order to activate the second place, and so on. Only when the last node is activated has the SEQ operation completed.*

Definition 12. *Given two CNETs* $N_1 = (S_1^{\oplus}, T_1, i_1, f_1, start_1, end_1)$ *and* $N_2 = (S_2^{\oplus}, T_2, i_2, f_2, start_2, end_2)$, *the* ***sequentitial composition*** *is* $N_1; N_2$, *where:*

- $S = S_1 + S_2$.
- $T = T_1 + T_2 + \{(f_1 \rightarrow i_2)\}$.
- $i = i_1$.
- $f = f_2$.
- $start = start_1$.
- $end = end_2$.

Order is key here. Making the first net's start and i the composition's start and i ensures that the composition begins with the net that is first in the sequence. Similarly, making the second net's end and f the composition's end and f ensures that the composition finishes upon the completion of the second in the sequence. Finally, the new transition between f_1 *(marking the end of the first net) and* i_2 *(marking the beginning of the second net) provides the necessary connection between the two. In order to be sequential, this operation must be associative but not symmetric, so now let us check those properties.*

Lemma 13. *The SEQ operation for two CNETs, A; B, is not symmetric.*

Proof. Let $A = (S_A^{\oplus}, T_A, i_A, f_A, start_A, end_A)$ and $B = (S_B^{\oplus}, T_B, i_B, f_B, start_B, end_B)$. Symmetry would imply that $A; B = B; A$. Let us compute the components of each to test for symmetry.
The components of $A; B$ are:

- $S_{A;B} = S_A + S_B$.
- $T_{A;B} = T_A + T_B + \{(f_A \longrightarrow i_B)\}$.
- $i_{A;B} = i_A$.
- $f_{A;B} = f_B$.
- $start_{A;B} = start_A$.
- $end_{A;B} = end_B$.

The components of $B;A$ are:

- $S_{B;A} = S_B + S_A$.
- $T_{B;A} = T_B + T_A + \{(f_B \longrightarrow i_A)\}$.
- $i_{B;A} = i_B$.
- $f_{B;A} = f_A$.
- $start_{B;A} = start_B$.
- $end_{B;A} = end_A$.

Relying on the symmetry of disjoint union, we see that these two compositions have identical places, $S_A + S_B = S_B + S_A$. However, whereas the construction of $A;B$ includes $\{(f_A \longrightarrow i_B)\} \in T_{A;B}$, the construction of $B;A$ includes $\{(f_B \longrightarrow i_A)\} \in T_{B;A}$. Furthermore, the start and end transitions and initial and final places are not the same. Thus, $A;B \neq B;A$ and the operation is not symmetric. □

Lemma 14. *The SEQ operation for CNETs, $A;B;C$, is associative.*

Proof. Let $A = (S_A^{\oplus}, T_A, i_A, f_A, start_A, end_A)$, $B = (S_B^{\oplus}, T_B, i_B, f_B, start_B, end_B)$, and $B = (S_C^{\oplus}, T_C, i_C, f_C, start_C, end_C)$. Composing the three, $A;B;C$, can be done one of two ways: $(A;B);C$ or $A;(B;C)$. For the operation to be associative, the following must be true: $(A;B);C = A;(B;C)$. Let us compute each to test for equality.
First, we compute $(A;B);C$ by taking $A;B$:

- $S_{A;B} = S_A + S_B$.
- $T_{A;B} = T_A + T_B + \{(f_A \longrightarrow i_B)\}$.
- $i_{A;B} = i_A$.
- $f_{A;B} = f_B$.
- $start_{A;B} = start_A$.
- $end_{A;B} = end_B$.

Now we add C:

- $S_{(A;B);C} = S_A + S_B + S_C$.
- $T_{(A;B);C} = T_A + T_B + \{(f_A \longrightarrow i_B)\} + \{(f_B \longrightarrow i_C)\}$.
- $i_{(A;B);C} = i_A$.
- $f_{(A;B);C} = f_C$.
- $start_{(A;B);C} = start_A$.
- $end_{(A;B);C} = end_C$.

Now let us follow the second path, computing $A;(B;C)$ by first taking $B;C$:

- $S_{B;C} = S_B + S_C$.
- $T_{B;C} = T_B + T_C + \{(f_B \longrightarrow i_C)\}$.
- $i_{B;C} = i_B$.
- $f_{B;C} = f_C$
- $start_{B;C} = start_B$.
- $end_{B;C} = end_C$.

Now we add A:

- $S_{A;(B;C)} = S_A + S_B + S_C$.
- $T_{A;(B;C)} = T_A + T_B + \{(f_A \longrightarrow i_B)\} + \{(f_B \longrightarrow i_C)\}$.
- $i_{A;(B;C)} = i_A$.
- $f_{A;(B;C)} = f_C$
- $start_{A;(B;C)} = start_A$.
- $end_{A;(B;C)} = end_C$.

These clearly give the same result, showing that $(A;B);C = A;(B;C)$.
In sequential composition, the first CNET's starting transition $start_A$ and initial place i_A always becomes the composition's start and initial place. Then, the final place of a preceding net is joined to the initial place of the subsequent net by adding to T for the composition. The final transition, *end*, of the last net in the sequence becomes the *end* of the composition, as does the final place, f_C. Thus, the operation is associative for any number of nets. □

Lemma 15. *The SEQ operator for CNETs takes repetition into account. That is,* $A;A \neq A$.

Proof. Let $A = (S_A^{\oplus}, T_A, i_A, f_A, start_A, end_A)$. Following the definition of sequential composition, we see that the elements of the two CNETs are joined by disjoint union. In this operation, members of each set are paired with a label to indicate their origin set. Since the origin nets have the same name, we will use $x \in \{1,2\}$to make them distinct. Now let us compute $A;A$.

- $S_{A;A} = S_A + S_A = (S_A \times \{1\}) \cup (S_A \times \{2\})$.
- $T_{A;A} = T_{A_1} + T_{A_2} + (f_{A_1} \longrightarrow i_{A_2}\}$.
- $i_{A;A} = i_{A_1}$.
- $f_{A;A} = f_{A_1}$.
- $start_{A;A} = start_{A_1}$.
- $end_{A;A} = end_{A_2}$.

BA: the T needs to respect which A it came from - does this just happen by definition?

Each occurence of A is marked such that it is distinct from other occurences of A, and $A;A$ also contains the transition $(f_{A_1} \longrightarrow i_{A_2})$, which is not in A. Thus, $A;A \neq A$. □

Example 16. *Next let us examine disjunction (OR) in CNETs, starting with a basic example. Consider an ordinary Petri net* $(S^{\oplus}, T)$. *A simple model of the disjunction* s_2 *OR* s_3 *is given by the following relations:*

- *Let* $S = \{s_1, s_2, s_3\}$, *and* $T = \{(s_1 \longrightarrow s_3), (s_2 \longrightarrow s_3)\}$.
- *Given this relation, as long as* s_1, s_2, *or both are activated,* s_3 *will be activated.*
- *If neither* s_1 *nor* s_2 *are activated,* s_3 *will also remain inactive.*

Within a single Petri net, we can see that disjunction is modeled graphically by two transition arcs leading to one place. If either transition fires, the place will be activated, enabling its subsequent transition(s) to fire. The disjunction of two CNETs will result in a similar pattern: if at least one of the CNET disjuncts reaches its final marking, then the disjunction itself will reach its final marking.

Definition 17. *Given two CNETs,* $N_1 = (S_1^{\oplus}, T_1, i_1, f_1, start_1, end_1)$ *and* $N_2 = (S_2^{\oplus}, T_2, i_2, f_2, start_2, end_2)$, *their* ***disjunction*** *is* $N_1 + N_2$, *where:*

- *The set of initial places* $i_{1+2} = i_1 \times i_2$.
- *The set of final places* $f_{1+2} = f_1 + f_2$.
- *The start transition* $start_{1+2} = start_1 \times start_2$.
- *The end transition* $end_{1+2} = end_1 + end_2$
- $S = S_1 + S_2 + i_{1+2} + f_{1+2}$
- $T = T_1 + T_2$.
- $[\mathsf{src}_1, \mathsf{src}_2], [\mathsf{tar}_1, \mathsf{tar}_2] : T \longrightarrow S^{\oplus}$.

BA: does this make sense?

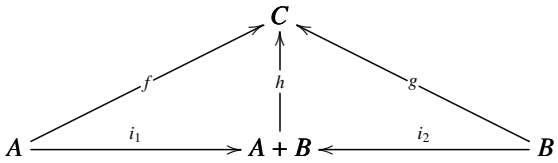
BA: is f1+2 really necessary here? already in S1 + S2

The pair of the initial places of the disjuncts is the initial place of the new net, and likewise the starting transition is the pair of the disjuncts' starting transitions. This way, each branch of the disjunction can be reached initially. The final place is a new place that is the target of the transition from each disjunct, so that the completion of either branch (or both) can trigger the disjunction's final place. The disjunction's places gather these elements together, along with the places of each branch. Finally, the transitions appropriate to each branch are used to compute runs, with the addition of the new transitions for f.

BA: don't think we need this in short notation, ASK

Lemma 18. *The disjunction of two CNETs, A + B, is a coproduct.*

Proof. $N_1 + N_2$ is a coproduct if it has morphisms $i_1 : A \longrightarrow A + B$ and $i_2 : B \longrightarrow A + B$ such that for any object C with morphisms $f : A \longrightarrow C$ and $g : B \longrightarrow C$, there is a unique morphism $h : A + B \longrightarrow C$ such that $f = i_1; h$ and $g = i_2; h$, meaning that this diagram commutes:



Let $A = (P_A, T_A, \mathsf{src_A}, \mathsf{tar_A}, i_A, f_A)$ and $B = (P_B, T_B, \mathsf{src_B}, \mathsf{tar_B}, i_B, f_B)$, and take A + B.
Let us define i_0 and i_1, which are injections for Petri nets. We will take it by parts for clarity.
$P_{A+B} = (P_A + P_B + \{(i_A, i_B)\} + \{f_A\} + \{f_B\})$
The coproduct of sets of places has the injective functions already, which covers all the constituents except the initial places. For the initial places, $i_A \longrightarrow i_A \times i_B$ and $i_B \longrightarrow i_A \times i_B$ just maps the individual initial place to the pair of initial places, using the position in the pair to indicate whether it is i_A or i_B.
$T_{A+B} = T_A + T_B$, so again we can use the coproduct of sets to obtain the injective functions for this part of the net.
$[\mathsf{src}_1, \mathsf{src}_2], [\mathsf{tar}_1, \mathsf{tar}_2] : T \longrightarrow P^{\oplus}$. These functions already respect set membership, and so their injective function is also clear. Since these composite pieces are all coproducts, we can rely on their injections to yield the injections of the larger structure.

BA: See definition 8; do I need to explain why we are using long notation for this when we've been doing short notation for CNETS up to this point?

Now, suppose we have another CNET C and morphisms $f : A \longrightarrow C$ and $g : B \longrightarrow C$; we define h as follows:

$$h(x) = \begin{cases} f(x), & \text{if x is from A} \\ g(x), & \text{if x is from B} \end{cases} \tag{1}$$

Given this definition of h, it is clear that $f = i_1; h$ and $g = i_2; h$, since h(x) applies f(x) if $x \in A$ and g(x) if $x \in B$.
To show that this satisfies the uniqueness requirement, let us consider an arbitrary function $k : (A + B) \longrightarrow C$, where $i_1; k = f$ and $i_2; k = g$. For $a \in A$ and $b \in B$, the following equalities hold:

$$i_1(a); k(a) = f(a) = h(a) \tag{2}$$

$$i_2(b); k(b) = g(b) = h(b) \tag{3}$$

Thus, $h = k$, showing that h is unique. This provides a morphism as required, showing that the CNET disjunction is a coproduct.

□

Definition 19. *Given two CNETs,* $N_1 = (P_1, T_1, \mathsf{src}_1, \mathsf{tar}_1, i_1, f_1)$ *and* $N_2 = (P_2, T_2, \mathsf{src}_2, \mathsf{tar}_2, i_2, f_2)$*, their* ***synchronous conjunction*** *is* $N_1 \,[\!]\, N_2$*, where:*

- *The initial place* $i = i_1 \times i_2$.
- *The final place* $f = f_1 \times f_2$.
- *The beginning transition* $start = start_1 \times start_2$.
- *The ending transition* $end = end_1 \times end_2$.
- $P = P_1 \times P_2$.
- $T = T_1 \times T_2$.
- $[\mathsf{src}_1, \mathsf{src}_2], [\mathsf{tar}_1, \mathsf{tar}_2] : T \longrightarrow P^{\oplus}$.

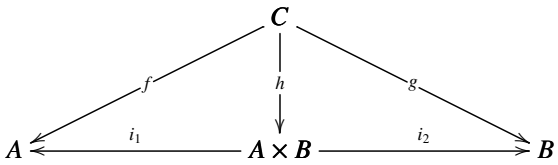
BA: Do I need this part?

The initial place of synchronous conjunction is the pair of initial places from the two branches. This ensures that each branch will be reached initially. The conjunction's places are formed by pairing the places of the two branches. The transitions are likewise paired, effecting the synchronization necessary for this operation. Finally, the end transition has the pair of final places from each branch as its source, requiring that both complete in order to activate the transition to the final place of the conjunction, which is the target of end.

BA: Probably should prove that this is synchronous. How can I do that?

Lemma 20. *The synchronous conjunction of two CNETs,* $A \,[\!]\, B$*, is a product.*

Proof. $N_1 \,[\!]\, N_2$ is a product if it has morphisms $i_1 : A \times B \longrightarrow A$ and $i_2 : A \times B \longrightarrow B$ such that for any object C with morphisms $f : C \longrightarrow A$ and $g : C \longrightarrow B$, there is a unique morphism $h : C \longrightarrow A \times B$ such that $f = h; i_1$ and $g = h; i_2$, meaning that this diagram commutes:



Let $A = (P_A, T_A, \mathsf{src_A}, \mathsf{tar_A}, i_A, f_A)$ and $B = (P_B, T_B, \mathsf{src_B}, \mathsf{tar_B}, i_B, f_B)$, and take $A \,[\!]\, B$. Let us define i_1 and i_2, which are projections for Petri nets. We will take it by parts for clarity.
$P_{A[\!]B} = (P_1 \times P_2) + i + f$. The product of the sets of places, $(P_1 \times P_2)$, has the projections already, so we need only examine projections for i and f. $i = (i_A \times i_B)$ and $f = (f_A \times f_B)$, so both of these already have the projections as well. T = $(T_1 \times T_2)$ + *start* + *end*. The product of the sets of transitions has projections already, so we can focus on *start* and *end*. Similarly, because these are the products of the start of A and B and the end of A and B, they also have the necessary projections. Thus, the synchronous conjunction is a product. □

Example 21. *Next let us examine another type of AND in CNETs, Stepwise AND. We will construct this formally using a combination of SEQ and OR.*

Definition 22. *Given two CNETs,* $N_1 = (P_1, T_1, \mathsf{src_1}, \mathsf{tar_1}, i_1, f_1)$ *and* $N_2 = (P_2, T_2, \mathsf{src_2}, \mathsf{tar_2}, i_2, f_2)$*, their* ***stepwise conjunction*** *is* $N_1 \wedge N_2 = (N_1; N_2) + (N_2; N_1)$.
Let us take it by parts for clarity, starting with the first disjunct:

- $P_{N_1;N_2} = P_1 + P_2$.
- $T_{N_1;N_2} = T_1 + T_2 + \{(f_1 \longrightarrow i_2)\}$.
- $i_{N_1;N_2} = i_1$.
- $f_{N_1;N_2} = f_2$.
- $start_{N_1;N_2} = start_1$.
- $end_{N_1;N_2} = end_2$.

Now let us move on to the second disjunct:

- $P_{N_2;N_1} = P_1 + P_2$.
- $T_{N_2;N_1} = T_1 + T_2 + \{(f_2 \longrightarrow i_1)\}$.
- $i_{N_2;N_1} = i_2$.
- $f_{N_2;N_1} = f_1$.
- $start_{N_2;N_1} = start_2$.
- $end_{N_2;N_1} = end_1$.

Now we can compute the disjunction of these sequences, (with duplication omitted):

- $P_{N_1;N_2+N_2;N_1} = P_1 + P_2$.
- $T_{N_1;N_2+N_2;N_1} = T_1 + T_2 + \{(f_1 \longrightarrow i_2)\} + \{(f_2 \longrightarrow i_1)\}$.
- $i_{N_1;N_2+N_2;N_1} = i_1 \times i_2$.
- $f_{N_1;N_2+N_2;N_1} = f_2 + f_1$.
- $start_{N_1;N_2+N_2;N_1} = start_1 \times start_2$.
- $end_{N_1;N_2+N_2;N_1} = end_2 + end_1$.

The development of this connective using SEQ and OR means that both must complete, in either order, to activate the next transition. However, this connective does not allow for partial progress on one, and then partial progress on the other. This feature, called interleaving, is accomplished with the next connective.

Example 23. *Consider an ordinary Petri net $(S^{\oplus}, T)$. A simple model of the conjunction s_1 AND s_2 is given by the following relations:*

- *Let $S = \{s_1, s_2, s_3\}$, and $T = \{((s_1, s_2) \longrightarrow\!\!\!\!\!\rightarrow s_3)\}$.*
- *Given this relation, s_1 and s_2 must both be activated for s_3 to be activated.*
- *If either s_1 or s_2 remain inactive, s_3 will also remain inactive.*

Within a single Petri net, we can see that conjunction is modeled graphically by two places feeding into one transition. Both places must be occupied for the transition to fire. Accomplishing the conjunction of more complex processes has long been a subject of debate, with the two main approaches being true concurrency and interleaving. The initial and final places that define the CNET help to resolve this difficulty. By pairing the initial places of each conjunct, we ensure that each thread in the conjunction begins to fire. Then, by taking the disjoint union of the transitions and places of each conjunct, we allow the two threads to complete in any order (while still following the flow relation of each thread). The threads could even make partial progress, first part of one and then part of the other. We verify that each thread has completed (fulfilling the necessary condition for conjunction) by pairing the final places in the conjunction.This construction allows the two conjuncts to remain independent of each other during their execution, with the initial and final places creating the conjunction.

Definition 24. *Given two CNETs, $N_1 = (P_1, T_1, \mathsf{src}_1, \mathsf{tar}_1, i_1, f_1)$ and $N_2 = (P_2, T_2, \mathsf{src}_2, \mathsf{tar}_2, i_2, f_2)$, their **simple conjunction** is $N_1 \amalg N_2$ where:*

- $P = P_1 + P_2$
- $T = T_1 + T_2$
- $start = start_1 + start_2$
- $end = end_1 + end_2$
- $i = i_1 \times i_2$
- $f = f_1 \times f_2$

Lemma 25. *The simple conjunction of two CNETs, $A \amalg B$, is a tensor product.*

Proof. To be a tensor product, simple conjunction must be both commutative and associative. First, let us check commutativity. Let $A = (P_A, T_A, \mathsf{src_A}, \mathsf{tar_A}, i_A, f_A)$ and $B = (P_B, T_B, \mathsf{src_B}, \mathsf{tar_B}, i_B, f_B)$, and take $A \amalg B$.

- $P_{A \amalg B} = P_A + P_B$
- $T_{A \amalg B} = T_A + T_B$
- $start_{A \amalg B} = start_A + start_B$
- $end_{A \amalg B} = end_A + end_B$
- $i_{A \amalg B} = i_A \times i_B$

- $f_{A \amalg B} = f_A \times f_B$

Now, let us compare this to $B \amalg A$.

- $P_{B \amalg A} = P_B + P_A$
- $T_{B \amalg A} = T_B + T_A$
- $start_{B \amalg A} = start_B + start_A$
- $end_{B \amalg A} = end_B + end_A$
- $i_{B \amalg A} = i_B \times i_A$
- $f_{B \amalg A} = f_B \times f_A$

The places, transitions, and start/end are all equivalent by virtue of the commutativity of disjoint union. While the paired initial places are reversed, in this context the reversed pairs are equivalent because either way the places are joined together, ensuring that they will both be activated once the conjunction is activated. Similarly, the paired final places are joined together to ensure that the conjunction does not complete until both branches complete. Thus, the simple conjunction is commutative. Now let us check for associativity. In addition to A and B above, let there be a third CNET C. Let $C = (P_C, T_C, \mathsf{src_C}, \mathsf{tar_C}, i_C, f_C)$ and take $(A \amalg B) \amalg C$.

- $P_{(A \amalg B) \amalg C} = (P_A + P_B) + P_C$
- $T_{(A \amalg B) \amalg C} = (T_A + T_B) + T_C$
- $start_{(A \amalg B) \amalg C} = (start_A + start_B) + start_C$
- $end_{(A \amalg B) \amalg C} = (end_A + end_B) + end_B$
- $i_{(A \amalg B) \amalg C} = (i_A \times i_B) \times i_C$
- $f_{(A \amalg B) \amalg C} = (f_A \times f_B) \times f_C$

Now let us take $A \amalg (B \amalg C)$.

- $P_{A \amalg (B \amalg C)} = P_A + (P_B + P_C)$
- $T_{A \amalg (B \amalg C)} = T_A + (T_B + T_C)$
- $start_{A \amalg (B \amalg C)} = start_A + (start_B + start_C)$
- $end_{A \amalg (B \amalg C)} = end_A + (end_B + end_B)$
- $i_{A \amalg (B \amalg C)} = i_A \times (i_B \times i_C)$
- $f_{A \amalg (B \amalg C)} = f_A \times (f_B \times f_C)$

BA: Check consistency; start and end are not paired, but initial and final are - is that right?

Again, the places, transitions, and start/end are all equivalent by virtue of the associativity of disjoint union. The paired initial and final places are isomorphic to one another, as the next proof shows. □

Lemma 26. $((A \times B) \times C) \cong (A \times (B \times C))$

Proof. To show this, we must define two homomorphisms between them, which we will call f and g. The function f is defined as follows: Taking the second projection of $(A \times B) \times C)$ yields C. Taking the first projection of $(A \times B) \times C)$ yields $(A \times B)$. Taking the first projection again yields A. Taking the second projection yields B. Then, we can rejoin the elements, first taking $B \times C$ and then $(A \times (B \times C))$. Thus,

$$f((A \times B) \times C) \quad = \quad (A \times (B \times C))$$

Now let us define g. Taking the first projection of $(A \times (B \times C))$ yields A. Taking the second projection of $(A \times (B \times C))$ yields $(B \times C)$. Taking the first projection of $(B \times C)$ yields B. Taking the second projection yields C. Then, we can rejoin the elements, first taking $A \times B$ and then $((A \times B) \times C)$. Thus,

$$g(A \times (B \times C)) \quad = \quad ((A \times B) \times C)$$

Finally, let us check that $g; f = Id$ and $f; g = Id$.

$$\begin{array}{rcl} f(g(A \times (B \times C))) & = & f((A \times B) \times C) \\ & = & (A \times (B \times C)) \end{array}$$

Thus, $g; f = Id$. Likewise,

$$\begin{array}{rcl} g(f((A \times B) \times C)) & = & g(A \times (B \times C)) \\ & = & ((A \times B) \times C) \end{array}$$

Thus, $f; g = Id$, and $((A \times B) \times C) \cong (A \times (B \times C))$ □

*** Note this in introduction. Also look up symmetric monoidal category for monoidal tensor product not just tensor product. ***

Example nets: K =
$S_K : \{a, b, c\}$
$T_K : \{t\}$
$F_K(a, t) = 2$
$F_K(b, t) = 1$
$F_K(t, c) = 2$
$F_K(else) = 0$
$S_K^{\oplus} : \{2a \oplus 1b \oplus 2c\}$
$t = \{2a \oplus 1b \longrightarrow 2c\}$
$\delta_{0K}(t) = 2a \oplus 1b\}$
$\delta_{1K}(t) = 2c\}$
M =
$S_M : \{d, e\}$
$T_M : \{t'\}$
$F_M(d, t') = 2$
$F_M(t', e) = 1$
$F_M(else) = 0$
$S_M^{\oplus} : \{2d \oplus 1e\}$
$t = \{2d \longrightarrow 1e\}$
$\delta_{0M}(t') = 2d\}$
$\delta_{1M}(t') = 1e\}$
$S_K^{\oplus} \times S_M^{\oplus} = (2a \oplus 1b, 2d) \longrightarrow (2c, 1e)$
$(S_K + S_M)^{\oplus} = (\{1\} \times S_K) \cup (\{2\} \times S_M)^{\oplus}$
$((1, 2a), (1, 1b), (2, 2d))^{\oplus} =$
$S_{KM}^{\oplus} : \{(1, 2a), (1, 1b), (2, 2d)\}$
$T_{KM} : \{t_1, t_2\}$
$t_1 = ((1, 2a) \oplus (1, 1b)) \longrightarrow (1, 2c)$
$t_2 = (2, 2d) \longrightarrow (2, 1e)$
$\delta_{0KM}(t_1) = (1, 2a) \oplus (1, 1b)\}$
$\delta_{0KM}(t_2) = (2, 2d)\}$
$\delta_{1KM}(t_1) = (1, 2c)\}$
$\delta_{1KM}(t_2) = (2, 1e)\}$
$(S_1^{\oplus} \times S_2^{\oplus})$ and $(S_1 + S_2)^{\oplus}$ are isomorphic.
Suppose there are two Petri nets $S_1 and S_2$, with sets of places $M_1 and M_2$ and sets of arcs $T_1 and T_2$.
Let F be the homomorphism from $(S_1 + S_2)^{\oplus}$ to $(S_1^{\oplus} \times S_2^{\oplus})$.
$F((S_1 + S_2)^{\oplus}) = (S_1^{\oplus} + S_2^{\oplus})$
where $S_1 + S_2 = (\{1\} \times S_1) \cup (\{2\} \times S_2)$
For F:
Calculating the set of places of the codomain:
$S_1 = \bigcup_{y \in domain(x,y)} | x = 1)$
$S_2 = \bigcup_{y \in domain(x,y)} | x = 2)$
Note: I mean to separate out the two sets using the disjoint markers here.
The set of places of the codomain = $(S_1^{\oplus} \times S_2^{\oplus})$
The arrows of the codomain maintain the structure defined in the domain, disregarding

the origin markers {1} and {2} added in by taking the disjoint union.
Since the function preserves the structure between the objects, only changing the names of the objects, this is a homomorphism.
Now, let G be the homomorphism from $(S_1^{\oplus} \times S_2^{\oplus})$ to $(S_1 + S_2)^{\oplus}$.
The objects or places of the codomain are the objects of the domain modified in the following way:
where objects in the domain have the form (x, y), $(\{1\} \times x) \cup (\{2\} \times y)$.
The arrows of the codomain maintain the structure of the domain, adding in origin markers {1} and {2} by position as described above.
Since G preserves the group's structure and only modifies objects' names, it is a homomorphism. Clearly,$G \circ F = F \circ G$, since F removes the origin markings {1} and {2} and G puts them back. Structure remains unchanged in either direction.

3 Equational Representation of Petri Nets

In [5], the authors propose an adaptation of propositional dynamic logic for reasoning about Petri nets. Others have translated Petri nets into languages for PDL. In this article, the authors offer a method for using PDL on Petri nets more directly and simply. They first define PDL, which consists of a finite number of proposition symbols with Boolean connectives $\neg$ and $\wedge$, and a finite number of basic programs, with program constructors ; (sequential composition), $\cup$ (non-deterministic choice), and * (iteration), and a modality $\langle\pi\rangle$ for each program π. Regarding the modality, $\langle\pi\rangle\alpha$ means that after π runs, α is true, assuming π stops. A transition diagram, also called a frame, defines the semantics for a PDL.

Definition 27. *A frame is a tuple $F = (W, R_\pi)$ where:*

- *W is a non-empty set of states .*
- *R_a is a binary relation over W, for each basic program $a \in \Pi$.*
- *The binary relation R_π can be defined inductively for each non-basic program π:*
  - $R_{\pi_1;\pi_2} = R_{\pi_1}; R_{\pi_2}$
  - $R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$
  - $R_{\pi^*} = R_\pi^*$, *the reflexive transitive closure of* R_π.

To represent Petri nets in PDL, the authors name the places and transitions, and define three patterns of transitions(representing the simple place to place, the and pattern, and the or pattern) from which more complex structures can be built by composition. Additionally, they define a sequence of names as a notation for tracking progress through a net, denoted as S. They define a firing function $f : S \times \pi_b \to S$ on pg 6, in which if s contains the preset of the transition, the transition is enabled. This algorithm enforces the flow relation of the Petri net, and I think consumes transitions while keeping places. Then they use this to define axioms which seem to present propositional dynamic logic equations for petri nets. [4]

References

[1] Nick Benton. Semantic equivalence checking for hhvm bytecode. In *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming*, PPDP '18, pages 3:1–3:8, New York, NY, USA, 2018. ACM.

[2] Eike Best and Harro Wimmel. Reducing k-safe petri nets to pomset-equivalent 1-safe petri nets. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2000*, pages 63–82, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[3] Roberto Bruni, Jos Meseguer, Ugo Montanari, and Vladimiro Sassone. Functorial models for petri nets. *Information and Computation*, 170(2):207 – 236, 2001.

[4] Bruno Lopes, Mario Benevides, and Edward Hermann Haeusler. Extending propositional dynamic logic for petri nets. *Electronic Notes in Theoretical Computer*

Science, 305:67 – 83, 2014. Proceedings of the 8th Workshop on Logical and Semantic Frameworks (LSFA).

[5] Bruno Lopes, Mario Benevides, and Edward Hermann Haeusler. Propositional dynamic logic for petri nets. *Logic Journal of the IGPL*, 22(5):721–736, 2014.

[6] Jos Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105 – 155, 1990.

[7] Mogens Nielsen, Lutz Priese, and Vladimiro Sassone. Characterizing behavioural congruences for petri nets. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory*, pages 175–189, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[8] V. Sassone. On the algebraic structure of petri nets. 2000.

[9] Vladimiro Sassone. An axiomatization of the algebra of petri net concatenable processes. *Theoretical Computer Science*, 170(1):277 – 296, 1996.