

Notes on Petri Nets and Attack Trees

Aubrey Bryant and Harley Eades III

November 24, 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 Attack Trees

Attack trees were first described by [20], and have become a widely-used tool for modeling both attacker behavior and system vulnerabilities [17]. Attack trees are constructed by placing the ultimate goal, compromise of the resource, at the root, and then dividing it into smaller and smaller sub-goals, which populate the branches and leaves of the tree. In this way, each path up the tree becomes a possible way to achieve the ultimate goal. The tree nodes are joined with logical connectives such as conjunction and disjunction. In the case of conjunction (AND), both children must be accomplished in order for the parent to be accomplished. In the case of disjunction (OR), at least one of the children must be accomplished for the attack to progress to the parent. Attack trees also make use of sequential connectives, which lay out the steps to a goal in a defined order.

Attack trees are useful because they encourage the defender to think like an attacker. They focus attention on the vulnerabilities, and force defenders to analyze the possible exploitations of these vulnerabilities and the precise steps one would need to take to achieve the goal of compromising the target resource. Furthermore, attack trees provide an excellent resource for collaboration and brainstorming over potential attacks, enabling professionals with varying areas of expertise to work together. However, attack trees have significant limitations as well. Although [20] saw them as reusable in similar scenarios, in practice reuse has not been easy. Attack trees are labeled with specific steps to achieve a particular goal in a given scenario; with that level of concreteness, it is difficult to adapt one attack tree for multiple situations. In a similar vein, their specificity makes attack tree construction laborious and time-consuming, and the same goes for attack tree analysis. Furthermore, as purely graphical models, it is difficult to determine properties of attack trees such as equivalence or hierarchy without significant effort [8]. Also, as static models attack trees do not capture the effect of timing on vulnerabilities [17], and also do not permit simulation testing.

3 Petri-Nets

Attack trees are a valuable but limited resource for cybersecurity analysis. To address their limitations while retaining their strengths, researchers have established a correspondence between attack trees and Petri nets, which are graphical and mathematical process models. With the addition of mathematical modeling, Petri nets gain significant power both in representation and in analysis. 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. McDermott [14] was the first to explore the correspondence between attack trees and Petri nets, modeling the states of the system as places, and actions or events that cause a change in state as transitions. He describes the attack net approach as contributing several strengths to penetration testing: modeling concurrency and the current state of an attack with tokens, showing intermediate goals more clearly as places, and showing the cumulative effects of different avenues of attack on the whole system.

While there is widespread agreement that attack trees can be usefully reformulated as Petri nets, there is little consensus about how this process should be done. Without a standard conversion algorithm, the representation of attack trees as Petri nets will fall victim to many of the same weaknesses already applying to attack trees, especially regarding the labor required and the potential for human error. To help address this, we have developed a type of Petri net that can be built in manageable pieces and then chained together to form ever-larger structures, until the entire attack tree is represented. Aside from the chainability of this new kind of Petri net, we have also developed connectives that capture the AND and OR often used in attack trees, as well as other connectives that will likely be useful in capturing the interaction of sub-goals to achieve a larger goal. Let us begin by formally defining a Petri net.

BA: substantiate this claim with citations

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.

4 Petri Nets as Monoids

As established in [15], 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. Viewing Petri nets through this lens allowed us to reason about them more abstractly, and to build complex nets out of simple building blocks using connectives we define. The fact that the places and transitions of a Petri net form free commutative monoids is crucial to our development of these connectives, because it gives us a way to gather these elements together and sort them out again. From there, we can define morphisms to structure the connections between two nets without altering them. The free commutative monoid is built upon the notion of a finite multiset, as the definitions below make clear.

Definition 2. A *finite multiset* over a set S can be defined as a formal sum denoted by $n_1a_1 \oplus \dots \oplus n_ia_i$, where $n_1, \dots, n_i \in \mathbb{N}$, $a_1, \dots, a_i \in S$, 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 are monoids, so their morphisms can be specified by defining a monoid homomorphism.

Definition 4. 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:

- $f(a_1 \oplus_1 a_2) = f(a_1) \oplus_2 f(a_2)$
- $f(0_1) = 0_2$

5 Petri Nets as Graphs

Petri nets fit more generally into the category of graphs, in that Petri net places correspond to nodes and the transitions correspond to edges or paths between nodes. Viewing Petri nets this way helps us in our understanding of the Petri net category. Let us begin to show this correspondence by first defining the category of graphs.

Definition 5. A *generalized graph*, $(V, T, \text{src}, \text{tar}, -, *, I, \otimes)$, consists of the following structure:

- A set of nodes V ,

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

Taking the graphs as objects, we can define morphisms between them to complete the definition of the category of graphs.

Definition 6. 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.

Thus, bringing in the definition of graph and monoid homomorphisms, we can define a Petri net morphism.

Definition 7. A **generalized graph morphism** between two generalized graphs $(V_1, T_1, \text{src}_1, \text{tar}_1, _{-}^{*1}, I_1, \otimes_1)$ and $(V_2, T_2, \text{src}_2, \text{tar}_2, _{-}^{*2}, I_2, \otimes_2)$ 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.

Now we have a complete categorical definition of Petri nets.

Definition 8. A **Petri net**, $(S, T, \text{src}, \text{tar}, _{-}^{\oplus}, \emptyset, \oplus)$, 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 $(S, T, \text{src}, \text{tar}, _{-}^{\oplus}, \emptyset, \oplus)$ by the pair (S^{\oplus}, T) for readability. Also, for $t \in T$ such that $\text{src}(t) = s_A$ and $\text{tar}(t) = s_B$, we will use the notation $t : s_A \rightarrow s_B$ for clarity and conciseness.

Thus, take as objects Petri nets (S^{\oplus}, T) and as morphisms Petri net morphisms. This defines a category Petri.

6 Isomorphism of FCM's Product and Coproduct

In order to define the connectives for Petri nets categorically, we need to first establish that the product and the coproduct of free commutative monoids are isomorphic. This will enable us to gather the elements of the net together in equivalent ways so that our connectives add to the structure or flow of the net as it already exists, non-destructively.

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

BA: is this shifting up a level from the previous definition of graph?

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 $(n_1 z_1 \oplus \dots \oplus n_b z_b)$, 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:

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

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 $(n_1 z_1 \oplus \dots \oplus n_b z_b)$, 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:

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

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:

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

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

□

7 Representing Attack Trees as Petri Nets

8 Chainable Petri Nets

So far, we have explored the category of Petri nets, its monoidal characteristics, and its relation to the category of graphs. In our exploration of the category of Petri nets, we have exposed key characteristics of the free commutative monoid over the places and transitions. The isomorphism between the product and coproduct of the free commutative monoid gives us the ability to connect Petri nets using these operations. This is crucial to our efforts to translate attack trees into Petri nets because it defines an important part of how Petri nets can be combined. However, in order to focus on the connectives, we first need to make sure that the pieces we are joining together will not be changed internally by the connection. This ensures that whatever patterns already built into a net will not be altered by making that net part of a larger net. By isolating each part this way, we make it possible to build a complex net out of simple parts. We achieve this isolation by distinguishing two transitions and two places for each Petri net. These elements mark the start and end of the net, so that one net can be joined to

another like so many garden hoses or extension cords. This net pattern has also been used in process modeling, where it is known as a workflow net.

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 , $\{i_1, i_2, \dots, i_n\} \in S$, called the initial places. This is the starting point of the net.
- A set drawn from S , $\{f_1, f_2, \dots, f_n\} \in S$, called the final places. This marks the completion of the net.
- A set drawn from T , $\{start_1, start_2, \dots, start_n\} \in T$, called the starting transitions of the net. If the initial marking is $(s_1 \oplus s_2 \dots \oplus s_n)$, then $src\{start\} = \{i\}$, and $tar\{start\} = (s_1 \oplus s_2 \dots \oplus s_n)$, 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 $(s_1 \oplus s_2 \dots \oplus s_n)$, $src(end) = (s_1 \oplus s_2 \dots \oplus s_n)$, 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 $\langle a, b \rangle: (N, start, end) \rightarrow (N', start', end')$ is an ordinary net morphism that preserves the markings $b(start) = start'$ and $b(end) = end'$.

Making Petri nets chainable in this way ensures that the internal structure of component Petri nets remains untouched. For example, if we have two Petri nets and we want to join them together in sequence, we want to make sure that the first net starts and finishes, and then the next net starts and finishes. We do not want the nets to make progress concurrently, or for the second one to go first. Since we have distinguished the start and end of each component, we can define the sequential connective so that it starts with the start of the first, and finishes with the end of the second. Then, we simply join the end of the first with the beginning of the second, and we have the behavior we want. There is no need to concern ourselves with what goes on inside each component because the flow relation specific to that component will govern progress through it, and the component places and transitions are not disturbed by joining with another net since the join happens only at the start and/or end. Thus, by distinguishing the start and end of each component, we can more easily and reliably define the behavior of the nets when they are joined by some connective. The chainable petri net, or CNET, 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.

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?

BA: did we make monads here?

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 *sequential 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:

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

The components of $B; A$ are:

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

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 $\{(f_A \rightarrow i_B)\} \in T_{A;B}$, the construction of $B; A$ includes $\{(f_B \rightarrow i_A)\} \in T_{B;A}$. 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. \square

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 $C = (S_C^\oplus, T_C, i_C, f_C, start_C, end_C)$. 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. \square

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

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 $end_{1+2} = end_1 + end_2$.
- $S = S_1 + S_2 + i_{1+2} + f_{1+2}$.
- $T = T_1 + T_2$.
- $[src_1, src_2], [tar_1, tar_2] : T \longrightarrow S^\oplus$.

BA: does this make sense?

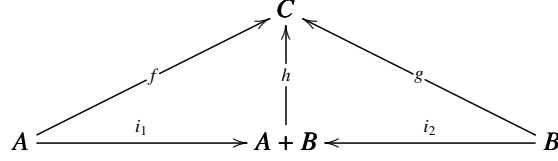
BA: is f_{1+2} really necessary here? already in $S_1 + S_2$

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

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 .

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 $A = (P_A, T_A, \text{src}_A, \text{tar}_A, i_A, f_A)$ and $B = (P_B, T_B, \text{src}_B, \text{tar}_B, i_B, f_B)$, 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, $i_A \rightarrow i_A \times i_B$ and $i_B \rightarrow i_A \times i_B$ 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.

$[\text{src}_1, \text{src}_2], [\text{tar}_1, \text{tar}_2] : T \rightarrow P^\oplus$. 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.

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

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

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 $k : (A + B) \rightarrow C$, where $i_1; k = f$ and $i_2; k = g$. For $a \in A$ and $b \in B$, the following equalities hold:

$$i_1(a); k(a) = f(a) = h(a) \quad (2)$$

$$i_2(b); k(b) = g(b) = h(b) \quad (3)$$

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, $N_1 = (P_1, T_1, \text{src}_1, \text{tar}_1, i_1, f_1)$ and $N_2 = (P_2, T_2, \text{src}_2, \text{tar}_2, i_2, f_2)$, their **synchronous conjunction** is $N_1 \parallel N_2$, where:

- The initial place $i = i_1 \times i_2$.
- The final place $f = f_1 \times f_2$.
- The beginning transition $\text{start} = \text{start}_1 \times \text{start}_2$.
- The ending transition $\text{end} = \text{end}_1 \times \text{end}_2$.
- $P = P_1 \times P_2$.

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?

- $T = T_1 \times T_2$.
- $[\text{src}_1, \text{src}_2], [\text{tar}_1, \text{tar}_2] : T \longrightarrow P^\oplus$.

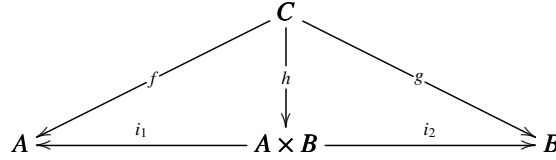
BA: Do I need this part?

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.

Lemma 20. The synchronous conjunction of two CNETs, $A \parallel B$, is a product.

Proof. $N_1 \parallel N_2$ 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:

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



Let $A = (P_A, T_A, \text{src}_A, \text{tar}_A, i_A, f_A)$ and $B = (P_B, T_B, \text{src}_B, \text{tar}_B, i_B, f_B)$, and take $A \parallel B$. Let us define i_1 and i_2 , which are projections for Petri nets. We will take it by parts for clarity.

$P_{A \parallel B} = (P_1 \times P_2) + i + f$. 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. $T = (T_1 \times T_2) + \text{start} + \text{end}$. 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. \square

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, $N_1 = (P_1, T_1, \text{src}_1, \text{tar}_1, i_1, f_1)$ and $N_2 = (P_2, T_2, \text{src}_2, \text{tar}_2, i_2, f_2)$, 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 $T = \{(s_1, s_2) \longrightarrow s_3\}$.
- 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, $N_1 = (P_1, T_1, \text{src}_1, \text{tar}_1, i_1, f_1)$ and $N_2 = (P_2, T_2, \text{src}_2, \text{tar}_2, i_2, f_2)$, 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 $A = (P_A, T_A, \text{src}_A, \text{tar}_A, i_A, f_A)$ and $B = (P_B, T_B, \text{src}_B, \text{tar}_B, i_B, f_B)$, 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 $C = (P_C, T_C, \text{src}_C, \text{tar}_C, i_C, f_C)$ and take $(A \amalg B) \amalg C$.

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

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

- $P_{A \amalg (B \amalg C)} = P_A + (P_B + P_C)$

- $T_{\text{All}(B \sqcup C)} = T_A + (T_B + T_C)$
- $\text{start}_{\text{All}(B \sqcup C)} = \text{start}_A + (\text{start}_B + \text{start}_C)$
- $\text{end}_{\text{All}(B \sqcup C)} = \text{end}_A + (\text{end}_B + \text{end}_C)$
- $i_{\text{All}(B \sqcup C)} = i_A \times (i_B \times i_C)$
- $f_{\text{All}(B \sqcup C)} = f_A \times (f_B \times f_C)$

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. \square

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

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,

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

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,

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

Finally, let us check that $g; f = \text{Id}$ and $f; g = \text{Id}$.

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

Thus, $g; f = \text{Id}$. Likewise,

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

Thus, $f; g = \text{Id}$, and $((A \times B) \times C) \cong (A \times (B \times C))$ \square

9 Equational Representation of Petri Nets

In [12], 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 $f : S \times \pi_b \rightarrow S$ 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. [11]

10 Quantitative Analysis of Petri Nets

[21] is not that informative but a good overview. [2] develops a Priced Timed Petri Net, which is a workflow net (that has the in/out places like our CNETs) with the added features of cost and time-weighted transitions. Gives detailed algorithms for methods of calculating net cost with one or many final places, with nets that have added features like timed transitions and tokens, inhibitor/enabler functionality, etc.

- Define a valuation function $V : T \rightarrow \mathbb{R}$ that associates a cost with each transition.
- Define a calculation function C_V that combines these values.
 - For cost or time calculation, this should be a straightforward sum of all the transitions on each possible path. If more than one possible path exists, could give the range.
- From each leaf, calculate all possible paths to the root. This is called a reachability graph.
- Let $\{t_0, \dots, t_n\}$ be the transitions that fire along a path from a leaf to the root.
- $C_V = \sum_{i=0}^n V(t_i)$
- Apply the valuation function and the calculation function, may need a way to decide between paths or can give a range if multiple values.

For probability, [10] define Probability Propagation Nets, which assign a probability to each transition (among other things not relevant to our concerns).

- Define a probability function $P : T \rightarrow \mathbb{R}$ that associates a probability with each transition.

- Define a calculation function C_P that combines these values.
- From each leaf, calculate the reachability graph.
- Let $\{t_0, \dots, t_n\}$ be the transitions that fire along a path from a leaf to the root.
- $C_P = \prod_{i=0}^n P(t_i)$

Note that the reachability tree, basically calculating all the possible paths and picking the shortest one, is a complex problem in itself. There are resources giving different algorithms to efficiently find the shortest path. [11]

11 Related Work

12 Petri-Nets-Categorically

In [3], the authors define Open Petri nets as Petri nets with the addition of designated input and output places in order to study the reachability semantics of Petri nets, a way of calculating what outputs are possible given some inputs to a Petri net. They follow the work of Meseguer and Montanari in looking at Petri nets as symmetric monoids, but further develop the application of category theory to Petri nets by understanding Open Petri nets as a double category, which allows them to ???not sure exactly, need to read and think some more??? The authors' notion of an Open Petri net, with input and output places, is similar to our CNETs, though they are concerned with a different problem. We formulated CNETs to build up attack trees, whereas their concern is with reachability in a pre-existing net. Thus their work and ours is complementary; their reachability semantics is a powerful tool to analyze the CNETs we can build.

In [15], the authors take a category-theoretical approach to study concurrency in Petri net models. Making the point that Petri nets can be understood as directed graphs, they introduce a category for Petri nets generally, and also for pointed Petri nets, which have an initial marking to show the beginning of the net. They also work through the categorical product and coproduct for Petri nets, and show that Petri nets are monoids where the objects are the places and the relation is given by the transitions. Our work builds on theirs by continuing the categorical approach. We also built upon their pointed Petri net to define our CNET, which is a Petri net with both a start and end place to allow for chaining. They develop two operators for the pointed Petri net, */oplus* and *;*, which we further developed for our CNET operators AND and SEQ, respectively.

In [18], the author surveys his own contributions to algebraic, compositional approaches to the semantics of Petri nets, approaches that, broadly speaking, develop a view of Petri nets as complex structures built up from simple parts and a limited number of connection types, or combinators. The author follows the development of the field from its beginnings, starting with the application of process algebras and categorical algebra to Petri nets. In particular, his research has focused on concatenable Petri nets, and ultimately has followed Meseguer and Montanari, 1990 into the categorical approach to Petri nets (as monoidal categories). This paper provides a helpful and thorough history of ideas that our paper builds upon. We take a very similar approach to

solving the problem of applying Petri nets to attack trees, but Sassone is not concerned with this specific problem.

In [19] Sassone adds to the categorical understanding of Petri nets, building on the conceptualization of concatenable processes as morphisms (or what he calls "arrows") of a symmetric monoid. He discusses the complications of concatenating processes when done by merging the first net's maximal places (the final marking in our paper) with the next net's minimal places (what we call the initial marking). The difficulty arises because the place-to-place correspondence can be done in multiple ways, resulting in different concatenation behaviors. His solution to this problem is to focus on the tokens that move through the net, rather than the places. Within this line of reasoning, he points out that the category of the concatenable processes of a net are themselves a symmetric monoidal category with a tensor product yielding the parallel composition of processes, which in turn shows that Petri net behavior can be understood algebraically. He proposes to take the work of Deguano, Meseguer, and Montanari (1989) and make it completely abstract and axiomatic by establishing a category of symmetries of the Petri net category. While this research is not concerned strictly with what we are examining, it provides some helpful insights into concatenation and parallelization in Petri nets viewed categorically.

In [6], according to the authors' interpretation of the field, the research into the semantics of place/transition Petri nets can be divided into two camps: collective and individual token philosophies. Whereas the collective side does not distinguish between different instances of tokens at the same place, reasoning that what really matters is whether a token occupies the place or not (i.e. for the firing of the next transition), the individual side does pay attention to the particulars of the token - especially as its identity relates to its causal history. The authors identify an issue with the algebraic semantics of place/transition Petri nets interpreted under the individual token philosophy, showing that it lacks universality and functoriality. To address this, they introduce the idea of pre-nets, whose states are strings of tokens (which are total orders) rather than the multisets we use in our work. However, our understanding of Petri nets is strongly based on pomsets as expressive of a particular path through the net, and so their pre-nets and the associated category-theoretical apparatuses might prove useful to us. Furthermore, our research falls more into the individual token philosophy, so our theory may be vulnerable to the weakness they describe and repair with the pre-net. This article may prove important in the theory of our paper, but our paper has a narrower application and develops the theory of Petri nets in a direction that they do not explore.

13 Attack-Trees

The seminal work in attack trees is [20]. In this paper, the author proposes using a tree as a way to model cyber threats. By clearly enumerating and describing the vulnerabilities of a given resource, he hopes to facilitate defense against those vulnerabilities. In attack trees, the goal is placed at the root, and the paths up the tree represent the different ways of achieving that goal. Like trees in general, the structure is hierarchical, with goals comprising subgoals, and in turn being part of larger goals. The child

trees can be joined by AND nodes or OR nodes, with these carrying the usual meaning. He goes on to describe a simple calculation of "possible" or "impossible" paths, obtained by assigning a mark of either possible or impossible to each leaf. Then, if one child of an OR node is possible, the OR node is marked possible. Both children of an AND node must be marked possible for the parent to be possible. This calculation pattern is repeated up the tree until all paths are calculated, giving a rough idea of how secure a resource is. Schneier then demonstrates a similar process for calculating the cost of each path. He then shows how such calculations can inform cyber defense, for example by targeting the cheapest attacks first, or considering how to make a path more expensive to follow. Having established the utility of attack trees, the author goes on to explain how to construct them. This process breaks the single goal of compromising the resource into smaller and smaller sub-goals until the paths are fully developed and exposed. He also recommends collaboration in the construction of attack trees, to provide a fresh perspective to uncover all possible attacks. Then, he defines the strengths of the attack tree model, highlighting its ability to uncover unexpected vulnerabilities and its potential for reuse in similar situations or in the protection of similar resources. He also points out that the reuse feature means that professionals who lack expertise in a particular area can still mount effective defenses in those areas by relying on the expertise of those who made the pattern.

14 Modeling-Attack-Trees-as-Petri-Nets

In [13], the authors are doing research very similar to our own. They recognize the value of attack-defense trees, a refinement of attack trees in which both attacks and defenses can be modeled, but also see the limitations of existing semantics for ADTrees. Thus, in this paper they develop a semantics based on stochastic Petri nets that can capture execution, order of steps, and dependencies. In stochastic Petri nets, transition firings are delayed by an assigned or randomized amount of time. Relying on the isomorphism between finite continuous time Markov chains and bounded stochastic Petri nets, they show how to quantitatively analyze ADTrees. And, using the established analysis methods for Petri nets, they show how to qualitatively analyze ADTrees. They also define a new operator for parallel execution of two branches in a Petri net. In their semantics, they distinguish a set of initial and final places for each stochastic Petri net (SPN), and then begin to construct an SPN from an ADTree by starting with the leaf nodes, building a single-transition net with an initial place and a final place. Conjunction they define as the Cartesian product of the conjuncts, where the initial places and final places are each paired to become the initial and final places of the conjunction, and then the places and transitions are unioned together. This is how we defined conjunction for CNETs, except that these authors have the added element of stochasticity. They define disjunction by merging the initial place of all disjuncts into one, and then allowing the final markings of each branch to count as the final marking for the disjunction. This again follows our construction. They define sequential composition in a similar way, by merging the final place of the first with the initial place of the second. Their parallel disjunction operator brings together the initial places of each branch, but then requires only one branch to complete. They then develop semantics for counter-

BA: I don't understand their notation for this, but if I'm reading it correctly I also don't understand why they did this - why

measures, and work through an example to show the effectiveness of their method. Our paper shares a lot of material with this paper; however, we do define other connectives that they do not have, such as parallel and synchronous conjunction. Also we approach the subject categorically and offer proofs of the properties of our connectives.

In [14], the author provides a brief introduction into penetration testing to discover potential attacks on a system, and discusses two approaches: flaw hypothesis and attack tree. Flaw hypothesis analyzes a system within a particular scope, generates hypothetical flaws and then tests for them, using any confirmed flaws as a basis for generalization (to find similar flaw patterns in other areas of the system) and further testing. Finally, any confirmed flaws are eliminated. The attack tree approach uses a tree graph with the goal as the root, and then the steps to achieving the goal as the branches and/or leaves. In this paper, McDermott proposes organizing an attack tree more formally as a Petri net, with places representing states of the system, and transitions representing actions or events that cause a change in state. He describes the attack net approach as contributing several strengths to penetration testing: modeling concurrency and the current state of an attack with tokens, showing intermediate goals more clearly as places, and showing the cumulative effects of different avenues of attack on the whole system. This is clearly a foundational article for our research. Besides establishing the central idea of modeling attack trees as Petri nets, this article also suggests important features for our system that we have worked to incorporate, such as using a flaw database to pattern-match attacks, and building up nets from simple attacks to more complex ones. Our work develops the central idea more specifically through our models of different connectives and our use of category theory, but this article is invaluable as a basis for our further efforts.

Chowdhury, Farida and Ferdous, Md. Sadek. (2017). Modelling Cyber Attacks. International Journal of Network Security and Its Applications. 9. 13-31. 10.5121/ijnsa.2017.9402
In this article, the authors aim to develop a model of cyber attacks which they claim is more comprehensive because it accounts for features of cyber attacks that other models do not include. They break the attack into three components: the attack itself, the target at which it is directed, and the victim (the entity that controls or otherwise is concerned with the fate of the target). Each attack is either visual, network-based, or a hybrid. Following this, they label each attack with the layer of the TCP/IP Protocol (or DARPA model) that it targets. They also distinguish between active attacks, where the target can be modified, and passive attacks, where the target can merely be observed. The authors codify several possible motivations an attacker might have, and develop taxonomies of possible attacks based on these motivations and general vulnerabilities a system might have. They gather each of these characteristics into a tuple of the form ***see picture to get Latex right***. While the authors are working to develop a mathematical model that captures important features of an attack, their model is essentially qualitative in nature, meaning that it is more of a classification system than a truly mathematical model. Our model is more detailed and allows for more effective analysis and comparison as a result. That being said, the model presented here does provide a helpful, detailed mechanism for general attack classification.

In [7, ?], the authors apply Petri nets as a model for attack trees, identifying a major weakness in using this model: the labor-intensive process of translating a visual attack tree into a mathematical Petri net. They propose to resolve this difficulty by creating

BA: Make a
BibTeX entry
for this one

a large Petri net out of many smaller Petri nets, all created by different people with different areas of expertise. The joining of one Petri net to another is accomplished by matching a place in one Petri net with its identical place in another Petri net. While their approach is, on its face, very similar to ours, ours provides a more effective solution because it relies on attack patterns, which are more abstract. This facilitates reuse of our component trees, reducing error and labor. It also makes it possible to analyze of the overall structure for equivalence and hierarchy, which adds important functionality for cybersecurity professionals.

BA: need to read more to understand how they establish identity

In [8], the authors describe the structure of attack trees and survey earlier attempts to represent them mathematically in (Mauw and Oostdijk 2005) and (Linger and Moore 2001). They also discuss using XML files to represent attack trees textually, in outline form, to facilitate automated analysis and simulation. They point out flaws with the attack tree model, particularly the lack of a standard form for the model, which hinders attack tree reuse. Then, they give an overview of Petri nets and show several behaviors that the Petri net can capture: concurrency, conflict (limitation and competition regarding resources), locality (some transitions only affect one area of the net, not the entire thing), boundedness (limitation on the number of tokens a place can hold), and liveness (no transition will become permanently unfireable). Moving to timed Petri nets, the authors explain that stochastic Petri nets provide random delay factors for all transitions in the net. They use these to simulate the net's runs and analyze the probability of various markings. They then describe a method for modeling attack trees as Petri nets, describing conjunction and disjunction patterns. However, their patterns lack the initial and final places of our CNETs. Also, they explain that implementing the conjunction requires consideration of the number of tokens in the net, since a pair of tokens is needed at once to activate classical conjunction. They offer sequential conjunction as a workaround for this, but in fact this is a conflation of two different kinds of conjunction, which captures two different behaviors. Also, building a Petri net model of an attack tree in their system requires a lot of thought, because their small-scale connectives are not chainable.

In [9], the authors provide an survey of directed acyclic graphical models for security. Some notable models they include are attack trees and various extensions, Bayesian attack/defense graphs, and Boolean logic driven Markov processes. Clearly, their scope in this paper is wide. They give an account of Petri nets in security-related applications as well. They point out that Petri nets have many specializations and extensions that allow them to be used in a variety of contexts, and provide bibliographic information to demonstrate that fact. They survey existing literature on applications of Petri nets

15 Workflow-Nets

In [1], the authors describe a type of Petri net called a workflow net that is very similar to our chainable Petri net. We did not discover this subclass of Petri nets before defining our CNETs because workflow nets are used to model processes and workflows (as the name implies) in areas such as business and manufacturing, whereas our focus has been in security. Like our CNETs, workflow nets have their initial and final places

marked, and every node is on a path between the initial and final places. Also, every transition must be live, meaning that it has to be enabled at some reachable marking. The last two requirements are not explicit in our CNETS, but seem to be a natural occurrence anyway given the way we aggregate small CNETS into bigger ones, and the general premise that we are mapping paths to a goal. How much CNETs will adhere to these requirements needs further study, as does the question of how closely process trees match attack trees. Interestingly, in this article the authors describe a process of translating a workflow net into a process tree, moving in the opposite direction from our research. Still, as a kind of dual to our research, this can offer important insights. The authors define the connectives of sequential composition, exclusive choice, and parallel composition using graphs, without the precision of defining each element in the Petri net tuple as we do in our paper. They omit these details because their focus is on the translation to process trees. They show that a workflow net is convertible to a process tree if it is acyclic, has no bridges, and is well-structured. It must be acyclic because the net must be able to complete in order to become a tree, since the tree structure includes completion as its structuring premise. Well-structuredness means that two paths coming off of a transition should not meet up in a single place, and two paths coming off of a single place should not meet up in a single transition. The authors then give an algorithm for translating a workflow net into a process tree. By evaluating the number of paths into and out of a node, together with the type of node (transition or place) they can infer the structure of the process tree, which their algorithm outputs as a formula in prefix notation. While this paper does cover similar ground to our research, our work still makes important additional contributions to the field because it applies a similar concept to a new area (cybersecurity) where it can be tremendously useful, and also makes the construction/aggregation of CNETs mathematically precise.

References

- [1] Convertibility and conversion algorithm of well-structured workflow net to process tree. *Computing and Networking (CANDAR), 2013 First International Symposium on*, December 2013.
- [2] Parosh Aziz Abdulla and Richard Mayr. Minimal cost reachability/coverability in priced timed petri nets. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures*, pages 348–363, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [3] J. C. Baez and J. Master. Open Petri Nets. *ArXiv e-prints*, August 2018.
- [4] 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.
- [5] 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.

- [6] Roberto Bruni, Jos Meseguer, Ugo Montanari, and Vladimiro Sassone. Functorial models for petri nets. *Information and Computation*, 170(2):207 – 236, 2001.
- [7] T. Chen, J. C. Sanchez-Aarnoutse, and J. Buford. Petri net modeling of cyber-physical attacks on smart grid. *IEEE Transactions on Smart Grid*, 2(4):741 – 749, 2011. © 2011 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
- [8] Dalton, Mills, Colombi, and Raines. Analyzing attack trees using generalized stochastic petri nets. In *2006 IEEE Information Assurance Workshop*, pages 116–123, June 2006.
- [9] Barbara Kordy, Ludovic Pitre-Cambacds, and Patrick Schweitzer. Dag-based attack and defense modeling: Dont miss the forest for the attack trees. *Computer Science Review*, 13-14:1 – 38, 2014.
- [10] Kurt Lautenbach and Alexander Pinl. Probability propagation in petri nets. 11 2018.
- [11] 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).
- [12] Bruno Lopes, Mario Benevides, and Edward Hermann Haeusler. Propositional dynamic logic for petri nets. *Logic Journal of the IGPL*, 22(5):721–736, 2014.
- [13] Karim Lounis. Stochastic-based semantics of attack-defense trees for security assessment. *Electronic Notes in Theoretical Computer Science*, 337:135 – 154, 2018. Proceedings of the Ninth International Workshop on the Practical Application of Stochastic Modelling (PASM).
- [14] J. P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms*, NSPW '00, pages 15–21, New York, NY, USA, 2000. ACM.
- [15] Jos Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105 – 155, 1990.
- [16] Mogens Nielsen, Lutz Priebe, 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.
- [17] Srdjan Pudar, G. Manimaran, and Chen-Ching Liu. Penet: A practical method and tool for integrated modeling of security attacks and countermeasures. *Comput. Secur.*, 28(8):754–771, November 2009.

- [18] V. Sassone. On the algebraic structure of petri nets. 2000.
- [19] Vladimiro Sassone. An axiomatization of the algebra of petri net concatenable processes. *Theoretical Computer Science*, 170(1):277 – 296, 1996.
- [20] Bruce Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.
- [21] Armin Zimmermann. *Stochastic Discrete Event Systems*. Springer Berlin Heidelberg, 1 edition, 1990.