Proposing a New Foundation of Attack Trees in Monoidal Categories

Harley Eades III

Computer and Information Sciences, Augusta University, Augusta, GA,
heades@augusta.edu

Abstract. This short paper introduces a new project studying at the intersection of threat analysis using attack trees and interactive theorem proving using linear logic. The project proposes a new semantics of attack trees in dialectica spaces, a well-known model of intuitionistic linear logic, which offers two new branching operators to attack trees. Then by exploiting the Curry-Howard-Lambek correspondence it seeks to develop a domain-specific linear functional programming language called Lina – for Linear Threat Analysis – for specifying and reasoning about attack trees.

1 Introduction

What do propositional logic, multisets, directed acyclic graphs, source sink graphs (or parallel-series pomsets), Petri nets, and Markov processes all have in common? They are all mathematical models of attack trees – see the references in [12,11] – but also, they can all be modeled in some form of a symmetric monoidal category[1] [20,2,6,7] – for the definition of a symmetric monoidal category see Appendix A. Taking things a little bit further, monoidal categories have a tight correspondence with linear logic through the beautiful Curry-Howard-Lambek correspondence [1]. This correspondence states that objects of a monoidal category correspond to the formulas of linear logic and the morphisms correspond to proofs of valid sequents of the logic. I propose that attack trees – in many different flavors – be modeled as objects in monoidal categories, and hence, as formulas of linear logic.

The Curry-Howard-Lambek correspondence is a three way relationship:

Categories	$\Longleftrightarrow$	Logic	$\Longleftrightarrow$	Functional Programming
Objects	$\Longleftrightarrow$	Formulas	$\Longleftrightarrow$	Types
Morphisms	$\Longleftrightarrow$	Proofs	$\Longleftrightarrow$	Programs

By modeling attack trees in monoidal categories we obtain a sound mathematical model, a logic for reasoning about attack trees, and the means of constructing a functional programming language for defining attack trees (as types), and constructing semantically valid transformations (as programs) of attack trees.

[1] I provide a proof that the category of source sink graphs is monoidal in Appendix B.

Linear logic was first proposed by Girard [8] and was quickly realized to be a theory of resources. In linear logic, every hypothesis must be used exactly once. Thus, formulas like $A \otimes A$ and A are not logically equivalent – here $\otimes$ is linear conjunction. This resource perspective of linear logic has been very fruitful in computer science and lead to linear logic being a logical foundation of processes and concurrency where formulas may be considered as processes. Treating attack trees as concurrent processes is not new; they have been modeled by event-based models of concurrency like Petri nets and partially-ordered multisets (pomsets) [11,13]. In fact, pomsets is a model in which events (the resources) can be executed exactly once, and thus, has a relationship with linear logic [16]. However, connecting linear logic as a theory of attack trees is novel, and strengthens this perspective.

Girard's genius behind linear logic was that he isolated the structural rules – weakening and contraction – by treating them as an effect and putting them inside a comonad called the of-course exponential denoted $!A$. In fact, $!A \otimes !A$ is logically equivalent to $!A$, and thus, by staying in the comonad we become propositional. This implies that a model of attack trees in linear logic also provides a model of attack trees in propositional logic, and a combination of the two. It is possible to have the best of both worlds.

In this short paper I introduce a newly funded research project[2] investigating founding attack trees in monoidal categories, and through the Curry-Howard-Lambek correspondence deriving a new domain-specific functional programming language called Lina for Linear Threat Analysis. I begin by defining an extension – inspired by our semantics – of the attack trees given in [11] in Section 2. Then I introduce a new semantics of attack trees in dialectica spaces, which depends on a novel result on dialectica spaces, in Section 3. The final section, Section 4, discusses Lina and some of the current problems the project seeks to answer.

2 Attack Trees

In this paper I consider an extension of attack trees with sequential composition which are due to Jhawar et al. [11], but one of the projects ultimate goals is to extend attack trees with even more operators driven by our choice of semantics. The syntax for attack trees is defined in the following definition.

Definition 1. *The following defines the syntax of* ***Attack Trees*** *given a set of base attacks* $b \in \mathsf{B}$*:*

$$t ::= b \mid t_1 \odot t_2 \mid t_1 \sqcup t_2 \mid t_1 \rhd t_2 \mid t_1 \otimes t_2 \mid \copyright t$$

I denote unsynchronized non-communicating parallel composition of attacks by $t_1 \odot t_2$*, choice between attacks by* $t_1 \sqcup t_2$*, sequential composition of attacks by* $t_1 \rhd t_2$*,*

[2] This material is based upon work supported by the National Science Foundation CRII CISE Research Initiation grant, "CRII:SHF: A New Foundation for Attack Trees Based on Monoidal Categories", under Grant No. 1565557.

and two new operators called unsynchronized interacting parallel composition, denoted $t_1 \otimes t_2$*, and copy, denoted* $ⓒt$*.*

The following rules define the attack tree reduction relation:

$$\frac{}{(t_1 \mathsf{op}\, t_2)\, \mathsf{op}\, t_3 \rightsquigarrow t_1\, \mathsf{op}(t_2\, \mathsf{op}\, t_3)}\ \textsc{assoc} \qquad \frac{}{t_1\, \mathsf{op_S}\, t_2 \rightsquigarrow t_2\, \mathsf{op_S}\, t_1}\ \textsc{sym}$$

$$\frac{}{ⓒt \otimes ⓒt \rightsquigarrow ⓒt}\ \textsc{copy} \qquad \frac{}{(t_1 \sqcup t_2) \odot t_3 \rightsquigarrow (t_1 \odot t_3) \sqcup (t_2 \odot t_3)}\ \textsc{dist}_1$$

$$\frac{}{(t_1 \sqcup t_2) \rhd t_3 \rightsquigarrow (t_1 \rhd t_3) \sqcup (t_2 \rhd t_3)}\ \textsc{dist}_2$$

where $\mathsf{op} \in \{\odot, \otimes, \rhd, \sqcup\}$ *and* $\mathsf{op_S} \in \{\odot, \otimes, \sqcup\}$*. The previous rules can be applied on any well-formed subattack tree, and can be straightforwardly extended into an equivalence relation.*

The syntax given in the previous definition differs from the syntax used by Jhawar et al. [11]. First, I use infix binary operations, while they use prefix n-ary operations. However, it does not sacrifice any expressivity, because each operation is associative, and parallel composition, choice, and interacting parallel composition are symmetric. Thus, Jhawar et al.'s definition of attack trees can be embedded into the ones defined here.

The second major difference is that the typical parallel composition operator found in attack trees is modeled here by unsynchronized non-communicating parallel composition which happens to be a symmetric tensor product, and not a disjunction. This is contrary to the literature, for example, the parallel operation of Jhawar et al. defined on source sink graphs [11] can be proven to be a coproduct – see Appendix B – and coproducts categorically model disjunctions. Furthermore, parallel composition is modeled by multiset union in the multiset semantics, but we can model this as a coproduct. However, in the semantics given in the next section if we took parallel composition to be a coproduct, then the required isomorphisms necessary to model attack trees would not exist.

The third difference is that I denote the choice between executing attack t_1 or attack t_2, but not both, by $t_1 \sqcup t_2$ instead of using a symbol that implies that it is a disjunction. This fits very nicely with the semantics of Jhawar et al., where they collect the attacks that can be executed into a set. The semantics I give in the next section models choice directly.

The fourth, and final, difference is that I extend the syntax with two new operators called unsynchronized communicating parallel composition and copy. The attack $t_1 \otimes t_2$ states that t_1 interacts with the attack t_2 in the sense that processes interact. Modeling interacting attacks allows for the more refined modeling of security critical systems, for example, it can be used to bring social engineering into the analysis where someone communicates malicious information or commands to a unsuspecting party. As a second example, interacting parallel composition could be used to model interacting bot nets.

The attack $ⓒt$ indicates that attack t can be copied and contracted. For example, $ⓒt \otimes ⓒt$ is equivalent to $ⓒt$. Thus, the attack trees given here can treat attack trees as processes/resources that cannot be freely copied and deleted,

as propositions that can be, and as a mixture of the two. Semantically, ⓒt is equivalent to the of-course exponential from linear logic mentioned in the introduction.

The reduction rules are a slightly extended version of equivalences given in Jhaware et al. [11] – Theorem 1. The main difference is the COPY rule which allows copies made by the copy operator to be contracted.

3 Semantics of Attack Trees in Dialectica Spaces

I now introduce a new semantics of attack trees that connects their study with a new perspective of attack trees that could highly impact future research: intuitionistic linear logic, but it also strengthens their connection to process calculi. This section has been formalized in the proof assistant Agda[3]. The semantics is based on the notion of a dialectica space:

Definition 2. *A **dialectica space** is a triple (A, Q, δ) where A and Q are sets and $\delta : A \times Q \to \mathsf{3}$ is a multi-relation where $\mathsf{3} = \{0, \perp, 1\}$ and $\perp$ represents undefined.*

Dialectica spaces can be seen as the intuitionistic cousin [4] of Chu spaces [15]. The latter have be used extensively to study process algebra and as a model of classical linear logic, while dialectica spaces and their morphisms form a categorical model of intuitionistic linear logic called $\mathsf{Dial_3(Sets)}$ (originally due to de Paiva [3]); I do not introduce dialectica space morphisms here, but the curious reader can find the definition in the formal development. I will use the intuitions often used when explaining Chu spaces as processes to explain dialectica spaces as processes, but it should be known that these intuitions are due to Pratt and Gupta [9].

Intuitively, a dialectica space, (A, Q, δ), can be thought of as a process where A is the set of actions the process will execute, Q is the set of states the process can enter, and for $a \in A$ and $q \in Q$, $\delta(a, q)$ indicates whether action a can be executed in state q.

The interpretation of attack trees into dialectica spaces requires the construction of each operation on dialectica spaces:

Parallel Composition. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct – due to de Paiva [5] – the dialectica space $\mathcal{A} \odot \mathcal{B} = (A \times B, Q \times R, \alpha \odot \beta)$ where $(\alpha \odot \beta)((a, b), (q, r)) = \alpha(a, q) \otimes_3 \beta(b, r)$ and $\otimes_3$ is the symmetric tensor product definable on $\mathsf{3}$[4]. Thus, from a process perspective we can see that $\mathcal{A} \odot \mathcal{B}$ executes actions of $\mathcal{A}$ and actions of $\mathcal{B}$ in parallel. Parallel composition is associative and symmetric.

---

[3] The complete formalization can be found at `https://github.com/heades/dialectica-spaces/tree/GraMSec16` which is part of a general library for working with dialectica spaces in Agda developed with Valeria de Paiva.

[4] See the formal development for the full definition: `https://github.com/heades/dialectica-spaces/blob/GraMSec16/concrete-lineales.agda#L328`

Choice. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the dialectica space $\mathcal{A} \sqcup \mathcal{B} = (A + B, Q + R, \alpha \sqcup \beta)$ where $(\alpha \sqcup \beta)(i, j) = \alpha(i, j)$ if $i \in A$ and $j \in Q$, $(\alpha + \beta)(i, j) = \beta(i, j)$ if $i \in B$ and $j \in R$, otherwise $(\alpha + \beta)(i, j) = 0$. Thus, from a process perspective we can see that $\mathcal{A} \sqcup \mathcal{B}$ executes either an action of $\mathcal{A}$ or an action of $\mathcal{B}$, but not both. Choice is symmetric and associative, but it is not a coproduct, because it is not possible to define the corresponding injections. Brown et. al. show that Petri nets can be modeled in dialectica spaces [2], but they use the coproduct as choice. The operator given here is actually the definition given for Chu spaces [9]. If we were to use the coproduct, then we would not be able to prove that choice distributes over parallel composition nor over sequential composition. As far as I am aware, this is the first time this has been pointed out.

Sequential Composition. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ be two dialectica spaces. Then we can construct – due to de Paiva [5] – the dialectica space $\mathcal{A} \rhd \mathcal{B} = (A \times B, Q \times R, \alpha \rhd \beta)$ where $(\alpha \rhd \beta)((a, b), (q, r)) = \alpha(a, q) \,\mathsf{land}\, \beta(i, r)$, and land is lazy conjunction defined for $\mathsf{3}$[5]. This is a non-symmetric conjunctive operator, and thus, sequential composition is non-symmetric. This implies that from a process perspective $\mathcal{A} \rhd \mathcal{B}$ will first execute the actions of $\mathcal{A}$ and then execute actions of $\mathcal{B}$ in that order. Sequential composition is associative.

Interacting Parallel Composition. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the dialectica space $\mathcal{A} \otimes \mathcal{B} = (A \times B, (B \to Q) \times (A \to R), \alpha \otimes \beta)$ where $B \to Q$ and $A \to R$ denote function spaces, and $(\alpha \otimes \beta)((a, b), (f, g)) = \alpha(a, f(b)) \wedge \beta(b, g(a))$. From a process perspective the actions of $\mathcal{A} \otimes \mathcal{B}$ are actions from $\mathcal{A}$ and actions of $\mathcal{B}$, but the states are pairs of maps $f : B \to Q$ and $g : A \to R$ from actions to states. This is the point of interaction between the processes. This operator is symmetric and associative.

Copying. Suppose $\mathcal{A} = (A, Q, \alpha)$ is a dialectica space. Then $\copyright\mathcal{A} = (A, A \to Q^*, \alpha^*)$ where Q^* denotes the free monoid with carrier Q and α^* is the free monoid extension of α. Copying defines a comonad $\copyright : \mathsf{Dial_3(Sets)} \to \mathsf{Dial_3(Sets)}$ on the category of dialectica spaces, and thus, we have dialectica morphisms $\varepsilon : \copyright A \to A$ and $\delta : \copyright A \to \copyright\copyright A$ satisfying the usual diagrams. Furthermore, it has enough structure to show the isomorphism $(\copyright\mathcal{A} \otimes \copyright\mathcal{A}) \cong \copyright\mathcal{A}$. This implies that under $\copyright$ we escape to propositional logic.

At this point it is straightforward to define an interpretation $[\![t]\!]$ of attack trees into $\mathsf{Dial_3(Sets)}$. Soundness with respect to this model would correspond to the following theorem.

Theorem 1 (Soundness). *If $t_1 \rightsquigarrow t_2$, then $[\![t_1]\!] \cong [\![t_2]\!]$ where $\cong$ is isomorphism of objects.*

Those familiar with Chu spaces and their application to process algebra may be wondering how treating dialectica spaces as processes differs. The starkest

[5] See the formal development for the full definition: `https://github.com/heades/dialectica-spaces/blob/GraMSec16/concrete-lineales.agda#L648`

difference is that in this model process simulation is modeled by morphisms of the model, but this is not possible in Chu spaces. In fact, to obtain the expected properties of processes a separate notion of bi-simulation had to be developed for Chu spaces [9]. However, I took great care to insure that the morphisms of our semantics capture the desired properties of process simulation, and hence, attack trees.

The ability to treat morphisms as process simulation was not easy to achieve. The definition of choice in the semantics presented here actually is the definition given for Chu spaces [9], but Brown et al. use the coproduct defined for dialectica spaces to model choice in Petri nets. However, taking the coproduct for choice here does not lead to the isomorphisms $(\mathcal{A} \sqcup \mathcal{B}) \rhd \mathcal{C} \cong (\mathcal{A} \rhd \mathcal{C}) \sqcup (\mathcal{B} \rhd \mathcal{C})$ and $(\mathcal{A} \sqcup \mathcal{B}) \odot \mathcal{C} \cong (\mathcal{A} \odot \mathcal{C}) \sqcup (\mathcal{B} \odot \mathcal{C})$, thus, we will not be able to soundly model attack trees. I have found that if choice is modeled using the definition from Chu spaces [9] then we obtain these isomorphisms which is a novel result[6].

This semantics can be seen as a generalization of some existing models. Multisets, pomsets, and Petri nets can all be modeled by dialectica spaces [2,9]. However, there is a direct connection between dialectica spaces and linear logic which may lead to a logical theory of attack trees.

4 Lina: A Domain Specific PL for Threat Analysis

The second major part of this project is the development of a staticly-typed polymorphic domain-specific linear functional programming language for specifying and reasoning about attack trees called Lina for Linear Threat Analysis. Lina will consist of a core language and a surface language. The core language will include a decidable type checker using term annotations on types. Programming with annotations can be very cumbersome, and so the surface language will use local type inference [14] to alleviate some of the burden from annotations. However, the surface language will provide further conveniences in the form of automation, to be used with labeled attack trees, and graphical representations of attack trees based on the various graphical languages used in category theory [17]. Thus, the security specialist will not be required to program directly in Lina, but instead will use a graphical interface to construct attack trees and prove properties about them in a completely graphical nature.

Types in Lina will correspond to attack trees while programs correspond to semantically valid transformations of attack trees, thus, a question we must answer then is **how do we sufficiently represent the model of attack trees in $\mathsf{Dial}_3(\mathsf{Sets})$ as a linear logic?** The problem is the fact that Lina will require both commutative monoidal operators and non-communicative monoidal operators. Supporting both types of operators within the same logic has been a long standing question. A starting point might be with Reedy's LLMS which

[6] For the proofs see the formal development: `https://github.com/heades/dialectica-spaces/blob/GraMSec16/concurrency.agda#L70` and `https://github.com/heades/dialectica-spaces/blob/GraMSec16/concurrency.agda#L150`

has already been shown to have a categorical model in $\mathsf{Dial}_3(\mathsf{Sets})$ by de Paiva [5]. In fact, the definition of non-interacting parallel composition given here is due to her model. A different path the project plans to develop is to start with a non-communicative linear logic and then split up the of-course exponential into three new exponential operators: one that adds symmetry, one that adds weakening, and one that adds contraction. This would allow for the specification of multiple types of monoidal operators using the various exponentials. Once a proper linear logic is laid out the next step will be to exploit the Curry-Howard-Lambek correspondence to obtain a linear functional programming language making up the core of Lina.

I do not consider this project, particularly Lina, to be at odds with existing work on using automated theorem proving to synthesize and analyze attack trees; see for example [10,18,21,22]. In fact, this project can benefit from automated generation of attack trees. Lina's primary goal is to make reasoning about attack trees safer by having a tight correspondence with the semantics of attack trees, and thus, will allow and help with the creation of attack trees. Lina will offer a manual way for one to create an attack tree, but by leveraging this existing work could allow for their automatic generation, but then could be used to restructure the tree and conduct further analysis in a semantically valid fashion. In addition, Lina can be seen as an interactive theorem prover for attack trees, and so could be used as a proof checker [19] for proof producing SMT backed automated generation of attack trees, thus, potentially allowing for some of the analysis of attack trees in Lina to be automated.

Another goal of this project is to make using Lina as close as possible to functional programming as usual to prevent a large overhead of using the language as well as the tool. As a programming language simplicity is of the utmost importance, and I think with the semantics given here Lina will not require very advanced syntactic features. This cannot be said for some of the existing work that is similar to Lina. For example, Vigo et al. [21] proposed the Quality Tree Generator which requires the user to program in a process calculus which is a non-trivial overhead. At the tool level the goal is to have a completely graphical environment for creating attack trees and reasoning about them by capitalizing on existing graphical reasoning tools from category theory. Thus, at the tool level the user will not have to write any programs at all unless they want to extend the environment.

5 Conclusion

The project described here is to first develop the semantics of attack trees (Section 2) in dialectica spaces (Section 3), a model of full intuitionistic linear logic, and then exploiting the Curry-Howard-Lambek correspondence to develop a new functional programming language called Lina (Section 4) to be used to develop a new tool to conduct threat analysis using attack trees. This tool will include the ability to design and formally reason about attack trees using interactive theorem proving.

References

1. Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1:159–178, 7 1991.
2. Carolyn Brown, Doug Gurr, and Valeria Paiva. A linear specification language for petri nets. *DAIMI Report Series*, 20(363), 1991.
3. Valeria de Paiva. Dialectica categories. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92, pages 47–62. Amerian Mathemaitcal Society, 1989.
4. Valeria de Paiva. Dialectica and chu constructions: Cousins? *Theory and Applications of Categories*, 17(7):127–152, 2006.
5. Valeria de Paiva. Linear logic model of state revisited. *Logic Journal of IGPL*, 22(5):791–804, 2014.
6. Marcelo Fiore and Marco Devesas Campos. *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky: Essays Dedicated to Samson Abramsky on the Occasion of His 60th Birthday*, chapter The Algebra of Directed Acyclic Graphs, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
7. Luisa Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of markov processes. *Applied Categorical Structures*, 19(1):425–437, 2010.
8. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1 – 101, 1987.
9. Vineet Gupta. *Chu Spaces: a Model of Concurrency*. PhD thesis, Stanford University, 1994.
10. D.J. Huistra. Automated generation of attack trees by unfolding graph transformation systems, March 2016.
11. Ravi Jhawar, Barbara Kordy, Sjouke Mauw, SaÅ!'a RadomiroviÄ, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer International Publishing, 2015.
12. Barbara Kordy, Ludovic Piétre-Cambacédés, and Patrick Schweitzer. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, 13â14:1 – 38, 2014.
13. Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In DongHo Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin Heidelberg, 2006.
14. Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Program. Lang. Syst.*, 22(1):1–44, January 2000.
15. Vaughan Pratt. Chu spaces. Notes for the School on Category Theory and Applications University of Cimbra, July 1999.
16. Christian Retoré. *Typed Lambda Calculi and Applications: Third International Conference on Typed Lambda Calculi and Applications TLCA '97 Nancy, France, April 2–4, 1997 Proceedings*, chapter Pomset logic: A non-commutative extension of classical linear logic, pages 300–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
17. Peter Selinger. A survey of graphical languages for monoidal categories. *ArXiv e-prints*, August 2009.

18. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 273–, Washington, DC, USA, 2002. IEEE Computer Society.
19. Aaron Stump, Andrew Reynolds, Cesare Tinelli, Austin Laugesen, Harley D. Eades III, Corey Oliver, and Ruoyu Zhang. Lfsc for smt proofs: Work in progress. In *Proceedings of the Second International Workshop on Proof Exchange for Theorem Proving (PXTP 2012)*, 2012.
20. A Tzouvaras. The linear logic of multisets. *Logic Journal of IGPL*, 6(6):901–916, 1998.
21. R. Vigo, F. Nielson, and H. R. Nielson. Automated generation of attack trees. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 337–350, July 2014.
22. N.H. Wolters. Analysis of attack trees with timed automata (transforming formalisms through metamodeling), March 2016.

Appendix

A Symmetric Monoidal Categories

This appendix provides the definitions of both categories in general, and, in particular, symmetric monoidal closed categories. We begin with the definition of a category:

Definition 3. *A* ***category****, $\mathcal{C}$, consists of the following data:*

- *A set of objects $\mathcal{C}_0$, each denoted by A, B, C, etc.*
- *A set of morphisms $\mathcal{C}_1$, each denoted by f, g, h, etc.*
- *Two functions* src*, the source of a morphism, and* tar*, the target of a morphism, from morphisms to objects. If $\mathsf{src}(f) = A$ and $\mathsf{tar}(f) = B$, then we write $f : A \to B$.*
- *Given two morphisms $f : A \to B$ and $g : B \to C$, then the morphism $f; g : A \to C$, called the composition of f and g, must exist.*
- *For every object $A \in \mathcal{C}_0$, the there must exist a morphism $\mathsf{id}_A : A \to A$ called the identity morphism on A.*
- *The following axioms must hold:*
  - *(Identities) For any $f : A \to B$, $f; \mathsf{id}_B = f = \mathsf{id}_A; f$.*
  - *(Associativity) For any $f : A \to B$, $g : B \to C$, and $h : C \to D$, $(f; g); h = f; (g; h)$.*

Categories are by definition very abstract, and it is due to this that makes them so applicable. The usual example of a category is the category whose objects are all sets, and whose morphisms are set-theoretic functions. Clearly, composition and identities exist, and satisfy the axioms of a category. A second example is preordered sets, $(A, \leq)$, where the objects are elements of A and a morphism $f : a \to b$ for elements $a, b \in A$ exists iff $a \leq b$. Reflexivity yields identities, and transitivity yields composition.

Symmetric monoidal categories pair categories with a commutative monoid like structure called the tensor product.

Definition 4. *A* ***symmetric monoidal category (SMC)*** *is a category, $\mathcal{M}$, with the following data:*

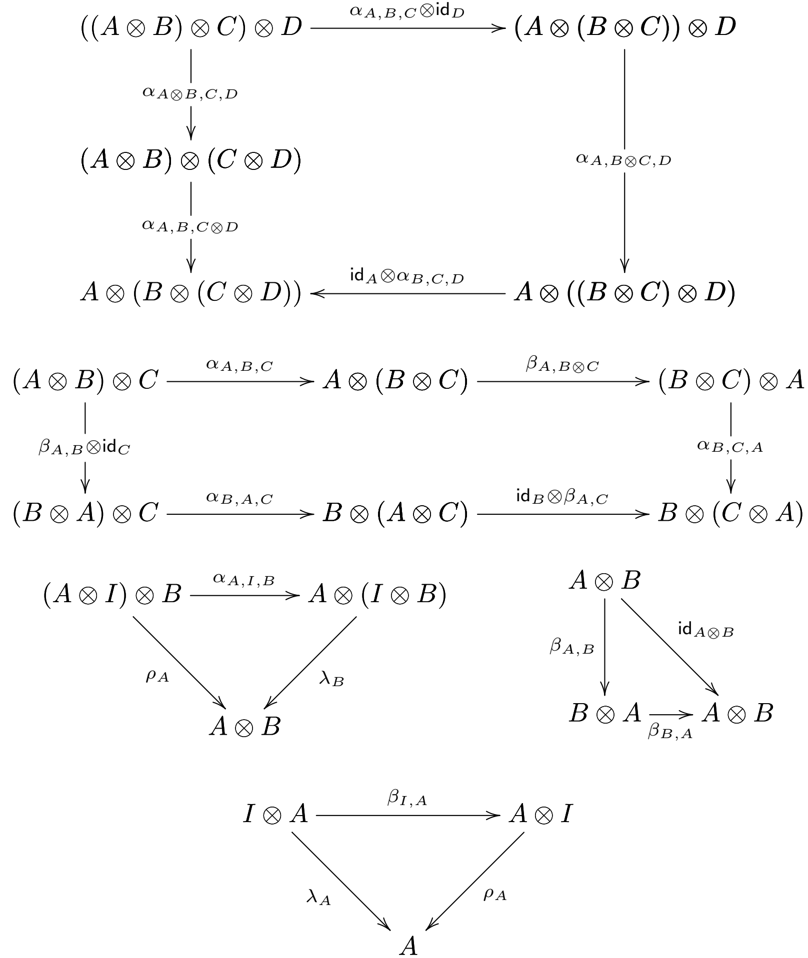
- *An object I of $\mathcal{M}$,*
- *A bi-functor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$,*
- *The following natural isomorphisms:*

$$\begin{array}{l} \lambda_A : I \otimes A \to A \\ \rho_A : A \otimes I \to A \\ \alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C) \end{array}$$

- *A symmetry natural transformation:*

$$\beta_{A,B} : A \otimes B \to B \otimes A$$

- *Subject to the following coherence diagrams:*



B Source Sink Graphs are Symmetric Monoidal

In this appendix I show that the category of source-sink graphs defined by Jhawar et al. [11] is symmetric monoidal. First, recall the definition of source-sink graphs and their homomorphisms.

Definition 5. *A* ***source-sink graph*** *over* B *is a tuple* $G = (V, E, s, z)$*, where* V *is the set of vertices,* E *is a multiset of labeled edges with support* $E^* \subseteq V \times \mathsf{B} \times V$*,* $s \in V$ *is the unique start,* $z \in V$ *is the unique sink, and* $s \neq z$*.*

Suppose $G = (V, E, s, z)$ *and* $G' = (V', E', s', z')$*. Then a* ***morphism between source-sink graphs****,* $f : G \to G'$*, is a graph homomorphism such that* $f(s) = s'$ *and* $f(z) = z'$*.*

Suppose $G = (V, E, s, z)$ and $G' = (V', E', s', z')$ are two source-sink graphs. Then given the above definition it is possible to define sequential and non-communicating parallel composition of source-sink graphs where I denote disjoint union of sets by $+$ (p 7. [11]):

$$\text{(Sequential Composition) } G \rhd G' = ((V \setminus \{z\}) + V', E^{[s'/z]} + E', s, z')$$
$$\text{(Parallel Composition) } G \odot G' = ((V \setminus \{s, z\}) + V', E^{[s'/s, z'/z]} + E', s', z')$$

It is easy to see that we can define a category of source-sink graphs and their homomorphisms. Furthermore, it is a symmetric monoidal category were parallel composition is the symmetric tensor product. It is well-known that any category with co-products is symmetric monoidal where the co-product is the tensor product.

I show here that parallel composition defines a co-product. This requires the definition of the following morphisms:

$$\mathsf{inj}_1 : G_1 \to G_1 \odot G_2$$
$$\mathsf{inj}_2 : G_2 \to G_1 \odot G_2$$
$$\langle f, g \rangle : G_1 \odot G_2 \to G$$

In the above $f : G_1 \to G$ and $g : G_2 \to G$ are two source-sink graph homomorphisms. Furthermore, the following diagram must commute:

$$\begin{array}{ccccc} & & G & & \\ & {\scriptstyle f} \swarrow & \uparrow {\scriptstyle \langle f,g \rangle} & \searrow {\scriptstyle g} & \\ G_1 & \xrightarrow[\mathsf{inj}_1]{} & G_1 \odot G_2 & \xleftarrow[\mathsf{inj}_2]{} & G_2 \end{array}$$

Suppose $G_1 = (V_1, E_1, s_1, z_1)$, $G_2 = (V_2, E_2, s_2, z_2)$, and $G = (V, E, s, z)$ are source-sink graphs, and $f : G_1 \to G$ and $g : G_2 \to G$ are source-sink graph morphisms – note that $f(s_1) = g(s_2) = s$ and $f(z_1) = g(z_2) = z$ by definition. Then we define the required co-product morphisms as follows:

$$\begin{array}{ll} \mathsf{inj}_1 : V_1 \to (V_1 \setminus \{s_1, z_1\}) + V_2 & \mathsf{inj}_2 : V_2 \to (V_1 \setminus \{s_1, z_1\}) + V_2 \\ \mathsf{inj}_1(s_1) = s_2 & \mathsf{inj}_2(v) = v \\ \mathsf{inj}_1(z_1) = z_2 & \\ \mathsf{inj}_1(v) = v, \text{ otherwise} & \end{array}$$

$$\begin{array}{l} \langle f, g \rangle : (V_1 \setminus \{s_1, z_1\}) + V_2 \to V \\ \langle f, g \rangle(v) = f(v), \ \text{where } v \in V_1 \\ \langle f, g \rangle(v) = g(v), \ \text{where } v \in V_2 \end{array}$$

It is easy to see that these define graph homomorphisms. All that is left to show is that the diagram from above commutes:

$$\begin{aligned} (\mathsf{inj}_1; \langle f, g \rangle)(s_1) &= \langle f, g \rangle(\mathsf{inj}_1(s_1)) \\ &= g(s_2) \\ &= s \\ &= f(s_1) \end{aligned} \qquad \begin{aligned} (\mathsf{inj}_1; \langle f, g \rangle)(z_1) &= \langle f, g \rangle(\mathsf{inj}_1(z_1)) \\ &= g(z_2) \\ &= z \\ &= f(z_1) \end{aligned}$$

Now for any $v \in V_1$ we have the following:

$$\begin{aligned} (\mathsf{inj}_1; \langle f, g \rangle)(v) &= \langle f, g \rangle(\mathsf{inj}_1(v)) \\ &= f(v) \end{aligned}$$

The equation for inj_2 is trivial, because inj_2 is the identity.