amU
Aix Marseille Université

NNT : 2025AIXM0000

THÈSE DE DOCTORAT

Soutenue à AMU — Aix-Marseille Université
le 6 Novembre 2025 par

Valentin Maestracci

A small step towards an Open Source Logic:
Investigating models of computation for Linear Realisability

Discipline
Informatique

École doctorale
ED 184 MATHEMATIQUES ET INFORMATIQUE

Composition du jury

Paul-André MELLIES Directeur de Recherche Université Paris-Cité/CNRS	Rapporteur
Shin-Ya KATSUMATA Full Professor Kyoto Sangyo University	Rapporteur
Samuel MIMRAM Professeur des Universités École Polytechnique	Président du jury
Aurore ALCOLEI Maitresse de conférences Université Paris-Est Créteil Val de Marne (UPEC)	Examinatrice
Masahito HASEGAWA Full Professor Kyoto University	Examinateur
Claudia FAGGIAN Chargée de Recherche Université Paris-Cité/CNRS	Examinatrice
Laurent REGNIER Professeur des Université Aix Marseille Université	Directeur de thèse
Thomas SEILLER Chargé de Recherche Université Paris 13/CNRS	co-Directeur de thèse

INSTITUT DE MATHÉMATIQUES DE MARSEILLE
I2M - UMR 7373

Affidavit

I, undersigned, Valentin MAESTRACCI, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific supervision of Laurent REGNIER and Thomas SEILLER, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with both the french national charter for Research Integrity and AMU charter on the fight against plagiarism.

This work has not been submitted previously either in this country or in another country in the same or in a similar version to any other examination body.

Belgentier 23/08/2025

Liste de publications et participation aux conférences

Liste des publications et/ou brevets réalisées dans le cadre du projet de thèse:

1. Valentin Maestracci et Paolo Pistone. « The Lambda Calculus Is Quantifiable ».
   In : 33rd EACSL Annual Conference on Computer Science Logic (CSL 2025). Sous la dir. de Jörg Endrullis et Sylvain Schmitz. T. 326. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 34:1-34:23.
   url:https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2025.34
2. Marie Kerjean, Valentin Maestracci et Morgan Rogers. « Functorial Models of Differential Linear Logic ».
   In : 10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025). Sous la dir. de Maribel Fernández. T. 337. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 26:1-26:17. FSCD.2025.26
   url:https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2025.26

Participation aux conférences et écoles d'été au cours de la période de thèse:

Conférences et Workshop

1. Conference on Computer Science and Logic (CSL 2025)
2. International Workshop on Trends in Linear Logic and Applications (TLLA 2023)

3. International Workshop on Trends in Linear Logic and Applications (TLLA 2025)
4. Workshop on Games for Logic and Programming Languages (GALOP 2025)

Écoles et GT

1. Midlands Graduate School (MGS 2023)
2. Midlands Graduate School (MGS 2024)
3. Summer School on Foundations of Programming and Software Systems (FoPSS 2023)
4. Advances in Interactive and Quantitative Semantics (CIRM 2025)
5. Differential λ-Calculus and Differential Linear Logic, 20 Years Later (CIRM 2024)
6. Structures formelles pour le CALcul et les Preuves (GT SCALP 2023)
7. Structures formelles pour le CALcul et les Preuves (GT SCALP 2024)

Résumé et mots clés

Cette thèse porte sur l'étude de modèles de calculs utilisés pour faire de la réalisabilité en logique linéaire (Geometrie de l'Interaction):

- Un premier chapitre explique l'histoire du domaine, en présentant au passage plusieurs modèles de calculs existant dans la littérature.
- Un deuxième chapitre étudie en détail un modèle de calcul précis: les graphes d'interactions de Seiller. On en définit une généralisation comme une instance d'un cadre abstrait nouveau de modele de calcul, dans lequel on prouve la propriété clé d'associativité de l'execution. Ainsi on peut définir un modèle de calcul pour la réalisabilité linéaire pour n'importe quelle instance du cadre. On étudie ensuite une autre instance du cadre qui a la propriété de résoudre un problème qui existait dans une généralisation des graphes d'interaction: les graphes épais. On explique enfin pourquoi le modèle des flots est trop général pour être une instance du cadre.
- Un troisième chapitre porte sur un modèle "intermédiaire": le modèle des étoiles. Ce modèle, définit dans la thèse de Eng, généralise les flots en un sens combinatoire, mais reste proche du cadre. Y est donnée une nouvelle version de ce modèle, ou sont corrigées et rafinnées les définitions et propriétés essentielles, qui bien que pour certaines présente dans la thèse de Eng, posaient problème.
- Un dernier chapitre, très exploratoire, tente de mieux comprendre certaines propriétés combinatoires et géométriques des graphes d'interaction en s'inspirant de domaines des mathématiques (cobordismes, topologie dirigée, catégories). Y est aussi défendu l'idée que la notion de "location statique" est en faite affaire de recollement et de "système ouvert".

Mots clés: logique linéaire, réalisabilité, réalisabilité linéaire, géométrie de l'intéraction, syntaxe transcendantale, model de calcul, catégorie monoidale à trace, graphe d'interactions, curry-howard

Abstract and keywords

This thesis studies computational models used for realizability for linear logic (as part of Geometry of Interaction):

- A first chapter explains the history of the field, presenting several computational models existing in the literature as well as their connection to linear logic.
- A second chapter studies in detail a specific computational model: Seiller's interaction graphs. We define a generalization of this model, we show that it is an instance of a new, "low-level" framework to define abstract computational model, in which we prove the key property of associativity of execution. This makes it possible to automatically prove this property, and thus to define a computational model for any instance of the framework. We study another instance of the framework solving a problem that existed in a generalization of interaction graphs (thick graphs). Finally, we explain why the model of flows is too general to be an instance of the framework.
- A third chapter focuses on an "intermediate" model: the star model, defined in Eng's thesis, which generalizes flows in a combinatorial sense, but remains somehow similar to the framework defined previously. We study a new version of this model, correcting and refining essential definitions and properties, some of which, although present in Eng's thesis, were problematic.
- The final, highly exploratory chapter, attempts to better understand certain combinatorial and geometric properties of the interaction graph model, drawing inspiration from several areas of mathematics (cobordism, directed spaces, category theory...). We also defend the idea that "static locations" are similar to gluing and to the notion of "open system".

Keywords: linear logic, realizability, linear realizability, geometry of interaction, transcendental syntax, computational model, monoidal traced category, interaction graph,curry-howard

Remerciements

There are so many people I would like to thank.

Damiano Mazza told me at the beginning of my PhD that "I [(Damiano)] used to think that I knew everything starting my PhD, and when I was done I realized I knew nothing". He was very right and I know share this feeling.

I used to think that science was made by isolated geniuses that would revolutionize our way of thinking, even though as a teen I was already very aware of the quote by Newton "If I have seen further [than others], it is by standing on the shoulders of giants", that I attributed to modesty at the time.

I am of course but a pebble in the sea of researcher that humanity will produce, and in no way comparable to Newton, but the glimpse of what research is that I could get during this PhD made me realise how right he was. How right they all were.

I would like to very pretentiously go a bit farther than Newton. If I was able to do what I did, be it a little brick in the infinite castle of Mathematics, it is not only thanks to the giants that came before me, it is thanks to humanity as a whole. Every person I met helped shape me in some way, made me think differently, some helped me in times of trouble, other made anrgy or sad. For all these encounters, for all these events, for my life, I am very grateful and I hope I will always be.

I am, of course, only going to thank the people that helped me, and yet, not all of them, since I also dependend on the food produced by farmers, the people that built my house etc... To all these anonymous people I send my gratitude.

Now to the "real" thank you, mostly in French for obvious reasons. I doubt anybody will ever read this that is not part of this list, but nonetheless I hope it will show you how much the people around you shape your life.

- A ma maman, Sandrine Lauransan, merci infiniment, du fond du coeur, pour t'être toujours occupée de moi, et ce même dans mes pire moments. Tu m'as montré ce que c'est d'être aimé inconditionellement et je n'aurais pas pu réver meilleure maman

- A mon papa, Eric Maestracci, merci papa pour avoir su développer en moi la curiosité et la passion pour le monde qui nous entoure, et le monde de l'imagination. C'est grâce à toi que je suis qui je suis aujourd'hui et je n'aurais pas pu réver meilleur papa
- A ma soeur, Aurore Maestracci, merci d'être la meilleure petite soeur de l'univers. Merci d'être toujours drôle et touchante et intelligente et je suis tellement heureux que tu sois encore la. Merci d'avoir fait de mon enfance un enfance meilleure.
- A ma grand mère, Olivia Lusetti dite "Mamie Tara", merci mamie d'avoir su t'occuper de moi et d'avoir dévellopé ma gentillesse
- A mes grands parents, Jacques et Denise, merci à vous de nous avoir gardé quand on était petit, et de merci papy de m'avoir soutenu dans ma thèse, j'espère que tu es fier
- A mes grands parents, Tony et Dany, merci d'avoir su vous intéresser à mes passions et qui j'étais, d'avoir su me donner envie de repousser mes limites, et d'avoir pris soin de moi
- A mes cousins: Victor, Constance et Laurianne. Merci aussi à ma Tatie. Merci pour tout les moments que j'ai pu passer chez vous, pour une enfance pleine d'aventure, et pour avoir partagé avec moi tout ce que vous aimiez, m'avoir fait découvrir tant de choses
- A ma famille d'Angoulême: Tatie Sylvette et Tonton Claude, Corinne et Bruneau Laurençon (ainsi que leurs enfants): merci à vous tous de m'avoir aidé quand j'étais au plus bas et loin de tout le monde. Je vous en suis éternellement reconnaissant, ma "famille d'Angoulême"
- A mon beau père, Jean-Luc: Merci d'être toujours sympa et de partager avec moi toutes les histoires des films que tu vois au cinéma !
- A mon meilleur ami d'enfance, Léo Petit-Barande: merci au plus incroyable des Grizzlys, d'avoir été à mes cotés pendant toutes les guerres de la vie, ainsi que d'avoir fait de moi le meilleur New Super Mario Brosseur. Les Tintonous vaincrons
- A mon meilleur ami-bis d'enfance, Quentin Thibonnet: merci d'être encore mon ami après toute ces années, merci pour tout ces duels dans les jeux, sur Yu-gi-oh ou au Poker, merci d'avoir entretenu mon amour pour la stratégie, et pour toutes les discussions philosophiques sur le sens de la vie
- A mon troisième meilleur ami d'enfance, Dylan Ysmal: merci Dylan d'être peut être mon plus fidèle ami. Merci de prendre souvent de mes nouvelles et de te réjouir pour moi quand je réussis. Je suis honoré d'être ton ami, pour toujours. Merci pour tout ces moments nostalgiques en ligne, sur

Minecraft, Gary's mod et Portal 2 ! Et merci d'avoir propulsé mon intêret pour l'informatique.

- A Jennifer Delhommeau, et Mathis Thibonnet, qui sont encore un peu de la famille: merci Jennifer t'être occupée de nous à chaque vacances en commun de nos familles, et merci Mathis d'être devenu un adulte cool ! C'est toujours un plaisir de vous revoir !
- A Renée Picard, une troisième tatie: Merci de t'être occupée de ma soeur et moi et de toujours t'être inquiétée de ce qui nous arrivais, même des années après qu'on soit parti !
- A mes amis de collège: Ehouarn, Cristopher, Rémi, Emeric. Les amis, je ne vous remercierai jamais assez d'être encore dans ma vie. Je suis vraiment heureux de vous avoir vu devenir des gens vraiment super, toujours aussi drôle en sympa. Que quand je vous revois, je sais que le monde, si changeant, n'as pas trop bougé non plus et que vous êtes toujours les mêmes amis que j'aime à chaque fois
- A mes amis de lycée, Adrien et Adrien: je vous suis reconnaissant d'être encore là avec nous malgré que vous n'ayez pas la vie facile, merci pour tout ces souvenirs aux club info et de jeux vidéos, et les gros délires à chaque nouvel an !
- A mes amis de lycée de l'autre groupe: Joë Larue, Thibault, Salim, ainsi que Geoffrey: merci pour toutes les teufs, pour être toujours aussi haut en couleur chacun de vous, et d'avoir vraiment influencé qui je suis dans mes relations sociales. Vous m'avez grandement inspiré, merci à tous
- A mes amis de prépa: Guillaume "Délégué" Charret, Scarlett Gatt, Vincent, Thibault, Mélik Maksem, Vassili Maillet, Andrea Negro et techniquement Lucas Rioust (que j'arriverai à revoir un jour !): merci d'avoir été là dans les tranchées avec moi pour une des plus grosses épreuves de notre vie (au moment ou j'écris ces lignes), vous êtes tous des gens formidables et brillants, et j'espère que j'aurais encore l'occasion de faire des choses avec vous, même à la maison de retraite !
- A mes amis de l'ENS, Corto Mascle, Yoan "Karnaj" Geran, Elies Harrington, Samy Dray, Nino "le Bro", Colin Geniet, Roman Kniazev, Elies Harrington, Lotte, Jonath', Yoann Barszezak, Yuli Daune-Funato, et aussi ceux qui ont plus ou moins disparu de ma vie (coucou Victor, Martin Baillon, Noah Loutchmia, Darchtac) mais auquel je tiens toujours: je n'ai pas tant aimé mes années d'ENS, mais c'est certainement grâce à vous tous que je ne les hais point. Vous avez mes remerciements les plus sincères, restez cools !

- A mes amis post-lycée du groupe lycée: Anne-Sophie Bouvard, Jonathan et Marine Bouzard: merci d'être toujours aussi sympathique, restez des soleils ! Je suis content de vous avoir rencontré.
- A mes amis d'Angoulême, Arthur Leroy, Chloé Rich, Paul Gredat, Maelys De Haro, Eloise et Fabien, Maxime, Justin, Farima et Clément, et tout mes autres potos: merci d'être tous resté des rêveurs et de me permettre de rêver un peu encore avec vous aussi. Chaque fois que je vais vous voir je ressors la tête dans les étoiles!
- A mes amis de la recherche: Rémi Pallen, Davide Barbarossa, Boris Eng, Sidney Congard, Rémy Cerda, Aymeric Walch, Adrienne Lancelot, Vincent Moreau, Cedric Delacroix, Rémi Di Guardia, Aurore Alcolei: merci pour tout ce que vous avez fait pour moi, d'avoir été la dans la grande galère qu'est le monde de la recherche, d'avoir été encourageant en partageant vos expériences à vous, et aussi pour tout les moments sympa ou on a bien rigolé, parce que la recherche c'est aussi humain et pas que de la science
- A mes amis de GN: Grag', Galia, FC Dakeron, Foxi', Firin, et les autres: merci d'être là pour Marco qui galère un peu autant que dans la vraie vie parfois
- A mes deux meilleurs amis de Marseille, Axel Gastaldi et Elie Antoine: merci à vous deux pour avoir été juste des gens super. Honnêtement je croise pas souvent des gens comme vous, mais tout les deux (complètement indépendamment, je ne crois pas que vous vous connaissiez ah ah), vous avez été vraiment géniaux. Vous m'avez même remis au sport ! Donc vraiment merci
- A mes autres amis de Marseille, Maxime Sommela, Pierre Tchamitchian, Yahia Idriss Benalioua, Samantha Thiel, Loréna Quatreville, Clément et Benoit Baude, Alexander "Alex" Taskov, Patricia Roxo: merci pour tout les moments passés ensemble à parler de ce qu'on aime, jeux de sociétés, jeux vidéos ou séries ou même la vie, peu importe, c'était toujours sympa, alors merci, voilà. J'espère que je vous reverrai, ça va pas être facile mais je croise les doigts.
- A mes Dynamis de Marseille: François Hamonic, Emanuelle Salin, Julien Leclercq, Célia Breaud, Clémentine Baccati, Clara Grégoire, Harshit Pateria, Alexandre "un Blob": merci pour l'aventure qu'on a passé ensemble (et celles à venir), tout les trucs qu'on a essayé d'organiser tant bien que mal, même avec le manque de motivations des autres thésard. C'était hyper cool de tous vous connaitre et de faire du Ski ou des balades en Corse. Merci à tous ! Vraiment !!

- A tout mes autres amis qui ne rentrent pas dans ces catégories: Ella et Chloé Petit, Gwenael Hervé, Dionis Debat-Dupre, Tanguy Bruschi: je vous connais tous depuis longtemps maintenant et j'espère que ça durera encore. Je vous souhaite le meilleur pour chacune de vos vies !

- —

- A mes potes restant de Paris-Nord: Jad, Aloys, Simon, Baptiste, Chirine, Caterina, Jacopo, Stefano, Quentin: c'est toujours sympa de jouer aux fléchettes avec vous, et merci pour la super ambiance que vous mettez dans l'équipe, grâce à vous Paris-Nord c'est vraiment une FAC exceptionelle

- A ma co-bureau Juliette Coutens de toujours, merci pour avoir écouté mes rants de temps en temps, merci pour avoir été la pendant 3 ans, pour toutes les petites interactions rigolotes qui faisaient que le quotidien du bureau c'était pas la solitude

- A mes pote restant de Marseille: Adrien Mounier, Mathilde Guillaud, Marie Roth, Louis Usala, Tommy-Lee Klein, Nino, Yanis Coutouly, Paul Boisseau, Sergio, Clément, Olivier, Safae, Siméon, Yohann, Noé, Xenia, Willem, Toto, Kenza et encore pleins d'autres: merci pour tout les moments dans le séminaire doctorant, ou dans la salle café, les babys et les mots croisés que j'aime pas tant que ça, mais je venais quand même parce que c'est vous que j'aimais! Merci d'avoir été au moins la moitié de l'ensoleillement du TPR2!

- Merci Samy, que je met un peu à part, pour les discussions de parapente, et d'avoir toujours pris mes questions de physique très au sérieux. Tu étais quelqu'un d'exceptionnel.

- A mes potos restant de l'Irif, notamment Victor Arial, Mariana Milicich et Lucie Guillou: vous êtes trop fun et bonne ambiance (en plus d'être fort oh la la), j'espère que vous continuerez à l'être et qu'on se recroisera !

- —

- Aux femmes que j'ai aimées jusqu'à présent, de tout mon coeur et qui y resteront, Marine Viviano, Coralie Feba, Aurore Marchelli, Cécile Ferrand. (Liste non exhaustive, mais certains noms sont secrets !). Merci à vous toutes, j'ai beaucoup appris de chacune de vous, chacune à votre manière vous m'avez apporté beaucoup, pas toujours dans les circonstances les plus agréables mais la vie est ainsi. Merci d'avoir fait de moi un adulte, en me confrontant aux vrais problèmes de la vraie vie. Tout est bien moins facile quand on est impliqué soit même émotionellement, moi qui me croyais mature. Encore merci.

- Et justement, à Cécile, qui m'a tant apporté. Tu sous estime très probablement ton importance, tant dans ma vie que dans la vie en générale. Merci

d'exister, et de montrer au monde que l'amour existe vraiment, qu'il y a encore des gens qui aiment avec leur coeur et pas juste leur tête, et qui sont vraiment profondément bon et apaisant. Merci d'être toi même, merci d'avoir ranimé la flamme en moi. Tu me manqueras éternellement, je te souhaite un bonheur infini là ou tu ira dans la vie. Une vie pleine de nature et de petits cochons mignons. Merci

Aussi, à tout ceux que j'ai oubliés, ma mémoire est faillible, je me connais, et ne pensez pas que vous ne comptez pas pour moi pour autant.

I will now give a more science-related thank you

- To Laurent Regnier, I thank you for accepting me as a student. It was a great honor, you are, still today, a great researcher. Thank you for listening to all my dumb questions and answering them multiple times since I kept asking the same ones. You made me understand many things and I am really grateful.
- To Thomas Seiller, I thank you for accepting to teach me your style of Linear Realizability, so that I could understand Transcendantal Syntax. I am really proud to be your student as well, you are a very innovative and motivated researcher, and you were a great mentor for me. I thank you again and forever
- To my big brother Boris Eng, thank you for being always there for me when I was completely lost facing some of the writings of JY, thank you for making me not lose hope in some complicated moments, thank you for being nice to me, always.
- To my senpai Tito, thank you for helping me stay grounded in the real world, and for being a very very empathetic "big bro". For always giving me advice for my research and caring about my mental health. Really, thank you Tito.
- To my big cousin Rémy, thank you for being the cool young researcher and "neighbour next door" (next office) that I so admire! Thank you for welcoming me in the team at my beginnings in Marseille, it was not the same once you left. I wish you the best
- To my other big cousin, Lison Blondeau-Patissier, you are a very impressive researcher, never forget that. It is impressive how you can do things very thouroughly. You were also a really nice person, I hope I will be able to see you again !
- To the rest of my research family: Elies Harrington, Adrien Ragot, Will Troiani, Chirine Laghjichi, Caterina Mosca; I want to thank you all so much for all the cool moments in conferences, and in the Monday Morning reunions. All of you I have fond memories of and I really wish I will be able to see you or even work with you in the future! You are like (research) family to me!

- To my team, Lionel Vaux, Dimitri Ara, Pierre Clairambault, Raphaelle Crubillé, Alexei Muranov, Tommy-Lee Klein, Victor Blanchi: thank you for being such a great team ! One of my biggest regret in Marseille is that I could not spend more time with you (since I was often with the PhD students). The internal working group was so very interesting thanks to all of you, and you were all very nice. Thank you!
- To my "Comité de suivi de thèse", Lionel and Claudia, I am forever grateful to you. I was very very down at some point, and you both picked me up and put me back on my feet, and I will never be able to repay you for that, for your kindness and the motivation you gave me. Thank you.
- To my reviewers, Paul-André and Shin-Ya, I deeply admire both your works and it was such an honor to get my PhD reviewed by both of you, I cannot thank you enough, reviewing can be quite some work, I thank you for accepting this task, especially on such short notice. I really thank you both, I hope I will get the luck of meeting you again in conferences.
- To the rest of my Jury, Aurore, Claudia, Masahito, and Samuel, thank you for being part of my jury and getting interested a bit in my work. I was really glad that you all accepted, I was really lucky to have such a great Jury. I want to thank you again for this, and I hope I will be able to collaborate with some of you in the future, I would be honored.
- To JY Girard, thank you for taking logic really seriously. Your work, even if I did not understand all of it, made me want to do pick up logic again. To understand what is logic, really. You dedicated your life to this, and you managed to convince me that there really was something deeper hidden behind what I knew. I decided to do a PhD thanks to you, I thank you for that.
- To my Grad advisors, Paolo and Marie: A really deep and personal thank you both, for making me like research again, I was very afraid getting back into it that I would fail, or not like it, but I did and it is in great parts thanks to your entusiasthm for what we did. You two are both amazing researchers and I hope I will be able to live up to your greatness!
- To my undergrad Advisors, Bruno Barras and Valentin Blot, Martin Raussen and Lisbeth Fajstrup, Andrea Vezzozi and Thierry Coquand: thank you for your kind supervision, it was never easy at all but we manage to make some things together and I learned a lot, you have my deep thanks.
- To my ENS teachers, particularly Gilles Dowek, Jean Goubault, Sylvain Schmitz, David Baelde, Paul-André Mellies and Hubert Comon: thank you for taking care of student me, during some difficult years, for teaching me the subjetcs I dreamed of understanding, in a really clear fashion. Thank you for

making it even more interesting than I thought it was by transmiting your passion for the subject.

- To Yoan Geran, who is my best "research friend", I thank you again for saving my life during the writing of all of this, with your absolute wizardry when it comes to Latex. And for staying my friend after all these years! Thank you Karnaj!
- To Kurt Gödel, thank you for being obsessive. I forgot when I stopped being like you, but it is something I regret.
- To Ludwig Wittgenstein, thank you for being a rockstar. I never was like you, but maybe one day I will manage to. And thank you for solving philosophy
- To Dr. Olivier Muzereau, my highschool math teacher, for teaching me about Gödel's incompleteness, making me read Logicomix and making me want to solve the mysteries of the Universe. I failed but I got a much deeper understanding of it all nonetheless, and that is priceless.

To all the other people, teachers, scientists who shaped me scientfically (there are way too many to count, the list is not even recursively enumerable even though finite, because of missing information ah ah), a great thank you as well.

To all other people who made my life possible, from the construction workers to the farmers, the teachers, doctors, etc... Thank you for being the backbone of this world!

Et pour finir, à tout mes amis et amours à venir ! Hâte de vous rencontrer ! :D

A small step towards an Open Source Logic

Valentin Maestracci

February 4, 2026

Contents

1. Introduction

1.1. What is this PhD about

This PhD is a summation of my work during the last 3 years. It is a study of example of models of computation, which will (hopefully) later be used to generate models of logic.

Scientifically, it fits within the continued legacy of the research program of Geometry of Interaction (GoI), now "rebranded" under the name Transcendantal Syntax (TS) by Girard, and within Seiller's program of "Mathematical Informatics" which aims at defining an abstract notion of model of computation. It was done under the supervision of Thomas Seiller and Laurent Regnier.

As previously stated, this research is part of a program known as the GoI (or TS) program. But it is important to clarify how exactly, for there is a bit of a split in how people understand the name "GoI " : most researcher todays associate GoI with Danos and Regnier's Lambda Algebra and the research that was born out of it, with the Interaction Abstract Machine etc...
This area has developed separately from the original program of GoI. In contrast, my work is more in line with the original line of the GoI program.

I would very much like to start this manuscript by explaining the motivations which led me to study these things, for it is quite a niche subject and I've been often asked about why I was interested in all that. I would also like to justify the name I gave to this manuscript, for I feel it is very unclear what this thesis is going to be about from just the title.

I can unfortunately not claim that I understand the ambitions of Girard, but reading his papers I felt like there was an original idea that I had not heard about before and so I wanted to explore just that.

This work is a study done at the core of the Proof-Program correspondence (Curry-Howard), there is thus not one but two motivations at work!

1.2. Motivation

1.2.1. The point of view of Programming

This paragraph is my attempt to convey what I understand of the motivation of my "research predecessor", Boris Eng. Any inaccuracies are my own.

In traditional programming languages, *type inference is the duty of the compiler.* We will discuss here a possible way of designing a programming language where *the programmer has somehow control over type inference.*

Say somebody is programming in C, then one is given primitive types (int, float, double etc...), primitive constructors ($\to$, struct etc...) and from this one can define new types or data structures: $\mathbf{int} \to \mathbf{int}$, etc...

Remark
It is unclear to me if there is a difference between abstract data structures and types: maybe types are formal specifications for data structures, but it seems data structures define types (typically, a simple struct in C defines a type).

The setting we will consider allows to define new primitive types (similary to programming languages such as Rocq which can define new primitive types using inductive types) but also new type constructors.

People familiar with Linear Logic are probably familiar with proof nets and their correctness criterion. In the world of programs, not every program is well-typed: some proof structures are not proof nets. But there is a way to check whether a program is well typed or not: the correctness criterion.

Say we have an abstract notion of program, such as defined in [54], we can do a form of "realizability construction" such as:

Definition 2 (Programming Language)
We are given a set of program $\mathbb{P}$, a notion of (associative) execution (as in the execution formula of linear logic) to plug programs p and q against each other $:: (p, q) \in \mathbb{P}$, and a set $\bot$ of program, corresponding to "valid" results of executions (observations that we can make on our programs, traditionally called an orthogonality).
We say that $p \bot q$ when $p :: q \in \bot$. We can define the orthogonal of a set $P^\bot := \{q \mid q \bot p, p \in P\}$. And we can lift $::$ to such sets: $p :: Q := \{p :: q \mid q \in Q\}^{\bot\bot}$

In such a setting, one can introduce two notions of typing: the syntactic typing $:$ and semantic typing $::$ (which is just execution). The main theorem of realizability ensures that $p : A \implies p :: A^\bot \subseteq \bot$, that is, if one can statically type p as having type A then p *behaves like an* A, also, p passes all tests of $A^\bot$.

This allows us to define behaviours of programs,

Definition 3 (Behaviour)
A behaviour is a set $B \subseteq \mathbb{P}$ such that $B^{\perp\perp} = B$.

Concretely, if one wants to tell whether $p : A \otimes B$, one just has to check that p respects $(A \otimes B)^{\perp}$ (for example, interact well with the Danos-Regnier switchings, has just one long trip etc...) and then one will get the guarantee that such a p "behaves" like an element of $A \otimes B$. (There are also sometimes results of completeness that guarantees that such an element is what you would expect, that is, a pair of independent element).

The only requirement to do something like that is that the tests inside of $(A \otimes B)^{\perp}$ that one is required to pass, should be finite (or else there would *never* be termination of such checking)

This can be done statically, by checking $p : A \otimes B$ recursively that p was constructed using "justified rules" (see the next section), which is the usual notion of typing, for example if it was combined from an $A \to B$ and A then it has type B.

This can also be done "interactively", by checking that $p :: (A \otimes B)^{\perp} \subseteq \perp$, making p pass all Danos-Regnier (or long trip, etc...) tests (note that, here, there can be termination problems, for example `loop` $:: \mathbb{N}$ will loop indefinitely at during typechecking). Nonetheless there can be guarantees for termination, in particular, these can often still be checked statically by looking at the topology of the programs. The trick is that it is often not necessary to do an infinite amount of tests, a function, such as $lambdax.x + 2$ has a form of "uniformity", which makes it possible to check in finite time that it is $\mathbb{N} -> \mathbb{N}$, without testing it for every input.

This blurs the separation between typing and computation, allowing to consider new forms of typing, for example, unit testing can now be seen as a form of typing: let T be a set of unit test, then $p :: T (\subseteq \perp)$ iff p passes all test in T.

In such a setting, type constructors are not primitive but defined through their test! The user can thus *add new type constructors*, be it well known ones, like additives from LL, or more excentric ones like the generalized LL connectives. There is no bias towards a certain type or another, all "logical types" are types, provided the user can define the correct tests for said type.

A simple example would be on natural numbers, $p : \mathbb{N}$ and $p :: \mathbb{N}$. With the peano definition of natural numbers, one can define a natural number directly, like 3 as the term $S(S(S(0)))$ and check that it has type $\mathbb{N}$ by doing $3 :: \mathbb{N}$, which will deconstruct 3 dynamically and see that it ends up at 0.
One could also check that $3 : \mathbb{N}$ statically, by constructing 3 inside the code as using the axiom $0 : \mathbb{N}$ and applying $S : \mathbb{N} \to \mathbb{N}$ to 0 three times, giving the program $3 := S :: S :: S :: 0$, and then asking $3 : \mathbb{N}$.

My "predecessor", Boris Eng, created the programming language Stellogen with these ideas in mind, but there are a lot of improvement one could do: he only

considered semantic typing, which he did purely computationally (even though the passing of tests is usually checking that there are no cycles in a graph, which can be checked statically instead of computing the paths in the graphs). In terms of typing, only MLL is clearly done at the moment. There are a lot of things one could add, exponentials, 2nd order (these 2 requires understanding correctness criterion well), higher order (through universes) and dependent types.

Another challenge would be to study predicate logic, with possible direction sketched in an article of Girard [29], which would allow to do "real world" mathematical proofs.

1.2.2. The point of view of Logic

Girard's original goal is, from what I can tell, more of a logical one, that is to "recreate logic" from scratch, without "bias", answering in the process questions such as what is it that makes something logical (like "and"), while "broccoli" is not.

For that purpose he tried to design a model of computation with the "least assumptions" he could (the more general, let us say), and completely alogical, aiming to capture a form of "pure computation". I assume, in the hope of defining the "ultimate model", which can has anything logical as behavior.

In traditional logic, there is a tension between axioms and rules: in a Hilbert System, there is one rule, modus ponens (notice how it is actually the cut rule), and just axioms. In Natural Deduction or Sequent Calculus, the axioms are transformed to become deduction rules, and there are no more axioms.

Girard then advocate returning to a Hilbert System to find a form of "universal logic":

Let's say we have a program $p \in (\Gamma \to A) \otimes (\Delta \to B) \to (\Gamma \otimes \Delta) \to (A \otimes B)$.

Another program $q \in (\Gamma \to A \parr B) \to \Gamma \to (A \parr B)$.

Then we have "justified" computationally the rules of MLL, that is the introduction of $\otimes$ and of $\parr$. We can then translate all sequent calculus proofs of MLL inside our Hilbert System, by translating introduction rules of $\otimes$ by p and of $\parr$ by q.

We just need to prove some meta-theoretical properties, that there is indeed strong normalisation when opposing a program behaving like a $\otimes$ to one behaving like a $\parr$. This way, termination of cut-elimination and consistency of the system are guaranteed by construction! And there is no need to derive in a case by case basis a procedure of cut-elimination: way the program interact together is already given, it is what we started with.

But this checking is done rule by rule, completely independently and not for an entire system! (Such a checking is done by Girard in Set Theory, but we could interface the language with a proof assistant such as Rocq). This means, a logic based on this principle could add new operators "for free" without the need of reproving cut-elimination: it would be truly open. Different users could contribute to adding new logical rules indepently, like a sort of open source project, and one could import sets of rules like one imports a module in a programming language.

This was what I saw as my lifelong aspiration as a researcher when I began this PhD, and the name I gave to this manuscript, "A small step toward an Open-Source Logic" is an hommage to that.

One might wonder what "extra operators" one could add to logic, and why one would like to add these, and the answer, to me, is that these operators describe the behaviours of perfectly valid programs, so, if one is interested in the programs they might provide good tools to study these.
One can do much more than just MLL to get even richer descriptions. One could add a lot of extra "fancy" operators, weak exponentials etc... Provided they are justified. Because Hilbert System do not have any assumptions about contexts etc... we can add anything as an axiom! And cut-elimination is guaranteed because of the justifiers that give computational content and the meta-theoretical proof above.

1.2.3. My hope for a research program

I do not actually think it is possible to create an "ultimate model of computation " (not even Stellar Resolution, the one from TS), and thus an ultimate logic. This is just a hunch I have and not a claim I can really back up, because intensionnality in programming, that is, comparing the behaviour that programs can have, is something that we understand very little, as it is very complex, and computer science is a young field of research.

I think every model of computation generates its own behaviours that are interesting in their own right, and I do not want to discriminate. Generalized MLL connectives should be as valid as connectives as the usual MLL $\otimes$ and $\parr$.

From an abstract notion of what a model of computation would be (for example, [58], which is an attempt at finding a satisfying definition, to try, among other things, to capture complexity classes through realisability), taking $\bot$ as the terminating programs, one can give rise to a logic with a guaranteed cut elimination.

Here is how I would envision an "open" proof system:

- Proofs would be done in a Hilbert System

- There would be Logical Modules, corresponding to axioms of the Hilbert System, that one could import.
- There would be Computational Modules corresponding to the definition of a model of computation, importing some logical modules, and pairs of programs justifying the axioms inside the imported modules (if we want to be sure, this would come with a proof in set theory of that claim)

If it so happens that you cannot prove or express what you wanted to because your model of computation is too weak, then one would just change of computational module, but the proof would be left unchanged thanks to the intermediate language of the Hilbert System which serves as an interface between logic and computation. There would be two possible way of keeping your proofs justified:

- Either re-justify all the axioms by hand
- Compile the your original model of computation into the new one. There would be a need to study the properties of program transformations (defunctionalization? continuation passing style? etc...) preserves justification. This is a really hard problem and I hope could become a field of study in its own, at the intersection of logic, compilation and realizability

I am unsure that this is a viable research program, nor that I will be able to conduct it but it sort of was my dream when I came and started my PhD.

I want to finish this introduction, by stating why I even got interested in doing an "open source logic" in the first place:
A long time ago, there was one proof of basic facts, such as the fact that if a number is divisble by 4, it is divisible by 2. But nowadays, there are many, not differing in the actual content, but in the language they are expressed in. Coq, Agda, Lean, Mizar, NuPrl and many others, there are many theorem provers in the world we live in, and an elementary proof of the previously mentionned fact for each of them. I was a young student at ENS when I was told this story by one of my favourite teacher there, Gilles Dowek. This lead him to create his project "Dedukti" to serve as a hub between proof assistants, where proofs would be stored so that they are not lost when a proof assistant goes unused, and can be used interoperably in different assitants. To do this, Gilles and others created LambdaPi-modulo theory, a sort of "universal" dependent type theory with rewriting rules, that could be used to encode anything. I always kept that story to mind, and even though I thought Dedukti's way of doing a "universal logic" was a bit unsatisfying, I always kept his ideas close to my heart. In hindsight, I think I was somewhat wrong about Dedukti, it is closer than I thought to the dream I sketched above.
I was very saddened to hear about his death just as I was writing these lines, and so I wanted to honour him and his ideas in this little paragraph. I think with his ideals in mind, one can see what led me to this shakey idea for a research project. I thank you, Gilles.

1.3. How to read this manuscript

I would like to disclaim that this manuscript was partly made by combining papers in the work that I have with co-authors. Thus, some definitions (notably in the section of unification, and a bit in the first section and Transcendantal Syntax), were written by my (future) coauthor Boris Eng. I did edit and adapt them when needed, but when they were recapitulating already existing definitions or proofs that can be found in the litterature for background prerequisites, I did not as they were already good as is. I also reused many illustrations from the paper that were good for example. Nonetheless, the "technical content" of the manuscript is new, but I wanted to give credit where credit is due.

Hole
I would also like to note the presence of an environment that I made up called "Hole". This will be used throughout the manuscript to hint at gaps (hence the name) either in my knowledge, or that could be filled in future research, and that I hope will be useful for future me to remember what directions my research could take.

Finally, a bit of discussion on notations.

Notation
I sometimes use traditional composition of function "$\circ$", and diagrammatic composition ";", which means $f;g = g \circ f$, depending on what is clearer in context. Note I have a natural preference for ";".

2. From LL to modern GoI

Summary
In this chapter, we will attempt a sort of a pedagogical and historical account of how the model of computation that this thesis studies came to be, to serve as a form of justification of why they would be interesting: they arise naturally from the study of logic.

2.1. A quick historical recap

Linear Logic is a refinement of intuitionnistic logic, introduced by Jean-Yves Girard in 1987. Unlike intuitionnistic logic, where assumptions can be reused freely, linear logic treats assumptions as sort of "resources" that must be used in a controlled manner. It introduces new logical connectives like tensor: $\otimes$, the mulitplicative "and" and with: & the additive "and", which makes it a richer proof system than intuitionnistic logic, being able to express subtler formulas. Girard discovered LL by doing semantics: that is, trying to give meaning to programs (which, by the Curry-Howard correspondence are equivalent to logical proofs) by interpreting these in a mathematical model [26]. Out of some models, he was able to see that in some cases the usual logical operators could be decomposed into "finer" ones, and so he imported that finding back into syntax.

But he did not want to stop at semantics: the core of Logic for Girard, was about computation and normalization of proofs (see the Curry-Howard correspondance). He thus wanted to do more than the usual "static" semantics, where the proofs were interpreted as static mathematical objects, such as a function. He then tried to define a form of dynamic semantic that would do justice to the complex computational content nested inside of proofs, and started his GoI research program. (Note it is still static in a sense, it captures the observational dynamics: the interaction with the environment, but not the internal workings of β-reduction) Over the year, he kept working in this direction, fliping over the point of view in trying to understand logic from computation and rebranding the program as Ludics, and then finally Transcendantal Syntax.

To get a better understanding of the dynamics of proof (one could say, give a stronger form of semantics to proof), Girard originally interpreted proofs as operators in a space, the dynamics of which consisted of acting on said space.

This led Seiller, years later, to define a really abstract notion of model of computation (somehow based on dynamical systems), "really abstract" in the sense that it encompasses "concrete" model of computation like turing machines, but also "impossible" or "abstract" models such as turing machines with oracles, operators in a space etc...

In this thesis, we will study some particular models of computation originating from these programs.

There are two reasons I can see for doing that:

- These models can be used to create models of LL, and one might be able to extract ideas from such models, such as was done multiple times throughout history (this is how LL was born after all)
- The abstract notion of model of computation of Seiller come from his interaction graph model. It is interesting to see if it can capture models that are supposed to be "more general". If it cannot, then there would be a need to define an even more general notion of model of computation.

One might wonder what we mean exactly by a model of computation? I cannot give an exact definition, as it is not something that is agreed upon in the litterature, and part of Seiller's work is to find a satisfying answer to this question. Nonetheless, I will give an informal definition:

Convention
We will call a "model of computation " anything that computes, without restriction on the properties of the system such as computability, typing or others...
Since this is hardly enlightening, here are many examples:

- Finite state automatas, Pushdown automatas, Turing Machines...
- Lambda-Calculus, Infinitary Lambda Calculus (all forms)...
- Prolog, Wang Tiles, ...
- Sequent-Calculus for LJ with cut-elimination, Proof Structures in LL...

Some model of computation might not be implementable in an actual machine in the real world, but they can still be of theoretical interest so they should still be considered as models of computation.

2.2. An extremely concise introduction to formal logic

Proof Theory is the field of mathematics whose object of study is *formal mahematical proofs*, just like, Arithmetic say, is the field of stuy of numbers.

For that, we need a mathematical definition of what a proof actually is. The working mathematician knows that proofs are done in the following way: take assumptions, combine them together and get the desired result out of them. From that it is clear that a "proof system" should have two things:

- A set of assumptions, called axioms, that serves as elementary bricks.
- A set of operations, called rules, used to combine and deduce things from these axioms.

This leads us to the 3 biggest families of deduction system:

- Inference System: Hilbert System (that I will call Logical Inference to make a nice trinity name with the other two, and also I think it is bad practice to name things after people): These have a minimal amount of rules (just one, the modus ponens, which is the most primitive deduction rule). The rest is just pure axiomatic, that is, even the way we can combine assumptions are encoded as axioms. These are annoying to use in practice, but they are actually very interesting for a simple reason: they are *minimal*. Nothing superfluous is assumed inside the system since there is but one rule.
- Natural Deduction: These have a plethora of rules to combine statements, and no axioms by default. Their particularity is that they have two "dual" types of rules, introduction rules and elimination rules. The names are fairly explicit: an introduction rule describe how to introduce an operator (from A and B, I can infer $A \,\&\, B$) and an elimination rule how you can infer something from the operator (from $A \,\&\, B$ I can infer A). It was noted that proofs here could be simplified by eliminating *detours*, that is rewrite proofs not using lemmas. The most well known rule is eliminating an introduction of $\to$ with an elimination of $\to$, which corresponds computationally to a beta reduction in a lambda term through the curry-howard correspondence.
- Sequent Calculus: These are a way to make natural deduction have a left/right symmetry. They only have introduction rules. To do this, they usually introduce an explicit context/environment. An elimination rule just become an introduction in the environment. The two dual worlds now interact through a rule called the cut-rule (also the axiom rule...), which gives the computational content to the system. This all is a consequence of a duality between environment/program, self/other, observer/observed.

Remark (What about an opposite sequent calculus ?)
Sequent calculus has only introduction rules, but it is possible to create a symmetrisation of Natural Deduction using only elimination rules instead. This was done by Parigot, when he created Free Deduction [22]. This then led to the creation of the $\lambda\mu$-calculus, and after a lot of work to a model of computation called System L, which really captures the symmetries between program and environment.

There are a lot of variants of such systems, depending on the choice of rules. There are also less known proof systems, such as deep inference systems, where one can make inference deep in the statements (for example, use the B directly in $(AB)C$), or combinatorial proofs [35]

I want to clarify something that I found annoying when learning logic, there are actually two different systems that are called "Natural Deduction ".

The one I presented under that name, and a form of Natural Deduction but using sequents (closed context) instead of having "floating assumptions" (an example of such Natural Deduction would be Martin-Löf Type Theory). I propose the name Sequent Deduction for such systems.

Table 2.1.: Classification of proof-systems

	Deduction	**Calculus**
Open-Context Style	Natural Deduction	Proof Nets
Closed-Context Style	Sequent Deduction	Sequent Calculus

Note this does not quite work for Logical Inference, as there is not really a notion of context (everything is global).

Note (Convention Summary)
To summarize my convention:

- A *deduction* is something using introduction/elimination rules.
- A *calculus* is something using cuts.
- The word *natural* is used for *open context*.
- The world *sequent* is used for *closed context*.

To have a form of *open source* logic, the best choice would probably be to use Logical Inference, because it makes no assumption on anything.

Hole
All modern proof assistants are somehow based on Higher-Order Logic or Martin-Löf Type Theory etc... Which are natural deductions, so have a lambda-calculus as computational system. Is it but a strange coincidence?
Would not it be possible to make dependent types with universe in a different

model of computation? For example, Interaction Nets. With dependent types, one could encode enough logic to make a "different kind" of proof assistant.

We will now quickly study a sequent calculus for a fragment of Linear Logic, to understand what Linear Logic is.

2.3. From Sequents to Proof-Nets

2.3.1. Sequent Calculus for LL

We first define the grammar of formulas of Multiplicative Linear Logic (MLL):

$$A, B = X_i \mid X_i^{\perp} \mid A \otimes B \mid A \parr B \qquad i \in \mathbb{N} \tag{$\mathcal{F}_{\text{MLL}}$}$$

Linear Logic is then usually presented via a sequent calculus with the following rules:

$$\frac{}{A \vdash A^{\perp}}\ \text{ax} \qquad \frac{\Gamma_1 \vdash \Gamma_2, A \quad \Delta_1, A \vdash \Delta_2}{\Gamma_1, \Delta_1 \vdash \Gamma_2, \Delta_2}\ \text{cut}$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \otimes B, \Delta, \Delta'}\ \otimes_R \qquad \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}\ \otimes_L$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'}\ \parr_R \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \parr B}\ \parr_R$$

Now, because of the symmetries of linear logic, one can choose to express all rules on one side, and get an equivalent system:

$$\frac{}{\vdash A, A^{\perp}}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}\ \text{cut}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$$

This is the basis of, and the system that people refer to when using Multiplicative Linear Logic. For people unfamiliar with Linear Logic, it is just a form of refinement of Intuitionistic Logic where the rules that allow to duplicate or add extra hypothesis can be used only under scrutiny, and are thus controlled.

As many logical systems are, this sequent calculus is actually a model of computation. Every proof can be rewritten, using specific computational rules, into a proof not using the rule (cut). This "fact" is actually a theorem, which is at the heart of the proof/program correspondence. It is known under the name of *cut-elimination theorem.*

There exists a remarkable extension of MLL with a rule called MIX (*cf.* figure 2.1), initially studied by Fleury and Rétoré [21]. This rule corresponds to the axiom scheme $A \otimes B \multimap A \parr B$ and constitutes, together with the other rules of MLL, a new proof system called MLL+MIX. Beside this new rule, MLL+MIX works with the same formulas as MLL. In particular, all MLL sequent calculus proofs are MLL+MIX sequent calculus proofs as well. We illustrate here the rule (mix) and a proof which uses it:

$$\dfrac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta}\ \text{mix} \qquad\qquad \dfrac{\dfrac{\dfrac{}{\vdash X_1^\perp, X_1}\ \text{ax} \qquad \dfrac{}{\vdash X_2^\perp, X_2}\ \text{ax}}{\vdash X_1^\perp, X_2^\perp, X_1, X_2}\ \text{mix}}{\vdash X_1^\perp \parr X_2^\perp, X_1, X_2}\ \parr}{\vdash X_1^\perp \parr X_2^\perp, X_1 \parr X_2}\ \parr$$

Figure 2.1: The MIX rule of MLL+MIX sequent calculus and an example of a sequent calculus proof of $\vdash X_1^\perp \parr X_2^\perp, X_1 \parr X_2$ which is not provable in MLL.

The MIX rule corresponds topologically to allowing disjoint union of proof structures as being “correct". Although not “logical" (*i.e.* not coming from MLL sequent calculus), MLL+MIX proofs keep interesting computational properties and naturally appear in various models of linear logic.

2.3.2. Equating proofs: ProofNets, a graphical syntax

Proofs in Sequent Calculus suffer from a problem, they are really verbose. Indeed, one would like to identify these two derivation:

$$\dfrac{\dfrac{\vdash \Gamma, A, B, C, D}{\vdash \Gamma, A \parr B, C, D}\ \parr}{\vdash \Gamma, A \parr B, C \parr D}\ \parr \qquad\qquad \dfrac{\dfrac{\vdash \Gamma, A, B, C, D}{\vdash \Gamma, A, B, C \parr D}\ \parr}{\vdash \Gamma, A \parr B, C \parr D}\ \parr$$

Which only differ by an "administrative" order, but affect independant parts of the proof.

For that purpose, a graphical syntax for proofs, called PN was introduced.

It has the particularity that not everything that looks like a PN is actually a proof, some objects are just computational, with no logical meaning.

We thus start by defining the notion of *proof structure*, which is the first "pure" (as in, not a logical) model of computation that we will see in this thesis. You will see that not every program in this model will correspond to a proof, that is what we mean by "alogical".

First, we give a general definition of directed hypergraph:

Definition 11 (Directed hypergraph)
A *directed hypergraph* is a tuple

$$H = (V, E, \mathtt{in}, \mathtt{out})$$

where V is the set of vertices, E the set of hyperedges, and the two functions $\mathtt{in} : E \to \mathcal{P}(V)$ and $\mathtt{out} : E \to \mathcal{P}(V)$ associate respectively to every e a set of "inputs" and "outputs".
Remark that we allow hyperedges with no input or no output (for a hyperedge e, we may have $\mathtt{in}(e) = \emptyset$ or $\mathtt{out}(e) = \emptyset$).
We will sometimes write $v \in e$ to say that $v \in \mathtt{in}(e)$ or $v \in \mathtt{out}(e)$ indiscriminately.

The notion of *ordered* directed hypergraphs will also prove to be convenient:

Definition 12 (Ordered directed hypergraph)
An *ordered directed hypergraph* $H = (V, E, \mathtt{in}, \mathtt{out}, \leqslant)$ is a directed hypergraph $(V, E, \mathtt{in}, \mathtt{out})$ extended with a partial order relation $\leqslant$ over $V \times V$.

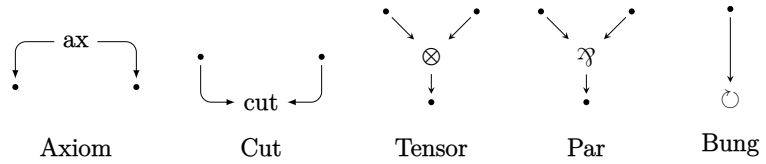
We will now introduce the notion of Proof Structure. We actually define as lightly more general notion, using a definition that is not exactly standard: there is an extra type of possible hyperedge compared to the traditional setting. This type of edge is called a "bung" and will be used to plug the "holes" of the proof structure, which will be a structure without bungs.

Definition 13 (Partially Bunged Proof-structure)
A partially bunged *proof structure* is defined by the data of a labelled ordered directed hypergraph $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$, with

$$\ell_E \colon E \to \{\otimes, \parr, \mathrm{ax}, \mathrm{cut}, \bigcirc\}$$

a labelling map on hyperedges.
A proof structure is subject to these additional constraints:

- Hyperedges satisfy the arities and labelling constraints shown below, where $\mathrm{ax}, \mathrm{cut}, \otimes, \parr, \bigcirc$ represent the hyperedges, incoming arrows denote that a vertex is an input and outcoming an output.
- Each vertex must be the output of exactly one hyperedge, and the input of at most one hyperedge; (we will call this property the wire-network property)
- Cut hyperedges must connect either:
  - The output of a $\parr$ hyperedge with the output of a $\otimes$ hyperedge, or
  - Two output of axioms.



Finally, the order has to be defined on vertices that are not inputs of a cut hyperedge, that is defined on $X \times X$ with $X := \{v \in V \mid v \notin \mathtt{in}(e), \ell_E(e) = \mathrm{cut}\}$.

Remark
The requirement on premises of cuts show that these are not completely pure/alogical, one still needs to oppose a $\otimes$ to a $\parr$

Convention (Left and right sources)
Thanks to the fact that we considered an ordered directed hypergaph, and for practical purposes, we can consider “left" and “right" sources of an hyperedge since there are never more than two.
For a proof structure $\mathcal{S}$ and a hyperedge e of $\mathcal{S}$:

- if $\ell_E(e) = \mathrm{ax}$ and $\mathtt{out}(e) = \{u \leqslant v\}$, we define $\overleftarrow{e} := u$ and $\overrightarrow{e} := v$ for the left and right conclusion of e;
- if $\ell_E(e) \in \{\otimes, \parr\}$ and $\mathtt{in}(e) = \{u \leqslant v\}$, we define $\overleftarrow{e} := u$ and $\overrightarrow{e} := v$ for the left and right premise of e.

In particular, thanks once again to the order, we will adopt the notation $\mathtt{in}(e) = (v_1, v_2)$ which should be understood as $\mathtt{in}(e) = \{v_1, v_2\}$ and $v_1 \leqslant v_2$, and similarly for $\mathtt{out}(e)$. The reason we do not order premisses of cuts is because we would like to be able to plug two proof structures together and consider the result the same structure, whether one was one the left and the other on the right when the were plugged.

As explained before, a *partially bunged proof structure* is a concept that will not really be used but encompass two cases that will be encountered:

Definition 16 (Proof Structure)
A Proof Structure (abreviated as PS) is a partially bunged proof structure with no bungs ($\circlearrowleft$).

Definition 17 (Bunged Proof Structure)
A Bunged PS is a partially bunged proof structure where every vertex is the input of exactly one hyperedge.

Observation (Wire-Network for bunged proof structures)
When a proof structure is bunged, the wire-network property implies that for every vertex v, there exists exactly two edges $\mathtt{e}^-(v) \neq \mathtt{e}^+(v) \in E$ such that $v \in \mathtt{in}(\mathtt{e}^-(v)), v \in \mathtt{out}(\mathtt{e}^+(v))$.
This property (and variants, which will be given the same name), will be used throughout the manuscript.
I nicknamed it this way because it implies that the graph is a sort of "network of wires" plugged on vertices.

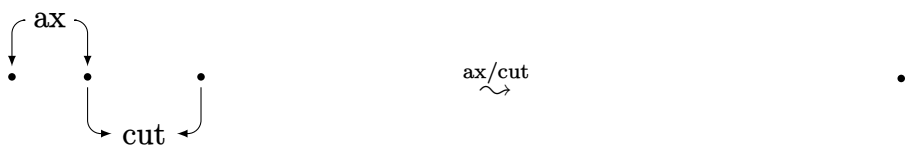
Definition 19 (Bunging)
Given a proof structure (so a structure without bungs), we call bunging the operation of adding extra $\circlearrowleft$ hyperedges for every $v \in \{v \in V \mid v \notin \mathtt{in}(e), \forall e \in E\}$.

Notation (Conclusions and atoms)
Given a proof structure $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$. We define:

- The *conclusions* of $\mathcal{S}$ are defined by the set of sources of bungs after having bunged the structure:
  $\mathtt{Concl}(\mathcal{S}) = \{v \in V \mid \exists e \in E, v \in \mathtt{in}(e), \ell_E(e) = \circlearrowleft\}$.
- The *atoms* of $\mathcal{S}$ are defined as the set of target of axioms hyperedge:
  $\mathtt{Atoms}(\mathcal{S}) = \{v \in V \mid \exists e, v \in \mathtt{out}(e), \ell_E(e) = \mathrm{ax}\}$.

As stated before, this is a model of computation, and we give it's computational rules:

Definition 21 (Computation rules of PS)
We give here the two computation rules making Proof Structures a model of computation:



Definition 22 (Formal MLL cut-elimination)
Let $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be an MLL proof structure with a cut $e_{\text{cut}} \in E$ such that $\mathtt{in}(e_{\text{cut}}) = \{v_1, v_2\}$ with both v_1 and v_2 being conclusions of some hyperedges e_1 and e_2.

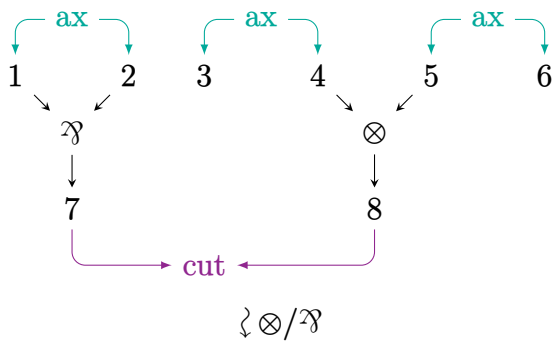
Left axiom If $\ell_E(e_1) = \text{ax}$ and $\mathtt{out}(e_1) = (v_0, v_1)$, with $v_0 \neq v_2$, then the elimination of e_{cut} is a new proof structure $\mathcal{S}' := (V', E', \mathtt{in}, \mathtt{out}', \ell_E)$ such that $V' := V - \{v_1, v_2\}$, $E' := E - \{e_{\text{cut}}, e_1\}$ and finally, $\mathtt{out}'(e') = v_0$ for any e' such that $\mathtt{out}(e') = v_2$ and $\mathtt{out}'(x) = \mathtt{out}(x)$ otherwise for any other $x \in E$.

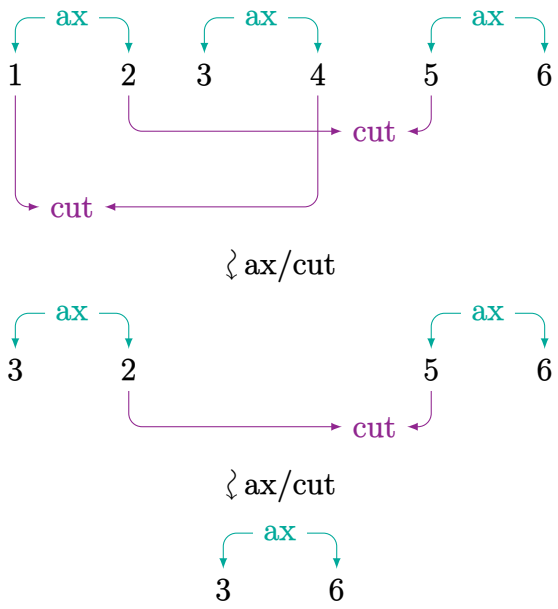
Right axiom We have a similar case for $\ell_E(e_2) = \text{ax}$ which correspond to the previous case in which we exchange e_1 (and v_1) for e_2 (and v_2).

Multiplicative Assume we have $\ell_E(e_1) = ⅋$ and $\ell_E(e_2) = \otimes$ (or the converse) with $\mathtt{in}(e_1) = (u_1, u_2)$ and $\mathtt{in}(e_2) = (w_1, w_2)$. The elimination of e_{cut} is a new proof structure $\mathcal{S}' := (V', E', \mathtt{in}', \mathtt{out}, \ell_E)$ with $V' := V - \{v_1, v_2\}$, $E' := (E - \{e_{\text{cut}}, e_1, e_2\}) \cup \{e^1_{\text{cut}}, e^2_{\text{cut}}\}$ and finally, $\mathtt{in}'(e^i_{\text{cut}}) = (v_i, w_i)$ for $i \in \{1, 2\}$ and $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise for any other $x \in E$.

Cut-elimination is really natural in proof structures; the case ax/cut is a graph contraction identifying two atomic formulas (and where the actual elimination happens) and the case $\otimes/⅋$ makes explicit the idea of rerouting of wiring.

Example (Cut elimination)
Here is an example of full cut-elimination:





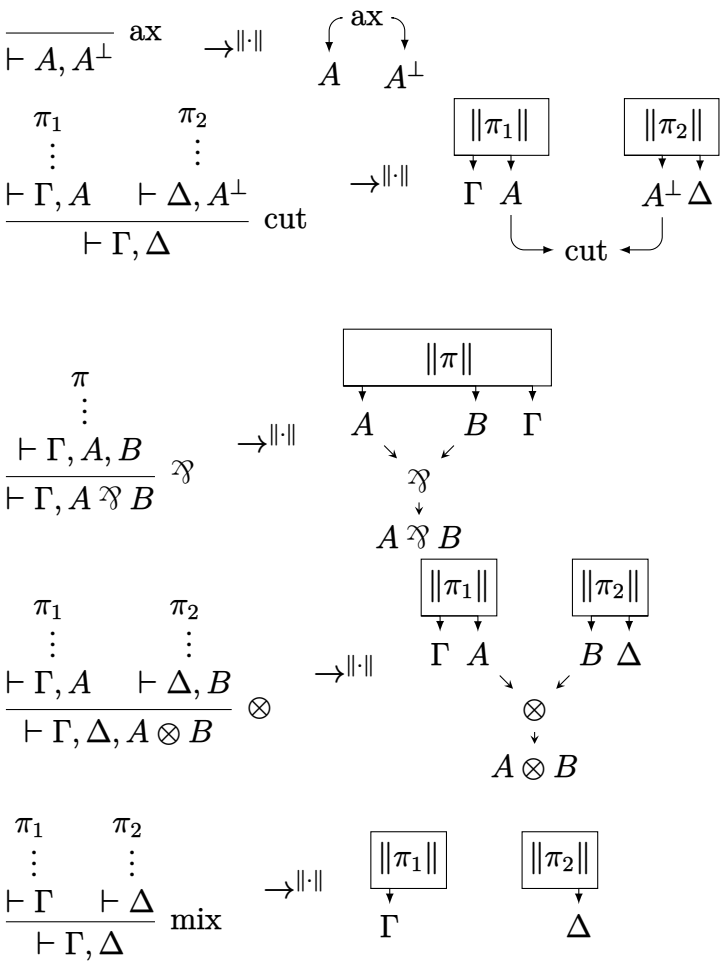
Since our previously defined Sequent Calculus is a model of computation, we can try to compile it into our system of Proof Structures. This is of course very easy since they were created for that very purpose.
To make this easier, we first define a labelling of the vertices of proof structures (for which labels correspond to formulas) in order to make proof structures closer to actual proofs. By doing so, we already implicitly give a little bit of meaning to almost purely computational notion of proof structure:

Definition 24 (Labelled proof structure)
A *labelled proof structure* is a tuple

$$\mathcal{S} = (V, E, \texttt{in}, \texttt{out}, \ell_V, \ell_E)$$

where $(V, E, \texttt{in}, \texttt{out}, \ell_E)$ is a proof structure and $\ell_V : V \to \mathcal{F}_{\text{MLL}}$ is a function labelling vertices of V by formulas.
We write $\vdash \mathcal{S} : \Gamma$ with a set of formula $\Gamma := \{\ell_V(v) \mid v \in \texttt{Concl}(\mathcal{S})\}$ in order to specify the formulas associated with the conclusions of $\mathcal{S}$.

Definition 25 (Translation from Sequent Calculus to PS)
One then just has to apply the rules of **??** to do the compilation:



In definition 25, we define a translation $\|\cdot\|$ from MLL+MIX sequent calculus derivations to labelled proof structures. Notice that this translation is not surjective, and that some proof structures do not represent sequent calculus proofs. This is tackled by the *correctness criterion*, which characterises those proof structures that do translate sequent calculus proofs through structural properties and which are considered "correct". For the time being, we give a preliminary definition of proof-net, the proof structures coming from sequent calculus proofs. The relationship between proof structures, proof-nets and sequent calculus proofs is illustrated in figure 2.2.

There is of course a form of simulation as well, with a correspondence between a step of cut-elimination in PS with multiple steps in the Sequent Calculus (there are more steps here because of the administrative work).

We are still interested in logic here, and so we can look at the PS that correspond to actual proofs.

Definition 26 (MLL and MLL+MIX proof-nets)
MLL Proof Nets are the images of sequent calculus proofs of MLL under the previously defined compilation into Proof Structure.

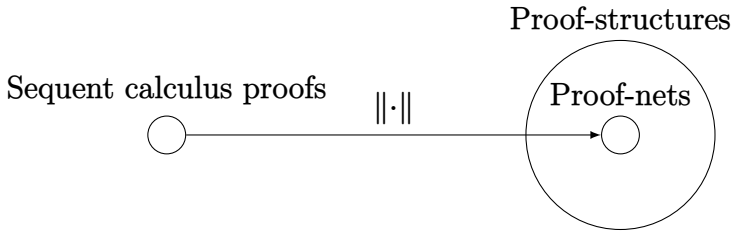


Figure 2.2: Proof Structures are general structures in which Proof Nets can be defined as special cases coming from the translation of sequent calculus proofs. Correctness criteria are ways to tell if a proof structure is correct, *i.e.* corresponds to a proof-net independently of sequent calculus proofs.

Formally, an MLL (*resp.* MLL+MIX) *proof-net* is a labelled proof structure $\mathcal{S}$ for which there exists an MLL (*resp.* MLL+MIX) sequent calculus proof π such that $\mathcal{S} = \|\pi\|$.

This definition can be thought of as a reference point of what a proof ought to be in the world of Proof Structures, because it was directly obtained from Sequent Calculus, which is agreed to be logical.

We deal here with computational objects that can be seen as proofs of different logical systems at the same time. For example, an MLL proof in the form of a PN is also an MLL+MIX proof. This justifies the following definition to factorize the two previous ones:

Definition 27 (Certifiability)
For a system S, we say that a proof structure $\mathcal{S}$ is S-certifiable when $\mathcal{S} = \|\pi\|$ with π a proof in S.

The problem with such a definition of correctness is that it is implicit: how can one tell whether a proof structure is actually a proof or not?
There is an important theorem of LL, arguably the most important, that gives an alternative, more intrinsic (and topological!) definition to PN, and which gave rise to a lot of litterature, the so-called "Correctness Criterions" [13, 12, 41, 46, 14, 49, 8].

There are many correctness criteria, to name just a few: the Counter-proofs criterion, the Mogbil-Naurois (Virtual Switchings) criterion, the permutation criterion.

In this thesis, we will be interested in the two, really close, most well known (and first discovered) criteria: the Long Trip criterion [25, Section III.2] and the

Danos-Regnier criterion [13, Section 3.2]. Similarly to how a product has to pass several tests in order to be certified, this criterion defines tests to pass in order to be logically correct.

Here, we will use Danos-Regnier's criterion.
We define the *correctness hypergraphs* associated to a proof structure $\mathcal{S}$ as undirected copies of $\mathcal{S}$ with one source of each $⅋$-labelled hyperedge removed. This decomposition of a proof structure into axioms and tests is illustrated in figure 2.4.
The Danos-Regnier criterion states that a proof structure is an MLL proof-net if and only if all its correctness hypergraphs are all connected and acyclic.

Notation
Given a proof structure $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$, we write $⅋(\mathcal{S})$ the subset of E of $⅋$-labelled edges, *i.e.* $⅋(\mathcal{S}) = \{e \in E \mid \ell_E(e) = ⅋\}$.

We will now define a variant of the notion of correctness hypergraph, where unary hyperedges are added as bungs to plug the conclusions (and which is the reason we introduced the notion of bunged proof structure):

Definition 29 (Correctness hypergraph)
Let $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be a proof structure.
A (Danos-Regnier) *switching* is a map $\varphi : ⅋(\mathcal{S}) \to \{⅋_L, ⅋_R\}$.
We will consider a copy of E written E', with the convention that to a certain e' corresponds a unique e.
The *correctness hypergraph* associated to $\mathcal{S}$ is the directed hypergraph with labelled hyperedges $\mathcal{S}^\varphi$ induced by the switching φ, obtained *after bunging* the hypergraph defined below:
We define $\mathcal{S}' = (V, E' \sqcup B, \mathtt{in}, \mathtt{out}, \ell_{E'})$ with E' a

- $\mathtt{in}(e') = \left\{\overleftarrow{e}\right\}$, $\mathtt{out}(e') = \mathtt{out}(e)$ when $e \in ⅋(\mathcal{S})$ and $\varphi(e) = ⅋_L$; Notice then how $\overrightarrow{e}$ becomes a conclusion and will be bunged.
- $\mathtt{in}(e') = \left\{\overrightarrow{e}\right\}$, $\mathtt{out}(e') = \mathtt{out}(e)$ when $e \in ⅋(\mathcal{S})$ and $\varphi(e) = ⅋_R$; Notice then how $\overleftarrow{e}$ becomes a conclusion and will be bunged.
- $\mathtt{in}(e') = \mathtt{in}(e)$ and $\mathtt{out}(e') = \mathtt{out}(e)$ in all other cases.

The labelling $\ell_{E'}$ is defined by $\ell_{E' \sqcup B}(e') = \varphi(e)$ when $e \in ⅋(\mathcal{S})$ and $\ell_{E' \sqcup B}(e') = \ell_E(e)$ otherwise.
Notice how $\mathtt{Concl}(\mathcal{S}^\varphi) = \emptyset$ since it is bunged, and if we remove bungs, $\mathtt{Concl}(\mathcal{S}') \simeq \mathtt{Concl}(\mathcal{S}) \sqcup ⅋(\mathcal{S})$.

Example
Here is an example of a proof structure with it's associated test in the most-widespread setting:

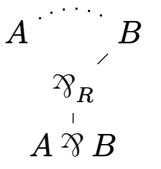
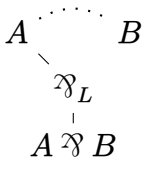
	Left switching	**Right switching**
Natural deduction	$\dfrac{\begin{array}{c}A^\perp \\ \vdots \\ B\end{array}}{A^\perp \multimap B}\ \multimap$	$\dfrac{\begin{array}{c}B^\perp \\ \vdots \\ A\end{array}}{B^\perp \multimap A}\ \multimap$
Proof-structures		

Figure 2.3: Switching for $\parr_L$ seen from the point of view of natural deduction. The premise $A^\perp$ and $B^\perp$ become conclusions A and B in monolateral sequent calculus and proof-nets.

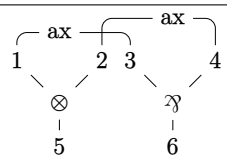
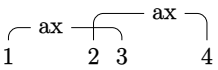
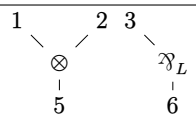
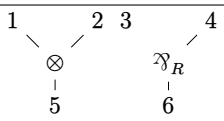
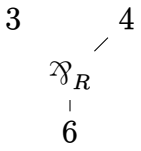
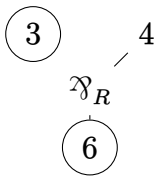
Structure	Axioms
	
Test 1	**Test 2**
	

Figure 2.4: The axioms and tests of a proof structure. The combination of axioms and a test corresponds to a correctness hypergraph representing a testing of the proof structure.

Example
In our setting, we go from having switching such as (notice how it has only one hyperedge, the $\parr_R$):



To switchings with extra bungs as hyperedges, with 3 hyperedges here:



We added hyperegdes bungs at location 3 and 6 since they are conclusions.

Theorem (Danos-Regnier correctness)
A proof structure $\mathcal{S}$ is MLL-certifiable (so, the image of a sequent MLL proof) if and only if it is MLL+MIX-certifiable and $\mathcal{S}^\varphi$ is connected for all switching φ.

Proof
Proven in Danos and Regnier's *"The structure of multiplicatives"* [13, Theorem 4]. □

The criterion for MLL+MIX is a variant of this one. Since MLL+MIX proof-nets allow disjoint union of MLL-certifiable proof-nets, it is sufficient to only consider acyclicity of correctness hypergraphs.

Theorem (MLL+MIX correctness)
A proof structure $\mathcal{S}$ is MLL+MIX-certifiable if and only if $\mathcal{S}^\varphi$ is acyclic for all switching φ. See [21, Theorem 4.7 and 4.8].

Remark
Intuitively, these "trials" correspond to a testing between the upper part of the structure, made of axioms, the *tested* (the program), which is opposed to switchings of the lower part of the structure, which are sort of *tests* (checking topologically that the behaviour of the program is going to be nice).

Note how these criteria are *external*, that is, it is a "meta" observer that can check whether a program is a proof or not, by looking at it's topology etc...

But if one wants to internalise a typechecker inside the programming language itself, it does not suffice: the typechecker, being a program, will need to use these criteria in an "interactive way". In the next chapter we will see how to get closer to that.

In this section, we saw how proof net can serve as a graphical syntax for equating proofs of Sequent Calculus. It is a somehow held belief that proof net are a form of Natural Deduction. I would argue that this is a somewhat misleading statement due to the mix-up between styles of Natural Deduction.

In the previously mentionned classification, it would be a form of open context Sequent Calculus, for it is cut-based.

Hole
Sequent Calculus has a nice left/right symmetry, which does not appear in Proof Nets, as they are a translation of the right handed version of the Sequent Calculus. It might be interesting to devellop a sort of proof net syntax with two "mirror worlds", one negated and the other not (and rules to move an edge from a world to the other) to capture these symmetries, which is, although I am not very familiar with it, probably part of the work of Pablo Donato: [16]
This should be a nice way of insisting on the duality environment/program. It might also serve as a better link with categories, where the symmetry part of the *-autonomy can be expressed through having an involutive negation functor between $\mathbf{C}$ and $\mathbf{C}^{\mathrm{op}}$

2.4. From Proof-Nets to Permutations

We saw in the previous section how a formal proof of LL in Sequent Calculus can be translated in the proof formalism of Natural Deduction (usually called PN).

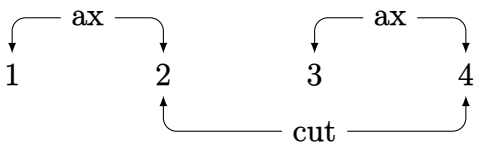
We saw how PS are a model of computation, but it is not completely alogical (the presence of $⅋$ and $\otimes$ inside is proof).

In this section, we will seek to extract the alogical *essence* of PN (at least the MLL ones), which will lead us to the model of computation of permutations, which will be our first model originating from GoI. We will also see how correctness criteria might be expressed in a more interactive way.

The model of permutations , and the ones that will follow are "long-trip style" model of computation.

Convention (Computational Content)
Take a proof net for MLL, with cuts. Notice how eliminating a $⅋\bot\otimes$ cut does not actually eliminate a cut, it just does rerouting
The real cut-elimination happens when a cut is against an axiom.
We can thus just preemptively do the "cut-elimination" without eliminating any cut and get an equivalent PS out of that process. This processed is described in more depth here: [47, section 5.1.2]
This PS will only have axioms and cuts. It will be called a pure PS in the following. In particular, it is important to note that in such a PS, the only vertices left are *atoms*, and that both cuts and axioms are only linked to said atoms.

Notice how, if you put little numbers on the extremities of axioms / cuts, then axioms and cuts looks like permutation on the set of number that you used:



Notation
For a given PS $\mathcal{S} := (V, E, \texttt{in}, \texttt{out}, \ell_E)$, we will write:

- $\texttt{Ax}(\mathcal{S}) \subseteq E := \{e \in E \mid e \in E, \ell_E(e) = \text{ax}\}$
- $\texttt{Cuts}(\mathcal{S}) \subseteq E := \{e \in E \mid e \in E, \ell_E(e) = \text{cut}\}$

Note how when the PS is pure in the sense defined above, then $\texttt{Ax}(\mathcal{S})$ and $\texttt{Cuts}(\mathcal{S})$ intersect if there are cuts.

Definition 38 (Permutations as a model of computation)
In the model of computation of permutations, the programs are pairs (L, σ) with $L \neq \emptyset$ a set and σ a permutation on L. We write σ_L to designate such a program, and we will call L *the location* of the permutation, it's elements will be *the atomic locations.*

Note (Locations)
Locations can be intuitively seen as points in a space. For MLL, they are also just the position of the different occurences of atoms in a formula.

Notation
We note $\ominus$ the symmetric difference of sets.

Definition 41 (Execution of permutation)
Given two permutations σ_L, τ_M, the set $C := L \cap M$ will be the location of the interaction (if one is a bunch of axioms, and the other a bunch of cuts, these are the set where they are glued together).
We define $\Cap (\sigma, \tau)_{L \ominus M}$ as the permutation on $L \ominus M$ associating to an element $x \in L \cap \overline{M}$, the first $y \notin C$ obtained by repeatdly applying σ and τ to x, and similarly for $\overline{L} \cap M$.

Note
This is basically computing a path from a conclusion to a conclusion.

Now that we have define our model of computation, we can translate our proof structures to it:

Definition 43 (Translation from Proof Structure to permutations)
Given a proof structure $\mathcal{S} := (V, E, \texttt{in}, \texttt{out}, \ell_E)$ we associate the permutation $\|\mathcal{S}\|_{ax} := \bigsqcup_{v,v' \in e \in \texttt{Ax}(\mathcal{S})} (vv')$, which is the disjoint union of the transpositions induced by every axiom on the set $\texttt{Atoms}(\mathcal{S})$

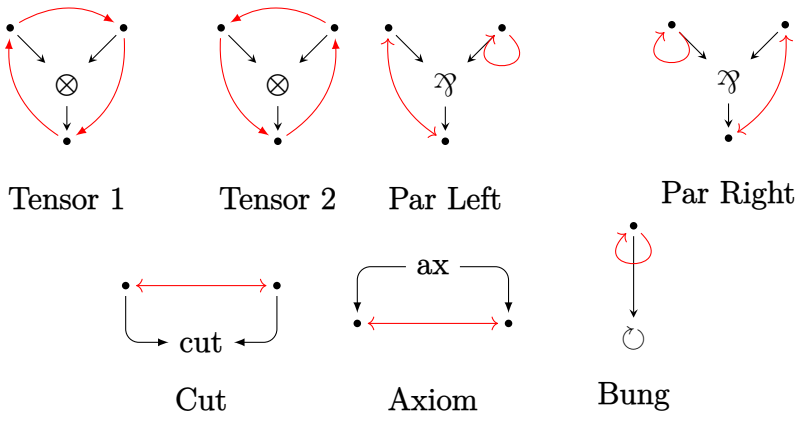
Definition 44 (Translation from PS to permutations for cuts)
Assume $\mathcal{S}$ is a pure proof structures in the sense if the convention above, that is there are no $\otimes$ or $\parr$ left.
We can also define a $\|\mathcal{S}\|_{cut}$ as the disjoint union of transposition induced on every cut on the set $\{v \mid v \in \texttt{out}(e), e \in \texttt{Cuts}(\mathcal{S})\} \subseteq \texttt{Atoms}(\mathcal{S})$.

The dynamics of cut-elimination will then be encoded as a sort of game of back and forth between the two permutations.

We saw how to interpret Proof Structures, not just Proof Nets. This means the model of computation we are defining will have programs that will not correspond to proofs, in the same way than some Proof Structures are not Proof Nets.

There is a correctness criterion that goes very well with this kind of model, called the Long Trip criterion (later on, we will study a model of linear realisability which is of Danos-Regnier style):

Definition 45 (Itinerary)
Let $P = (V, E, \dots)$ be a *bunged* PS. An itinerary I is given by a permutation σ_e for every $e \in P$, which, depending on the label of e, is in the following lists:



We note $I(P)$ the set of all itineraries in P.

Notation
Notice how, because the structure is *bunged*, the wire-network property implies that every vertex $v \in V$ is in exactly two hyperedges $\mathsf{e}^-(v)$ and $\mathsf{e}^+(v)$, and thus is in the domain of exactly two permutations.
We will write $\sigma^\bullet(v)$ to designate the permutation $\sigma_{\mathsf{e}^\bullet(v)}$ with $\bullet \in \{+, -\}$.
Finally, we define $\overline{+} = -, \overline{-} = +$ and define the notation $\overline{\sigma^\bullet(v)} = \sigma^{\overline{\bullet}}(v)$.

Definition 47 (Trip, long trip)
Given a bunged proof structure $\mathcal{S}$, a vertex $v \in \mathcal{S}$, and an itinerary I, we define a *trip* starting from v as the sequence of vertex $v_0 \to_{\sigma^\bullet(v_0)} v_1 \to \dots v_n$ with $v_0 = v$,

$v_{i+1} = \overline{\sigma^\bullet}(v_i)$ when $v_i = \sigma^\bullet(v_{i-1})$, and ending at v_n when $v_n \to_{\overline{\sigma^\bullet}(v_{n-1})}$ already appears in the sequence. (Such a sequence has to terminate for it has max length 2 times the number of vertices)
Notice there are thus two trips starting at v, since there are two directions to choose from a the beginning (σ^+ and σ^-.)

Property
In a trip, $v_n = v_0$.

Proof
A trip has to loop because there are finitely many possible way to go forward. If it looped on a certain $v_i, i > 0$, let's say without loss of generality that it is that we cannot take $\sigma^+(v_i)$ anymore, we would have $v_0 \to \ldots \to v_{i-1} \to_{\sigma^\bullet(v_{i-1})} v_i \to_{\sigma^+(v_i)} \ldots \to_\tau v_i$.
Because of the way the sequence is defined, $\sigma^\bullet(v_{i-1}) = \sigma^-(v_i)$ and $\tau = \sigma^-(v_i)$, so there was already a loop on v_{i-1} and the process should've stopped at the previous step. □

Definition 49 (Long Trip)
A trip is said to be a *long trip* if for all v, both $v \to_{\sigma^+(v)}$ and $v \to_{\sigma^-(v)}$ appear in the sequence.
Note how, if a trip is long, the starting edge does not matter (we would get the same trip in reverse), and every trip starting from a different v is long.

Theorem (Long-Trip criterion)
Let $\mathcal{S}$ be a PS.
It is correct iff all trips induced by itineraries $i \in I(\mathcal{S})$ are long trips.

Notice how an itinerary induces a permutation on our set of location:

Definition 51 (Permutation induced by an itinerary)
Given a PS $\mathcal{S}$, an itinerary $i \in I(P)$ induces a permutation τ_i on $\texttt{Atoms}(\mathcal{S})$, defined by $\tau_i(x) = y$ where y is the first vertex in $\texttt{Atoms}(\mathcal{S})$ reached by the trip starting from x and using as first permutation $\sigma^+(x)$ (the one that is not an axiom).

From this, the criterion can be reexpressed computationally inside our model:

Proposition (Computational Long Trip)
The trip p of origin x, defined by an itinerary $i \in I(\mathcal{S})$ is long iff the composition of two permutation $\|\mathcal{S}\|_{ax}\tau_i$ is a cyclic permutation.

Definition 53 (Orthogonality)
Two permutations σ, τ on L are said to be orthogonal, written $\sigma \perp \tau$ when their composition $\sigma\tau$ is cyclic.

This correspond to a notion of "execution going well" because of the following theorem:

Theorem
Let $\mathcal{S}$ be a PS, $\mathtt{T} := \{\tau_i \mid i \in I(\mathcal{S})\}$ be the set of tests generated from trips in $\mathcal{S}$. We have that $\mathcal{S}$ is correct iff $\|\mathcal{S}\|_{ax} \perp \mathtt{T}$

Proof
This is essentially because the computation of $\|\mathcal{S}\|_{ax}$ against a τ_i is the computation of a trip. The cyclicity expresses the fact that it is long (everything is visited). □

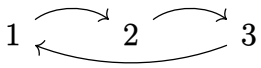
2.5. From Permutation to MLL Interaction Graphs

The previous section were about extracting the "computational essence" of proofs, and led to the definition of the model of computation of permutations .

We will now study the model of MLL interaction graph, which is a simple generalization of the model of permutations . This model was first defined by Seiller in [54] and is discussed in details in his PhD: [56]

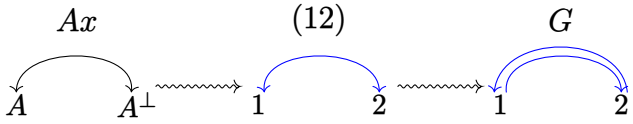
2.5.1. Interaction Graphs

Notice once again how a permutation (123) (this is an itinerary around a $\otimes$) can be represented as the following graph:



An axiom can be represented by this one:

The idea behind interaction graph is to simply generalise permutations: why not consider any graph instead of just the graph of a permutation?



Note
There are actually two "variants" of model for interaction graphs. Some with just edges, others with weighted edges.
If one considers weighted edges (what we will call the quantiative case), the model becomes powerfull enough to encode any kind of matrix by seeing the matrix as an adjacency matrix for the graph.
It is more expressive than matrix models, because there can be multiple arrows with different weight between two vertice, and because when considering matrix models, doing the execution $\pitchfork$ (which corresponds to computing paths in graphs) is obtained by summing powers of matrices, and the result has to be a matrix. Therefore, there has to be restrictions on convergence and considerations on norms. In Interaction Graphs, there are no restrictions on convergence.

Definition 56 (Directed Graph)
A directed graph G is a tuple (V_G, E_G, s_G, t_G), where V_G is a finite set of vertices, E_G is the set of edges, and s_G, t_G – the source and target maps – are functions from E_G to V_G.
We saw, and will see other models of computation in this manuscript. To keep notations consistent between these different models, we will denote by L_G the set of "locations" of G, here the set V_G.

Definition 57 (IG (Interaction Graph))
An interaction graph is a directed graph.
A weighted interaction graph is a weighted directed graph.

Remark
The name originates from [55]. This could also be called the "(directed) graph model", since the "permutation model" is named after the basic objects. The interaction name was chosen to echo the name Geometry of Interaction. Choosing a name allows to insist on the fact that it is not the basic object which is of interest but the dynamics it is equipped with. Finally, we sometimes abusively designate by Interaction Graph all the extensions that Seiller considered, so having a name for all of these is convenient.
It would be nice to reorganize a bit the naming convention of this field, since different models have different naming conventions (we could have called Interaction Graphs the "edge model" if we followed another used convention).

These graphs are programs, and will be the basic blocks whose behaviour we will describe using realisability.

Convention
We say an interaction graph G it is *at (location)* L when $L_G = L$
We will write $(IG)_L$ to designate the set of interaction graph at L and more generally S_L for any set S of interaction graph to say all elements of S are at L.

Definition 60 (Translation permutations → interaction graph)
Given a permutation σ on a set V, we define the interaction graph $\|\sigma\|$ as the graph (V, V, s, t) with $s(v) = v, t(v) = \sigma(v)$, and painted with the color "σ".

Definition 61 (Paths)
A path in a graph G is a sequence of edges $e_1 e_2 \dots e_n$ such that for all $i \in \{1, \dots, n-1\}$, $s_G(e_{i+1}) = t_G(e_i)$.
The source $s_G(\pi)$ (resp. the target $t_G(\pi)$) of the path π is defined as $s_G(e_1)$ (resp. $t_G(e_n)$). We denote by $\mathrm{Paths}_{X \to Y}(G)$ the set of paths p in G such that $s(p) \in X, t(p) \in Y$.
Paths will sometimes be written with their vertex explicitly, that is $v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \cdots v_n$

The paths that are of interest to us will be the alternating paths, which corresponds to an alternation between axioms and cuts, to question and answers, to call to the functions by an environment and response of said function. We need introduce a notion of color to encode an alternating path as a path where the color changes at every edge:

Definition 62 (Colored (Directed) Graph)
A colored directed graph is a directed graph equipped with a painting function $p : E \to \mathbf{C}$. We say that a graph is n-colored when $|C| = n$.

We can thus redefine our primitive notion:

Definition 63 (IG, again)
An IG is a colored directed graph with just one color.

Definition 64 (Alternating Paths)
An alternating path in a colored graph G is a path $e_1 e_2 \dots e_n$ such that for all $i \in \{1, \dots n-1\}$, such that $p(e_i) \neq p(e_{i+1})$.
The set of alternating paths in G from X to Y will be denoted $\mathrm{AltP}_{X \to Y}(G)$.

2.5.2. How to compose programs / graphs: the dynamics

Definition 65 (Glueing)
Given 2 colored directed graph F, G of disjoint colors, one can define the colored directed graph $G \leftthreetimes F$ as their (*non necessarily disjoint*!) locative union, that is:

$$G \leftthreetimes F := (V_G \cup V_F, E_G \sqcup E_F, s_G \sqcup s_F, t_G \sqcup t_F)$$

The coloring is given by $p = p_F \sqcup p_G$ (this is why we need disjoint colors)
More generally, we can define a glueing of graphs for an arbitrary (albeit with different colors) set of graph.

Note
All notions here can also be defined without colors: an alternating path between two graphs G, H is a sequence of edges $e_1 e_2 \dots e_n$ such that for all $i \in \{1, \dots n-1\}$, $e_i \in E_G$ if and only if $e_{i+1} \in E_H$.
But colors are a convenient way of talking about n-ary alternation inside a unique graph. To color a graph in a unique way, simply use it's name as a color.

Definition 67 (Binary Execution)
The *execution* of two interaction graphs G, H is the graph $G :: H$, with location V being $V_{G::H} := V_G \ominus V_H$ (symmetric difference) and whose edges are, letting G' be G painted with color "G", and H' be H painted with color "H", $\mathrm{AltP}_{V \to V}(G' \barwedge H')$, the alternating paths in $G' \barwedge H'$ of source and target in V.

Remark
There is a really interesting phenomena here: we colored the graph G with the color "G". I assumed implicitely that by G we refer to a particular *instance* of a graph. So even if mathematically, $G = G'$, this just mean that they are the same graph, not the same instance of a graph, and G' should have a different color. This is reminiscent of real-life programming, for example in object-oriented programming, with the mathematical equality being different than equality of adresses in the pointers, which often confuses beginners.
This is a true locative phenomenon, just like the remark by Girad that even when $a = b$, a is on the left and b on the right so they are different. This is something that traditional mathematics "does not see".

Hole
I go on a short tangent here, but part of what Girard did in Transcendantal Syntax is try to understand equality. Leibnitz equality, which is "$a = b$ iff $P(a) = P(b)$ for all P" is a bit unsatisfactory, in the sense that it depends on a $\forall P$, and this makes equality depend on the possible observations P we can make in our logic (typically, not the observation "being on the left/right of the operator "="). In particular, I think extending the possible observations one can make might "break" this equality. If one adds a predicate that would implement the function "quote" of lisp, then one would get access to the names of a and b and be able to differentiate them.
Maybe it is possible to understand "true equality" (whatever that means) using colors or locativity, saying two programs are the same when they have the same unique id, and this implies they are observally equivalent (reflexivity) but they could be observably equivalent through other means (one being a copy of the other). This is highly speculative, I unfortunately do not really understand what is griefs and plans are about equality.

Example
We illustrate here an exemple of execution, where we use regular expressions to describe paths:

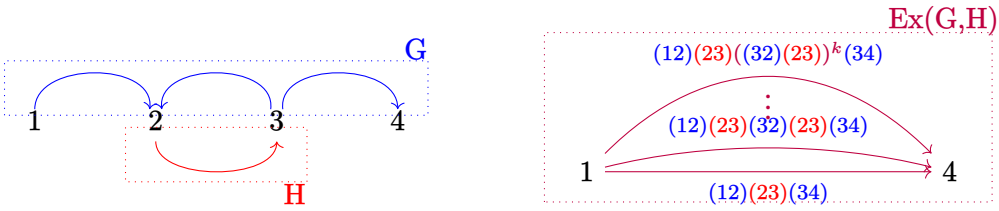


Figure 2.5: Two graphs and their execution

Note
Note how during execution, we forget that there ever was a "circuit"/"cycle" inside the graph, only remembering some information about these through the paths that went through them. It is not possible to tell whether the resulting graph was "already like that" or came from something with a cycle: we forgot part of it's geometry!
Even worse, in some cases, there is not even a trace left of cycles, if they are purely internal.
The last chapter is dedicated to discussing this.

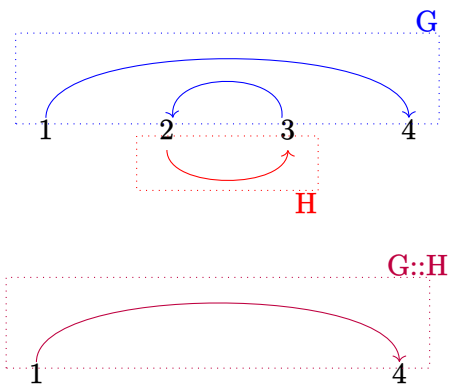
Proposition (Locative Associativity)
Given three graphs G, H, K with $V_G \cap V_H \cap V_J = \emptyset$, then $G :: (H :: J) = (G :: H) :: J$.

A nice representation of this, (see [57]) as a sort of Venn diagram with empty triple intersection, gives rise to a trefoil-like picture, hence the name of the following section.

2.5.3. The Trefoil Property

Notice how when doing execution, internal cycles between the programs can disappear.

See these examples of two graphs against each other, on respective locations $\{1, 2, 3, 4\}$ and $\{2, 3\}$.



But to be able to express the long-trip criterion (the whole composition visually forms just one cycle), we need to be able to know whether there was a cycle or not!

Definition 73 (Loops / Cycles)
An (alternated) loop c (also named a cycle) in an interaction graph is an (alternated) path $v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \cdots v_{n+1}$ with $v_0 = v_{n+1}$ and $p(e_1) \neq p(e_n)$ (that is, the last and first edge can be composed).
We call a loop c a d-loop for $d \geqslant 1$ the biggest integer such that $c = \rho^k$. We write $\mathcal{L}_d^k$ for the set of d loops of length k, dropping the k for the set of d loops of any length.

You will se later that 1-loops in $\mathcal{L}_1$ play the role of sorts of "prime" loops. Now, what we are interested in is the long-trip criterion, that is the existence (or not) of a "hole" in the graph, a visible loop. This formally corresponds to the notion of circuit:

Definition 74 (Circuit)
A circuit is an equivalence class of loops under cyclic permutation. For example $e_0e_1e_2 = e_2e_0e_1 = e_1e_2e_0$.
Intuitively, it is the "loop" that is "physically visible" in the drawings.
We define a 1-circuit as a circuit whose underlying loops are 1-loops.

We will be particularly interested in 1-circuits, since having exactly one indicates the presence of a long trip.

Notation
We will denote by $\mathrm{C}(G)$ the set of circuits of G, and $\mathrm{C}_1(G)$ the 1-circuits.

Remark
Notice that things such as the length of the circuit are not invariant under execution: some edges get merged together. So in a setting without a notion of weight, the only property we can "express" on graphs is the existence or not of circuits.

Notation
We will also write as $+$ the union of sets.

Proposition (Trefoil Property)
When $L_F \cap L_G \cap L_H = \emptyset$:

$$\mathrm{C}(F \parr (G :: H)) + \mathrm{C}(G \parr H) = \mathrm{C}(F \parr G \parr H)$$

$$\mathrm{C}(H \parr (F :: G)) + \mathrm{C}(F \parr G) = \mathrm{C}(F \parr G \parr H)$$

$$\mathrm{C}(G \parr (H :: F)) + \mathrm{C}(H \parr F) = \mathrm{C}(F \parr G \parr H)$$

(This is also true of 1-circuits, and can be made true of other family of circuits, for example when extending graphs by adding a coherence relation). The proof is

simple and can be found in [50].

Corollary (Adjunction)
When $L_G \cap L_H = \emptyset$, we can obtain:

$$\mathrm{C}(F \leftthreetimes (G \sqcup H)) = \mathrm{C}((F :: G) \leftthreetimes H) + \mathrm{C}(F \leftthreetimes G)$$

This is the adjunction (curryfication) of linear logic, which will give that $f \perp g \otimes h$ iff $f :: g \perp h$, by defining the orthogonality as having the same number of circuits.

2.5.4. The Quantitative Case

As stated in the introduction, there are variants of interaction graphs with weights on the edges that can be used to encode matrices, hence the following definition:

Definition 79 (Weighted directed graph)
Given a commutative monoid Ω, a weighted directed graph is a directed graph equipped with a quantification function $w \colon E \to \Omega$

We can re-do the previous considerations in this quantitative setting.

In the original setting of Seiller the weights of the edges were inside $\Omega =]0, 1]$. A good intuition to have is that this weight represent a sort of probability of taking a particular edge. The weight of a path is then the probability of taking said path, and the weight associated to a circuit is the probability of making one loop.
In such a setting, we are not just counting the circuits abstractly but making a sort of "bet": whether we are going to terminate or loop.

Convention
We denote by $\bar{\mathbb{R}}$ the set of real numbers completed with $+\infty$ and $-\infty$. We use the convention $ln(0) = -\infty$.

Proposition (Lifting of Quantification Function)
Given an G and a quantification function $w_G : E \to \Omega$
There is an obvious lifting of w to the set of circuits:

$$\hat{w} : \qquad \mathrm{C}(G) \longrightarrow \Omega$$

$$(e_0 \dots e_n) \longrightarrow \prod_i w(e_i)$$

(Note how it is well defined since Ω is commutative and thus is invariant under the action of the cyclic permutation).

Note
We could relax the requirement for Ω to be commutative, and just ask it's operation to be invariant under cyclic permutation. We will now use $\Omega :=]0,1]$ because we will have sums and we need a notion of convergence.

Definition 83 (Measure)
Given two interaction graphs F, G, we define the measure of the interaction between F and G: $[\![F, G]\!] := \sum_{\pi \in \mathrm{Circ}(G)} \pi$, which is a formal sum of the circuits between the two.

Now, this measure usually appears in the litterature expressed differently. For that, consider A to be a form of adjacency matrix of the graph $H := G \leftrightarrows F$ (the coefficients i, j being a formal sum of the name of the edges $i \to j$, and such that a non-alternated product goes to 0).

Notice that the coefficients of A^k are exactly the paths of length k, thus the elements of $A^k_{i,i}$ are in $\mathcal{L}_k$. If we want to count the circuit, and not the loop, we count it k times too much (every cyclic permutation of it).

Thus we have:

$$
\begin{aligned}
&\sum_{\pi \in \mathrm{Circ}(H)} \pi && \text{Counting} \\
&= \sum_{k \in \mathbb{N}} \frac{\mathrm{Tr}(A^k)}{k} && \text{Linearity} \\
&= \mathrm{Tr}\left(\sum_{k \in \mathbb{N}} \frac{A^k}{k}\right) && \text{Power series} \\
&= \mathrm{Tr}(-\log(1 - A)) && \text{Linearity} \\
&= -\mathrm{Tr}(\log(1 - A)) && \text{Link between exp of matrix and trace} \\
&= -\log(\det(1 - A))
\end{aligned}
$$

Which would be Girard's presentation of this measure.

Seiller has it's own presentation, which makes the sum more "compact" by using the fact that the 1-loops somehow "generate" the rest.

Notice how a loop $\pi \in \mathcal{L}^d_{d.n}$ corresponds to exactly one $\rho \in \mathcal{L}_1$ of length n, the one such that $\pi = \rho^d$. We will denote this ρ by $\tilde{\pi}$.

$$
\begin{aligned}
&\sum_{\pi \in \mathrm{Circ}(H)} \pi && \text{Counting}\\
&= \sum_{k \in \mathbb{N}} \frac{\mathrm{Tr}(A^k)}{k} && \text{def.}\\
&= \sum_{k \in \mathbb{N}} \sum_{\pi \in \mathcal{L}_k} \frac{\pi}{k} && \text{decomposing}\\
&= \sum_{k \in \mathbb{N}} \sum_{d|k} \sum_{\pi \in \mathcal{L}_k^d} \frac{\pi}{k} && \text{rearranging with } k = d.n\\
&= \sum_{d \in \mathbb{N}} \sum_{n \in \mathbb{N}} \sum_{\pi \in \mathcal{L}_{d.n}^d} \frac{\pi}{d.n} && \text{Observation above}\\
&= \sum_{d \in \mathbb{N}} \sum_{\tilde{\pi} \in \mathcal{L}_1} \frac{\tilde{\pi}^d}{d.|\tilde{\pi}|} && \text{Loop was counted too many times}\\
&= \sum_{d \in \mathbb{N}} \sum_{\tilde{\pi} \in \mathrm{C}_1(H)} \frac{\tilde{\pi}^d}{d.|\tilde{\pi}|}.|\tilde{\pi}| && \text{simpl and inversion}\\
&= \sum_{\tilde{\pi} \in \mathrm{C}_1(H)} \sum_{d \in \mathbb{N}} \frac{\tilde{\pi}^d}{d} && \text{power series}\\
&= \sum_{\tilde{\pi} \in \mathrm{C}_1(H)} -\log(1-\tilde{\pi})
\end{aligned}
$$

I of course did not define what $-\log(1-\tilde{\pi})$ means (it does not mean anything), but when considering weighted models (which is what Seiller does) then $-\log(1-\hat{w}(\tilde{\pi}))$ has a meaning.

Seiller discovered that there is "nothing special" about the function $-\log(1-x)$ in the sense that other functions also define orthogonalities, even if they might not capture the long trip criterion. We call such functions measure of (1-)circuits.

Definition 84 (Measure of Circuit)
Given an IG G, a measure (of circuit) is a function $m : \mathrm{C}_1(G) \to \bar{\mathbb{R}}$. The induced measurement is $[\![G]\!]_m = \sum_{\pi \in \mathrm{Circ}_1(G)} m(\hat{w}(\pi)) \in \bar{\mathbb{R}}$

Proposition (Numeric Trefoil and Adjunction)
When $L_F \cap L_G \cap L_H = \emptyset$, given a quantification function w on $F \between G \between H$, and a measure of circuit m, there is the *numeric trefoil (/ adjunction)*:

$$[\![F \between (G :: H)]\!]_m + [\![G \between H]\!]_m = [\![F \between G \between H]\!]_m = \dots$$

Where now $+$ is the addition of real numbers (and not the union of sets)!

Remark
There is a link between all this cycle counting and zeta functions, discovered by Seiller [59].

2.5.5. The Wager as a solution to the trefoil problem

Let's have a look at the adjunction again:

$$\mathrm{Circ}(F \between (G \sqcup H)) = \mathrm{Circ}(H \between (F :: G)) + \mathrm{Circ}(F \between G)$$

What we want is an adjunction between $\otimes$ and $\multimap$, that is something "akin to" $\mathrm{Circ}(F \between (G \sqcup H)) = \mathrm{Circ}(H \between (F :: G))$. Unfortunately, there is an extra term $\mathrm{Circ}(F \between G)$ that we have to account for.

An idea of Girard was to add a number, called *the wager* that will count the circuits that disappear during execution to compensate for that. This leads to extending the model to create a new one.

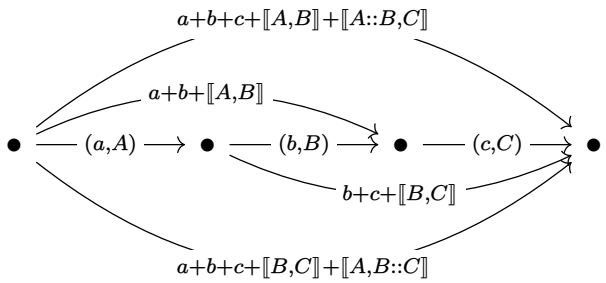
Hole
There is a certainly a link to make with the work of Kelly and Laplaza [40]:
In their explicit construction of the free compact close category, they introduce elements that are used to count cycles in a similary way.

Convention
We assume given a monoid Ω.

Definition 89 (The model of computation of projects)
A project is a pair (a, p) with p an interaction graph, and $a \in \Omega$. The elements of Ω are called the wagers (the intuition being, I think, that you are betting on non termination: if you see the weight of an edge as the probability of taking it, this is linked to the probability of getting stuck in an infinite loop) When dealing with wagers, the first projection is written $\ll . \gg$

Definition 90 (Dynamics of projects)
We can extend execution to projects: $(a, A) :: (b, B) = (a + b + [\![A, B]\!]_m, A :: B)$.

Note (Trefoil and associativity)
Note how the trefoil property is precisely the condition that makes it so that composition of projects is associative (we focus only on the wagers here):



2.5.6. A first taste of Linear Realisability

From this newly defined model, we can create a model of MLL. We will redo this in the next chapter in a more abstract way.

Convention
In this section, we will take $\Omega = \bar{\mathbb{R}}$.
We assume given a set $\mathbb{L}$ of locations that will be used as vertices for our graphs.

Definition 93 (Orthogonality)
Given projects α, β at L, we say $\alpha \bot \beta$ when $\ll \alpha :: \beta \gg \notin \{0, \infty\}$.

To understand this, let us look at the sum $\sum_{\tilde{\pi} \in \mathrm{C}_1(H)} -\log(1-\tilde{\pi})$. Let us pretend there is only one $\tilde{\pi}$. The function $-log(1-x)$, it is going from 0 at 0 to $+\infty$ at 1.

Thus, if it is 0 then the weight of $\tilde{\pi}$ is 0, there is actually no circuit. If it is ∞ then there is a circuit but of weight 1, it loops infinitely and we cannot escape it. This orthogonality thus looks at whether there exists "manageable" circuits.

Proposition (Adjunction)
Take p, q, r projects, then $p :: q \bot r$ iff $p \bot q :: r$.

Proof
Obvious since $\ll p :: q \bot r \gg = \ll p \bot q \otimes r \gg$ by the numeric adjunction. □

Definition 95 ((pre-)Behaviour)
A pre-behaviour is a localised set of interaction graph P_L with $L \subseteq \mathbb{L}$, that is all graphs in P_L have location L.
We will often omit the set L of locations, writing just P. In such context, we will denote by L_P the set of locations of P.
From a pre-behaviour P_L, we can define it's orthogonal $(P^\bot)_L := \{q \mid q \bot p, \forall p \in P\}$.
Notice how it is defined at L as well.
Finally, a behaviour is a pre-behaviour generated from a set of tests, that is a

$P = T^\perp$ with T a pre-behaviour.
The set of behaviours will be denoted by `Behav`

Intuition
Programming is about using functions, that is giving arguments to a program whose type uses $\multimap$, in the same way than the core of logic is modus ponens.
The point of this construction is to define the behaviour of functions: keeping in minde the idea that the orthogonality $\perp$ (here we take a pole) represents a good computation, then a set $A^\perp$ is akin to $A \to \perp$. In particular, $(A \otimes B)^\perp = (A \otimes B) \multimap \perp$ and will thus be made of programs behaving like "functions that behave well on arguments of the form $a \otimes b$".

Proposition (Closure)
A set P is a behaviour iff $P = P^{\perp\perp}$.

We define the two main behaviours of MLL:

Notation (Tensor on Programs)
From a_L, b_K, two projects with $L \cap K = \emptyset$, we write $a \otimes b := a :: b$ to indicate that the locations are disjoint.

Definition 99 (Tensor on Behaviours)
From A_L, B_K, with $L \cap K = \emptyset$, we define $(A \otimes B)_{L \sqcup K} := \{a \otimes b \mid a \in A, b \in B\}^{\perp\perp}$.

Note
Notice how these two behaviours are equal: $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ and not simply "isomorphic" as they would be in a non locative setting (for example, category theory). This kind of properties of locativity allow to avoid dealing with things such as Mac Lane's pentagon, which is nice.

Definition 101 (Implication)
From A_L, B_K, with $L \cap K = \emptyset$, we define $(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A, p :: a \in B\}$.

Remark
Note how $\otimes$ is defined through an introduction rule and $\multimap$ is defined through an elimination rule.

Note (Finiteness)
It is easy to create an object and be assured that it is in $A \otimes B$ (just make one of the form $a :: b$) but it is definitely not easy to check that something is in $A \multimap B$, for it naively would require infinitely many tests. One of the point of TS is to define everything in a *finite, implementable way.* Here we are just doing regular GoI.

Proposition (Duality)
$(A \multimap B)$ is equal to $(A \otimes (B^\perp))^\perp$ and (thus is a behaviour).

Proof
The proof will be done in the next chapter, we skip it for now. □

Note (What about the rest of LL?)
I sometimes call interaction graph under the name MLL interaction graph.
They can actually realise exponentials as well, but they cannot do so in a "finite way", by that we mean here that the number of edges and nodes (so, the graph description) stays finite.
In a quantitative model, a lot of weight thus become infinite, which is troublesome. The usual exponential is $- \times \mathbb{N}$ and is not finite, but it can still be finitely implemented in a computer provided one does not explicitly write the entire graph, but handle edges and locations in a "smart way". This is the kind of power that is brought about by the models in [24] and [23], which in turn led to the creation of `Flows`. Fortunately, there are also extensions of interaction graphs that are finer and more expressive, allowing for a new kind of exponential: [51], which is *entirely finite*, that is both graphs and weights stay finite when using said exponential.

We end this subsection with a quick discusssion on an abstract way of going back to this form of "proof theoretical semantics" to the "truth value semantics", which is the notion of correct/winning program:

Definition 106 (Correct (winning) Program)
A project (a, A) is correct when $a = 0$ and A is a disjoint union of transposition (*i.e.* , the image of an axiom part in a PN).

Definition 107 (Truth)
A behaviour A is *true* when there is a correct program $p \in A$.

We expect two things from a notion of truth:

Proposition (Consistency)
A and $A^{\perp}$ cannot be true at the same time.

Proposition (Compositionality)
If $f \in A \multimap B$ is correct and $a \in A$ is, then $f :: a \in B$ is. In particular, if A is true and $A \multimap B$ is true then B is true.

2.5.7. Interpreting MLL

Now that we defined the behaviour of interaction graphs, we know, from all the things that were previously discussed, that there is theoretically a chain of interpretation MLL $\to$ PN $\to$ permutations $\to$ interaction graph, and that we could thus interpret MLL inside the model.
It is still very instructive to see how the interpretation is done directly, and thus we do it here:

Definition 110 (Implementation)
Given locations L, K and a function $f : (\texttt{Proj})_L \to (\texttt{Proj})_K$, we say that a project p *implements* f, written $p \Vdash f$ when for all project $q \in (\texttt{Proj})_L$, $p :: q = f(q)$.

Definition 111 (Delocations)
Let $L \cap K = \emptyset$ be disjoint locations isomorphic through $\Phi : L \xrightarrow{\sim} K$, and $p = (a, G)$ a project on $L \sqcup K$.
We call *the delocation of p through Φ* the project $\Phi(p) := (a, \Phi(G))$ with $\Phi(G)$ the same graph as G but on location K.
We call *a delocation* the project with wager 0 and program the graph $\texttt{mov}_{L \to K} := (L \sqcup K, \{a \to \Phi(a) \mid a \in L\} \sqcup \{\Phi(a) \to a \mid a \in L\}, \dots)$.
This project implements Φ, that is, $\texttt{mov}_{L \to K} :: p = \Phi(p)$ for all p at L.

Usually, propositional logic deals with fixed second order variables called propositional variables. In a localised setting, we must distinguish every occurence of every variable, hence the following convention:

Convention (Set of Variables)
We assume given two infinite sets $\texttt{Names}$ and $\texttt{Instances}$ (usually $\mathbb{N}$).
We now assume given a set $\mathcal{V}\text{ar} := \{X_i(j) \mid i \in \texttt{Names}, j \in \texttt{Instances}\}$ of propositional variables.

Definition 113 (Localised Variables)
A family $X_i(-)$ is called *a variable name.*
An element $X_i(j)$ of such a family is called *an instance*, or a *localised variable* of name $X_i(-)$.

Convention (Localisation of the Localised Variables)
We assume given a family of disjoint locations $L_i(\bullet)_{i \in \texttt{Names}}$, so that a variable name has a "location of reference".
We also assume given a family of disjoint locations $L_i(j)_{i \in \texttt{Names}, j \in \texttt{Instances}}$, with the property that $L_i(j)$ are all isomorphic to the location of reference $L_i(\bullet)$ through $\texttt{mov}^i_{j \to \bullet}$.

Remark
Note this is possible to do when the set of location is infinite, for example $\mathcal{P}(\mathbb{N})$.

Notation (Localized formula)
We will write A_L to say that A is a formula with location L.

As stated before, the interpretation requires a logical system that is slightly different than MLL, called location MLL, where every atom appears only once. It is easy to see they are completely equivalent.

Definition 117 (Formulas of location MLL)
We define inductively the formulas and their locations:

- $X_i(j)$ is a formula of location $L_i(j)$.
- $\sim X_i(j)$ is a formula of location $L_i(j)$.
- Given A_L, B_K with $L \cap K = \emptyset$, then $(A \otimes B)_{L \sqcup K}$ is a formula.
- Given A_L, B_K with $L \cap K = \emptyset$, then $(A \parr B)_{L \sqcup K}$ is a formula.
- $\bot$ is a formula of location $\emptyset$.
- 1 is a formula of location $\emptyset$.

Remark
Note how we can go back and forth from this system to MLL by forgetting the j indices, or by reading the formula from left to right and adding back j indices to differentiate between instances of variables.

The interpretation

Definition 119 (Basis of interpretation)
A *basis of interpretation* is a (depentently typed!) function $\Phi : i : \texttt{Names} \to (\texttt{Proj})_{L_i(\bullet)}$, that associate to every name of variable i a project (program) at location of reference $L_i(\bullet)$.

Definition 120 (Interpretation of Formulas)
We interpret formulas as behaviours, defining $\|-\|_\Phi :\to \texttt{Behav}$ recursively:

- $\|X_i(j)\| := \texttt{mov}^i_{\bullet \to j} :: \Phi(i)$
- $\|\sim X_i(j)\| := (\|X_i(j)\|)^\bot$
- $\|(A \otimes B)_{L \sqcup K}\| := (\|A\| \otimes \|B\|)_{L \sqcup K}$
- $\|(A \parr B)_{L \sqcup K}\| := (\|A\| \parr \|B\|)_{L \sqcup K}$
- $\|\bot\| := \bot$.
- $\|1\| := 1$.

As for sequents, $\|\vdash A_1, \dots A_n\| := \|A_1\| \parr \dots \parr \|A_n\|$.

Definition 121 (Interpretation of Proofs)
We use a notation $\text{cut} : \pi_1, \pi_2$ to designate the fact that a proof was obtained as a cut between proof π_1 and proof π_2, and similarly for other rules.
We interpret proofs as programs, that is projects, defining $\|-\|_\Phi$ by induction on a proof π', we look at the last rule used:

- $\|\sim X_i(j), X_i(j')\| := \texttt{mov}_{j \to j'}$, initialisation case (ax)
- $\|\text{cut} : \pi_1, \pi_2\| := \|\pi_1\| :: \|\pi_2\|$

- $\|\parr : \pi\| := \|\pi\|$
- $\|\bot : \pi\| := \|\pi\|$
- $\|1\| := (0, \emptyset)$
- $\|\otimes : \pi_1, \pi_2\| := \|\pi_1\| \otimes \|\pi_2\|$

Proposition (Correctness)
If $\vdash \pi : A$ then $\|\pi\| \in \|\vdash A\|$, moreover $\|\pi\|$ is correct (thus A is true).

Proposition (Invariance by Cut-Elim)
If π is a proof of location MLL, and $\pi \to \pi'$ via cut elimination, then $\|\pi\| = \|\pi'\|$.

The proofs can be found in [56].

Note
Notice how we actually interpret proofs by programs in normal forms.

Reflexion
If we take a step back, there is something "a little bit weird" about what we are doing: we are interpreting a model of computation, the one location MLL plus cut-elimination in another model of computation, the one of interaction graph. But we are doing something of "denotational" flavor, that is, to each program we associated a program in normal form, completely forgetting about the dynamics of interaction graph in the process.
A "more natural" way of doing such a proof would be to do it in two steps: define a compilation of location MLL inside interaction graph, traditionally called a simulation.
Then one can associate to every interaction graph it's normal form due to associativity, giving back the "denotational flavor", although the computationally interesting part is the previous step, is not it?
This might be possible if one extends the innovations appearing in the next chapter, where we will give a form of "medium step semantics" to interaction graphs.

2.5.8. Categorically

There is a way to fall back on "usual semantics shenanigan" by showing that interaction graphs can be used to create a $*$-autonomous Category. We just give the definition here:

Definition 126 (Project Category)
Objects: := Localised behaviours with locations in $\mathbb{N}$.

Morphisms: Morphisms of type $A \to B$ are the elements of $\Psi_0(A) \multimap \Psi_1(B)$

Composition: Composition is done by identifying $\Psi_1(B)$ and $\Psi_0(B)$, and then doing execution.

Identity: The identity 1_A is the straight line graph from $\Psi_0(A) \multimap \Psi_1(A)$

With $\Psi_i : \mathbb{N} \to \mathbb{N} \times \{0,1\}$, doing $x \to (x,i)$.

Note
Notice how we apply an operation to the left and right side of $\multimap$ to form the homset! This is a form of delocalisation.
A simple computation shows that it allows a composition of three morphism to verify the locative conditions for associativity.
There is no such thing as a $\texttt{mov}_{A\to A}$, so without delocation there would not be any identity.

Proposition
The category **Project** is a $*$-autonomous Category, see [54].

2.6. A discussion on locativity : different point of views

In this really short section I would like to discuss what is locativity exactly?

Originally, locativity is about making the difference between boths A in $\vdash A, A$. After all, one is on the left and one is on the right. They somehow refer to the same "A" but in terms of pure computation, this is not something that can be seen.

More generally, locativity is about giving identities to things, to be able to talk about *instances* of things, which is something rarely seen in math (but really frequent in computer science). To make an analogy, it is a bit like having pointers everywhere, and two a priori identical things might be different because they live on two different memory cells. The adress stored in the pointer would be the location.

Using locativity has strenghts and drawbacks. Its main weakness is that it can be really annoying/bureaucratic to keep track of all locations. On the other side of the coin, it makes some things simpler, for example, in a locative system: $A \otimes B = B \otimes A$, same for associativity. This is famously not the case in category theory, and it requires extra structures, or even theorems such as *MacLane's coherence theorem* to be used.

Hole
The name locativity is also used in Indexed Linear Logic, and the word copy indices, which is used in game semantics seem to have similar responsabilities.

I would like to understand this better. A good reference on this should be the PhD of Etienne Duschesne: [18]

3. An overview of GoI as Linear Realisability

In the previous chapter, we discussed a really "primitive" model of GoI, Seiller's MLL interaction graph.

As its name implies, its a fairly really weak model of computation, only able to *finitely* implement MLL.
By that we mean that it one can realise MLL by the means of finite (finite vertices and edges) graphs. Realising the exponential requires the use of infinitely many vertices.
It can of course be extended to more expressive variants:

- Sliced interaction graphs: this add slices (*i.e.* taking formal sums of interaction graph), which is a fairly traditional way of implementing additives: see [50]
- Thick Graphs : this links slices together, allowing them to communicate (they were independent in the additive construction, as they were just a formal sum). This allows to get an exponential that has the seely isomorphism: $!(A \,\&\, B) \simeq !A \otimes !B$. (Notice how there are 2 slices on the left side, before the !, that get fused in just one slice on the right, as a $\otimes$). A thing of note is that in these Seiller discovered a new kind of exponential, which has really good quantitative properties. See [51] and [52].
- Graphings: this thickens the locations, making them topological spaces. This allows to implement second order : see as defined in [53].

Something noteworthy is that in the same way topological spaces have a "quantitative variant", that of metric spaces, to get more expressive measures one can switch from topological spaces to metric spaces. As for interaction graph, every model has a quantitative variant.

Since we have discussed interaction graphs in the previous section, we now have an example of model of computation that we can use as reference to do more linear realisability. Because we will consider multiple models of computation, we will do the realisability once and for all, in an abstract way, at the end of the chapter.

We will first discuss other models of computation, that share the fact that their location system is "dynamic" instead of static.

By that we mean that we do not statically enforce that only "compatible" edges can be composed, we will have an element $\perp$ representing a failure of composition.

3.1. A new variant of Interaction Graphs

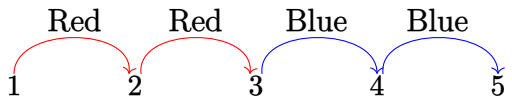
In this section, we will study a variant of interaction graphs, where we will compute locations dynamically.
This will lead us to introduce an abstract notion of model of computation, and the abstract notion of diagram. Our model, the model of bi-colored interaction graphs, will be a guiding example.

They will also have a "smaller-step" semantics than the usual interaction graphs, that we will call "medium step semantics", in the sense that they will have more control over the finer details of execution.

To prove associativity, it is often convenient to define an n-ary execution, and to prove the actual associativity theorem via the following statement: $\mathrm{Ex}(\mathrm{Ex}(A,B),C) = \mathrm{Ex}(A,B,C) = \mathrm{Ex}(A,\mathrm{Ex}(B,C))$ which goes through a ternary execution.
In the case of interaction graphs, Ex is ::. Remember how it was defined going through an intermediary notion of colored graph. In particular, this statement implicitly went through an intermediary definition that uses a 3-colored graph $A \amalg B \amalg C$. This naturally leads to the direct study of multi-colored graphs instead of interaction graph, which could be considered a sort of "1-colored graph". Instead of executing a graph against others, we will glue all the graphs together, with different colors, and execute them all at once.

Note
Notice how in the more general world of colored graphs the glueing operation $\amalg$ is a partial endomorphism (partial since only defined with non interfering colors).

There is a problem doing just that, especially if we want to be able to do "progressive reductions", in a smaller fashion than the exeuction :: which is a big step semantics (there is only 1 step). Imagine composing the two middle edges in the following graph:



What color should the resulting arrow have?
It cannot be Red, or else we could compose with the last arrow. Same for Blue. And it cannot be a new fresh color or else we could compose with both.

This leads us to consider edges that are bicolored. In the same way than an edge has a source and target location, it will have a source and target color.

Definition 131 (Bicolored Interaction Graph)
A *Bicolored Interaction Graph*, abbreviated as bic(olored)ig is a directed graph equipped with a painting function $p : E \times \{s, t\} \to \mathbf{C}$ with $\mathbf{C}$ a set whose elements are called colors.

Remark
By bi-colored we mean that the edges are bi-colored, but the graphs can use many colors.

Definition 133 (Alternating Paths in bicolors)
A path $e_0 \dots e_n$ in a bicig is said to be alternated when $p(e_i)(t) \neq p(e_{i+1})(s)$ for all i.

Definition 134 (Redefinition of IG)
An interaction graph has the same data than a bicig with only one color (constant p function).

Note
The idea behind this is that in an edge we want to keep exactly the data that is relevant in a path. An edge and a path should contain exactly the same data.
This leads to the observation later on that maybe we shouldn't bother with edges and directly deal with what will correspond to paths.

This change of point of view will allow to define an execution formula on a single graph. The usual execution $G :: H$ was indeeed defined as $\mathrm{Ex}_K(G \between H)$, an execution on the multicolored graph $G \between H$.

In the original definition of interaction graph, the execution was done in three steps:

- First, we glue G and H.
- Then we compute the alternated paths between G and H, that go from and to outside of the location of the cut $L_G \cap L_H$.
- Then we remove the vertices $L_G \cap L_H$.

We will define a model where these dynamics are done in a finer grained way, by decomposing the two last steps: one could indeed remove vertices one by one, and when removing a certain vertex, compute the paths that went through this vertex. This will give us a notion of execution Ex_K where K is the set of locations we want to declare "inside the cut".

Hole
This semantics I dubbed medium-step, for there is an even smaller-step semantics (and this is the one that would deserve such a title) which is the one where we

actually compute the paths: here, we remove the locations one by one, but in doing so we compute possibly infinitely many paths. A real small-step semantics should compute the paths little by little, and there should be a rule to allow removing a vertex when all necessary paths have been computed.
This would be very technical but it might be interesting to look at, since it would be closer to the real world, with a natural notion of non termination etc...

All the models we consider in this section are of a family of graph models. Because we want to avoid to repeat the proofs involved in defining a model of computation (associativity of execution etc...) and try to abstract away some details of said proofs to be able to compare them, we will do this in an abstract setting using a notion of "diagram".

A diagram, in the case of our models, will just be a very "verbose" way of describing a path. The reason we do this is discussed in a bit more detail in the next section, for now, we abstract what a path is:

Definition 137 (Partial semigroup)
A partial semigroup is given by $(\mathbb{P}_a, ;, \bot)$:

- Where $(\mathbb{P}_a, ;)$ is a semigroup.
- $\bot \in \mathbb{P}_a$ an absorbing element, representing failure of computation.

In the setting of interaction graphs, the edges form a partial semigroup, where composition fails when the endpoints do not match:

Definition 138 (Composition of edges)
Let $S := \mathbb{N}^2$, we write $n \to m$ for the pair $(n, m) \in S$.
Let $\mathbb{P}_a(IG) := S \sqcup \{\bot\}$, we define a composition ";": $\mathbb{P}_a(IG) \times \mathbb{P}_a(IG) \to \mathbb{P}_a(IG)$ as follows:

$$n \to m; m \to l := \begin{cases} n \to l \text{ if } m = m' \\ \bot \text{ otherwise} \end{cases}$$

And $\bot$ absorbing.

Proposition
$(\mathbb{P}_a, ;, \bot)$ as defined above is a partial semi-group.

Because we want to consider a setting with bicolored edges, we have to extend this definition as follows:

Definition 140 (Composition of edges)
Let $S := (\mathbb{N} \sqcup \mathbf{C})^2$, we write $(n, c) \to (m, c')$ for the quadruple $(n, c, m, c') \in S$.
Let $\mathbb{P}_a(\text{bicig}) := S \sqcup \{\bot\}$, we define a composition ";": $\mathbb{P}_a(\text{bicig}) \times \mathbb{P}_a(\text{bicig}) \to$

$\mathbb{P}_a(\text{bicig})$ as follows:

$$(n,c) \to (m,d); (m',c') \to (l,d') := \begin{cases} (n,c) \to (l,d') \text{ if } m = m' \text{ and } d \neq c' \\ \bot \text{ otherwise} \end{cases}$$

And $\bot$ absorbing.

Proposition
$(\mathbb{P}_a, ;, \bot)$ as defined above is a partial semi-group.

Notice how in the notation $1 \to 2$, the locations of the edge are given. Because of this, an interaction graph is exactly given by a set V of its locations, and a multiset of elements in $\mathbb{N}^2$, because there can be multiple edges $a \to b$ in a single graph. Moreover, when considering a path, it is convenient to give an identity to an edge in said graph, to be able to say that the path took this specific edge and not another one, in the same kind of tensions present all throughout linear logic, between the use of multisets (no identity, implicit quotient) and families up to reindexation (identities, but explicit quotient). This leads us to consider edges as *an indexed family* and not just a multiset.

We arrive at the following definition:

Definition 142 (Redefinition of a bicig)
A bicig is given by:

- A set of $L \in \mathcal{P}(\mathbb{N})$, called *the set of locations* (vertices).
- An indexed family of "atomic wires" $I \to \mathbb{P}_a(\text{bicig})$ (edges).
- Functions $s, t : \mathbb{P}_a(\text{bicig}) \to \mathbb{N}$.
- A painting function $p : I \times \{s,t\} \to \mathbf{C}$.

Quotiented by reindexing of the family.
In particular, an *interaction graph* is a bicig with constant painting function.

In the following, we abstract this away to get a definition for a family of models of computation, depending on a system of locations. These abstract notions will be used to do our proofs once and for all.

Definition 143 (Locative System)
A locative system $(\mathbb{L}, \cup, 0, \cap, 1, \overline{\bullet})$ is given by a boolean algebra. Its elements are called locations.
We will write $\subseteq$ for the order induced by the algebraic structure.

Definition 144 (Locative model of computation)
A locative model of computation is given by: $\mathbb{L}, ((\mathbb{P}_a, ;, \bot), \mathtt{in}, \mathtt{out}, p)$

- With $\mathbb{L}$ a locative system.

- With $(\mathbb{P}_a, ;, \perp)$ a partial semigroup, whose elements are called atomic wires.
- Two functions $\texttt{loc}^+, \texttt{loc}^- : \mathbb{P}_a \to \mathbb{L}$.
- p a painting function $\mathbb{P}_a \times \{s, t\} \to \mathbf{C}$.

Satysfying the following axioms:

- Axioms of plugging: If $e; e' \neq \perp$ then:
  - $\texttt{loc}^+(e; e') = \texttt{loc}^+(e)$
  - $\texttt{loc}^-(e; e') = \texttt{loc}^-(e')$
- Axiom of consistency with respect to location:

$$\texttt{loc}^+(e) \cap \texttt{loc}^-(v) = \emptyset \Longrightarrow e; v = \perp$$

- Axioms of painting: If $e; e' \neq \perp$ then:
  - $p(s)(e; e') = p(s)(e)$
  - $p(t)(e; e') = p(t)(e')$

Define as $L : \mathbb{P}_a \to \mathbb{L}$ the function $\texttt{loc}^+ \bigcup \texttt{loc}^-$ (possible since $\mathbb{L}$ is a boolean algebra).
We extend this function to families by union. It will be used throughout the manuscript to describe locations of programs (which will be families of atomic wires), and we will write L_P for $L(P)$.

Remark
There are three places where the axioms of painting might be placed:

- "Externally", as we chose here. We made this choice because the need for colors, that is, alternation, is a phenomena that doesn't appear in the later seen model of Transcendantal Syntax. We would like to be able to compare the two models one day and putting it externally allows the rest to be closer.
- In locations or something akin, as one can see by the similarity of the axioms of plugging and painting. This could prove convenient because both colors and locations require disjointness hypothesis, they would then be merged. But in a sense colors are sort of "co"-locations: when in between two consecutive edges in a path of colors red and blue, one must be "outside of red and blue", not "in red and blue".
- In the definition of program. This can be quite nice, because we could then avoid the use of $\mathbb{P}_a$(bicig) which is a sort of "ad-hoc" and unelegant extension of $\mathbb{P}_a(IG)$. There would then be a need of extending the notion of execution to define a new painting for the resulting program, and add side conditions on requiring disjoint painting in some theorems.

It is very unclear to me as of today which choice is the most elegant.

Convention
We will often put alternation under the rug in the rest of the dissertation for that very reason (for example, confound the bicolored version of a model of computation with its "natural" counterpart).
I hope it is clear that it is but a technicality but that adapting the definition would be easy to make.

Example
We illustrate the consistency axioms:
In interaction graphs, it is expected that $t(e_0) = s(e_1)$ to be able to compose $1 \xrightarrow{e_0} 2$ and $2 \xrightarrow{e_1} 3$, if this happens we have $s(e_0; e_1) = 1 = s(e_0)$.

Definition 148 (Locative model of computation of bicig)
We define the locative model of computation of bicig:

- $\mathbb{P}_a := ((\mathbb{N} \sqcup \mathbf{C})^2 \sqcup \{\bot\}, ;, \bot)$
- $\mathbb{L} = \mathcal{P}(\mathbb{N})$
- Functions $\mathtt{loc}^+ \colon (n, c, m, c') \to \{n\}$, $\mathtt{loc}^- \colon (n, c, m, c') \to \{m\}$ and $\mathtt{loc}^+(\bot) = \mathtt{loc}^-(\bot) = \emptyset$.
- Painting function p defined by $s, (n, c, m, c') \to c$ and $t, (n, c, m, c') \to c'$.

Proposition
The data $\mathbb{L}, ((\mathbb{P}_a, ;, \bot), \mathtt{in}, \mathtt{out}, p)$ of the previous definition is indeed a locative model of computation.

Proof
We verify the axioms:

- $\mathtt{loc}^+(n \to m; m \to u) = \{n\} = \mathtt{loc}^+(n \to m)$
- $\mathtt{loc}^+(n \to m; m \to u) = \{u\} = \mathtt{loc}^+(m \to u)$

And $n \to m; u \to v = \bot$ when $m \neq u$.
The axioms of painting are verified in the same way. □

Now that we have defined a notion of model of computation, we can define an abstract notion of program, and reason on it.

Convention
We now assume given a locative model of computation:
$\mathbb{L}$, $\mathbb{P}_a$, $(\mathtt{loc}^-, \mathtt{loc}^+)$, p.

Definition 151
A program P in a locative model of computation is given by:

- A location $L \in \mathbb{L}$
- An indexed family $P : |P| \to \mathbb{P}_a$ (up to reindexing, and the presence of $\perp$), where $|P|$ is a set of indexes. To insist on the fact that we use indices, we will use an array notation $P[i]$ instead of function notation $P(i)$.

With the property that $L_P \subseteq L$. The set of programs will be written $\mathbb{P}$.

Convention
We will use the notion of (quotiented) indexed family. We will sometimes write such a family as a multiset [...] when we do not want to specify the indexes.
We will also use an additive convention on families, that is to say $P + Q$ is the family obtained by doing the disjoint union of P and Q, and 0 is the empty family.

Remark
We could also do a definition where we exclude $\perp$ from the family instead of quotienting but it would be a little bit more restrictive.

Hole
We do a quick analogy with lambda-calculus here. The programs we consider are all β-normal. What GoI is traditionnaly interested in is the interaction *between programs*, not of a program "with itself" in its internal workings.
It could be interesting to define an alternative notion of model of computation, where there can be non β-normal programs.
Such a definition should have two sets of locations, L_∂ and L_o, one representing the locations that are used to communicate with the outside (so, "on the boundary"), and the other the locations used internally to represent internal reductions.
Of course there is a need to avoid clashes of locations, so the usual L_G should be the disjoin union of the two.
Such a definition could be convenient when it comes to "programming" with the model of computation.

We can now define the dynamics of our programs and the execution formula. This will be done using the abstract notion of *diagram*.

3.2. Dynamics through diagrams

In this section, we will define the notion of *diagram*, that will be used to define the dynamics of some other models of linear realisability that we will consider in this chapter.

It is important to note that, this notion is static locatively. As such, they are not general enough to define the behaviours of more expressive models (in particular, the latter defined `Flows`), whose location will not only be dynamic, but not even local (they will depend on computations done possibly far away).

Beware
We will define here a notion, the notion of diagram.
A program is made of a family of wires, that can be composed together.
A diagram is a natural extension of wires that is "compositional": in our particular case, it is just a line. For interaction graph, that would correspond to the notion of paths.
Thus, this quite abstract section might seem a bit like "too much work for what it is worth". There are two reasons for doing this definition:

- It allows to abstract away some proofs that are similar in the following models of computation.
- It can be compared with the notion of diagram that is defined for the model of computation used in Transcendantal Syntax. There, the atomic wires are much "wilder" than wires with one input and one output, since the atoms are hypegraphs, and thus the diagrams made by composing them are combinatorially much more complex.
  But this section was made so that there is as little use as possible of the structure of the linearity of diagrams inside the proofs, so that they can be used almost verbatim in this hypergraph contex.
  This is because I have hope that given enough time I can unify the notion defined here with the notion used in TS, and for that we need to abstract them both to compare them.

When doing this step of abstraction, I had in mind the idea that these models look a lot like open systems, with inputs and outputs that can be plugged together, and that have a sort of underlying geometry: they connect together by getting glued on their boundaries.

Convention
We now assume given a locative model of computation $\mathbb{L}, \mathbb{P}_a, \mathtt{loc}^-, \mathtt{loc}^+$.

Definition 157 (Partiality)
Given a set S, define $S^* := S \sqcup \{\oslash\}$, adding an element $\oslash$.
This element will be used as a case "undefined" when defining partial operations.

Definition 158 (Arities of a vertex)
In a graph $G = (V, E, s, t)$, a vertex is said to have:

- $+$-arity the integer $+ar(v) := |\{e \in E \mid v = t(e)\}|$
- $-$-arity the integer $-ar(v) := |\{e \in E \mid v = s(e)\}|$
- arity, the integer $ar(v) := +ar(v) + -ar(v)$

Definition 159 (Wire-Network)
A graph $G = (V, E, s, t)$ is said to have the wire network property when for all v in V, $1 \leqslant ar(v)$ and $+ar(v), -ar(v) \leqslant 1$.

This is easily seen to be equivalent to be made of linear graphs (that is of the shape $\bullet \to \ldots \to \bullet$) as connected components. Note this also requires all these connected components to at least have one edge.
We say that $v \in V$ is in:

- The interior of G when $ar(v) = 2$
- The boundary of G when $ar(v) = 1$

Definition 160 (Diagram)
A diagram δ on a program P is given by:

- A graph $|\delta|$ with the wire-network property, called *the shape* of the diagram
- A function (also written) $\delta : E_{|\delta|} \to |P|$, selecting (through indexes) an atomic wire of $\mathbb{P}$ to be "above" every edge.

We will often identify an atomic wire $f \in \mathbb{P}_a$ with the one edge diagram with f above the only edge.

Example (An example for interaction graph)
Pictorially, one should see the atomic-wires as living above the edges of the graph. For example, take an interaction graph $G = (\{1, 2, 3\}, E, s, t)$ with $E = \{e_0 : 1 \to 2, e_1 : 1 \to 2, e_2 : 2 \to 3\}$, this is a diagram for G:

$$a \xrightarrow{e_0} b \xrightarrow{e_1} c \xrightarrow{e_2} d$$

The graph represented here, with vertex $\{a, b, c, d\}$ is the shape, with the data of the atomic wires associated to each edge written above.
Note that we don't care yet that the arrows cannot be composed.

Because diagrams have the wire-network properties, there is at most 2 edges touching a given vertex, which motivates the following definition:

Definition 162 (In and Out)
Given a diagram δ, for a vertex $v \in V_{|\delta|}$, define:

- $\mathtt{e}^-(v) \in E^*$ the only edge such that $v = s(e)$ if it exists, $\oslash$ otherwise
- $\mathtt{e}^+(v) \in E^*$ the only edge such that $v = t(e)$ if it exists, $\oslash$ otherwise

Convention
We will often use the notation $\mathtt{e}^\bullet$ to designate any of the two functions indiscriminately. We will do the same for other similar notations, such as $\mathtt{loc}^\bullet$.

Definition 164 (The `loc` functions)
Similarly, there are at most two locations above a given vertex.
Given a diagram δ with $|\delta| = (V, E, s, t, p)$, we define two functions:

- $\texttt{loc}^+_\delta : V \to \mathbb{L}^*$ as:

$$v \in V \to \begin{cases} \texttt{loc}^+(P[\delta(\texttt{e}^+(v))]) & \text{if } \texttt{e}^+(v) \neq \oslash \\ \oslash & \text{otherwise} \end{cases}$$

- $\texttt{loc}^-_\delta : V \to \mathbb{L}^*$ as:

$$v \in V \to \begin{cases} \texttt{loc}^-(P[\delta(\texttt{e}^-(v))]) & \text{if } \texttt{e}^-(v) \neq \oslash \\ \oslash & \text{otherwise} \end{cases}$$

Remark
Note since $ar(v) \geqslant 1$, we can't have $\texttt{loc}^+(v) = \texttt{loc}^-(v) = \oslash$.

As stated earlier, a diagram can be seen as an "attempt" at computing a path in the graph. It can thus be reduced to said path:

Definition 166 (Retraction/Actualisation)
Given a connected diagram δ, it has to be of the shape $\bullet \xrightarrow{i_0} \dots \xrightarrow{i_n} \bullet$.
We define an operation of retraction $\Downarrow$ from connected diagrams to atomic wires: $\Downarrow \delta := P[i_0]; \dots ; P[i_n]$.
More generally, given a set S of diagrams, we define *the family* (written as a multiset) $\Downarrow S := [\Downarrow s \mid s \in S]$

Note
The original name of actualisation was chosen by Eng and Seiller, but here we chose a different name to allude to the concept of deformation retract. In Transcendantal Syntax, the corresponding concept is an "attempt" to retract a space on a single point (which would make the space contractible)

Definition 168 (Compatibility)
A connected diagram δ is said to be *compatible* when $\Downarrow \delta \neq \bot$.

Remark
This definition of compatibility is dynamic, in the sense that we check whether there was an error while computing the path.
There are often other ways to characterize compatible diagrams (for example, in the case of interaction graph, it can be forced statically by enforcing $t(e_i) = t(e_{i+1})$ in a path).

To express the fact that paths must stay inside the cut, except on the endpoints who lies outside the cut, we will need to define a notion of interior and boundary of a diagram.
This seems to hint that a diagram is somehow a "topological" object/diagrams are a form of "open system", even if they are described discretely. We give definitions to talk about diagrams with this intuition in mind:

Definition 170 (Boundaries of a Diagram)
Given a diagram δ:

- Its source boundary is defined as the set:
  $(\partial\delta)^- := \{v \in V_\delta \mid \mathtt{loc}^+(v) = \oslash\}$.
- Its target boundary is defined as the set:
  $(\partial\delta)^+ := \{v \in V_\delta \mid \mathtt{loc}^-(v) = \oslash\}$.
- Its boundary $\partial\delta$ is the disjoint union $(\partial\delta)^- \sqcup (\partial\delta)^+$. (It is disjoint since $ar(v) \geqslant 1$).

Now, these notions are defined on the shape of the diagram. There are equivalent notions for the locations at the shape, which are the actually important definitions:

Definition 171 (Located Boundary)
Given a diagram δ:

- Its located source boundary is defined as:
  $(\partial\delta)^-_{loc} \in \mathbb{L} := \bigcup_{v \in (\partial\delta)^-} \mathtt{loc}^+(v)$
- Its located target boundary is defined as the set:
  $(\partial\delta)^+_{loc} \in \mathbb{L} := \bigcup_{v \in (\partial\delta)^+} \mathtt{loc}^-(v)$
- Its located boundary $(\partial\delta)_{loc}$ is defined as $(\partial\delta)^-_{loc} \bigcup (\partial\delta)^+_{loc}$

Definition 172 (Interior of a Diagram)
Given a diagram δ, we define its interior as the set $\delta^\circ := V - \partial(\delta) = \left\{ v \in V_\delta \;\middle|\; \bullet \xrightarrow{e} v \xrightarrow{e'} \bullet \right\}$.

Definition 173 (Restriction of $\mathtt{loc}$)
On the set δ°, both functions $\mathtt{loc}^+, \mathtt{loc}^-$ are total (never any $\oslash$).
In this context we define $\mathtt{loc}_\delta \colon V \to \mathbb{L}$ the function $v \to \mathtt{loc}^-(v) \cap \mathtt{loc}^+(v)$.
The δ will be dropped from indices when it is clear from context.

Proposition (No empty location)
Let δ be a diagram, $v \in \delta^\circ$, $\mathtt{loc}(v) = \emptyset \implies \Downarrow \delta = \bot$.

Proof
In the composition $P[e_1]; \dots P[e_n]$ of $\Downarrow \delta$ is the term $P[\delta(\mathtt{e}^+(v))]; P[\delta(\mathtt{e}^-(v))]$.
But $\mathtt{loc}(v) = \mathtt{loc}^+(v) \cap \mathtt{loc}^-(v) = \mathtt{loc}^+(P[\delta(\mathtt{e}^+(v))]) \cap \mathtt{loc}^-(P[\delta(\mathtt{e}^-(v))]) = \emptyset$.
By the consistency with locations axioms, $P[\delta(\mathtt{e}^+(v))]; P[\delta(\mathtt{e}^-(v))] = \bot$ and since $\bot$ is absorbing, so is the whole product. □

Definition 175 (Locative Interior)
Given a diagram δ, define its located interior as $(\delta^\circ)_{loc} \in \mathbb{L} := \bigcup_{v \in \delta^\circ} \mathtt{loc}(v)$.

Definition 176 (Location and Saturation of a diagram)
Given $L \in \mathbb{L}$ a location, a diagram δ is said to be:

- *At* L when $(\delta^{\circ})_{loc} \subseteq L$.
- L-saturated when $(\partial\delta)_{loc} \cap L = \emptyset$, which can be reexpressed as $(\partial\delta)_{loc} \subseteq \overline{L}$

We write $\texttt{Diags}_L(G)$ the set of diagrams of G that are on L and L-saturated.

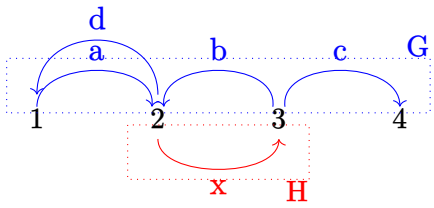
Intuition
Being at L means that the edges we take stay in the cut.
Being L-saturated means that the endpoints are outside of the cut.

Note
Saturation formalises the fact that one cannot compose by an atomic wire from/to L anymore, since two atomic wires with incompatible locations will give $\perp$ when composed.

Definition 179 (Alternated Diagram)
A diagram δ for a program P is said to be alternated when for all e, e' such that $t(e) = s(e')$, $p(t)(P[\delta(e)]) \neq p(s)(P[\delta(e')])$ in G.

Note
Note how, as remarked brefore, in the above definition, the function p could indeed be part of the data of the model, the program, or the system of location.

Example (Counter-examples)
We illustrate these notions with counter examples.
Take the following interaction graphs:



We define $L := \{2, 3\}$ the set where the graphs should interact.
A diagram on it can be:

Not L-saturated :

$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \bullet \xrightarrow{x} \bullet \longrightarrow^{b} \boxed{2 \in L}$$

Not alternated :

$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \boxed{\bullet} \xrightarrow{d} \bullet \xrightarrow{c} \bullet$$

Not compatible :

$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} 3 \neq 1 \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{c} \bullet$$

Definition 182 (Valid Diagrams)
A diagram δ is said to be L-valid whe it is:

- connected
- Alternated
- At L
- L-saturated.

We write $\texttt{VDiags}_L$ for the set of L-valid diagrams.

Definition 183 (Execution)
Given a program (L_P, P), and a set of locations K, define the program $\mathrm{Ex}_K((L_P, P)) := ((L_P \cap \overline{K}), \Downarrow \texttt{VDiags}_K)$. This is a program because the $(\partial \Downarrow \texttt{VDiags}_K)_{loc} \subseteq \overline{K}$ by saturation.
We insist on the fact that $\Downarrow \texttt{VDiags}_K$ is *a family.*
This gives a one to one correspondence between diagrams in $\texttt{VDiags}_K$ that are compatible, and elements of $\Downarrow \texttt{VDiags}_K$.

Remark
Logically, Ex_L amounts to consider L as the set of locations where the cuts are and eliminating them.

Note
It is important to note that we *did not use compatibility* in this definition, because the diagrams that reduces to $\bot$ do not matter, they are quotiented out.
For computational considerations, it is often useful to restrict the computation to compatible diagrams as there might be finitely many (this is formally the definition for the notion of Strong Normalisation)

Definition 186 (Correction)
A diagram δ is said to be correct when it is *valid and compatible.* We write $\texttt{CDiags}_L(G)$ for the subset of $\texttt{Diags}_L(G)$ of correct diagrams of G.

Proposition (Restriction to compatible diagrams)

$$\Downarrow \texttt{VDiags}_K = \Downarrow \texttt{CDiags}_K$$

Remark
This notion of diagram is extremely similar to the notion of diagram discussed in the chapter of TS.

There are two key differences, one being that here we require alternation. The other difference is that these are dual (in the sense of hypergraphs) to the usual notion used in TS. This is discussed further in the appropriate chapter.

Hole
From the previous remark, there is probably a way to abstract this setting even more to encompass the model defined in TS.
There is a need to find a right notion to describe atomic wires as a sort of open system and everything should flow (pun intended) smoothly from that.

Associativity of execution

Now that we have defined an execution, we are going to prove its associativity, which is the main requirement to do realisability theory.

We will now need to prove two lemmas, that shows that diagrams are "observationally equivalent" to their reduced form.

Note
As stated before, alternation will be put under the rug: in the current definition, we exclude non alternated diagrams and diagrams stay alternated when being reduced due to the axioms of painting.

A diagram is always defined given program P. We build out of P a program that can be used as a reference to build more diagrams.

Definition 191 (Closure of a Program)
Given a program P, we define:

- $P_0 := P$
- $P_{n+1} := P; P_n$, with $P; Q := [p; q \mid p \in P, q \in Q]$

Define the closure of the program of all possible compositons $P_\infty := \bigsqcup_{i \in \mathbb{N}} P_i$.

Definition 192 (Partial Reduction)
Given a diagram $\delta = ((V, E, s, t), \delta)$ on P and $v \in V$, define $\Downarrow^v(\delta)$ as the diagram on P_∞:

- $|\Downarrow^v(\delta)| := (V_\delta - v, E - \{\mathsf{e}^-(v), \mathsf{e}^+(v)\} + w, s', t')$ with w a fresh edge, and $s'(w) := s(\mathsf{e}^-(v))$, $t'(w) := t(\mathsf{e}^+(v))$ and equals to s, t otherwise.
- $\Downarrow^v(\delta)(w)$ is the index of $P[\delta(\mathsf{e}^-(v))]; P[\delta(\mathsf{e}^+(v))]$ in $\overline{P}$.

We extend the definition to a set $W \subseteq V$ in the natural way. Note that $\Downarrow \delta = \Downarrow^{\delta^\circ}(\delta)$.

When attempting at reducing a diagram partially, it is often too constraining to really look at the program on which the diagram is. Indeed, we defined it here on P_∞ and thus the reduction cannot natively be compared with the original diagram. This justifies the following definition:

Definition 193
Two diagrams δ on P and γ on Q are said to be equivalent, written $\delta \simeq \gamma$ when they are isomorphic as graphs through a function φ, and $P[\delta(e)] = Q[\delta(\varphi(e))]$ for all e.

Proposition (Unchanged locations)
Let $\delta \simeq \gamma$ through an iso φ.
Then $L_\delta(v) = L_\gamma(\varphi(v))$ for all v.

Definition 195 (Expansion)
Let δ be a diagram on P, γ be a diagram on Q and $e \in E_{|\delta|}$ such that $P[\delta(e)] = \Downarrow \gamma$.
The expansion of δ along $e \to \gamma$, written $\mathtt{Exp}_{e\to\gamma}(\delta)$ is the diagram defined as:

- $\left|\mathtt{Exp}_{e\to\gamma}(\delta)\right| = (V_\delta \bigcup V_\gamma, E_\delta \sqcup E_\gamma - \{e\}, s_\delta \sqcup s_\gamma, t_\delta \sqcup t_\gamma)$ (note the union of vertices isn't disjoint because $s(e) := s(\gamma)$ and similarly for t)
-

$$\left|\mathtt{Exp}_{e\to\gamma}(\delta)\right|(x) = \begin{cases} \delta(x) & \text{if } x \in E_\delta. \\ \gamma(x) & \text{if } x \in E_\gamma \end{cases}$$

More generally, given $S \subseteq E_{|\delta|}$ and $f : S \to \mathtt{Diags}$ such that $f(s)$ is a diagram on Q_s with the property that $\delta(s) = \Downarrow f(s)$, define the expansion of δ along f, written $\mathtt{Exp}_f(\delta)$, by expanding this definition linearly.
In these conditions, $\mathtt{Exp}_f(\delta)$ is a diagram on $P + \sum\limits_{s\in S} Q_s$

Proposition (Expansion-Reduction duality)
Let δ be a diagram on P, γ be a diagram on Q and $e \in E_{|\delta|}$ such that $P[\delta(e)] = \Downarrow \gamma$.
$\Downarrow^{\gamma^\circ} \mathtt{Exp}_{e\to\gamma}(\delta) \simeq \delta$.
More generally, given $S \subseteq E_{|\delta|}$, $f : S \to \mathtt{Diags}$ such that $f(s)$ is a diagram such that $\delta(s) = \Downarrow f(s)$ and letting $V_R := \bigcup\limits_{s\in S} f(s)^\circ$, we have $\Downarrow^{V_R} \mathtt{Exp}_f(\delta) \simeq \delta$.

Proof
We prove it by induction on the size of $\mathrm{dom}(f)$. The inductive case is trivial, we just do the case for one edge:
Looking only at the region around e: e is replaced by a path $f_0, \ldots, f_n$ which is then contracted to $P[f_0]; \ldots; P[f_n]$
But by definition $P[f_0]; \ldots; P[f_n] = \Downarrow \gamma = P[\delta(e)]$.
So the diagrams are isomorphic as graphs, and have exactly the same atomic wires,

so are $\simeq$ □

Proposition (Location stable by contraction)
Let δ be a diagram, $V \subseteq \delta^\circ$, $v \notin V$.
Then $\mathtt{loc}^\bullet_{\Downarrow^{V(\delta)}}(v) = \mathtt{loc}^\bullet_\delta(v)$

Proof
Consequence of the axioms of plugging for wires: $\mathtt{loc}^+(e;e') = \mathtt{loc}^+(e)$, and $\mathtt{loc}^-(e;e') = \mathtt{loc}^-(e')$, so reducing around a vertex doesn't change its location. □

Remark
This is the reason these models I dub "statically locative", their locations are independent of computation.
The main problem with `Flows` is that this is definitely not the case.

Corollary
If $v \in (\partial\delta)^\bullet$ then $v \notin \delta^\circ$ so $\mathtt{loc}^\bullet_{\Downarrow\gamma}(v) = \mathtt{loc}^\bullet_{\Downarrow\gamma^\circ\,\gamma}(v) = \mathtt{loc}^\bullet_\gamma(v)$

Corollary
$\Downarrow \mathtt{Exp}_f(\delta) = \Downarrow \delta$

Corollary
If δ is correct so is $\mathtt{Exp}_f(\delta)$.

Definition 199
A subdiagram δ of γ a connected diagram on P is a diagram on P whose underlying graph is isomorphic to a subgraph of $|\delta|$, and with the same index of wires above every edge.

Proposition (Subdiagrams keep locations)
Let δ be a diagram, $v \in \delta$, and γ a subdiagram with $v \in \gamma$.
Then $\mathtt{loc}^\bullet_\gamma(v) = \mathtt{loc}^\bullet_\delta(v)$.

Remark
This proposition is actually very important. This means that locations are independent of the context. This is not the case when locations change when plugging wires together, which is the case in `Flows` and is the reason this setting is not powerfull enough to describe them.
It is my hope though that it is abstracted enough so that it can be extended more easily in the future to the setting of `Flows`.

Convention
Remember for $I := |\mathrm{Ex}_K(H)|$, for any $i \in I$ we have $\mathrm{Ex}_K(H)[i] := \Downarrow \mathtt{CDiags}_K[i]$, that is each atomic wire in $\mathrm{Ex}_K(H)$ comes from a specific diagram using $\Downarrow$. By *standard indexation of* $\mathtt{CDiags}_K$ we mean the aforementionned indexation of $\mathtt{CDiags}_K$, that is the function $f : i \in I \to \mathtt{CDiags}_K[i]$.

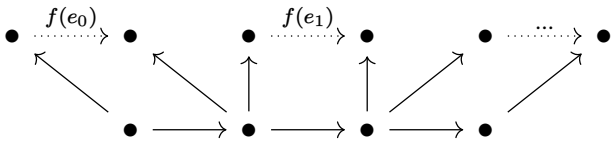
We now state and prove the main theorem of this section:

Theorem (Church-Rosser)
When $L \cap K = \emptyset$, we have the Church-Rosser property:
$\mathrm{Ex}_L(\mathrm{Ex}_K(H)) = \mathrm{Ex}_{L \sqcup K}(H) = \mathrm{Ex}_K(\mathrm{Ex}_L(H))$

The idea is intuitive: we can expand back diagrams that were reduced, or reduce diagram, to go back and forth between $\mathrm{Ex}_L(\mathrm{Ex}_K(H))$ and $\mathrm{Ex}_{L \sqcup K}(H)$, and this does not change the result.

Proof
We prove $\mathrm{Ex}_L(\mathrm{Ex}_K(H)) = \mathrm{Ex}_{L \sqcup K}(H)$. The other case is done symmetrically.
To do this, we compare the valid diagrams: we show there is a bijection φ between the valid diagrams in both families, such that $\Downarrow \delta = \varphi(\delta)$. Thus the executions will be the same.

- Take $\delta = ((V, E, s, t), \delta) \in \mathrm{Ex}_L(\mathrm{Ex}_K(H))$. Let $r : i \in I \to \mathtt{CDiags}_K(H)[i]$ be the standard indexation of $\mathtt{CDiags}_K(H)$. We define $f : e \in E \to r(\delta[e]) \in \mathtt{CDiags}_K(H)[\delta[e]]$, associating to every edge the diagram such that the edge is its retraction.
  We will be interested in $\varphi : \delta \to \mathtt{Exp}_f(\delta)$.
  Injectivitiy comes from proposition 196.
  We now show that is has the correct typing, that is that $\mathtt{Exp}_f(\delta) \in \mathtt{VDiags}_{L \sqcup K}(H)$:



  - It is at $L \sqcup K$:
    Take $v \in \mathtt{Exp}_f(\delta)^\circ$, either:
    * $v \in f(e)^\circ$ for some e and then because $f(e)$ is at K, $\mathtt{loc}(v) \subseteq K \subseteq L \sqcup K$.
    * $v \in (\partial f(\mathtt{e}^-(v)))^-$ and $v \in (\partial f(\mathtt{e}^+(v)))^+$. Let γ be the subdiagram of δ with only $\mathtt{e}^\bullet(v)$, we have:

$$
\begin{aligned}
&\mathtt{loc}^\bullet_{\mathtt{Exp}_f(\delta)}(v) && \\
&= \mathtt{loc}^\bullet_\gamma(v) && \text{By proposition 200 (Subdiagrams keep loc} \\
&= \mathtt{loc}^\bullet_{\Downarrow f(\mathtt{e}^\bullet(v))}(v) && \text{By definition of } f(\mathtt{e}^\bullet(v)) \\
&= \mathtt{loc}^\bullet_\delta(v) && \text{By proposition 200} \\
&\subseteq L && \delta \text{ is at } L
\end{aligned}
$$

  - It is $L \sqcup K$ saturated: let $v \in (\partial\delta)^\bullet$, we have:

*

$$
\begin{aligned}
&\mathtt{loc}^{\bullet}_{\mathtt{Exp}_f(\delta)}(v) && \\
&= \mathtt{loc}^{\bullet}_{\Downarrow \mathtt{Exp}_f(\delta)}(v) && \textit{By corollary 2} \\
&= \mathtt{loc}^{\bullet}_{\Downarrow \delta}(v) && \textit{By corollary 3} \\
&= \mathtt{loc}^{\bullet}_{\delta}(v) && \textit{By corollary 2} \\
&\subseteq \overline{L} && \textit{δ is L-saturated}
\end{aligned}
$$

* Let γ be the subdiagram with only $\mathtt{e}^{\bullet}(v)$, we have:

$$
\begin{aligned}
&\mathtt{loc}^{\bullet}_{\mathtt{Exp}_f(\delta)}(v) && \\
&= \mathtt{loc}^{\bullet}_{\gamma}(v) && \textit{By proposition 200} \\
&= \mathtt{loc}^{\bullet}_{\Downarrow f(\mathtt{e}^{\bullet}(v))}(v) && \textit{By definition of } f(\mathtt{e}^{\bullet}(v)) \\
&= \mathtt{loc}^{\bullet}_{f(\mathtt{e}^{\bullet}(v))}(v) && \textit{By corollary 2} \\
&\subseteq \overline{K} && \textit{$f(\mathtt{e}^{\bullet}(v))$ is K-saturated}
\end{aligned}
$$

Thus the union of the $\mathtt{loc}^{\bullet}_{\mathtt{Exp}_f(\delta)}(v)$ is in $\overline{L} \cap \overline{K} = \overline{L \sqcup K}$ and the diagram is $L \sqcup K$ saturated.

From that we conclude $\mathtt{Exp}_f(\delta) \in \mathtt{VDiags}_{L \sqcup K}(H)$ and such that $\Downarrow \mathtt{Exp}_f(\delta) = \Downarrow \delta$.

- Take a diagram $\delta \in \mathrm{Ex}_{L \sqcup K}(H)$. We show surjectivity.
  First, notice how for every $v \in \delta^{\circ}$, since the diagram is at $L \sqcup K$, we have $\mathtt{loc}(v) \subseteq L \sqcup K$ and thus either $\mathtt{loc}(v) \subseteq K$, or $\mathtt{loc}(v) \subseteq L$. We focus on K:
  Let $V_K := \{v \in \delta^{\circ} \mid \mathtt{loc}(v) \subseteq K\}$, we define for two edges in δ, eRe' if there is $v \in V_K$ such that $e = \mathtt{e}^{-}(v)$ and $e' = \mathtt{e}^{+}(v)$, and $\sim$ to be the reflexive transitive closure of R.
  Take a set $H \subseteq E$ in the partition of E by $\sim$.

  - Such a H induces, by construction, a connected subdiagram of δ that we write $D(H)$
  - We now show that $D(H)$ is a K-diagram:
    * It is at K:
      Let $v \in D(H)^{\circ}$, $\mathtt{loc}(v) \subseteq K$ by construction.
    * It is K-saturated:
      Let $v \in \partial D(H)$, we do a case analysis, either:

- $v \in \partial\delta$, then $\texttt{loc}_{D(H)}(v) = \texttt{loc}_\delta(v)$ by proposition 200, and $\texttt{loc}_\delta(v) \subseteq \overline{L \sqcup K} \subseteq \overline{K}$ (since δ is $L \sqcup K$ saturated)
- $v \in \delta^o$, then if it was such that $\texttt{loc}^\bullet(v) \subseteq K$, we would have $\texttt{loc}^-(v) \subseteq K$ and $\texttt{loc}^+(v) \subseteq K$, so $\texttt{e}^+(v) \sim \texttt{e}^-(v)$, which would both be in H, which contradicts the fact that $v \in \partial D(H)$.
  Thus $\texttt{loc}^\bullet(v) \not\subseteq K$, so it has to be that $\texttt{loc}^\bullet(v) \subseteq L \subseteq \overline{K}$

We thus know that all these sub diagrams are correct K-diagrams, and thus all $\Downarrow D(H) \in \text{Ex}_K(\delta)$.
We now contract all the $D(H)$, and prove that the resulting diagram, $\gamma := \Downarrow^{V_K} \delta$ is an L-diagram:

- It is at L: let $v \in \gamma^o$, $\texttt{loc}^\bullet_\gamma(v) = \texttt{loc}^\bullet_\delta(v)$ since $v \notin V_K$, by proposition 197. But $\texttt{loc}^\bullet_\delta(v) \subseteq L$ since δ is at L.
- It is L-saturated: $(\partial\gamma)_{loc} = (\partial\delta)_{loc}$ by corollary 2, and $(\partial\delta)_{loc} \subseteq \overline{L} \cap \overline{K} \subseteq \overline{L}$.

This proves that $\Downarrow^{V_K} \delta$ is a correct L-diagram using wires in $\text{Ex}_K(\delta)$, so is in $\text{Ex}_L(\text{Ex}_K(\delta))$.
With f defined as before, a simple calculation shows $\texttt{Exp}_f(\Downarrow^{V_K} \delta) = \delta$. (There is a slight technicality here, they are defined on different programs but we can translate the result back on P in a nice enough way that we keep the desired result).
This shows surjectivity. □

Proposition (Independance)
Let G, H be programs and K a location such that $L_G \cap K \cap L_H = \emptyset$.
Then $\text{Ex}_K(G + H) = \text{Ex}_K(G) \sqcup \text{Ex}_K(H)$

Proof
Let δ be a correct diagram in $\text{Ex}_K(G+H)$, then it is either exclusively a G diagram or a H diagram:
Assume it is not the case, without loss of generality, there is a $v \in \delta^o$ such that $\delta[\texttt{e}^-(v)] \in |G|$ and $\delta[\texttt{e}^+(v)] \in |H|$.
But then, since $\texttt{loc}(v) := \texttt{loc}^-(v) \cap \texttt{loc}^+(v) \subseteq L_G \cap L_H$, we have $\texttt{loc}(v) \subseteq L_G \cap L_H \cap K = \emptyset$, so $\texttt{loc}(v) = \emptyset$.
From proposition 174 we get $\Downarrow \delta = \bot$ which contradicts our correctness assumption. □

Proposition (Saturation)
$\text{Ex}_K(\text{Ex}_K(H)) = \text{Ex}_K(H)$

Proof
All correct diagrams in $\text{Ex}_K(\text{Ex}_K(H))$ have only one edge since elements of $\text{Ex}_K(H)$ have locations in $\overline{K}$, and thus make K-saturated diagrams. □

Proposition (Useless Locations can be removed)
When $L_G \cap K = \emptyset$, $\mathrm{Ex}_{L \sqcup K}(G) = \mathrm{Ex}_L(G)$

Proof

- Both programs have the same sets of locations:
  From $L_G \cap K = \emptyset$ we get that $L_G \subseteq \overline{K}$.
  Thus $L_{\mathrm{Ex}_{L \sqcup K}(G)} = L_G \cap \overline{L} \cap \overline{K} = L_G \cap \overline{L} = L_{\mathrm{Ex}_L(G)}$
- We now need to prove $\texttt{VDiags}_{L \sqcup K}(G) = \texttt{VDiags}_L(G)$, then reducing them will give the same multiset.
  For both proofs, notice how $L_G \cap (L \sqcup K) = L_G \cap L + \emptyset = L_G \cap L$.
  Take a diagram in $\texttt{VDiags}_{L \sqcup K}(G)$:
  - It is at $L \sqcup K$: $(\delta^\circ)_{loc} \subseteq L \sqcup K$.
    Obviously, $(\delta^\circ)_{loc} \subseteq L_G$, hence $(\delta^\circ)_{loc} \subseteq L_G \cap (L \sqcup K) = L_G \cap L \subseteq L$, thus the diagram is actually at L.
  - Similarly, it is $L \sqcup K$ saturated: $(\partial\delta)_{loc} \cap (L \sqcup K) = \emptyset$, in particular $(\partial\delta)_{loc} \cap L = \emptyset$ so it is L saturated.

  Take a diagram in $\texttt{VDiags}_L(G)$:
  - $(\delta^\circ)_{loc} \subseteq L \subseteq L \sqcup K$ so it is on $L \sqcup K$
  - It is L saturated: $(\partial\delta)_{loc} \cap L = \emptyset$.
    But $(\partial\delta)_{loc} \subseteq L_G \subseteq \overline{K}$, so $(\partial\delta)_{loc} \cap K = \emptyset$.
    Thus $(\partial\delta)_{loc} \cap (L \sqcup K) = \emptyset \sqcup \emptyset = \emptyset$. It is $L \sqcup K$-saturated. □

We finish with a very useful lemma:

Proposition (Pre-Execution)
Given $K \subseteq L$ locations, G and H programs that are K-disjoint, that is $L_G \cap L_H \cap K = \emptyset$, we can partially pre-execute H: $\mathrm{Ex}_L(G + H) = \mathrm{Ex}_L(G + \mathrm{Ex}_K(H))$

Proof
Let $L' = L \cap \overline{K}$ such that $L = L' \sqcup K$.

$$
\begin{aligned}
&\mathrm{Ex}_L(G + \mathrm{Ex}_K(H)) \\
&= \mathrm{Ex}_{L'}(\mathrm{Ex}_K(G + \mathrm{Ex}_K(H))) && \text{By theorem 203} \\
&= \mathrm{Ex}_{L'}(\mathrm{Ex}_K(G) + \mathrm{Ex}_K(\mathrm{Ex}_K(H))) && \text{By proposition 204} \\
&= \mathrm{Ex}_{L'}(\mathrm{Ex}_K(G) + \mathrm{Ex}_K(H)) && \text{By proposition 205} \\
&= \mathrm{Ex}_{L'}(\mathrm{Ex}_K(G + H)) && \text{By proposition 204} \\
&= \mathrm{Ex}_L(G + H) && \text{By theorem 203}
\end{aligned}
$$

□

Link with the traditional notion of execution

We must now check that this is indeed a generalisation of the traditional notion of execution employed in GoI models, such as interaction graph. We thus give the definition corresponding to the :: operator, and prove its associativity:

Definition 208 (Traditional Execution)
Given programs $(L_{A_i}, A_i)_{i\in I}$, define their traditional execution (also written Ex) as:

$$\mathrm{Ex}((A_i)_{i\in I}) := \mathrm{Ex}_{\overline{\ominus_{i\in I} L_{A_i}}}(\sum_{i\in I} A_i)$$

In particular, $\mathrm{Ex}(A,B) := \mathrm{Ex}_{L_A\cap L_B}(A+B)$

To get back the model of interaction graph, we must consider an interaction graph as a mono-colored bicolored graph (every edge has twice the same color), for example, color a graph G with the color "G".
Now, given G' and H' as G and H colored with "G" and "H", we can define $G :: H := U(\mathrm{Ex}_{L_G\cap L_H}(G' \sqcup H'))$, where U is the function forgetting the colors.

Proposition (Associativity of traditional Execution)
When $L_A \cap L_B \cap L_C = \emptyset$, $\mathrm{Ex}(\mathrm{Ex}(A,B),C) = \mathrm{Ex}(A,B,C) = \mathrm{Ex}(A,\mathrm{Ex}(B,C))$.
With explicit locations that is:

$$\begin{aligned}&\mathrm{Ex}_{\overline{L_A\ominus L_B\ominus L_C}}(A+B+C)\\ &= \mathrm{Ex}_{(L_A\ominus L_B)\cap L_C}(\mathrm{Ex}_{L_A\cap L_B}(A+B)+C)\end{aligned}$$

(The rest follows by symmetry)

Proof
Notice how from the hypothesis we have:

- $L_A\cap L_B + L_A\cap L_C = L_A\cap(L_B+L_C) = L_A\cap((L_B\ominus L_C)+(L_B\cap L_C)) = L_A\cap(L_B\ominus L_C)$
- $(L_A+(L_B\ominus L_C))\cap(L_B\cap L_C)=\emptyset$

$$\begin{aligned}&\mathrm{Ex}_{\overline{L_A\ominus L_B\ominus L_C}}(A,B,C)\\ &= \mathrm{Ex}_{(L_A\cap L_B)+(L_A\cap L_C)+(L_B\cap L_C)}(A+B+C) && \text{Definition}\\ &= \mathrm{Ex}_{(L_A\cap L_B)+(L_A\cap L_C)+(L_B\cap L_C)}(A+\mathrm{Ex}_{L_B\cap L_C}(B+C)) && \text{Pre-execution lemma}\\ &= \mathrm{Ex}_{(L_A\cap L_B)+(L_A\cap L_C)+(L_B\cap L_C)}(A+\mathrm{Ex}(B,C)) && \text{Definition}\\ &= \mathrm{Ex}_{(L_A\cap L_B)+(L_A\cap L_C)}(A+\mathrm{Ex}(B,C)) && \text{Useless location lemma}\\ &= \mathrm{Ex}_{L_A\cap(L_B\ominus L_C)}(A+\mathrm{Ex}(B,C)) && \text{Observation above}\\ &= \mathrm{Ex}(A,\mathrm{Ex}(B,C)) && \text{Definition}\end{aligned}$$

□

Proposition (Commutativity of Execution)
$\mathrm{Ex}(A, B) = \mathrm{Ex}(B, A)$, because the operator involved in the definition are all commutative.

Note
The fact that the operation of execution is commutative and associative shows that this operation is really geometrical in nature: it is about plugging wires together after all...

Hole
The lemmas used to prove this key proposition seem to hint at the fact that diagrams are compositional objects, similar to objects that are found in *open systems*.
The proposition in itself shows that it is comparing the diagrams that is important, not their contracted self.
I think it might be possible to find a better and "higher level" definition of program, where programs would be multisets of diagrams directly.
This would mean having an atomic low-level "model of computation " generating a compositional intermediary-level model of computation, the higher level being the logic describing the behaviours of said programs.

Now that we defined a new setting, we are going to extend it naturally to the additives, which will solve a problem that the usual construction had.

3.3. Slider;Graphs : a new model with dynamic additivity and thickness

Traditionally, additives (and thickness, that is, exponential with additives) are handled via a slice construction which is somehow static ("statically located") in nature: one can define a "big graph" containing all the slices and compute paths inside this graph. The problem with such a method is that there can be leftovers "useless" slices obtained from the computation. They are usually quotiented out afterwards, by showing that the program with a leftover slice is observationally equivalent to the one without.

We define a similar in spirit but different construction that might solve some (although not the main one, that we will refer here by *the problem of the additives*) of the problems of the usual slice constructions. We will directly do the construction corresponding to thick graphs here [51], that is the handling of additives in such a way that they can be compatible with exponentials, with arrows that can go from a slice to another. The purely additive construction, whose arrows stays in their slices, can be deducted from this one.

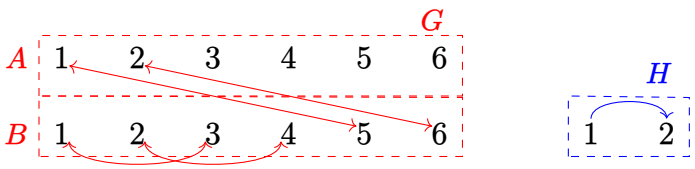
Because the definition might seem quite obutse, we first explain intuitively the idea behind the construction by explaining how they solve the problem that was the motivation for their definition:

What problem do Slider;Graphs solve?

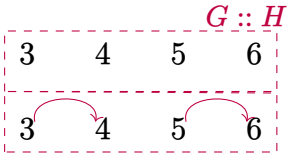
Slider;Graphs do not solve the *"problem of additives"*: by that we mean the fact that GoI models of computation with a local handling of slices indexes suffer from a "leftover" problem that we describe below:
Say we have a cut of a program $p\colon \Gamma \multimap A \,\&\, B$ with a program $q\colon A \,\&\, B \multimap \Delta$, and say this q (which is an "or", so was introduced from either A or B) was obtained from a proof of A. In p, there are two slices, waiting to interact with an A or a B. The slice waiting for B is located on $\Gamma \sqcup B$. The cut is on $A \sqcup B$ so the B part will disappear, but there might be "leftover" information (edges) that were of type $\Gamma \to \Gamma$ after execution.
This is dealt with by Seiller by showing that these leftovers are "invisible" from the point of view of interaction, and thus can be quotiented out when doing logic. [50]
This was also dealt with by Hamano [32] by using a "global" approach to indexes of slices, which allowed to erase dynamically a slice that is not where interaction is happening, but has the disadvantage of becoming very bureaucratic in the handling of indexes.

Our model does not solve this, but another problem that the additive construction of Seiller has: the one of "empty slices".

Consider the exponential contraction program (on the left) against a simple program (right), as defined in [51]:



The slices are represented with dashes. There is thus in the contraction an upper slice A (or slice 1) and lower slice B (or slice 2). Computing the normal form is done by glueing the graphs together, duplicating the graph H to glue it on every slice (theoretically, we do a cartesian product on the slices, to consider all possible case of interaction, but here H has only 1 slice). This normalizes to a graph with an empty slice:

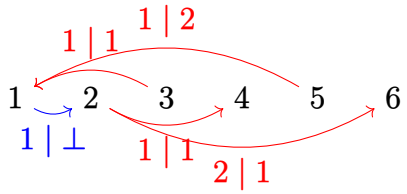


Although it is not relevant to the explanation, we quickly explain the reason G corresponds to the exponential contraction intuitively: notice it transforms $A \,\&\, B$ which has 2 slices into a graph with only one slice which is a tensor, and from the point of view of slice B, it duplicated graph H. This is the kind of behaviour expected of a morphism $!(A \,\&\, B)$ into $!A \otimes !B$, which is the Seely morphism obtained from the exponential contraction (the operator ! collapses slices).

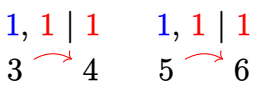
In sliced interaction graphs, a graph is made of slices, and every slice is a graph containing edges.
The idea behind slider;graphs is to reverse this: the relevant data are the edges, we thus have "one big graph" with edges, and for every edge we keep track of the information of which slice it is in. In particular, there can be no "empty slice", since slices only exists when their edges are there. (Note in a sense there are infinitely many empty slices, they are just implied/not part of the explicit data).
We actually need a little bit extra information: in the example above, H was duplicated on top and bottom. The edge of the A of H has thus it's source in slice 1 of H and slice 1 of G.

Moreover, we want to be able to change from slice: some edges have a source in a slice and a target in another: we thus need an additional information to remember where the endpoint of the edge would be.
This works similarly to bi-coloration: there needs to be data for both the source and target of an edge.

We convert the above example informally in such a setting, where we represent only the relevant edges (the edge $1 \to 5$ was not used above), and we put bi-coloration under the rug.
We will write $a \mid b$ to express "this edge starts in slice a", or "one needs to be in slice a to be able to enter this edge", and "this edge ends in slice b", or "one ends up in slice b after taking this edge.



(a) A blue graph against contraction



(b) The result of execution

Any mention of the slice 2 has disappeared: this is because $1|2;2|1$ goes through 2 but forgets it ever went there, and gives an edge that is morally $1|1$.
Notice how our remaining edges are actually $\text{blue}(1), \text{red}(1) \mid \text{red}(1)$. This is because, in the spirit of bicig, we want to define a medium-step semantics, and for that we need to remember a bit more information: we accumulate the "checks" that are done by a path. Here, to take the edge $3 \to 4$ in the resulting graph, one needs to be able to follow the path in the graph on the left, thus be in slice 1 in the blue world and 1 in the red world. This is coherent with the fact that the edge in the "upper copy" of H had it's source in slice 1 in G and 1 in H.

Defining the model of computation of Slider;Graphs

The idea behind a slice is of a "possible world", that is an $A \& B$ expects, to interact, either an A or a B and so has 2 components ready, for each eventuality (possible world); but there is only one real world (At least that we can observe :)), and it will either interact with an A or with a B. As I prefer to keep this intuition in mind when reasoning, we will use the following naming convention instead of slices:

Convention (World Lines)
In this section, we are given a set $\mathscr{W}_l$ called the set of world lines (this is similar to the set of indices to the additive slices). A worldline is an element of $\mathscr{W}_l$, it will represent a possible world, an eventuality.

Remember in a bicig, a color corresponds to an identity, the identity of a program (in previous example, we would color the graph G by the color "G")
Imagining this program as a universe containing several possible world lines (what can be sometimes called parrallel universes, but really is a possible instance of this universe, a sort of "what could have been"), we can imagine that having multiple colors means having multiple universes, each with their own worldlines. Notably, even if we are dealing with a lot of different worldlines, the intuition is that in each universe, at any moment, there is only one "true" worldline.

Convention
As in the other parts, we will assume given a set $\mathbf{C}$ whose elements are called colors.

This motivates the following definition:

Definition 215 (Tracker)
A worldline-tracker, abbreviated as tracker is a partial function $\mathbf{C} \rightharpoonup \mathscr{W}_l$.
They are used to keep track of which worldline we are exploring in each program/universe.

Remark
We used $\mathbf{C}$ to avoid adding an extra set to think about, and because it is intuitive to associate to the "identity of a graph" a certain worldline, but we could use an arbitrary set. This set would represent the possible programs/universes.

Note also that the partiality is important: when a tracker does not pronounce itself on a specific program, it means one could be in any of it's worldlines. This justifies the following definition:

Definition 217 (Coherence of trackers)
We say that two trackers T, T' are coherent on S a set, written $T \frown_S T'$, when $\forall u \in S, T(u) = T'(u)$ *whenever they are both defined.* We will also write this equation $T =_S T'$. In particular we write $T \frown T'$ to say $T \frown_{\mathrm{dom}(T) \cap \mathrm{dom}(T')} T'$.

Note
This is a form of "AND" version of the Kleene equality, which states that they are both equal when either is defined (asking for the other to then be defined).

Intuitively, this means that they agree on the worldlines on the programs they are defined. They do not contradict each other on any observation. We can thus combine their observations together by clumping them together (agglutination):

Definition 219 (Addition of Trackers)
When $T \frown T'$, we define the addition of T and T':

$$T + T' : u \to \begin{cases} T(u) & \text{if } u \in \mathrm{dom}(T) \\ T'(u) & \text{if } u \in \mathrm{dom}(T') \end{cases}$$

(Well defined because of the coherence assumption).

Definition 220 (Indicator Functions)
Take an indicator function $\mathbf{1}_A : \mathscr{W}_l \to \{1, 0\}$ defined by a set $A \subseteq \mathscr{W}_l$ and a tracker T.
We define the following tracker:

$$\mathbf{1}_A.T : u \to \begin{cases} T(u) & \text{if } u \in \mathrm{dom}(T) \cap A \\ \text{undefined otherwise} & \end{cases}$$

Notation
When dealing with a tracker T, we define the notation $1_T = 1_{\mathrm{dom}(T)}$ to avoid writing $\mathrm{dom}(-)$ everywhere.
When using indicator functions, we will also write set using a boolean algebraic notation, that is $\cap = .$, $\cup = +$, and the complement is $\overline{\bullet}$.

Remark
This definition will prove useful because it reduces most reasoning on trackers to

algebraic manipulation.

Lemma (Algebraic Structure of Trackers)
We give some examples of properties of indicator functions and trackers:

- $\mathbf{1}_{\mathrm{dom}(T)}.T = T$
- $\mathbf{1}_A.(T + T') = \mathbf{1}_A.T + \mathbf{1}_A.T'$
- $\mathbf{1}_{A.B} = \mathbf{1}_A.\mathbf{1}_B$

Note
I do not know of any name for the algebraic structure of Trackers and indicator functions, but it is akin to a form of "partial module" with addition defined only when there is coherence.

We now introduce the model of computation that we are interested in:

Definition 225 (A slider bicig)
A *slider bicig* is a bicig with the additional data of a function $T\colon E \times \{s,t\} \to (\mathbf{C} \to \mathscr{W}_l)$ associating two trackers (one at the source, one at the target) to every edge. It will be used to track worldlines along time (composition of arrows).

This definition implies that an atomic wire e will be an atomic wire from bicig with the additional data of two trackers $T(e,s)$ and $T(e,t)$.

Notation (Guarding and Sliding trackers)
We will write G_i when consider a trackers of $T(-,s)$, because they will be used as "guardians".
They represent the information of where the edge is coming from, in what worldlines is its source. It can therefore be seen as a test to pass, that checks whether it is consistent to take an arrow or not when following a path (we might be in the wrong worldline when trying to take this edge and then the composition should fail and the diagram reduce to $\bot$).
We will write S_i for trackers of $T(-,t)$, because they will be used to "slide" from the current worldline (going from a slice to another).
These represent the worldline where the end of the arrow is, that is, in which worldline we end up after taking an arrow. In particular, it might be different from the worldline we were at before, because we mimick the construction of "thick graphs", where edges can go from a slice to another.
When composing edges, these two types of trackers will compose in slightly different ways, but both of their composition will be a special case of a general composition that we will define below.

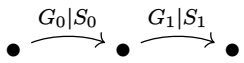
Intuition (How to take an edge)
The idea behind these definition is that when following a path inside the graph, we will keep track of the different worldlines that each programs are in, in a linetracker

T_0.
To accept a new edge e, we will require that $T_0 \supset G_0 := T(e, s)$, that is, in the current worldline, we can indeed use e. We will then update T_0 to a T_1, with the data of $S_0 := T(e, t)$, which will allow us to change frome worldline.

Ultimately, what we want to understand is how to compose wires, that is how to reduce a diagram/path to a single edge.

Notation (Arrow)
An atomic wire in this model of computation will be written as $m \xrightarrow{G|S} n$. Because we already know how to compose the locations (the m and n endpoints), we will just write the $G \mid N$ part and explain how this part of the arrows behave.

Definition 229 (Compatible Arrows)
A pair of arrows such as:



Is said to be a *compatible* (or *timeline coherent*) when $\mathbf{1}_{\overline{S_0}}.G_0 + S_0 \supset G_1$.

Intuition
Intuitively, two arrows $G_0 \mid S_0$ and $G_1 \mid S_1$ are compatible when:

- S_0 passes the test of G_1, that is, the lines that S_0 slides us to are lines that are compatible with G_1 (or else we cannot take the second edge since it is not in the right worldline).
- G_0 checked that we were in some lines. Some were overwritten by S_0, but those that were not have to be compatible with those in G_1, if not then one could not pass both one after the other.

Definition 231 (Overriding)
Given G_0, S_0, G_1, we define the partial overriding operation, defined when they come from compatible arrows, as in the previous definition (*i.e.* , $\mathbf{1}_{\overline{S_0}}.G_0 + S_0 \supset G_1$):

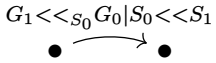
$$G_1 <<_{S_0} G_0 : u \to \begin{cases} G_1(u) & \text{if } u \in \overline{\mathrm{dom}(G_0)} \cap \overline{\mathrm{dom}(S_0)} \\ G_0(u) & \text{otherwise} \end{cases}$$

Algebraically, this is written as $G_1 <<_{S_0} G_0 := G_0 + \mathbf{1}_{\overline{G}_0.\overline{S}_0} G_1$.
We will sometimes drop the index S_0, which should be understood as taking for S_0 the empty tracker with empty $\mathrm{dom}(-)$.

The operation of overriding will be used to compose both guardians and sliders. We will give two different names to their composition to help in context to understand if we are dealing with guardians or sliders.

Lemma
Using the convention notation 221, we have: $\mathrm{dom}(\mathbf{1}_{\overline{S_0}}.G_0 + S_0) = S_0 + \overline{S_0}.G_0$ with the right hand side being a set.

Definition 233 (Composition of Arrows)
When two arrows $G_0 \mid S_0$ and $G_1 \mid S_1$ are compatible, their composition $G_0 \mid S_0; G_1 \mid S_1$ is defined as the arrow:



- The composition $G_1 <<_{S_0} G_0$ is a composition of guardians, and such a compostion will be referred to as *a fortification*.
  Here, this will be read "The fortification of G_0 with G_1 except on S_0".
  The intuition behind this choice is that we create stronger and stronger tests/preconditions to pass the arrow: we test that one can pass G_0 and then G_1 (but there is no need to test the lines in the domain of S_0 since they will be changed, this is already tested by compatibility)
  This is like building a bigger and bigger wall, hence the name.
- The composition $S_0 << S_1$ is a composition of sliders, and such a composition will be referred to as *an overriding*.
  Here, this will be read "The overriding of S_0 by S_1".
  The intuition behind this choice is that we slided to a worldline taking S_0 and then we slided to another place by taking S_1, forgetting in the process that we went somewhere with S_0. Thus S_1 "overrides" some of the information in S_0, hence the name.

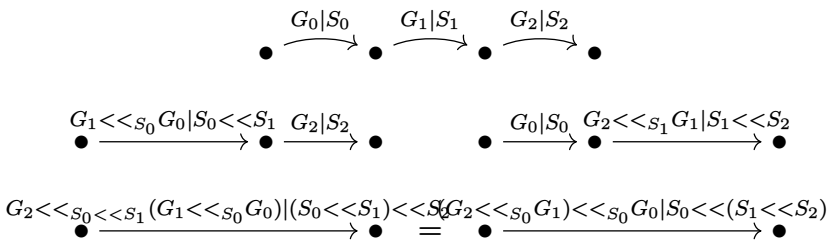
Remark
Note how the composition of guardians is in reverse order compared to the composition of sliders.

Hole
Guardians and Sliders compose in opposite directions: guardians make pre-conditions stronger, more test to pass, and sliders make post-conditions (slighlty, since a lot of information is lost) stronger, more places we can be.
Their composition is a bit reminiscent of lenses in category theory. This would be an interesting link to explore.

Intuition
To slide, one slides along S_0 and then along S_1, which might override what S_0 did.
To guard, one must pass first the test of G_0, then if a worldline is not overridden by S_0 we test it with G_1, explaining why the composition is in reverse.

Consider these 3 arrows, that can be composed two ways. The two possible

compositions give the same result:



Because the equalities of the right hand sides are a particular case of the ones on the left hand side, this can be summed up by the following theorem:

Theorem (Associativity of composition)
Given trackers G_0, G_1, G_2 and S_0, S_1, we have:

$$G_2 <<_{S_0<<S_1} (G_1 <<_{S_0} G_0) = (G_2 <<_{S_1} G_1) <<_{S_0} G_0$$

Proof
This is a really technical proof, because we need to check that both expressions are defined at the same time.
That is, we need to check that $G_2 <<_{S_0<<S_1} (G_1 <<_{S_0} G_0)$ is defined iff $(G_2 <<_{S_0} G_1) <<_{S_0} G_0$ is defined:

$$G_2 <<_{S_0<<S_1} (G_1 <<_{S_0} G_0) \text{ defined iff } \begin{cases} S_0 + \mathbf{1}_{\overline{S_0}}.G_0 \supset G_1 \\ (S_0 << S_1) + \mathbf{1}_{\overline{S_0}.\overline{S_1}}.(G_1 <<_{S_0} G_0) \supset G_2 \end{cases}$$

We can sum it up as!

(1) $S_0 \supset G_1$

(2) $\mathbf{1}_{\overline{S_0}}.G_0 \supset G_1$

(3) $S_1 \supset G_2$

(4) $\mathbf{1}_{\overline{S_1}}.S_0 \supset G_2$

(5) $\mathbf{1}_{\overline{S_0}.\overline{S_1}}.G_0 \supset G_2$

(6) $\mathbf{1}_{\overline{S_0}.\overline{S_1}}.\mathbf{1}_{\overline{G_0}.\overline{S_0}}.G_1 \supset G_2$

Similarly, $(G_2 <<_{S_0} G_1) <<_{S_0} G_0$ is defined when:

(A) $\mathbf{1}_{\overline{S_0}}.G_0 \supset G_1$

(B) $\mathbf{1}_{\overline{S_0}}.G_0 \supset \mathbf{1}_{\overline{S_1 G_1}}.G_2$

(C) $S_0 \supset G_1$

(D) $S_0 \supset \mathbf{1}_{\overline{S_1 G_1}}.G_2$

(E) $\mathbf{1}_{\overline{S_1}}.G_1 \subset G_2$

(F) $S_1 \subset G_2$

We do a double implication to show these conditions are equivalent. The "hard part" being because $\subset$ is not transitive, there is thus a need to look at restrictions of spaces to make $\subset$ an equality and apply transitivity here. We will write $=_S$ to mean trackers are equal on S:

$\rightarrow$ (A) is (2)

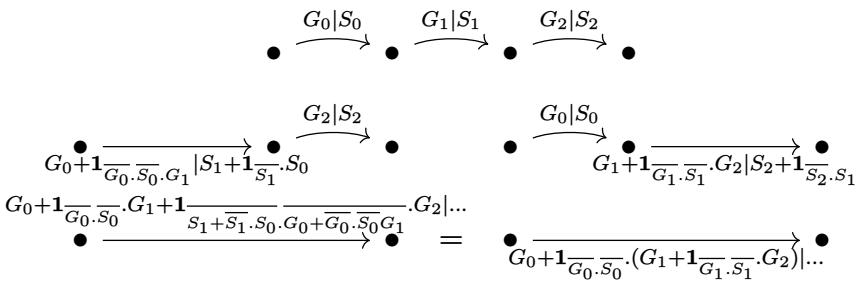
- (B) $\mathbf{1}_{\overline{S_0}}.G_0 \subset \mathbf{1}_{\overline{S_1 G_1}}.G_2$ iff $\mathbf{1}_{\overline{S_0 S_1}}.G_0 \subset \mathbf{1}_{\overline{S_1 G_1}}.G_2$ true by (5)
- (C) is (1).
- (D) $S_0 \subset \mathbf{1}_{\overline{S_1 G_1}}.G_2$ by (4)
- (E) This case is the hard one:
  - First, $G_1 =_{\overline{S}_1 G_1 S_0} S_0 =_{\overline{S}_1 S_0 G_2} G_2$ by (1) then (4). We can do transitivity on the intersection of the condition. Now, on $\overline{S}_0$, there is a case disjunction on G_0.
  - If in $\overline{G}_0$: $G_1 =_{\overline{S}_1 G_1 G_2 \overline{S}_0 \overline{G}_0} G_2$ by (6)
  - If in G_0: $G_1 =_{\overline{S}_1 G_1 G_2 \overline{S}_0 G_0} = G_0 =_{\overline{S}_1 \overline{S}_0} G_2$ by (2) then (5).

  We get the desired result by recombining everything.
- (F) is (3)

$\leftarrow$ (1) is (C)

- (2) is (A)
- (3) is (F)
- (4) We have $S_0 =_{S_O G_2 \overline{S}_1 \overline{G}_1} G_2$ by (D).
  And $S_0 =_{S_0 G_1} G_1 =_{G_1 G_2 \overline{S_1}} G_2$ from (C) and (E). Thus $S_0 =_{S_0 G_2 \overline{S_1} G_1} G_2$
  Hence $S_0 =_{S_0 G_2 \overline{S_1}(G_1 + \overline{G_1})} G_2$, which once simplified gives the result.
- (5) Similarly $G_0 =_{G_0 G_2 \overline{S}_0 \overline{S}_1 \overline{G}_1} G_2$ from (B)
  And $G_0 =_{G_0 G_1 \overline{S_0}} G_1 =_{G_1 G_2 \overline{S_1}} G_2$ from (A) and (E).
  Combining the two we get the result.
- (6) $\mathbf{1}_{\overline{S_0}.\overline{S_1}}.\mathbf{1}_{\overline{G_0}.\overline{S_0}}.G_1 \subset G_2$ is a consequence of (E)

We redo the composing of arrows in the algebraic setting:



We simplify the complicated indicator function on the left using boolean algebraic manipulations:

- $\overline{S_1 + \overline{S_1}.S_0} = \overline{S_1}.(S_1 + \overline{S_0}) = \overline{S_1}.\overline{S_0}$
- $\overline{G_0 + \overline{G_0}.\overline{S_0}G_1} = \overline{G_0}.(G_0 + S_0 + \overline{G_1}) = \overline{G_0}.S_0 + \overline{G_0}.\overline{G_1}$

Now, multiplying the two we get: $\overline{S_1}.\overline{S_0}.(\overline{G_0}.S_0 + \overline{G_0}.\overline{G_1}) = \overline{S_1}.\overline{S_0}.\overline{G_0}.\overline{G_1}$, and thus the big indicator function was just $\mathbf{1}_{\overline{S_1}.\overline{S_0}.\overline{G_0}.\overline{G_1}}$.
Finally, we extract the two equalities on the left of the $|$, which gives:

$$G_0 + \mathbf{1}_{\overline{G_0}.\overline{S_0}}.G_1 + \mathbf{1}_{\overline{S_1}.\overline{S_0}.\overline{G_0}.\overline{G_1}}.G_2 = G_0 + \mathbf{1}_{\overline{G_0}.\overline{S_0}}.(G_1 + \mathbf{1}_{\overline{G_1}.\overline{S_1}}.G_2)$$

These are the same by simple algebra. □

As promised, we state the associativity of composition of sliders, obtained as a special case:

Corollary (Associativity of Overwritting)
Given S_0, S_1, S_2:
$S_0 << (S_1 << S_2) = (S_0 << S_1) << S_2$

We give a definition of *timeline compatibility*, which is not strictly necessary but gives a bit of intuition of what is happening:

Definition 238 (Timeline Compatibility)
A sequence of edges $e_0, \cdots e_n$ is said to be *timeline compatible* when the composition $e_0; \ldots; e_{n-1}$ is defined.
In particular (hence the intuition of timeline):
Start from a linetracker T_0 which is total. (Intuitively, this gives a position somewhere, for each world, the worldline has been decided.
Define $T_{i+1} = T_i << T(e_i, t)$ when $T_i \supset T(e_i, s)$ and fail otherwise.
A diagram is *timeline compatible* when there is a T_0 such that this procedure does not fail.

Proposition (Slider;Graphs form a partial semi-group)
We define the set
$\mathbb{P}_a$(Slider;Graphs) as the set of quadruplets (n, G, S, m) with $n, m \in \mathbb{N}$ and G, S trackers, extended with $\bot$.
The composition of two wires $(n, G, S, m); (m, G', S', p)$ is $(n, G_1 <<_{D_0} G_0, D_0 << D_1, p)$ when the two wires are compatible in the sense of definition 229 and $n = m$, and $\bot$ otherwise.
This composition is associative, as proven above.

Definition 240 (Locative model of computation of Slider; Graphs)
We define the locative model of computation of slider graphs as:
$\mathbb{L}, ((\mathbb{P}_a(\text{Slider;Graphs}), ;, \bot), \texttt{loc}^-, \texttt{loc}^+)$

- With $\mathbb{L} := \mathcal{P}(\mathbb{N})$
- With $(\mathbb{P}_a(\text{Slider;Graphs}), ;, \bot)$ as partial semigroup.
- $\texttt{loc}^+(n, G, S, m) = \{n\}, \texttt{loc}^-(n, G, S, m) = \{m\}$.

Proposition
The locative model of computation described above is indeed a locative model of computation.

Proof
We verify that the axioms are satisfied. (They are because we use exactly the same locations as interaction graph).
Take $e = (n, G, S, m)$, $e' = (m', G', S', p)$

- Axioms of plugging: If $e; e' \neq \bot$ then it is equal to $(n, G_1 <<_{D_0} G_0, D_0 << D_1, p)$ and then:
  - $\texttt{loc}^+(n, G_1 <<_{D_0} G_0, D_0 << D_1, p) = \{n\} = \texttt{loc}^+(e)$
  - $\texttt{loc}^-(n, G_1 <<_{D_0} G_0, D_0 << D_1, p) = \{m\} = \texttt{loc}^-(e')$
- Axiom of consistency with respect to location: when $\texttt{loc}^-(e) \cap \texttt{loc}^+(e') = \emptyset$ then $m \neq m'$ so $e; e' = \bot$.

As before, we put colors and alternation under the rug. □

Remark (Correct Diagram)
Note how, even though we reuse the locations of interaction graphs, there are more compositions that go to $\bot$ because sometimes things are not timeline compatible, while in the case of the interaction graph the only things going to $\bot$ are due to the axiom of coherence *w.r.t.* location.
In this case, a diagram for a slider bicig is correct when it is correct in the sense of bicig and timeline compatible.

Remark (Computing explicitely the actualisation)
If we compute explicitely the actualisation of a diagram $\delta = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots v_{n+1}$, with $T(e_i) = G_i \mid S_i$, let $e = e_0; \ldots; e_{n-1}$.
That is, $S := S_0 << S_1 << \ldots$ (the result of the timeline procedure) and $G := G_n <<_{S_{n-1}} G_{n-1} \ldots$ (note how it goes backwards compared to the sliders).
It's actualisation is $\Downarrow \delta := v_0 \xrightarrow{e} v_{n+1}$ and $T(e) = G \mid S$.

Hole
The fact that we reuse the locations of interaction graph makes it that locations do not change when composing, and thus our theory of diagrams apply. It might be possible to make locations out of trackers, but this would be hard, like in `Flows` since then locations would become dynamic/non-local.
It would nonetheless be fairly interesting as this model is really new and might be different from the more traditional models.
But because of this, it might be possible that the current version of Slider;Graphs is not powerfull enough to actually handle the additives. This is something that should be investigated.

Note
The name "Slider;Graph" is an hommage to two sci-fi shows that I love and to what my father used as name for the "Mandela effect" (sliding).

3.4. Term unification and first-order resolution as a location system

In order to encode 2nd order logic, interaction graph were generalized to the model of graphings, where locations become not just points but measured spaces. This is because Seiller does the quantitative case directly, but one can also do the same construction for arbitrary topological spaces.
Checking when two arrows can be composed is done by looking at the intersection of their endpoints (and then composing the restrictions to the intersection).

Unfortunately, arbitrary topological spaces are not something that can be implemented on a computer (it is a sad fact of life).

One possible way to still use the power of topological space is to represent them via terms and unification, and this is the subject of this section.

3.4.1. Recaps on Unification Theory

Note
The next recap (around 5 pages) were taken from the appendix of a paper in the works with my coauthor Eng, which were already present in his PhD, and as they are just recaps on an already well known theory I did not edit them much. My real contributions start to this part starts with the part on seeing locations as space.

Convention
We will use uppercase letters such as X, Y, Z for variables and lowercase letters a, b, c, f, g and h for function symbols.

Definition 248 (Signature)
A *signature* $\mathbb{S} = (V, F, \mathtt{ar})$ consists of a countable set V of variables, a at-most countable set F of function symbols whose arities are given by $\mathtt{ar} : F \to \mathbb{N}$.

Convention
We will sometimes describe a signature as in the following way: "$f : 2, a : 0$" means f is of arity 2 and a of arity 0.

Definition 250 (First-order terms)
The set of (first-order) *terms* $\mathbb{T}(\mathbb{S})$ over a signature $\mathbb{S} = (V, F, \mathtt{ar})$ is inductively defined by the following grammar:

$$\begin{aligned} t, u, v, w ::= & \\ & \mid \quad X \\ & \mid \quad f(t_1, ..., t_n) \\ & \quad \text{for } X \in V, f \in F, \mathtt{ar}(f) = n \end{aligned} \tag{Terms}$$

The signature will be often dropped and implied in the context.

Definition 251 (Set of variables)
The set of variables of a term is defined inductively as follows:

$$\mathtt{vars}(X) = \{X\} \text{ (for any } X \in V) \qquad \mathtt{vars}(f(t_1, ..., t_n)) = \bigcup_{1 \leqslant i \leqslant n} \mathtt{vars}(t_i)$$

Example
Let $\mathbb{S} := (V, F, \mathtt{ar})$ be a signature such that $V = \{X, Y\}$, $F = \{c, f, g\}$ and $\mathtt{ar}(c) = 0$, $\mathtt{ar}(f) = 1$, $\mathtt{ar}(g) = 2$.
We can construct (among others) the following terms, ordered by depth here:

Depth 0 : c, X, Y

Depth 1 : $f(c), f(X), g(c, c), g(X, Y) \ldots$

Depth 2 : $f(g(c,c)), g(f(X), f(f(Y)))\ldots$

Definition 253 (Substitution)
A *substitution* is a function $\theta : V \to \mathbb{TS}$. Substitutions are extended from variables to terms by

$$\theta(f(u_1, ..., u_k)) = f(\theta(u_1), ..., \theta(u_k)).$$

The application $\theta(t)$ of a substitution θ on a term t can also be written $t\theta$. Substitutions will sometimes we explicitly written with a list of associations $\{X_1 \mapsto t_1, ..., X_n \mapsto t_n\}$ and sometimes simply $\{X := t\}$ if there is only one association.
A *renaming* is a substitution α such that $\alpha(X) \in V$ for all $X \in V$ and such that it is bijective. Renamings are thus invertible, with inverse of a renaming α written α^{-1}.

Definition 254 (Composition of Substitution)
From two substitutions θ_1, θ_2, one can construct their composition $\theta_1 \circ \theta_2$ such that $(\theta_1 \circ \theta_2)t = \theta_1(\theta_2(t))$.
The composition is associative [42, Corollary 6].

Example
Let $\theta := \{X \mapsto c, Y \mapsto c\}$ and $\psi := \{X \mapsto f(Y), Y \mapsto g(c,c)\}$. We have:

$$\theta f(X) = f(\theta X) = f(c)$$
$$\psi g(X,Y) = g(\psi X, \psi Y) = g(f(Y), g(c,c))$$

Note how substitutions replace simultaneously and not sequentially, hence there is no clash between the two occurrences of Y. We also have:

$$\begin{aligned}
&(\theta \circ \psi) g(X,Y) \\
&= g((\theta \circ \psi)X, (\theta \circ \psi)Y) \\
&= g((\theta \circ \psi)X, (\theta \circ \psi)Y) \\
&= g(\theta f(Y), \theta g(c,c)) \\
&= g(f(c), g(c,c))
\end{aligned}$$

Definition 256 (Equation and unification problem)
An *equation* is an unordered pair $t \stackrel{?}{=} u$ of terms in $\mathbb{TS}$.
A *unification problem* or simply *problem* is a finite set of equations $\{t_1 \stackrel{?}{=} u_1, ..., t_n \stackrel{?}{=} u_n\}$.
A *solution* or *unifier* for a problem P is a substitution θ such that for all $t \stackrel{?}{=} u \in P$, $\theta t = \theta u$. In this case, we say that the terms t and u are *unifiable* and that θ.

Example
We give two example of problems:

- The problem $\{f(X, f(Y)) \stackrel{?}{=} f(g(c,c), Z)\}$ has for solution $\theta := \{X \mapsto g(c,c), Z \mapsto f(Y)\}$.
- The problems $\{f(X) \stackrel{?}{=} g(c,c)\}$ and $\{X \stackrel{?}{=} f(X)\}$ have no solution.

Definition 258 (Alpha-equivalence)
Two terms t and u are *α-equivalent*, written $t \approx_\alpha u$, if there exists a renaming α such that $t = \alpha(u)$.

Definition 259 (Alpha-unification)
An *α-unifier* for a problem $P = \{t_i \stackrel{?}{=} u_i\}_{1 \leqslant i \leqslant n}$ is a pair (θ, α) of a substitution θ and a renaming α such that θ is a solution for $\{t_i \stackrel{?}{=} \alpha u_i\}_{1 \leqslant i \leqslant n}$. Two terms t and u are *α-unifiable* if there exists an α-unifier for $\{t \stackrel{?}{=} u\}$.

Example
A typical example is the problem $\{X \stackrel{?}{=} f(X)\}$ which has no unifier but has an α-unifier.
A possible α-unifier is (θ, α) :

$$\theta := \{X \mapsto f(c), X' \mapsto c\}$$
$$\alpha := \{X \mapsto X'\}$$

Then:

$$\theta X = f(c)$$
$$\theta\alpha f(X) = \theta f(X') = f(c) = \theta X$$

Lemma (Symmetry of α-unifiability)
Let t and u be two terms. We have that $\{t \stackrel{?}{=} u\}$ has an α-unifier if and only if $\{u \stackrel{?}{=} t\}$ has one.

These definitions define a preorder on terms:

Definition 262 (Generality Preorder on terms)
We define the preorder of *specialisation*: given t, u two terms, we say that u is an instance of t, written $t \preceq u$ when there is a substitution θ such that $t = \theta u$. Note this is also known as the *reverse scott order*, where the bigger we are the most information we have.

Proposition
The relation $\preceq$ defines a preorder.

The definition of α-unification comes from a simplification of Aubert and Bagnol's definition of *matching* [3, Definition 6] itself appearing in Girard's definitions [27,

Section 1.1.2]. However, since *matching* already exists with a different definition in the literature, a different name is chosen. In the proposition below, the two variants are shown to be equivalent.

Proposition
Two terms t_1 and t_2 are α-unifiable if and only if there exists two renamings α_1 and α_2 such that $\alpha_1 t_1$ and $\alpha_2 t_2$ are unifiable and that $\texttt{vars}(\alpha_1 t_1) \cap \texttt{vars}(\alpha_2 t_2) = \emptyset$.

It is also possible to define a relation $\leqslant$ on substitutions instead of terms. This relation is well-known as a preorder in the literature [4, Definition 4.5.1]. Its intuitive meaning is that $\theta \leqslant \psi$ is ψ is "more general" than θ.

Definition 265 (Generality preorder on Substitutions)
Let θ and ψ be two substitutions. We define the relation $\leqslant$ such that $\theta \leqslant \psi$ when there exists a substitution σ such that $\psi = \sigma\theta$.

The problem of deciding if a solution to a given problem P exists is known to be decidable, we will discuss in the following an algorithm to decide it, the Martelli-Montanari unification algorithm [45].

Proposition (`MGU`, the Most General Unificator)
For a unification problem P, there exists a maximal solution *w.r.t.* the preorder $\preceq$ on substitutions, called $\texttt{MGU}(P)$ which is unique up to renaming.

Definition 267 (Solved form)
A unification problem $P = \{t_i \stackrel{?}{=} u_i\}_{1 \leqslant i \leqslant n}$ is in *solved form* if:

- for $1 \leqslant i \leqslant n$, t_i is a variable $X_i \in V$ and
- no X_i appears in the right-hand side of an equation, *i.e.*

$$\{X_i\}_{1 \leqslant i \leqslant n} \cap \bigcup_{i=1}^{n} \texttt{vars}(u_i) = \emptyset.$$

The *underlying substitution* of P is defined by $\vec{P} := \{X_i \mapsto t_i\}_{1 \leqslant i \leqslant n} = \texttt{MGU}(P)$ (up to renaming).

Definition 268 (Rules for Martelli-Montanari algorithm)
The Martelli-Montanari algorithm is defined by a binary relation $\rightsquigarrow$ (in infix notation) over pairs of a unification problem and a. It is defined as follows:

Clear $P \cup \{t \stackrel{?}{=} t\} \stackrel{c}{\rightsquigarrow} P$;

Open $P \cup \{f(t_1, ..., t_n) \stackrel{?}{=} g(u_1, ..., u_n)\} \stackrel{op}{\rightsquigarrow} P \cup \{t_1 \stackrel{?}{=} u_1, ..., t_n \stackrel{?}{=} u_n\}$ when $f \supset g$;

Orient $P \cup \{t \stackrel{?}{=} X\} \stackrel{or}{\rightsquigarrow} P \cup \{X \stackrel{?}{=} t\}$ with $t \notin \texttt{vars}(t)$;

Replace $P \cup \{X \stackrel{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \{X \mapsto t\}P \cup \{X \stackrel{?}{=} t\}$
with $X \in \texttt{vars}(P)$ and $X \notin \texttt{vars}(t)$.

We simply write $\rightsquigarrow$ when the rule is left implicit and we write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$.

Remark
Note how we could easily transform this algorithm into a variant acting on pairs (P, θ) of a problem and a substitution, where the rule **Replace** would also do the transformation $\theta \rightsquigarrow \theta\{X \mapsto t\}$, deleting the equation $\{X \stackrel{?}{=} t\}$. All other rules would leave θ unchanged.
In this variant, a normal form would have an empty set of equation, and the `MGU` as substitution. It will be used later in the chapter on TS.

Definition 270 (Martelli-Montanari execution)
A (Martelli-Montanari) *(partial) execution* is a non-empty finite sequence of $\rho = (P_1, ..., P_{n+1})$ of unification problems such that $P_i \rightsquigarrow P_{i+1}$ for $1 \leqslant i \leqslant n$. The execution ρ is:

- *successful* if P_{n+1} is in solved form;
- *unsuccessful* if P_{n+1} is not in solved form;
- *full* if there is no P such that $P_{n+1} \rightsquigarrow P$.

Theorem (Correctness and termination of unification algorithm)
There exists a full execution $\rho = (P_1, ..., P_n)$.
P_n is then in solved form if and only if P_1 has a solution. Otherwise, ρ is unsuccessful.

Proof
Proven in Lassez's article [42, Theorem 3.1]. Other proofs are found in Baader and Nipkow's book [4, Lemma 4.6.5 & Lemma 4.6.7 & Lemma 4.6.10]. □

Theorem (Confluence of the unification algorithm)
Let $\rho = (P_1, ..., P_n)$ be an execution starting from a solvable problem P_1. There exists an extension $\rho' = (Q_1, ..., Q_m)$ such that $\rho \cdot \rho' := (P_1, ..., P_n, Q_1, ..., Q_m)$ is successful.

Theorem (UnicityMGU of solution)
If $\rho = (P_1, ..., P_n)$ is a full execution starting from a solvable problem P_1, then $\vec{P}_n$ is the unique solution of P_1 modulo $\approx_\alpha$.

Proof
Indirectly proven in Lassez's article [42, Theorem 3.17]. The original statement says that any solvable problem P has a unique solution modulo $\approx_\alpha$ but the Martelli-Montanari algorithm computes such a solution by correctness of the algorithm. □

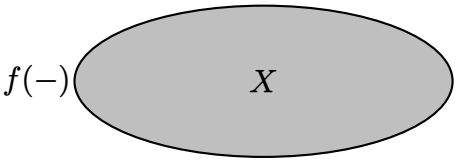
Corollary (Execution lifting)
Let $(P, P_1, ..., P_n)$ be an execution. For all P' such that $P \subseteq P'$, there exists an execution $(P', P'_1, ..., P'_n)$ such that $P_i \subseteq P'_i$ for $1 \leqslant i \leqslant n$.

Proof
By confluence of the unification algorithm (*cf.* theorem 272), all paths of execution ultimately lead to a unique result modulo $\approx_\alpha$. We decompose P' into $P \cup P''$ (since it contains P). It is possible to focus only on the equation in P without impact on the result. We can then construct the execution $P \cup P'' \rightsquigarrow P_1 \cup P'' \rightsquigarrow^* P_n \cup P''$ corresponding to the execution $P' \rightsquigarrow P'_1 \rightsquigarrow^* P'_n$. □

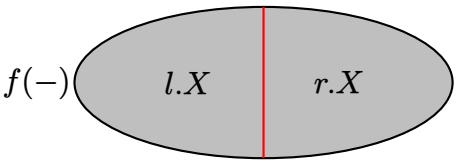
3.4.2. Terms and unification as a system of locations

Now that we've seen what terms are, we quickly explain how they can be thought of (really special, as we want them to form a boolean algebra to do realisability, see Seiller's HDR [58]) topological spaces:

Consider the term $f(X)$, it can be seen as a sort of primitive location in space:



With X being a placeholder indicating that there is a sort of thickness to the space: it can be subdivided, for example $f(l.X)$ is a subspace (we might call it the "left subspace"):



We can now define another poset on terms (with order reversed from the previous definition). We will need to make a boolean algebra out of it, because we need intersections, and to check whether things are inside/outside the cut (complement).

Intuition (Poset of terms seen as spaces)
With the topological space intuition in mind, one can read $t \leqslant u$, that is $t = \theta u$ for a certain θ, as $t \subseteq u$.

Note
This is indeed consistent with the intuition about open sets: knowing that something is in a really small open set is much more information than knowing it is in the entire space.

Definition 276 (Intersection of terms)
We define the intersection of u and v, when there exists a $\theta := \texttt{MGU}(u \stackrel{?}{=} v)$ as $u \wedge v := \theta u$ (note this is the same as θv), which is, the smallest t such that $t \leqslant u, v$.
We adjoin a special term bellow all other terms in $\mathbb{T}$, written $\bot$, and define $u \wedge v = \bot$ when there is no such θ.
It is clearly absorbant for $\wedge$.

Unfortunately, terms do not form a boolean algebra:

Example (Counter examples: terms are not a boolean algebra)
Take as signature $f : 2, g : 1, h : 1, a : 0, b : 0$:

- We had to add an ad-hoc smallest element for intersection (although this is not very problematic in itself)
- The union does not work. The natural union operator $\vee$ is to take the biggest term containing both sides. As a simple example, one would like to have $f(a, y) \vee f(x, b) = f(x, y)$.
  Unfortunately, with this definition $h(a) \vee g(b) = x$, but we have that $h(b) \leqslant x$ so x is too big, it contains more than just the intuitive "union" of the two spaces.
- The complement does not work either: there is no term t such that $t \vee g(b) = x$

We will show how to remedy to this problem.

Downward closed sets

In practice, we do not deal with "atomic" locations, but with sets of locations. We do not want to deal with arbitrary subsets of $\mathcal{P}(\mathbb{T})$ because we still need to have properties such as $f(a) \subseteq f(x)$.
The first intuition one could have to try and solve this problem is to identify $f(x)$ with the downward closed set of elements that are below it, $\downarrow f(x)$:

Definition 278 (Downward closed set)
Given a poset $(\mathbb{T}, \leqslant)$, a downward closed set $S \subseteq \mathcal{P}(\mathbb{T})$ such that if $s \in S$ and $s' \leqslant s$ then $s' \in S$.
We will write $\downarrow\mathbb{T}$ for the set of downward closed sets on $\mathbb{T}$.

Lemma
$\mathbb{T}, \emptyset$ are downward closed.

Definition 280 (The function $\downarrow$)
We define a function $\downarrow : \mathbb{T} \to \downarrow\mathbb{T}$ as $t \to \downarrow t := \{t' \in \mathbb{T} \mid t' \leqslant t\}$.

Definition 281 (Complement)
Given a set of terms L, define $\overline{L} := \{t \mid \forall l \in L, l \not\leqslant t \text{ and } t \not\leqslant l\}$.

Complement is (naively) infinite. It can be expressed in a finite way when the signature is. This is not a problem in practice though, because we never need to compute the actual complements. To check that something is outside the cut L, we just need to check that it is $\in \overline{L}$, so that it is not comparable to any elements of L.

Lemma
$D \subseteq D' \Rightarrow \overline{D} \supseteq \overline{D'}$.

Lemma
$D \subseteq \overline{\overline{D}}$.

Proposition (Stability)
Given two downward closed sets L, L':

- $L \cup L'$ is a downward closed set.
- $L \cap L'$ is a downward closed set.
- $\overline{L}$ is a downward closed set.

Proof
The only interesting case is the last one. Let $t \in \overline{L}$, $u \leqslant t$. If there was a $v \in L$ comparable to u, then either:

$u \leqslant v$ and then $u \in L$ by downward closure bu then t is comparable with an element in L which contradicts $t \in \overline{L}$.

$v \leqslant u$ but then $v \leqslant t$, same contradiction.

Thus u is not comparable with elements of L and is in $\overline{L}$. □

Unfortunately, this definition is not going to give a boolean algebra. Indeed, the complement $\overline{\bullet}$ is not involutive. This is due to the following phenomena:

Example (Counter example to involutivity)
Assume given a signature $f : 1, a : 0$. Consider the downward closed set $S = \downarrow(f(a)) \cup \downarrow(f(f(x)))$, we have that $f(x) \notin S$, but $f(x) \in \overline{\overline{S}}$.
This is because there is nothing else below $f(x)$ but S, thus as we stated before, we should have that as spaces, $f(x) \simeq f(f(x)) \cup f(a)$. In S, we forgot to "close" the set upward.
These two sets are different but should be assimilated

We give some other examples for intuition:

Example
Assume a signature $f:2, a:0, b:0$

- Since $f(a,y) \vee f(x,b) = f(x,y)$ so $\{f(x,y)\}$ should represent the same topological space than $\{f(a,y), f(x,b)\}$.
- $\{\bot\}$ and $\emptyset$ should represent the same space

Take as signature $f:1, a:0, b:0$, we have:

- $\{f(a), f(b), f(f(x))\}$ should represent the same space as $\{f(x)\}$ because of the signature, $f(x)$ can be "maximally" decomposed into 3 incomparable parts.
- Similarly, the whole space x can be seen as decomposed into $f(x), a, b$.

But how can we formalise that these set should be the same ? Notice how S and $\overline{\overline{S}}$ are almost the same, and in particular, representing the term as trees, they have exactly the same leafs (closed terms, that we might call points).

Convention
We assume given a sufficiently rich signature so that we can have closed terms.

We are going to use this remark as a sort of trick to quotient away some sets and get the desired boolean algebra.

3.4.3. Quotienting the right space

We will now quotient our sets by identifying sets having the same underlying ground terms. This will lead us to define a boolean algebra. This will turn out to be $\mathcal{P}(\mathbb{T}^c)$ in disguise. Nonetheless, the representation we define here as equivalence class of downward closed sets might help when considering finite representation of spaces.

Definition 288 (Ground Substitution)
A substitution θ is said to be ground when $\theta(x)$ is a ground/closed term for every $x \in \mathrm{dom}(\theta)$.

Definition 289 (Closure)
Let t be a term, define $t^c := \{\theta t \mid \theta \text{ ground, with } \mathrm{dom}(\theta) = \mathtt{vars}(t)\}$.
Let S be a set of terms, define $S^c := \{t^c \mid t \in S\}$.
We define the equivalence relation $S \simeq S'$ when $S^c = S'^C$

Lemma
$D \cap \overline{D} = \emptyset$

Proof
If $a \in D \cap \overline{D}$ then $a \leqslant a$ contradicts $d \in \overline{D}$. □

Corollary
$D \cap \overline{D}^c = \emptyset$, in particular $D \cap N(D) \simeq \emptyset$

Lemma
If $t \in D$ then there is a term $c(t) \in D^c$ such that $t \geqslant c(t)$.

Proof
If t is closed, $c(t) := t$ works, if not then $c(t) := t\theta$ with θ ground works. □

Lemma
$\overline{D^c} = \overline{D}$

Proof
$\supseteq$: $D^c \subseteq \overline{D}$ so $\overline{D^c} \supseteq \overline{D}$.

$\subseteq$: Let $t \in \overline{D^c}$. It cannot be compared with elements in D^c. Assume it can be compared with $l \in D$:

$t \leqslant l$, then $t \in D$ by downward closure, and $t \geqslant c(t) \in D^c$, impossible.

$l \leqslant t$, then $c(l) \leqslant l \leqslant t$, impossible.

None of the case are possible, so $t \in \overline{D}$ □

Corollary
$(\overline{D^c})^c = (\overline{D})^c$

Definition 293 (Locations)
Define $\mathbb{L} := (\downarrow\mathbb{T})/\sim$ to be the equivalence classes of downward closed sets of terms by the relation $\sim$

Definition 294 (Negation)
We define a function we call negation $N : \mathbb{L} \to \mathbb{L}$ which associates to the equivalence class $L := [D]$ the equivalence class of sets $N(L) := [(\overline{D})]$.

Convention
We dealing with a set $L \in \mathbb{L}$, we will also name L a witness of the equivalence class (a downward closed set).

Proposition (Negation is well defined)
$L^c = L'^c \Rightarrow N(L)^c = N(L')^c$

Proof
$N(L)^c = (\overline{L})^c =^1 (\overline{L^c}^c) = (\overline{L'^c}^c) =^1 (\overline{L'})^c = N(L')$, with 1 by corollary 8. □

Lemma
$A \subseteq B \Rightarrow A^c \subseteq B^c$

Lemma
$(D \cap D')^c = D^c \cap D'^c$

Proof
Let $t \in D^c \cap D'^c$, then $t = u\theta$ with $u \in D$ which is downward closed so $t \in D$.
Similarly $t = v\theta$ with $v \in D'$ so $t \in D'$.
Finally, with θ the empty substitution, $t = t\theta \in (D \cap D')$
The other inclusion is clear. □

Corollary
$\sim$ is stable by $\cap$ since it only depends on the values D^c and D'^c

Lemma
$(D \bigcup D')^c = D^c \bigcup D'^c$

Corollary
$\sim$ is stable by $\cup$ since it only depends on the values D^c and D'^c

Lemma
$(D \cup \overline{D}^c)^c = \mathbb{T}^c$

Proof
The $\subseteq$ inclusion is clear.
Let t be any ground term. Either:

- $t \in \overline{D}$, then $t \in \overline{D}^c$
- $t \notin \overline{D}$, then t can be compared with an element $d \in D$, and either:

  $d \leqslant t$, but then $d = t$ since t is ground, and we fall in the second case.

  $t \leqslant d$, but then $t \in D$ by downward closure.

  In all cases, $t \in \overline{D}^c \bigcup D$, and thus since t is ground, $t \in (\overline{D}^c \bigcup D)^c$. □

Corollary
$D \bigcup N(D) \sim \mathbb{T}$

Theorem
$(\mathbb{L}, \bigcup, \emptyset, \cap, \mathcal{P}(\mathbb{T}), N)$ is a boolean algebra.

Note
Here is a somewhat funny anecdote: I spent an entire afternoon, in the final rush, proving that N was involutive. I then checked the definition of boolean algebra to learn that it is a consequence of the axioms and not an axiom in itself...

Proof
$\sim$ is a congruence from corollary 10 and corollary 9, so all axioms concerning $\bigcup$ and $\cap$ are automatically true since they are for sets.
The two axioms of negation are corollary 11 and corollary 7 □

Hole
There is probably a way to abstract away this construction through the notion of sieve and grothendieck topology in topos theory, as they are used to make formal the notion of "behaving like a topological space".
This is work that I would like to investigate with an expert. I had some preliminary discussion with my coauthor Morgan Rogers which seem to indicate that it might be feasible.

We now show that this algebra ias actually isomorphic to $\mathcal{P}(\mathbb{T}^c)$ all along (this justifies the set theoretical convention for the operators):

Theorem
As boolean algebras, $(\mathbb{L}, \cup, \emptyset, \cap, \mathcal{P}(\mathbb{T}), N)$ and $(\mathcal{P}(\mathbb{T}^c), \cup, \emptyset, \cap, \mathbb{T}^c, \mathbb{T}^c - (\bullet))$ are isomorphic.

Proof
We define the bijection $\varphi \colon \mathbb{L} \to \mathcal{P}(\mathbb{T}^c)$ which associates $[S] \to S^c$ (it is well defined because the invariant under the quotient is precisely S^c).
We also define $\psi \colon \mathcal{P}(\mathbb{T}^c) \to \mathbb{L}$ which associates $X \to [\downarrow X]$.
It is clear that $\psi ; \varphi = id$. The converse is the fact that $S \sim \downarrow(S^c)$ which is clear since they have the same closed terms by definition. Checking that the functions preserve the structure of algebra is easy. □

Convention
Because it is more convenient, we will not use the previously defined notations but use $(\mathbb{L}, \cup, \emptyset, \cap, \mathbb{T}^c, \overline{\bullet})$ for the boolean algebra $\mathcal{P}(\mathbb{T}^c)$ from now on.

We now have a system of location similar to the one used in the revisited version of interaction graph. We quickly finish this section with basic results in "finite representability" of these locations.

Representability of locations

For `Flows`, locations will be in $\mathbb{L}(\texttt{Flows})$, which will be abbreviated to $\mathbb{L}$ in this section.

Definition 306 (The function ι)
We define a function $\iota : \mathbb{T} \to \mathcal{P}(\mathbb{T}^c)$, which associates to t the set t^c.
We extend it by finite union: $\iota \colon \mathcal{P}_{\text{fin}}(\mathbb{T}) \to \mathcal{P}(\mathbb{T}^c)$ which associates to S the set S^c.
Finally, we have the convention that $\iota(\bot) = \emptyset$ (note ι is just the function $-^c$ with a specific typing).

Lemma
We have $\iota(u) \subseteq \iota(v)$ iff $u \leqslant v$.

Proof
The only non-trivial case is the left to right. We prove the first point. If $u \not\leqslant v$ there are two possible cases:

- When $v < u$, then because we assumed a "sufficiently rich" signature, there are sufficiently many branches when representing terms as a tree so that we can ground u in a direction other than v and have $\iota(u) \not\subseteq \iota(v)$.
- When u and v are not comparable, we can do the same.

The only possible case to have $\iota(u) \subseteq \iota(v)$ is $u \leqslant v$. □

Convention
We will now use a multilinear sum notation for terms to define sets of terms.
For example, writing $f(a, g(x)+g(b))$ should be understood as the set $\{f(a, g(x)), f(a, g(b))\}$

One of the point of interest of `Flows` is that they use term to represent in a finite manner locations which are (potentially) infinite sets.

Convention
We now assume that we work in a fixed *finite* signature $\mathbb{S} = (V, F, \mathtt{ar})$.

Definition 310 (Representation)
A location $L \in \mathbb{L}$ is said to be (finitely) representable when there is a finite set $S \subseteq \mathbb{T}$, called a representation of L such that $\iota(S) = L$.

We now show some result of stability:

Proposition (Stability of representation by $\cup$)
Let S be a representation of L, S' of L', then $S \cup S'$ represents $L \cup L'$.

Proposition (Stability of representation by $\cap$)
Let S be a representation of L, S' of L', then $S \wedge S' := \{s \wedge s' \mid s \in S, s' \in S'\}$ represents $L \cap L'$.

Proof

$$\begin{aligned} L \cap L' &= \iota(S) \cap \iota(S') \\ &= \bigcup_{s \in S} \iota(s) \cap \bigcup_{s' \in S'} \iota(s') \\ &= \bigcup_{s \in S, s' \in S'} \iota(s) \cap \iota(s') \end{aligned}$$

The result is because $\iota(s) \cap \iota(s') = \iota(s \wedge s')$. □

Definition 313 (Partition)
A representation S of L is said to be a partition when for $\iota(s) \cap \iota(s') = \emptyset$ for all s, s'.

How can we get partitions of the space in practice? An easy way is to obtain them via splitting:

Definition 314 (Splitting)
We define a function $\texttt{split} \colon \mathbb{T} \to \mathcal{P}_{\text{fin}}(\mathbb{T})$ which associate to a term $t[\vec{x}]$, letting $\vec{z_i}$ be fresh variables, the finite set of terms $t[x_i \leftarrow \sum_{f \in F} f(\vec{z_i})]$.
Let S be a finite set of terms. We define the relation $S \to S'$ when $S' = \texttt{split}(t) + S - \{t\}$ for $t \in S$.
We define $\to^*$ as the transitive closure of this $\to$ relation.

Proposition (Splitting Invariance)
For any $t \in \mathbb{T}$, $\iota(t) = \iota(\texttt{split}(t))$

Corollary (Representation is invariant by splitting)
Let S be a representant of a location K, and $S \to^* S'$ then S' is a representant of K.

Definition 316 (Split-Partition)
Let S be a partition of a location L.
A set S' is a split-partition of S when $S \to^* S'$.

Proposition
For all $u \neq v \in \texttt{split}(t)$, $\iota(u) \cap \iota(v) = \emptyset$, because they are not unifiable (all possible "decisions" of direct subterm are made in the multilinearity and since $u \neq v$ they have to differ on at least one choice).

Corollary (Split-partitions are partitions)
A split-partition is a partition.

Lemma
Let $S = S \sqcup S'$ be a partition of $\mathbb{T}^c$ with S a partition of L, then S' is a partition of $\overline{L}$.

Proposition
Let L be representable, then $\overline{L}$ is.

Proof
Let S such that $L = \iota(S)$.

$$\begin{aligned}\overline{L} &= \overline{\bigcup_{s \in S} \iota(S)} \\ &= \bigcap_{s \in S} \overline{\iota(s)}\end{aligned}$$

We already know that representability is stable by $\cap$, we just need to check that $\overline{\iota(s)}$ is representable.
By splitting the term X until we get s, we get a partition $R \sqcup s$ of $\{X\}$ thus a partition of $\iota(X) = \mathbb{T}^c$. Since $\{s\}$ is a partition of $\iota(s)$, we conclude by the previous lemma that R represents $\overline{\iota(s)}$. □

Convention
Since representability is stable under all operation of $\mathbb{L}(\texttt{Flows})$, we can restrict our setting to the subboolean algebra of representable locations, that we will also write $\mathbb{L}$. Therefore, any location from now one is implicitely assumed to be representable (which in this context is a *finite* notion)
It might be that the subalgebra of representable is just $\mathbb{L}$ but we do not need such a completeness result.

3.5. Flows : a model with a very rich location system

In this section, we will study another model of computation, introduced by Bagnol in his thesis, [7], that of Flows (which is a generalisation of an earlier model by Girard).

We will show how this model compares to the one of interaction graph, and how it can be seen as sort of natural extension of it, akin to a particular case of topological graphing.

The advantage of flows is that they are an *effectively implementable* instance of graphings, using terms to represent the (a priori) infinitary concept of topological space.

3.5.1. Flows

We take a look at the traditional presentation of flows, as seen in Bagnol's PHD [7], but using the vocabulary that we introduced before:

Definition 321 (Atoms : Flows)
A *flow* f is an oriented pair of terms, written as follows: $f : u \to v$. We write $\texttt{Flows}$ the set of flows.
They (atomically) compose by unification: $(u \to v).(v' \to w) = \theta u \to \theta w$ with $\theta = \texttt{MGU}(u = v)$, and $= \bot$ if there is no such MGU.
These represent edges/paths in a proof net, just like edges in interaction graph. We thus introduce notation similar to graphs: $s(f) = u$ and $t(f) = v$.

Definition 322 (Wire semigroup of Atomic Wires)
For flows, the semigroup of atomic wires would $(\texttt{Flows}, ., \bot)$, the functions s and t are as described above.

Definition 323 (Programs : Wiring)
A *wiring* is a (possibly infinite) set of flows. They are written in an additive fashion $f_1 + \cdot + f_n$, and compose by extending . by linearity: $F.G = \{f.g \mid f \in F, g \in G\}$. We will call . the *elementary composition* of wirings. By that we mean it correspond to *one step* of the actual computation.

Remark
Of course, on a computer we only care about finite wirings.

As a model of geometry of interaction, `Flows` have a notion of computation of programs derived from the elementary operation above.

Remark
The model of Flows is really a model of Wirings. Other geometry of interaction models have a similar strange naming convention where they are named after the primitive components of programs (similarly to the lambda calculus), like the *stellar resolution*, which is really about *stars*. While others (interaction graphs) are not (or they would be named interaction edges), and are named after the programs.

Unfortunately, `Flows` do not satisfy the axioms of plugging, indeed:

Example (Counter-example to the axioms of plugging)
Assume a signature $f : 1, a : 0$. We have $s(f(x) \to f(x)) = f(x)$ (or the location associated, $iota(f(x))$), but $s(f(x) \to f(x).f(a) \to f(a)) = f(a) \neq f(x)$.
Even worse, a location in a diagram is completely non local, and can depend on something very far, for example, in:

$$f(x) \to f(x).f(x) \to f(x) \ldots . f(x) \to f(x).f(a) \to f(a)$$

The location of the first $f(x)$ cannot be known before all the execution has been done.

Indeed, locations changes during the computation. The only thing one can expect from the remainding location is that $s(f.f') \subseteq s(f)$.

There are two simple things one might try to salvage the theory behind diagrams, and both fail:

- Add weaker versions of axioms of plugging, such as the proposition $s(f; f') \subseteq s(f)$ described above.
  This does not work, because when expanding an L-saturated diagram, the resulting diagram might not be at L or L-saturated anymore since the locations would now be larger.

- The one might accept the fact that locations are dynamic, and redefine being "at L" and "L-saturated" as being there "once the computation is done". This has the opposite problem: when trying to reduce, we take subdiagrams, but these diagrams are only L-saturated *in context* under such a definition, and might not be saturated anymore when isolated as sub-diagrams (their location depend on the execution of a chain and can come from very far).

Thus, to get associativity of the execution of wirings in an explicit manner, one would need to use the machinery of Graphings [53] (this, in a sense, is expected, since `Flows` are a form of implementable graphings). The idea is that, just like the locations, a flow f can be decomposed into a wiring as a sum of θf, and one can then decompose wiring into parts where L-saturation/being at L are stable/easier to define notions. We quickly give a bit more precision on that:

A bit more details on the decomposition

Given a flow $u \to v$. Let K be a location, we have $\iota(v) = (\iota(v) \cap K) \sqcup (\iota(v) \cap \overline{K})$.
Let U_K be a partition of $\iota(v) \cap K$ and $U_{\overline{K}}$ a partition of $\iota(v) \cap \overline{K}$.
Now take U_K, $l \in \iota(l) \subseteq \iota(U_K) = \iota(v) \cap K \subseteq \iota(v)$, so $l \leqslant v$.
Thus there is a θ_l such that $\theta_l v = l$, and similarly for elements of $U_{\overline{K}}$.
From that we can define:

Definition 327 (Θ_K and $\Theta_{\overline{K}}$)
Given a term v and location K define $\Theta_K(v) := \{\theta_l \mid l \in U_K\}$ and $\Theta_{\overline{K}}(v) := \left\{\theta_l \;\middle|\; l \in U_{\overline{K}}\right\}$.

Definition 328 ($K/\overline{K}$-decomposition of a flow)
Given a flow $f = u \to v$, we define the wiring $f_K = \sum_{\theta \in \Theta_K(v)} \theta f$, and similarly we define $f_{\overline{K}} = \sum_{\theta \in \Theta_{\overline{K}}(v)} \theta f$.
Symmetrically, define ${}_K f = \sum_{\theta \in \Theta_K(u)} \theta f$ and ${}_{\overline{K}} f = \sum_{\theta \in \Theta_{\overline{K}}(u)} \theta f$.

We should then have that $f \sim f_K + f_{\overline{K}}$, and then that $f_K \sim {}_K(f_K) + {}_{\overline{K}}(f_K)$ and similarly for $f_{\overline{K}}$, which gives a decomposition of f into four parts: $\overline{K} \to \overline{K}$ outside the cut, and $K \to K, \overline{K} \to K, K \to \overline{K}$ the inside and "boundaries" of the cut.

There would be a need to prove that the order of decomposition does not change the result, and that in such a setting we can do the execution on things strictly outside the cut etc...As this is quite complex and is work in progress currently, it is not done in this manuscript. The hope of doing this properly, is to be able to adapt it to a generaliaztion of the setting coming in the next chapter (stellar resolution) which seems to be similar but more general than graphings.

Nonetheless, an indrect proof of associativity is done in [7], when he proves that wirings form a "GoI situation" as defined by [31]. We will still use the vocabulary

of diagrams, because the definitions do not require any axioms, it is only the proposition that do.

3.5.2. Static and dynamic locativity

We first define a precondition to check compatibility of an *already computed diagram*:

Definition 329 (Equation and Problem)
Given a diagram δ on a wiring W, a vertex v in δ°, we define the *equation at* v as the equation $\mathbf{eq}(v) := \mathtt{loc}^-(v) \stackrel{?}{=} \mathtt{loc}^+(v)$. This equation is a test, to see whether the atomic wires can be plugged together or not.
This can be extend the definition to any set V immediately to get an unification problem: $\mathbf{eq}(V) := \{\mathbf{eq}(v) \mid v \in V\}$.
Finally, this can be extended to diagrams to create an unification problem called *the problem of the diagram*: $\mathtt{Prob}(\delta) := \mathbf{eq}(\delta^\circ)$

Proposition
δ is compatible iff $\mathbf{eq}(\delta^\circ)$ has a solution.

Remark
Note how here there is a need for *global compatibility* to get a correct diagram, one need to solves *the entire problem of unification.*
In the case of slider;graphs or interaction graph, local compatibility induced the global one.

In this thesis, we will defend the idea that there are two possible presentation of geometry of interaction models linked to location: the static one (where compatibility is checked at compile time, such as in the traditionnal presentation of interaction graph), and the dynamic (or algebraic) one.

Flows are runtime/dynamically located while MLL interaction graph are traditionnaly compile-time/statically located, even though we gave through diagrams a dynamical presentation of interaction graph. What this means is that there are static ways to enforce in **iG** that a diagram does not fail from a mismatch of locations. This is not possible in flow, as checking compatibility is a global problem.

Indeed, as seen in the previous chapter, locating interaction graph to compose them is easy: just place them in the space (on the page) on the right locations. The categorical encoding is also very easy.

For flows, it is another matter, where should one place the target of $a \to f(a, y)$ in a sort of "graph"? $f(a, y)$?

But then how could one compose it with $f(x,b) \to a$ since $f(a,y)$ and $f(x,b)$ are not the same location?

The (real) placement of flows is done at runtime: they will be both be placed at $f(x,y) = f(a,y) \cap f(x,b)$ (which is computed by unification *during execution*) and this is why it may fail and one needs to add $\bot$ as a failure. (This also gives good algebraic properties).

Hole (Compile-time Located Flows)
To have a model of flows with locations at "compile time", one would probably need to use something akin to a virtual double category. Take $\theta = \texttt{MGU}(u \stackrel{?}{=} v)$:

$$\begin{array}{ccc} t & \longrightarrow & u \\ {\scriptstyle\theta}\vdots & & \vdots{\scriptstyle\theta} \\ t\theta & \longrightarrow & u\theta \end{array} \quad \neq \quad \begin{array}{ccc} v & \longrightarrow & w \\ {\scriptstyle\theta}\vdots & & \vdots{\scriptstyle\theta} \\ v\theta & \longrightarrow & w\theta \end{array}$$

If we create such a graph with 2 types of arrows, then the result of execution should be a path $t \to w$ alternating between flows and projections/instantiations. Notice both "commutes" in the sense that one can do all the projections first and then do the composition.
Note also how by precomputing the different `MGU`s, we would keep a finite representation.

3.5.3. The usual notion of dynamics: the execution formula

The traditional notion of execution is usually defined using a certain mathematical formula called the execution formula, which we will mention here for reference. We first define another way to express saturation, by throwing out everything that is not in a cut through projections:

Definition 333 (Projection)
Given a term t, there is a way of turning it into a flow: denote by $t\downarrow$ the flow $t \to t$. We call such a flow a projection.
Given a location L (here a set of closed term), define $\Pi_L := \{l\downarrow | \ l \in L'\}$. (Note to keep thing finite we could use a finite set of "representative" of L).

Remark
Projections can be used to avoid dealing with a boolean algebra: one can filter on the diagrams that are L-saturated by projecting on $\overline{L}$ (but there is no need to consider quotient if one does it this way).

Although, truth is, there are actually two different execution formulas that differ in the form of convergence towards a normal form they capture (strong and weak):

Definition 335 ((Strong) Execution)
We can compute all correct paths and then filter using projections to keep the ones that are "saturated":
$\mathrm{Ex}(F,G)^S := \Pi_{\overline{L}}.(\sum_{n=0}^{\infty}(F.G)^n).\Pi_{\overline{L}}$
That is, whose endpoints lie outside of $\overline{L} = L_F \cap L_G$.

Definition 336 ((Weak) Execution)
We can compute saturated paths directly incrementally:
$\mathrm{Ex}(F,G)^W := \sum_{n=0}^{\infty} \Pi_{\overline{L}}.(F.G)^n.\Pi_{\overline{L}}$

The subtle difference in where the projections are. In the strong case, the projections are done outside of the infinite sum, thus after and so this requires a form of strong convergence: the sum might diverge before we even have a chance of projecting. In contrast, in the weak case the sum might be 0 after a certain index because of the projection and thus converge, and it only requires a form of weak convergence.

We give a simple example to illustrate the difference: $L = \{a\}$ and $F := \{f(x) \to f(x)\}$,$G := \{f(x) \to f(x)\}$.

The sum $\sum_{n=0}^{\infty}(F.G)^n$ does not converge, in the sense that every partial sum gets larger and thus Ex^S is undefined.
On the contrary, $\Pi_{\overline{L}}.(F.G)^n.\Pi_{\overline{L}} = \emptyset$. Thus Ex^W converges to $\emptyset$ (because locations on the boundaries get smaller and smaller, there is not even the need to check more than the first term).

3.6. Abstract Linear Realisability : a recipe for semantic typing

In this section, we will explain how to "recreate logic from computation", that is, we will give a mathematical notion of behaviour of program based on linear logic. I nickname this semantic typing, because of the main theorem that every realisability model has: $p : A \Rightarrow p \in \|A\|$, if p has static type A, then it has semantic type A (it behaves like A).

Most of considerations here can be found in Seiller's HDR [58], but the discussion on higher order types is new.

Convention
In the following, we will consider a set $\mathbb{P}$ (intuitvely seen as a set of programs), representing here a model of computation, such as interaction graph, with localisation in $\mathbb{L}$ (assumed to be at least a boolean algebra).
An element of $\mathbb{P}$ will be of the shape (L,p) with $L \subseteq \mathbb{L}$ and p a program with location L. We write $\mathbb{P}_L$ for the subset of programs located at L, and $p \downarrow L$ to say p is located at L.

Convention
We assume given an abstract notion of execution :: with type $\mathbb{P}_L \times \mathbb{P}_K \to \mathbb{P}_{L \ominus K}$, which is associative.
An example would be the model of computation of interaction graph, and it's execution.
We will assume it is also commutative to avoid complications such as left/right types, but it is *not required.*

Definition 339 (Observation)
An *observation* (also called orthogonality) on an interaction is a (family of) predicate $\bot_L : \mathbb{P}_L \times \mathbb{P}_L \to \{0,1\}$, with L sets of locations, respecting the adjunction property: $p :: q \bot r$ iff $p \bot q :: r$.

We now give a formal definition of model of computation that we will use in this section:

Definition 340 (Formal model of computation)
A formal model of computation, is given by two sets $\mathbb{P}$ of programs, $\mathbb{L}$ of locations, an execution :: that is associative, and an observation (that has the adjunction property).

Definition 341 (Validity Condition)
An *validity condition* (also called pole) is a set of programs $\bot \subseteq \mathbb{P}$ deemed "correct" by the user.
(The name "condition" comes from the fact that we are interested in the predicate $\in \bot$)

Proposition (Generating an Observation from a Validity Condition)
When given a validity condition $\bot$, we can define an observation as $p \bot q := p :: q \in \bot$.

Proof
The adjunction is simple: $p :: q \bot r$ iff $(p :: q) :: r \in \bot$ but $(p :: q) :: r = p :: (q :: r)$ so iff $p :: (q :: r) \in \bot$ iff $p \bot q :: r$. □

To do a computer implementation, there are two things that can go wrong: "::" might not terminate, but this is unavoidable, and $\bot$ might not be decidable (although this is still interesting mathematically, especially strong normalization).
What is interesting is that sometimes, "decidable things" implies "undecidable things", as in a correctness criterion: the absence of loop implies a termination.

We now assume given also assume given an observation.

Definition 343 ((pre-)Behaviour)
A pre-behaviour is a localised set of programs P_L with $L \subseteq \mathbb{L}$. We will often omit the set L of locations, writing just P. In such context, we will denote by L_P the set of locations of P.

From a pre-behaviour P_L, we can define it's set of tests:

$$(P^\perp)_L := \{q \mid q \perp p, \forall p \in P, q \downarrow L\}$$

Notice how it is defined at L as well.
Finally, a behaviour is a pre-behaviour generated from a set of tests, that is a $P = T^\perp$ with T a pre-behaviour.

Proposition (Closure)
A set P is a behaviour iff $P = (P)^{\perp\perp}$.

Corollary
$P^{bot} = P^{\perp\perp\perp}$

3.6.1. General constructions

We define here some usual constructions of logical operators in this setting.

Definition 345 (Subtyping)
We write $A \leqslant: B := B^\perp \subseteq A^\perp$, that is A is a subtype of B.

Remark
Subtyping is a feature that is absent from many logical considerations, and that comes for free in such a setting.

Multiplicatives

Definition 347 (Tensor)
From A_L, B_K, with $L \cap K = \emptyset$.
$(A \otimes B)_{L \sqcup K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$.
We write sometimes $a \otimes b$ to mean $a :: b$ and $L_a \cap L_b = \emptyset$.

Definition 348 (Implication)
From A_L, B_K, with $L \cap K = \emptyset$.
$(A \multimap B)_{L \sqcup K} := \{p \mid p :: a \in B, a \in A\}$.

Proposition (Duality)
$(A \multimap B)$ is equal to $(A \otimes (B^\perp))^\perp$ and thus a behaviour.

Proof

$$
\begin{aligned}
f \in A \multimap B \text{ iff } & \forall a \in A, f :: a \in B && \text{B behaviour} \\
\text{iff } & \forall a \in A, f :: a \in B^{\perp\perp} && \text{Definition of } (B^{\perp})^{\perp} \\
\text{iff } & \forall a \in A, \overline{b} \in B^{\perp}, f :: a \perp \overline{b} && \text{Adjunction} \\
\text{iff } & \forall a \in A, \overline{b} \in B^{\perp}, f \perp a :: \overline{b} && \text{Definition} \\
\text{iff } & f \in (A \otimes B^{\perp})^{\perp} &&
\end{aligned}
$$

□

From this we can define another operator:

Definition 350 (Parr)
From A_L, B_K, with $L \cap K = \emptyset$, we can define $(A \parr B)_{L \sqcup K} := A^{\perp} \multimap B$.

We finish by a proposition which is central to this theory:

Proposition (Generator Lemma)
From a behaviour $A := (S)^{\perp\perp}$, $q \in (A)^{\perp} \iff q \perp s, \forall s \in S$, thus, generators of a behaviour are sufficient tests.

Corollary
$p \in A \otimes B \multimap C$ iff $p :: a \otimes b \in C, \forall a \in A, b \in B$.

Higher Order Types

Convention
In this section we *assume* :: *is cylic* (it is currently the case in most known models, as most have a commutative execution).
That is, we assume that $a :: b \perp c$ iff $a \perp b :: c$ iff $c :: a \perp b$.

Notice how, from the situation we are in, we can define the following behaviour:

Definition 353 (Execution of Behaviour)
From A_L, B_K, define $(A :: B)_{L \ominus K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$
Which is a sort of lifting of :: to behaviour.

Note
Notice how $A :: A \multimap B \leqslant: B$.
Notice also how it is clearly commutative if :: is.
Finally, notice how it is probably impossible to define tests for such a behaviour.
Nonetheless, we continue.

Proposition (Associativity of the lifting)
$(A :: B) :: C = A :: (B :: C)$

Proof
The generator lemma is used implictely almost everywhere and in both directions. The proof is just syntax manipulation:

$$
\begin{aligned}
& q \in ((A :: B) :: C)^{\perp} \\
& \text{iff } q \perp g :: c, \forall g \in A :: B, c \in C && \text{cyclicity} \\
& \text{iff } c :: q \perp g, \forall g \in A :: B, c \in C && \text{adjunction} \\
& \text{iff } c :: q \in (A :: B)^{\perp}, \forall c \in C \\
& \text{iff } c :: q \perp a :: b, \forall a \in A, b \in B, c \in C && \text{cyclicity} \\
& \text{iff } q \perp (a :: b) :: c, \forall a \in A, b \in B, c \in C && \text{associativity} \\
& \text{iff } q \perp a :: (b :: c), \forall a \in A, b \in B, c \in C && \text{adjunction} \\
& \text{iff } q :: a \perp (b :: c), \forall a \in A, b \in B, c \in C \\
& \text{iff } q :: a \in (B :: C), \forall a \in A \\
& \text{iff } q :: a \perp p, \forall a \in A, p \in (B :: C) && \text{adjunction} \\
& \text{iff } q \perp a :: p, \forall a \in A, p \in (B :: C) \\
& \text{iff } q \in (A :: (B :: C))^{\perp}
\end{aligned}
$$

□

Remark
Cyclicity and adjunction are used to unfold the generators of $(A :: B) :: C$ as a behaviour, which suffices to define said behaviour as seen in proposition 351
With the right lemmas to express the generators, this proof might be made much simpler:

$$
\begin{aligned}
& q \in ((A :: B) :: C)^{\perp} \\
& \text{iff } q \perp (a :: b) :: c, \forall a \in A, b \in B, c \in C && \text{assoc} \\
& \text{iff } q \perp a :: (b :: c), \forall a \in A, b \in B, c \in C \\
& \text{iff } q \in (A :: (B :: C))^{\perp}
\end{aligned}
$$

This property is quite strange: it shows that *our behaviours actually form a model of computation.*
We can lift the observation to behaviours as well:

Definition 357 (Observation of Behaviour)
We say that $A \perp B$ with A and B behaviour when $a \perp b, \forall a \in A, b \in B$.

Proposition
$A \perp B$ iff $B \subseteq (A)^{\perp}$.

Proposition (Adjunction of the lifting)
$A :: B \perp C$ iff $A \perp B :: C$

Proof

$$
\begin{aligned}
&(A :: B) \perp C \\
&\text{iff } A :: B \subseteq (C)^{\perp} \\
&\text{iff } (A :: B)^{\perp} \supseteq C \\
&\text{iff } c \perp a :: b, \forall a, b, c \\
&\text{iff } b :: c \perp a, \forall a, b, c \\
&\text{iff } (B :: C)^{\perp} \supseteq A \\
&\text{iff } B :: C \subseteq (A)^{\perp} \\
&\text{iff } A \perp (B :: C)
\end{aligned}
$$

□

We can consider the set of behaviours as our "new set of programs", and do this construction iteratively, which gives us a sort of "type hierarchy":

Definition 360 (Type hierarchy)
We can define:

- $(\mathtt{Type}_0)_L := (\mathbb{P})_L$.
- $\mathtt{Type}_{n+1} := \left\{ A \subseteq \mathtt{Type}_n \;\middle|\; A = A^{\perp\perp} \right\}$.

Also define $\mathtt{Type} := \mathtt{Type}_1$

For now, the layers of the hierarchy are completely independent, but we can easily make them interact:

Definition 361 (Execution Program vs Type)
For $a \in \mathbb{P}$, $B \in \mathtt{Type}$:
$a :: B := (\{a :: b \mid b \in B\})^{\perp\perp}$

Proposition
$a :: B = < a > :: B$ where $< a >$ is $(\{a\})^{\perp\perp}$.

Proof

$$
\begin{aligned}
&f \in (< a > :: B)^{\perp} \\
&\text{iff } f \perp a' :: b, \forall a' \in < a >, b \in B \quad \text{cylic} \\
&\text{iff } b :: f \perp a', \forall a' \in < a >, b \in B \\
&\text{iff } b :: f \in (< a >)^{\perp}, \forall b \in B \\
&\text{iff } b :: f \perp a, \forall b \in B \quad \text{cylic} \\
&\text{iff } f \perp a :: b, \forall b \in B \\
&\text{iff } f \in a :: B
\end{aligned}
$$

□

Remark
This proof is very similar to the lifting of associativity, there might be a way to factorit might be simplified in the same way.

Theorem (Computational Content of Types)
Given a (cylic / commutative) model of computation: $(\mathbb{P}, \mathbb{L}, \bot, ::)$, with $\mathtt{Type}_0 := \mathbb{P}$:

- $(\mathtt{Type}_i, \mathbb{L}, \bot, ::)$ is a (cyclic/commutative) model of computation.
- $(\bigcup_{i \in \mathbb{N}} \mathtt{Type}_i, \mathbb{L}, \bot, ::)$ is a (cyclic/commutative) model of computation.

Hole
It is hard to have any hindsight on one's own work, but I feel like this is might be the biggest "find" I ever did.
This lifting through types does something really unusual, but quite intuitive: say you have a lambda term $t : A \to B$ and another term $u : B$. The term (tu) is usually typed as having type B. There is no dynamics in typing (this is a slight lie, there are dynamics *in practice*, when implementing the theory because in type theory, the definitional equality $\equiv$ has to be implemented somehow), types are just "labels" used to force good behaviours.
But from this discussion here, one could give it the type $A \to B :: A$, introducing dynamics inside the typing itself. After a bit of investigation, this is known in the type theory community as "conflating the Π type (function type) with λ". The only work I could find on that is unfinished work done by Xavier Montillet, a former PhD Student of Guillaume Munch-Maccagnoni, and it is apparently a really complex subject, related to the conjecture on PTS (pure type systems) that weak-normalization might implies strong-normalization.

We now give other definition that can appear in linear realizability, to be a tad bit more exhaustive:

Second Order

The second order construction was already done in Seiller's work, but we show it can be expressed in this framework:

Definition 366 (Universal Quantification)
From a function $P : \mathtt{Type} \to \mathtt{Type}$. $\forall \alpha.P := \bigcap_{\alpha \in \mathtt{Type}} P(\alpha)$.

Definition 367 (Existential Quantification)
From a function $P : \mathtt{Type} \to \mathtt{Type}$. $\exists \alpha.P := \bigcup_{\alpha \in \mathtt{Type}} P(\alpha)$.

Now, the major thing to get from this is the difference between $\multimap$ (syntax) and $\to$ (semantics):

Proposition
If $p : \mathtt{Type} \multimap \mathtt{Type}$ then the function P which to T associates $p :: T$ has meta-theoretical type $\mathtt{Type} \to \mathtt{Type}$.

This kind of distinction might be used to give a status to proofs that a certain type is well-formed (statement of the form $\vdash A : \mathtt{Type}$) in type theory.

Linear Dependent Types

I would like to conclude this section with a discussion on Girard's attempt at Linear Dependent Types (which is not available in English, so this is a first). His proposition and is not actually much more complicated than the definition for linear types:

Definition 369 (Tensor)
From A_L, $B_K : A \to (\mathtt{Type})_K$, with $L \cap K = \emptyset$.
$((a : A) \otimes B(a))_{L \sqcup K} := \{a :: b \mid a \in A, b \in B(a)\}^{\perp\perp}$.

Definition 370 (Implication)
From A_L, $B_K : A \to (\mathtt{Type})_K$, with $L \cap K = \emptyset$.
$(a\colon A \multimap B(a))_{L \sqcup K} := \{p \mid \forall a \in A, p :: a \in B(a)\}$.

Notice that we require $B_K : A \to (\mathtt{Type})_K$, a semantic choice, and not $B_K \in A \multimap (\mathtt{Type})_K$ which is more syntactic in nature.

Proposition (Duality)
$(a\colon A \multimap B(a))_{L \sqcup K}$ is equal to $(((a\colon A) \otimes B(a)^{\perp})_{L \sqcup K})^{\perp}$ and thus a behaviour.

Proof
The proof is exactly the same as before:

$$
\begin{aligned}
f \in a\colon A \multimap B(a) \text{ iff } & \forall a \in A, f :: a \in B(a) && \text{(} B(a) \textit{ behaviour}\\
\text{iff } & \forall a \in A, f :: a \in B(a)^{\perp\perp} && \text{(} \textit{Definition of } (B(a)^{\perp})^{\perp}\\
\text{iff } & \forall a \in A, \overline{b} \in B(a)^{\perp}, f :: a \perp \overline{b} && \text{(} \textit{Adjunction}\\
\text{iff } & \forall a \in A, \overline{b} \in B(a)^{\perp}, f \perp a :: \overline{b} && \text{(} \textit{Definition}\\
\text{iff } & f \in (a\colon A \otimes B(a)^{\perp})^{\perp} && \square
\end{aligned}
$$

I would like to state that Girard sketched a proof of duality that I was never able to formalize because of a step that I conjecture to be faulty. I think his motivations for this were to try and get a form of "commutativity" for the tensor product, even in the dependent case (where commutativity does not really make sense because of the typing). Girard being Girard, he did not prove anything but I try to reconstruct his reasoning here. Letting $F := (a : A) \otimes B(a)$, he defines:

- The pre-behaviour $B := \{f :: a \mid a \in A\}$
- The family $(A(\bar{b}))_{\bar{b} \in B^\perp}$ with behaviours $A(\bar{b}) := \left\{\bar{b} :: f \;\middle|\; f \in F\right\}^\perp$.

Lemma
Given $\bar{b} \in B^\perp$, we have $A(\bar{b})^\perp \subseteq A^\perp$.

Proof
We prove that $A \subseteq A(\bar{b})$:
Let $a \in A, f \in F$, we have by cyclicity (or commutativity and the adjunction):

$$a \perp \bar{b} :: f \text{ iff } f :: a \perp \bar{b}$$

By definition, $f :: a \in B$ thus the right hand side is true. Hence so is the left hand side, which means by definition that $a \in A(\bar{b})$. □

From that Girard states, letting $G := \bar{b} \colon B^\perp \multimap A(\bar{b})^\perp$, that $F = G$. It is easy to prove that $F \subseteq G$. I was not able to prove the converse. From the lemma above, one can conclude that for $g \in G$, $g :: a \in B^{\perp\perp}$. But this does not seem sufficient to conclude that $g :: a \in B(a)$ (I would be interested in a proof if it is). Even in a really good case where $B = B^{\perp\perp}$, I think we can only get $g :: a = f :: a'$ but there is no way to force $a = a'$.
Another way to state this is that one can prove that g is orthogonal to some elements of $F^\perp$, the $rlineb :: a$, but this might not be enough to be orthogonal to all elements of $F^\perp$ and thus be in $F^{\perp\perp} = F$.

From that he would conclude that $a \colon A \otimes B(a) = b \colon B \otimes A(b)$, the problem being that B is just a *pre-behaviour*. He thus states to "iterate his construction" to close it under orthogonality.

There were many attempts to do linear dependent types in the litterature, so I am unsure whether these definitions are satisfactory. Even if they are, finding a syntax for linear dependent types would still be an open problem. There are two possible problems that I can think of, depending on the choice of definition that we care about:

- We can choose $(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A, p :: a \in B(a)\}$, (which was the choice presented above).
  Then the main problem would be how to force that $a = a'$ when trying to combine $\Gamma \vdash a \colon A$ and $\Delta \vdash b \colon B(a')$. I guess this could be a side condition on the rule of tensor?
- We can choose $(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A, p :: a \in B :: a\}$ (this means that $B \colon A \multimap \texttt{Type}$).
  Then the main problem would be how to combine $\Gamma \vdash a \colon A$ and $\Delta, a' \colon A \vdash b \colon B(a')$?

The expected result would be $a \otimes b : a : A \otimes B(a)$, but this would somehow duplicate a since it appears as a program and as an argument for B. The most naive way I could see to solve such a problem is to allow duplication "upwards" (from $\texttt{Type}_i \to \texttt{Type}_j$ with $j > i$). Especially since in many models type do not "really compute".

Hole

The last idea suggests another form of model of computation that I did not define in the higher order case:
It would be interesting to consider programs as (potentially infinite) sequences $a_0 : A_1 : A_2 : \dots$ and execution of sequences done level by level.
A program would thus come with its type above (and its type with its own type and so on...). In such a setting we could then maybe add arrows during execution going from a level to above and not count these as duplications. If such a thing was done, it would look a lot like the treatment of exponentials in thick graphs [51], whose contraction was explained in section 3.3, and used arrows going from a slice to another slice (instead of from a level to another level).

What about finiteness ?

This was just a bunch of abstract mathematical definitions, needed to check whether a model of computation give rise to a model of a logic or not.

But to achieve the goal of real implementation, we will need, in the future, to change our perspective on these definitions: they are "behaviour oriented", that is, the element we really are caring about is the behaviour. But what we would need is to define everything not from a "behaviour B" perspective, but from a "$T^\perp$ with T finite" perspective, especially when trying to do higher order. Notice also how seconder order $P(X)$ depends not really just on X, but on $X^\perp$ as well, which is infinite! This would be problematic! We would need to consider both a finite X (approximation of $(X)^{\perp\perp}$) and finite $\sim X$ (approximation of $(X)^\perp$).

This was attempted by Girard in [30] and similar attempts were made by Ragot, Seiller and Tortora de Falco in [48].

4. Transcendantal Syntax

Transcendantal Syntax is a form of continuation, or "rebranding" of the program of GoI, initiated by Girad around the 2010's.

Like every model of GoI, it is based on a model of computation, called Stellar Resolution.

In this chapter, we will study the computational properties of said model of computation.

Note
This chapter is mostly based on a joined article between Eng, Seiller and me.
A lot of the content is work that was done and thus is already present in Eng's PhD [19] (I thank him for the tikz drawings, in particular, which I just adapted). But as I did a lot of work on revamping definitions and proofs for the sake of the article, I join the result as part of this dissertation. One could say this is "my take" on their model of computation.

4.1. Stellar Resolution

Stellar resolution is a new model of computation introduced as a computational ground for logic. In the previous section, all models were based on the same idea: computing the "paths" inside the body of a program, which allow to see the flow of information (hence the name of the last model we saw)

Here, instead of computing the possible paths inside the programs, we try do directly compute the "network of wires" that the program is.

4.1.1. Stars and constellations

We use Girard's terminology of *stars* and *constellations* [28]. The model is akin to a model of tiles, called *stars*, and flexible arms, called *rays*, and this was a real subject of investigation by Eng in is PHD.

In our approach, stars can be seen as an hypergraph generalisation of `Flows`.
In `Flows`, the atomic wires were edges, with a source and a target. Here, we will

have hyperedges as atomic wires. The combinatorics will not be as easy because there might be more than one source or target. For that purpose, we will introduce polarities $+$ and $-$ that will be added to terms to indicate whether they are in input or output position.

The fact that this model will use hypergraphs will allow to encode naturally models of computation such as the model of computation of Proof Structure defined in the first part of this PhD.

A program will be called a *constellation*. It's atomic wires stars, and the endpoints of stars will be called rays.

Because it is very hard to deal with the dynamics when locations can change during execution, we will use a trick similar to the one that was done in Slider;Graphs: we will use static locations, and the terms will be encoded outside of locations, as part of the dynamics only.

For historical reasons, the name *colors* was chosen for these static locations by Girard, and then Eng. It is probably completely unrelated to the notion of colors used in the previous sections, which was more akin to the identity of a program. The reason the name "color" was chosen is because a location, say 1, can appear as either $+1$ or -1 in the execution, and Girard had the intuition that these two locations are like "dual colors". On top of that, some locations do not interfere in the execution (those are called unppolarised locations, or internal memory), but there are times when one might want to polarize them, and this was called at the time "coloring" the location.

Because of this, we will still use in this manuscript the terminology color.

We give a simple example: say c is a function symbol in a signature, we also call it a color, and then $+c$ and $-c$ are two *dual* colors (for example, green and magenta). We then expect terms such as $+c(f(X))$, $f(X)$, Y, $+d(X,e)$ and $+c(f(f(X,X),Y))$ to be rays.

Definition 375 (Polarity)
The constructions will be parametrized by a set of *polarities* $\mathcal{P}$, equipped with an auto-involution $\bullet \to \overline{\bullet}$. We will use $\mathcal{P} = \{+,-\}$ in the following, with $\overline{+} = -$.
As stated in the introduction, this will give a convenient directed hypergraph representation of things in the model.

Convention
We will sometimes have to deal with reasoning on polarities where the polarity does not matter. We will then use $\bullet$ as a symbol to designate an abstract polarity in $\{+,-\}$.

Hole
In Eng's PHD was evoqued the idea of extending the set of polarity, for example with an autodual polarity.
The drawback would be that this would require to invent a new definition of hypergraph to adapt the definitions. I do not see what the purpose of this would be to encode logic, but this would make a slightly richer model of computation and might thus be interesting to study potentially unseen phenomena elsewhere.

Definition 378 (Colored Signature)
A *colored signature* is a tuple:

$$\mathbb{P} = (S, \mathbb{C})$$

where S is a first order signature, and $\mathbb{C} = (\mathcal{P} \times |\mathbb{C}|) \sqcup \{\oslash\}$ with $|\mathbb{C}|$ a set.
The elements of $|\mathbb{C}|$ are called *colors*, and the elements of $\mathbb{C}$ are called *polarized colors* except for $\oslash$, which will be used to describe "unppolarized" objects.
Hence a *polarized color* is a pair $(\bullet, c)$ with $c \in |\mathbb{C}|$ and $\bullet \in \mathcal{P}$.
For concision, polarized colors will be written as $+c$, $-c$ or even $\bullet c$ instead of pairs.
The *opposite* of a color is defined as $\overline{(\bullet c)} := \bar{\bullet} c$, an operation undefined on $\oslash$.

Remark (Colors)
We insist one last time on the fact that colors in the context of stars are of a different nature than the ones used in models such as interaction graph.
Here, colors are nothing more than "special locations", used to control the flow of execution. What is really new is that they can become polarize. But is is very important to remember that *a set of colors is exactly what a location was in the previous models.*
In interaction graph, colors are used to represent the identity of a graph, then, to force alternation between axioms and cuts, one ask that the colors alternate.
It is unfortunate that such a clash of names exists.

The idea of coloring is akin to a "strategy": it is a way of deciding what rays should be plugged, *i.e.* what should be reduced.

In similar fashion in lambda calculus, one can "color" the redexes that one wants to reduce and only allow reduction on colored redexes to control how normalisation happens (which is similar to the lemma of finite devellopments, in the proof of Church-Rosser by Tait and Martin-Löf)

One way to think about it is of "cables", where one would plugs cables of the same color together. They have inputs/outputs to be plugged, but what really matters is the fact that they are cabled together.

An atomic wire in our model, a star, will be made of mmultiple elements of $\mathbb{C} \times \mathbb{T}$, with the $-$ elements being the equivalent of sources and $+$ elements the equivalent of targets in interaction graphs. The novelty here is that some elements will be *unpolarized* when they are of the shape $(\oslash, t)$.

Definition 380 (Polarized, Unpolarized)
An element of $\mathbb{C} \times \mathbb{T}$ is said to be:

- Unpolarized, when it is of the form $(\oslash, t)$.
- Polarized otherwise.

Definition 381 (Rays, Unpolarized terms)
A *ray* r on a signature $\mathbb{P}$ is a polarized element of $\mathbb{C} \times \mathbb{T}$.
Explicitely, it is a pair (p, t) of a polarized color p and a first order term t on the signature of P.
The *underlying term* of a ray $r = (p, t)$ is defined as $\lfloor r \rfloor := t$
The set of rays on $\mathbb{P}$ is denoted by `Rays`.
When C is a set of colors, we write $r : C$ to say that $r = ((\bullet, c), t)$ with $c \in C$.

Definition 382 (Pluggable Rays)
Two rays $r_i = (c_i, t_i)$ for $i \in \{1, 2\}$ are said to be pluggable, written $r_1 \Subset r_2$ when $c_1 = \overline{c_2}$.

Definition 383 (Compatibility between rays)
Two rays r and r' are *compatible*, also *coherent* written $r \frown r'$, when they are pluggable and $\lfloor r \rfloor$ and $\lfloor r' \rfloor$ are α-unifiable.

Convention
We assume the existence of a colored signature $\mathbb{P} = (S, \mathbb{C})$ unless we explicitly use a specific one.
A priori, colors and function symbols are two different concepts used for different purposes. But it is convenient to not have a lot of different set of symbols, and thus we will also assume $|\mathbb{C}| = F$ in most examples, which allows the abusive notation $+c(t_1, \cdots, t_n)$ for $((+, c), c(t_1, \cdots, t_n))$.

Example (Coherences)

$$
\begin{aligned}
&+ c(X) \frown -c(0) \\
&- d(X) \frown +d(f(X)) \\
&+ c(X) \not\frown -d(X) \text{ (different colors/head symbol)} \\
&+ c(X) \not\frown +c(f(Y)) \text{ (polarities are not opposite)} \\
&+ c(f(X)) \not\frown -c(g(Y)) \text{ (terms are not } \alpha\text{-unifiable)}
\end{aligned}
$$

Proposition
The relation $\frown$ is symmetric but anti-reflexive and anti-transitive.

Proof
Symmetry follows from the symmetry of α-unifiability, the rest from having same polarity. □

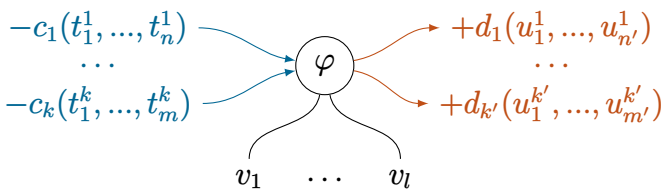


Figure 4.1: Star with rays seen as either input, output along with its internal memory

We can now define the atomic (hyper)wires of our model of computation:

Definition 387 (Star)
A star φ over a colored signature $\mathbb{P}$ is an indexed family $|\varphi| \to \mathbb{C} \times \mathbb{T}$.
We will use the notation $\varphi[i]$ instead of $\varphi(i)$ to insist on the fact that i is an index, taking inspiration from arrays in programming.
The set of stars is denoted by $\mathbb{S}$.

Definition 388 (Rays and unpolarized terms)
A star φ can be decomposed into two different families. We define:

- $|\varphi|_M := \{i \in |\varphi| \mid \varphi[i] \text{ unpolarized}\}$
- $|\varphi|_R := \{i \in |\varphi| \mid \varphi[i] \text{ polarized}\}$

The family φ can then be decomposed into:

$$\begin{cases} \varphi_M : |\varphi|_M \to \{\oslash\} \times \mathbb{T} & \text{, the } \textit{internal memory} \\ \varphi_R : |\varphi|_R \to \texttt{Rays} & \text{, the } \textit{interface rays} \end{cases}$$

The elements whose index are in $|\varphi|_M$ will be called "the (internal) memory" of φ. We denote by $\texttt{Mem}(\varphi)$ the image of φ_M.
The elements whose index are in $|\varphi|_R$ will be called "the (interface) rays" of φ. We denote by $\texttt{Rays}(\varphi)$ the image of φ_R.

Convention
For convenience, to define finite stars, we will sometimes write them as two unordered sequence $[r_1, \dots, r_k]\{m_1, \cdots, m_n\}$ when we want to insist on the fact that the memory is invisible from the outside, or as one sequence $[r_1, \dots, r_k, m_1 \dots m_n]$, which is not ambiguous since the r_i and m_j can be split based on their polarities.

Definition 390 (Variables of a star)
The set of variables appearing in φ is defined by $\texttt{vars}(\varphi) := \bigcup_{i \in |\varphi|} \texttt{vars}(\varphi[i])$.

Definition 391 (Isolated star, Empty star)
A star is *isolated* [1] if $|\varphi|_R = \emptyset$ (there is no interface).
Since for such a star, the rays are [], we will omit them and write it directly as $\{m_1, \cdots m_n\}$.
The empty star is written $\{\}$ and is defined as the isolated star with empty memory.

Example
The "shape" of a star $[-c_1(t_1^1, ..., t_n^1), \ldots, -c_k(t_1^k, ..., t_m^k),$
$+ d_1(u_1^1, ..., u_{n'}^1), \ldots, +d_{k'}(u_1^{k'}, ..., u_{m'}^{k'})]\{v_1, \ldots v_l\}$ is illustrated in figure 4.1.

Intuition
A star is the primitive element of the model of computation, akin to an edge in an interaction graph, or a partial isometry in older GoI models, but it is a bit different in nature: to make an analogy with electricity, an edge would be a current flowing in a direction (or the other) while a star would be more like the wire itself.

Definition 394 (Substitution applied on a star)
Let θ be a substitution and φ a star. The application of θ on φ is defined by a star $\theta\varphi$ with $|\theta\varphi| := |\varphi|$ and $(\theta\varphi)[i] = \theta(\varphi[i])$.

Definition 395 (Alpha-equivalence of stars)
We say that two stars φ_1 and φ_2 are α*-equivalent*, written $\varphi_1 \approx_\alpha \varphi_2$, when there exists a renaming α such that $\varphi_1 = \alpha\varphi_2$.

Convention (Alpha-Renaming Stars)
Stars are considered up to α-equivalence.
We therefore define $\texttt{Stars}(\mathbb{P})$ as the set of all stars over a colored signature $\mathbb{P}$, quotiented by $\approx_\alpha$ and reindexing.

We can now define the programs of our model of computation, akin to a graph in an interaction graph, a wiring in `Flows`

Definition 397 (Constellation)
A *constellation* Φ is an indexed family of stars over a set $|\Phi|$.
For convenience, a finite constellation will often be written as a sum of stars $\Phi = \varphi_1 + ... + \varphi_n$.
We define the set of (indexes) rays of a constellation Φ by
$\texttt{IdRays}(\Phi) := \{(i, j) \mid i \in |\Phi|, j \in |\Phi[i]|_R\}$ (note how we keep track of which instance of star the rays come from).
The empty constellation is written 0 and is defined by $|0| = \emptyset$.

Intuition
Constellations are meant to be the programs of the model of computation, .

[1] This corresponds to scalars in previous GoI models

Convention (Variable Scope)
Similarly to how variables are bound in logic programming (*e.g.* Prolog) or functional programming (*e.g.* λ-calculus), variables in constellations will be considered bound to their star (which can be seen as sort of declarations), hence the two x in $[+t(x)] + [-u(x), y]$ are unrelated.
This is similar to how the two x in the λ-term $(\lambda x.t)(\lambda x.u)$ are different.

Now that all the elementary objects of the stellar resolution are defined, we can give example of programs in our model. The fact that we have terms means that we can encode logic programs which are a great source of examples. We thus define an encoding of natural number in terms:

Definition 400 (Encoding of natural numbers)
When dealing with natural numbers in examples, we will use the signature $nat : 1, s : 1, 0 : 0$ inspired by Peano Arithmetic.
We define the term $\overline{n}$ by induction on the natural number $n \in \mathbb{N}$:

- $\overline{0} = 0$
- $\overline{n+1} = s(\overline{n})$

We will then use terms of the shape $+nat(\overline{n})$ to encode the affirmation "n is a natural number".

Example
We give examples of finite and infinite constellations:

Unary Naturals: $\Phi_{\mathbb{N}}$ is defined by $|\Phi_{\mathbb{N}}| = \mathbb{N}$ and $\Phi_{\mathbb{N}}[i] := [-nat(\overline{i}), +nat(\overline{i+1})]$;

Addition (Logic Program):

$$\Phi^{+}_{\mathbb{N}} := [+add(\overline{0}, Y, Y)] + [-add(X, Y, Z), +add(s(X), Y, s(Z))]$$

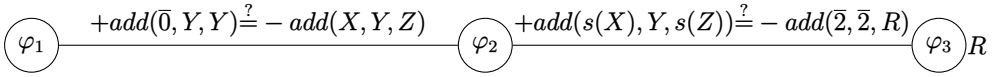
Query for addition:

$$\Phi^{n+m}_{\mathbb{N}} := \Phi^{+}_{\mathbb{N}} + [-add(\overline{n}, \overline{m}, R)]\{R\}$$

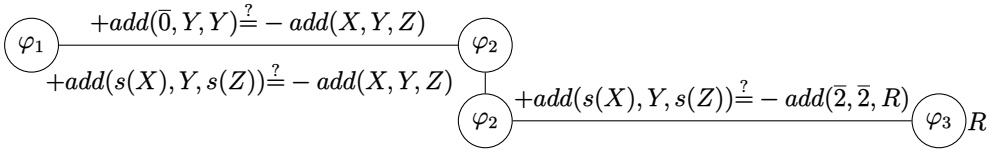
(This will normalise to $\{n+m\}$).

These were define over the signature defined by the variables $V = \{X, Y, Z, R\}$, the symbols $F = \{add, nat, s, 0\}$, $\mathtt{ar}(add) = 3, \mathtt{ar}(nat) = \mathtt{ar}(s) = 1, \mathtt{ar}(0) = 0$.
The constellation $\Phi^{n+m}_{\mathbb{N}}$, interpreting $Add(X, Y, Z)$ as stating that $X + Y = Z$, corresponds to the following Horn clauses: [62]

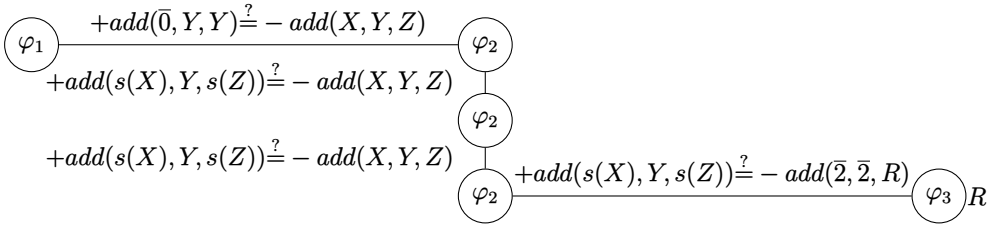
$$Add(0, Y, Y) \qquad \text{and} \qquad Add(X, Y, Z) \Rightarrow Add(s(X), Y, s(Z)).$$



(a) 0 recursive call.



(b) 1 recursive call.



(c) 2 recursive calls.

Examples of diagrams for the constellation $\Phi_{\mathbb{N}}^{2+2}$. The number of occurrences of φ_2 corresponds to the number of recursive calls. They correspond to unfolding of the loop of figure 4.6 corresponding to the possibility of recursive call.

4.1.2. Evaluation of diagrams and execution of constellations

We are now interested in the formation of *diagrams*, as was done in section 3.2, to give computational meaning to our programs.

In the setting of stars, diagrams will visually correspond to tilings of stars (without any planarity constraints). But unlike usual tiling systems, it is possible to evaluate these diagrams by an edge contraction using unification, retracting the diagram.

Intuitively, such a tiling is formed of a graph which will give the shape of a tiling, and of a form of fibration which tells which instance of a star is above which vertex (and how).

Note that in TS, diagrams might have self loops. On the technical side, we try to handle self loops in a uniform way with "regular" edges, but it requires some special definitions. We start with an alternative definition for graph with "half-edges" in the sense that there are witnesses that endpoints of edges go to a certain vertex.

Definition 402 (r(elevant)Graph)
A (proof-relevant) graph, abbreviated as rgraph, is given by V a set of vertices, E a set of edges, and a span $E \xleftarrow{\pi} W \xrightarrow{\partial} V$ such that the cardinal $|\pi^{-1}(e)| = 2$ for every $e \in E$ (W should be seen as the set of witnesses that edges have certain

vertices as endpoints, and the condition saying it is $\simeq E \times \{s,t\}$).
An edge e is thus entirely characteriszed by two witnesses. We will note $e : v - v'$, $e = w, w' : v - v'$ or even $w, w' : v - v'$ to denote an edge e such that $\pi(w) = \pi(w') = e$ and $\{\partial(w), \partial(w')\} = \{v, v'\}$. We will also often abusively write $f(e)$ for $\{f(w), f(w')\}$.
Finally, we will only actually consider isomorphism classes of rgraphs.

Definition 403 (Neighbours)
Given an rgraph $G = (V, E, (\pi, W, \partial))$, and a vertex $v \in V$, define $\texttt{Neigh}(v) := \{w \in W \mid \partial(w) = v\} \subseteq W$ the set of witnesses that edges are neighbours of v.

Remark
Using witnesses this way allow to handle self loops in a uniform way: they are counted twice as neighbors of their endpoint v, since there are two witnesses of that fact.

Definition 405 (Choice of Star in a constellation)
Given an rgraph G, and a constellation Φ a *choice of star index* for a vertex v is given by an index i of a star $\Phi[i]$, and an injective function $\delta_v : \texttt{Neigh}(v) \to |\Phi[i]|_R$ choosing an index of ray to be "above" every (proof relevant) edge of v.
Note this is a *choice of index* of star and ray. One could directly define a choice of star and ray, but to be able to take a deep dive in the small step dynamics it is necessary to deal with at least ray indexes.

Definition 406 (Pluggable Choice)
Two choices of star in a constellation Φ, (i_v, δ_v), $(i_{v'}, \delta_{v'})$ in an rgraph G are pluggable, also written $\delta_v \Finv \delta_w$, letting $\varphi_v := \Phi[i_v]$ and $\varphi_{v'} := \Phi[i_{v'}]$, when we have for all $e = w, w' : v - v'$ in G_δ, $\varphi_v[\delta_v(w)] \Finv \varphi_{v'}[\delta_{v'}(w')]$.

We now need to explain how stars can be plugged together like edges can be composed in interaction graph. The particularity of stars being that they will form complex shapes:

Definition 407 (Diagram)
A *C-diagram* on a program Φ is a pair $(|\delta|, \delta)$ (abusively written just δ), where

- $|\delta| = (V_\delta, E_\delta, (\pi_\delta, W_\delta, \partial_\delta))$ is a non-empty finite connected rgraph, called *the shape* of the diagram.
- δ is a family $(\delta_v)_{v \in V_\delta}$ of pluggable choice of stars, that is, (packing some things together):
  A function $\delta : V_\delta \to |\Phi|_R$ and a family of functions δ_v such that $(\delta(v), \delta_v)$ is an effective choice of index of stars in Φ for all vertices v;
  Such that all these choices are pairwise pluggable: $\delta_v \Finv \delta_w$ for all v, w (note $v = w$ is a possible case).

We denote by 0 the empty diagram, representing a failure of computation.
For an edge $e = w, w' : v - v'$, we use notation $\delta(e) := \{\delta_v(w), \delta_{v'}(w')\}$ (this is a set of indices).

Remark
We could drop the condition on pairwise pluggability, a diagram not respecting this condition would have a failing computation anyway, but it is kept here to be close to the original definition.

Remark
Note that δ_v is a choice of (indexes of) *rays*, the internal memory is completely irrelevant to the dynamics of stars.

Note
This definition depends on the choice of a constellation. We could do a similar "weaker" definition where, instead of choosing an index of a star, whe choose a star directly (we will not because it would bealmost exactly the same).
Nonetheles, given a diagram one can extract a weak diagram out of it that would have exactly the same properties when it comes to computation (but avoids referencing a particular program).
An alternative would be to use the saturation method, where we consider diagrams on a program Φ_∞ containing the result of all possible n-ary combinations of stars. Here, we will use the weaker version when it comes to some dynamics to avoid the technicalities that come with dependence on a program.

Remark (Diagrams in Stellar Resolution, Diagrams in Interaction Graphs)
In this thesis, we developed a notion of diagram for interaction graphs and similar models.
We remind quickly of the notion of dual hypergaph: it exchanges vertices and hyperedges. Vertices become hyperedges (which so happens to be edges, in the case of an hypergraph with the wire-network property) and hyperedges become vertices.
The notion of diagram developed above is actually different, but deeply linked to the first one: what we consider here is the dual, in the sense of hypergraphs, to the notion of diagram defined in other models of computation.
It so happpens that thanks to the wire-network property, this dual hypergraph can be expressed as a graph. I decide to nickname the main approach location-first, because the vertices are locations, and the approach used here wire-first, because the vertices are wires.

We quickly give another definition of diagrams, more in line with the usual style of diagrams, adapting in a straightforward manner the definitions of the previous chapter:

Definition 412 (Arities of a vertex)
In an hypergrah $(V, E, \texttt{in}, \texttt{out})$, a vertex is said to have:

- $+$-arity the integer $+ar(v) := |\{e \in E \mid v \in \mathtt{out}(e)\}|$
- $-$-arity the integer $-ar(v) := |\{e \in E \mid v \in \mathtt{in}(e)\}|$
- arity, the integer $ar(v) := +ar(v) + -ar(v)$

Definition 413 (Wire-Network)
A directed hypergrah $G = (V, E, \mathtt{in}, \mathtt{out})$ is said to have the wire network property when for all v in V, $1 \leqslant ar(v)$ and $+ar(v), -ar(v) \leqslant 1$.
We say that $v \in V$ is in:

- The interior of G when $ar(v) = 2$
- The boundary of G when $ar(v) = 1$

Convention (Partiality)
We remind the reader of our convention for partiality:
Given a set S, we defined $S^* := S \sqcup \{\oslash\}$, adding an element $\oslash$.
This element will be used as a case "undefined" when defining partial operations.

Definition 415 (In and Out)
Given an hypergaph with the wire-network property and a vertex $v \in V$, define:

- $\mathtt{e}^-(v) \in E^*$ the only edge such that $v \in \mathtt{in}(e)$ if it exists, $\oslash$ otherwise
- $\mathtt{e}^+(v) \in E^*$ the only edge such that $v \in \mathtt{out}(e)$ if it exists, $\oslash$ otherwise

Definition 416 (Diagrams (Hypergraph form))
A diagram on a program Φ is given by a directed hypergraph $G = (V, E, \mathtt{in}, \mathtt{out})$ with the wire-network property, and the data of a function $\delta : E \to |\Phi|$ equipped with bijections $\delta_e \colon \mathtt{in}(e) \sqcup \mathtt{out}(e) \simeq |\Phi[\delta(e)]|_R$ preserving polarities, by that we mean that a term with $+$ polarity should be sent on $x \in \mathtt{out}(e)$ and similarly a term of $-$ polarity.
The compatibility condition becomes $\forall v, \delta_{\mathtt{e}^+(v)}(v) \supset \delta_{\mathtt{e}^-(v)}(v)$.
Note in this definition, the choice is made that rays that are not used by a star are still attached to a vertex, but that vertex has arity 1 (so is on the boundary). We also omit the coherence condition

Intuition
The vertices correspond to locations where stars can be plugged together.
The hyperedges are stars, and the wire-network property implies that either there is one ray on a location, and it is a boundary of the diagram, an open end, or there is two and then the two rays are plugged together. (Note how we cannot branch three rays together).

Hole
The advantage of such a definition is multiple: no need to deal with rgprahs, the boundary of a diagram is easier to define and many definitions are as well in consequence...

Since this thesis deals with the two approach, location first for most models, and wire-first here, it should be possible to compare them and see the pros and cons in more details in future work.
I personally prefer the hypergraph one but to stay consistent with historical use, I will use the other one in this manuscript.

We will explain how the notions are dual a bit later in the chapter.

We must now state when two diagrams are equivalent: indeed, diagrams should not depend on the names used for the vertices or edges in their shape. Intuitively, diagram equivalence should be related to tiling equivalence. Two diagrams should be equivalent when they refer to an equivalent tiling of stars in a given constellation.

Definition 419 (Sub-diagram)
We define a binary relation $\sqsubseteq$ (illustrated in figure 4.5) on diagrams by $\delta' \sqsubseteq \delta$ if there exists $H \subseteq |\delta|$, an rgraph isomorphism $\varphi : H \simeq |\delta'|$ such that

- $\delta|_H = \delta' \circ \varphi$.
- $\delta_v|_H = \delta'_{\varphi(v)}$

Note 0 is such that $0 \sqsubseteq \delta$ for all δ.

Proposition (Diagram equivalence)
The relation $\sqsubseteq$ is a preorder. It induces a notion of equivalence $\simeq$ on diagrams.

Convention
We consider diagrams only up to the equivalence $\simeq$ defined above.

Saturation and boundaries

Definition 422 (Diagram over a constellation, using a specific color)
We say that a diagram on Φ is C *colored*, or at C, with C a set of colors, noted $\delta : C$, when all the rays it uses are of color in C, that is $\delta_v(w) : C$ for all $v \in |\delta|, w \in \texttt{Neigh}(v)$.

Example
An example of three diagrams for the constellation $\Phi_{\mathbb{N}}^{2+2}$ (which is an instance of the constellation $\Phi_{\mathbb{N}}^{n+m}$ of example 401) is given in section 4.1.1.

In a diagram, not every ray of every star was used. The remaining ones (those on the boundary) are called *free* rays:

Definition 424 (Free rays, closed diagrams)
Given a diagram δ on Φ, we define its multiset of indexes of *free rays*:

$$\texttt{IdFreeRays}(\delta) := \{(v, i) \in V_\delta \times |\Phi[\delta(v)]|_R \mid i \notin \delta_v(\texttt{Neigh}(v))\}$$

These are the indexes of the unused rays of every *instance* of star used in the diagram. These are the indices of the rays on the boundary of the diagram.
Given a colors C, we can also define the free rays of colors in C:

$$\texttt{IdFreeRays}_C(\delta) := \{(v,i) \in \texttt{IdFreeRays}(\delta) \mid \Phi[\delta(v)][i] : C\}$$

Note these constructions are similar to `IdRays` in the sense we remember via v which instance of the star the ray came from.
Finally, we define in similar fashion all indexes of the memories of stars collected in one set: $\texttt{IdMem}(\delta) := \{(v,i) \mid v \in V_\delta, i \in |\delta(v)|_M\}$

Remark
We have $\pi_2(\texttt{IdFreeRays}(\delta)) \subseteq \pi_2(\texttt{IdRays}(\Phi))$.

Now that we have defined the notion of free rays, which is the "boundary" of the diagram, we can explain how the notion of hypergraph diagram is dual to the graph definition of diagram.
There is a slight annoying technicality which is that the notion of dual hypergraph only exists for *undirected hypergraphs* (usually called just "hypergraphs").
The fact is that, in a diagram, the polarity of terms gives extra information allowing to make the dual hypergraph directed. We first define the notion of hypergraph and dual hypergraph as a reference too se how our construction is a duality of the sort:

Definition 426 (Hypergraph)
A hypergraph H is a tuple (V, E, ∂) of a set V of elements called *vertices*, a set E of elements called *hyperedges* and a function $\partial : E \to \mathcal{P}(V)$ mapping hyperedges to their associated *endpoints sets*, such that $\partial(e) \neq \emptyset$ for all $e \in E$.

Definition 427 (Dual Hypergraph)
Given an hypergraph $G = (V, E, \partial)$, its dual hypergraph is the hypergraph $\overline{G} = (E, V, \partial')$ with $\partial'(v) = \{e \mid e \in V\}$.

Definition 428 (Dual hypergraph of a diagram)
Given a diagram $\delta = ((V, E, (\pi, W, \partial)), \delta)$ on Φ, it's dual hypergraph diagram is $\overline{\delta} := ((E \sqcup \texttt{IdFreeRays}(\delta), V, \texttt{in}, \texttt{out}), \overline{\delta})$ with:

- $$\begin{aligned}\texttt{in}(v) :=& \{\pi(w) \in E \mid w \in W, \partial(w) = v, \Phi[\delta_v(w)] \text{ is } -\text{polarized}\} \\ &\cup \{(v,i) \in \texttt{IdFreeRays}\delta \mid \Phi[\delta(v)][i] \text{ is } -\text{polarized}\}\end{aligned}$$

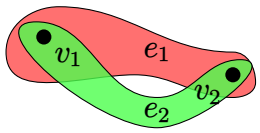
•

$$\begin{aligned}\texttt{out}(v) :=&\{\pi(w) \in E \mid w \in W, \partial(w) = v, \Phi[\delta_v(w)] \text{ is } +\text{polarized}\}\\ &\cup\{(v,i) \in \texttt{IdFreeRays}\delta \mid \Phi[\delta(v)][i] \text{ is } +\text{polarized}\}\end{aligned}$$

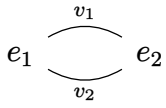
Finally, $\overline{\delta}(v) = \delta(v)$ and the bijections are defined by case with $\overline{\delta_v}(w) = \delta_v(w)$ and $\overline{\delta_v}((v,i)) = i$.

Definition 429 (Dual diagram of an hypergraph diagram)
Given an hypergraph diagram $G = (V, E, \texttt{in}, \texttt{out})$, we define its dual diagram $\overline{\delta} = ((E, \{v \in V \mid ar(v) = 2\}, (\pi, \{-v, +v \mid v \in V, ar(v) = 2\}, \partial)), \overline{\delta})$ where $\partial(\bullet v) = v$ and $\pi(\bullet v) = \texttt{e}^\bullet(v)$.
Finally, the function $\overline{\delta}$ is defined by $\overline{\delta}(e) = \delta(e)$ and $\overline{\delta}_e(w) = \delta_e(\pi(w))$.

Example
Here is an exemple of an hypergraph G:



Here would be its dual hypergraph $\overline{G}$, which happens to be a graph because G has the wire-network property:



Now say that G is a directed hypergprah with $v_1, v_2 \in \texttt{in}(e_2)$ and $v_1, v_2 \in \texttt{out}(e_1)$. Then, if G was extended into an hypergraph diagram δ, we would have $-t_1 := \delta_{e_2}(v_1)$ and $\delta_{e_2}(v_2)$ are $-$ terms. We would also have $+u_1 := \delta_{e_1}(v_1)$ and $\delta_{e_1}(v_2)$ are $+$ terms.
Thus, above v_1 in the graph $\overline{G}$, we could add $-t_1 \overset{?}{=} +u_1$. Similarly for v_2. This would make $\overline{G}$ a diagram.

Saturation and Retraction

Remember, diagrams are a generalisation of the composition of flows: we need to define "when to stop" composing stars together, that is when our boundary lies outside of the cut.

This is the notion of saturated diagram: in terms of tiling it is understood as the construction of the largest constructible tiling with occurrences of tiles from a given tile set.

Note that this chapter has a top-down approach to computation: we assume given all saturated diagrams (finished computations) and we will filter all the inccorrect ones (where unification will fail). This approach is saturated first and then correct.

There would equivalently be a bottom up, or correct-first approach: construct all possible diagrams by plugging stars again and again, but only if this does not fail. (This is the original approach of Girard, which is correct first and then saturated).

There are some differences: here we will not care about non-normalisation, if it is impossible to create a saturated diagram then there will just not be any. The bottom up approach is the more "realistic" one but is more annoying to deal with. These differences are discussed quickly in section 4.4.1, the main problem being a form of non-confluence: see proposition 548.

To make some parrallels, the bottom up approach is akin to a regular model of computation, like lambda calculus, while the top down is more like Bohm-trees.

Definition 431 (C-Saturated diagrams)
A diagram δ is *C-saturated* with C a set of colors when $\texttt{IdFreeRays}_C(\delta) = \emptyset$.
If $\texttt{IdFreeRays}(\delta) = \emptyset$, we say that δ is *closed*.

This means that one cannot plug anything of color C on this diagram. Just to be exhaustive, we also give what would naively appear to be a correct-first definition, but is in fact not

Definition 432 (Maximal diagram)
A diagram δ on Φ is said to be maximal when there are no correct $\delta' \neq \delta$ on Φ such that $\delta' \sqsupseteq \delta$.
This means we cannot add any star of Φ to the diagram without creating an error.

Remark (Bottom-up approach to saturation)
This might naively look like the "correct-first" approach to saturation when one only has the intuition that we are building diagrams by composing stars and stop when we cannot, adding as much stars as we can.
It is not: these are the maximals correct elements for the poset of the order previously defined (actually the subposet of Φ diagrams). These maximal elements are *not necessarily saturated*: a lot of them correspond to incomplete computations (a path stuck in the middle of a proof net, that cannot be extended without getting incorrect).
To do a correct-first/bottom-up approach, there is a need to compute these, but then still filter the ones that are saturated, which is "dual" to the approach here of computing the saturated and filtering by keeping only the correct ones.

Remember that variables are local to stars. To avoid any problem of accidental capture of variable we assume the following thing:

Convention
We considering a diagram δ, we assume for convenience that we have an injection $\texttt{inj} : \texttt{vars} \times V_\delta \to \texttt{vars}$ (possible because V_δ is finite and $\texttt{vars}$ is infinite), and we α-rename every variable appearing in a instance of star $\delta(v)$ with the renaming $x \to \texttt{inj}(x, v)$, so that instances of stars are indeed pairwise with disjoint variables.

Reduction of a diagram

Now that we defined the notion of diagram, which intuitevly should be seen as *an attempt at composing stars*, we have to explain how to actually perform the composition.

Convention
In this section, *we will reason on the weaker notion of diagram*, where we choose stars above every edge instead of indexes of stars. This means $\delta(v)$ is now a star and not just an index.
This is really convenient and does not change the dynamics.

Every edge induces an equation between the two rays above it (asking whether the rays are coherent):

Definition 436 (Equation of an edge)
In a diagram δ, the *underlying equation* of an edge $e = w, w' : v - v' \in E_{|\delta|}$ is defined as follows: Let $i := \delta_v(w)$ and $j := \delta_{v'}(w')$ be the indices of the rays above e in their respective stars, then the equation of e is

$$\texttt{eq}(e) := \lfloor \delta(v)[i] \rfloor \stackrel{?}{=} \lfloor \delta(v')[j] \rfloor$$

Links in a diagram have an underlying equation. It follows that a whole diagram is associated with a unification problem.

Definition 437 (Underlying equation and problem)
Let δ be a diagram.
The *underlying problem* of δ is defined by

$$\texttt{Prob}(\delta) = \{\texttt{eq}(e) \mid e \in E_\delta\}.$$

Like instances of stars, underlying problems are considered up to α-equivalence: variables should be distinguished when they come from two distinct stars.

Definition 438 (Compatible diagrams)
A diagram δ is *compatible* if $\texttt{Prob}(\delta)$ has a solution.

We are now ready to define the dynamics of our computational system, and we first define how to reduce a diagram:

Definition 439 (Actualisation: The one-step semantics)
The *actualisation* of a compatible diagram δ is the star $\Downarrow\delta$, defined as follows (remember we assume that there everything is α renamed).
For indexes, we have:

$$\begin{cases} |\Downarrow\delta|_R := \texttt{IdFreeRays}(\delta) & \text{, all rays remaining that were not used in the diagram} \\ |\Downarrow\delta|_M := \texttt{IdMem}(\delta) & \text{, all the internal memories collected together} \end{cases}$$

And from this, taking $\theta := \texttt{solution}(\texttt{Prob}(\delta))$ be the $\texttt{MGU}$ of the problem, we define the family: $(v,j) \in |\Downarrow\delta| \to \theta(\delta(v)[j])$, updating the rays and memories from the information computed.

Remark
Topologically, this is retracting the diagram on a point, which is "evil" since it does not preserve cycles and so lose topological information.

We can finally define our execution formula, the dynamics of our programs:

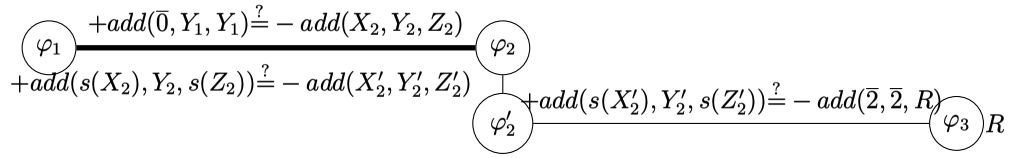
Definition 441 (C-Valid Diagram)
A diagram is said to be C-valid when it is connected, at C and C-saturated.
The set of C-valid diagrams on Φ is written $\texttt{VDiags}_C(\Phi)$.

Definition 442 (Correct Diagram)
A diagram is said to be C-correct when it is C-valid and compatible.
The set of C-correct diagrams on Φ is written $\texttt{CDiags}_C(\Phi)$.

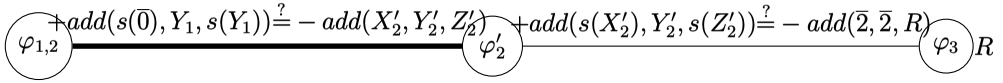
Proposition (Execution)
Given a constellation Φ, a set of color C, its execution $\pitchfork_C(\Phi)$ is defined as the multiset $M := [\Downarrow\delta \mid \delta \in \texttt{CDiags}_C(\Phi)]$

Note
Here, we did not consider an element $\perp$ yet to represent a failure of retraction, and a quotient of families by the presence of $\perp$.
We thus define execution as the retraction of the correct diagrams, excluding incompatible ones, instead of valid diagrams.

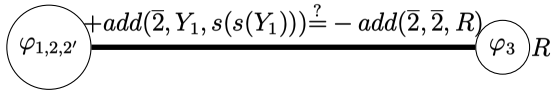
In the case of interaction graphs, retracting a diagram was just collapsing a line. In this case where diagrams are more complex, the lower-level dynamics are a bit more interesting, and we thus do an interlude to study in more details how to reduce diagram step by step instead of through actualisation.



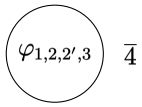
(a) Correct diagram computing $2 + 2$ (1 recursion) with memory R



(b) Fusion of φ_1 and φ_2 with $\theta := \{X_2 \mapsto \overline{0}, Y_2 \mapsto Y_1, Z_2 \mapsto Y_1\}$.



(c) Fusion of $\varphi_{1,2}$ and φ_2' with $\theta := \{X_2' \mapsto s(\overline{0}), Y_2' \mapsto Y_1, Z_2' \mapsto s(Y_1)\}$



(d) Fusion of the two remaining stars with $\theta := \{R \mapsto \overline{4}\}$.

Fusion of the diagram from figure 4.2b.

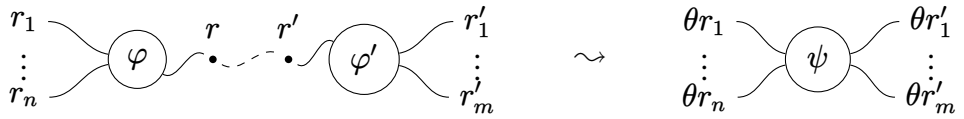


Figure 4.4: Illustration of a step of fusion where θ is the principal unifier of the underlying unification problem of the pair of rays (r, r'). The fusion of the two stars φ and φ' along the rays r and r' produces a new star ψ.

4.1.3. Step by step retraction

In Girard's original paper [28, Section 2.3], the evaluation of diagrams is defined as an edge contraction. An edge e between two stars φ and φ' contains an equation which is resolved and then the associated solution is propagated to both φ and φ'. The two connected rays associated with e are finally contracted in the process, fusing the stars. It reminds of chemical interactions, and more generally of the propagation of the flow information in a network. This process can fail, when attempting to plug wires on non matchable-terms, causing errors during the execution of the unification algorithm. This corresponds exactly to Robinson's resolution in first-order logic, and it generalises the composition of flows (*cf.* section 3.5.1).

We define this step-by-step procedure of diagram contraction based on an operation of *fusion*. It is related to *actualisation* in a similar way than a small step evaluation differs from big step evaluation in the theory of programming languages [2, Section 1.1].

Definition 445 (Self Fusion)
Let $\varphi := \varphi_1 \sqcup \{r_1\} \sqcup \{r_2\}$ be a star with $r_i = \varphi[j_i]$, such that $\varphi[j_1] \asymp \varphi[j_2]$. We define its *self fusion* over indexes j_1, j_2 as the star ${}^{j_1,j_2}\!\circlearrowleft\!\varphi := \theta\varphi_1$ where $\theta := \texttt{solution}\{\varphi[j_1] \stackrel{?}{=} \varphi[j_2]\}$.
(Note that this is only defined when the rays are coherent!)

Definition 446 (Star Mixing)
Let φ_1 and φ_2 be stars. We define their star-mixing as their disjoint union, *i.e.* a new star φ with $|\varphi| := |\varphi_1| \sqcup |\varphi_2|$, with obvious assignation of ray and memory and alpha renaming of local variables.

There are possible variants of transcendental syntax where one does not consider self loops. In this case, one can also use this definition, which here is simply a composition of one mix and one self:

Definition 447 (Link Fusion)
Let $\varphi_1 := \varphi_1' \sqcup \{r_1\}$ and $\varphi_2 := \varphi_2' \sqcup \{r_2\}$ be two stars with $r_i = \varphi_i[j_i]$. We define their *link fusion* $\varphi_1 {}^{j,j'}\!\leftrightarrow\! \varphi_2$ by $\theta\varphi_1' \sqcup \theta\varphi_2'$ where $\varphi_1[j] = \varphi_2[j']$ and $\theta := \texttt{solution}\{\varphi_1[j] \stackrel{?}{=} \varphi_2[j']\}$.
We simply write $\varphi_1 \leftrightarrow \varphi_2$ (and $\varphi_1 \leftrightarrow_\alpha \varphi_2$) without indexes above when there is a unique

possible choice of index and we choose to leave them implicit. Fusion is illustrated in figure 4.4.

Remark
This step is the only small step that truly respect the topology of the diagram.

Example
Assume we have two stars $\varphi^{+}_{\mathbb{N}} := [-add(X,Y,Z), +add(s(X),Y,s(Z))]$ and $\varphi^{2+2}_{Q} := [-add(\overline{2},\overline{2},R)]\{R\}$ indexed by natural numbers corresponding to their position.
We have:

$$\varphi^{+}_{\mathbb{N}} {}^{1,0}_{\leftrightarrow} \varphi^{2+2}_{Q} := [-add(\overline{1},\overline{2},Z)]\{s(Z)\}$$

$$\varphi^{+}_{\mathbb{N}} {}^{1,0}_{\leftrightarrow_\alpha} (\varphi^{+}_{\mathbb{N}} {}^{1,0}_{\leftrightarrow_\alpha} \varphi^{2+2}_{Q}) := [-add(0,\overline{2},Z')]\{s(s(Z'))\}$$

As said in the intro of this section, the small step semantics is based on reducing edge after edge until we are left with one star. We define:

Definition 450 (Edge Fusion)
Given a diagram δ, and an edge $e = w, w' : v - v' \in |\delta|$, we define the star (or error message in case of failure) resulting of the contraction of said edge as follows:

$$\star(e) = \begin{cases} \bot & \text{if } \mathbf{eq}(e) \text{ has no solution} \\ \overset{\delta(e)}{\circlearrowleft}\delta(v), & \text{if } v = v'. \\ \delta(v) \overset{\delta(e)}{\leftrightarrow} \delta(v') & \text{otherwise.} \end{cases}$$

Remember $\delta(e)$ stands for $\{\delta_v(w), \delta_{v'}(w')\}$.

We now define a notion of edge-contraction on rgraphs to define the shape of the resulting diagram. This should retract an edge in a graph and fuse its two endpoints. In the case of a self loop though, the edge just disappears.

Convention
We define a formal operation $[v' \leftarrow v]$, which designates the formal replacement of v' by v in a set or a function.
The point of this notation is to handle self loops in a uniform way in the contraction, because replacing v by v does not change a thing.
We illustrate the notation on examples:

- $\{a, v', v\}[v' \leftarrow v] = \{a, v\}$, so here v' is "contracted" on v.
- $\{a, v\}[v \leftarrow v] = \{a, v\}$, so here it does not change a thing.
- $f[v \leftarrow v']$ is the function associating to w the element $f(w)[v \leftarrow v']$, this is used so that elements who had v' as an endpoint gets v as an endpoint.

Definition 452 ("Proof relevant" edge contraction)
Given an rgraph $G = (V, E, (\pi, W, \partial))$ and an edge $e = w, w' : v - v'$, one can define a new rgraph G/e as follows:

- $V_{G/e} = V[v' \leftarrow v]$
- $E_{G/e} = E - \{e\}$
- $W_{G/e} = W - \{w, w'\}$
- $\partial_{G/e} = \partial[v' \leftarrow v]$

(Note that we contracted v on v', but contracting v' on v gives an isomorphic graph, so the same for our considerations. Also note that thanks to our previous convention, the case $v = v'$ is handled by this definition).

Remark
We could also define an equivalent version of said fusion where fusing an edge when $v \neq v'$ does not remove it but just mix the stars.

We can now define our rewriting system on diagrams:

Definition 454 (Diagram contraction)
We define a rewriting relation on diagrams $\rightsquigarrow$ called *diagram contraction*: take a diagram δ, an edge $e : a - b \in E_\delta$ (possibly with $a = b$).

$$\begin{cases} \delta \rightsquigarrow_e 0, & \text{if } \star(e) = \bot. \\ \delta \rightsquigarrow_e \gamma, & \text{otherwise.} \end{cases}$$

Note that there are 2 notions of neutral, one for the stars named $\bot$ representing a local failure, and one for the diagrams/constellation 0 representing an incorrect diagram (this notation is consistent with the additive notation for constellations).
With γ defined as $|\gamma| := |\delta|/e$ the graph quotient where endpoints of e are identified, and

$$\begin{aligned} \gamma(v) &= \begin{cases} \star(e), & \text{if } v = a \\ \delta(v), & \text{otherwise} \end{cases} \\ \gamma_v(w) &= \delta_{\partial_\delta(w)}(w) \end{aligned}$$

That is, we fuse a and b, removing edge e in the process. (All other edges between them become self loops).
Above a we put the new star $\star(e)$, and to find the ray above a witness w, we go find the vertex to which it was attached $\partial_\delta(w)$ (possibly b) and we check the ray that was above in δ.
We define $\delta \rightsquigarrow \gamma$ if $\delta \rightsquigarrow_e \gamma$ for any e.

Note
Note that one could also define a rewriting system with just a "mix" rule and "self", since link-fusion can be expressed using both. In such a system, one can push all mix rules at the beginning, creating one big star, and use only self rules afterwards.

Definition 456 (Simulation)
Assume given two transition system $T := (S, \to)$, $T' := (S', \to')$.
We say T' simulates T when there is a relation $R \subseteq S \times S'$ such that if $(p, p') \in R$ and $p \to q$ then there exists q' such that $p' \to' q'$ and $(q, q') \in R$.

Proposition (Simulation into Martelli-Montanari)
Given a diagram δ, define $\|\delta\| := \texttt{Prob}(\delta)$, a translation from diagrams to unification problems.
Then $R := (\delta, \|\delta\|)$ is a simulation of $\rightsquigarrow$ inside the rewriting $\to^*$ of a variant of Martelli-Montanari, where solved equations are put "outside of the system" once used in the replace rule.

Proof
We do the proof for both "variants" of our rewriting system, the one above and the self + mix one. (The self and link rules are similar here so we just add one extra case).

- If $\delta \overset{\text{mix}}{\rightsquigarrow} \gamma$, then $\|\delta\| = \|\gamma\|$. We have $\|\delta\| \to^* \|\delta\| = \|\gamma\|$ in 0 steps and $(\gamma, \|\gamma\|) \in R$.
- If $\delta \overset{\text{self}}{\rightsquigarrow}_e \gamma$, then $\|\delta\| := \texttt{eq}(e) \cup R$, $\|\gamma\| := R\theta$ with $\theta = \texttt{MGU}(\texttt{eq}(e))$. By focusing on $\texttt{eq}(e)$ and the recursively generated equations, we will get what we want: we have an execution of $\texttt{eq}(e) \to^* S$ with S encoding the MGU θ as oriented equations.

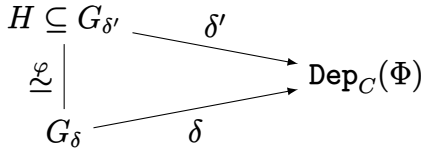
  We can lift this execution to find an execution $\|\delta\| \to^* S \cup R$ which can be extended by using rule replace on every element of S, giving us an execution $\|\delta\| \to^* R\theta$ since S encodes θ. (Note how we use the variant where we consider the solved equations to be added to a result external to the problem, hence why S "disappeared").

  This gives a rewriting $\|\delta\| \to^* \|\gamma\|$. □

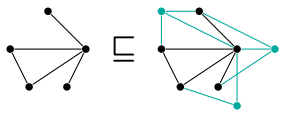
Corollary (Properties of $\rightsquigarrow$)
This rewriting relation, $\rightsquigarrow^*$, is confluent, terminating, and compatible diagrams rewrites to compatible diagrams. The normal form of δ is $\Downarrow \delta$.

4.1.4. Dependency Graph and Diagram Properties

There is another presentation that one can make of diagrams, that is really useful to get information out of them (for example, if there is no hope of termination



(a) Maximality expressed with graph homomorphisms.



(b) We have $\delta \sqsubseteq \delta'$ when δ' is an extension of δ with further links and possibly repetitions of stars. Both follow the connexs of a dependency graph $\mathtt{Dep}_C(\Phi)$.

Figure 4.5: Order $\sqsubseteq$ on diagrams representing an idea of saturation.

etc...)

Remark
This is similar to the static vs dynamic presentation of locations in interaction graphs: our locations are stars, and thus the dependency graph makes things statically located.
In a future extension of stars where locations are arbitrary terms, this a sort of unfolding of order 1 of the possible locations.

In this section we first define the *dependency graph* of a constellation, which defines the allowed connections between stars along dual rays. A diagram corresponds to an actual plugging of stars along dual rays, following those allowed connections, and thus will also be characterized by a morphism to said dependency graph.

Definition 459 (Dependency graph)
The *dependency graph* of a constellation Φ *w.r.t.* a set of colors $C \subseteq \mathbb{C}$ is an rgraph $E \xleftarrow{\pi} W \xrightarrow{\partial} V$ with:

- $V := |\Phi|$ (the indexes of stars in Φ);
- $E := \{\{(i,j),(i',j')\} \mid r := \Phi[i][j], r' := \Phi[i'][j'], r \supset r'$ and $r, r' : C\}$. We use edges as tokens that prove that two rays are coherent.
- $W := \{((i,j),e) \mid e \in E, (i,j) \in e\}$, these are witnesses that a certain ray is part of an edge.
- $\partial : ((i,j),e) \to i$
- $\pi : ((i,j),e) \to e$.

  To sum up, edges between two stars here represents rays that can be plugged.

We simply write $\mathtt{Dep}_{\mathbb{C}}(\Phi)$ when links for all colors appearing in Φ are considered, *i.e.* when $C = |\mathbb{C}|$.

Proposition (Morphism from Diagram to Dependency Graph)
A diagram $\delta : \Phi$ induces a graph homomorphism $\texttt{dep}$ to the dependency graph $\texttt{Dep}_C(\Phi) := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \partial_{\mathfrak{D}})$, with $\texttt{dep}(v) := \delta(v)$ and $\texttt{dep}(e : v - v') := \{\delta_v(e), \delta_{v'}(e)\}$.

Remark
In the case of an interaction graph G, the graph itself would be its own dependency graph. And this representation would be statically located because locations need to match locally only, thus following a path is always correct. Here there is a unification problem so even if all equations agree locally (as they do in the dependency graph) there might be a global obstruction.

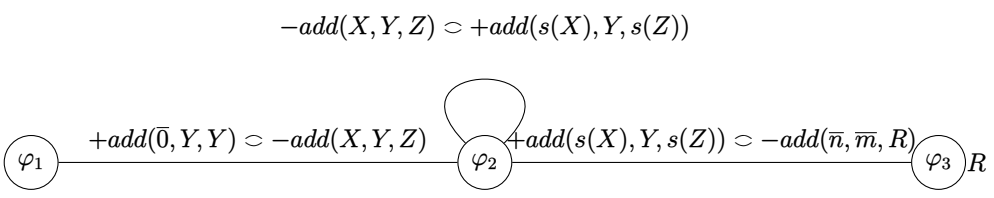
It is possible to give (yet another) alternative definition to diagrams, as a particular kind of morphism to the dependency graph.

Proposition (Diagram - alternative definition)
Let Φ be a constellation, and $\texttt{Dep}_C(\Phi) := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \partial_{\mathfrak{D}})$ be its dependency graph. The data of a diagram $\gamma : C$ on Φ is the same data as a pair (G_δ, δ) where G_δ is an rgraph and $\delta : G_\delta \to \texttt{Dep}_C(\Phi)$ an rgraph homomorphism, such that the rays of every instance of a star are used at most once,.
Mathematically, that is: for every vertex $v \in G_\delta$, and $w \neq w' \in \texttt{Neigh}(v)$, let $((i,j),e) = \partial(\delta(w))$ and $((i,j'),e') = \partial(\delta(w'))$, we have $j \neq j'$.

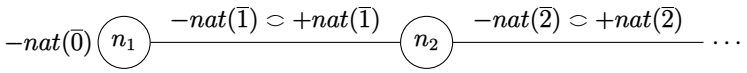
Proof
We can reconstruct the diagram γ essentially because given a witness $w \in W_{G_\delta}$, $\delta(w)$ will be a pair (i,j) and the index of the ray above w will be j □

Remark
This was the original definition of diagram in [19].

Example
Two examples of dependency graphs for the two constellations $\Phi_{\mathbb{N}}^{n+m}$ and $\Phi_{\mathbb{N}}$ of example 401 are presented in figure 4.6:



(a) Dependency graph of $\Phi_{\mathbb{N}}^{n+m}$ when $n > 0$.



(b) Dependency graph of $\Phi_{\mathbb{N}}$.

Figure 4.6: Examples of dependency graphs for constellations of example 401.

Remark
The first definition ever given of diagram was just as a graph homorphism to the dependency graph (not an relevant graph homorphism).
Since edges make explicit which star the rays come from, there is usually no ambiguity. However, in the case of loops *in the dependency graph* (not loops in general as a concept), a problematic ambiguity occured: when linking a star with one of its copies in a diagram, it was not possible to tell which ray was related to the link.
Consider the constellation $\Phi := [[+c(X), -c(X)]]$ made of a unique star $\Phi[0] := [+c(X), -c(X)]$. The problem is illustrated in figure 4.7:

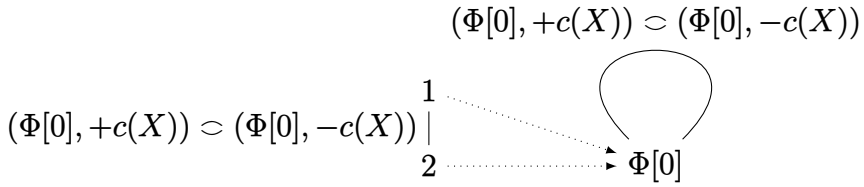


Figure 4.7: Technical remark about the dependency-graph definition of diagram. On the left we have a diagram which is associated with a dependency graph on the right. Since we only have the information of a link from a star to itself in the dependency graph, we cannot infer the membership of rays in the diagram.

The dependency graph is made of a link from $\Phi[0]$ to itself. Now, constructing a diagram of size 2 (for instance), we have two occurrences of $\Phi[0]$. But in a simple graph, the membership of rays is not distinguished, so above the edge would be the set $\{+c(X), -c(X)\}$. There are then two possibilities: either the $+c(X)$ is coming from the first instance of the star, or from the second instance. Although in this case it is not really a problem (the result will be the same), it shows that this method would not give a faithful interpretation of the graphical and intuitive presentation of the model, which may be problematic in some cases. This explains the use of rgraphs (for each rays of the associated equation must be distinguished).
We illustrate this with a truly problematic case: consider the constellation $\varphi_1 + \varphi_2$ with $\varphi_1 := [+a(f(X)), -a(X), +b(X)]$ and $\varphi_2 := [-b(X)]\{X\}$. There is a loop in the dependency graph between $-a(X)$ and $+a(f(X))$ in the first star.
Now look at the following diagram:

$$\varphi_1 \overset{\{+a(f(X)),+a(X)\}}{\frown} \varphi_1 \underset{\{+b(X),-b(X)\}}{\frown} \varphi_2$$

φ_1 is linked to a copy of itself, then to φ_2.
Depending on whether $-a(X)$ was the ray chosen on the left or on the right, we may or may not transmit $f(X)$ to $-b(X)$ and to the result $\{X\}$ by unification.

This explains the need for injections δ_v for all vertices v of a diagram using the definition of neighbours *in rgraphs*, *i.e.* of half-edges.
It explicitly shows what rays we are referring to.

4.1.5. Confluence and associativity of Execution

As usual when doing realizability for linear logic, the main theorem we want to prove is that we can eliminate cuts in the order we want.

The proof we will do here is exactly the same as the one done in the abstract setting of diagrams in section 3.2. Yet, it is not expressed in the same vocabulary and might be a little simpler conceptually since it is not done in an abstract setting. Truthfully, as explained in said section, the abstract setting was done with this model in mind, so that none of the proofs actually uses the linearity of diagrams for interaction graphs, but uses an intermediary language that can also be defined for Transcendantal Syntax (albeit, one needs to use the hypergraph definition of diagrams), using boundaries and interior. So the main proofs "really" would the same, although some intermediary lemmas needed for the main proof differ because they are properties of diagrams (and we have a more general definition here). It is important to note that here we do not prove church rosser directly, but it is obtained as a corrollary to a lemma, that uses similar arguments in its proof. This will be discussed a bit more when said lemma arrives .

Now that we discussed, this, remember that what we want to prove is a result of confluence: $\pitchfork_D (\pitchfork_C (\Phi)) = \pitchfork_{C \cup D} (\Phi) = \pitchfork_C (\pitchfork_D (\Phi))$.

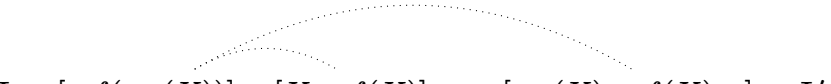
Note
Note in this setting there are two levels at play: the local level, where we are given a diagram and we want to reduce it. We already saw confluence of this level by Martelli-Montanari, and it is the reason we can define the retraction as we did.
Here, we are looking at confluence of the global level: the confluence of generation of diagrams. This in turn require confluence at the local level which gives a notion of "sameness" in diagrams (same normal form).

For that, we will prove that we can perform a partial execution: $\pitchfork_{C \cup D} (\pitchfork_D (\Phi) \sqcup \Phi') = \pitchfork_{C \cup D} (\Phi \sqcup \Phi')$ for some sets of colors C and D. (This was the final lemma in the other setting).

However, this such a partial pre-execution is not possible in general as presented in figure 4.8:

(a) We have $\pitchfork_{\{c\}}(\Phi) = [\mathtt{l} \cdot X]$ and $\pitchfork_{\{c\}}(\pitchfork_{\{c\}}(\Phi) \sqcup \Phi') = [-c(\mathtt{r} \cdot X)] + [\mathtt{l} \cdot X]$, but $\pitchfork_{\{c\}}(\Phi \sqcup \Phi') = [\mathtt{l} \cdot X] + [\mathtt{r} \cdot X]$ which is different. Notice that both $[-c(\mathtt{l} \cdot X)]$ and $[-c(\mathtt{r} \cdot X)]$ need $[X, +c(X)]$ but when executing Φ, Φ' cannot be connected to it anymore.

(b) We have $\pitchfork_{\{c\}}(\Phi) = [X, a]$ and $\pitchfork_{\{c\}}(\pitchfork_{\{c\}}(\Phi) \sqcup \Phi') = [-c(X), b] + [X, a]$, but $\pitchfork_{\{c\}}(\Phi \sqcup \Phi') = [X, a] + [X, b]$ which is different.



(c) We have $\pitchfork_{\{f\}}(\Phi) = [+g(X)]$ and $\pitchfork_{\{g\}}(\pitchfork_{\{f\}}(\Phi) \sqcup \Phi') = [+f(X), a]$, but $\pitchfork_{\{f,g\}}(\Phi \sqcup \Phi') = []$ (no saturated diagrams) which is different.

Figure 4.8: Counter-examples for partial pre-execution.

The problem is that stars from two distinct constellations want to access the same ray. This ray can disappear if it used during execution, breaking in the process connexs in the dependency graph with other constellations that would also interact with it. This problem is reminiscent of the idea of *mutual exclusion* in concurrent programming [15].

We need to design a precondition for which these properties are valid. For that, we will first try to partially execute a constellation, and see the link between the diagrams before and after pre-execution.

We will only consider colors as locations, although this could be easily generalized to arbitrary terms.

Definition 467 (Star Expansion: the compositionality of diagrams)
Given a diagram $\gamma \Downarrow \varphi$, a diagram δ with a vertex v_0 such that $\delta(v_0) = \varphi$, we can buld a diagram $\varepsilon = \mathbf{Exp}_{v_0 \to \gamma}(\delta)$, the expansion of δ along the function $v_0 \to \gamma$ as follows:

- As a function, $\varepsilon := \delta \sqcup \gamma$
- The only difference being the underlying rgraph. In $|\varepsilon|$ we remove v_0 and reconnect every edge that was connected to it to the vertex in γ whose star's

free ray was used to form the edge (since the diagram was contracted, said free ray appeared above v_0 in δ). Intuitively, we injected γ where v_0 as previously. Formally: $|\varepsilon| := ((V_\delta - v_0) \sqcup V_\gamma), E_\delta \sqcup E_\gamma, (\pi_\varepsilon, W_\gamma \sqcup W_\delta, \partial_\delta \sqcup \partial_\gamma)$

$$\pi_\varepsilon(w) = \begin{cases} \pi_\gamma(w) & \text{if } w \in W_\delta. \\ \pi_\delta(w) & \text{if } w \in W_\gamma \text{ and } \pi_\delta(w) \neq v_0. \\ \hat{v} & \text{where } \hat{v} \text{ is the vertex to which the free ray belonged in } \gamma \\ & \text{(that is the first projection of } \delta_{v_0}(w) \in \texttt{IdFreeRays}(\gamma)) \end{cases}$$

Since $\texttt{Exp}_{v' \to \gamma'}(\texttt{Exp}_{v \to \gamma}(\delta)) = \texttt{Exp}_{v \to \gamma}(\texttt{Exp}_{v' \to \gamma'}(\delta))$, this definition can be extended to any function $f : V \subseteq |\delta| \to \texttt{Diags}$ mapping vertices to diagrams. It will then be denoted by $\texttt{Exp}_f(\delta)$.

Remark
This definition and linked lemmas could be generalized to a setting where diagrams are our primitive objects. It would be then reformulated as saying that: diagrams with the same boundaries can be put in the same positions, and that two diagrams with the same normal form (retract) are indistinguishable computationally. This is really similar to what is expected of an open system.

Definition 469 (Disjointness of locations)
Two constellations Φ, Ψ are said to be D-disjoint when there is no edge in $\texttt{Dep}_{\mathbb{C}}(\Phi + \Psi)$ of color D linking the connected components of Φ and Ψ.
Intuitively, this means that we cannot connect stars of Φ and stars of Ψ using rays of color D.
This is written as $\Cap_D(\Phi, \Phi') = \emptyset$.

Lemma (Fundamental Lemma of expansion)
For every diagram γ, there is an f such that $\gamma = \texttt{Exp}_V(\delta)$ iff $\gamma \to^* \delta$.
Note this implies $\Downarrow \texttt{Exp}_V(\delta) = \Downarrow \gamma$.

Proof

- (Easy case) If $\gamma = \texttt{Exp}_f(\delta)$, we reduce all edges internal to $|Im(f)|$ (seeing $Im(f)$ as a non connected subdiagram of γ). There will be no failure (or else one of the diagrams in f was already not correct)
- (Less easy case) By induction on the reduction $\gamma \to^* \delta$. The base case is trivial. For the inductive case, notice that hence doing one step of "fuse", a vertex v gets collapsed on a vertex v': the expansion is then just considering the function $v' \to \varepsilon$ with ε the subdiagram of γ with just v and v' (we basically undo the step). In reality, these two might come from diagrams that we were planning on expanding. The inductive step is thus just glueing these diagrams together when they get contracted on the same vertex. $\square$

Lemma (Expansion preserves correctness)
If δ was correct, then so is $\mathtt{Exp}_V(\delta)$.

Proof
By the fundamental lemma, $\Downarrow \mathtt{Exp}_V(\delta) = \Downarrow \gamma \neq \bot$. □

We finally prove the main lemma of pre-execution. This is done using the same technique that was used to prove Church-Rosser in the previous chapter, albeit a bit more informally here:

Lemma (Fundamental lemma for associatvity)
Let $C \subseteq D$ be sets of colors, and let Φ and Φ' be C-disjoint constellations ($\Cap_C(\Phi, \Phi') = \emptyset$).
There is a bijection $f : \mathtt{CDiags}_D(\pitchfork_C (\Phi) \sqcup \Phi') \simeq \mathtt{CDiags}_D(\Phi \sqcup \Phi')$. Moreover, f and f^{-1} preserve the normal form.

Proof
We show there is a bijection between D-diagrams of $\pitchfork_C (\Phi) \sqcup \Phi'$ and of $\Phi \sqcup \Phi'$, and that said bijection preserves the normal form.

- Take a D-saturated diagram δ on $\Phi \sqcup \Phi'$, and let $R := \left\{ e : v - v' \in E_{|\delta|} \;\middle|\; \delta(v), \delta(v') \in |\Phi|_R, \delta_v(e) : C \right\}$ be the set of C colored edges linking stars of Φ, that we want to reduce to get to $\pitchfork_C (\Phi)$:

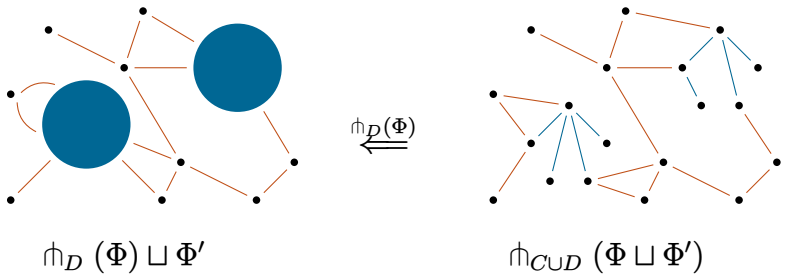


Figure 4.9: Partial execution acts as a partial diagram contraction, which is only possible when Φ and Φ' are not connected by an edge of the condidered colors. The “blow-up" obtained by inverting execution preserves the connexs between rays.

We have $\delta \to_R^* \gamma$, for a certain correct γ (since δ is).

This amounts to reducing maximal connected components of stars of Φ (which forms subdiagrams) in δ.

The difficulty being these subdiagrams might not be C-saturated, and hence reducing to stars in $\pitchfork_C (\Phi)$.

If a certain component was not, then there would be a C colored ray in the reduced star of said component. Then this ray would either:

- be a free ray of color C, which thus would already be free in δ, which in turn would not be C-saturated, contradicting the hypothesis.
- be a connected ray in an edge in δ, which has to be connected to a star in Ψ (else we have not completely reduced the connected component). This is not possible, since they are C-disjoint.

- This other direction of the proof is always true, even without disjointness: take a diagram δ in $\pitchfork_D (\pitchfork_C (\Phi) \sqcup \Phi')$. Every vertex v such that $\delta(v) \in \pitchfork_C (\Phi)$ comes from a diagram γ_v (note that it is crucial here that we consider stars with indices/multiplicity, because there is a need for a 1 to 1 correspondence with diagrams, else a star could come from two different diagrams and this would not work anymore).

  By doing all expansions $v \to \gamma_v$, we obtain a diagram γ in $\pitchfork_D (\Phi \sqcup \Phi')$

It is clear that these two processes, reduction and expansion, are dual to one another and thus we get a bijection between the two sets. Since the expanded form reduces to the other, they have same normal form (this uses confluence of the rewriting system, so the "local confluence" is needed for the "global one"). □

Hole
This proof shows that dealing with stars is not completely natural.
Here we had to go through the real interesting objects, which are the diagrams in themselves. This suggests a future work: a theory of linear realisability based on sort of "open systems", since diagrams are compositional, and in which there would be a lemma stating that a diagram and its reduced are observationally equivalent (this is confluence at the local level).

Corollary (Partial pre-execution)
Let $C \subseteq D$ be sets of colors, and let Φ and Φ' be C-disjoint constellations ($\Cap_C(\Phi, \Phi') = \emptyset$).
We have $\pitchfork_D (\pitchfork_C (\Phi) \sqcup \Phi') = \pitchfork_D (\Phi \sqcup \Phi')$ (up to reordering of indices).

Theorem (Church-Rosser / Confluence)
For any constellation Φ, and C, D two disjoint sets of colors, we have $\pitchfork_D (\pitchfork_C (\Phi)) = \pitchfork_{C \sqcup D} (\Phi) = \pitchfork_C (\pitchfork_D (\Phi))$.

Proof
Using corollary 17 with colors $C \subseteq C \sqcup D$:

$$\begin{aligned} &\mathrm{Ex}_{C \sqcup D}(\Phi) \\ &= \mathrm{Ex}_{C \sqcup D}(\mathrm{Ex}_C(H)) && \text{Pre-execution lemma with } \Phi' = \{\} \\ &= \mathrm{Ex}_D(\mathrm{Ex}_C(H)) && \text{Useless location} \end{aligned}$$

The other case is done symmetrically. □

Generalizing Stellar Resolution to more complicated locations?

As mentionned earlier, we only used colors as location. This was due to the fact that the dynamics become much more complex when dealing with changing locations.

A system with dynamic locations would be a location system considering the term of the polarized ray as the location. In this case, the color becomes redundant, as a $(+a, t)$ could become just $(+, a(t))$.

The first step towards a consideration of stars with dynamic locations, is to try and reexpress the condition on stars being disjoint on terms. This can easily be done, and we will do it here.

Remark
The condition of disjointness that we will define, is, to my knowledge, the best static precondition one can get to ensure the partial pre-execution lemma.
This is not a complete condition: some pairs of constellations might verify the conclusion of the lemma but not the hypothesis. For example, there might be bridges between the two constellation but taking said bridge will always lead to an incorrect diagram, making it "invisible" somehow. This of course cannot be determined statically.

Looking at the proof of the pre-execution lemma, what we needed was to prevent the two constellations to form a bridge using a certain color. This can be expressed in the dependency graph more generally: there has to be no edge between Φ and Φ' on the locations we want the computation to happen.

This is not the case in figure 4.8: the ray $[+c(X)]$ is accessible both from Φ and Φ'.

Definition 476 (Shared rays, or bridges)
We define the set of *shared rays* of the constellations Φ_1 and Φ_2 (also called set of *bridges* between said constellation) with respect to a location L (here in the system of locations of terms $\mathbb{L}(\mathbb{T})$), as the set of edges:
$B_L(\Phi_1, \Phi_2) := \{e : w, w' : v - v' \in \mathtt{Dep}_{\mathbb{C}}(\Phi_1 + \Phi_2) \mid \Phi_1[w] \cap \Phi_2[w'] \subseteq L\}$

Definition 477 (General Disjointness)
Two constellations Φ, Φ' are said to be L-disjoint with L a set of locations when there is no bridge in $B_L(\Phi, \Phi')$.
This would also be written as $\Cap_L(\Phi, \Phi') = \emptyset$.

Example
In figure 4.8, we have $\Cap(\Phi, \Phi') = \{+c(X)\}$ in the two first examples and $\Cap(\Phi, \Phi') = \{-f(X)\}$ in the third one.

From this, there would be the need to adapt the theory of `Flows`, which is work in progress and an adaptation of the theory of Graphings.
We do not see as of today an obstruction that would prevent such an adaptation of being done.

4.2. Proof Structures and Proof Nets inside Stellar Resolution

4.2.1. Encoding proof structures

One of the aim of Transcendantal Syntax was to "reconstruct" logic from computation.

It is then natural to start from linear logic [25] since it is a refinement of intuitionistic logic.

We choose to work with Girard's representation of proofs called "proof-nets"[2].

We remind the reader that a reminder of our alternative definitions of Proof Structure, Proof Net and MLL were explained in the first chapter of the thesis section 2.3.1.

4.2.2. Encoding of Proof Structure into Stellar Resolution

The $\otimes/\parr$ cut-elimination rule pushes cuts to the top of the proof-structure (axioms) and the ax/cut rule identifies some atoms by contraction. It shows that we can see a proof structure as a connex between a permutation of atoms representing axioms and a partial permutation on atoms representing cuts (*cf.* convention 36).
The cut-elimination procedure is then seen as a computation of maximal alternating paths between the graph of these two permutations or equivalently as the complete edge (or topological) contraction of a bipartite graph with two sides ax and cuts.
Cut-elimination is thus done in two parts:

- A rerouting/rewiring (through tensor and parr) where one computes the location where interaction happens.
- Then the actual *elimination* where one plug the program and the cuts.

[2]Our definitions differ from the usual definitions of the literature but are more convenient for the results presented in this paper.

In this section, we will compile Proof Structures into the model of stars. Note that this compilation/interpretation will be the same for both MLL and MLL+MIX since they have the same cut-elimination.

Convention (Encoding Signature)
In order to encode proof-structures, we fix an *encoding signature*.
This is a colored signature $\mathcal{B} := (V, F, \mathtt{ar}, \lfloor\cdot\rfloor)$ with any set of variables such that $X \in V$, function symbols $F := \{\mathtt{l}, \mathtt{r}, \cdot\} \cup \bigcup_{u\in U}\{u\}$, for U a set of elements which be used to represent vertices of proof-structures (we can set $U := \mathbb{N}$).
We have $\mathtt{ar}(u) = 1$ for all $u \in U$, $\mathtt{ar}(\cdot) = 2$ and $\mathtt{ar}(\mathtt{l}) = \mathtt{ar}(\mathtt{r}) = 0$. The symbol $\cdot$ is considered right-associative, *i.e.* $t \cdot u \cdot v := t \cdot (u \cdot v)$.

Similarly to unlabelled proof-structures, constellations are purely "locative": only "physical locations" appearing in a proof-structure $\mathcal{S}$ are translated, without giving any serious meaning to labels. We would like to associate to every atom $v \in \mathtt{Atoms}(\mathcal{S})$ of a proof-structure $\mathcal{S}$ a unique address in $\mathbb{T}(\mathcal{B})$.

The address of v will be a term $v'(t)$ where t is a path encoded as a sequence of $\mathtt{l}$ (left) and $\mathtt{r}$ (right) symbols representing the direction to follow in $\mathcal{S}$ to get from the conclusion $v' \in \mathtt{Concl}(\mathcal{S})$ to the atom v. In other words, we are hard-coding the wire structure of proof-structures directly in axioms.

For convenience, we use an alternative definition of proof-structures (still hypergraph based) but which is *inductive*. This will allow us to define addresses inductively.

Definition 480 (Inductive definition of proof-structures)
Given a proof structure $\mathcal{S}$ with n hyperedges, it has to fall in one of these three cases, which gives an inductive definition for PS:

Base Case: If $n = 1$, it has to be an axiom with two conclusions. We write $S := \mathtt{Ax}_{u,v}$.

Union: It is the union of two proof-structures (with respectively k and $n-k$ hyperedges). We write $S := \mathcal{S}_1 \sqcup \mathcal{S}_2$.

Self: It was obtained from a proof-structure with $n-1$ hyperedges using a $\otimes$, $\parr$, or cut hyperedge on two of its conclusions u (left) and v (right).

We write $S := \mathtt{Tens}^{u,v}(\mathcal{S}')$, $\mathtt{Par}^{u,v}(\mathcal{S}')$ or $\mathtt{Cut}^{u,v}(\mathcal{S}')$ respectively.

Definition 481 (Address of an atom)
We define the *path address* (from a conclusion) $\mathtt{pAddr}_{\mathcal{S}}(v)$ of an atom v in a proof-structure $\mathcal{S}$ inductively (*cf.* definition 480):

- if $\mathcal{S} \in \{\mathtt{Ax}_{v,*}, \mathtt{Ax}_{*,v}\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = X$;
- if $\mathcal{S} = \mathcal{S}_1 \sqcup \mathcal{S}_2$ and $v \in V^{\mathcal{S}_i}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}_i}(v)$;

- if $\mathcal{S} \in \{\texttt{Par}^{v',*}(\mathcal{S}'), \texttt{Tens}^{v',*}(\mathcal{S}')\}$ then $\texttt{pAddr}_{\mathcal{S}}(v) = \texttt{l} \cdot \texttt{pAddr}_{\mathcal{S}'}(v)$ with v the conclusion of the edge (under v');
- if $\mathcal{S} \in \{\texttt{Par}^{*,v'}(\mathcal{S}'), \texttt{Tens}^{*,v'}(\mathcal{S}')\}$ then $\texttt{pAddr}_{\mathcal{S}}(v) = \texttt{r} \cdot \texttt{pAddr}_{\mathcal{S}'}(v)$ with v the conclusion of the edge (under v');
- $\texttt{pAddr}_{\mathcal{S}}(v) = \texttt{pAddr}_{\mathcal{S}'}(v)$ otherwise.

The path address to v is uniquely defined *w.r.t.* to a conclusion $c \in \texttt{Concl}(\mathcal{S}')$ where $\mathcal{S}'$ is $\mathcal{S}$ without cuts, *i.e.* $E^{\mathcal{S}'} = E^{\mathcal{S}} \backslash \texttt{Cuts}(\mathcal{S})$ and the rest is left unchanged. The *address* of v is then defined as the term $\texttt{addr}_{\mathcal{S}}(v) := c(\texttt{pAddr}_{\mathcal{S}}(v))$.

Example
The address of the atom 1 in example 490 is $7(\texttt{l} \cdot X)$ because it is reachable from the conclusion 7 by going to the left premise and the address of the atom 3 is $3(X)$ because it is directly reachable.

Intuition
These will serve as locations, see how $7(X)$ is a form of space, with X that can be carved into multiple subspaces with $l \cdot -$ and $r \cdot -$.

Remark
All adresses are defined relatively to conclusions of structures, this can be annoying sometimes when plugging things because conclusion changes during rerouting.
There might be other ways to defined adresses, but I do not know of any.

Definition 485 (Vertex above another one)
A vertex v is *above* another vertex u (and u is under v) in a proof-structure if there exists a directed (by that we mean from vertices in $\texttt{in}(e)$ to vertices in $\texttt{out}(e)$) sequence of vertices from v to u going through only $\otimes$ and $\parr$ hyperedges.

Proposition
Let $\mathcal{S}$ be a proof-structure. For all $v, v' \in \texttt{Atoms}(\mathcal{S})$ such that $v \neq v'$, we have that $\texttt{addr}_{\mathcal{S}}(v) \neq \texttt{addr}_{\mathcal{S}}(v')$, meaning that all atoms have pairwise distinct addresses.

Proof
By definition, all vertices of $\mathcal{S}$ are distinct, and in particular all conclusions are. Tale $v \neq v'$ two vertices, we consider two cases:

- If v is reachable from a conclusion c, and v' from $c' \neq c$. Then $\texttt{addr}_{\mathcal{S}}(v) = c(\dots) \neq c'(\dots) = \texttt{addr}_{\mathcal{S}}(v')$.
- If v and v' are reachable from conclusion c, then $\texttt{addr}_{\mathcal{S}}(v) := c(t)$ and $\texttt{addr}_{\mathcal{S}}(v') := c(t')$ for some t, t'. Such a t induces a path in the graph going up from conclusions to vertices, by going left on l and right on r. If it were true that $t = t'$, they would induce the same path, ending up in the same vertex and then $v = v'$ which contradicts our hypothesis. So $\texttt{addr}_{\mathcal{S}}(v) = c(t) \neq c(t') = \texttt{addr}_{\mathcal{S}}(v')$. □

Definition 487 (Set of addresses)
We define the *set of addresses* of $\mathcal{S}$:

$$\mathrm{Addr}_x(\mathcal{S}) := \{c(f_1 \cdot ... \cdot f_n \cdot X) \mid c \in \mathtt{Concl}(\mathcal{S}), f_i \in \{\mathtt{l}, \mathtt{r}\}\}.$$

It contains all possible terms $\mathtt{addr}_{\mathcal{S}}(v)$ for any $v \in \mathcal{S}$ (and much more).

We can now express our translation of PS:

Definition 488 (Translation of a proof)
Given a Proof Structure $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$, we can separate it into three parts: the axioms, the cuts, and the "body" (everything else). We will compile these three parts separatly into the model of stars:

- $\Phi_{\mathcal{S}}^{\mathrm{ax}} := \sum_{e \in \mathtt{Ax}(\mathcal{S})} [+v(X), +w(X)]$, with $v \neq w \in \mathtt{out}(e)$.
- $\Phi_{\mathcal{S}}^{\mathrm{cut}} := \sum_{e \in \mathtt{Cuts}(\mathcal{S})} [-v(X), -w(X)]$, with $v \neq w \in \mathtt{in}(e)$.
- $\Phi_{\mathcal{S}}^{\leftrightarrow} := \sum_{e \notin \mathtt{Cuts}(\mathcal{S}), \mathtt{Ax}(\mathcal{S})} \begin{bmatrix} -u(X) \\ +w(l.X) \end{bmatrix} + \begin{bmatrix} -v(X) \\ +w(r.X) \end{bmatrix}$, with $u \neq v \in \mathtt{in}(e)$ and $w \in \mathtt{out}(e)$.

The notation for the internal part, $\Phi_{\cdots}^{\leftrightarrow}$ was chosen because this correspond to a rerouting of locations.
We call the union of the axioms and the rerouting the *vehicle* of $\mathcal{S}$. It is defined as $\Phi_{\mathcal{S}}^{veh} := \Phi_{\mathcal{S}}^{\mathrm{ax}} \sqcup \Phi_{\mathcal{S}}^{\leftrightarrow}$.
Finally, we define the *computational content* of $\mathcal{S}$ as the constellation $\Phi_{\mathcal{S}}^{\mathtt{comp}} := \Phi_{\mathcal{S}}^{\mathrm{ax}} \sqcup \Phi_{\mathcal{S}}^{\leftrightarrow} \sqcup \Phi_{\mathcal{S}}^{\mathrm{cut}}$.

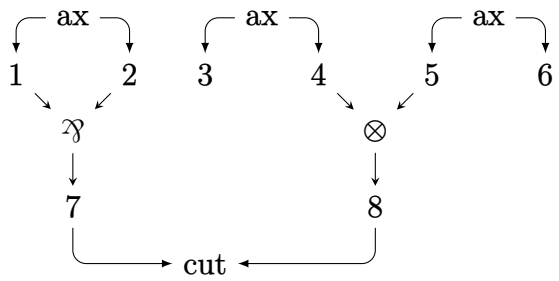
We will often pre-execute the vehicle to get a more compact form of it. What happens is just that the rewiring computes the adress of the conclusions:

Proposition (Compact Vehicle)
We have that

$$\pitchfork(\Phi_{\mathcal{S}}^{veh}) = \sum_{e \in \mathtt{Ax}(\mathcal{S})} [\mu(\mathtt{addr}_{\mathcal{S}}(\overleftarrow{e})), \mu(\mathtt{addr}_{\mathcal{S}}(\overrightarrow{e}))]$$

such that $\mu(c(t)) = +c(t)$ when $c \in \mathtt{in}(e)$ for some $e \in \mathtt{Cuts}(\mathcal{S})$ (it is related to a cut) and $\mu(x) = x$ otherwise.

Example
Here is an example of PS:



Its interpretation after being compacted is the constellation $[+7(\mathtt{l}\cdot X), +7(\mathtt{r}\cdot X)] + [3(X), +8(\mathtt{l}\cdot X)] + [+8(\mathtt{r}\cdot X), 6(X)] + [-7(X), -8(X)]$.
Notice that the star encoding the cut can interact with the atoms whose conclusion of reference are 7 and 8.
We have only one star to represent the cut, exactly as in proof-structures, and it will get duplicated (more precisely, split in two) during execution to connect to sub-formulas, because its $-7(X)$ can be unified with both $+7(l\cdot X)$ and $+7(r\cdot X)$.

4.2.3. Simulation of logical correctness

As proof structures are not always the image of a proof, there is a need to check whether they actually are when it is the case. That is, check if they are proof nets (*cf.* Definition 26).

The expressive power of Stellar Resolution is so that they can also encode Danos-Regnier tests (correctness hypergraphs without axioms), and they are the first GoI model to be able to do so (previous models use the Long-Trip criterion, as seen in the introduction).
This will allow to get an internal and computational form of completeness (encoding typing via passing tests).
They are translated by a constellation which simply reproduces the hypergraph structure of the test. In particular, it has no "complex" dynamics, and is just designed to be plugged to a vehicle and reconstruct the topology of the test.

Example
The 3-ary tensor link relating two inputs u, v to one output w becomes a 3-ary star $[-u(X), -v(X), +w(X)]$.

We translate conclusions of the whole proof-structure by uncolored rays.

Convention
To make tests more readable, they will sometimes be written as *blocks* with inputs (rays of polarity $-$) above and outputs below (rays of polarity $+$). For instance,

$[-u(X), -v(X), +w(X)]$ becomes

$$\begin{bmatrix} -u(X),\ -v(X) \\ +w(X) \end{bmatrix}$$

Links are then connected like tiles or bricks, as in sequent calculus.

We give the following definitions to insist on one fact: *tests only depend on the conclusion formula*, not on the proof structure itself. We will still define tests directly on proof structures afterwards.

Definition 493 (Syntax Tree of a Formula, Sequent)
Let $\vdash \Gamma$ be a sequent of MLL where $\Gamma \subseteq \mathcal{F}_{\text{MLL}}$ and all variables are distinct.
We define the syntax (hyper)tree of an MLL formula A, $ST(A)$ inductively:

- $ST(A_i)$ and $ST(A_i^\perp)$ are vertices labelled by A_i and $A_i^\perp$ respectively;
- $ST(A \otimes B)$ is built by using an hyperedge labelled by $\otimes$ to link the conclusion of $ST(A)$ and $ST(B)$ (as sources) and a new vertex labelled $A \otimes B$ (as target);
- $ST(A \parr B)$ is built by using an hyperedge labelled $\parr$ to link the conclusion of $ST(A)$ and $ST(B)$ (as sources) and a new vertex labelled by $A \parr B$ (as target):

Finally, the syntax hypergraph $ST(\vdash \Gamma)$ of a sequent $\vdash \Gamma$ is defined as the hypergraph disjoint union of all $ST(A_i)$ for $A_i \in \Gamma$ where we add to every conclusion v of the constructed hypergraph unary hyperedges e with $\mathtt{in}(e) = (v)$ and label $\circlearrowleft$.

Note
This is just the lower part of a bunged proof net.

Definition 495 (Switching of a Syntax Hypergraph)
A switching (*cf.* definition 29) φ still applies on $ST(\vdash \Gamma)$ as for correction hypergraphs. We write $ST(\vdash \Gamma)^\varphi$ for the switching φ applied on the syntax hypergraph $ST(\vdash \Gamma)$, but with a slight twist: we do not bung the vertices that become conclusion because of the switching. The result is thus partially bunged.

Convention
In this section, we will test the correctness of a proof only on its axioms, not on the vehicle.
We could test the vehicle in the same way by delocating it so that it becomes the original axiom that it was before the rerouting was done in the vehicle. This is done by adding delocations $[-\mathtt{addr}_{\mathcal{S}}(v), +v(X)]$ for every v conclusion of an axiom in $\mathcal{S}$ and normalizing so that the $-$ part gets plugged.
This would give less elegant theorems for this section so we do not do it.

Definition 497 (Test of a sequent)
The *test* associated with the sequent $\vdash \Gamma$ and the switching φ, giving a syntax tree $ST(\vdash \Gamma)^\varphi = (E, V, \mathtt{in}, \mathtt{out})$, is defined as the constellation $\mathtt{Test}(\vdash \Gamma)^\varphi$ such that $|\mathtt{Test}(\vdash \Gamma)^\varphi|_R := E$ and $\mathtt{Test}(\vdash \Gamma)[e] := e^\star$, where $e^\star$ is defined by a case analysis on $\ell_E(e)$:

- if $\ell_E(e) = \parr_L$ and $\mathtt{in}(e) = (u, v)$, $\mathtt{out}(e) = w$ then
$$e^\star = \begin{bmatrix} -u(X) \\ +w(X) \end{bmatrix} + \begin{bmatrix} -v(X) \\ \end{bmatrix}$$
- if $\ell_E(e) = \parr_R$ and $\mathtt{in}(e) = (u, v)$, $\mathtt{out}(e) = w$ then
$$e^\star = \begin{bmatrix} -u(X) \\ \end{bmatrix} + \begin{bmatrix} -v(X) \\ +w(X) \end{bmatrix}$$
- if $\ell_E(e) = \otimes$ and $\mathtt{in}(e) = (u, v)$, $\mathtt{out}(e) = w$ then
$$e^\star = \begin{bmatrix} -u(X),\ -v(X) \\ +w(X) \end{bmatrix}$$
- if $\ell_E(e) = \circlearrowleft$ and $\mathtt{in}(e) = (u)$ then
$$e^\star = \begin{bmatrix} -u(x) \\ u(x) \end{bmatrix}$$

The *set of tests* associated with the sequent $\vdash \Gamma$ is defined by $\mathtt{Tests}(\vdash \Gamma) := \{\mathtt{Test}(\vdash \Gamma)^\varphi \mid \varphi \text{ is a switching of } ST(\vdash \Gamma)\}$.

Remark
A subtle detail to note is that here we add two stars in the par case, so that a bung is added to the side of the $\parr$ that became a conclusion (here, it is a conclusion but is not bunged, the switching hypergraph is only partially bunged).
We could avoid that and translate every hyperedge to exactly one star (which would be very natural), but this wwould require two types of bungs, as the original conclusions are treated differently (they have memory) than the new ones (who just make things vanish).
If we really want to use one type of bung, we would need to adapt theorem statements and the resulting statements would not be as simple.

Example
We give the compact (executed) form of tests coming from some sequents with a

given switching:

$$\texttt{Tests}(\vdash A, A^\perp)^\varphi \mapsto \begin{bmatrix} -A(X) \\ A(X) \end{bmatrix} + \begin{bmatrix} -A^\perp(X) \\ A^\perp(X) \end{bmatrix}$$

When $\varphi(⅋) = ⅋_L$:

$$\texttt{Tests}(\vdash A^\perp ⅋ B^\perp, A \otimes B)^\varphi \mapsto \begin{bmatrix} -A^\perp(X) \\ (A^\perp ⅋ B^\perp)(X) \end{bmatrix} + \begin{bmatrix} -B^\perp(X) \\ \end{bmatrix} + \begin{bmatrix} -A(X), -B(X) \\ (A \otimes B)(X) \end{bmatrix}$$

When $\varphi(⅋) = ⅋_R$:

$$\texttt{Tests}(\vdash A^\perp ⅋ B^\perp, A \otimes B)^\varphi \mapsto \begin{bmatrix} -A^\perp(X) \\ \end{bmatrix} + \begin{bmatrix} -B^\perp(X) \\ (A^\perp ⅋ B^\perp)(X) \end{bmatrix} + \begin{bmatrix} -A(X), -B(X) \\ (A \otimes B)(X) \end{bmatrix}$$

Note
This is a way of decoupling the logic (tests) from the computational content (vehicle) of the PS.

Definition 501 (MLL test)
Let $\mathcal{S}$ be a proof-structure and φ one of its switchings. The *test* associated with $\mathcal{S}^\varphi$ is the constellation defined by

$$\Phi_{\mathcal{S}}^{\varphi} := \Phi_{\mathcal{S}}^{\text{cut}} \sqcup \sum_{e \in E^{\mathcal{S}^\varphi}} e^\star.$$

We defined the notion on sequents, we now generalize it to PS:

Proposition (Tests of proof structures)
Let $\mathcal{S}$ be a labelled proof structure of conclusion Γ, and φ one of its switchings, such that the vertices are named after their labels. (This is so that the lower part is indeed the syntax tree of the sequent).
We define $\Phi_{\mathcal{S}}^{\varphi}$ as $\Phi_{\mathcal{S}}^{\text{cut}} \sqcup \texttt{Test}(\vdash \Gamma)^\varphi$

Tests for a proof-structure $\mathcal{S}$ are actually designed so that $\texttt{Dep}_{\mathbb{C}}(\Phi_{\mathcal{S}}^{\varphi})$ is (almost) isomorphic to $\mathcal{S}^\varphi$ without axioms, as illustrated in figure 4.10:

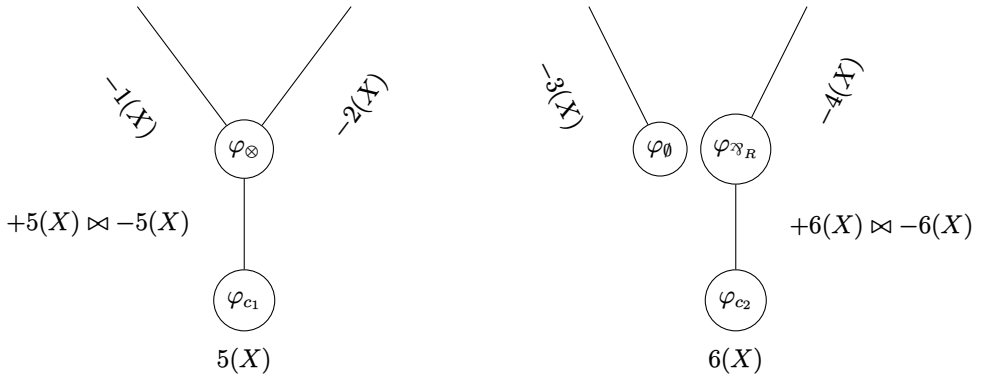


Figure 4.10: Dependency graph of the constellation corresponding to Switching 2 in figure 2.4.

The point is that when constructing the union of a vehicle with a test, we obtain a constellation with a dependency graph structurally corresponding to a Danos-Regnier correctness hypergraph (*cf.* definition 29).

Similarly to what we did for vehicles, we will sometimes *pre-execute* tests to obtain compact tests exactly translating partitions over the set of atoms. Such compact tests are made of stars for each connected components linking inputs for atoms to unpolarised outputs corresponding to conclusions. In other words, the internal structure of tests does not matter, what is important is only the *visible interface* of tests and their organisation in terms of reunion/separation.

Example
We give a simple example: the test of figure 4.10 are:

$$\begin{bmatrix} -1(X) & -2(X) \\ +5(X) & \end{bmatrix} + \begin{bmatrix} -5(X) \\ 5(X) \end{bmatrix} + \begin{bmatrix} -3(X) \end{bmatrix} + \begin{bmatrix} -4(X) \\ +6(X) \end{bmatrix} + \begin{bmatrix} -6(X) \\ 6(X) \end{bmatrix}$$

They simply become, once compactified:

$$\begin{bmatrix} -1(X) & -2(X) \\ 5(X) & \end{bmatrix} + \begin{bmatrix} -3(X) \end{bmatrix} + \begin{bmatrix} -4(X) \\ 6(X) \end{bmatrix}$$

Finally, if we delocalise them so that they can be plugged with the vehicle, and not just axioms:

$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ 5(X) & \end{bmatrix} + \begin{bmatrix} -6(\mathtt{l}\cdot X) \end{bmatrix} + \begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}$$

Beware
Remember that we will only test the axiom here, as explained in convention 496.
In the case where we want to test the vehicle, the test has to be delocated adding a star $[-\mathtt{addr}_{\mathcal{S}}(v), +v(X)]$ for every v conclusion of an axiom in $\mathcal{S}$. (The result being exactly the axiom part before rerouting).
Notice how adding these stars create ambiguities: here, $-5(l.x)$ appears in the delocations, so the $-5(x)$ of the bung is not the only ray that can be connected to the $+5(X)$ output of the tensor.
This makes it so that composition is not associative. The underlying reason is that the rerouting done in the vehicle uses the same locations as the tests so there is a clash.
The easiest ways to deal with that is to either do the relocation after compating, or to rename every location pertaining to tests (except for the unpolarized one in bungs), for example $5 \to q_5, \dots$, giving stars such as $[-q_1(X), -q_2(X), +q_5(X)]$, and $[-\mathtt{addr}_{\mathcal{S}}(v), +q_v(X)]$ (q was chosen to stand for question).

This compactification is used sometimes because we do not want to deal with *internal locations* and all the rewiring, exactly like in the vehicle.

The idea of the simulation of logical correctness is that we would like to make tests interact with a fully positively polarised vehicle to reproduce a Danos-Regnier correctness hypergraph. In stellar resolution, testing is done with execution and we would like to obtain as result $[\{v_1(X), ..., v_n(X)\}]$ with $\mathtt{Concl}(\mathcal{S}) = \{v_1, ..., v_n\}$, ensuring all conclusions can be reached (several connected components induce several stars in the normal form) only once (cycles are designed to produce several $v_i(X)$), hence the associated correctness hypergraph $\Phi^{\varphi}_{\mathcal{S}}$ is connected and acyclic as required by correctness criteria for MLL.

Example
We do an example where we test correctness of a vehicle, not just axioms.
For that, we pre-execute the tests with $[-\mathtt{addr}_{\mathcal{S}}(v), +v(X)]$ for every v conclusion of an axiom in $\mathcal{S}$. Doing this makes so that from the point of view of the tests, the vehicle is now seen as if it was just the axioms.
If we consider the test:

$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ & 5(X) \end{bmatrix} + \begin{bmatrix} -6(\mathtt{l}\cdot X) \\ \end{bmatrix} + \begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}$$

of figure 4.10, we can plug it by union of constellation with the vehicle $[+5(\mathtt{l}\cdot X), +6(\mathtt{l}\cdot X)] + [+5(\mathtt{r}\cdot X), +6(\mathtt{r}\cdot X)]$ and obtain:

$$[+5(\mathtt{l}\cdot X), +6(\mathtt{l}\cdot X)] + [+5(\mathtt{r}\cdot X), +6(\mathtt{r}\cdot X)] +$$

$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ & 5(X) \end{bmatrix} + \begin{bmatrix} -6(\mathtt{l}\cdot X) \\ \end{bmatrix} + \begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}.$$

The first star $[+5(\mathtt{l}\cdot X), +6(\mathtt{l}\cdot X)]$ reunites the two first blocks and we obtain a new block that we can use (among all the others):

$$\begin{bmatrix} -5(\mathtt{r}\cdot X) \\ 5(X) \end{bmatrix}$$

There are two blocks that we have not used yet:

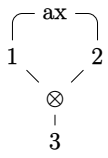
$$[+5(\mathtt{r}\cdot X), +6(\mathtt{r}\cdot X)] + \begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}.$$

The star $[+5(\mathtt{r}\cdot X), +6(\mathtt{r}\cdot X)]$ of the vehicle reunites the other two and we obtain $[5(X), 6(X)]$.
If the vehicle is “too small” or “too big”, it leaves unpolarised ray either in the test or in the vehicle. If we had a star $[+5(\mathtt{l}\cdot X), +5(\mathtt{r}\cdot X)]$ in the vehicle, it would form a cycle with the first block. This cycle yields infinitely many stars $[5(X)] + [5(X), 5(X)] + ...$ by unfolding the cycle to construct as many cyclic diagrams as we wish.

Note
Note that it can already been observed that there is a problem without creating multiple diagrams: the "minimal" diagram is a loop, and so the use of the "self" rule in its reduction indicates there was a problem. A "good" diagram should only use the rule "fuse" and contract edges.

It is important that we consider cyclic diagrams and empty stars: in his original paper [28, Section 2.3], Girard forbade these. However, if we accept his definition, cyclic proof structures $\mathcal{S}$ would be reduced into the empty constellation 0. But if we put such proof-structure next to a correct proof-structure $\mathcal{R}$, we would have that $\mathcal{S} \sqcup \mathcal{R}$ is correct. This not what we want.

As an example, consider the correctness hypergraph of figure 4.11:



$\pitchfork(\Phi^{veh}_{\mathcal{S}}) = [+3(\mathtt{l}\cdot X), +3(\mathtt{r}\cdot X)]$
$\pitchfork(\Phi^{\varphi}_{\mathcal{S}}) = [-1(X), -2(X), 3(X)]$
To consider the vehicle and not just axioms we would add this rerouting: $+[-3(\mathtt{l}\cdot X), +1(X)] + [-3(\mathtt{r}\cdot X), +2(X)]$

Figure 4.11: Incorrect correctness hypergraph for a proof-structure $\mathcal{S}$ and its translation. Notice that the cycle is turned into a computational cycle (a loop in a program).

Infinitely many diagrams can be constructed because of the loop in its dependency graph (an example of loop unfolding is given in figure 4.12). None of these diagrams would be considered correct in Girard's execution thus the result would be 0:

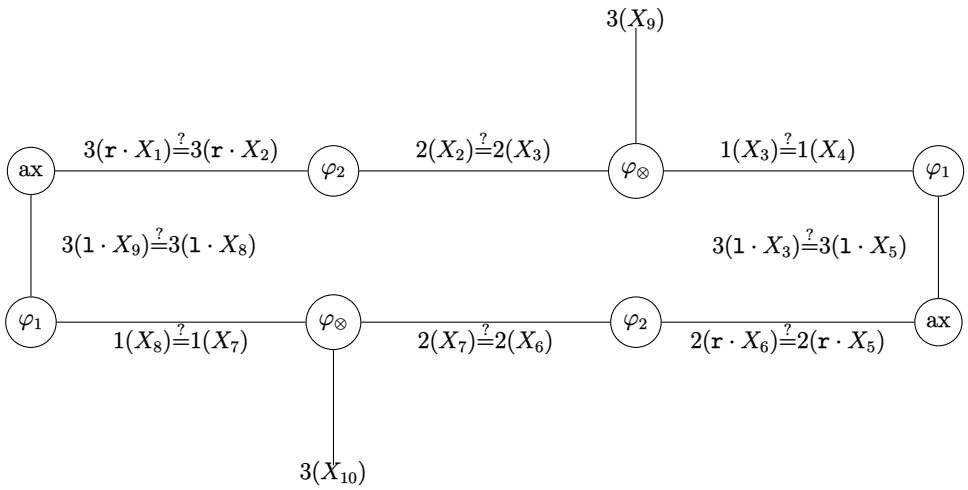


Figure 4.12: Example of a correct and saturated cyclic diagram for the constellation from figure 4.11 actualising into $[3(X_9), 3(X_{10})]$. The cycle can be extended infinitely many times by adding copies of three stars of the constellation.

Our definition of execution makes incorrect proofs visible in the normal form. Cycles yield cyclic diagrams, which are accepted. Since proof-structures are always translated into constellations with trivial equations, such diagrams can always be extended into cyclic diagrams which will be evaluated into infinitely many stars (as in figure 4.12). This makes incorrectness visible in the output of execution.

This problem is actually not new and already existed in previous GoI models. For instance, in Seiller's work, it was necessary to be able to detect cycles. The problem has been solved with a notion of *wager* which is a value associated with proofs indicating the presence of cycles. Here, there is no need for wagers since cycles do not disappear.

Remark
We ultimately forget that they are cycles though, which is somewhat "evil" and might be problematic in the future.

We now get back on track to define the Danos-Regnier correctness criterion in stellar resolution.

Definition 508 (Trial)
We define trials, which are simply the opposition of a test to a program. Given a proof structure $\mathcal{S}$ and a switching φ, we define:

$$\mathrm{T}_{\mathcal{S}}^{\varphi} := \Phi_{\mathcal{S}}^{\mathrm{ax}} \sqcup \Phi_{\mathcal{S}}^{\varphi}$$

These programs are the different plugging of tests and so executing these will serve as attempts at proving correction.

Note
The following lemma would have been much more natural provided we had chosen the hypergraph definition of diagrams (and stars etc...), as it would have been just an isomorphism between two hypergraphs. Here we need to use duality:

Lemma (Structural equivalence of correctness hypergraphs)
Let $\mathcal{S}^{\varphi} := (V, E, \partial, \ell)$ be a correctness hypergraph for a switching φ and let

$$\mathtt{Dep}_{\mathbb{C}}(\mathrm{T}_{\mathcal{S}}^{\varphi}) := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \partial_{\mathfrak{D}})$$

be the dependency graph of its translation using definition 497. There is an hypergraph homomorphism between $\mathcal{S}^{\varphi}$ (seen as an unoriented hypergraph) and the dual hypergraph of $\mathtt{Dep}_{\mathbb{C}}(\mathrm{T}_{S}^{\varphi})$.

Note
See how in figure 4.10, on the side of the $\parr$, $\varphi_{\emptyset}$ is now a conclusion. Taking the hypergraph dual will make it a bung on its free ray $-3(X)$ (which becomes a vertex) and which will be connected with a conclusion and thus become an edge.

Corollary
Let $\mathcal{S}^{\varphi}$ be a correctness hypergraph for a switching φ and let $\mathtt{Dep}_{\mathbb{C}}(\mathrm{T}_{S}^{\varphi})$ be the dependency graph of its translation by definition 497.

One is connected or acyclic if and only if the other is.

Definition 512 (Simple Dependency Graph)
A dependency graph for a constellation Φ is said to be simple when:

- The equations are identities.
- There is at most one edge between two vertices.

In particular, this implies that the dependency graph is a valid diagram when executing over all internal colors.

Lemma (Trials are simple)
$\mathrm{T}_{\mathcal{S}}^{\varphi}$ is a simple constellation for every φ, that is its dependency graph is simple.

Beware
We arrive at a crucial point where we cannot ignore any longer (what can be

considered) a flaw in the current design of TS, and so we must discuss it.
It is the fact that TS does not respect the topology (or geometry) of proofs: because of the self rule, loops disappear, changing the genus of the underlying topological space.
This is a necessary evil when the main focus of study is stars, for else some cyclic diagrams might not reduce to stars (but circles, 1 dimensionnal donuts etc...).
This truly shows something that was hinted before, and that is unfortunately future work: stars are not the right notion that should be the center of study, *diagrams are*. Not only are they compositional, but more importantly they respect the topology since "they are" the topology. Stars should just be seen as generators of diagrams (and thus special cases).
In case of a "vicious circle" in the proof, a diagram containing said cycle would reduce to a diagram with a loop, which is not a single star, and thus contradict the hypothesis of the following theorem, making it's proof simple.
Fortunately, it is still true somewhat by "coincidence": in the presence of a loop, said loop can be iterated to create infinitely many diagrams, which would give an infinite normal form, contradicting the fact that there should be only one element in the normal form.

Theorem (Stellar correctness criterion)
A proof-structure $\mathcal{S}$ such that its set of conclusion is $\texttt{Concl}(\mathcal{S}) = \{v_1, ..., v_n\}$ is MLL-certifiable (*cf.* definition 27) if and only if for all switchings φ, we have:

$$\pitchfork (\mathrm{T}_S^\varphi) = [\{v_1(X), ..., v_n(X)\}]$$

Topologically, that is the fact that every trial defines a unique space (diagram) that is contractible (here, a tree). Note how $v_1(X), ..., v_n(X$ form the boundary of said space

Proof
We show two implications.

($\Rightarrow$) Assume $\mathcal{S}$ is MLL-certifiable. Then there exists a vertex labelling ℓ making $\mathcal{S}$ an MLL proof-net. By the Danos-Regnier correctness criterion (*cf.* theorem 32), it means that for all switching φ of $\mathcal{S}$, we have a correctness hypergraph $\mathcal{S}^\varphi$ which is connected and acyclic. By corollary 18, T_S^φ must be connected and acyclic as well (a tree). We thus have a unique diagram (said tree), which we can actualise to a single star. The only rays left in this actualisation are the unpolarised conclusions of the test, that is $v_1(X), ..., v_n(X)$.

We obtain the normal form $\{v_1(X), ..., v_n(X)\}$.

($\Leftarrow$) **(Expected)** As noted before the proof, this is what the most natural proof would be in the right setting:
Assume $\pitchfork (\mathrm{T}_S^\varphi) = [\{v_1(X), ..., v_n(X)\}]$ for all φ.
For each φ, there was a unique correct diagram since there is only one star in the result. Since it normalized to a star, it had to be contractible, so a tree (else there would have been a loop, note here we assume there is no self rule,

this is the part where we need to respect the topology). This tree has to be isomorphic to the dependency graph, because this graph is a also a correct diagram when a constellation is simple but there is only one correct diagram. By lemma 510 so was the correctness hypergraph, which thus passed the Danos-Regnier tests. So $\mathcal{S}$ was MLL-certifiable

($\Leftarrow$) **(Real)** We discuss a proof of the result in the current setting.

Assume $\Uparrow(\mathrm{T}_S^\varphi) = [\{v_1(X), ..., v_n(X)\}]$ for all φ.

By contraposition:

- The connected components of the dependency graph gives valid diagrams. Since there is only one star in the normal form, there is only one such component.
- Assume this component is cyclic:
  - Either it has no conclusion, and then it yields a closed diagram actualising into the empty star $\{\}$.
  - Or it has at least one conclusion, and then we can iterate it while constructing diagrams to create infinitely many different diagrams (containing said conclusion the number of times we iterated, hence why they are different).

  In both case, the normalisation is different from $[v_1(X), ..., v_n(X)]$, contradicting the hypothesis. Therefore, $\mathcal{S}^\varphi$ must also be acyclic.

This proves that $\mathcal{S}^\varphi$ must be a tree for any switching φ, *i.e.* that $\mathcal{S}$ is MLL-certifiable. □

Remark
We decided to make the boundaries here internal memory by adding bungs, following [28], which makes it so that we cannot compute anymore after passing tests (we have to externally repolarize everything).
It is probably possible to use polarized boundaries by just carefully defining on which locations we compute, and not compute on the boundary locations (or else the diagrams would not be saturated).

The following corollary extends the logical correctness to MLL+MIX and suggests a more general variant which also captures MLL.

Corollary
Let $\mathcal{S}$ be a proof-structure, we have:

- $\mathcal{S}^\varphi$ is acyclic $\Leftrightarrow$ $\mathtt{Dep}_{\mathbb{C}}(\mathrm{T}_S^\varphi)$ is acyclic $\Leftrightarrow$ $|\Uparrow(\mathrm{T}_S^\varphi)| < \infty$;
- $\mathcal{S}^\varphi$ is connected and acyclic $\Leftrightarrow$ $\mathtt{Dep}_{\mathbb{C}}(\mathrm{T}_S^\varphi)$ is simple $\Leftrightarrow$ $|\Uparrow(\mathrm{T}_S^\varphi)| = 1$;

- $\mathcal{S}^{\varphi}$ is connected and acyclic $\Leftrightarrow$ T_S^{φ} normalises into the star of its uncolored rays.

Note
Notice how test behaves in the presence of cuts:
Imagine that we have the proof-structure of example 490. Its executed test for $\parr_L$ is the constellation:

$$\left[\begin{array}{c} -1(X) \\ 7(X) \end{array}\right] + \left[\begin{array}{c} -2(X) \end{array}\right] + \left[\begin{array}{c} -3(X) \\ 3(X) \end{array}\right] + \left[\begin{array}{c} -6(X) \\ 6(X) \end{array}\right] + \left[\begin{array}{c} -4(X), -5(X) \\ 8(X) \end{array}\right].$$

If we execute the constellation corresponding to this proof-structure in order to simulate cut-elimination, we obtain a constellation $[3(X), 6(X)]$ representing the normal form after cut-elimination. We can polarize/activate it to get $[+3(X), +6(X)]$, and this passes the test above.
Conversely, imagine that we have the following tests for axioms:

$$\left[\begin{array}{c} -3(X) \\ 3(X) \end{array}\right] + \left[\begin{array}{c} -6(X) \\ 6(X) \end{array}\right].$$

It works for $[+3(X), +6(X)]$ and also for the example, but provided we put it against not just the axioms, but the entire program.

Note
One of the objectives of TS is to have a finite treatment of everything.
But as one can see, cut-elimination and correctness checking can loop. There might be a way to solve this problem: loops in proof-structures always involve equations between terms containing common variables. For instance, $\varphi := [+1(X), +2(X)] + [-1(X), -2(X), 3(X)]$ can be reduced to $\varphi' := [+2(X), -2(X), 3(X)]$. In φ, the connex between $+1(X)$ and $-1(X)$ makes variables distinct (by α-unification). However, in φ', the connex between $+2(X)$ and $-2(X)$ involve exactly the same variable because they are part of the same star. It means that loops can be detected by looking for equations involving exactly the same variables. There would be a need to make such diagrams incorrect to detect this. Another alternative brought up before would be to detect uses of the self rule, this can be seen in the same example, as $+2(X)$ and $-2(X)$ can be plugged together.

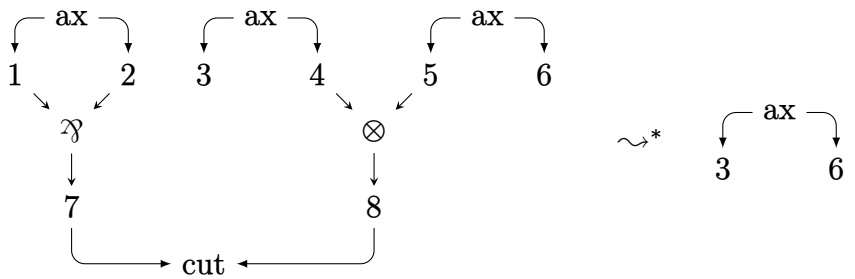
4.2.4. Simulating Cut-Elimination

Cut-elimination in stellar resolution makes the vehicle interact with cuts by execution. It is nothing more than a process of relocation by resolution of addresses (plugging elements in the rewiring together), and the ax/cut case of cut-elimination is the only "true" case of cut-elimination (merging the ax and cut into a long wire).

The idea is simply to execute the translation of the vehicle and cuts of a proof-structure.

We start by looking at examples of execution of such constellations representing proof-structures. We expect diagrams to correspond to maximal paths from two ends of a proof-structure alternating between vehicle and cuts:

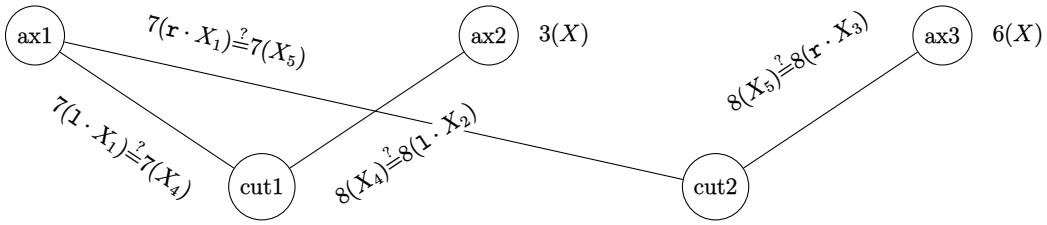
Example (Correct cut-elimination)
We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:



As shown in example 490, the proof-structure $\mathcal{S}$ is translated into:

$$\begin{aligned}\Phi_{\mathcal{S}}^{\mathtt{comp}} := \\ &[+1(X), +2(X)] + [3(X), +4(X)] + [+5(X), 6(X)] \\ +&[-1(X), +7(\mathtt{l} \cdot X)] + [-2(X), +7(\mathtt{r} \cdot X)] + [-4(X), +8(\mathtt{l} \cdot X)] + [-5(X), +8(\mathtt{r} \cdot X)] \\ +&[-7(X), -8(X)]\end{aligned}$$

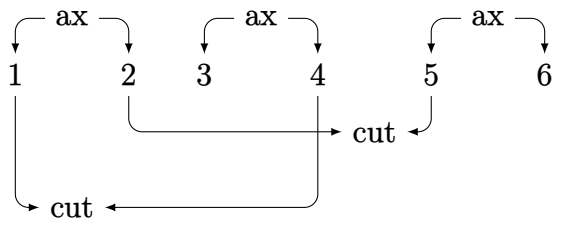
If we do the usual compacting of the vehicle by doing $\Uparrow_{1,2,4,5} (\Phi_{\mathcal{S}}^{\mathtt{comp}})$, we would have obtained this dependency diagram:



Here, because we want to faithfully simulate cut elimination, we will *not* compact the vehicle.
Indeed, notice how:

$$\begin{aligned}\Uparrow_{7,8} (\Phi_{\mathcal{S}}^{\mathtt{comp}}) = \\ &[+1(X), +2(X)] + [3(X), +4(X)] + [+5(X), 6(X)] \\ +&[-4(X), -1(X)] + [-2(X), -5(X)]\end{aligned}$$

Since there are two "non trivial" diagrams, one where the cut is used on the location $\mathtt{r} \cdot X$ and one on $\mathtt{l} \cdot X$ (which gives the two new stars at the bottom by actualisation).
Now, looking at this proof-structure obtained after one step of multiplicative cut-elimination:



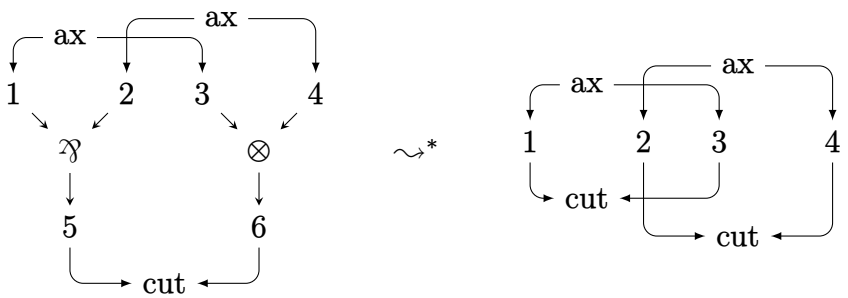
Then its vehicle would be:

$$[+1(X), +2(X)] + [3(X), +4(X)] + [+5(X), 6(X)].$$

And cuts would be $[-1(X), -4(X)] + [-2(X), -5(X)]$.
This would give as computational content exactly the result of the previous execution.

To sum up, in this section we will look at the process of cut-elimination, which starts by *compactifying the cuts with the rewiring, not with the vehicle.*

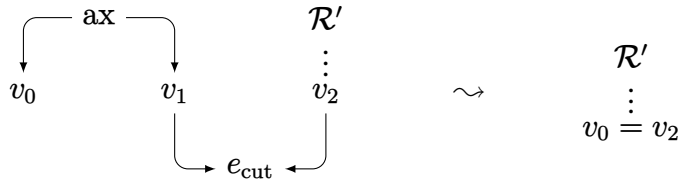
Example (Incorrect cut-elimination)
We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:



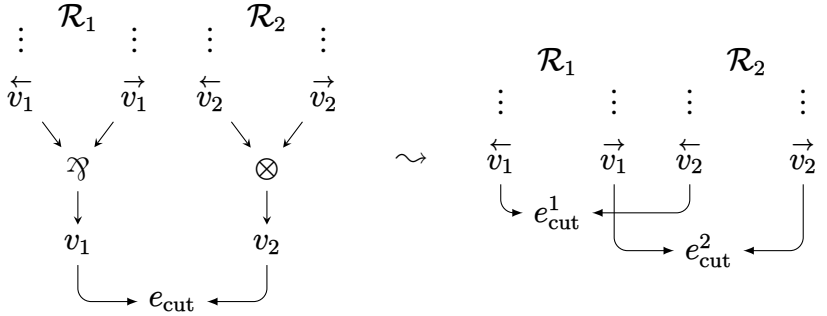
The proof-structure $\mathcal{S}$ is translated (just for this example we also pre-execute the axioms and rewiring) into $\Phi_{\mathcal{S}}^{\mathtt{comp}} :=$

$$[+5(\mathtt{l} \cdot X), +6(\mathtt{l} \cdot X)] + [+5(\mathtt{r} \cdot X), +6(\mathtt{r} \cdot X)] + [-5(X), -6(X)]$$

with the following dependency graph $\mathtt{Dep}_{\mathbb{C}}(\Phi_{S}^{\mathtt{comp}})$:

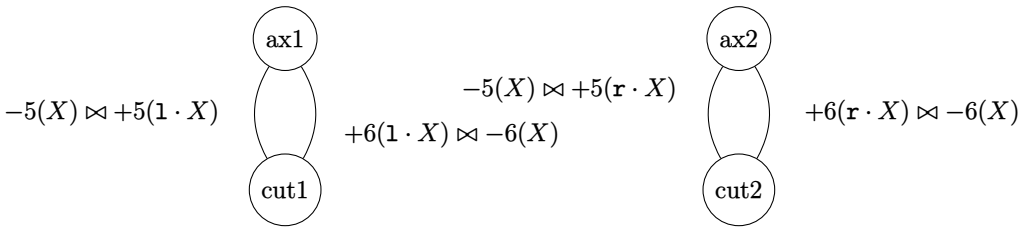


(a) Case of an ax/cut cut with v_0 not connected to a cut.



(b) Case of an ax/cut cut with v_0 connected to a cut.

Figure 4.13: Illustration of the simulation of cut-elimination. We have $\mathcal{R}$ on the left-hand side and $\mathcal{S}$ on the right-hand side.



The cycles in $\mathtt{Dep}_{\mathbb{C}}(\Phi_S^{\mathtt{comp}})$ can be unfolded and it yields infinitely many saturated correct diagrams, all actualising into $\{\}$. We have $\Cap(\Phi_{\mathcal{S}}^{\mathtt{comp}}) = \sum_{i=1}^{\infty}\{\} = \Cap(\Phi_{\mathcal{S}'}^{\mathtt{comp}})$.

Lemma (Simulation of cut-elimination)
Let $\mathcal{R} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be a proof-structure. If $\mathcal{R} \rightsquigarrow \mathcal{S}$ by eliminating an edge e_{cut} that would be translated as $[-v_1(X), -v_2(X)]$, then $\Cap_{v_1,v_2}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) = \Phi_{\mathcal{S}}^{\mathtt{comp}}$.

Proof
Let (v_1, v_2) be the entry vertices of $\mathtt{in}(e_{\text{cut}})$.
Let $e_1, e_2 \in E$ be the hyperedges whose conclusion are v_1, v_2, belonging to respective proof-structures $\mathcal{S}_\infty, \mathcal{S}_\in$.
Notice in the definition of $\|S\|$ how the colors added at each step do not appear in the inductive part. This means that when reducing along said colors, the inductive part will stay the same.
We do a case analysis on $\ell_E(e_1)$ and $\ell_E(e_2)$. The cases are illustrated in figure 4.13.
In all cases, the cut is $[-v_1(X), -v_2(X)]$.

Ax/Cut case We have $\Phi^{\mathtt{comp}}_{\mathcal{R}} := \|\mathcal{R}'\| + [\bullet v_0(X), +v_1(X)] + [-v_1(X), -v_2(X)]$, with $\bullet = +, \oslash$ depending on the remainder of the structure. This reduces using color v_1 to $\Phi^{\mathtt{comp}}_{\mathcal{R}} := \|\mathcal{R}'\| + [\bullet v_0(X), -v_2(X)]$, and to $\|\mathcal{R}'\|[v_2 \leftarrow \bullet v_0]$ using color v_2, since there are only $+v_2$ (no $-v_2$) in $\mathcal{R}'$. This is exactly $\Phi^{\mathtt{comp}}_{\mathcal{R}'}$

Tensor / Parr case We have

$$\begin{aligned}\Phi^{\mathtt{comp}}_{\mathcal{R}} := & \\ & \|\mathcal{R}_1\| + \|\mathcal{R}_2\| \\ & + \begin{bmatrix} -\overleftarrow{v_1}(X) \\ +v_1(l.x) \end{bmatrix} + \begin{bmatrix} -\overrightarrow{v_1}(X) \\ +v_1(r.x) \end{bmatrix} \\ & + \begin{bmatrix} -\overleftarrow{v_2}(X) \\ +v_2(l.x) \end{bmatrix} + \begin{bmatrix} -\overrightarrow{v_2}(X) \\ +v_2(r.x) \end{bmatrix} \\ & + [-v_1(X), -v_2(X)]\end{aligned}$$

This reduces using colors v_1, v_2 to:

$$\begin{aligned}\Phi^{\mathtt{comp}}_{\mathcal{R}} \to^* & \|\mathcal{R}_1\| + \|\mathcal{R}_2\| \\ & + [-\overrightarrow{v_1}(X), -\overrightarrow{v_2}(X)] + [-\overleftarrow{v_1}(X), -\overleftarrow{v_2}(X)] \\ & = \Phi^{\mathtt{comp}}_{\mathcal{R}'}\end{aligned}$$

□

Remark
This proof is of a stronger result and yet much simpler than the one appearing in Eng's thesis [19, §67.9] due to the fact that we did not compactify the vehicle.

Corollary (Simulation of reduction for proof-nets)
Given an MLL+MIX proof-net $\mathcal{R}$ such that $\mathcal{R} \rightsquigarrow^* \mathcal{S}$ with $\mathcal{S}$ in normal form, we have $\Uparrow(\Phi^{\mathtt{comp}}_{\mathcal{R}}) \simeq \Phi^{veh}_{\mathcal{S}}$.

4.3. Realisability in Stellar Resolution

4.3.1. Behaviours and interactive typing

As usual in linear realisability, we can create a notion of semantic typing.

For that, we need to fix a symmetric binary relation between constellations formalising what we mean by "correctly passing a test". For instance, Corollary 19 suggests three such relations we call $\perp^{\text{fin}}$, $\perp^1$ and $\perp^R$ but others can be designed depending on what we want.

The logical intention behind orthogonality relations is that they define linear negations for linear logic.

Definition 523 (Orthogonality)
We define binary relations of *orthogonality* $\perp \subseteq P_L \times P_R$ between two constellations Φ_1 and Φ_2 *w.r.t.* a set of colors $C := L \sqcup R$:

- $\Phi_1 \perp^{\text{fin}} \Phi_2$ when $|\pitchfork_C (\Phi_1 \sqcup \Phi_2)| < \infty$;
- $\Phi_1 \perp^1 \Phi_2$ when $|\pitchfork_C (\Phi_1 \sqcup \Phi_2)| = 1$;
- $\Phi_1 \perp^R \Phi_2$ when $\pitchfork_C (\Phi_1 \sqcup \Phi_2) = \texttt{Roots}(\Phi_1 \sqcup \Phi_2)$ where $\texttt{Roots}(\Phi)$ is the isolated star of collected memories of Φ.

The orthogonal of a set of constellations $\mathbf{A}$ is defined by:

$$\mathbf{A}^\perp := \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$$

for a relation of orthogonality $\perp$.

Note (The choice of multisets/families for programs)
Now that we have defined these orthogonalities, we can explain why the choice was made to consider multisets and not simple sets of programs:
Take as an example the constellations $\Phi_1 = [-a, +a], \Phi_2 = [s, +a] + [-a, t]$. We have that if it was a set then $\pitchfork_a (\Phi_1 + \Phi_2) = \{[s, t]\}$, while here $\pitchfork_a (\Phi_1 + \Phi_2) = [[s, t], [s, t], \dots]$.
This means it is possible to generate infinitely many times the same diagram. If programs were sets, we could mistake the set $\{s, t\}$ for a finite execution, with only one diagram, while it was in fact made with infinitely many.
This would break the definition of strong normalization as having finite execution, and would break orthogonalities based on finiteness.

Proposition (Adjunction)
For all Φ_1, Φ_2, Φ_3 with $L_{\Phi_1} \cap L_{\Phi_2} = \emptyset$, $\perp \in \left\{\perp^{\text{fin}}, \perp^1, \perp^R\right\}$:

$$\Phi_1 \otimes \Phi_2 \perp \Phi_3 \text{ iff } \Phi_1 \perp \Phi_2 :: \Phi_3$$

Proof
The proof is thrice because of the associativity and pre-execution. Let $C = A \sqcup B$ be the color of execution, A the color of Φ_1, B of Φ_2, we do it for one example:

$$
\begin{aligned}
&\Phi_1 \otimes \Phi_2 \perp^{\text{fin}} \Phi_3 \\
&\qquad \text{iff } \Cap_C \left(\Cap_{A\cap B} \left(\Phi_1 + \Phi_2\right) + \Phi_3\right) \text{ is finite} && \big) A \cap B = \emptyset \\
&\qquad \text{iff } \Cap_C \left(\left(\Phi_1 + \Phi_2\right) + \Phi_3\right) \text{ is finite} && \big) \textit{Associativity} \\
&\qquad \text{iff } \Cap_C \left(\Phi_1 + \left(\Phi_2 + \Phi_3\right)\right) \text{ is finite} && \big) \textit{Pre-execution} \\
&\qquad \text{iff } \Cap_C \left(\Phi_1 + \Cap_B \left(\Phi_2 + \Phi_3\right)\right) \text{ is finite} \\
&\qquad = \Phi_1 \perp \Phi_2 :: \Phi_3 && \square
\end{aligned}
$$

From that, one can do realisability as was done in the previous chapters.

4.3.2. Adequacy

In this section, we study two links:

- One between our model and MLL: a proof of conservativity *w.r.t.* the original model of proof-nets for $\perp^{\text{fin}}$ in order to capture MLL+MIX provability and for both $\perp^R$ and $\perp^1$ in order to capture MLL provability. For that purpose, we state *soundness* and *completeness* theorems;
- One between "Usine" and "Usage"(called *adequacy* by Girard) showing that the correctness criterion is sufficient to guarantee a sound use of proofs (interaction by cuts).

We can also choose to prove more traditional properties of soundness and completeness *w.r.t.* proof-structures. This shows that we correctly captured the behaviour of proof-structures, their cut-elimination and their correctness. We start by associating a behaviour to formulas.

Formula labels are interpreted by behaviours where distinct behaviours are associated with occurrences of variables by a function called *basis of interpretation*. This is linked to the famous observation by Girard that propositional variables are actually second order, but with implicit quantifiers (at the "meta" level).

Following previous works of Seiller [55, Definition 46], the behaviours corresponding to formula labels are *localised* formulas: they are defined using the same grammar as MLL formulas, except that variables are of the form $X_i(t)$, where t is a term (here representing the path address described in definition 481) used to distinguish occurrences of a same atomic formula X_i. Two behaviours $X_i(t)$ and $X_i(u)$ with $t \neq u$ represent the same atom at different locations and should correspond to the same behaviour modulo application of a delocation.

Definition 526 (Basis of interpretation)
Given a certain orthogonality, a *basis of interpretation* is a function ω producing a behaviour $\omega(X_i)$ when given an atom $X_i \in \mathcal{F}_{\text{MLLLoc}}$.

This behaviour is localised at a fixed location $+u$ disjoint from any atom name (the letter u stands for universal).

Lemma (Delocations are behaviours)
Given a behaviour A, located at $+u$, $[⫛_u ([+c, -u] + \varphi_u) \mid \varphi_u \in A]$ is a behaviour located at $+c$.

Definition 528 (Basis of interpretation (extended))
Given a basis of interpretation ω, one can generate a function $\Omega(X_i, C, t)$ (also called basis of interpretation) associating a behaviour to an atom X_i, a conclusion C and a term $t \in \mathrm{Addr}_x(\mathcal{S})$ (*cf.* definition 481) encoding the adress of the atom relatively to the conclusion C:
$\Omega(X_i, C, t) := [⫛_u ([+X_i(t), -u] + \varphi_u) \mid \varphi_u \in \omega(X_i)]$.

Remark
The location t serves as a more precise variation of copy indices.

Definition 530 (Interpretation of MLL formulas)
Given a basis of interpretation Ω, we define inductively the *interpretation* $\|A, C, t\|_\Omega$ of A, along Ω, a conclusion C and a term t (encoding the address of A *w.r.t.* a conclusion C) inductively:

- $\|X_i, C, t\|_\Omega = \Omega(X_i, C, t)$;
- $\|X_i^\perp, C, t\|_\Omega = [⫛_u ([-\bar{X}_i(t), -u] + \varphi_u) \mid \varphi_u \in \omega(X_i)]^\perp$;
- $\|A \otimes B, C, t\|_\Omega = \|A, C, \mathtt{l} \cdot t\|_\Omega \otimes \|B, C, \mathtt{r} \cdot t\|_\Omega$;
- $\|A \parr B, C, t\|_\Omega = \|A, C, \mathtt{l} \cdot t\|_\Omega \parr \|B, C, \mathtt{r} \cdot t\|_\Omega$.

We write $\|C\|$ for $\|C, C, X\|$ and extend the interpretation to sequents with:

$$\|\vdash C_1, ..., C_n\|_\Omega := \|C_1\|_\Omega \parr ... \parr \|C_n\|_\Omega.$$

Note we chose to write $\bar{X}$ for the location of $X^\perp$ to clarify when we are speaking of the atom in the syntax and the location in the program.

4.3.3. A complete model of MLL+MIX

We want to prove soundness and completenes of our interpretation for MLL+MIX.

theorem 515 shows that MLL+MIX correctness corresponds in our model to asking for a strongly normalising union between vehicle and test. This gives a hint as to which orthogonality will be of interest to us:

Convention
In this section, we fix $\perp = \perp^{\mathrm{fin}}$.

Instead of the usual soundness property, we prove an extension called *full soundness* [55, Theorem 55] which takes cut-elimination into account. In terms of the adequacy used in realisability interpretations, proving the soundness property corresponds to showing that $\vdash \Phi : \Gamma$ implies $\Phi \in \|\vdash \Gamma\|_\Omega$ for some basis of interpretation Ω.

Here, for $\vdash \Phi : \Gamma$ we only consider constellations coming from proof-nets. Thus, we can interchangeably consider they were translated from proofs in Sequent Calculus.

Lemma (Introduction of Tensor)
Let A, B, C, D be MLL formulas, and Ω be a basis of interpretation:

$$(\mathbf{C} \multimap \mathbf{A}) \otimes (\mathbf{D} \multimap \mathbf{B}) \subseteq (\mathbf{C} \otimes \mathbf{D}) \multimap (\mathbf{A} \otimes \mathbf{B})$$

Proof
We reason on constellations generating these behaviours. We also conflate formulas and their locations (colors) for convenience.
Take $\Phi_{C,A} \in (\mathbf{C} \multimap \mathbf{A})$ and $\Phi_{D,B} \in (\mathbf{D} \multimap \mathbf{B})$, then we have that $\Phi_{C,A} \sqcup \Phi_{D,B} \in (\mathbf{C} \multimap \mathbf{A}) \otimes (\mathbf{D} \multimap \mathbf{B})$.
Let $\Phi_C \in \mathbf{C}, \Phi_D \in \mathbf{D}$, we aim to show that $\Psi :=\pitchfork (\Phi_{C,A} \sqcup \Phi_{D,B} \sqcup \Phi_C \sqcup \Phi_D) \in (\mathbf{A} \otimes \mathbf{B})$.

$$\begin{aligned}
\Psi :=& \pitchfork (\Phi_{C,A} \sqcup \Phi_{D,B} \sqcup \Phi_C \sqcup \Phi_D) \\
=& \pitchfork_{C,D} (\pitchfork_C (\Phi_{C,A} \sqcup \Phi_C) \sqcup \pitchfork_D (\Phi_{D,B} \sqcup \Phi_D)) && \text{pre-execute twice} \\
=& \pitchfork_{C,D} (\pitchfork_C (\Phi_{C,A} \sqcup \Phi_C)) \sqcup \pitchfork_{C,D} (\pitchfork_D (\Phi_{D,B} \sqcup \Phi_D)) && \text{disjoint location} \\
=& \pitchfork_C (\Phi_{C,A} \sqcup \Phi_C) \sqcup \pitchfork_D (\Phi_{D,B} \sqcup \Phi_D) && \text{unused locations} \\
=& \Psi_1 \sqcup \Psi_2
\end{aligned}$$

With $\Psi_1 \in A$ and $\Psi_2 \in B$ by definition of $\multimap$ and thus $\Psi \in A \otimes B$. □

Remark
This is proof does not use any particularity of our model, it is true in any computational model for linear realizability, not just stars.

Corollary (Tensor of Sequent)
Let A, B be MLL formulas, $\Gamma = C_1, ..., C_n, \Delta = D_1, ..., D_m$ be sets of MLL formulas and Ω be a basis of interpretation.
We have the following inclusion:

$$\begin{aligned}
&(\|\Gamma\|_\Omega \parr \|A, A \otimes B, l.X\|_\Omega) \otimes (\|\Delta\|_\Omega \parr \|B, A \otimes B, r.X\|_\Omega) \\
\subseteq &\|\Gamma\|_\Omega \parr \|\Delta\|_\Omega \parr \|A \otimes B, A \otimes B, X\|_\Omega
\end{aligned}$$

.

Proof
Let $C := C_1^\perp \otimes ... \otimes C_n^\perp$ and $D := D_1^\perp \otimes ... \otimes D_m^\perp$, what we need to show is:
$(\|C\|_\Omega \multimap \|A\|_{\Omega, A\otimes B}) \otimes (\|D\|_\Omega \multimap \|B\|_{\Omega, A\otimes B}) \subseteq (\|C\|_\Omega \otimes \|D\|_\Omega) \multimap \|A \otimes B\|_\Omega$.This

is a special case of the previous lemma $\square$

Lemma (Introduction of Mix)
For any behaviour A and B, we have $\mathbf{A} \otimes \mathbf{B} \subseteq \mathbf{A} \parr \mathbf{B}$

Proof
We have $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$, hence we have to show that $\mathbf{A} \otimes \mathbf{B} \subseteq (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$.
Let $\Phi_1 \sqcup \Phi_2 \in \mathbf{A} \otimes \mathbf{B}$ and $\Phi'_1 \sqcup \Phi'_2 \in \mathbf{A}^\perp \otimes \mathbf{B}^\perp$.
We know that $\Phi_1 \perp \Phi'_1$ and $\Phi_2 \perp \Phi'_2$.
We need to prove that $\Phi := \Phi_1 \sqcup \Phi_2 \sqcup \Phi'_1 \sqcup \Phi'_2$ is strongly normalising.
Now, say we construct a diagram $\delta : \Phi$. Once the first star has been chosen, say in Φ_1, we can only use stars in Φ_1, Φ'_1 to stay connected, because of the disjointness of locations of $\mathbf{A}$ and $\mathbf{B}$, and thus $\delta : \Phi_1 + \Phi'_1$. Similarly for Φ_2.
From this we can see that $\texttt{VDiags}(\Phi) = \texttt{VDiags}(\Phi_1 \sqcup \Phi_2) + \texttt{VDiags}(\Phi'_1 \sqcup \Phi'_2)$ and thus $\pitchfork(\Phi) := \pitchfork(\Phi_1 \sqcup \Phi_2) + \pitchfork(\Phi'_1 \sqcup \Phi'_2)$ which is finite. $\square$

Remark
This proof depends on the particular choice of orthogonality we made (linked to the correctness criterion).

Theorem (Full soundness for MLL+MIX)
Let $\vdash \mathcal{S} : \Gamma$ be an MLL+MIX proof-net (that we will identify to its proof in Sequent Calculus) and Ω a basis of interpretation. We have $\pitchfork(\Phi_{\mathcal{S}}^{\texttt{comp}}) \in \|\vdash \Gamma\|_\Omega$.

Proof
We start with the case of cut-free proofs. The proof is done by induction on π:

- Assume we have $\vdash \mathcal{S} : A_i, A_i^\perp$.
  Let $\Psi := [\pitchfork_u ([-\bar{A}_i(t), -u] + \Phi_u) \mid \Phi_u \in \omega(X_i)]$, we would like to show that:

$$\begin{aligned}
\Phi_{\mathcal{S}}^{veh} &\in \|A_i\|_\Omega \parr \|A_i^\perp\|_\Omega \\
&= \|A_i, A_i, X\|_\Omega \parr \|A_i^\perp, A_i^\perp, X\|_\Omega \\
&= \left(\Omega(A_i, A_i, X)^\perp \otimes \|A_i^\perp, C, X\|^\perp\right)^\perp \\
&= \left(\Omega(A_i, A_i, X)^\perp \otimes \Psi\right)^\perp.
\end{aligned}$$

  Let $\Phi_A \in \Omega(A_i, i, X)^\perp$ and $\Phi_{\bar{A}} \in \Psi$.
  We need to show $\left|\pitchfork_{A,\bar{A}} (\Phi_A + \Phi_{\bar{A}} + \Phi_{\mathcal{S}}^{veh})\right| < \infty$, *i.e.* that the axiom strongly normalises with its tests.

Let $Ax := \Phi_{\mathcal{S}}^{veh} = [+A_i(X), +\bar{A}_i(X)]$:

$$
\begin{array}{ll}
& \pitchfork_{A,\bar{A}} (\Phi_A + \Phi_{\bar{A}} + Ax) \\
= & \pitchfork_A (\Phi_A + \pitchfork_{\bar{A}} (\Phi_{\bar{A}} + Ax)) \\
= & \pitchfork_A (\Phi_A + \pitchfork_{\bar{A}} (\pitchfork_u ([-\bar{A}_i(X), -u] + \Phi_u) + Ax)) \\
= & \pitchfork_A (\Phi_A + \pitchfork_{\bar{A},u} ([-\bar{A}_i(X), -u] + Ax + \Phi_u)) \\
= & \pitchfork_A (\Phi_A + \pitchfork_u (\pitchfork_{\bar{A}} ([-\bar{A}_i(X), -u] + Ax) + \Phi_u)) \\
= & \pitchfork_A (\Phi_A + \pitchfork_u ([+A_i(X), -u] + \Phi_u))
\end{array}
$$

pre-execute (since disjoint colo
definition of $\Phi_{\bar{A}}$
de-preexecute
+ rearrange
pre-execute
simple computation

But $\pitchfork_u ([+A_i(X), -u] + \Phi_u) \in \Omega(A_i, A_i, X)$ by definition, and thus normalises when executed against $\Phi_A \in \Omega(A_i, A_i, X)^\perp$.

- Assume we have $\vdash \mathcal{S} : \Gamma, \Delta, A \otimes B$ coming from $\vdash \mathcal{S}_1 : \Gamma, A$ and $\vdash \mathcal{S}_2 : \Delta, B$ via a $\otimes$ rule.
  We have to show, letting $\pitchfork (\Phi_{\mathcal{S}}^{veh}) \in \|\vdash \Gamma\|_\Omega \parr \|\vdash \Delta\|_\Omega \parr \|A \otimes B\|_\Omega$.
  Now remember that $\pitchfork (\Phi_{\mathcal{S}}^{veh}) := \pitchfork (\|\mathcal{S}_1\| + \|\mathcal{S}_2\| + \begin{bmatrix} -A(X) \\ +A \otimes B(l.X) \end{bmatrix} + \begin{bmatrix} -B(X) \\ +A \otimes B(r.X) \end{bmatrix})$.
  By induction hypothesis, we have:
  - $\|\mathcal{S}_1\| \in \|\vdash \Gamma, A\|_\Omega = \|\vdash \Gamma\|_\Omega \parr \|A\|_\Omega = \|\Gamma, A, X\| \parr \|A, A, X\|$.
  - $\|\mathcal{S}_2\| \in \|\vdash \Delta, B\|_\Omega = \|\vdash \Delta\|_\Omega \parr \|B\|_\Omega = \|\Delta, B, X\| \parr \|B, B, X\|$.

  We also have that $\|\mathcal{S}_1\|$ and $\|\mathcal{S}_2\|$ have disjoint locations, with, letting C, C' be the internal colors (those not on the boundary) $\pitchfork_C (\|\mathcal{S}_1\|)$ located at the boundary $+A$ and $\pitchfork_{C'} (\|\mathcal{S}_2\|)$ located at the boundary $+B$ (this crucially uses the fact that we are translating a proof of Sequent Calculus).

$$
\begin{array}{l}
\pitchfork_{C+A+C'+B} (\Phi_{\mathcal{S}}^{veh}) \\
\pitchfork_{C+A+C'+B} \left(\|\mathcal{S}_1\| + \begin{bmatrix} -A(X) \\ +A \otimes B(l.X) \end{bmatrix} + \|\mathcal{S}_2\| + \begin{bmatrix} -B(X) \\ +A \otimes B(r.X) \end{bmatrix} \right) \\
\pitchfork_{C+A} \left(\|\mathcal{S}_1\| + \begin{bmatrix} -A(X) \\ +A \otimes B(l.X) \end{bmatrix} \right) + \pitchfork_{C'+B} \left(\|\mathcal{S}_2\| + \begin{bmatrix} -B(X) \\ +A \otimes B(r.X) \end{bmatrix} \right)
\end{array}
$$

definition
pre-execute
+ useless locatio

Using the induction hypothesis, we get that the left hand side of the disjoint union,

$$\pitchfork_{C+A} \left(\|\mathcal{S}_1\| + \begin{bmatrix} -A(X) \\ +A \otimes B(l.X) \end{bmatrix} \right) \in \|\Gamma\|_\Omega \parr \|A, A \otimes B, l.X\|_\Omega.$$

Similarly for the right hand side:

$$\pitchfork_{C'+B}\left(\|\mathcal{S}_2\| + \begin{bmatrix} -B(X) \\ +A\otimes B(r.X)\end{bmatrix}\right) \in \|\Delta\|_\Omega \parr \|B, A\otimes B, r.X\|_\Omega.$$

(Notice how Γ and Δ are completely unaltered).
The result is thus in the tensor (since $+A\otimes B(l.X)$ and $+A\otimes B(r.X)$ are disjoint). We conclude using corollary 21.

- The case $\vdash \mathcal{S} : \Gamma, A\parr B$ (coming from $\vdash \mathcal{S}' : \Gamma, A, B$) follows from the induction hypothesis and the fact that we have $\|\vdash \Gamma, A, B\|_\Omega = \|\vdash \Gamma\|_\Omega \parr \|A\|_\Omega \parr \|B\|_\Omega$ by definition.
- Assume we have $\vdash \mathcal{S} : \Gamma, \Delta$ coming from $\vdash \mathcal{S}_1 : \Gamma$ and $\vdash \mathcal{S}_2 : \Delta$ (by using the MIX rule). We have to show $\pitchfork(\Phi^{veh}_{\mathcal{S}}) \in \|\vdash \Gamma\|_\Omega \parr \|\vdash \Delta\|_\Omega$.
  By induction hypothesis:
  - $\Phi^{veh}_{\mathcal{S}_1} \in \|\vdash \Gamma\|_\Omega$.
  - $\Phi^{veh}_{\mathcal{S}_2} \in \|\vdash \Delta\|_\Omega$.

  Since the MIX rule places two proofs next to each other, we have that by definition $\Phi^{veh}_{\mathcal{S}} = \Phi^{veh}_{\mathcal{S}_1} + \Phi^{veh}_{\mathcal{S}_2}$.
  And thus $\Phi^{veh}_{\mathcal{S}} \in \|\vdash \Gamma\|_\Omega \otimes \|\vdash \Delta\|_\Omega$ by definition of $\otimes$.
  Finally, $\Phi^{veh}_{\mathcal{S}} \in \|\vdash \Gamma\|_\Omega \parr \|\vdash \Delta\|_\Omega$ by lemma 534.

If the proof has cuts, then by corollary 20, we can execute its translation (a constellation) so that the normal form corresponds to the normal form of the proof. This proof is necessarily cut-free, hence the case of cut-free proofs also applies to this case. □

Remark
Remark that the theorem is a statement about $\pitchfork(\Phi)$. This is because in the current setting of TS, there is no way to distinguish locations /colors used in the internal workings of a program against the colors used in the boundary to interact with other programs.
The simulation of cut-elimination is a theorem in the internal dynamics while this theorem is a theorem about the external behaviour (note that in the end, they are done under the same rules).
A redefinition inspired by open systems would solve this problem.

Hole
Is it possible to create a model of computation where the internal dynamics are ruled with a different set of rules as to the external dynamics? It would of course not be compositional, but it might be interesting nonetheless.

Completeness

Convention
In this section we will use the variant of testing where tests are done against the vehicle (so there is a rewiring) and not just the axiom part.

Lemma
Given a behaviour A such that $A^\perp \neq \emptyset$, then for all $\Phi \in A$, we have $|\pitchfork(\Phi)| < \infty$.

Proof
Let $\Psi \in A^\perp \neq \emptyset$, $|\pitchfork(\Phi)| < |\pitchfork(\Phi + \Psi)|$ because of the inclusion of diagrams, and $|\pitchfork(\Phi + \Psi)| < \infty$ because they are orthogonal. □

Definition 541 (Strong Basis of interpretation)
A strong basis of interpretation ω is a basis of interpretation such that $\omega(X_i)^\perp \neq \emptyset$ for all X_i.

Convention
In this section, we will assume our bases of interpretation are strong. We will thus omit the strongness assumption in the statements.

Lemma
Let Ω be a basis of interpretation and $\vdash \Gamma$ an MLL sequent. Then, we have $\mathtt{Tests}(\vdash \Gamma) \subseteq \|\vdash \Gamma\|_\Omega^{\perp_{\text{fin}}}$.

Proof
Take a test $\mathtt{Test}(\vdash \Gamma)^\varphi \in \mathtt{Tests}(\vdash \Gamma)$ for a switching φ of $\vdash \Gamma$. The proof is done by induction on the numbers of connective in $\vdash \Gamma$.

- If $\Gamma = \{A_1, ..., A_n\}$ where the A_i are formulas X_i or $X_i^\perp$, then there is a single switching φ. Because typing is invariant under execution of the internal locations, we can consider a simplification of tests by fusion:

$$\pitchfork(\mathtt{Test}(\vdash \Gamma)^\varphi) = \sum_{i=1}^{n} [-A_i(t_i), A_i(X)]$$

  Where t_i is the expected encoding of the address of the atom A_i. We would like to show that

$$\mathtt{Test}(\vdash \Gamma)^\varphi \in \|\vdash A_1, ..., A_n\|_\Omega^\perp = \|A_1, A_1, t_1\|_\Omega^\perp \otimes ... \otimes \|A_n, A_n, t_n\|_\Omega^\perp$$

  We show that $[-A_i(t_i), A_i(X)] \in \|A_i, A_i, t_i\|_\Omega^\perp$.
  Let $\Phi_i \in \|A_i, A_i, t_i\|_\Omega$. Because $\|A_i, A_i, t_i\|_\Omega$ is a behaviour, we can use lemma 540 and infer that $|\pitchfork(\Phi_i)| < \infty$.
  Adding $[-A_i(t_i), A_i(X)]$ to a strongly normalising constellation cannot cause divergence, hence we must have $[-A_i(t_i), A_i(X)] \perp \Phi_i$ and $[-A_i(t_i), A_i(X)] \in$

$\|A_i, A_i, t_i\|_\Omega^\perp$. Now, since $\mathtt{Test}(\vdash \Gamma)^\varphi$ is made of a disjoint union of constellations of $\|A_i, A_i, t_i\|_\Omega^\perp$, it follows that $\mathtt{Test}(\vdash \Gamma)^\varphi \in \|\vdash A_1, ..., A_n\|_\Omega^\perp$.

- If $\vdash \Gamma$ is $\vdash \Delta, A \parr B$, then a switching φ of $\vdash \Delta, A \parr B$ is a switching $\bar{\varphi}$ of $\vdash \Delta, A, B$ extended with a left or right selection of premise between A and B, both linked by a $\parr$ connective. We choose the selection of the left premise without loss of generality.
  By the induction hypothesis, we have

$$\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in \|\vdash \Delta, A, B\|_\Omega = \|\vdash \Delta\|_\Omega \parr \|A\|_\Omega \parr \|B\|_\Omega$$

  We would like to show that:

$$\begin{aligned} \mathit{test} \vdash \Delta, A \parr B^\varphi \in \|\vdash \Delta, A \parr B\|_\Omega \\ = \|\vdash \Delta\|_\Omega \parr \|A \parr B\|_\Omega \\ = \|\vdash \Delta\|_\Omega \parr \|A, A \parr B, \mathtt{l} \cdot X\| \parr \|B, A \parr B, \mathtt{r} \cdot X\|_\Omega \end{aligned}$$

  The first important thing to note is that $\|A, A \parr B, \mathtt{l} \cdot X\|$ is just a delocation of the behaviour $\|A, A, X\|$.
  The second is that to our test was added $\begin{bmatrix} -q_A(X) \\ +q_{A \otimes B}(X) \end{bmatrix} + \begin{bmatrix} -q_B(X) \\ \end{bmatrix}$, and then the test was compacted. This cannot interfere in the interaction between programs in the delocation of the behaviour, which are located at $+A \parr B(\dots)$ now instead of $+A(\dots)$ and the test, which by the way, was relocated using the block $\begin{bmatrix} -A \parr B(\dots) \\ +\dots \end{bmatrix}$ so it can interact with it, instead of $\begin{bmatrix} -A(\dots) \\ +\dots \end{bmatrix}$
  This means that the diagrams obtained from an interaction of $\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}}$ with a program in the right location will be (almost) the same as the ones obtained by making $\mathtt{Test}(\vdash \Delta, A \parr B)^\varphi$ interact with a delocated program. The only difference will be in the conclusions, but when compacting the conclusions one will fall back on an isomorphic diagram.
  Thus, from $\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in (\|\vdash \Delta\|_\Omega^\perp \otimes \|A\|_\Omega^\perp \otimes \|B\|_\Omega^\perp)^\perp$ we get that $\mathtt{Test}(\vdash \Delta, A \parr B)^\varphi \in (\|\vdash \Delta\|_\Omega^\perp \otimes \|A \parr B\|_\Omega^\perp)^\perp$.

- If $\vdash \Gamma$ is $\vdash \Delta, A \otimes B$, a switching of $\vdash \Gamma$ is a switching of $\vdash \Delta, A, B$ extended to the additional $\otimes$ connective linking A and B, and $\mathtt{Test}(\vdash \Delta, A \otimes B)^\varphi$ can be defined from $\mathtt{Test}(\vdash \Delta, A, B)^\varphi$ by removing the uncolored rays $A(x)$ and $B(x)$, and adding new stars $[-A(X), -B(X), +(A \otimes B)(X)] + [-(A \otimes B)(X), (A \otimes B)(X)]$. Notice how, when unfolding the definition of $\|\vdash \Delta, A \otimes B\|_\Omega$, it is actually generated by a pre-behaviour E, in the sense that $\|\vdash \Delta, A \otimes B\|_\Omega = E^{\perp\perp}$ for some E. Moreover, this pre-behaviour E is obtained as a disjoint union of some sets, including and A and B. Thus the locations of A and B are disjoint.
  This means that adding the test above cannot create a cycle, and thus the

execution will stay finite.
From the induction hypothesis $\texttt{Tests}(\vdash \Delta, A, B) \subseteq \|\vdash \Delta, A, B\|_\Omega^{\perp^{\text{fin}}}$, one can deduce $\texttt{Test}(\vdash \Delta, A \otimes B)^\varphi \in E^{\perp^{\text{fin}}}$ (modulo some relocations).
Finally, $\texttt{Test}(\vdash \Delta, A \otimes B)^\varphi \in E^{\perp\perp\perp} = \|\vdash \Delta, A \otimes B\|_\Omega^\perp$ since $E^\perp = E^{\perp\perp\perp}$ (corollary 14). □

Remark
These two proofs can be a little bit hand-wavy when it comes to relocations. Since the correctness criterion was simpler to prove on the interaction between the tests and the axioms, there might be a nice way of making them more formal and simple in the following way:

- Prove the equivalent of these, but not for the vehicle, just for the axiom part against the test.
- Prove that delocating the behaviours (which in turn delocates both the test and the vehicle) does not change the interaction

Definition 545 (Proof-like constellation)
The syntax tree $ST(\vdash \Gamma)$ of a sequent induces a set of rays by definition 481 by computing the address of each atom in $ST(\vdash \Gamma)$. We note this set $\sharp\Gamma$. A constellation Φ is *proof-like w.r.t.* an MLL sequent $\vdash \Gamma$ if it is well-formed, *i.e.* a binary star, and $\texttt{Rays}(\Phi) = \sharp\Gamma$.

Example
A constellation which is proof-like *w.r.t.* $\vdash X_1^\perp \parr X_2^\perp, X_1 \otimes X_2$ is

$$[+(X_1^\perp \parr X_2^\perp)(\mathtt{l}\cdot X), +(X_1 \otimes X_2)(\mathtt{l}\cdot X)] + [+(X_1^\perp \parr X_2^\perp)(\mathtt{r}\cdot X), +(X_1 \otimes X_2)(\mathtt{r}\cdot X)].$$

However, the wrong linking

$$[+(X_1^\perp \parr X_2^\perp)(\mathtt{l}\cdot X), +(X_1^\perp \parr X_2^\perp)(\mathtt{r}\cdot X)] + [+(X_1 \otimes X_2)(\mathtt{l}\cdot X), +(X_1 \otimes X_2)(\mathtt{r}\cdot X)]$$

is proof-like as well.

Theorem (Completeness for MLL+MIX)
If a constellation $\Phi \in \|\vdash \Gamma\|_\Omega$ is proof-like *w.r.t.* $\vdash \Gamma$, then there exists an MLL+MIX proof-net $\vdash \mathcal{S} : \Gamma$ such that $\Phi = \Phi_{\mathcal{S}}^{veh}$.

Proof
Let $\Phi \in \|\vdash \Gamma\|_\Omega$ be a proof-like constellation. Since Φ is proof-like, it has only binary stars, and thus can be translated as the axiom part of a proof-structure obtained as the image of the axiom part of a proof structure $\mathcal{S}'$.
We can then reconstruct the bottom part from $ST(\vdash \Gamma)$.
We define now a proof structure $\mathcal{S}$ obtained by considering the union of the two parts, placing the axioms on the right places in $ST(\vdash \Gamma)$ (at this point, the linking can still be wrong).

Since $\Phi \in \|\vdash \Gamma\|_\Omega$, from lemma 543 we know it passes all tests, *i.e.* that for all switchings φ of $\vdash \Gamma$ (equivalently, of $\mathcal{S}$), $\texttt{Test}(\vdash \Gamma)^\varphi = \Phi^\varphi_{\mathcal{S}} \perp \Phi$, excluding the possibility of "wrong linking".
By corollary 19, it follows that $\mathcal{S}$ is acyclic, *i.e.* satisfies the correctness criterion for MLL+MIX. Therefore, $\mathcal{S}$ must be a MLL+MIX proof-net of vehicle Φ. □

These results can also be adapted to get a form of completeness for MLL.

4.4. A quick comparison to π-calculus

4.4.1. Alternative form of executions

In the previous section, we defined a "top-down" notion of execution, $⫛ \Phi$. By top-down, we mean we consider all diagrams given (possibly infinitely many) and then we filter the correct ones.

But as explained earlier, Stellar Resolution equipped with this execution is a computational supermodel: there might be infinitely many diagram and yet we reduce all of them in parallel.

In this section, we quickly discuss alternatives:

- One could define a *bottom-up* execution $\uparrow⫛ (\Phi)$, which computes all correct diagrams in parallel from the ground up.
- One could also define a *contracted* execution, $\Downarrow \uparrow⫛ (\Phi)$, which would do the same thing, but contracts the diagram on the fly (and thus only deal with stars).

A process calculus should be similar to the second one, since it deals with terms, not with geometric objects.

We would like to mention that such a presentiation of Transcendantal Syntax suffers from a well known problem:

Proposition (No confluence for $\uparrow⫛$)
Stellar Resolution with $\uparrow⫛$ as execution is not confluent.

Proof
The problem lies in normalisation: take a star $s := [+a, -a, +b]$, and let $S := [s]$.
$\uparrow⫛_a S$ does not normalize, hence neither does $\uparrow⫛_b (\uparrow⫛_a S)$.
On the contrary, $\uparrow⫛_b (S) = []$, hence $\uparrow⫛_a (\uparrow⫛_b (S)) = []$.
Thus depending on the order of the colors/locations we decide to execute, it is possible to not reach the result. □

Note
Note that this problem is solved in the top down approach, because this non normalizing execution gives an empty constellation [] as result.

There are multiple attempts by Girard to fix this: for example, keep non saturated correct diagrams, but make them "invisible", so that $\uparrow\!\!\Cap_b (S)$ is non-empty and thus will not not normalise afterwards, see [30].

As of today, none are considered to be completely satisfactory, although in this case, there is not really a formal definition of "satisfactory".

4.4.2. A process calculus to compute diagrams

We will now sketch a form of process calculus that can be used to compute diagrams.

Unfortunately, there are many choices that one can make and thus many variants. I propose one here, that is flawed but shows similarities to a π-calculus, and discuss how it could be improved or changed based on needs.

There are two "philosophy" that one could try to make a good calculi to capture the model of computation of stars:

- Either we aim for a computation of all diagrams in parralel. We would also aim for a unique normal form corresponding to the execution of a constellation Φ, having all it's retracted diagrams in parallel.
  This is not really feasible: if one generates diagrams in parallel and reduce them automatically, there is no way to check whether two stars came from the same diagram or different diagrams, because we lost information: we are forced to compute the same diagram multiple times, and so such a model can only be qualitative. But not representing the geometrical object (diagrams), but just its reduced form is very desirable: representing a diagram means having to deal with a graph like object in a process calculi which is unheard of.
- The other option is to aim for a correspondence between executions and diagrams.
  There would then be many normal forms, each one corresponding to a correct saturated diagram.
  This is what we will do here. It will still compute multiple times the same diagram, for similar reasons as explained above, which will be explained after. But here, there is a hope: one might be able to equate externally some reductions, as will be discussed. Since this identification of reductions (which correspond to diagrams) would be done externally, there would not be any "obstructions".

Here is the grammar we will use:

$$
\begin{array}{rrll}
\text{(program) } P ::= & B \diamond F & \mid F & \\
\\
\text{(focus) } F ::= & 1 & & \text{(start)} \\
\mid & *(S) & & \text{(star)} \\
\mid & \circ(S) & & \text{(done)} \\
\mid & \bot & & \text{(failure)} \\
\\
\text{(body) } B ::= & 0 & & \text{(inaction)} \\
\mid & B \parallel B & & \text{(parallel composition)} \\
\mid & !(*(S)) & & \text{(replication)} \\
\\
\text{(stars) } S ::= & -T(\vec{x}) & & \text{(input)} \\
\mid & +T(\vec{x}) & & \text{(output)} \\
\mid & S \mid S & & \text{(rays)} \\
\\
\text{(terms) } T ::= & f(t_1, ..., t_n) \text{ with } f \in F & & \text{(function)} \\
\mid & x \text{ with } x \in V & & \text{(var)}
\end{array}
$$

(Remember that F is the set of function symbols of our first order signature and V of vars).

Convention (Programs)
Our programs will be terms in this grammar of the form $B \diamond 1$ with $B \neq 0$.
They will be executed and change form throughout, to be of form $B \diamond F$.
In the last step of reduction, we will break the form $B \diamond F$ and go to just F, to simplify statements about the normal form of such programs.

Definition 551 (Translation of a star)
Given a star $S = [r_1, \ldots, r_n]$, since in this subsection we consider stars without colors r_i is of the form $+t$ or $-t$ with t a term.
We define from that $\|S\| := r_1 \mid \cdots \mid r_n$ which is in the grammar.

Definition 552 (Translation of constellations)
Given a constellation $\Phi = [S_1, \ldots, S_n]$, we define $\|\Phi\| :=!(*(\|S_1\|)) \parallel \cdots \parallel !(*(\|S_n\|)) \diamond 1$.

Remark
We will directly do our calculus for the version of the model of computation with richer locations than just colors (just polarities and rays), although it is currently work in progress.
Adapting it to the setting with colors is easy.

Example
We give some example of terms in this formal grammar:

- A star: $*(-f(a) \mid +f(x))$
- A program: $*(+g(a,y)) \parallel *(-f(b) \mid +g(a,b) \mid +g(a,c))$

As a form of process calculus we require some rules:

Convention (Structural Congruence)
We have some structural rules:

- $A \mid B = B \mid A$ and similarly for $\parallel$.
- $((A \mid B) \mid \ C) = (A \mid (B \mid \ C))$ and similarly for $\parallel$.
- $P \parallel 0 = P$

There is also α renaming of variables inside $*(S)$.

Note
Our model will have some steps that can be considered "big steps":

- A subsitution, which is computed locally, will be applied globally. One could add extra steps to make the subsitution go up and when it reaches the $*$, it fires and goes down.
  We did not do it because it felt like it would clutter the calculus unnecessarily.
- Similarly, we could add rules to encode the unification algorithm and the computation of a unificator as a small step in the system. This would not only clutter the calculus, but would add extra complications, as one would need to either avoid computing multiple unification at the same time, or to handle a merging of substitutions with potential failure.

We will use the operator $\mid$ of stars as a way to make some "processes" (here, rays) interact.
As a quick example, here is an analogy: the program $\overline{f}\langle a\rangle.0 \mid f(x).0$ in π-calculus, is somehow similar to $+f(a) \mid -f(x)$, with an exchange of information between the two. There are some differences, discussed after our definitions.

Convention
We now assume given a set L of locations where we want the computation to happen.

Definition 558 (Dynamics)
We define the first set of rules (computational one) of the dynamics of our process calculus:

$$(B \parallel !(*(S))) \diamond 1 \rightsquigarrow (B \parallel !(*(S))) \diamond *(S)$$
(start)

$$*(S_L \mid +u \mid -v \mid S_L) \rightsquigarrow * (\theta(S_L \mid S_R))$$
(self unification), with $\emptyset \neq u \cap v \subseteq L$

$$(B \parallel !(*(S_L \mid +u))) \diamond *(-v \mid S_R) \rightsquigarrow (B \parallel !(*(S_L \mid +u))) \diamond *(\theta(S_L \mid S_R))$$
(guarded connected replication), with $\emptyset \neq u \cap v \subseteq L$

$$*(S_L \mid +u \mid -v \mid S_L) \rightsquigarrow \bot$$
(self unification failure), with $\emptyset = u \cap v$

$$*(S_L \mid +u) \parallel *(-v \mid S_R) \rightsquigarrow \bot$$
(connected failure), with $\emptyset = u \cap v$

Where $\theta = \texttt{MGU}(u \stackrel{?}{=} v)$.
These rules are extended via reduction under context.

Note
Unification happens in both rules (unification) and (connected).
This is because we want to compute connected diagrams. The rule (connected) enforces that the resulting star is the retract of a connected diagram.
We chose to allow the (self) rule. To do without, there would be a need to fire multiple unifications in the connected rule.

Remark
The usage of unification, can be seen as a sort of analog of the usual sending of value through channels in π-calculus but with two differences:

- The sending of information is not local to a process, but to a star (so there is a form of globality/binder): we have that $\overline{f}\langle a \rangle.0 \mid f(x).P \mid Q \rightarrow P[x \leftarrow a] \mid Q$ in usual π-calculus. Here, this would reduce to $Q[x \leftarrow a]$ while Q was unchanged before. There is of course an absence of P because there are no "." operator, but technically, the $f(x)$ does become $f(a)$ so the substitution is indeed applied where the P would be, it is just that this $f(a)$ disappears since we only express the boundary of the star here (we could split stars in two and add the locations used internally to it so that the information does not disappear if we wish).
- This exhange of information is symmetric, thus the output/input can both exchange information, for example in $+f(a,x) \mid -f(y,b) \mid S \rightsquigarrow S[x \leftarrow b, y \leftarrow a]$, both the $+$ and $-$ technically received information.
  This is "equivalent" to asymmetry, in the sense that:

- It is more general than asymmetry: $+f(a) \mid -f(x)$ is asymmetric, it sends a to x and gets nothing back.
- Symmetry can be encoded in the regular π-calculus by doing 2 steps of communication (one in each direction) thanks to . as a way of sequencing.

In a sense, stars could be seen as an alternative form of the usual notion of process, with no "depth" since there is no continuation "." (so it is weaker), but with a symmetric $\mid$ that makes the communication over channel symmetric and in parallel, which does not work on simply names, but on terms, with a complex system of unification (so it is more general). Note unification seems to be related to a sort of "recursive" form of the $[n = m]$ operator that exists in some π-calculi.

Hole
There might be a way of adding a . operator to generalise the model, although it is unclear if this can be done without breaking the desired associativity.

Definition 562 (Dynamics: rules to end computation)
We define the second set of rules (the termination one) as follows:

$$
\begin{array}{rl}
*(S) \rightsquigarrow \circ(S) & \text{(saturation), when } \forall s \in S, s \cap L = \emptyset \\
B \diamond \bot \rightsquigarrow \bot & \text{(cleanup: failure)} \\
B \diamond \circ(S) \rightsquigarrow \circ(S) & \text{(cleanup: success)}
\end{array}
$$

These are extended to be usable under context as well.

Note
The rule (cleanup: failure) forces that when failing we reach a normal form $\bot$. This avoids "dumb" ways of not terminating. One could, for example, generate a diagram, make it incorrect via failure, and then start a new attempt, recomputing the same diagram, failing again etc...

Proposition (Normal forms)
The normal forms of programs are of two possible shapes:

- Either $\circ(S)$, and then S is the boundary of a saturated correct diagram.
- Or $\bot$.

Remark
It is completely possible that the program have non terminating executions. This happens in particular when there are infinitely many correct diagrams that are not saturated.
We could also make an alternative calculus where there are no failure. We would then have stuck programs of the form $B \parallel *(S)$ with S the boundary of a correct diagram that is not saturated, and cannot be extended without breaking compatibility.

Finally, we could add a rule $P \rightsquigarrow 0$ to compute no only the maximal correct diagrams, but all correct diagrams.

Now that we defined our calculi, we get back to what was said in the introduction of this subsection: what we would truly want of course, is to compute the execution of a constellation with such a process calculi.
There are big obstructions to doing this, that we discuss now.

Remark (The problem of construction)
Imagine you are attempting to build a diagram from the ground up, with the following shape:

If both vertices are already put on paper, there are two ways one could make it, by first pluggin the top edge and then the bottom, or the opposite.
There are thus two possible reductions giving the same diagram. The reductions of the calculi correspond to a notion of "construction" of a diagram.
Thus we only have a correspondence between $s \in \Downarrow \Phi$ and reductions *equated modulo construction* of its translation $\|\Phi\|$ in the process calculus. (Note that this correspondence is quite annoying to state formally).

There is thus a need to study "2-cells" between reduction step that can be done commutatively.

Definition 567 (Construction of a graph)
Given a graph $G = (V, E, s, t)$, a construction is a sequence of edge and vertices $s_1, \dots, s_n$, such that:

- (Totality): every vertex v and edge e appears exactly once.
- (Justification): for every edge $e = s_i$, there are $j, k < i$ such that $\partial(e) = \{s_j, s_k\} \subseteq V$.

Finally, we will denote by $c(G)$ the number of constructions of G.

Although we discovered the concept of construction on our own, it is quite a simple concept so it already existed in the litterature, even though there does not seem to be a lot of papers on the subject.
There are currently no known explicit formula for $c(G)$, even if some were computed for specific families of graphs: see [39].

Note that the model defined above actually uses a variant of the notion of construction: the diagram is kept connex at all times. This gives the following definition:

Definition 568 (Connected Construction of a graph)
Given a graph $G = (V, E, s, t)$, a connected construction is a construction $\vec{s}$ such that $s_0 \in V$ and for every $s_i \in V$ with $i \neq 0$ we have that $s_i \in \partial(s_{i+1})$ with

$s_{i+1} \in E$.
We will denote by $W(G)$ the number of connected constructions of G.

This definition we did not find in the litterature, it is an open problem to find a formula or algorithm to compute $W(G)$ given a certain G.
In particular, it would be interesting to be able to compute $W(G)$ inductively while computing a specific construction since computing connected constructions is what the calculus is doing.
It would be even better to be able to identify which steps of reduction commute together and which steps do not, and findi effective strategies (maybe even key steps when doing the computation) where one is given the choice to compute a diagram or another, so that one can choose in all possible reductions some canonical ones as to get exactly one canonical reduction per construction. This is a big and seemly quite complex open problem.

We finish this section with an explanation on how to make the correspondence a formal statement.

Statement of the correspondence

Convention
When given a sequence $\vec{s} = s_1, \dots, s_n$, we will write $t(\vec{s}) = s_n$ the last element of the sequence.

Definition 570
Given a program $P := B \diamond 1$, we call a good execution a maximal sequence $\vec{s}$ of $\rightsquigarrow^*$ reductions of P with $t(\vec{s}) \neq \bot$.

Lemma
Given a good execution $\vec{s}$, we have $t(\vec{s}) = \circ(S)$ for a certain S.

The next definition use the equivalent for rgraphs of definition definition 568.

Definition 572 (Construction of a diagram)
A construction of a diagram δ on Φ is given by the pair $(\vec{s}_c, \delta)$ with $\vec{s}_c$ a construction of $|\delta|$.

Proposition
This is easily seen to be equivalent to the data of a strictly increasing sequence of diagrams $\delta_0 \sqsubseteq \delta_1 \cdots \sqsubseteq \delta$ such that δ_i, δ_{i+1} differ only by one edge or one edge and vertex.

Proposition
Given Φ a good execution $\vec{s}_e$, we can associate to it a unique pair $(\vec{s}_c, \delta)$ with δ a diagram such that $\vec{s}_c$ is a construction of δ and $\circ(\|\Downarrow \delta\|) = t(\vec{s}_e)$.

The process on how to do it is sketched below, and is quite cumbersome, which is the reason we consider this subsection as being a sketch only.

Definition 575 (Reconstructing a construction from a good execution)
It is important to note that we will cheat a bit: the calculus can reorder its body by structural rules, and so can the constellation Φ since it is tehcnically a multiset. We assume we always keep track of indices, so that when using an instance of a ray, we can always tell from which star/instance of a star it came from.
We proceed inductively by building a sequences of diagrams.
The first rule applied in the sequence has to be (start). We define δ_1 to be the diagram on 1 vertex v with $\delta_1(v)$ the index of the star used in the rule (start). Now assume we are given a δ_i, there are three cases:

- If the next rule is (guarded connected replication), then δ_{i+1} is the same as δ_i with a new vertex a, a new edge $e : w, w' : a - a'$, linking a to the a', such that he instance of v used in the unification was originally a ray of this a'. Now δ_{i+1} associates to a the index of the star $S_L + u$, and associates to w, w' respectively the terms u, v.
- If the next rule is (self unification), then δ_{i+1} is δ_i with a new edge $e : w, w' : a - a'$ with a the vertex associated to the term u, a' associated to the term v. We extend δ_i such that $w \to u$ and $w' \to v$.
- If the next rule is (saturation), then we stop.

The other cases are not possible because we do not end up on $\bot$
We write τ the function wich associates to the good execution $\vec{s_e}$ the pair $(\vec{s_c}, \delta)$.

Proposition
Let Φ be a constellation. The function τ defined previously is a bijection between the good executions of $\|\Phi\|$ and the constructions of diagrams in $\texttt{CDiags}(\Phi)$, such that for any good execution s_e we have $\circ(\|\Downarrow \tau(\vec{s_e})\|) = t(\vec{s_e})$.

Proof
We show that τ is a bijection by explaining how to reverse the processus: given a construction of a correct diagram $(\vec{e_c}, \delta)$, follow sequentially $\vec{e_c}$:

- It has to start with a vertex, apply rule (start) with this vertex
- When reading an edge $e : w, w' : a - a'$, use rule (self unification) on the terms $\delta_a(w), \delta_{a'}(w')$.
- When reading a vertex a, read also the next edge $e : w, w' : a - a'$, use rule (guarded connected replication) with $\delta(a)$ aas a star in the body, and the terms $\delta_a(w), \delta_{a'}(w')$.
- Finish by using (cleanup: success).

This clearly gives back the original sequence. □

Hole
If one wants to implement this on a real computer, the naive idea is to find an algorithm to check whether two reductions are the same or not. This must be "hard", since it is basically a form of the graph isomorphism problem (which is currently in it's own complexity class).

4.5. Link with Flows

We will now discuss the link between flows and stars: it seems that stars can be seen as a form of hypergraph generalisation of flows.

Definition 578 (Interpretation)
Given colors a, b, we define the interpretation of flow $f := u \to v$ as the star $\|f\|_{ab} := [-a.u, +b.v]$.
This can be extended to wirings, giving rise to a constellation $\|F\| := \{\|f\| \mid f \in F\}$.

Theorem (One step simulation)
Given a wiring F and a wiring G, we have that $\Cap (\|F\|_{ab} + \|G\|_{bc}) = \|F.G\|_{ac}$.

Proof
The proof is easy, one just has to notice that the only possible diagrams are of the form $a \to b \to c$ for the 1-step case. The execution formula is similar. □

This result allows to encode the execution formula by taking $c = a$, which creates a sort of feedback loop.

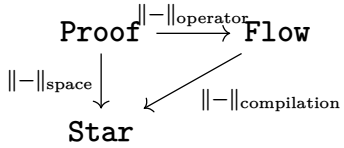
Theorem (Execution of flows)
Given two flows F, G, we have:

$$\begin{aligned}\|\mathrm{Ex}(F,G)\|_{io} = \Cap_{a,b} (& \\ & \|F_{ib}\| + \|F_{ao}\| \\ & + \|G_{ia}\| + \|G_{bo}\| \\ & + \|F_{a,b}\| + \|G_{b,a}\|)\end{aligned}$$

Remark
We could probably get a finer theorem in a setting with richer locations than colors.

This is an easy way to show that these give rise to a big model of Linear Logic (not just MLL): they are more expressive than flows, so we can interpret all proofs by flows and compile flows into stars. Note it might be possible that stars cannot interpret second order, as it is the only part of logic that really uses arbitrary locations and not just the "outermost layer" (which is equivalent to using colors).

This leads to something a bit unexpected: there are two ways to interpret proofs inside Transcendantal Syntax: the one defined in this section, where we interpret proofs as constellations, in a "Danos-Regnier" style (we denote it by $\|-\|_{\text{space}}$), and the one where we interpret it by a wiring, in a "Long-Trip" style ($\|-\|_{\text{operator}}$) and then compile said wiring to a constellation:



Hole
It would be interesting to compare the two: both have *very similar interpretations*, but they are *tested in very different ways.*
One could try to do a program transform $\|-\|_{\text{trans}}$ that would take as input a proof encoded as a space and output its encoding as a constellation of "operators" (binary stars), but this process feels like it lost information:
The axiom at A, which is

$$Ax := [+A(x), +A^{\perp}(x)]$$

Would become something like

$$[-a(A(x)), +c(A^{\perp}(x))] + [+a(A(x)), -c(A^{\perp}(x))]$$

Which becomes two independent operators, while in Ax there was a sort of "binding": say we want to apply a substitution θ on Ax, in the encoding, we would need to apply it to *both operators.*

Remark
In this comparison, strange things happened: alternation "disappeared" while generalizing (we get it back via the encoding, but it does not appear in the notion of diagram).
Another weird thing is the status of the polarities $+$ and $-$:

- In flows, they seem to encode a notion of "input/output", as flows can be seen as "operators".
- In stars, this is not the case: the star $[+a.x, +a.x]$ represent an axiom (seen as a sort of "wire"), its more akin to a polarity than an input/output.

5. Understanding Trefoil in Interaction Graphs

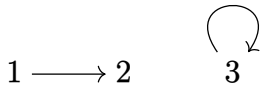
This chapter is a very exploratory one, where we will try to discuss and propose solutions to solve the previously mentionned problem of the disappearance of circuits in interaction graph (this is reexplained in more details at the beginning of the section). What I learned while doing this, and the point of view that I will try to defend is that *geometry* (more precisely, topology) is what truly matters in these models. And that the wager section 2.5.5 is a sort of patch that allows to save some geometrical information that went missing when computation is not respecting the geometry of programs.
With this in mind, it might be possible one day to define a more abstract notion of model of computation, in line with Seiller's notion [58] and use tools coming from some branches of mathematics (homotopy theory? category theory?) to extract informations about programs.

5.1. A categorical perspective on interaction graph

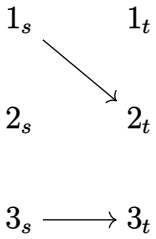
We first study a categorical definition of interaction graphs. We will reintroduce concepts of categorical GoI as we use them. Nonetheless, I give a good and concise reference on the subject which helped me a lot at the beginning: Shirahata's tutorial [61].

Definition 584 (Obip representation of Graphs)
A directed graph $G = (V, E, s, t)$ can be represented by a bipartite graph $\hat{G} = (V_s \sqcup V_t, sE, s', t')$ where we take the disjoint union of V_s and V_t, which are two identical copies of V (named differently through indexes s/t) and using canonical injections $i_s : V \to V_s \sqcup V_t$ and i_t: $s'(e) := i_s(s(e))$ and $t'(e) := i_t(t(e))$.
We call such a graph (representing another) an oriented bipartite graph (obip). They will be considered to be equal up to renaming of edges (graph isomorphism but with same vertices).

Example
Here is a simple example of an interaction graph and it's obip representation:



(a) An interaction graph



(b) The obip representation of the graph

Notation
In this chapter, given a set X, we will use the notation X_s and X_s to designate copies of X whose elements have indexes s, like x_s.
This is use to make some sets disjoints. Indeed, we will be considering in this chapter obip graphs as morphisms $X \to X$, with X a set.
Such a graph will be given as set of vertices $X_s \sqcup X_t$. To differentiate between use in this chapter, as naming convention, indices s/t to indicate that a set is a renmaing of another with said indices.

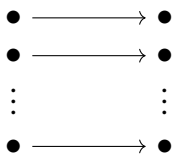
This naturally leads to the definition of the following category:

Definition 587 (IOGrph)
Objects: Any set.

Morphisms: $G : A \to B$ is an obip graph whose edges have their source in A_s and target in B_t.

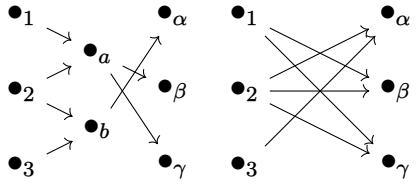
Composition: The composition of $G \in \mathrm{Hom}_{\mathbf{IOGrph}}(\mathrm{A}, \mathrm{B})$ and $H \in \mathrm{Hom}_{\mathbf{IOGrph}}(\mathrm{B}, \mathrm{C})$ is given by glueing the graph G and H along B, and then computing the paths of length 2 in this graph (and forgetting about everything else). The result is the bipartite graph whose edges are the paths of length 2 with one edge in G followed by one edge in H.

Identity: The identity on X, written $1_X : X_s \to X_r$ is the following graph:



(Notice how we use the previously defined convention; by calling X_s and X_t the source and target set of the identity.)

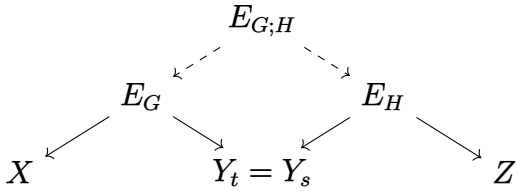
Example
The sequence of morphisms $\{1, 2, 3\} \to \{a, b\}$ and $\{a, b\} \to \{\alpha, \beta, \gamma\}$ shown on the left side of example 588, once composed,give the graph on the right hand side of the same Figure.



(a) Two morphisms (b) Paths of length 2

Remark (Lack of empty morphism)
Note how $\text{IO}\mathbf{Grph}$ does not have any graphs without I/O other than the empty graph: $\text{Hom}_{\mathbf{IOGrph}}(\text{A}, \emptyset) = \text{Hom}_{\mathbf{IOGrph}}(\emptyset, \text{B}) = \{0\}$.

Observation (As a Span)
The category $\text{IO}\mathbf{Grph}$ is actually almost a "Span category": it is similar to the 1-truncation (also called decategorification or homotopy category) of $\mathbf{Span}(\mathbf{Set})$. Indeed, an obip graph can be represented as a span with disjoint feet (so not exactly a span category). Composition is then given by pullback, creating an arrow for every pair of "composable arrows":



Note (The problem of localisation)
This "almost", in "almost a span" is the reason why dealing with interaction graph categorically can be really bureaucratic.
Indeed, in the composition we end up having to take the pullback on $Y_t \simeq Y_s$, there is thus a need to use this identification (called a delocation) before performing the actual glueing. Delocations are thus used everywhere.
The convention of writing s/t as indices, such as in Y_s instead of just Y, allows to have disjoint feet, and we put under the rug the delocations by pretending $Y_s = Y = Y_t$. Note also that the result of composition still has disjoint feet for the same reason: X_s and Z_t.

Proposition
The category $(\text{IO}\mathbf{Grph}, \sqcup, I)$ is monoidal.

Remark
The category $\text{IO}\mathbf{Grph}$ can be seen as a form of "proof-relevant" version of the category $\mathbf{Rel}_+$, the categories of relation, which is itself a generalisation of the category **PartFunc** of partial functions. These are well-known models of GoI, see [31].

The category $\text{IO}\mathbf{Grph}$ can also be equipped with a trace, to make it a model of GoI. We define two very similar constructions, that can be used somehow interchangeably, because the paths they induce do not differ:

Definition 594 (Fuse)
Given an obip graph $G : I \otimes X_l \to O \otimes X_r$, let u be the forgetful function: $u(x_s) = u(x_l) = x$ and $u(v) = v$ otherwise, we define the graph $G^{\square X}$ (read as "fuse X")by $V_{G^{\square X}} = I \sqcup O \sqcup X$, $E_{G^{\square X}} = E^G$, and

$$s^{G^{\square X}}(e) = u(s^G(e)), \qquad t^{G^{\square X}}(e) = u(t^G(e))$$

Definition 595 (Feed)
Given an opib graph $G : I \otimes X_l \to O \otimes X_r$, we define the graph $G^{\circlearrowleft X}$ (read as "feed X")by $V_{G^{\circlearrowleft X}} = V_G$, $E_{G^{\circlearrowleft X}} = E_G \sqcup E_{1_X}$ and

$$s_{G^{\circlearrowleft X}}(e) = \begin{cases} s_G(e) & \text{if } e \in E_G \\ t(e) & \text{if } e \in E_{1_X} \end{cases}, \qquad t_{G^{\circlearrowleft X}}(e) = \begin{cases} t_G(e) & \text{if } e \in E_G \\ s(e) & \text{if } e \in E_{1_X} \end{cases}$$

Observation
These two constructions are not really different from the input/output point of view: $\forall i, o \in I, O : \text{Paths}_{i \to o}(G^{\square X}) = \text{Paths}_{i \to o}(G^{\circlearrowleft X})$.
Thus we will use them somehow interchangeably in drawings, particularly because it can be hard to represent fuse while it is easy to make nice drawings for feed.

We can now define a categorical trace in $\text{IO}\mathbf{Grph}$.

Theorem ($\text{IO}\mathbf{Grph}$ is Traced)
Given $G \in \text{Hom}_{\mathbf{IOGrph}}(\text{I} \otimes \text{X}, \text{O} \otimes \text{X})$, we define $\text{Tr}^X_{I,O}(G)$ by:

$$V^{\text{Tr}^X_{I,O}(G)} = A \sqcup B, \qquad E^{\text{Tr}^X_{I,O}(G)} = \bigcup_{i \in I, o \in O} \text{Paths}_{i \to o}(G^{\square X})$$

with the source and target maps defined by the lifting to paths of the source/target of edges.
The category $\text{IO}\mathbf{Grph}$ endowed with this operation is a traced monoidal category.

Proof
Since the category is symmetric monoidal, we have fewer axioms to verify. The proof is really easy to check visually but we still do it in details to show exactly what properties are used where.
We take the convention e_{AB} means an edge is from A to B.
Consider $f : I \otimes X \to O \otimes X$, we will have to prove that two traced graphs are equals. We will use regular expressions to compare the paths $I \to O$ in the graphs before they are traced/composed (we will designate by "the shape of the graph" the "geometric" graph obtained by doing all necessary glueing but not computing the paths). Having the same paths, the graphs, once traced, will have same edges

and thus be equal.

- **Naturality in I:**
  Given $g : L \to I$, we need to prove $\mathrm{Tr}^X_{I,O}(G)(g \otimes 1_X; f) = g; \mathrm{Tr}^X_{I,O}(G)(f)$.

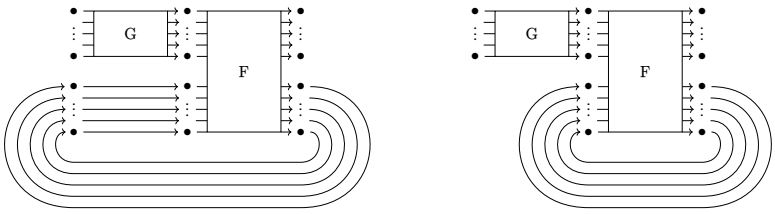


Figure 5.3: Naturality

  - Paths in the shape of $\mathrm{Tr}^X_{I,O}(G)(g \otimes 1_X; f)$ are of the form $e_g; e_{IO} + e_g; e_{IX}; (e_1 e_{XX})^*; e_1; e_{XO}$.
  - Paths in the shape of $g; \mathrm{Tr}^X_{I,O}(G)(f)$ are of the form $e_g; (e_{IO} + e_{IX}; (e_{XX})^*; e_{XO})$.

  Clearly the paths of the shape described by these regular expressions are in one to one correspondence.

- **Naturality in O:** this case is symmetrical to the previous one.
- **Dinaturality in X:** Given $g : X \to X$, we need to prove $\mathrm{Tr}^X_{I,O}(G)(f; 1_O \otimes g) = \mathrm{Tr}^X_{I,O}(G)(1_I \otimes g; f)$

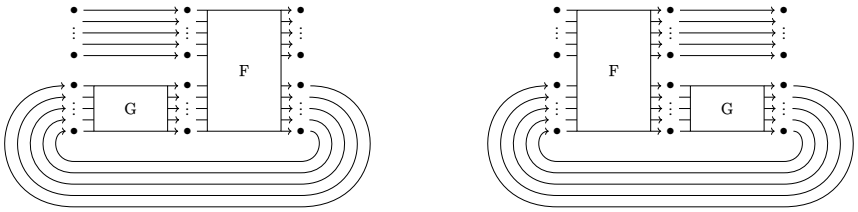


Figure 5.4: Dinaturality

  - Before any reduction, paths in the shape of $\mathrm{Tr}^X_{I,O}(G)(f; 1_O \otimes g)$ are of the form $e_{IO}; e_1 + e^f_{IX}; (e^g_{XC}; e^f_{CX})^*; e^g_{XC}; e^f_{CO}; e_1$
  - Before any reduction, paths in the shape of $\mathrm{Tr}^X_{I,O}(G)(1_1 \otimes g; f)$ are of the form $e_1; e_{IO} + e_1; e^f_{IX}; (e^g_{XC}; e^f_{CX})^*; e^g_{XC}; e^f_{CO}$

There is a clear bijection between these paths, which shows that the two resulting graphs after tracing are the same.

- **Yanking:** We need to prove $\mathrm{Tr}^X_{X,X}(\sigma) = 1_X$

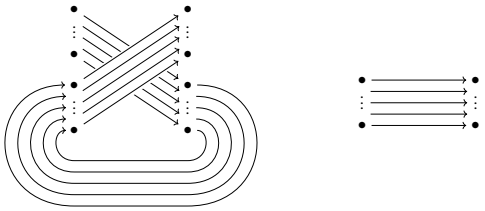
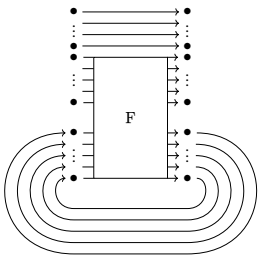


Figure 5.5: Yanking

The paths inside $\sigma^{\square X}$ are of length 2 and are precisely the $e_{ij}e_{ji}$ with $i \in I, j \in X$, and j being the correspondent of i in the bijection $I \simeq X$. This means for each i there is a unique j giving such a path, and thus the resulting graph has one arrow from $i \to i$ for every i, thus it is 1.

- **Vanishing:** There are two case to consider.
  - The vanishing for $\mathrm{Tr}^1_{I,O}(f) = f$ is obvious since $1 \otimes I = I$.
  - $\mathrm{Tr}^{X\otimes Y}_{A,B}(f) = \mathrm{Tr}^X_{I,O}(\mathrm{Tr}^Y_{X\otimes I, X\otimes O}(f))$ is the most interesting case.
    The paths in the shape of $\mathrm{Tr}^{X\otimes Y}_{I,O}(f)$ can be "compacted", rearranging the parenthesis to form maximal subpaths only using edges e_{YY}. These subpaths are edges in $\mathrm{Tr}^Y_{X\otimes I, X\otimes O}(f)$, which makes this rearranging of parenthesis a path in $\mathrm{Tr}^X_{I,O}(\mathrm{Tr}^Y_{X\otimes I, X\otimes O}(f))$.

- **Superposing:** We need to prove $\mathrm{Tr}^X_{C\otimes A, C\otimes B}(1_C \otimes f) = 1_C \otimes \mathrm{Tr}^X_{A,B}(f)$.



It is clear by doing a case disjunction, either a path falls in 1_C and it is trivial or it falls in f and it is as usual. □

Theorem (IO\Grph's structure)
The category $(\mathbf{IO\backslash Grph}, \sqcup, \sigma, \mathrm{Tr}^-_{-,-}(-))$ is a (symmetric) traced monoidal category.

We now want to define the actual model of interaction graphs. It is based on the **Int**-construction, we thus do a quick "pedagogical" reminder on how it works. Imagine one wants to compute the alternating paths between two graphs G and H. The idea is to unfold G and H and present them as bipartite graphs from $V_G \to V_G$ (resp, V_H), hence the definition of $_{IO}\mathbf{Grph}$. Because of the geometry of such a construction, paths cannot take an edge from the same graph twice in a row, and thus have to be alternated by construction. The trace is then used as a sort of "feedback", allowing to go back and continue the path to obtain paths of length greater than 2. Formally:

Definition 599 (Int-construction)
Given a category $\mathbf{C}$, we define the category $\mathbf{Int}(\mathbf{C})$ as follows:

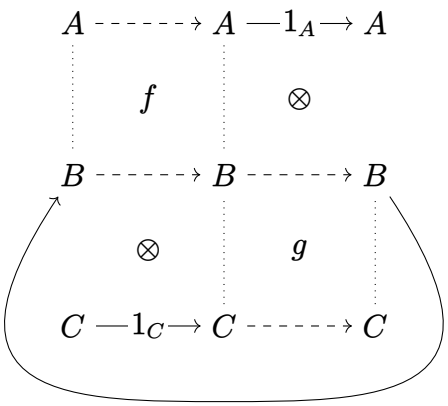
Objects: Pairs $(+A, -A)$ of objects of $\mathbf{C}$

Morphisms: $f : (+A, -A) \to (+B, -B)$ are morphisms in $\mathbf{C}$ of the type $+A \otimes -B \to -A \otimes +B$

Given $f : (+A, -A) \to (+B, -B)$ and $g : (+B, -B) \to (+C, -C)$, the composition is defined as first plugging f and g on $+B$ as follows:

$$+A \otimes -B \otimes C \xrightarrow{f \otimes 1_{-C}} -A \otimes +B \otimes -C \xrightarrow{1_{-A} \otimes g} -A \otimes -B \otimes +C$$

And then tracing over $-B$. Pictorially, this is what happens:

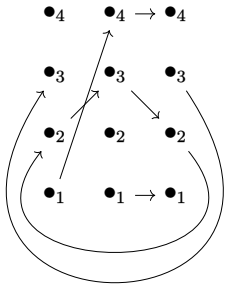


Paths go from left to right, except when looping. They are forced to alternate by construction.

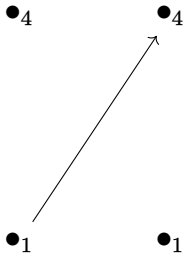
Theorem
Given a traced monoidal category $(C, \otimes, I, \dots)$, $\mathbf{Int}(C)$ is the free compact closed category containing C.
Moreover, the functorial embedding $\mathbf{C} \to \mathbf{Int}(\mathbf{C}) : A \to (A, I)$ is fully faithfull.
See [38] and [31].

Definition 601 (The category iG of interaction graphs)
We define **iG** as $\mathbf{Int}(\mathbf{_{IO}Grph})$.

Remark (Lack of Scalars)
Note how this category is lacking any interesting scalars to do a double glueing with a focus as defined in [36]: $\mathrm{Hom}_{\mathbf{iG}}(\emptyset, \emptyset) = \{1_\emptyset\}$. (This is a consequence of remark 589).
Note it is still possible to do a general double-glueing coming from a family of relations.
Notice also how the definition of trace uses $\mathrm{Paths}_{i \to o}(G)$, and thus all internal cycles disappear since they have no connection to i, o, as the cycle $2 \to 3 \to 2$ in the following example where we trace along $2, 3$:



(a) A composition of two interaction graph with trace, as in the **Int**-construction



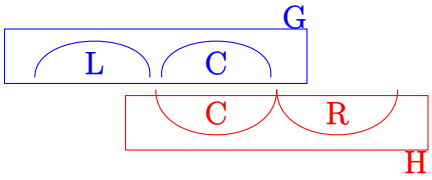
(b) The execution of said graph, internal cycles have disappeared

The point of this chapter is to understand better why that is and if it can be fixed.

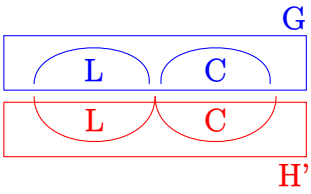
How does such a category capture the definition of interaction graphs defined in the previous section?

Observation
Take two graphs G with $V_G = L \sqcup C$ and H with $V_H = C \sqcup R$. Then we can see G as a morphism $\mathrm{obip}(G) : L \otimes C \to L \otimes C$ in $\mathbf{_{IO}Grph}$, hence as a morphism $iG : L \to C$.
Similarly H can be seen as a $iH : C \to R$ in iG.
From that we have that $i(G :: H) = iG; iH$

Remark (The problem of locativity)
Like every categorical model dealing with locativity, these models suffer from the fact that category theory is oriented and typed. Indeed, a graph $G : L \to C$, interacting on C with a graph $H : C \to R$:



Can also be represented as a graph $\hat{G} : \emptyset \to L \sqcup C$ if it interacts with a graph H' which is also on $L \sqcup C$:



Changing the place where a certain graph will interact is done categorically through the use of the adjunction coming from the closure: $\mathrm{Hom}_{\mathbf{iG}}(\mathrm{L}, \mathrm{C}) = \mathrm{Hom}_{\mathbf{iG}}(\mathrm{L} \sqcup \emptyset, \mathrm{C}) \simeq \mathrm{Hom}_{\mathbf{iG}}(\emptyset, \mathrm{L} \multimap \mathrm{C})$.
Since the category is compact closed (result of the **Int**-construction) then $\multimap = \sqcup$ and we get the desired result.
There is thus an extensive and unnatural use of the closure of the category to encode what would be a simple execution of three graphs $F :: G :: H$ in the original "set-theoretical"/location-based model.
There is a transfer of locations in and out of the center C to the extremities L/R to transcribe that.
As of today, there is no known way to fix this.
This is referred to in this thesis as *the problem of locativity*, although it could be argued that *category theory is the problem*, in the sense that it is not the right tool to study these objects.

As previously mentionned, the category **iG** has a sister category that has a richer set of scalars: the category of projects, which holds the count of cycles inside, by adjoining to every program an element in a monoid, called *the wager*, which intuitively counts the number of internal cycles.

Convention
In the following, we assume given a measure $[\![-]\!]_\Omega : (IG)_L \times (IG)_L \to \Omega$ with Ω a monoid. (Intuitvely, it counts the number of alternated cycles between two graphs).

Definition 606
The category **Project** is defined as follows:

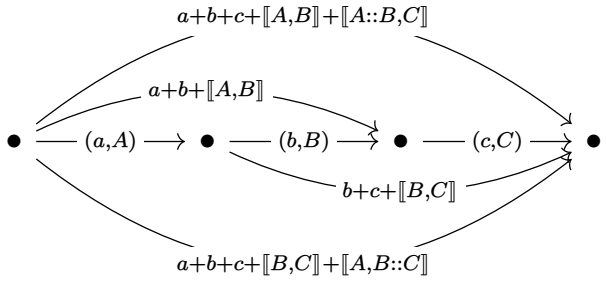
Objects: Any set

Arrows: Morphisms from X to Y are pairs (A, a) where $a \in \Omega$ and $A \in \mathrm{Hom}_{\mathbf{iG}}(\mathrm{X}, \mathrm{Y})$.

Composition: Composing two morphisms (A, a) and (B, b) gives $(A; B, a + b + [\![A, B]\!]_\Omega)$.

Notice also that $A \to (A, 0)$ of type **iG↪Project** is *not a functor*, precisely because of the fact that there can be non-zero measures of the interaction of two graphs. Only the projection $(A, b) \to A$ is a functor, which allows to see projects as a generalsiation of interaction graphs.

Remark (Trefoil Property)
Composition has to be associative for **Project** to form a category. Checking associativity means comparing the top and bottom arrow in the following diagram:



These two are equal precisely because of the *(Numeric) Trefoil Property*. (Notice also how it uses only 2 sides of the "threefold" equality that the trefoil property is, which is also due to the problem of locativity)

Hole
This trefoil property can be seen abstractly as a statement of the measure being a 2-cocycle (this is a cohomological notion), and the reconstruction of the category **Project** is a classical construction in group cohomology theory, see [1] for the definition of 2-cocycle and [59] for some reference on this fact.
It would be interesting to study further the link between cohomology and interaction graph, since cohomology is used to find invariant of spaces, and denotational semantics invariant of computation, there might be a way to connect the two fields. Although I am not very familiar with her work, part of the PHD of Samantha Jarvis, a student of John Terilla (coauthor of Seiller) is about such links, see [37].

It thus seems that the category **iG** has "lost" some data compared to **Project**, hence the lack of scalars, data that was recovered using the wager construction.
This phenomenon of losing data is something that does not happen in another category that has some similarities to $_{10}$**Grph**, the category of cobordisms, that we now turn our attention to.

5.2. Cobordisms

Cobordisms were introduced by René Thom in 1954 [63] in the search of nice geometric invariants of spaces.

We recall that a smooth manifold is a topological space which is locally homeomorphic to a Euclidean space (so, a topological manifold) and such that the gluing functions which relate these Euclidean local charts to each other are smooth.
One can then glue together such manifolds M and N on a common boundary B by taking the disjoint union $M \sqcup N$ here, and quotienting the result by identifying the two copies of B. This gives another smooth manifold, equipped with the quotient topology.
We first present the category of cobordisms as a form of cospan:

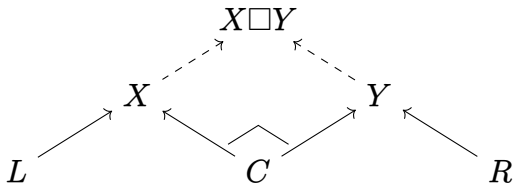
Definition 609
The category $\mathbf{Cob}[n]$ is defined as follows:

Objects: are *closed* smooth manifolds of dimension $n-1$.

Morphisms: The set $\mathrm{Hom}_{\mathbf{Cob}[n]}(\mathrm{L},\mathrm{C})$ of morphisms from L to C is the set of smooth manifolds with boundarys $\mathcal{M}$ of dimension n whose boundary $\partial\mathcal{M}$ is equal to $L \sqcup C$.

Composition: Composition is given by gluing cobordisms along their shared boundaries. Formally, given $\mathcal{M} \in \mathrm{Hom}_{\mathbf{Cob}[n]}(\mathrm{L},\mathrm{C})$ and $\mathcal{N} \in \mathrm{Hom}_{\mathbf{Cob}[n]}(\mathrm{C},\mathrm{R})$, the cobordism $\mathcal{M};\mathcal{N}$ is defined as the smooth manifold $(M \sqcup N)_{/\sim}$ where the quotient is defined w.r.t. the equivalence $c_{\mathcal{M}} \sim c_{\mathcal{N}}$ for all $c \in C$.

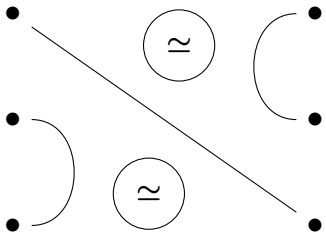
Categorically, the composition is a pushout:



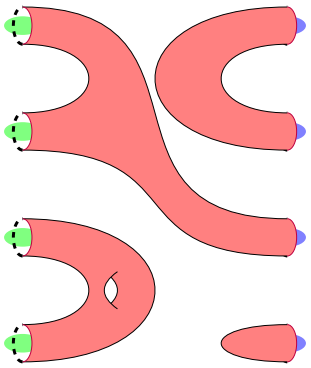
It is important to note that we only consider cobordisms here up to diffeomorphism, contrary to the original motivations where they are considered up to $(n+1)$ cobordisms (this would lead to ∞-categorical consideration, cob is hypothesized to be the free (∞, n)-compact closed category)

Example (1Cob)
The category $\mathbf{Cob}[1]$ is a simple example.
Its objects are closed smooth manifolds of dimension 0, that is finite sets of points, i.e. finite sets.

The set $\mathrm{Hom}_{\mathbf{Cob}[1]}(\mathrm{A}, \mathrm{B})$ is then a smooth manifold of dimension 1 whose boundary is the disjoint union $A \sqcup B$, that is a collection of disconnected segments with endpoints in $A \sqcup B$ and circles. Here is an example of 1-cobordism:



Example (A 2-cobordism)
Since the only connected manifold without boundary of dimension 1 is the circle, the objects in $\mathbf{Cob}[2]$ are disjoint unions of circles. Here is an example of a 2-cobordism:



Proposition (Folklore)
The category $\mathbf{Cob}[n]$ is dagger-compact.

Corollary
In particular, $\mathbf{Cob}[n]$ is compact closed (hence, also a traced monoidal category).

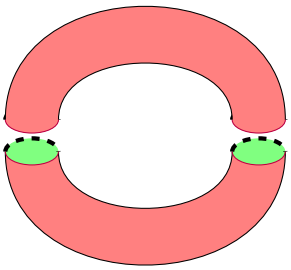
Hole
About compact-closed category, there are two ways to make a model of LL out of them: by doing a double-glueing to get a "usual" semantic model, or do the **Int**-construction since they are traced, and then a double-glueing to get a GoI model. Is there a link between the two?

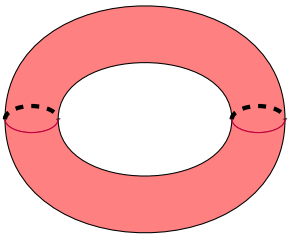
It seems that there is a bit of structure in common with **iG** (both categories are compact closed, also "geometrically" compose by a form of glueing). But as previously hinted, this category does not lose any scalar:

Observation (Scalars in Cob[1])
The category $\mathbf{Cob}[1]$ has a rich set of scalars: the manifolds without boundaries. For example, spheres and tori are morphism $\emptyset \to \emptyset$ in $\mathbf{Cob}[2]$.

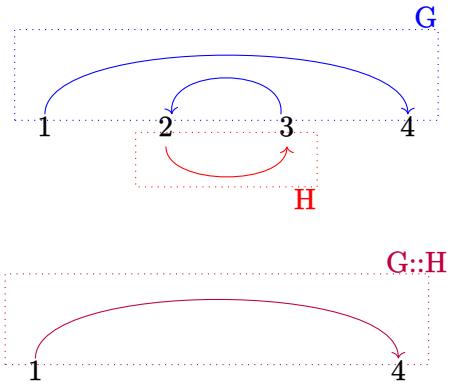
What do cobordisms have that interaction graph do not that they do not lose any scalars? The answer is hinted in the previous observation: manifolds *without boundaries*.

Example
Glueing 2 C shaped cobordisms give a donut:





Indeed: the problem with interaction graph is that *execution cares only about I/O*, and thus is lossy. Consider the following example which is an instance of remark 602:



The internal cycle between G and H is lost during execution because it *only cares about paths from the boundary of the graph to itself.*

Wagers are a way of reintroducing the information that was lost because the execution of interaction graph did not respect the geometry of the programs.

But there might be another way, which is the one that cobordism uses, that respects the geometry: *cospans, and glueing.* If the model is allowed to have programs without boundaries, then there is no need for wager or trefoil property anymore since no information will be lost.

Note
In dimension 1, which is the dimension of interaction graphs, the only object without boundary is the circle.
Thus, in scalars, the only interesting thing that can happen topoligcally is to have circles, and thus, counting circles (which is what the wager does) suffices.
If one finds an higher dimensional definition generalizing interaction graph, the monoid of wager would probably be more complex, and have elements such as 2 sphere + 3 tori?

If one looks again at how the category $\imath\mathbf{o}\mathbf{Grph}$ is defined, the computation of the composition $G;H$ is actually done in 3 steps:

- First glue G and H together (this is a pushout, which is the way cospans are composed). This is very visible in the proof that the category is traced, where we consider the geometrical object and look at paths on it in most reasonings. This step, we call *the Glueing.*
- Then we compute the paths (of length 2 or more). This step we call *the Computing.* Together with the previous step, they are known in game semantics under the name *Composition.*
- Finally, we remove the internal workings, leaving only the paths. This step is called *the Hiding.*

Observation
Note two things:

- The fact that cobordism are continuous makes it so that the Hiding is done automatically, since boundaries "merge" with the geometrical object once they are glued.
- The fact that they have elements without boundaries makes it so that fully internal cycles are not forgotten.

Note
Something weird is at play in **iG**: the computational part, that is, the computing of paths, is duplicated during the definition: It happens both in the composition of

IO**Grph** and in its trace, while one could expect to have a separation between the glueing and the computing.
Another thing is that the real "dynamics" are actually the computation of paths, while the glueing is more of a locative nature, but they are somehow mixed together.

If we want to make a model of computation with dynamics, we thus need a way to study how paths compose. It seems cobordisms and interaction graphs are glued in a similar way, but there is no computation of paths in cobordisms. We try to introduce paths in cobordisms in an ad-hoc way and see if we can take anything from it.

The Π_1 functor, which associates to a topological space X its fundamental groupoid, that is, the groupoid of all paths inside the space X, preserves certain pushouts of groupoids:

Theorem (Van Kampen's Theorem for Groupoid, Ronald Brown)
A set A is called representative in X if A meets each path-component of X. When $C = L \cap R$, and $\mathring{L} \cup \mathring{R} = X$, then:

$$\begin{array}{ccc} \Pi_1(C, A) & \longrightarrow & \Pi_1(R, A) \\ \downarrow & & \downarrow \\ \Pi_1(L, A) & \longrightarrow & \Pi_1(X, A) \end{array}$$

is a pushout of groupoids.
In our setting, we will always take $A = X$ which is representative.
The proof of this version of the theorem can be found in[10].

We will now prove that because of this, one can "compute paths" in a functorial way.

For that we will use a central theorem to the theory of cobordisms:

Theorem (Collar Neighbourhood Theorem, Morton Brown)
Let X be a smooth manifold with boundary ∂X. Then the inclusion $\partial X \xrightarrow{i} X$ has an open neighbourhood $U \simeq_{\text{diff}} \partial X \times [0, 1)$ (an open collar).
This fundamental theorem in the theory of cobordism was proven by Morton Brown in [9]

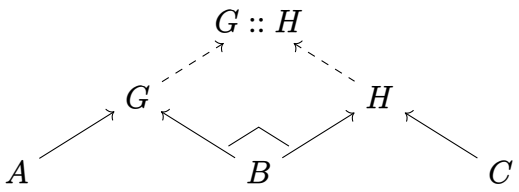
Definition 621 (IOGpd)
We define the semi-category IO**Gpd**:

Objects: Sets

Morphisms: Morphisms in $\text{Hom}_{\mathbf{IOGpd}}(\text{A}, \text{B})$ are groupoids over $A \sqcup B$ (up to isomorphism of groupoids, *not just equivalence*).

Composition: The composition of $G \in \mathrm{Hom}_{\mathbf{IOGpd}}(\mathrm{A}, \mathrm{B})$ and $H \in \mathrm{Hom}_{\mathbf{IOGpd}}(\mathrm{B}, \mathrm{C})$ is the (free) groupoid of paths from $A \sqcup C$ to $A \sqcup C$ in the glueing. We write it $G :: H$.

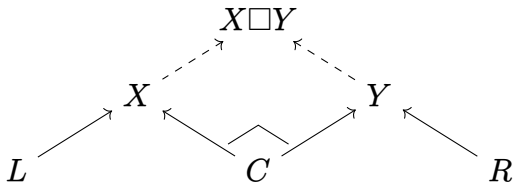
This can be presented as a pushout of cospans:



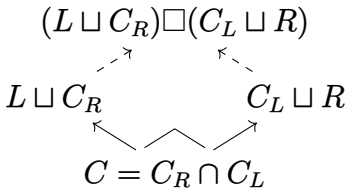
Note there is no identity, this is a *semi-category*.

Theorem
Π_1 is a semi-functor from $\mathbf{Cob}[n] \to {}_{\mathbf{IO}}\mathbf{Gpd}$.

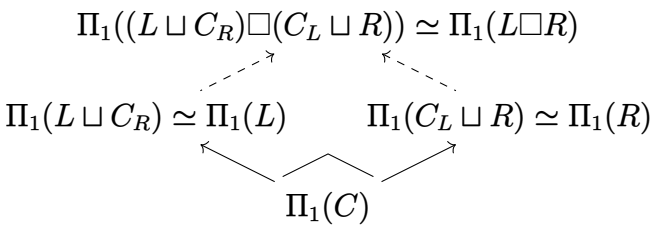
Proof
Take two cospans in $\mathbf{Cob}[n]$ that can be composed together:



We then use the collar neighbourhood theorem on $(L \sqcup C \to X)$. This gives an open collar $O \simeq L \sqcup C \times [0,1)$, from which we we extract an open set $C_L \simeq C \times [0,1)$. And similarly for $(R \sqcup C \to Y)$ to get a C_R.
We can form the following cospan:



Where we can apply Van Kampen's theorem:



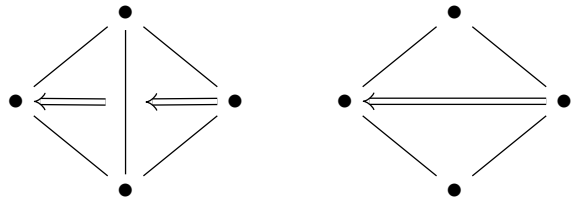
All $\simeq$ are isos of groupoids being because glueing collars is just an equality when considering things up to diffeomorphism. (This is all thanks to the collar theorem, which allows to properly glue cobordisms together)
This diagram shows $\Pi_1(L\square R)$ is a pushout and thus we get our desired result: $\Pi_1(L\square R) \simeq \Pi_1(L) :: \Pi_1(R)$. □

Observation
It seems that the Van-Kampen theorem is a proof that Π_1 is somehow analogous to the execution formula.
This seems especially true when looking at 1-dimensional cobordism which are similar to a weak form of undirected interaction graph.

This suggests using a category similar to $_{10}\mathbf{Gpd}$ as a category to keep both the geometry and dynamics.
There is one thing of note here, the fact that cobordisms are not directed spaces, and thus the paths on cobordisms are invertible (hence the groupoid).
But this is not the case for graphs, which are more like directed spaces. The notion corresponding to fundamental groupoid in the directed case is the one of *fundamental category*. This suggests studying a category whose morphisms are not just graphs but semi-categories.

Hole
We saw the category $\mathbf{Cob}[n]$ is a compact closed category with a lot of scalars, which mean we could build a model of LL from it by double glueing (for the !, just use the usual one $- \times \mathbb{N}$) Would that be an interesting model ?

Hole
Since it seems models are about glueing along boundaries (here vertices): the model of permutation that was defined in section 2.4 is about glueing undirected graphs together, and undirected graphs are the 1-dimensional case of simplicial complexes. Thus there might be ways to generalize interaction graphs with things akin to simplicial sets.
For example two triangle could be glued along an edge to form a square with a diagonal, and the execution could remove said diagonal to form a square as in this example:



It might be possible to make a more general version of the model where we glue higher dimensional complexes on lesser dimensional boundaries, as in cobordism As of today, the existence of a "right way" to make the dynamics of it work is an open problem.

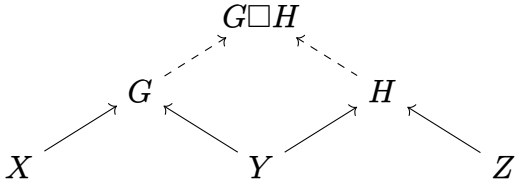
5.3. Perspectives on dealing with Trefoil

In this section we will do multiple attempts at defining a new category similar to the one of interaction graph, but with a twist inspired from cobordisms: we will try to do things "geometrically", to avoid forgetting about internal loops. This means we will also need a way to make the internal machinery of a graph invisible (these loops cannot be seen from the outside) but not forget them (there will still be a way of Hiding things to check that we can fall back the usual categories).

In the same way that when glueing two cobordisms along a boundary, said boundary disappears from the outside. We can use cospans for such a purpose:

Intuition (IG as cospans)
From the cospan point of view, the usual composition of interaction graphs is done in three steps:

- The glueing:



- The computing (dynamics): computing all possible paths, which corresponds to taking a free construction over the graph.
- The "hiding": erasing the internal machinery (but this step is only necessary if one wants a *denotational* semantics, here we are trying to subsume this by doing a semantics with dynamics).

Note that here, the hiding is actually decomposed in two steps: part of the hiding is done by the pushout, this makes the inner workings "invisible" from the outside. The other part, the "inner" hiding, is the one that usually equalises many programs, and makes them the same.

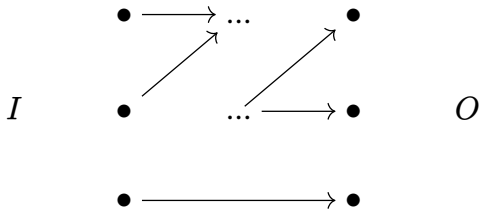
Note
In cobordisms, there is an extra phenomenon: the locations "merge" with the wiring. There is no notion of vertex or edges, a boundary is part of the "edge", and thus disappear when the edges are glued.

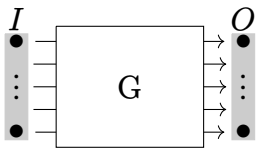
5.4. Preliminal attempt: what goes wrong

We do a first attempt at defining a category removing all mention of execution, and where the composition of graphs would be purely glueing (without removing the vertices where the glueing happens). The dynamics would then be added by a sort of functor computing paths. Unfortunately, this will fail.

Definition 628 (Input/Output property)
A graph G is said to have the input/output property (relatively to two sets $I \cap O = \emptyset$) when for every $e \in E_G$, $t(e) \notin I$ (I is an input) and $s(e) \notin O$ (O is an output).

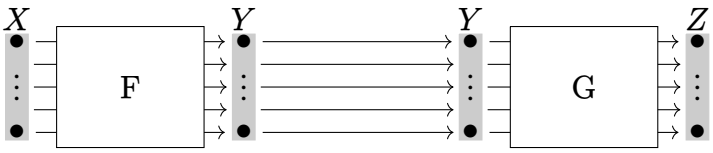
Example
These are graphical representation of what a graph with the Input/Output property could look like:





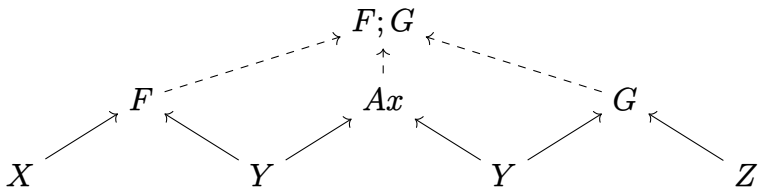
With input I represented on the left, and output O on the right.

Definition 630 ((MISTAKEN) A category $_{IO}$Quiv)
We would like to define a category $_{IO}$**Quiv** where objects are finite sets, and morphisms $G : I \to O$ are graphs on $I \sqcup O$ with the input/output property.
We would then compose two graphs $f : X \to Y, g : Y \to Z$ in the following way:



We could also do an actual glueing but this makes the pictures clearer.

Remark
This categorically can be obtained as the following colimit (more precisely, a composition of two pushout), where Id is the bipartite graph of identity in ₁₀**Grph**:



Where Ax is just the identity graph Id_Y in ₁₀**Grph**. (This choice of composition adding an extra Ax is just because it makes things clearer visually. It is the choice one has to make if the trace is taken to be feed ($\circlearrowleft$) and not fuse ($\square$)).

Unfortunately, this definition *does not quite work*:

Beware
This definition of ₁₀**Quiv** give rise to a semi-category: there is no identity.

Proof
Composing with a morphism strictly increases the number of vertices. Since it is not possible to leave a graph unchanged by composition, it is not possible to have an identity. $\square$

We can investigate this a little more, notice this:

Proposition (Normal Form semi-functor)
There is a semi-functor $\Pi_1 : {}_{10}\mathbf{Quiv} \to {}_{10}\mathbf{Grph}$, that associates to an IO graph $G : I \to O$ the bipartite graph of its paths from I to O.
Formally it is defined as follows:

- $\Pi_1(X) := X$.
- $\Pi_1(f : X \to Y) := (X \sqcup Y, \mathrm{Paths}_{X \to Y}(f), \hat{s}, \hat{t})$ for f a graph (V, E, s, t) and $\hat{\cdot}$ the lifting from edges to paths.

Observation
Note that $\Pi_1(1_X) = 1_X$, so up to computation, the morphism 1_X of $\imath\mathbf{o}\mathbf{Grph}$ seen as a morphism in $\imath\mathbf{o}\mathbf{Quiv}$ is indeed an identity.

Inspired by the example of cobordisms, we will discuss different approaches that one could take to solve such a problem:

- The discrete way: one could just accept to deal with semi-categories (or other kinds, as you will see). This would imply writing up a lot of litterature to adapt the existing theory to semi-categories, stating equivalent theorems (**Int**-construction etc...) in these settings etc...
- The continuous way: in cobordisms, this problem does not occur. Indeed, since the objects are considered up to diffeomorphism (*i.e.* purely geometrically) the composition with a cylinder is indeed an identity: when glueing together two wires, there is no vertices left over in the middle in the continuous case, it "merges" with the wire.
  One could thus consider a geometrical realization of our graphs. The question being, which kind. This is also slightly unsatisfying in that it requires continuous math, which requires slightly more advanced mathematics than the discrete case, which is a setting more familiar to computer scientists.
- The "wiring first" approach: interaction graph are a "locations first", in the sense that we put down vertices, and glue interaction graphs on said vertices. We could instead change the focus of the model on edges: we could have graphs, with special ingoing/outgoing edges that have no source/target (maybe neither). Two such graphs would be glued by glueing two such "half edges", one with (at least) no target with one with (at least) no source, making a complete edge.
  An identity would be possible here: a graph with edges with neither sources nor targets, once glued would not change anything.
  But such considerations would lead to a wager-like property: what would happen if one attempts to glue by tracing along two half arrows not linked to vertices? This would be an object without boundary, and there would then be a need to keep track of these, introducing a form of wager.
  This is too close to the current theory so we will avoid this option, as it has already been somehow explored (note it is still a fresh perspective on the wager).
- The higher way: one could study a 2-categorical version of this category, where 2 morphisms would correspond to hiding some internal locations. There might be links with the 2-dimensional models of lambda calculus (where 2-morphisms represent beta equivalence).
  Notice we will not use a setting of "weak" 2-categories, because we want the 2 cells to represent normalisation, hence to have $f; id \Rightarrow f$ but not as an invertible cell.

We feel that all these possibilities are actually interesting, not only in their own right, but also in the comparison that one could get from them.

This thesis focuses mainly on the last option, the "higher way", but we discuss quickly the others before to see their strengths and weaknesses.

Beware
One thing that is important to note and that I would like to stress is that the topological property we want to not forget is technically the presence of *circuits*, not of *cycles*. Circuits are the quotient of cycles under cyclic permutation.
Depending on the order of execution, cycles will change: they can get longer or shorter etc... and so the cycles are not really preserved throughout computation. But the underlying topology, the presence of circuit, is.

We now turn our attention to three of the four possibility evoqued above:

5.5. The discrete way, a semi-categorical approach

We quicly recall the notion of semi-category:

Definition 636 (Semi-Category / Kategory)
A semi-category, that we will shorten to *kategory* is almost a category but without the additional data of identities.
Formally, it is given by:

- A set of objects O
- Sets of morphisms $\mathrm{Hom}_{\mathbf{C}}(\mathrm{a},\mathrm{b})$ for all $a, b \in O$.
- A composition function $";" : \mathrm{Hom}_{\mathbf{C}}(\mathrm{a},\mathrm{b}) \times \mathrm{Hom}_{\mathbf{C}}(\mathrm{b},\mathrm{c}) \to \mathrm{Hom}_{\mathbf{C}}(\mathrm{a},\mathrm{c})$, which is associative.

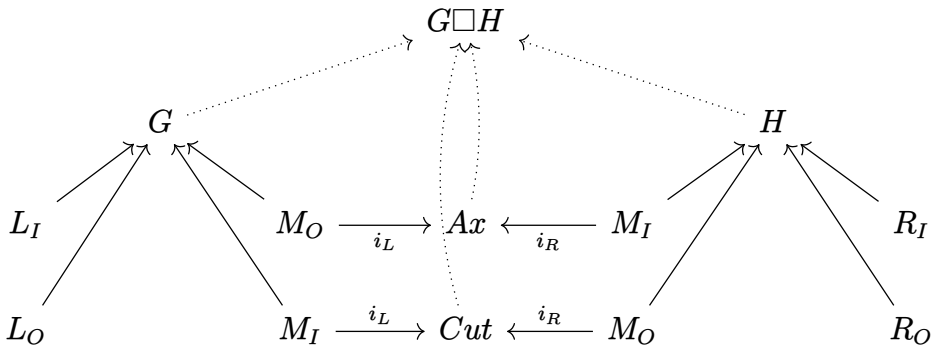
The usual notions of functors, natural transformations etc... can be adapted to this setting, see [33] for an exemple of use in computer science.
We write **Kat** the category of kategories.

The first option considered is trying to stay discrete and 1-categorical. We will see that this will would require introducing a new sort of "category":

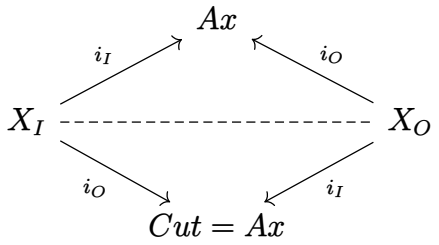
Remark (G_{IO} is (not really) Traced)
The category ought to be the equivalent for kategories of spatial traced categories as defined in [60] since it has all good properties one can expect except for yanking, or a Feedback category.
Indeed, the feed operator $(-)^{\circlearrowleft X}$ of definition 595 is really similar to a trace

geometrically, where we see the result of doing feed on $f : I \otimes X \to O \otimes X$ as being of type $I \to O$ (hiding the X in the process).
The difference with usual traced categories is that the axiom of yanking should not be satisfied, (in fact, it cannot even be stated since there are no identities), and hence usual constructions will not preserve the symmetry.
But if one looks at the drawings, it *morally* has all other good properties that are not always given in the more general case (left and right trace coincide etc...)
The problem being that *we cannot even state* these other properties, for, once again, there is no identity.

Remark
This "trace" is also obtained as a pushout which is symmetric to the one for compositon. After an **Int**-like construction, would give rise to a composition of this shape:



Where Ax is the identity of $\imath\mathbf{o}\mathbf{Grph}$ and Cut is just syntacic sugar for Ax with flipped injections:



(Note that this beautiful pushout is the reason we chose feed "$\circlearrowleft$" as a trace and not fuse "$\square$").

How could we solve the problem of not being able to define a trace?

The usual definition of trace uses identity to allow for types to match in the composition of morphisms but it is *not a necessity*, rather it is a form of encoding: the tensor product $\otimes$ is used to internalise multiple inputs/outputs in a single one, since categories compose only morphisms with arity 1/1. Identities then serve as paddings to accomodate for the discrepancy in the types. But one could simply accept to deal with multiple inputs/outputs, which is very natural and what is usually done diagramatically, since the identities "are not really there".

Having multiple inputs/outputs is usually done throught the setting of polycategories (but one can only compose along one), or more generally colored PROPs:

Definition 639 (Colored PROP)
Given a set of color C, a colored prop is a category whose objects are precisely the elements of the free monoid on C (we write $\otimes$ for the monoid operation)

From which, the notion that we need would seem to be easily be derived: the notion of colored semi-PROP.

Definition 640 ((WRONG) csPROP)
Given a set of color C, a colored semi-PROP is a kategory whose objects are precisely the elements of the free monoid on C (we write $\otimes$ for the monoid operation)

But this definition is mistaken, and the saying that PROPs corresponds to categories with multiple inputs and outputs is misleading: it is the case precisely because identities are used as a way of making type match to plug things, but this cannot be generalized to kategories.

One would need a new combinatorial definition, similar to the one of polycategories, but as general as the one for PROPs, allowing to compose along multiple input/output at the same time.

Conclusion
Keeping kategories is a lot of easy bureaucratic work: one would need to generalize for this setting every notion of category theory (the **Int**-construction, traces, compact closure etc...) and reprove every theorem. It would not bring a lot of immediatly obvious value: this feels like this could be considered a "low hanging fruit".
To avoid using kategories, one could just add "fake identities" everywhere, but this would lead to annoying considerations because every definition using morphisms would need to make a case distinction between "fake" and "real" morphisms (although this could be put under the rug because the fake morphisms should be indistinguishable from the "real" (non-)identity from the outside).
All considered, we will not pursue it.

Hole
There might be a way to use polycategories to describe n-ary cospans, which in

turn might be a way to describe interaction graph with glueing, but this would suffer from the problem of locativity even more.

5.6. Geometrical Realisation: IG as Spaces

We will now try to keep the lost information that was saved using the wager through geometry (here, homotopy theory).
As seen in cobordism, there are two reason internal cycles disappear in interaction graphs: that there is a computation of paths that erases them, and that since the model is discrete, there is no way of having a "circle" without vertices.

A natural way of solving this problem would be to consider not graphs, but their geometric realization, since the notion of circle is well defined on its own there, as a connected component, while in a discrete setting we need a vertex to loop on.

5.6.1. IG as directed topological space

This approach seem to me to be a very promising one, although I am stuck at quite the beginning. We discuss it nonetheless.

The idea is to make our graphs truly geometric by looking at their realisation in a topological space. Then they would somehow correspond to the actual drawing that we are doing of them. There is of course a need to identify some graphs that differ only by non-important deformations (like making an edge longer or shorter), but if this is done correctly then we would truly save the topological data.

Because our graphs are oriented graphs, and there is a notion of direction of our edges, it seems natural to not consider just "usual" spaces, but to consider the notion of *directed space*. We quickly recall the notion (and related notions). The book of reference in the field is [20] and a good PHD on homology for this setting is [17].

Notation
We denote by I the unit interval of $\mathbb{R}$, $s(I) = 0 \in I$ and $t(I) = 1 \in I$.

Definition 644 (Directed Framework)
A directed framework (df) on a topological space X is a set $dX \subseteq Top[I, X]$ of distinguished path that are called the *directed path of* X, and that satisfy the three following axioms, for all $f, g : I \to X$:

- (Constant Path): If f constant then $f \in dX$.
- (Stability by Reparametrization) For all $r : I \to I$ increasing, $r; f \in dX$.

- (Stability by concatenation) If $f(1) = g(0)$ (the boundaries match) then $f; g \in dX$.

Note I is equipped with the directed framework dI of increasing functions.

Definition 645 (Directed Space)
A d-space is a pair (X, dX) with X a topological space and dX a directed framework on that space. We denote by $dX(a, b) \subseteq dX$ the set of dipaths from a to b.

Definition 646 (Directed Map)
A dmap $f : (X, dX) \to (Y, dY)$ is a continuous function from X to Y such that $dX; f \subseteq dY$.

Definition 647 (Category of dSpaces)
We call **dTop** the category whose objects are directed spaces and morphisms dmaps. We will write $X \simeq_d Y$ to say spaces are isomorphic as directed spaces (so through directed maps, not just homeomorphism).

Convention
Since we will consider only directed spaces, we will often forget the dX part in this part.

The following property is very important, as it is the mathematical formulation of "one can glue two dspaces together":

Proposition (Cocompletion of dTop)
The category **dTop** is cocomplete. In particular, it has pushouts.

We are now ready to define the *geometric realization* of a graph, that is, make a dspace out of it.

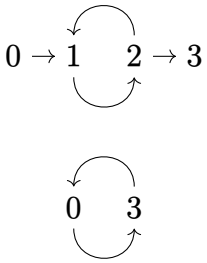
Proposition (Geometric Realization)
To a graph $G = (V, E, s, t)$ we can associate the dspace $\mathcal{R}(G) := (V \sqcup_{e \in E} \bigsqcup I_e)/\sim$ with V seen as a discrete space, $\simeq$ the equivalence relation such that $s(I_e) \simeq s(e)$ and $t(I_e) \simeq t(e)$, the topology obtained via disjoint union and quotient.
The directed framework is obtained via amalmagation of the points that are identified: given X a dspace, and $x, y \in X$, the amalmagation $X[x = y]$ is the quotient space of X where x and y are identified, and the directed path are (up to reparametrization) the finite sequences of $f_i : a_i \to b_i$ with $a_i, b_i \in \{x, y\}$ except (possibly) for a_0, b_n.
Its effect on morphisms is the obvious one.

Proposition (Realisation is a functor)
The function $\mathcal{R}$ is a functor from the category of directed multi-graphs to the category of directed space and dmaps.

Proposition (Colimits)
The functor $\mathcal{R}$ preserves colimits (it is left adjoint to a functor called the Nerve)

We now have a way of making an actual space of our graphs. But in the same way we need to identify graphs up to renaming of edges for example, we need to identify spaces up to certain deformation, the "drawing" of the space should not matter. This correspond to homotopy equivalence of spaces. In the directed case, there is a related notion of dihomotopy equivalence. But something weird happens:

Beware (Dihomotopy equivalence of space is not the right notion)
These two figures are not dihomotopy equivalent (note they are homotopy equivalent):



Indeed, there is no path from 3 to 0 in the top one, while there is one in the bottom one. (Notice how although, it does not really change the paths from 0 to 3).
This is problematic because the top picture is obtained by composing the bottom one with what should be identities (the problem we were trying to solve).
So it is unclear what notion of equivalence of space we should consider.

Hole
What notion of equivalence would be the right notion to use in this setting? Maybe regular homotopy equivalence, but there is a need to check that this does not break the directed structure when considering directed paths.
I think the best bet is that it is isomorphism of dspaces, just like (as discussed in the next sections) when considering semi-categories, the right notion is isomorphism and not equivalence of semi-categories.

Now, we turn our attentions away from the space to look at paths.
Similarly to the case of cobordisms, we have the following functor:

Definition 655 (Fundamental Category)
There is a functor $d\Pi_1 : \mathbf{dTop} \to \mathbf{Cat}$, associating to a dspace (X, dX) the category of its paths, which is defined as follows:

Objects: Objects are points in the space X.

Morphisms: Morphisms $x \to y$ are equivalence classes of path $[\gamma] : x \to y$ up to dihomotopy (see below).

Composition: Compositions is given by composition of paths (doing one after the other).

Identity: The trivial "do not move" path.

The action on dmaps is clear: $d\Pi_1(f)$ is a functor, with $d\Pi_1(f)(x) = f(x)$, $d\Pi_1(f)(\gamma : x \to y) = \gamma; f$.

The problem with such an approach is that there would be "too many paths" (in the same way there was too many spaces) for example every reparametrization (intuitvely, taking the same path faster/slower) would be considered its own distinct path. There is a need to equate some of these paths. The right notion for the job is usually the one of (directed, since we want to respect the orientation of edges) homotopy. We quickly discuss the notion.

In topology, it is possible to create a "function space" $A \to B$ by equipping the set of continuous maps $A \to B$ with the compact-open topology.

Definition 656 (Compact Open Topology)
Take K a compact set in A, X open in B, $V(K, X)$ is the set of functions f such that $K; f \subseteq X$. The *compact open topology* is the topology on the space of functions, generated by taking the $V(K, X)$ as subbasis.

Proposition (dX as a Space)
The set dX can be seen as a topological space, as a subspace of the space $\mathbf{Top}(I, X)$ equipped with the compact-open topology.

Definition 658 (diHomotopy of path)
Given two paths $\gamma, \delta : x \to y$ in dX, a dihomotopy is a path $H : \gamma \to \delta$ in dX, that is, a continuous function $H : I \to dX$ such that $H(0) = \gamma, H(1) = \delta$.

Notice how this relates to the usual notion of homotopy: H is an homotopy in the usual sense, but such that all the paths in between γ and δ are directed.

Note
Note how instead of equating such paths, we could make a 2-category by adding 2-cells representing the dihomotopies.

Assuming we find the right notion of equivalence, we will have solved our problem.

Hole (The category $_{1\mathrm{o}}\mathbf{dTop}$)
It should be possible to define a category whose objects are boundaries of "dspaces with boundaries" (for a nice enough definition of dspace with boundary), morphisms would be disjoint cospans with the right notion of equivalence of space and composition would be obtained by pushout.
This category would then be a traced monoidal category, with monoidal structure similar to $_{1\mathrm{o}}\mathbf{Grph}$, and geometric realization of $(-)^{\circlearrowleft}$ as its trace.
To every proof, one would then be able to associate a space, with the invariant throughout computation that they have the same $d\Pi_1$ when restricted to I/O.

Note
Note how the $\circlearrowleft$ operator is really reminiscent of the geometric realization of a while loop in [20].

Note
An interesting thing to note is that the number of circuit, that is the wager, could be computed by homology.

Beware
In the previous note was mentionned the concept of homology.
I assumed that homotopy was the right notion of equivalence of paths because what we are trying to achieve is to get a definition that is continuous and "geometric" to solve our motivational problem.
It might be a red herring though, because what we care about in the end are circuits. Say one has a single vertex with two loops e_1 and e_2. The two circuits $e_1; e_2$ and $e_2; e_1$ are equal (since one is a cyclic permutation of the other), which is the case in homology but not in homotopy.

Conclusion
This feels like it is a promising approach in the sense that it would solve a lot of problem. But there are a lot of things that are unclear:

- What is the right notion of equivalence of space? This is crucial but it is unclear what it should be.
- How to do a quantitative model in such a setting? It is not possible to associate "to every edge" a weight, since there are not edges anymore. Encoding the weight in the geometry such as considering the length of paths would break everything.

Nonetheless, it would be great to be able to use tools coming from algebraic topology to get information on programs once interpreted inside interaction graph (lambda terms for example).

5.6.2. Comparing permutation and cobordisms

Remember that interaction graphs are a generalisation of the model of permutations . A simple way of "bypassing" the problem of finding the right notion of equivalence of space, as mentionned in beware 653, is to get back to the undirected case.

This first definition is extremely similar to $IO - GRAPH$, with the sole difference that the graphs are undirected.

Definition 665 (IOPerm)
Objects: Any set.

Morphisms: $G : A \to B$ is a partial bipartite graph.

The composition of $G \in \mathrm{Hom}_{\mathbf{IOGrph}}(\mathrm{A}, \mathrm{B})$ and $H \in \mathrm{Hom}_{\mathbf{IOGrph}}(\mathrm{B}, \mathrm{C})$ is given by glueing the graph G and H along B, and then computing the paths of length 2 in this graph (and forgetting about everything else).
The result is the partial bipartite graph whose edges are the paths of length 2 with one edge in G followed by one edge in H.

We could thus identify these morphisms up to homotopy equivalent (undirected) geometric realization. This would give a "weaker" model but it would still be interesting nonetheless.

Something to note is how this category relates to cobordism. Even without considering boundaryless elements, cobordisms have cups on the side, such as in the one dimensional case of example 610.

The category of partial cobordisms can be thus seen as a very loose generalization of the category above (it is the free compact closed category after all). Yet it is unclear to us what computational sense one could make of these cups (in a sense, they allow to bypass the cut/ax alternation, allowing the environment to use the program to answer to itself).

5.7. The two-categorical way

After this quite extensive review of other possibilities, we now fall on the point of view that we will study in more details in this manuscript: "the higher categorical" point of view.

There are multiple ideas at play, but the first one is to make the action of Π_1, or at least the hiding part, into two-cells, so that the "fake" identities of the semi-category become identities again.
This is a way of encode a form of "execution" as 2-morphism, which is very natural.

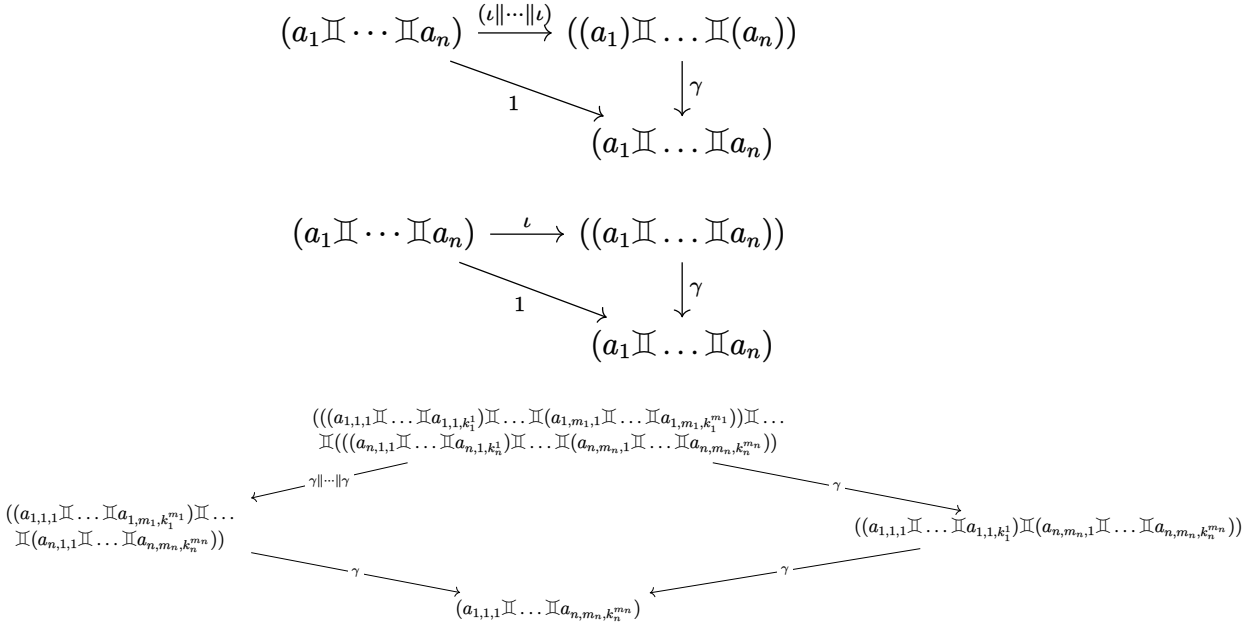
Usually, the dynamics are treated categorically with bicategories, but this is "unsatisfactory" for us: in bicategories, 2-arrows are invertible, and so the direction of computation is lost, which would make it impossible to generalise to a setting with a form of small step semantics for computation where we actually compute the paths step by step.

The natural setting for such a thing seem to be a form of 2-category with non invertible 2-arrows. The only notion we could find in the literature is taken from [43], and is an unbiased definition. This would be unfortunately quite hard to work with, as you can see:

Definition 666 (Lax Bicategory)
A lax bicategory $(B, \barwedge, \gamma; \iota)$ (the symbol $\barwedge$ can be read as "glue" in english) is given by

- A set B_0 of objects, 0-cells (locations).
- For $x, y \in B_0$, a category $B(x, y)$, whose
  - Objects are called 1-cells (programs).
  - Arrows are called 2-cells (execution traces).

And for every $n \in \mathbb{N}, a_0, \ldots, a_n$:, a functor $\barwedge_n : B(a_0, a_1) \times \cdots \times B(a_{n-1}, a_n) \to B(a_0, a_n)$, the n-fold composition.
We write $f_1 \barwedge \ldots \barwedge f_n$ when composing 1-morphisms and $\alpha_1 \parallel \cdots \parallel \alpha_n$ when composing 2-morphisms.
Note how the identity is given by the 0-ary composition: $\top \to B(a_0, a_0)$.
We also require an associator 2-cell: $\gamma : ((f_1^1 \barwedge \ldots \barwedge f_1^{k_1}) \barwedge \ldots \barwedge (f_n^1 \barwedge \ldots \barwedge f_n^{k_n})) \to (f_1^1 \barwedge \ldots \barwedge f_n^{k_n})$ for all compatible double sequences.
And unitor 2-cell: $\iota : f \to (f)$. The left hand side f can be seen as the "static" object, (the constatation in Girard's terminology), while (f) would correspond to putting this object into the dynamic world (the performance).
There are also commutative diagrams (We will not put the indexes on γ and ι to make them more readable):



Note how the point of locativity is to avoid such complex bureaucratic considerations: with locativity, ι and almost all the associators γ are identities (except the ones involving 0-ary composition).

Because of this, we are actually interested in a really degenerated case of this abstract structure.

Beware
We think that, based on the literature, the notion above is the right notion (in particular, if considering paths up to dihomotopy is the right call in the previous section, there is a need for non identities γs in such a context, as they would correspond to reparametrization etc...)
We now give an ad-hoc definition that will suffice, and that *we hope* is a biased equivalent definition to the one above when ι is an equality, and the only non trivial γs are the one involving 0-ary $\amalg$ (note how even then there are a lot of unbiased coherences).
It is unclear whether it is truly the case and whether this notions really fits well in the literature.

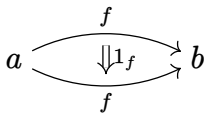
Convention
In the next definition and after, we will use $\amalg$ for the composition of 1-morphism. As for 2-morphisms, we will use ; for the vertical composition, and || for the horizontal compositon.

Definition 669 (Discrete Glue Category)
A Discrete Glue Category is given by:

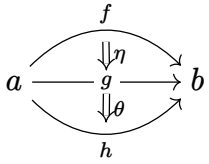
- A set of objects C.
- A set of 1-morphisms between two objects $C(A, B)$.
- A set of 2 morphisms between morphisms $C(f, g)$.

We also require the following data:

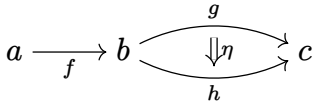
- A weak identity : $1_A : A \to A$ for every A.
- An identity when given a morphism f as in:



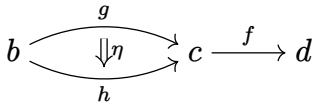
- A notion of composition of 1-morphisms: given $f : A \to B, g : B \to C$, we have $f \amalg g : A \to C$.
- A notion of composition of 2-morphisms, written $\eta; \theta : f \Rightarrow h$ in the following situation:



- A left whiskering, written $f \triangleleft \eta : f; g \Rightarrow f; h$, in the following situation:

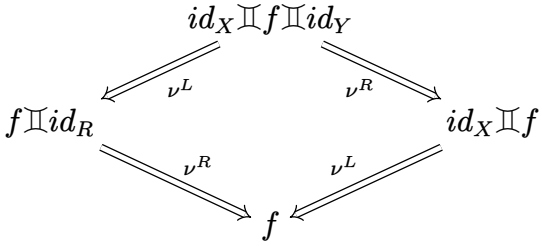


- A right whiskering, written $\eta \triangleright f : g; f \Rightarrow h; f$, in the following situation:



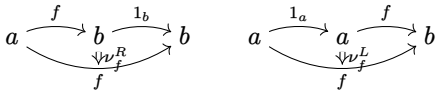
- Special 2 cells representing how there is a need to do an hiding step to get an identity: $\nu^L_f : id_X \amalg f \Rightarrow f$ and $\nu^R_f : f \amalg id_Y \Rightarrow f$.

These special 2 cells have simple 2-coherences:

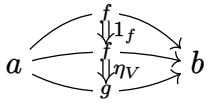


These data also need to respect all axioms for a 2-category except the ones concerning identity:

1. There are two cells ν^L, ν^R for every morphism f:

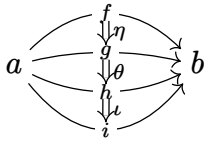


2. Associativity: for all f, g, h : we have $(f \amalg g) \amalg h = f \amalg (g \amalg h)$.
3. In this situation:



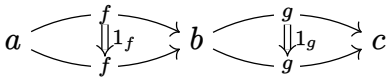
we have $1_f ; \eta_V = \eta_V$. Similarly, $\eta_v ; 1_g = \eta_v$

4. Associatvity at order 2: in this situation:



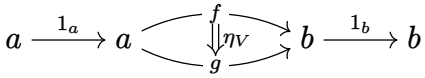
we have $(\eta ; \theta) ; \iota = \eta ; (\theta ; \iota)$.

5. In this situation:



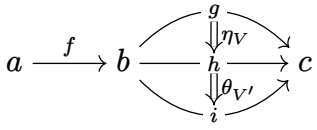
We have $f \triangleright 1_g = 1_{f \amalg g}$ and $1_f \triangleleft g = 1_{f \amalg g}$.

6. In this situation:



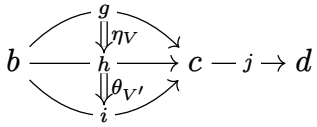
We have $1_a \triangleright \eta_V = \eta_V$ and $\eta_V \triangleleft 1_b = \eta_V$.

7. In this situation:



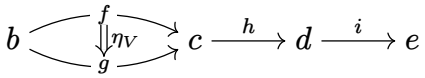
We have $f \triangleright \eta_V ; f \triangleright \theta_{V'} = f \triangleright (\eta_V ; \theta_{V'})$.

8. In this situation:



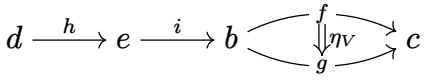
We have $\eta_V \triangleleft j ; \theta_{V'} \triangleleft j = (\eta_V ; \theta_{V'}) \triangleleft j$.

9. In this situation:



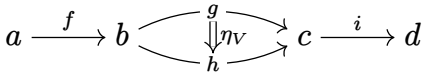
We have $(\eta_v \triangleright h) \triangleright i = \eta_V \triangleright (h \barwedge i)$.

10. In this situation:



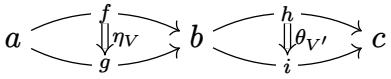
We have $i \triangleleft (h \triangleleft \eta_V) = (h \barwedge i) \triangleleft \eta_V$.

11. In this situation:



We have $f \triangleright (\eta_V \triangleleft i) = (f \triangleright \eta_V) \triangleleft i$.

12. In this situation:



We have $(\eta_V \triangleright h) ; (g \triangleleft \theta_{V'}) = (f \triangleright \theta) ; (\eta \triangleleft i)$.
(This composition is what we mean by parallel composition $||$).

Note
We chose the whisker presentation of 2 categories, which has more axioms, but they are individually simpler to verify.

We are interested in paths in a space with a notion of direction, so in something akin to the fundamental category evoqued earlier in the section on homotopy. Replacing graphs by a category means representing a graph by an object where all paths are computed automatically (which is nice), while keeping the internal workings to not forget circuits. But we do not want to count "not moving" as a circuit. We will thus replace graphs by kategories (semi-categories) instead of categories.

Definition 671 (Quiver)
A *quiver* is an oriented graph. The intention behind the name quiver usually being that it is *the underlying graph of a category.*

Convention
In this manuscript, when we use the word quiver, we will only consider obip graphs, and see those as underlying graphs of *kategories.*

Property (Free adjunction)
There is an adjunction between the forgetful functor U and the free kategory construction $\mathcal{D}$ as follows:

$$\mathbf{Quiv} \underset{U}{\overset{\mathcal{D}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Kat}$$

The letter $\mathcal{D}$ stands for "dynamics", as it is the functor that brings dynamics inside the purely geometrical notion of graph.

Corollary
The functor $\mathcal{D}$ preserves colimits, in particular, it preserves pushouts.

Definition 674 (Hiding the interior of a kategory)
Given a kategory $f = (F, \mathrm{Hom}(-,-))$, and a set $V \subseteq F$, define $H_V(f)$ as the subkategory of f with V as objects.

Definition 675 ($_{10}$Kat)
We now define the Discrete Glue Category of input/output kategories $_{10}\mathbf{Kat}$:

Objects: Any set.

Morphisms: A morphism $\mathbf{K} : I \to O$ is a kategory on a graph K with $I \sqcup O \subseteq V_K$ and the input/output property.

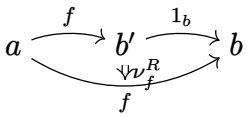
2-**Morphisms:** Two morphisms: there is a two morphism $\alpha : f \Rightarrow g$ if there exists V such that $g = H_V(f)$.

Note that this category is posetal (V is necessarily unique) in the second layer. Thus in the definition, the equalities are verified for free, provided everything is well defined.
It is equipped with the additional data:

- The weak identity $1_A : A \to A$ is the id_A from $_{10}\mathbf{Grph}$.

- The identity for 2-morphism 1_f attest the fact that $\mathbf{f} = H_\emptyset(\mathbf{f})$.
- The notion of composition of 1-morphism $\mathbf{f} : A \to B, \mathbf{g} : B \to C$ is given by the pushout $\mathbf{f} \amalg \mathbf{g}$ of $\mathbf{f}$ and $\mathbf{g}$.
- The composition of 2 morphism $\eta; \theta : f \Rightarrow h$ attests to the fact that if $g = H_V(f), h = H_{V'}(g)$ then $h = H_{V \sqcup V'}(f)$.
- The left whiskering $f \triangleleft \eta : f \amalg g \Rightarrow f \amalg h$ attests to the fact that if $h = H_V(g)$ then $H_V(f \amalg g) = f \amalg H_V(g) = f \amalg h$ since $V \notin f$.
- The right whiskering $\eta \triangleright f : g; f \Rightarrow h; f$ attests to the symmetric fact.

We quickly show some of the coherences:

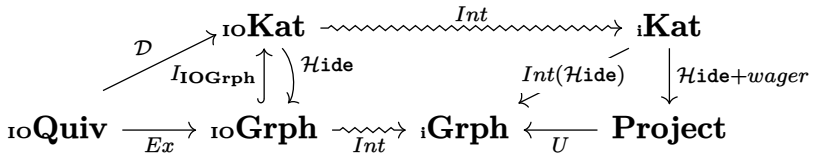
1. 

   The 2-cell ν_f^R attests to the fact that $f = H_{b'}(f \amalg id_b)$. Similarly for ν_f^L.
2. Associativity holds because $\amalg$ is associative.
3. Associatvity of order 2 holds because $H_V(H_B(H_N(f))) = H_{V \sqcup B \sqcup N}$ and $\sqcup$ is associative.
4. The equality $1_f; \eta_V = \eta_V$ holds because $H_\emptyset(H_V(f)) = H_{\emptyset \sqcup V}(f) = H_V(f)$.

In short, all coherences holds because the category is posetal so there is just one morphism of the right type and thus since both side of the equality have the right type they are equal.

Definition 676
$(\text{IO}\mathbf{Kat}, \sqcup, \emptyset)$ is monoidal.

Here is a diagram summing up the situation we will be getting in:



Proposition
The functor $\mathcal{D} : \mathbf{Quiv} \to \mathbf{Kat}$ can be lifted as a monoidal semi-functor $\text{IO}\mathbf{Quiv} \to \text{IO}\mathbf{Kat}$, such that $\mathcal{D}(1_a) = 1_a$ (note 1_a in $\text{IO}\mathbf{Quiv}$ is not an identity).

Proof
This is a consequence of the preservation of pushouts. □

Proposition (Inclusion)
There is an inclusion functor $I_{\mathbf{IOKat}} : \text{IO}\mathbf{Grph} \to \text{IO}\mathbf{Kat}$.

Definition 679 (Hiding)
We define the functor $\mathcal{H}\texttt{ide} : \text{IO}\mathbf{Kat} \to \text{IO}\mathbf{Grph}$ (where $\text{IO}\mathbf{Grph}$ is seen as a 2-category with trivial 2-morphisms) as:

- $\mathcal{H}\texttt{ide}(X) := X$.
- For a kategory $\mathbf{f} : I \to O$, $\mathcal{H}\texttt{ide}(\mathbf{f})$ is the sub-kategory with objects $I \sqcup O$, that is $H_C(\mathbf{f})$ if $\mathbf{f} = (I \sqcup C \sqcup O, \text{Hom}_{\_,\_}(-))$.
- $\mathcal{H}\texttt{ide}(\alpha)$ is the trivial 2-morphism.

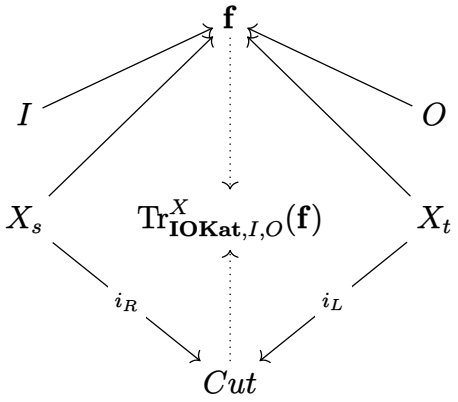
The last step is well defined because if $\mathbf{f} \Rightarrow \mathbf{g}$ then $\mathcal{H}\texttt{ide}(\mathbf{f}) = \mathcal{H}\texttt{ide}(\mathbf{g})$.

We will now define a notion of trace, which is just the equivalent of $\circlearrowleft$ (definition 595) in this setting.

Note
There is no official notion of Trace (that we know of) in the setting of 2-category theory. Nonetheless, this construction *should be one*, and as such, some of the properties such a notion should have could be infered from it.
It is most likely that the right notion is weaker, for we are dealing here with a strictly associative category. A more abstract example that would have to deal with non-strictness probably exists in the context of directed spaces (where associativity does not strictly hold, but only up to reparametrisation).

Hole
There is thus work to do in finding a right abstract notion of trace (and related notions) for 2-categories.

Definition 682 (Trace in $\text{IO}\mathbf{Kat}$)
Given a kategory $\mathbf{f} : I \otimes X \to O \otimes X \in \text{IO}\mathbf{Kat}$, we define $\text{Tr}^X_{\mathbf{IOKat},I,O}(\mathbf{f})$ as the following pushout:



(We add new arrows to loop back on X, and the pushout computes the paths).

Proposition (Hide is monoidal functor)
The functor $\mathcal{H}\texttt{ide}$ is a monoidal functor.

Proposition (Hiding as a retraction)
We have $I_{\mathbf{IOKat}}; \mathcal{H}\texttt{ide} = 1$ (retraction property).

Proposition ($\mathcal{H}\texttt{ide}$ and 2-cells)
Given a kategory $\mathbf{f} : I \otimes X \to O \otimes X \in \text{IO}\mathbf{Kat}$, we have:

$$\text{Tr}^X_{\mathbf{IOKat},I,O}(\mathbf{f}) \Rightarrow^* I_{\mathbf{IOKat}}(\text{Tr}^X_{\mathbf{IOGrph},I,O}(\mathcal{H}\texttt{ide}(\mathbf{f})))$$

(We add a $*$ to the 2-morphism just to indicate that it might be decomposed into smaller 2-morphisms, just like a reduction).

Remark
This is basically the same idea than in the vanishing case of proving that $\text{IO}\mathbf{Grph}$ is traced.

Proof
Let $\mathbf{f} : I \otimes X \to O \otimes X \in \text{IO}\mathbf{Kat}$ be a kategory with $C := \{x \in \mathbf{f} \mid x \notin I, O\}$ the objects that are not directly I or O.
We prove the right hand side of the equation is $= H_C(\text{Tr}^X_{\mathbf{IOKat},I,O}(\mathbf{f}))$.
First, $\mathcal{H}\texttt{ide}(\mathbf{f})$ is just $H_C(\mathbf{f})$ (where the only thing lefts are composition of morphisms of type $I \to O$) seen in $\text{IO}\mathbf{Grph}$. After tracing (geometrically here, not reducing) on X in $\text{IO}\mathbf{Grph}$, one considers paths $I \to O$, which correspond to formal compositions of morphisms $I \to X$, $X \to X$, $X \to O$, where the morphisms $X \to X$ are interleaved with a "fake edge" e^* to do the looping.
These are exactly the morphisms from $I \to O$ in $\text{Tr}^X_{\mathbf{IOKat},I,O}(\mathbf{f})$, as one can see by doing the actual composition of the previously mentionned formal composition.□

Remark
Notice a confusing phenomenon: there is no correspondence between paths $I \to O$ in the geometric shape of $\text{Tr}^X_{\mathbf{IOGrph},I,O}(\mathcal{H}\texttt{ide}(\mathbf{f}))$ and paths in $\text{Tr}^X_{\mathbf{IOKat},I,O}(\mathbf{f})$, because multiple paths in the latter get identified when composed, the correspondence is really with *morphisms* from $I \to O$.

Corollary ($\mathcal{H}\texttt{ide}$ is traced)
From the section property we get as corollary that $\mathcal{H}\texttt{ide}(\text{Tr}^X_{\mathbf{IOKat},I,O}(G)) = \text{Tr}^X_{\mathbf{IOGrph},I,O}(\mathcal{H}\texttt{ide}(G))$.

Observation (Int Construction)
Defining a variant of the **Int** construction in our setting is done by keeping the same definitions, and leaving 2-morphisms unchanged.

Definition 689 (ᵢKat)
We define the category $_{\mathbf{i}}\mathbf{Kat} := \mathbf{Int}(_{\mathbf{IO}}\mathbf{Kat})$.

Definition 690 (Int "as a functor")
Given a monoidal functor $F : \mathbf{C} \to \mathbf{D}$, we define $\mathbf{Int}(F) : \mathbf{Int}(\mathbf{C}) \to \mathbf{Int}(\mathbf{D})$ as follows:

- $\mathbf{Int}(F)((+A, -A)) := (+F(A), -F(A))$.
- $\mathbf{Int}(F)(f) := F(f)$.

Under specific conditions, this construction gives out a functor. This is the case here:

Proposition (Int($\mathcal{H}$ide) as a functor)

$$\mathbf{Int}(\mathcal{H}\mathtt{ide})(F \between G) = \mathbf{Int}(\mathcal{H}\mathtt{ide})(F) :: \mathbf{Int}(\mathcal{H}\mathtt{ide})(G)$$

Proof

$$\begin{aligned}
&\mathbf{Int}(\mathcal{H}\mathtt{ide})(F \between G) \\
&= \mathcal{H}\mathtt{ide}(\mathrm{Tr}^{B}_{\mathbf{IOKat},A,C}(F \otimes 1_C \between G \otimes 1_A)) && \text{def} \\
&= \mathrm{Tr}^{B}_{\mathbf{IOGrph},A,C}(\mathcal{H}\mathtt{ide}(F \otimes 1_C \between G \otimes 1_A)) && \text{by corollary 24} \\
&= \mathrm{Tr}^{B}_{\mathbf{IOGrph},A,C}(\mathcal{H}\mathtt{ide}(F) \otimes \mathcal{H}\mathtt{ide}(1_C) \between \mathcal{H}\mathtt{ide}(G) \otimes \mathcal{H}\mathtt{ide}(1_A)) && \mathcal{H}\mathtt{ide} \text{ functor + monoida} \\
&= \mathrm{Tr}^{B}_{\mathbf{IOGrph},A,C}(\mathcal{H}\mathtt{ide}(F) \otimes 1_C \between \mathcal{H}\mathtt{ide}(G) \otimes 1_A) && \mathcal{H}\mathtt{ide} \text{ is a retraction} \\
&= \mathbf{Int}(\mathcal{H}\mathtt{ide})(F) :: \mathbf{Int}(\mathcal{H}\mathtt{ide})(G)
\end{aligned}$$

□

To sum up what we did: we defined a category $_{\mathbf{i}}\mathbf{Kat}$ which has more information than $_{\mathbf{i}}\mathbf{Grph}$, in particular, it is now possible to define a pole/focus to state that two graphs f, g are orthogonal when $f \between g$ has a cycle (there is a c such that $\mathrm{Hom}(c, c)$ is not empty!).

Hole
By doing this generalization, something interesting happened: the usual interaction graphs would correspond to *free* kategories. But we now can consider non-free kategories. Which means this model is more general. What could we do with this power of being able to make different path "collapse" to one?
There might be links to an operation in [54] which collapses multiple edges into one, to convert an a priori generic graph into a graph with an adjacency matrix.

We can extend this definition to encapsulate more general notions of graphs, such as the weighted models. Assume given a monoid $(\Omega, .)$ seen as a kategory:

Definition 693 (Extended interaction graph over Ω)
An extended interaction graph model is a kategory whose morphisms are pairs $(\mathbf{f}, F)$ with f a morphism from $\text{ıo}\mathbf{Kat}$ and $F : f \to \Omega$ a semi-functor.
Composition is given as $(f, F)⟌(g, G) := (f⟌g, F \sqcup G)$, with $F \sqcup G : f⟌g \to \Omega$ is the "union" functor (see example below).

Example
The union functor is better understood on an example: it associates to the formal composition of morphism in $f⟌g$ an element in Ω as follows: $f_1; f_2; g_1; f_3 \to F(f_1; f_2).G(g_1).F(f_3)$ etc...

Note
It is required for Ω to be a monoid, because any two pair of arrows might be composed, due to the trace. The atomic composition (that of edge) is untyped.

Remark
It was often said in this thesis that interaction graphs compute paths, but I think this shows that it is more than that: *interaction graphs compose arrows.*

Now we could look at the extended interaction graph model whose morphisms are sent to numbers in $\bar{\mathbb{R}}$ with multiplication as composition.

This category would almost correspond to the model of weighted projects. It would in fact be slightly more general, because projects only count cycles, while here we might have any shape, even "non-cycles" inside the kategories. So multiple kategories could give the same projects, for example, the two examples in beware 653, having the same number of circuit, can be interchanged inside a kategory when collapsing it to a project without changing the outcome.

Finally, we could define a functor to the category of projects, which would associate to each kategory a pair of its usual hiding and the sum of the "composition" of its cycles.

Now a comment and possible future development. By considering categories instead of graphs, we managed to reintroduce the dynamics inside a glueing definition.

But the dynamics we capture are "big step": once we compose two interaction kategories, the paths are all instantely computed.

It would be nice to do this in two steps to capture the true dynamics of the model: the composition is the glueing, and the two cells compute the paths little by little, giving at the end an actual semi-category with all paths computed. Note how this allows to look at finer computational properties (for example strong normalization: if there is a loop, there are infinitely many paths and thus computation would not terminate)

We would need for that a notion of kategory with partially defined composition, which corresponds to the notion of Para (semi-)category.

Note the definition is unbiased, which allows to consider n-ary paths.

Definition 697 (Parakategory)
A parakategory C is informally a kategory where composition is a partially defined operation.
Formally, it is given by:

- A Quiver:
$$C_1 \underset{t}{\overset{s}{\rightrightarrows}} C_0$$
(we note C_n the iterated pullback corresponding to strings of length n of composable arrows in the quiver).
- Partial composition operations $;_n : C_n \to C_1$.
- The operation $;_1$ is the identity.
- When $;_n (\vec{y})$ is defined, then if one of the two side of $;_{m+1+k} (\vec{x} +;_n (\vec{y}) + \vec{z}) = ;_{m+n+k} (\vec{x} + \vec{y} + \vec{z})$ is defined then so is the other and they are equal.

See [34] for the original definition of paracategory.

But something "weird" happens; we have two notions of 2-morphisms: one represents the hiding, that is, the identification of which interaction kategories should be seen as "the same" geometrically, while the other represent a computational step of computing paths in said space.

Hole
These morphisms are not completely independent: one can only remove a vertex when all paths that uses said vertex have been computed (else we could not compute said path)
There might be future work to explore this direction. Something interesting would happen if one manages to describe these, as this would create a "decoupling" in the models as described in this thesis.
For now, computation happens precisely when one removes a vertex, so both form of 2-arrows are "tied together" in the concrete non-categorical description of the model.
I would like to add that paracaegories have been used in the context of traces and GoI before, in the paper [44], so there might be links to make.

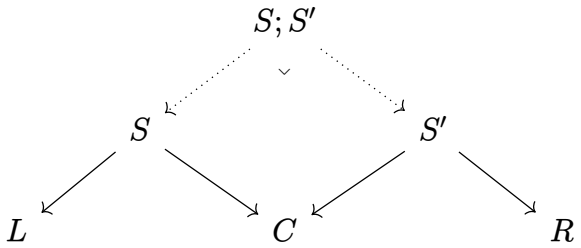
5.8. A focus on the retraction, with Span and Cospan

We would like to try and sum up what we learned in this chapter, and talk about potential connexions.

It seems that interaction graphs, in $_{10}\mathbf{Grph}$ are similar to spans, but lose too much information. We used $_{10}\mathbf{Kat}$, a form of cospan category, to bypass this problem, which are more geometric in nature, and we needed to introduce a 2-layer to our categories to add some of the dynamics back.

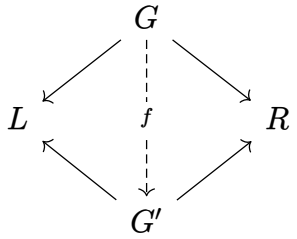
We recall what a span is, formally:

Definition 699 ((Bi)Category of Span)
A span $L \leftarrow C \rightarrow R$ in a category C is given by three objects and two morphisms. They can be composed through pullbacks, which is summed up in the following diagram:



These usually form a weak bicategory (with non trivial associator: the pullback is only defined up to isomorphism). One can quotient by span isomorphism to only deal with 1-categorical considerations (this is the same as saying that our graphs are defined up to renaming of arrows).

Definition 700 (Span Morphism)
A morphism of spans from $G \rightarrow G'$ is a morphism f in $\mathbf{C}$ making the following diagram commute:



This gives rise to a notion of isomorphism by which we can quotient.

Remark
Anecdote: Spans were first introduced by the (in)famous Yoneda of eponymous lemma.

The same can be done for cospans.

Note how $_{10}\mathbf{Kat}$ is a form of 2-category, but it is not a bicategory in the same way than co$\mathbf{Span}(\mathbf{C})$ is a bicategory. Indeed, in the bicategory co$\mathbf{Span}(\mathbf{C})$ the second layer is a layer that is quotiented in $_{10}\mathbf{Kat}$ (renaming of arrows) as explained above.

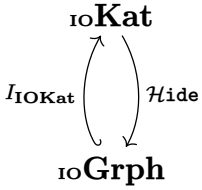
Note
$\mathbf{IO}\mathbf{Kat}$ is not co$\mathbf{Span}(\mathbf{C})$ for a certain $\mathbf{C}$, as the feet are sets while the top is a graph, so feets and top are of a different nature.
This category probably falls under the (fairly recent) notion of structured cospan, see [5], [6], and [11] which was a notion introduced to study open systems.
In this setting, the foot can come from another category and be included in $\mathbf{C}$ via a functor. Here, we would use the functor of inclusion $\mathbf{FinSet} \to \mathbf{Graph}$.

Hole
It is my sincere belief that there are deep connexions to be done between open systems and GoI. I am now convinced that compositionality is at the core of both domains: the glueing of location is a form of compositionality, but in the context of GoI we have dynamics built on top.

In a sense, it seems that *interaction graphs are a form of open-system.*

Finally, we would like to put the focus again on the retraction that was defined in the previous section.

Observation (The retraction)



This retraction has a cospan model on top and span model on the bottom.

It seems that cospans models are a way to reintroduce control over the computational steps, while in span models it is "automatic", which sometimes makes it not expressive enough.

Hole
Is there a way to use such an observation in more traditional denotational settings? I also know from a discussion with Clairambault that it happens sometimes in game semantics that there is a need to avoid hiding in a similar fashion to what was done here (although I do not currently have a reference for that), it would be interesting to see if one can make a general theory for that.

6. Conclusion

There are a lot of holes left unexplored that I stumbled upon during this thesis. I would like to take a few lines to sketch what seems to me to be a feasible research plan on a medium time-scale, trying to unify the work done in the different sections: it would be interesting to create a sort of pipeline to generate models of computation.

1. There is a need to find the right definition of "atomic" model of computation, as sketched in section 3.2. This is a sort of low-level notion of model of computation.

2. From this, using diagrams, one can generate an abstract and compositional model of computation, which is of a higher level. This definition should be abstracted using open systems.
   There should be theorems that prove how an atomic model of computation, from its properties, indeed generates a model that is compositional, has expanding/rectraction properties etc...
   There are a lot of variants of such systems, depending on the choice of rules. There are also less known proof systems, such as deep inference systems, where one can make inference deep in the statements (for example, use the B directly in $(AB)C$), or combinatorial proofs [35]. From these abstract properties, one should prove associativity of execution to show that we indeed have a model of computation in the sense we are interested in.
   There should be a link between this notion and the notion of "abstract model of computation " from Seiller's HDR [58].

3. From this abstract higher level notion, it is important to continue the abstract linear realisability theory that Seiller started, and see what is required at the different levels to obtain results at the highest one, the one of types.

Bibliography

[1] *2-Cocycle for a Group Action - Groupprops*. URL: https://groupprops.subwiki.org/wiki/2-cocycle_for_a_group_action (visited on 08/21/2025).

[2] Roberto Amadio. "Operational methods in semantics". 2016. URL: https://hal.science/cel-01422101v2/.

[3] Clément Aubert and Marc Bagnol. "Unification and logarithmic space". In: *Rewriting and Typed Lambda Calculi*. Springer, 2014, pp. 77–92.

[4] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1998.

[5] John C. Baez and Kenny Courser. *Structured Cospans*. Nov. 10, 2020. URL: http://arxiv.org/abs/1911.04630 (visited on 05/20/2025). Pre-published.

[6] John C. Baez, Kenny Courser, and Christina Vasilakopoulou. "Structured versus Decorated Cospans". In: *Compositionality* 4 (Sept. 1, 2022), p. 3. URL: http://arxiv.org/abs/2101.09363 (visited on 03/17/2025).

[7] Marc Bagnol. "On the Resolution Semiring". In: ().

[8] Marc Bagnol, Amina Doumane, and Alexis Saurin. "On the dependencies of logical rules". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2015, pp. 436–450.

[9] Morton Brown. "Locally Flat Imbeddings of Topological Manifolds". In: *Annals of Mathematics* 75.2 (1962), pp. 331–341. URL: https://www.jstor.org/stable/1970177 (visited on 05/22/2025).

[10] Ronald Brown. "Topology and Groupoids". In: ().

[11] Kenny Courser. *Open Systems: A Double Categorical Perspective*. Aug. 5, 2020. URL: http://arxiv.org/abs/2008.02394 (visited on 08/21/2025). Pre-published.

[12] Vincent Danos. "La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)". PhD thesis. Paris 7, 1990. URL: https://perso.ens-lyon.fr/pierre.lescanne/PUBLICATIONS/DanosPhD.pdf.

[13] Vincent Danos and Laurent Regnier. "The structure of multiplicatives". In: *Archive for Mathematical logic* 28.3 (1989), pp. 181–203.

[14] Paulin Jacobé De Naurois and Virgile Mogbil. “Correctness of linear logic proof structures is NL-complete”. In: *Theoretical Computer Science* 412.20 (2011), pp. 1941–1957. URL: https://www.sciencedirect.com/science/article/pii/S0304397510007115.

[15] Edsger W. Dijkstra. “Solution of a problem in concurrent programming control”. In: *Pioneers and Their Contributions to Software Engineering*. Springer, 2001, pp. 289–294.

[16] Pablo Donato. “Deep Inference for Graphical Theorem Proving”. These de doctorat. Institut polytechnique de Paris, May 29, 2024. URL: https://theses.fr/2024IPPAX015 (visited on 08/13/2025).

[17] Jérémy Dubut. “Directed Homotopy and Homology Theories for Geometric Models of True Concurrency”. These de doctorat. Université Paris-Saclay (ComUE), Sept. 11, 2017. URL: https://theses.fr/2017SACLN032 (visited on 06/24/2025).

[18] Etienne Duchesne. “La Localisation En Logique : Géométrie de l’interaction et Sémantique Dénotationnelle”. These de doctorat. Aix-Marseille 2, Jan. 1, 2009. URL: https://theses.fr/2009AIX22080 (visited on 08/13/2025).

[19] Boris Eng. “An Exegesis of Transcendental Syntax : A Journey into the Logical Machinery”. These de doctorat. Paris 13, June 20, 2023. URL: https://theses.fr/2023PA131018 (visited on 08/18/2025).

[20] Lisbeth Fajstrup et al. *Directed Algebraic Topology and Concurrency*. Cham: Springer International Publishing, 2016. URL: http://link.springer.com/10.1007/978-3-319-15398-8 (visited on 06/24/2025).

[21] Arnaud Fleury and Christian Retoré. “The mix rule”. In: *Mathematical Structures in Computer Science* 4.2 (1994), pp. 273–285.

[22] *Free Deduction: An Analysis of “Computations” in Classical Logic | SpringerLink*. URL: https://link.springer.com/chapter/10.1007/3-540-55460-2_27 (visited on 08/13/2025).

[23] J.-Y. Girard. “Geometry of Interaction III: Accommodating the Additives”. In: *Advances in Linear Logic*. Ed. by Jean-Yves Girard, Yves Lafont, and Laurent Regnier. 1st ed. Cambridge University Press, June 22, 1995, pp. 329–389. URL: https://www.cambridge.org/core/product/identifier/CBO9780511629150A028/type/book_part (visited on 08/08/2025).

[24] Jean-Yves Girard. “Geometry of Interaction 2: Deadlock-free Algorithms”. In: *COLOG-88*. Ed. by Per Martin-Löf and Grigori Mints. Berlin, Heidelberg: Springer, 1990, pp. 76–93.

[25] Jean-Yves Girard. “Linear logic”. In: *Theoretical computer science* 50.1 (1987), pp. 1–101. URL: https://www.sciencedirect.com/science/article/pii/0304397587900454.

[26] Jean-Yves Girard. “Normal Functors, Power Series and Lambda-Calculus”. In: *Annals of Pure and Applied Logic* 37.2 (Feb. 1, 1988), pp. 129–177. URL: https://www.sciencedirect.com/science/article/pii/0168007288900255 (visited on 08/16/2025).

[27] Jean-Yves Girard. “Three lightings of logic (Invited Talk)”. In: *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2013. URL: https://drops.dagstuhl.de/opus/volltexte/2013/4185/.

[28] Jean-Yves Girard. “Transcendental syntax I: deterministic case”. In: *Mathematical Structures in Computer Science* 27.5 (2017), pp. 827–849. URL: https://girard.perso.math.cnrs.fr/trsy1.pdf.

[29] Jean-Yves Girard. “Transcendental Syntax III: Equality”. In: ().

[30] Jean-Yves Girard. “Wo, clef de voûte logique”. In: ().

[31] E. Haghverdi and P. Scott. “Geometry of Interaction and the Dynamics of Proof Reduction: A Tutorial”. In: *New Structures for Physics*. Ed. by Bob Coecke. Berlin, Heidelberg: Springer, 2011, pp. 357–417. URL: https://doi.org/10.1007/978-3-642-12821-9_5 (visited on 08/08/2025).

[32] Masahiro Hamano. “A MALL Geometry of Interaction Based on Indexed Linear Logic”. In: *Mathematical Structures in Computer Science* 30.10 (Nov. 2020), pp. 1025–1053. URL: https://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/mall-geometry-of-interaction-based-on-indexed-linear-logic/94895D6D28B8CD45F314E2D21CD9C7B9 (visited on 05/20/2025).

[33] Susumu Hayashi. “Adjunction of Semifunctors: Categorical Structures in Nonextensional Lambda Calculus”. In: *Theoretical Computer Science* 41 (Jan. 1, 1985), pp. 95–104. URL: https://www.sciencedirect.com/science/article/pii/0304397585900623 (visited on 05/27/2025).

[34] Claudio Hermida and Paulo Mateus. “Paracategories I: Internal Paracategories and Saturated Partial Algebras”. In: *Theoretical Computer Science* 309.1 (Dec. 2, 2003), pp. 125–156. URL: https://www.sciencedirect.com/science/article/pii/S030439750300135X (visited on 06/29/2025).

[35] Dominic Hughes, Lutz Straßburger, and Jui-Hsuan Wu. *Combinatorial Proofs and Decomposition Theorems for First-order Logic*. Apr. 27, 2021. URL: http://arxiv.org/abs/2104.13124 (visited on 08/13/2025). Pre-published.

[36] Martin Hyland and Andrea Schalk. “Glueing and Orthogonality for Models of Linear Logic”. In: *Theoretical Computer Science*. Category Theory and Computer Science 294.1 (Feb. 15, 2003), pp. 183–231. URL: https://www.sciencedirect.com/science/article/pii/S0304397501002419 (visited on 08/21/2025).

[37] Samantha Jarvis. “A Novel Closed Monoidal Structure on the Nucleus of a Profunctor”. In: *Dissertations, Theses, and Capstone Projects* (June 1, 2025). URL: https://academicworks.cuny.edu/gc_etds/6231.

[38] André Joyal, Ross Street, and Dominic Verity. “Traced Monoidal Categories”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 119.3 (Apr. 1996), pp. 447–468. URL: https://www.cambridge.org/core/product/identifier/S0305004100074338/type/journal_article (visited on 05/23/2025).

[39] Paul C. Kainen. *Construction Numbers: How to Build a Graph?* Nov. 30, 2024. URL: http://arxiv.org/abs/2302.13186 (visited on 08/22/2025). Pre-published.

[40] G. M. Kelly and M. L. Laplaza. “Coherence for Compact Closed Categories”. In: *Journal of Pure and Applied Algebra* 19 (Dec. 1, 1980), pp. 193–213. URL: https://www.sciencedirect.com/science/article/pii/0022404980901012 (visited on 08/16/2025).

[41] Yves Lafont. “From proof nets to interaction nets”. In: *Advances in Linear Logic*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1995, pp. 225–248.

[42] J-L. Lassez, Michael J. Maher, and Kim Marriott. “Unification revisited”. In: *Foundations of logic and functional programming*. Springer, 1988, pp. 67–113.

[43] Tom Leinster. *Higher Operads, Higher Categories*. May 2, 2003. URL: http://arxiv.org/abs/math/0305049 (visited on 05/23/2025). Pre-published.

[44] Octavio Malherbe, Philip J. Scott, and Peter Selinger. “Partially Traced Categories”. In: *Journal of Pure and Applied Algebra* 216.12 (Dec. 2012), pp. 2563–2585. URL: http://arxiv.org/abs/1107.3608 (visited on 08/08/2025).

[45] Alberto Martelli and Ugo Montanari. “An efficient unification algorithm”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.2 (1982), pp. 258–282. URL: https://dl.acm.org/doi/10.1145/357162.357169.

[46] Andrzej S. Murawski and C. H. Luke Ong. “Dominator trees and fast verification of proof nets”. In: *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*. IEEE. 2000, pp. 181–191. URL: https://ieeexplore.ieee.org/document/855768.

[47] Alberto Naibo, Mattia Petrolo, and Thomas Seiller. “On the Computational Meaning of Axioms”. In: *Epistemology, Knowledge and the Impact of Interaction*. Ed. by Juan Redmond, Olga Pombo Martins, and Ángel Nepomuceno Fernández. Cham: Springer International Publishing, 2016, pp. 141–184. URL: https://doi.org/10.1007/978-3-319-26506-3_5 (visited on 03/17/2025).

[48] Adrien Ragot, Thomas Seiller, and Lorenzo Tortora de Falco. “Linear Realisability over Nets: Multiplicatives (Extended Version)”. In: *EACSL Annual Conference on Computer Science Logic* (Nov. 26, 2024). URL: https://hal.science/hal-04805459 (visited on 03/17/2025).

[49] Christian Retoré. “Handsome proof-nets: perfect matchings and cographs”. In: *Theoretical Computer Science* 294.3 (2003), pp. 473–488. URL: https://www.sciencedirect.com/science/article/pii/S030439750100175X.

[50] Thomas Seiller. “Interaction Graphs: Additives”. In: *Annals of Pure and Applied Logic* 167.2 (Feb. 1, 2016), pp. 95–154. URL: https://www.sciencedirect.com/science/article/pii/S0168007215000998 (visited on 03/17/2025).

[51] Thomas Seiller. “Interaction Graphs: Exponentials”. In: *Logical Methods in Computer Science* Volume 15, Issue 3 (Aug. 30, 2019). URL: https://lmcs.episciences.org/5730 (visited on 03/17/2025).

[52] Thomas Seiller. “Interaction Graphs: Full Linear Logic”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’16. New York, NY, USA: Association for Computing Machinery, July 5, 2016, pp. 427–436. URL: https://doi.org/10.1145/2933575.2934568 (visited on 03/17/2025).

[53] Thomas Seiller. “Interaction Graphs: Graphings”. In: *Annals of Pure and Applied Logic*. Eighth Games for Logic and Programming Languages Workshop (GaLoP) 168.2 (Feb. 1, 2017), pp. 278–320. URL: https://www.sciencedirect.com/science/article/pii/S0168007216301300 (visited on 03/17/2025).

[54] Thomas Seiller. “Interaction Graphs: Multiplicatives”. In: *Annals of Pure and Applied Logic* 163.12 (Dec. 1, 2012), pp. 1808–1837. URL: https://www.sciencedirect.com/science/article/pii/S0168007212000759 (visited on 03/17/2025).

[55] Thomas Seiller. “Interaction graphs: multiplicatives”. In: *Annals of Pure and Applied Logic* 163.12 (2012), pp. 1808–1837. URL: https://www.sciencedirect.com/science/article/pii/S0168007212000759.

[56] Thomas Seiller. “Logique Dans Le Facteur Hyperfini : Géométrie de l’ Interaction et Complexité”. These de doctorat. Aix-Marseille, Nov. 13, 2012. URL: https://theses.fr/2012AIXM4064 (visited on 08/08/2025).

[57] Thomas Seiller. “Logique dans le facteur hyperfini: géometrie de l’interaction et complexité”. PhD thesis. Aix-Marseille Université, 2012. URL: https://theses.hal.science/tel-00768403/.

[58] Thomas Seiller. “Mathematical Informatics”. thesis. Université Sorbonne Paris Nord, June 18, 2024. URL: https://theses.hal.science/tel-04616661 (visited on 03/17/2025).

[59] Thomas Seiller. “Zeta Functions and the (Linear) Logic of Markov Processes”. In: *Logical Methods in Computer Science* Volume 20, Issue 3 (Aug. 29, 2024), p. 10303. URL: https://lmcs.episciences.org/10303 (visited on 08/21/2025).

[60] Peter Selinger. “A Survey of Graphical Languages for Monoidal Categories”. In: vol. 813. 2010, pp. 289–355. URL: http://arxiv.org/abs/0908.3347 (visited on 06/24/2025).

[61] Masaru Shirahata. “Geometry of Interaction Explained”. In: ().

[62] Sten-Åke Tärnlund. “Horn clause computability”. In: *BIT Numerical Mathematics* 17.2 (1977), pp. 215–226.

[63] René Thom. “Quelques propriétés globales des variétés différentiables”. In: *Commentarii Mathematici Helvetici* 28.1 (Dec. 1, 1954), pp. 17–86. URL: https://doi.org/10.1007/BF02566923 (visited on 05/21/2025).