Continuity in Type Theory

Martin Baillon

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

THESE DE DOCTORAT

NANTES UNIVERSITE

ECOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies du numérique, de l'Information et de la Communication
Spécialité : Informatique

Par
Martin BAILLON

Continuity in Type Theory

Continuité en théorie des types

Thèse présentée et soutenue à Nantes, le 21 décembre 2023
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Rapporteurs avant soutenance :

Thierry Coquand	Full Professor, University of Gothenburg, Gothenburg.
Andrej Bauer	Full Professor, University of Ljubljana, Ljubljana.

Composition du Jury :

Président :	Benjamin Werner	Directeur de recherche, INRIA, détaché à l'École polytechnique, Palaiseau.
Examinateurs :	Thierry Coquand	Full Professor, University of Gothenburg, Gothenburg.
	Andrej Bauer	Full Professor, University of Ljubljana, Ljubljana.
	Théo Winterhalter	Chargé de recherche, Inria Saclay.
	Étienne Miquey	Maître de Conférence des Universités, Institut de Mathématiques de Marseille.
	Vincent Rahli	Associate Professor, University of Birmingham, Birmingham.
Dir. de thèse :	Assia Mahboubi	Directrice de recherche, Inria Rennes-Bretagne-Atlantique.
Co-dir. de thèse :	Pierre-Marie Pédrot	Chargé de recherche, Inria Rennes-Bretagne-Atlantique.

Invité(s)

Ambrus Kaposi	Associate Professor, Eötvös Loránd University, Budapest.

Doctoral thesis

Continuity in Type Theory

Pythias, trees and a lot of discussion

Martin Baillon

Acknowledgements

First and foremost, I should thank Assia and Pierre-Marie for these three years and three months of supervision, providing both strong scientific and emotional support, as well as linguistic and political enlightenment. As far as my thesis goes, I could not name a more iconic duo than you.

Regarding the manuscript, I am very grateful to the reviewers Andrej and Thierry for accepting to spend some time reading this long document and for providing intuitions and remarks about this work. I would also like to thank the whole jury: Ambrus, Vincent, Benjamin and Théo for their useful comments about my PhD. I had very fruitful interactions with all of you, and my short stays in Birmingham with Vincent and Ljubljana with Andrej helped me greatly in understanding continuity.

I would also like to thank Martín and Pierre-Évariste for being members of my *Comité de Suivi Individuel*, you helped me take some steps back about my work and I am grateful for that. Moreover, a large part of this thesis is based on some of Martín's work, so I should thank you for this too.

Finally, I would like to thank all researchers I met during my thesis, be it in the Gallinette team, in a conference somewhere or during a short stay in a lab. Type theorists are in my experience often sarcastic, tongue-in-cheek, ready to discuss almost everything, possess a lot of good alcoholic beverages and kindness in their heart. I am proud to be part of this great community.

Remerciements

Proximité géographique faisant foi, je tiens tout d'abord à remercier les différentes personnes à avoir partagé mon bureau au cours de ces trois années de thèse. Au commencement était Meven, solide titulaire de la super-chaise-près-de-la-fenêtre, que je me suis empressé de récupérer à son départ. Il n'était pas encore devenu le franc colocataire qu'il fut ensuite, mais il montrait déjà ses caractéristiques les plus évidentes : un grand sourire, une maigreur aussi inquiétante que les slides de Mekaouche, un rire reconnaissable entre tous, un amour incompréhensible pour le typage bidirectionnel et un certain manque de suite dans les idées quand il s'agit de faire des reprises d'Iron Maiden. Merci à toi, *senpai numéro 1.*

Merci aussi à Pierre, cothésard éclectique, toujours prêt à proposer de nouveaux modèles de calcul Turing-complet, à improviser un jazz au Mélodica ou à venir assister à un cours de théâtre. J'espère que tu trouveras un lieu où abriter l'incroyable énergie d'apprendre et d'enseigner dont tu resplendis. Merci à Nils pour son apprentissage parcellaire du français, toi dont les "Bien ou bien ?" retentissent encore dans les murs de ce bâtiment ; merci à Yann-le-mangeur-lent, à Josselin-le-catégoricien-shiny et à Sidney-le-deuxième-mangeur-lent pour les discussions sur des vrais langages de programmation concrets de la vraie vie véritable. La super-chaise-près-de-la-fenêtre sera bientôt libre, ne vous battez pas. Merci à Loïc enfin, *senpai numéro 2*, prophète de l'égalité observationnelle, toi plus pâle que la neige et plus solide que cette métaphore, acteur d'une journée héroïque passée à écrire sous forme de vagues et contempler des canards. Clairement la meilleure Ascension possible. Hâte de venir te voir dans ton igloo nordique un de ces jours.

Merci aussi à tous les stagiaires et doctorants en visite venus animer ce bureau, Arthur, Peio, Thomas, Robin, Tomas, Tomas (le deuxième surtout). Merci à Théo de nous avoir laissé te pousser en haut de la colline d'Édimbourg, le paysage était sublime et on s'est bien amusé.

Dans l'enfilade de bureaux adjacents du bâtiment 11, j'appelle Enzo à la barre. Merci à toi, chair de ma chair, frère de thèse, indépendantiste breton diplômé avant moi à la photo finish. Ton apparition, café à la main, dans l'encadrement de la porte pour parler de tes derniers algorithmes paramétriques était toujours une éclaircie dans la journée la plus sombre, et notre périple écossais, en randonnée dans les Highlands, debouts dans un train surchauffé ou à te nourrir à la becquée covidé dans ta chambre, reste un des meilleurs souvenirs de 2022. Toi aussi tu pars t'enterrer dans le froid et la neige, drôle d'idée à mon avis mais qui suis-je pour critiquer les choix d'un docteur ? Merci à toi, Hamza, grand connaisseur de musique classique et détenteur d'un tapis du plus bel effet ; j'avoue que je n'ai jamais compris ton sujet de thèse, merci d'avoir été moins nul et de t'être intéressé au mien. Merci à Nicolas, figure tutélaire, chef d'équipe en marcel et tongs, d'une disponibilité à toute épreuve, merci à Kazuhiko pour m'avoir montré comment faire du vrai café, à Kenji-le-brûleur-de-blattes pour les discussions autour d'extensions de Kan dans une direction quelconque et de splits dans des théories "à poils bleus", à Mathieu pour les éclairages mathématico-mathématiques sur la continuité, merci à Yannick pour m'inculquer un peu de rigueur dans ma rédaction de thèse et pour demander *when is the party?* à la fin du processus, merci à Koen, Yee Jian et Nicolas. Merci à Xavier désormais sans bureau pour ton point de vue intéressant sur les notations et les débats que tu suscites de temps en temps sur le canal doctorants.

Enfin, dernier cité du bâtiment 34 mais pas le moindre, merci Pierre-Marie pour avoir été Pierre-Marie du début à la fin, avec tes imitations d'accents indiens aux subtilités inaudibles pour nos oreilles d'européens sourdingues, ta théière à piston qui se décompose jour après jour et tes statégies politiques inaccessibles au tout-venant. Merci d'avoir été un mauvais encadrant de thèse qui m'a laissé écrire les pires jeux de mots dans mon manuscrit, pour ton amour du Sur Mesure et ta chartreuse toujours disponible en séminaire au vert, pour tes histoires de voiture perdue en Espagne les répliques de la *Classe Américaine* que tu cries dans les couloirs et les menaces de mort sur ma tête que tu échanges avec Andrej. Merci pour les mouvements de doigts que tu fais lorsque tu parles des catégories ; rarement les mathématiques auront été aussi proches de la pantomime.

Par la fenêtre, j'aperçois de l'autre côté de la passerelle le bureau d'Assia, Guilhem et Guillaume. Merci Guillaume pour tes remarques caustiques, pour toutes les fois où tu m'as caché ma sacoche de vélo et pour m'avoir montré que certaines personnes faisaient n'importe quoi avec OCaml, merci Guilhem pour ta bonne humeur et ton courage lorsqu'il a fallu monter au front contre Mekaouche-le-terrible. Merci à Assia pour m'avoir donné un sujet de stage de théorie des types alors que je t'avais envoyé un mail à propos d'induction sur les réels, merci d'avoir permis qu'une thèse commencée en plein confinement se passe aussi bien. Merci d'avoir répondu à mes questions, d'avoir posé à Pierre-Marie les questions nécessaires pour débroussailler sa parole touffue, de m'avoir abrité des tempêtes administratives, de m'avoir dit de rédiger quand il fallait rédiger et d'arrêter de rédiger quand il fallait arrêter de rédiger, merci d'avoir réservé les salles sur GRR et de m'avoir proposé de rester comme ingénieur de recherche après ma thèse. Merci de ne pas avoir compté tes heures lorsque tu devais gérer à la fois ta recherche, des enfants qui s'ingénient à tomber malade au pire moment et deux thésards au bord du précipice. Visiblement je ne suis pas le premier doctorant que tu encadres qui parte faire des bêtises artistiques une fois diplômé ; crois bien que ce n'est pas parce que tu nous fait peur.

Toujours au sein du bâtiment 11, merci à Gaëtan pour tes éclairages sur le fonctionnement de Coq, merci à Matthieu - le seul vrai rockeur de Gallinette - pour les parties de flipper endiablées, les cours de Forró, pour avoir nommé ta fille Cléopâtre et pour ta gentillesse, tout simplement. Merci à Anne-Claire pour les multiples remboursements de frais de mission qui m'ont rendu riche aux dépens du contribuable, merci à Virginie pour ses éclairages administratifs sur le fonctionnement complexe de l'université.

Quittant le labo par le nord, on atteint rapidement le gymnase de la Barboire, qui abrite l'une des équipes de volley les plus admirables que j'aie rencontrées. Merci à Alice et Alex les capicoachs, à Tom qui smashe super bien, Dimitri le réparateur de grues, Arnaud le protecteur de biodiversité, Orama l'archéologue, Corentin le sudiste, Monika et Bastien pour avoir formé une équipe de RAV4 d'exception. Et bien sûr, merci à Audrey la boîte aux lettres, Enora la puncheuse, Ilaria *vai vai vai*, Simon le bourrin, Marine la gagneuse, Paul le contreur fou, Laure l'entorse aux règles et Tom le footeux pour avoir repris le flambeau et être devenus une si belle équipe de raviolis.

Merci aux participants de mes autres activités que je ne pourrai bientôt plus appeler "extra-scolaires", Nina, Mathieu, Lucie et les autres d'avoir joué Shakespeare avec moi, et merci à Marie, Claire, Carine, Aurore, Cyrille, Tom et Jade pour ces soirées danse endiablées.

Ce tour rétrospectif de la ville de Nantes ne serait pas complet sans un immense merci aux colocataires qui se sont succédés au sein de La Boulange. Merci à Meven, à nouveau, membre fondateur de cette colocation, merci à Joanne pour ton intégrité, ta gentillesse, les footings-crêpes-nems du jeudi matin et les longues discussions sur le monde merveilleux de la fantasy. Ce chapitre 1 est pour toi, bien sûr, tu sauras l'apprécier. Merci à Léo pour avoir été le coloc le plus constant, mi-hobbit mi-gobelin, pour m'avoir indiqué mille bons plans nantais et m'avoir permis de rencontrer moult personnes fascinantes. J'espère que la vie en éco-lieu se passera bien. Merci à Léa, propriétaire mais toujours révolutionnaire, de te battre pour rallumer les étoiles, merci à Élisa d'avoir monté *Les couilles sur la table* au fond de la grotte du Fluffystan, merci à Emma d'avoir apporté sa bonne humeur depuis Marseille, à Natan d'avoir tenté de me vendre le SNU et à Elsa de t'accrocher bon an mal an à ce service civique aux airs de torture. Merci à Lara d'avoir peint le mur en bleu, de m'avoir initié à la course et aux gâteaux citron-basilic, d'avoir joué au foot comme on conduit un TGV et d'avoir une petite soeur encore plus cool que toi ; merci, enfin, pour ces quelques mois passés ensemble et hâte de venir te voir à Dresde.

Nantes n'étant, finalement, qu'une banlieue un peu éloignée de Paris, il est temps d'aborder une nouvelle salve de remerciements à tous les parisiens que j'ai pu côtoyer durant cette thèse. En commençant par les cinéastes, merci d'abord à Pierre, fidèle compagnon de route artistique, homme aux multiples talents, auteur, réalisateur, acteur, chanteur, guitariste et même développeur web à ses heures perdues. Merci de m'avoir attendu ces trois ans, ça y est c'est fini, promis, on va pouvoir faire des films. Merci à Claire le koala philosophe pour ta production industrielle de mèmes qui m'a maintenu à flot lors de la rédaction, pour avoir essayé de lire ma thèse et m'avoir fait bénéficier de ton vaste empire immobilier. Merci pour

avoir cité régulièrement Skoll, le Lillet Tonic et Paul Schräder, ta sainte trinité à toi. Merci surtout pour avoir été et être toujours la personne la plus stylée de la vie en jaune. Merci à Clélia, enfin, pour apporter un peu de vie dans ce groupe de penseurs dépressifs et pour ne pas m'avoir tué à de multiples reprises lorsque tu en as eu l'occasion. Merci à Paco d'avoir filmé quand on lui demandait de filmer, à Roman d'avoir été notre sherpa tout autant que notre script doctor, à Marie d'être la meilleure personne sur terre et à toute l'équipe de L'entorse pour nous avoir permis de faire ce film qui nous ressemble.

Merci à Cécile et Guillaume pour tous les repas dans votre appartement, pour le confinement dans un chalet montagnard et pour avoir prévu votre mariage après ma soutenance, c'est quand même plus simple pour s'organiser. Merci à Grégoire et Nico pour les randos-vélos, à Antoine et Guillaume pour m'avoir donné une raison de revenir à Massy-Palaiseau, et plus généralement à toute la section bad de rester ce groupe d'amis sur lesquels compter. Merci à FMF, Maverick et toute la lignée pour les Xiao Long Bao et merci à Charles de filer du boulot à Persistance sous le SMIC horaire. Tu es le destinataire du plus long poème que j'ai écrit, et tu en mérites chaque ligne. Merci à Lucas de rester la personne la plus musclée que je connaisse, à Armand d'être un frère d'armes inamovible, à Louis de nous rappeler chaque jour l'absence d'analyse modale chez les ingénieurs italiens, à Meryem d'être la meilleure respo archives que le monde ait connu et à tous les membres du JTX d'avoir fait des vidéos de qualité discutable mais toujours avec enthousiasme. Merci à Sylvestre pour ton amour inconditionnel du badminton, pour les tournois partout en France et les conseils que tu m'as prodigués. Hâte de te voir à Berlin ! Merci aussi à tout le reste de l'EBPS12, Florian, Théo, Thomas, Camille et tous les autres de partager votre passion avec bonne humeur.

Merci à tous les participants des différents confinements, qu'ils soient dans les Alpes ou en Charentes-Maritimes, pour avoir fait de ces mois de restrictions des moments hors du temps, à mi-chemin du travail et des vacances.

Merci à Antonin d'avoir fait ma thèse à ma place sans que personne ne s'en rende compte, et pour l'achat prochain d'une Switch.

Merci à Sylvie, Violette et Maud pour votre enthousiasme à la vue de la moindre brocante et pour avoir tenté de me donner du style.

Merci à mes parents de m'avoir toujours soutenu, à mon frère de m'encourager à attaquer en justice tout ce qui bouge, et à mes grands-parents qui sont, ou seraient, j'espère, fiers de moi.

Merci aux Halogen Mushrooms, aux Brews Brothers et à tous les autres.

Merci à tous les professeurs qui m'ont transmis leur amour du savoir au cours de mes années d'études, Mme Giboraud, M. Deseez, Mme Plassard, M. Combault, Mme Kolago, M. Real, M. Taïeb, M. Bournez et tous les autres.

Enfin, je n'oublie pas qu'en tant qu'homme blanc de famille aisée, parisien, cisgenre et hétérosexeuel, j'étais dans une position privilégiée pour faire cette thèse. Espérons qu'un jour il en soit autrement.

Résumé en français

Des types et des hommes La *théorie des types de Martin-Löf* (MLTT) est un système formel dont les objets peuvent être interprétés indifféremment comme des preuves de théorèmes ou des programmes informatiques. Par exemple, étant donnés deux objets A et B (appelés *types)*, l'objet

$$A \to B$$

peut être compris comme le type des fonctions de A dans B, ou comme le type des *preuves que A implique B*.

En ce sens, la théorie des types est à l'avant-garde d'un domaine de recherche en logique informatique qui prend pour paradigme l'idée fondatrice qu'un *théorème est un type et une preuve est un programme*. Ce prisme de lecture a pour nom, la *correspondance preuve-programme*, ou encore *l'isomorphisme de Curry-Howard*, du nom de ses découvreurs, Haskell Curry et William Alvin Howard.

Cependant, si cette correspondance s'est révélée un outil conceptuel fécond qui a mené à de nombreuses découvertes et inventions, la moindre n'étant pas les *assistants à la preuve* (comme Coq ou Agda), elle n'est à ce jour pas complète et se heurte à quelques principes logiques dont le comportement calculatoire est difficile à définir. Ces principes logiques sont appelés *classiques* et s'opposent aux principes *constructifs*, dont le comportement calculatoire est bien connu.

Les plus célèbres des principes classiques sont sans doute *l'axiome du choix* et *le tiers-exclu*, aussi appelé *élimination de la double-négation*, ou encore *preuve par l'absurde*. C'est ce dernier qui permet, à partir d'une preuve qu'il est impossible que les licornes n'existent pas, de déduire que les licornes existent. Cependant, on ressent bien dans cet exemple le caractère *non constructif* de cette preuve: les licornes existent, quelque part, mais nous ne savons pas où : leur existence a quelque chose de vaporeux, d'éthéré, qui convient mal à l'utilisation dans des calculs sur machine.

Ainsi, en Coq, une preuve de l'existence d'un entier vérifiant une certaine propriété donnera accès à la valeur concrète de cet entier.

Or, il y a aujourd'hui, du point de vue d'un mathématicien, deux intérêts principaux aux assistants à la preuve : premièrement, ils sont capables de certifier qu'une preuve est correcte. Deuxièmement, par le biais de la correspondance preuve-programme, cette preuve correspondra à un calcul, et on peut l'utiliser pour obtenir des résultats concrets. Si l'on ne se soucie que de la première propriété, on peut certes rajouter le tiers-exclu comme un axiome, au prix d'une utilisation un peu plus fastidieuse du logiciel. Mais si l'on souhaite bénéficier des deux avantages suscités, les principes classiques sont pour l'instant à bannir.

Tout espoir n'est pas perdu, pour autant. Des travaux récents [115, 121] ont mis en lumière la relation qui existe entre les principes logiques classiques d'une part, et les *effets* en programmation d'autre part. Les effets sont des outils de programmation qui ne se contentent pas de calculer une valeur et de la renvoyer, mais qui jouent sur l'état global du logiciel : on peut citer la levée d'exceptions, l'écriture dans un journal (*logbook*), l'affichage à l'écran (*print*), l'interaction avec un utilisateur ou un autre programme, la sauvegarde d'un état du système avec

[115]: Pédrot (2020), "Russian Constructivism in a Prefascist Theory"
[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

possibilité de revenir à cet état plus tard (*backtracking*)... Pour la plupart de ces effets, un équivalent logique a pu être trouvé, souvent une variante affaiblie du tiers-exclu ou de l'axiome du choix. Leur intégration au sein de Coq ou d'Agda n'est pas aisée, mais l'espoir d'obtenir un jour un assistant à la preuve capable de calculer avec n'importe quelle preuve, ou de vérifier n'importe quel programme, n'est pas mort.

Bien sûr, intégrer des effets en théorie des types est également intéressant en soi : du point de vue du développeur, Coq est un langage qui permet de spécifier très finement le comportement désiré du programme que l'on est en train d'écrire. Malheureusement, il ne permet de vérifier que des programmes sans effets, qui sont pourtant utilisés partout dans les logiciels de la vie de tous les jours. Intégrer les effets à Coq permettrait de certifier sans bug de larges pans de l'informatique, augmentant la confiance que l'on peut raisonnablement accorder à des logiciels qui, pour certains, mettent en danger des vies humaines lorsqu'ils fonctionnent mal.

Le chapitre 1 raconte la théorie des types, ses particularités et son histoire. On y propose notamment en section 1.4 des exemples concrets d'effets auxquels un contenu logique a pu être associé.

L'hypothèse du continu C'est dans ce cadre que nous nous intéressons dans cette thèse à la *continuité*, que nous prenons comme un effet potentiellement porteur d'un principe logique semi-classique, que nous espérons intégrer à terme dans la théorie des types.

La continuité d'une fonction

$$f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

énonce que *f ne peut interroger qu'un nombre fini de fois son argument avant de renvoyer une valeur*. Prenons le temps de détailler cet exemple : le type de f est

$$(\mathbb{N} \to \mathbb{B}) \to \mathbb{N}.$$

C'est donc un type fonctionnel de la forme $A \to B$ comme expliqué plus haut. En l'occurence,

$$A \coloneqq \mathbb{N} \to \mathbb{B} \qquad \text{et} \qquad B \coloneqq \mathbb{N}.$$

Cela signifie que f est une fonction qui, étant donné un argument

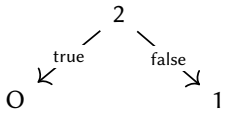
$$\alpha : \mathbb{N} \to \mathbb{B},$$

produira un entier $f\,\alpha : \mathbb{N}$. L'argument α est lui-même une fonction, qui étant donné un entier $n : \mathbb{N}$, renvoie un booléen, c'est à dire une valeur de vérité true ou false. En termes mathématiques, on dira que f est une fonction de l'espace de Cantor dans les entiers.

Un exemple concret d'une telle fonction serait

$$f \coloneqq \alpha \mapsto \texttt{if}\ \alpha\ 2\ \texttt{then O else}\ 1.$$

On peut représenter graphiquement cette fonction sous cette forme :

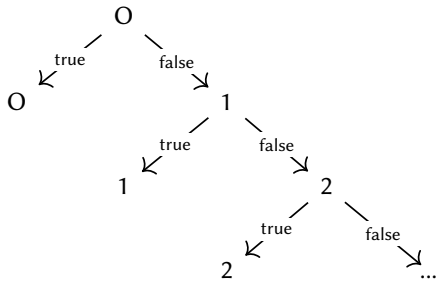


Une telle représentation graphique s'appelle un *arbre de dialogue*. Ici, l'étiquette 2 à la racine de l'arbre indique que f interroge son argument α sur 2. Puis, les deux réponses possibles de α sont représentées par les deux branches de l'arbre. Enfin, les valeurs de retour de f sont données par les feuilles de l'arbre, à savoir O et 1.

On dit qu'une fonction est *continue* si elle peut être représentée par un tel arbre, tel que toutes les branches de l'arbre soient finies. Un cas typique de fonction non continue serait :

$$\begin{array}{lll} f & \coloneqq & \alpha \mapsto h\,\alpha\,\mathrm{O} \\ & & \text{where} \\ h & \coloneqq & \alpha,\ n \mapsto \texttt{if}\ \alpha\ n\ \texttt{then}\ n\ \texttt{else}\ h\,\alpha\,(n+1). \end{array}$$

Ici, f renvoie la première valeur pour laquelle α est vraie. Malheureusement, si α est la fonction constante qui renvoie toujours false, le calcul bouclera indéfiniment. Un embryon de représentation graphique serait celui-ci:



La branche la plus à droite est infinie, f n'est donc pas continue.

Une des particularités des assistants à la preuve comme Coq ou Agda est que toutes les fonctions que l'on peut écrire dans ces logiciels renvoie une valeur en un temps fini. Intuitivement, cela signifie que toutes les fonctions définissables en Coq sont continues. On pourrait espérer profiter de ce fait pour récupérer des principes logiques proches du tiers exclu. Ainsi, un principe équivalent à ce dernier est *l'élimination de la double négation*. Informellement, ce principe énonce que si une affirmation n'est pas fausse, alors elle est vraie. En Coq, étant donné un type A, cela s'écrit

En théorie des types, on parle de *méta-théorie* pour désigner le système logique externe, celui qu'on utilise pour raisonner sur Coq depuis notre cerveau de logicien.

$$((A \to \bot) \to \bot) \to A.$$

Le type $\bot$ est le type vide, qui ne contient aucun élément, et la négation d'une proposition H s'écrit $H \to \bot$. L'intuition est que nier H revient à dire "si H est vrai alors je peux prouver n'importe quoi, même la présence d'un élément dans le type vide".

Or, en regardant attentivement le type de la double négation de A,

$$(A \to \bot) \to \bot,$$

on peut voir qu'il est formé de la même manière que

$$(\mathbb{N} \to \mathbb{B}) \to \mathbb{N},$$

le type de notre fonction f donnée en exemple précédemment. On peut donc parler de continuité pour les fonctions qui prennent en entrée une preuve de $A \to \bot$ et renvoient en sortie une preuve de $\bot$. De la même façon que pour f, étant donné

$$F : (A \to \bot) \to \bot,$$

on peut dessiner l'arbre de dialogue de F et en déduire un habitant de A. Il y a essentiellement deux possibilités :

- Soit F n'interroge pas son argument $\alpha : A \to \bot$ et produit directement une preuve $e : \bot$, auquel cas son arbre ressemble à ceci :

  Or, $\bot$ est le type vide ; si F est capable de construire un élément de $\bot$ sans utiliser son argument c'est que notre théorie est incohérente. On peut donc y prouver n'importe quoi, et en particulier le fait que A est habité;
- Soit F interroge son argument $\alpha : A \to \bot$ sur un certain $a : A$, auquel cas son arbre de dialogue ressemble à cela :

  Or, nous cherchons à produire un habitant de A et F nous en fournit un sur un plateau. Nous nous contentons donc d'utiliser $a : A$.

Ainsi, en utilisant la continuité de toutes les fonctions définissables en Coq, on peut espérer dompter le tout-puissant tiers-exclu. Le chapitre 2 détaille les différentes définitions historiques de la continuité, leurs différences et leur puissance logique respective. La définition de la continuité par arbres de dialogues y est donnée plus formellement, et une version plus précise de la preuve ci-dessus y est proposée en section 2.4.

Ladite preuve présente cependant deux limites. Tout d'abord, nous n'avons proposé ici qu'un raisonnement intuitif, et il manque encore beaucoup de travail pour transformer cela en une preuve véritable. D'autre part, la continuité de toutes les fonctions définissables en Coq est une connaissance que nous avons d'un point de vue *externe* à Coq. Or, depuis le théorème d'incomplétude de Gödel, on sait que certaines propriétés d'une théorie qui sont prouvables depuis l'extérieur (la cohérence, dans le cas de Gödel), ne sont pas forcément vraies vues de l'intérieur (dans le cas du théorème d'incomplétude, on peut prouver depuis l'extérieur qu'une théorie est cohérente, mais si une théorie suppose sa propre cohérence, elle devient aussitôt incohérente).

En l'occurence, *Escardó et al* [52] ont prouvé que supposer en Coq que

[52]: Escardó et al. (2015), "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation"

toutes les fonctions

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

définissables en Coq étaient continues menait à une incohérence. Cette preuve est reproduite dans le chapitre 2, en section 2.4.

De manière intéressante, ce résultat négatif intervient alors même que des preuves formelles existent d'un point de vue externe que toutes les fonctions

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

définissables dans des systèmes plus petits que Coq sont continues [51].Cela nous invite à pousser nos efforts dans deux directions :

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

- essayer d'élargir à Coq entier la preuve externe que toutes les fonctions sont continues;
- essayer de comprendre quelles sont les conditions sur A, B et C pour qu'on puisse supposer en Coq que toutes les fonctions

$$f : (A \to B) \to C$$

sont continues.

Dans cette thèse, nous nous sommes principalement attaqués au premier de ces deux points. Le chapitre 3 étend la preuve d'*Escardó* [51] que toutes les fonctions définissables en System `T` sont continues à un système logique plus complexe, appelé *Baclofen Type Theory* (BTT). Nous utilisons pour ce faire un certain nombre d'outils développés par *Pédrot et Tabareau* [116, 121] sur l'intégration d'effets en théorie des types dépendants. Nous obtenons une preuve externe que toutes les fonctions

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

[116]: Pédrot et al. (2020), "The fire triangle: how to mix substitution, dependent elimination, and effects"
[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

définissables en BTT sont continues, mais BTT n'est pas le système logique implémenté par Coq. Il en diffère sur un point crucial : la preuve par récurrence, aussi appelé *élimination dépendante*. Le problème se pose pour une large classe d'effets, appelés *observables*, et est assez clair dans le cas des booléens : une règle importante de théorie des types est que, pour n'importe quel type A, l'on doit pouvoir définir des fonctions de type $\mathbb{B} \to A$ par le biais d'un `if` ... `then` ... `else` .... C'est le cas de notre fonction

$$f \coloneqq \alpha \mapsto \texttt{if}\ \alpha\ 2\ \texttt{then O else 1}$$

utilisée plus haut. Cependant, l'attente implicite de cette règle est que les seuls booléens qui existent en théorie des types sont true et false. Ainsi, le `if` ... `then` ... `else` ... couvre l'intégralité des cas possibles. Malheureusement, rajouter des effets brise cette règle : si l'on rajoute des levées d'exceptions, par exemple, le terme `raise` e peut tout à fait avoir le type $\mathbb{B}$. Il faut alors fournir une règle de calcul pour le cas où on se retrouve avec

$$\texttt{if raise}\ e\ \texttt{then} \ldots \texttt{else} \ldots$$

et cela ne se fait pas sans heurt.

Dans notre cas, nous ne rajoutons pas des exceptions mais des structures d'arbres de dialogue dans tous les types. L'effet est le même, et

l'interaction avec le `if ... then ... else ...` est délicate. BTT permet de résoudre le problème, au prix d'un affaiblissement logique de la théorie. Le fonctionnement de Baclofen Type Theory est expliqué plus en détail en section 1.4 du chapitre 1, la preuve de continuité des fonctions définissables en BTT et ses limites sont détaillées dans le chapitre 3.

Tout est normal L'échec relatif de l'extension de cette preuve à la totalité de la théorie des types de Martin-Löf nous fait changer notre fusil d'épaule : ajouter superficiellement les arbres de dialogue en tant qu'effets à l'aide de modèles syntaxiques est trop faible, une preuve de continuité nécessite plus de contrôle sur le comportement calculatoire des objets considérés.

La preuve externe que toutes les fonctions définissables en Coq retournent une valeur en un temps fini repose sur une propriété de la théorie des types appelée *normalisation*. Celle-ci énonce que tout terme de théorie des types peut être transformé en un autre terme, convertible au premier, qui se trouve en forme *normale*, c'est à dire une version canonique du premier terme. Les preuves de normalisations sont connues pour être des exercices compliqués, qui réclament des principes logiques puissants dans la méta-théorie et présentent de forts degrés d'abstraction. La structure la plus commune utilise des *relations logiques*, et interprète un type comme la donnée de trois prédicats : un prédicats d'égalité à ce type, un prédicat d'appartenance à ce type en tant que terme, et un prédicat d'égalité sur les termes à ce type.

Cette structure de preuve a été implémentée en Agda par *Abel et al* [3], puis étendue de différentes manières [59, 120]. Récemment, *Adjedj et al* [4] ont porté en Coq ce développement, et l'ont utilisé pour prouver la normalisation et la décidabilité du typage pour la théorie des types de Martin-Löf (avec un seul univers cependant). Nous plaçant dans leur roue, nous avons défini une extension de MLTT, nommée $\mathfrak{f}$TT, présentant une fonction particulière

$$\mathfrak{f} : \mathbb{N} \to \mathbb{B},$$

appelée *oracle*, dont la liste d'appels est intégrée aux règles de typage et de conversion de $\mathfrak{f}$TT. Ainsi, étant donnée une fonction

$$F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N},$$

la preuve de normalisation de $F\,\mathfrak{f} : \mathbb{N}$ nous permettra d'obtenir l'arbre des questions que F pose à $\mathfrak{f}$ avant de retourner une valeur. Cet arbre est exactement l'arbre de dialogue dont nous avons besoin pour prouver la continuité de F, ce qui permet d'affirmer la continuité de toutes les fonctions

$$F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

définissables en $\mathfrak{f}$TT. Comme cette théorie est une extension directe de MLTT, l'on en déduit aisément la continuité de toutes les fonctions

$$F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

définissables en MLTT.

[3]: Abel et al. (2017), "Decidability of Conversion for Type Theory in Type Theory"

[59]: Gilbert et al. (2019), "Definitional Proof-Irrelevance without K"

[120]: Pujet et al. (2022), "Observational Equality: Now for Good"

[4]: Adjedj et al. (2023), "Martin-Löf à la Coq"

Le chapitre 4 détaille tout d'abord les différentes règles de ꟻTT en section 4.1, puis explique plus précisément en section 4.2 comment la normalisation de cette théorie nous permettra de déduire la continuité des fonctions définissables en MLTT. Un premier aperçu de la preuve de normalisation est donné dans le cas simplifié de System T en section 4.4. Enfin, en section 4.5 nous présentons la preuve de normalisation de ꟻTT. Celle-ci n'est pas entièrement formalisée en Coq, il s'agit d'un travail en cours que nous espérons voir aboutir bientôt.

Overview

In this thesis, we highlight a particular operator, the dialogue monad, and provide two attempts at bringing together dependent type theory and continuity encoded with dialogue trees.

Chapter 2 surveys the different notions of continuity, and assesses their respective strength. In particular, we show that some logical principles such as function extensionality or bar induction make some definitions equivalent.

Chapter 3 shows how far we can go in the realm of program translations and provides a purely syntactic proof that functionals of *Baclofen Type Theory* are continuous (albeit from the target theory only, an internalization of continuity stays out of reach). Not only is the argument syntactic, but it is also expressed as a program translation into another dependent type theory. Thus, everything computes by construction and conversion in the source is interpreted as conversion in the target. Despite being a generalization of a simpler proof by Escardó, the dependently-typed presentation gives more insight about the constraints one has to respect for it to work properly, and highlights a few hidden flaws of the original version. Finally, the model gives empirical foothold to the claim that BTT is a natural setting for dependently-typed effects.

Chapter 4 presents ꟻTT, an extension of MLTT, together with a proof of normalization. This proof is pen-and-paper, but also comes with a partial formalization in Coq, up to validity predicate. We also provide a full formalization of the normalization proof for a similar extension of a smaller theory, System T, which gives us hope that formalization for full MLTT is reachable.

Contents

Part I.

Past

1. Prolegomenon and technicalities

The notion of continuity is widely present in several area of mathematics. It is well-known that different definitions of continuity can be given, and are equivalent over foundational systems such as Zermelo Fraenkel (ZF) Set theory [148, 149]. This thesis tells a tale about the status of continuity in type theory, an alternative foundation for mathematics that takes a more computational view.

[148]: Zermelo (1904), "Beweis, dass jede Menge wohlgeordnet werden kann: Aus einem an Herrn Hilbert gerichteten Briefe"
[149]: Zermelo (1908), "Untersuchungen über die Grundlagen der Mengenlehre. I"

Every good story needs a good starting point. In our case, we choose to pick 1934 when, while studying combinatory logic, *Curry* [46] noted a close resemblance between types of combinators and axioms of intuitionistic logic. Building on this seminal observation, *Howard [77]* extended it in 1969 to a more complex theory. The *Curry-Howard isomorphism*, also known as the *proposition as types, and proofs as programs correspondence*, was born. It claims that any logical statement can be seen as specification for a computer program, and any program respecting this specification can be seen as proof of said statement.

[46]: Curry (1934), "Functionality in combinatory logic"
[77]: Howard (1980), "The formulae-as-types notion of construction"

The Curry-Howard tree is one amongst many in a forest of similar observations done in history. The *Brouwer–Heyting–Kolmogorov interpretation* might be the most famous, but if we were to make a survey of the history of logic and computation, many other names would surely resurface. Yet we refrain from doing so, as we do not want to wander too far from the focal point of this thesis. Back to our story, let us simply say that the Curry-Howard isomorphism and its relatives bore many fruits. In 1970, *de Bruijn* [47] described what is maybe the first *proof assistant*, a system that is both a typed programming langugage and a theorem prover, able to certify proofs written by mathematicians. In 1971, *Martin-Löf* [104] presented the type theory that would bear his name. In 1975, *Appel and Haken* [11] proved the *four colour theorem*, an endeavour only possible through the use of computers, later refined and formalized in Coq by *Gonthier* [65, 66]. *Coq* itself, a proof assistant based on a variant of *Martin-Löf Type Theory* named *Calculus of Inductive Constructions* and still in place nowadays [135], was born in the 1980s by the hands of *Coquand and Huet* [41, 79] and extended by the work of *Luo [98, 99]* and *Coquand and Paulin-Mohring* [45, 112]. Most results of this thesis will be proved in Coq.

[47]: de Bruijn (1994), "The Mathematical Language Automath, its Usage, and Some of its Extensions"
[104]: Martin-Löf (1971), "A Theory of Types"
[11]: Appel et al. (1977), "Every planar map is four colorable. Part I: Discharging"
[65]: Gonthier (2023), *A computer-checked proof of the Four Color Theorem*
[66]: Gonthier (2007), "The Four Colour Theorem: Engineering of a Formal Proof"
[135]: (2023), "The Coq Proof Assistant (8.17)"
[41]: Coquand et al. (1988), "The Calculus of Constructions"
[79]: Huet (1989), "The Constructive Engine"
[98]: Luo (1990), "An extended calculus of constructions"
[99]: Luo (1989), "ECC, an Extended Calculus of Constructions"
[45]: Coquand et al. (1988), "Inductively defined types"
[112]: Paulin-Mohring (1993), "Inductive Definitions in the system Coq - Rules and Properties"

It seemed as if nothing could stop the wave of Curry-Howard conquests and yet, some logical principles and programming tools still evade the grasp of the overarching isomorphism and fight back foot by foot. On the logical front, the most prominent group of untamed principles is the tribe of *classical axioms*, led by two figureheads, *excluded middle* and *axiom of choice*. On the programming side, rebels are a plethora, many of them going by the nickname of *effects*: *exceptions*, *global state*, *backtracking*, *non-termination*, *non-determinism*... are just some of the names that populate this impious swarm.

The war still rages on, though, and some progress has been made over the years to bind together rebels of the two sides, be it in a simply-

typed [68] or dependently-typed setting [115]. Even though defeats have been conceded [15, 73], one can still hope to live and see the day when all of logic and computation is united under the same roof.

[68]: Griffin (1990), "A Formulae-as-Types Notion of Control"
[115]: Pédrot (2020), "Russian Constructivism in a Prefascist Theory"
[15]: Barthe et al. (2002), "CPS translating inductive and coinductive types"
[73]: Herbelin (2005), "On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic"

In this thesis, we resolutly walk in the footsteps of these great elders, and place ourselves both as firm believers of the proofs as programs correspondence and as advocates of formal verification. This effectively means that we will try to give Coq formalizations of every result we show, and that the goal of this manuscript is to study the interaction between dependent type theory and *continuity*, a mathematical concept that we believe can be considered a particular kind of effect. Hence, for every system we will present in this manuscript, two paradigms will be available (and in our view they are but the same):

- either we are studying a programming language featuring *types*, *i.e.* program specifications. Then the wording $t : A$ is spelled "t has type A", or else "the program t validates the specification A". We will call this paradigm the *computational view*;
- or we are looking at a proof system featuring *propositions*, *i.e.* logical statements. Then the wording $t : A$ is spelled "t proves A", or else "the proof t indeed proves the logical statement A". We will call this paradigm the *proof-theoretic view*.

The aim of this Chapter is to detail the different systems, technical notations and wordings we will encounter in this thesis.

- Section 1.1 presents System T, first introduced by *Gödel [70]*. It is the simplest theory displayed in this thesis, and will be the running example to detail proof techniques in later chapters.
- Section 1.2 adds a universe $\square$ and dependent products to System T, leading to *Martin-Löf Type Theory* (MLTT), our first dependent type theory. We will prove normalization for an extension of this theory in Chapter 4.
- Section 1.3 extends MLTT to general inductive types and an infinite hierarchy of universes to retrieve the *Calculus of Inductive Constructions* (CIC). We will mainly use this system as a target theory for models in Chapter 3, or more generally as a metatheory since most of our results will be formalized in Coq. However, as none of our formalizations rely on it, we do not present the impredicative sort of propositions `Prop` which is often associated to CIC. What we call *Calculus of Inductive Constructions* in this manuscript is only predicative.
- Section 1.4 introduces a more exotic theory, *Baclofen Type Theory* (BTT) [121], a variant of CIC where dependent elimination is weakened. This allows us to build effectful models of BTT, which will come in handy in Chapter 3.
- Finally, Section 1.5 develops *program translations* [25, 126] as a particular kind of *syntactic models*, a tool that will be the focal point of Chapter 3. In particular, we give the example of the *times-bool* model, a simple program translation negating funext.

[70]: Gödel (1958), "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes"
[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"
[25]: Boulier (2018), "Extending type theory with syntactic models. (Etendre la théorie des types à l'aide de modèles syntaxiques)"
[126]: Simon Boulier et al. (2017), "The next 700 syntactical models of type theory"

We emphasize that every system will be explained through a Curry-Howard interpretation, even simpler ones such as System T, a choice which might seem unusual for the experienced reader. We nonetheless believe it is a principled and informative way to explain these concepts to non-specialists of the proof-as-programs world.

1.1. System T

$$
\begin{array}{lcl}
A, B & \coloneqq & \mathbb{N} \mid A \to B \\
t, u & \coloneqq & x \mid t\ u \mid \lambda x : A.t \mid \mathrm{O} \mid \mathrm{S}\ t \mid \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n \\
\Gamma, \Delta & \coloneqq & \cdot \mid \Gamma, x : A
\end{array}
$$

$$
\frac{}{\vdash \cdot} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N}} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \to B}
$$

$$
\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}
$$

$$
\frac{\vdash \Gamma}{\Gamma \vdash \mathrm{O} : \mathbb{N}} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{S}\ n : \mathbb{N}} \qquad \frac{\begin{array}{cc}\Gamma \vdash P & \Gamma \vdash t_{\mathrm{O}} : P \\ \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P & \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : P}
$$

Figure 1.1.: Typing rules of System T

In the beginnning were the *terms*. Mindless crowd of stumbling fools, melting, mingling, shape-shifting into one another in the ever-lasting darkness of computation. Endless loops were everywhere; hunters ate preys that hunted them the day before, rivers flowed in circles and mothers gave birth to themselves, as if time itself was yet to be born. A few were standing still; regrouping in communities, they tried to self-discipline, urging each other to be *normal*. At night, they whispered dreams of a better world. At some point, was it light? was it fire? *Types* were born. Never working, never computing, going through time unchanged and unfazed, they hovered above the fray, revered by terms who followed their godly rules, establishing small islets of law in the delictuous ocean of untyped reduction. From the worldly archipelago, System T emerged.

System T is a simply-typed variant of λ-calculus introduced by Gödel [70], featuring a base type $\mathbb{N}$ and a recursor $\mathbb{N}_{\mathbf{rec}}$. Its typing rules are presented in Figure 1.1. We believe it achieves a nice balance between ease to read and expressivity, hence it will be our running example in Chapter 3 and Chapter 4. As we view it as an example, we use notations similar to more complex theories, such as MLTT or CIC, even though some concepts are not needed to describe System T, due to its sheer simplicity. We will for instance describe well-formation predicates for types even though every type in System T is by construction well-formed. That being said, let us wander through its landscape.

[70]: Gödel (1958), "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes"

In Chapter 4 we will however add booleans.

An eerie scenery The first headache for non-specialists often comes from the syntax of terms, which will be the same for System T and subsequent languages. The most uncommon notation is the use of λ-abstraction [32] to denote functions. It is similar to the $x \mapsto t$ maths notation. For instance, the function

[32]: Church (1941), "The calculi of lambda-conversion."

We use addition + and 3 in this example although they are not part of the syntax. They will however be defined later.

$$\lambda x.\ 3 + x$$

is the function

$$x \mapsto 3 + x.$$

When we want to evaluate such a function f, we can *apply* it to an argument u to get the term $f\ u$. If we applied the previous function to 4 we would get

$$(\lambda x.\ 3 + x)\ 4 \equiv 3 + 4 \equiv 7.$$

The notation $t \equiv u$ is called *conversion* and will be defined later. For now, the reader can take it as equality in the usual, mathematical sense.

To derive this result, we *substituted* the variable x by 4 in the term $3 + x$. Substitution is in fact not a trivial matter, but we will leave it at that for the moment.

Some terms, like

$$2\ y$$

(read 2 *applied to* y) have no meaning; we call them *ill-formed*. *Typing* is a discipline enforced to make sure that we only consider *well-formed* terms. The type of

A famous slogan of programming languages is *well-typed programs do not go wrong*.

$$\lambda x.\ 3 + x$$

is

$$\mathbb{N} \to \mathbb{N},$$

meaning that it takes as input a natural number and produces a natural number as output. When we apply it to the natural number 4, we retrieve another natural number, which is 7 in this example.

We write

$$t : A$$

to mean that t *is of type* A. If we consider the function

Set theorists may think of terms as *elements* and types as *sets*.

$$\lambda x.\ 3,$$

there is no clear way to deduce the type of x from the structure of the term. To disambiguate, we need to annotate the argument and write

When there is no ambiguity, we may skip such annotations in the rest of the thesis.

$$\lambda x : \mathbb{N}.\ 3.$$

Now that we know the type of its argument, and since we also know that 3 is of type $\mathbb{N}$, we can infer the type of the function and write

$$\lambda x : \mathbb{N}.\ 3 : \mathbb{N} \to \mathbb{N}.$$

At some point, we might encounter a term like

$$3 + x$$

with no λ in sight to coerce the variable x. We call such terms *open terms* and say that they contain *free variables*. On the other hand, a variable is called *bound* when coerced by a λ; subsequently, λ is called a *binder*.

It is impossible to know at first sight whether the open term

$$3 + x$$

is well-formed or ill-formed, as we do not know the type of x. What we need in such cases is a *context* Γ. Contexts are stacks of pairs of variables and types. From a proof-theoretic point of view, they can be seen as lists of hypotheses. For instance, having

$$x : \mathbb{N}$$

appear in a context Γ means that we assume that the statement $\mathbb{N}$ is true. Of course, we would hope to write more complex statements than simply $\mathbb{N}$, and we will get there when we look at more complex theories.

In the case when $x : \mathbb{N}$ is in the context, then

$$3 + x : \mathbb{N},$$

meaning that $3 + x$ is well-typed, of type $\mathbb{N}$. On the other hand, if

$$x : \mathbb{N} \to \mathbb{N}$$

appears in the context Γ then

$$3 + x$$

is ill-formed, thus non typable. To summarize, System $\mathtt{T}$, as any type theory, is built of the following bricks:

Definition 1.1.1: Type theory
A *type theory* is made of the following components:

1. *Contexts* Γ, Δ together with a unary predicate $\vdash \Gamma$ called the *well-formation predicate*;
2. *Types* A, B together with a binary relation binding contexts and types. It is called the well-formation predicate for types (under a specific context Γ) and is written $\Gamma \vdash A$;
3. *Terms* t, u, together with a ternary relation binding contexts, terms and types. It is called the typing predicate for terms (under a specific context Γ and with respect to a specific type A) and is written $\Gamma \vdash t : A$;
4. *Conversion*, a quaternary relation binding together a context, a type and two terms that will be called *equal terms*. It is written $\Gamma \vdash t \equiv u : A$.

Well-formation of contexts, of types, well-typedness and conversion of terms are mutually defined in an inductive way, through a set of *rules*. *Rules* in typing systems will all follow the same pattern:

$$\frac{\mathtt{Hyp}_1 \qquad \mathtt{Hyp}_2 \qquad \ldots \qquad \mathtt{Hyp}_n}{\mathtt{Conclusion}}$$

This idiomatic writing reads "Assuming hypotheses $\mathtt{Hyp}_1, \mathtt{Hyp}_2, \ldots, \mathtt{Hyp}_n$, we can derive `Conclusion`". All statements that are provable in System $\mathtt{T}$ can be broken down into a tree of atomic bits: System $\mathtt{T}$ rules.

Rules specific to contexts are displayed in Table 1.1. There are only two of them:

- The empty context $\cdot$ is well-formed ;
- If Γ is well-formed and A is a well-formed type under Γ then

$$\Gamma, x : A$$

  is well-formed. We implicitly assume here that a variable x does not appear twice in a context, with two different types. Tracking variables can be troublesome, and we will get to that later.

$$\frac{}{\vdash \cdot} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A}$$

Table 1.1.: Well formed contexts of System T

Rules specific to *types* are displayed in Table 1.2. Once again, there are two of them:

- If Γ is well-formed then $\Gamma \vdash \mathbb{N}$;
- If A and B are well-formed under context Γ then $\Gamma \vdash A \to B$.

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N}} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \to B}$$

Table 1.2.: Well-formation for types in System T

An astute reader might notice that, despite the notation $\Gamma \vdash A$, well-formation of types does not really depend on contexts. In fact, in System T every type is well-formed: the syntax does not allow for ill-formation. For the same reason, in System T every context is also well-formed. However, as System T is essentially a stepping stone to reach more intricate theories such as CIC or BTT, where types are terms and can thus be ill-formed, we stick with the $\vdash \Gamma$ and $\Gamma \vdash A$ predicates.

As the arrow suggests, from a proof-theoretic point of view

$$A \to B$$

can be understood as implication. On the computational side, it can be described as the type of functions from A to B.
As for $\mathbb{N}$, it is the type of natural numbers. Its existence is axiomatic, as it exists even under the empty context, *i.e.* with no hypothesis. Such a type is called a *base type*. Natural numbers are the only base type in System T but there will be others in subsequent theories.

We have seen well-formation of contexts and types; all that is left is *typing*.
The first rule is called the *variable* rule and states that if $x : A$ is in the context Γ then

$$\Gamma \vdash x : A.$$

$$\frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ \textsc{var}$$

Table 1.3.: Variable rule for System T

In a proof-theoretic view, this simply means that a hypothesis is supposed true. It is displayed in Table 1.3.

Still in the proof-theoretic view, thinking of contexts as lists of hypotheses, one intuitive fact we would want to verify is that adding hypotheses should not change the validity of a proof: one cannot *suppose so much hypotheses that a proof stops working*. This is the *weakening* rule, displayed in Table 1.4.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}$$

Table 1.4.: Weakening rule for System T

The arrow type is the subject of the next two rules, that are displayed in Table 1.5. They allow us to construct or destruct a term of type $A \to B$, reflecting the way an implication is built or destructed in first-order logic. In the computational view, this means:

- If, adding $x : A$ to Γ, one can build a term of type B then one can build a term of type $A \to B$ under context Γ. It is called the *λ-abstraction rule*, in reference to the λ syntax used for functions;

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}\ \lambda\textsc{-abs}$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B}\ \textsc{app}$$

Table 1.5.: Rules for the function type in System T

- With a term of type $A \to B$ and a term of type A, one can retrieve a term of type B. This is the *application rule*.

The counterpart in the proof-theoretic view is without much surprise: to prove $A \to B$ using a list of hypotheses Γ, one assumes A and proves B with the extended list $\Gamma, x : A$. Conversely, given a proof of $A \to B$ and a proof of A, one can derive a proof of B.

We are done with variables and function types. Next come the rules to build terms of type $\mathbb{N}$. They are displayed in Table 1.6. Once again, there are two of them:

- O is a term of type $\mathbb{N}$ under any well-formed context Γ;
- Assuming a well-typed term $n : \mathbb{N}$ under context Γ, then its successor S n is of type $\mathbb{N}$ under the same context.

Using these two rules, we can recover any natural number. The number 3 we have been using as an example, for instance, is a notation for the term S (S (S O)). This effectively means that natural numbers in System T are written in unary form.

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathrm{O} : \mathbb{N}}\ \textsc{zero}$$

$$\frac{\vdash \Gamma \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{S}\, n : \mathbb{N}}\ \textsc{succ}$$

Table 1.6.: Constructors of type $\mathbb{N}$ in T

The only term left is the destructor of type $\mathbb{N}$. It is called the *recursor*, denoted $\mathbb{N}_{\mathbf{rec}}$.
Its proof-theoretic interpretation is close to mathematical induction. Mainly: given a type P (proof-theoretically a logical statement), a term

$$t_{\mathrm{O}} : P$$

(proof-theoretically a proof of the statement in the O case), a term

$$t_{\mathrm{S}} : \mathbb{N} \to P \to P$$

(proof-theoretically a proof of the *induction step*), then we retrieve a proof of

$$P \qquad \text{for any natural number} \qquad n : \mathbb{N}.$$

Of course, this is not mathematical induction *per se*. The obvious weakness of our analogy is that P does not depend on n, which means that this induction scheme would not be able to prove complex sentences such as *for all n, there exists a number m greater than n.* All we can do for now is prove statements P for which we already have a proof $t_{\mathrm{O}} : P$, which is rather poor! This issue will however be solved in dependent type theories such as MLTT or CIC.

$$\frac{\begin{array}{c}\Gamma \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P \\ \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P \\ \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n : P}$$

Table 1.7.: Recursor for System T

Let us now tackle the computational behaviour of terms which, among other things, will help understand the meaning of $\mathbb{N}_{\mathbf{rec}}$ from a computational point of view.

Conversion is the fourth musketeer of the System T predicates. It is a quaternary relation binding together a context, a type and two terms that will be called *equal terms*. It is written

$$\Gamma \vdash t \equiv u : A,$$

reading *t and u are convertible at type A under context* Γ.
System T being simply-typed, conversion and typing work as separate

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash u \equiv v : A}{\Gamma \vdash t \equiv v : A} \qquad \frac{\Gamma \vdash u \equiv v : A \qquad \Gamma \vdash f \equiv g : A \to B}{\Gamma \vdash f\ u \equiv g\ v : B}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash f : A \to B \qquad \Gamma \vdash g : A \to B \qquad \Gamma, x : A \vdash f\ x \equiv g\ x : B}{\Gamma \vdash f \equiv g : A \to B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A.\, t)\ u \equiv t\{x := u\} : B}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t \equiv u : B}{\Gamma \vdash \lambda x : A.\, t \equiv \lambda x : A.\, u : A \to B} \qquad \frac{\Gamma \vdash n \equiv m : \mathbb{N}}{\Gamma \vdash \mathrm{S}\ n \equiv \mathrm{S}\ m : \mathbb{N}} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} \equiv t'_{\mathrm{O}} : P \qquad \Gamma \vdash t_{\mathrm{S}} \equiv t'_{\mathrm{S}} : \mathbb{N} \to P \to P \qquad \Gamma \vdash n \equiv n' : \mathbb{N}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n \equiv \mathbb{N}_{\mathbf{rec}}\ P\ t'_{\mathrm{O}}\ t'_{\mathrm{S}}\ n' : P}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P \qquad \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ \mathrm{O} \equiv t_{\mathrm{O}} : P} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P \qquad \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ (\mathrm{S}\ n) \equiv t_{\mathrm{S}}\ n\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n) : P}$$

Figure 1.2.: Conversion rules of System T

fields of the theory, unlike dependent type theory where everything mingles. This is why we presented typing in a first Figure, and only now care about computation, which would not make sense in MLTT or CIC.

Some systems choose to have an *untyped* definition of conversion, written

$$t \equiv u.$$

The question of what properties can be derived for typed or untyped conversion, and whether the two systems are equivalent, is a research topic in itself, and we will not delve in it. The interested reader can look at *Lennon-Bertrand*'s PhD [96] for a more in-depth study of this question. In the remainder of this thesis we will assume both systems to be equivalent, and stick to the typed version for the most part. The only exception is when dealing with *program translations* in Section 1.5 and Chapter 3.

[96]: Lennon-Bertrand (2022), "Bidirectional Typing for the Calculus of Inductive Constructions"

Greek letters everywhere A first rule of conversion is that it must relate different writings of the same term. For instance, given two different spellings of the same function,

$$\lambda x : A.\, 3 + x \qquad \text{and} \qquad \lambda y : A.\, 3 + y,$$

we want conversion to equate them. This conversion up to renaming is called *α-conversion*. Note however that the open terms

$$3 + x \qquad \text{and} \qquad 3 + y$$

are *not* equal, as x and y could be substituted by different terms later on.

This is already the second time we mention *substitution*, a feature crucial in every theory based on λ-calculus and its guiding light, the in-

famous *β-conversion*, which is displayed in Table 1.8.

This conversion rule states that when a function

$$\lambda x : A.\, t$$

is applied to an argument u, then it is convertible to the body t of the function, where *every occurence of x has been replaced by u.*
Both substitution and α-conversion are easy to understand on an intuitive level but more difficult to implement in a proof assistant. The way Coq deals with them is through *de Bruijn* indices [47], but many other choices are possible [13]. In this thesis, we will use *de Bruijn* indices for formalization purposes, and keep named variables in the manuscript, as they are more readable.

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A.\, t)\, u \equiv t\{x := u\} : B}\ \beta\text{-Red}$$

Table 1.8.: Typed β-conversion in System T.

[47]: de Bruijn (1994), "The Mathematical Language Automath, its Usage, and Some of its Extensions"
[13]: Aydemir et al. (2005), "Mechanized Metatheory for the Masses: The PoplMark Challenge"

Having dealt with α and β-conversion, the last subtlety regarding variables has to do with functions: given

$$f : A \to B,$$

we would like to equate

$$f \qquad \text{and} \qquad \lambda x : A.\, f\, x,$$

as they compute the same function, even though they are syntactically different. This is the whole point of *η-conversion*, displayed in Table 1.9: when comparing two functions

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash f : A \to B \qquad \Gamma \vdash g : A \to B \qquad \Gamma, x : A \vdash f\, x \equiv g\, x : B}{\Gamma \vdash f \equiv g : A \to B}\ \eta\text{-conv}$$

Table 1.9.: Typed η-conversion in System T

$$\Gamma \vdash f, g : A \to B$$

the algorithmic idea is to apply them to a *fresh* variable $x : A$ (where *fresh* means a variable not already present in the terms) and recursively compare the bodies of the two functions at type B. In our case, this would mean comparing

$$f\, x \qquad \text{and} \qquad (\lambda x : A.\, f\, x)\, x$$

at type B. Thanks to β-conversion,

$$\Gamma \vdash (\lambda x : A.\, f\, x)\, x \equiv f\, x : B$$

thus

$$f \qquad \text{and} \qquad \lambda x : A.\, f\, x$$

are convertible.

The notation

$$(\lambda x : A.\, f\, x)\, x$$

may be confusing. Remember that the x in

$$\lambda x : A.\, f\, x$$

is *bound* by λ, while the x outside is *free.* Thus, they have nothing to do with each other. Thanks to α-conversion, we can disambiguate and write

$$(\lambda y : A.\, f\, y)\, x.$$

Once again, de Bruijn indices lets us escape that kind of headache.

Conversion is often called *judgmental equality* or *definitional equality*, a name befitting its proof-theoretic interpretation: proofs *by definition*, where the only argument is pure computation. As the name *equality* suggests, we want it to be an equivalence relation. Hence, it should be reflexive, symmetric and transitive. Table 1.10 recalls these rules.

Congruence rules are displayed in Table 1.11. They make conversion compatible with constructors of System T:

- *Application congruence*: if two functions

  $$\Gamma \vdash f, g : A \to B$$

  are convertible and if their arguments $\Gamma \vdash u, v : A$ are convertible then

  $$f\ u \qquad \text{and} \qquad g\ v$$

  are convertible under context Γ;
- *λ-congruence* : if two terms

  $$\Gamma, x : A \vdash t, u : B$$

  are convertible under the extended context $\Gamma, x : A$ then their λ-abstractions

  $$\Gamma \vdash \lambda x : A.\, t, \lambda x : A.\, u$$

  are convertible at type $A \to B$ under context Γ;
- *Successor congruence*: if two natural numbers $n, m : \mathbb{N}$ are convertible then their successors are convertible;
- *Recursor congruence*: given a type P, if

  $$t_{\mathsf{O}}, t'_{\mathsf{O}} : P$$

  are convertible, if

  $$t_{\mathsf{S}}, t'_{\mathsf{S}} : \mathbb{N} \to P \to P$$

  are convertible and if $n, n' : \mathbb{N}$ are convertible then

  $$\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n \qquad \text{is convertible to} \qquad \mathbb{N}_{\mathbf{rec}}\ P\ t'_{\mathsf{O}}\ t'_{\mathsf{S}}\ n'.$$

Successor and recursor congruence can be seen as consequences of application congruence through curryfication. Indeed,

$$\lambda x : \mathbb{N}.\ \mathsf{S}\ x \equiv \lambda x : \mathbb{N}.\ \mathsf{S}\ x : \mathbb{N}$$

by reflexivity and

$$n \equiv n' : \mathbb{N}$$

by hypothesis, thus

$$\mathsf{S}\ n \equiv\ \mathsf{S}\ n' : \mathbb{N}$$

by congruence of application. The same proof applies to $\mathbb{N}_{\mathbf{rec}}$. Nonetheless, we find that explicitly adding congruence rules for successor and recursor emphasizes that they are constructors of the language and not functions. This point of view alleviates some troubles when proving normalization of type theories, as we will do in Chapter 4, hence we wanted to highlight this distinction here. The two definitions are however equivalent.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A}\ \textsc{refl} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A}\ \textsc{sym}$$

$$\frac{\Gamma \vdash t \equiv u : A \quad \Gamma \vdash u \equiv v : A}{\Gamma \vdash t \equiv v : A}\ \textsc{trans}$$

Table 1.10.: Equivalence rules of conversion for System T

$$\frac{\begin{array}{l}\Gamma \vdash u \equiv v : A \\ \Gamma \vdash f \equiv g : A \to B\end{array}}{\Gamma \vdash f\ u \equiv g\ v : B}\ \textsc{app-cg}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t \equiv u : B}{\Gamma \vdash \lambda x : A.\, t \equiv \lambda x : A.\, u : A \to B}\ \textsc{lam-cg}$$

$$\frac{\Gamma \vdash n \equiv m : \mathbb{N}}{\Gamma \vdash \mathsf{S}\ n \equiv \mathsf{S}\ m : \mathbb{N}}\ \textsc{succ-cg}$$

$$\frac{\begin{array}{l}\Gamma \vdash P \\ \Gamma \vdash t_{\mathsf{O}} \equiv t'_{\mathsf{O}} : P \\ \Gamma \vdash t_{\mathsf{S}} \equiv t'_{\mathsf{S}} : \mathbb{N} \to P \to P \\ \Gamma \vdash n \equiv n' : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n \equiv \mathbb{N}_{\mathbf{rec}}\ P\ t'_{\mathsf{O}}\ t'_{\mathsf{S}}\ n' : P}\ \textsc{rec-cg}$$

Table 1.11.: Congruence rules of System T.

$$\frac{\begin{array}{l}\Gamma \vdash P \qquad \Gamma \vdash t_{\mathsf{O}} : P \\ \Gamma \vdash t_{\mathsf{S}} : \mathbb{N} \to P \to P\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ \mathsf{O} \equiv t_{\mathsf{O}} : P}$$

$$\frac{\begin{array}{l}\Gamma \vdash P \\ \Gamma \vdash t_{\mathsf{S}} : \mathbb{N} \to P \to P \\ \Gamma \vdash t_{\mathsf{O}} : P \\ \Gamma \vdash n : \mathbb{N}\end{array}}{\begin{array}{l}\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ (\mathsf{S}\ n) \equiv \\ \qquad t_{\mathsf{S}}\ n\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n) : P\end{array}}$$

Table 1.12.: Recursor rules of System T

The last conversion rules are specific to $\mathbb{N}_{\mathbf{rec}}$ and are displayed in Table 1.12. There are two of them:

- When applied to O, $\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}$ returns its base argument t_{O};
- When applied to S n, $\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}$ applies its step argument t_{S} and calls itself recursively on n.

Through Curry-Howard glasses, this looks exactly like the computational behaviour we would expect from mathematical induction. Moreover, these rules also allow us to define functions such as addition:

$$n + m \coloneqq \mathbb{N}_{\mathbf{rec}}\ \mathbb{N}\ m\ (\lambda(i : \mathbb{N})\,(H_i : \mathbb{N}).\ \mathrm{S}\ H_i)\ n$$

From an imperative programming language perspective, one might be tempted to compare it to a `for` loop, iterating n times the t_{S} body of the loop before returning t_{O}. Another intuition would be to describe $\mathbb{N}_{\mathbf{rec}}$ as a recursive function: a computer scientist accustomed to functional programming may indeed notice that $\mathbb{N}_{\mathbf{rec}}$ behaves like *structural pattern-matching* on n, calling itself recursively in the successor case. Pattern-matching in the style of functional programming will indeed be our go-to dialect to define functions in this manuscript. In this syntax, addition looks like this:

$$\begin{array}{lcl} \mathsf{add} & : & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \mathsf{add}\ \mathrm{O}\ m & \coloneqq & m \\ \mathsf{add}\ (\mathrm{S}\ k)\ m & \coloneqq & \mathrm{S}\ (\mathsf{add}\ k\ m) \end{array}$$

In Coq, pattern-matching comes first and recursors are derived from it. However, it was noted that in some cases it was dubious whether pattern-matching and recursors were indeed equivalent [60, 61]. More recently, *Cockx* [33] provided an algorithm able to translate pattern-matching programs to eliminators in a systematic way, for a significant fragment of the type theory implemented by the Agda proof assistant. In this thesis, we will make the assumption that every function we define via pattern-matching could be defined with recursors.

[60]: Giménez (1995), "Codifying guarded definitions with recursive schemes"
[61]: Giménez (1998), "Structural recursive definitions in type theory"
[33]: Cockx (2017), "Dependent Pattern Matching and Proof-Relevant Unification"

1.2. MLTT

System T is a very stratified society: at the bottom live the terms, slaves by divine right, forever working under the constant watch of higher beings. They are the *Men*. At the top stand the types, unchallenged tyrants reigning over terms, enforcing laws to make sure they *behave well*; eternal rulers forever evading the grasp of mere mortals. They are, of course, *Gods*.

Dependent type theory is the result of a *spiritual revolution*, overthrowing types from their divine throne and plunging them into the mass of terms. Their golden wings stuck in the mire, they too start dissolving in the blissful mud of computation. However, not all types are born equal and one escapes that deadly fate, as MLTT turns out not to be an atheist world, but chooses to indulge itself in *monotheism*. At the top, $\square$, the *type of all types*, stands strong and all shall worship its absolute reign.

$$A, B, t, u := \square \mid x \mid t\ u \mid \lambda x : A.t \mid \Pi x : A.t \mid \mathbb{N} \mid \mathrm{O} \mid \mathrm{S}\ t \mid \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n$$

$$\Gamma, \Delta := \cdot \mid \Gamma, x : A$$

$$\vdash \cdot \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \square} \qquad \frac{\Gamma \vdash A : \square}{\Gamma \vdash A}$$

$$\frac{\Gamma \vdash A : \square \qquad \Gamma, x : A \vdash B : \square}{\Gamma \vdash \Pi x : A.B : \square} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A.B} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \square}$$

$$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B\{x := u\}} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathrm{O} : \mathbb{N}} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{S}\ n : \mathbb{N}} \qquad \frac{\begin{array}{c}\Gamma, x : \mathbb{N} \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P\{x := \mathrm{O}\} \\ \Gamma \vdash t_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x := y\} \to P\{x := \mathrm{S}\ y\} \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : P\{x := n\}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B}$$

Figure 1.3.: Syntax and typing rules for MLTT

We have now left the paddling pool behind to bathe in *dependent type theory*. Our first system of that kind will be *Martin-Löf Type Theory* (MLTT) [103, 104]. Slight changes in definitions are numerous in type theory, and almost every researcher in the field has a different idea of what MLTT means, and how it differs from the *Calculus of Inductive Constructions* (CIC), our next Section.

[103]: Martin-Löf (1984), *Intuitionistic type theory*
[104]: Martin-Löf (1971), "A Theory of Types"

For impatient expert readers, let us say that our own version of MLTT is quite minimal: Π-types, a base type $\mathbb{N}$ and one universe; conversion given as the reflexive, symmetric, transitive and congruent closure of $\eta\beta\iota$-typed-conversion. In particular we do not ask for an identity type. Typing rules of MLTT are displayed in Figure 1.3.

In Chapter 4, it will also feature booleans, but for brevity we do not present them here.

On equal terms Most of the changes with respect to System T can be explained by treating types as terms, leading to the birth of $\square$. Let us unfold its consequences through an example: adding $\square$ as the *type of types* to System T means that $\mathbb{N}$ is now a term, typable as follows:

$$\mathbb{N} : \square.$$

Therefore, the following function now becomes definable:

When the body t of the function

$$\lambda x : A.t$$

does not make use of x, we write

$$\lambda\_ : A.t$$

to avoid giving names to unused variables.

$$F \coloneqq \lambda n : \mathbb{N}.\, \mathbb{N}_{\mathbf{rec}}\, \square\, \mathbb{N}\, (\lambda(\_ : \mathbb{N})\, (\_ : \square).\, \mathbb{N} \to \mathbb{N})\, n.$$

This function takes as input a term $n : \mathbb{N}$ and returns a type: it outputs $\mathbb{N}$ when applied to O, and $\mathbb{N} \to \mathbb{N}$ otherwise. This raises several questions:

- when we apply F to O, the result is

  $$\mathbb{N}_{\mathbf{rec}}\, \square\, \mathbb{N}\, (\lambda(\_ : \mathbb{N})\, (\_ : \square).\, \mathbb{N} \to \mathbb{N})\, \mathrm{O},$$

  which is not strictly speaking $\mathbb{N}$, but is *convertible* to $\mathbb{N}$. Indeed, now that types are terms, they also compute and we have to deal with *conversion at the level of types*. Now, considering for instance $\mathrm{O} : \mathbb{N}$, do we want to enforce that

  $$\mathrm{O} : \mathbb{N}_{\mathbf{rec}}\, \square\, \mathbb{N}\, (\lambda(\_ : \mathbb{N})\, (\_ : \square).\, \mathbb{N} \to \mathbb{N})\, \mathrm{O},$$

  i.e. do we want typing to be stable by conversion on types? In MLTT the answer is positive, which leads to the conversion rule presented in Tab 1.13.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B}$$

Table 1.13.: Conversion rule for typing in MLTT

- For any given $n : \mathbb{N}$, it is possible to provide a term of type $F\, n$. Indeed,

when	$n = \mathrm{O}$	we can return	O	and
when	$n = \mathrm{S}\, k$	we can return	$\lambda x : \mathbb{N}.\, x$.	

  Formally, this means that we would want to write the following:

  $$n : \mathbb{N} \vdash \mathbb{N}_{\mathbf{rec}}\, (F\, n)\, \mathrm{O}\, (\lambda(x : \mathbb{N})\, (p_x : F\, x).\, x)\, n : F\, n.$$

  However, even though this term is well-typed for any concrete instance of n, in System T there is no way to prove that it is well-typed *in general*. Indeed, were we to try and apply the rule for $\mathbb{N}_{\mathbf{rec}}$, we would end up having to prove

  $$n : \mathbb{N} \vdash \mathrm{O} : F\, n$$

  for the t_{O} case, which is not true. The only thing we can prove is

  $$\vdash \mathrm{O} : F\, \mathrm{O},$$

  which is not the same. We thus need to accomodate for computation in the types. We can solve this problem with an updated rule for $\mathbb{N}_{\mathbf{rec}}$. To distinguish the old and the updated version, we call the latter one $\mathbb{N}_{\mathbf{ind}}$, a name refering to *induction*. The updated version is presented in Table 1.14; we also recall the former one to highlight the difference.

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P \qquad \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n : P}$$

$$\frac{\Gamma, x : \mathbb{N} \vdash P \qquad \Gamma \vdash t_{\mathrm{O}} : P\{x \coloneqq \mathrm{O}\} \qquad \Gamma, y : \mathbb{N} \vdash t_{\mathrm{S}} : P\{x \coloneqq y\} \to P\{x \coloneqq \mathrm{S}\, y\} \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n : P\{x \coloneqq n\}}$$

Table 1.14.: First version of the recursor rule in MLTT

- Unfortunately, our troubles with $\mathbb{N}_{\mathbf{rec}}$ do not end there. Since we changed the recursor rule, we can type

  $$\begin{array}{lll} \Gamma & \vdash & t_{\mathrm{O}} : F\, \mathrm{O} \coloneqq \mathrm{O} \\ \Gamma, y : \mathbb{N} & \vdash & t_{\mathrm{S}} : P\{x \coloneqq y\} \to P\{x \coloneqq \mathrm{S}\, y\} \coloneqq \\ & & \quad \lambda H_y : P\{x \coloneqq y\}.\, \lambda n : \mathbb{N}.\, n \end{array}$$

  which means we are now able to type

  $$n : \mathbb{N} \vdash \mathbb{N}_{\mathbf{rec}}\, (F\, n)\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n$$

but now the corresponding function

$$\lambda n : \mathbb{N}.\, \mathbb{N}_{\mathbf{rec}}\ (F\ n)\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n$$

is *not typable*.
This is cumbersome: we can write complex terms by adding variables into the context, but we cannot turn those terms into functions because our λ-abstraction rule is *too weak*. Indeed, $F\ n$ is a type that *depends* on the value of n but in the functional types

$$A \to B$$

of System T the return type B is not allowed to *depend* on the value of the argument. This problem is solved by relaxing the λ-abstraction rule, leading to the formation of *dependent products*, displayed in Table 1.15. Using such dependent products, we can rephrase the recursor rule for MLTT, as displayed in Table 1.16. This latter version is the one we choose for the remainder of the thesis.

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}$$

Table 1.15.: λ-abstraction rule in MLTT

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{N} \vdash P & \\ \Gamma \qquad \vdash t_{\mathrm{O}} : P\{x \coloneqq \mathrm{O}\} & \\ \Gamma \qquad \vdash t_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x \coloneqq y\} \to P\{x \coloneqq \mathrm{S}\ y\} & \\ \Gamma \qquad \vdash n : \mathbb{N} & \end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : P\{x \coloneqq n\}}$$

Table 1.16.: Recursor rule in MLTT

Incidentally, in the *Agda* proof assistant, as in Coq, the $\forall$ symbol is often used as notation for dependent products.

Easy as Π The name *dependent product* and the letter Π are both quite exotic, which may obfuscate the small miracle we just witnessed. It should however become obvious when facing this alternative writing for dependent product:

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \forall x : A.B}$$

Indeed, when relaxing the functional type $A \to B$ by allowing the codomain B to depend on A, we just gave computational content to the logical symbol $\forall$.

Another way to see this is the following: in usual pen-and-paper mathematics, when facing a logical sentence

$$\forall (n \in \mathbb{N}),\ P\ n$$

with P a logical expression, one would typically write something along those lines: *let us take an arbitrary natural number n. Then we have to prove $P\ n$.* This is what this rule describes: to inhabit

$$\forall n : \mathbb{N}.\, P\ n,$$

one adds a variable

$$n : \mathbb{N}$$

into the context, then tries to inhabit $P\ n$. We can go further: our imaginary mathematician works exactly as a function, taking an input (here a natural number n) and producing an output (here a proof of $P\ n$). Such is the power of the Curry-Howard motto: *propositions as types, and proofs as programs.*

One might wonder whether ∀ is the only logical quantifier that can be given computational interpretation. This is not the case, and an analogous work is possible for the existential quantifier ∃. This is however not part of our version of MLTT, and we encourage the reader to wait until we reach CIC to enjoy dependent sums. The impatient reader who could not grin and bear it can jump forward to Section 1.3.

As we already saw in System T, substitution lies at the core of computation. Since types now compute, it is only natural that substitution should play its part. This leads us to an updated application rule, displayed in Table 1.17.

$$\frac{\Gamma \vdash t : \Pi x : A.\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B\{x := u\}}$$

Table 1.17.: Application rule in MLTT

We are almost done with typing rules. All that's left is to clarify typing for types. As we already explained, MLTT features a new type, called the *universe*. It is sometimes written Type as in *Coquand* [40], sometimes U or $\mathcal{U}$ as in *Abel et al* [2]; *Martin-Löf* himself named it V in his seminal paper [103, 104]. In this thesis, we will follow the Pure Type Systems tradition and take *Barendregt*'s notation [14], meaning we write $\square$ to denote the universe.

$$\frac{\vdash \Gamma}{\Gamma \vdash \square}$$

Table 1.18.: The universe in MLTT

Formation rule for $\square$ is given in Table 1.18. It boils down to saying that the universe is well-formed when the ambiant context Γ is well-formed.

The interesting rule regarding the universe affects its elements: any well-typed term

$$\Gamma \vdash A : \square$$

can be turned into a well-formed type

$$\Gamma \vdash A.$$

It is displayed in Table 1.19. Note that this way of phrasing things is not the only one; it is called having universes *à la Russel*. Another way, called *à la Tarski*, involves an explicit function El turning terms of type $\square$ into proper types. It looks like this:

$$\frac{\Gamma \vdash A : \square}{\Gamma \vdash \mathtt{El}\ A}$$

Table 1.19.: Elements of the universe in MLTT

Luo [100] pointed out that some care should be taken when presenting the two versions as they were not always equivalent. *Assaf* [12] provided a formulation where the two styles are equivalent. In Chapter 3, when we interpret elements of $\square$ as algebras of a monad and types as the underlying set of said algebra, a function reminiscent of El will make its appearance. We will mostly use universes *à la Russel* in the rest of the thesis though, as they ease the reading.

[40]: Coquand (1986), "An Analysis of Girard's Paradox"

[2]: Abel et al. (2007), "Normalization by Evaluation for Martin-Löf Type Theory with One Universe"

[103]: Martin-Löf (1984), *Intuitionistic type theory*

[104]: Martin-Löf (1971), "A Theory of Types"

[14]: Barendregt (1991), "Introduction to Generalized Type Systems"

[100]: Luo (2012), "Notes on universes in type theory"

[12]: Assaf (2014), "A Calculus of Constructions with Explicit Subtyping"

And God created a rock He could not lift In the first version of his system, *Martin-Löf* [103, 104] stated that $\square$ should have the type $\square$. This rule

$$\square : \square$$

is evocative of the *set of all sets* that led to Russel's paradox [54]. *Martin-Löf*'s rule, too, is inconsistent and led to a similar paradox found by *Girard* [62], as a variant of the *Burali-Forti* paradox [29]. This paradox was later analyzed by *Coquand* [40] who found another paradox in a slightly weaker setting [39]. Finally, *Hurkens* [81] provided a simpler version of the latter.

There are ways to circumvent this issue, and we will present one of them later on. For now, let us simply say that $\square$ stands alone, the only type with no type.

The same fear of paradoxes leads us to having two rules for the well-formation of dependent products, both of them displayed in Table 1.20. Indeed, a dependent product such as

$$\Pi(x : \square).\ \square$$

cannot be allowed to be of type $\square$, or we could once again build a paradox. This means that we effectively live in a two-levels type theory. As long as we only mention "small types", *i.e.* types of type $\square$, our dependent product will be of type $\square$. But as soon as we get it on with bigger types such as the big boss $\square$ itself, we enter the big league and there is no turning back: our dependent product will never be of type $\square$ again.

All the nitty-gritty details of MLTT typing rules have now come under scrutiny, and we can set our sight on conversion. Compared to System T, things do not change much: we still have β and η-conversion, conversion is still an equivalence relation and rules for $\mathbb{N}_{\mathbf{ind}}$ are the same as for $\mathbb{N}_{\mathbf{rec}}$ in System T. We mainly replace functional types by dependent products, and duplicate some rules at the level of types. All things considered, there are two salient additions:

- As conversion is typed, we need to add a rule to account for convertibility of types: if t and u are convertible at type A and if A is convertible to B then t and u are convertible at type B. This particular rule is presented in Table 1.21.
- We add a rule describing convertibility of dependent products, similar to λ-congruence: if A and A' are convertible and if, given $x : A$, B and B' are convertible, then

  $$\Pi x : A.\ B \qquad \text{and} \qquad \Pi x : A'.\ B' \qquad \text{are convertible.}$$

  This rule is displayed in Table 1.22.

For the sake of completeness, we present every rule of conversion for MLTT in Figure 1.4.

[103]: Martin-Löf (1984), *Intuitionistic type theory*
[104]: Martin-Löf (1971), "A Theory of Types"

[54]: Frege (1893), *Grundgesetze der Arithmetik: begriffsschriftlich abgeleitet*

[62]: Girard (1972), "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur"

[29]: Burali-Forti (1897), "Una questione sui numeri transfiniti"
[40]: Coquand (1986), "An Analysis of Girard's Paradox"
[39]: Coquand (1995), "A new paradox in type theory"
[81]: Hurkens (1995), "A Simplification of Girard's Paradox"

$$\frac{\Gamma \vdash A : \square \qquad \Gamma, x : A \vdash B : \square}{\Gamma \vdash \Pi x : A.\ B : \square}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A.\ B}$$

Table 1.20.: Dependent products in MLTT

$$\frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t \equiv u : B}$$

Table 1.21.: Conversion rule for conversion in MLTT

$$\frac{\Gamma \vdash A \equiv A' : \square \qquad \Gamma, x : A \vdash B \equiv B' : \square}{\Gamma \vdash \Pi x : A.\ B \equiv \Pi x : A'.\ B' : \square}$$

$$\frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B'}{\Gamma \vdash \Pi x : A.\ B \equiv \Pi x : A'.\ B'}$$

Table 1.22.: Convertibility of Π-types in MLTT

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \equiv A} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma \vdash A \equiv B}{\Gamma \vdash B \equiv A} \qquad \frac{\Gamma \vdash A \equiv B : \square}{\Gamma \vdash A \equiv B}$$

$$\frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t \equiv u : B} \qquad \frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash u \equiv v : A}{\Gamma \vdash t \equiv v : A} \qquad \frac{\Gamma \vdash A \equiv B \qquad \Gamma \vdash B \equiv C}{\Gamma \vdash A \equiv C}$$

$$\frac{\Gamma \vdash A \equiv A' : \square \qquad \Gamma, x : A \vdash B \equiv B' : \square}{\Gamma \vdash \Pi x : A.\, B \equiv \Pi x : A'.\, B' : \square} \qquad \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B'}{\Gamma \vdash \Pi x : A.\, B \equiv \Pi x : A'.\, B'}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A.\, t)\, u \equiv t\{x := u\} : B\{x := u\}} \qquad \frac{\Gamma \vdash f \equiv g : \Pi x : A.\, B \qquad \Gamma \vdash u \equiv v : A}{\Gamma \vdash f\, u \equiv g\, v : B\{x := u\}}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash f : \Pi x : A.\, B \qquad \Gamma \vdash g : \Pi x : A.\, B \qquad \Gamma, x : A \vdash f\, x \equiv g\, x : B}{\Gamma \vdash f \equiv g : \Pi x : A.\, B}$$

$$\frac{\Gamma \vdash n \equiv m : \mathbb{N}}{\Gamma \vdash \mathrm{S}\, n \equiv \mathrm{S}\, m : \mathbb{N}} \qquad \frac{\begin{array}{ll} \Gamma, x : \mathbb{N} \vdash P & \\ \Gamma, \quad \vdash t_{\mathrm{S}} \equiv t'_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x := y\} \to P\{x := \mathrm{S}\, y\} & \Gamma \vdash n \equiv n' : \mathbb{N} \end{array} \qquad \Gamma \vdash t_{\mathrm{O}} \equiv t'_{\mathrm{O}} : P\{x := \mathrm{O}\}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n \equiv \mathbb{N}_{\mathbf{ind}}\, P\, t'_{\mathrm{O}}\, t'_{\mathrm{S}}\, n' : P\{x := n\}}$$

$$\frac{\begin{array}{ll} \Gamma, x : \mathbb{N} \vdash P & \\ \Gamma \quad \vdash t_{\mathrm{O}} : P\{x := \mathrm{O}\} & \\ \Gamma \quad \vdash t_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x := y\} \to P\{x := \mathrm{S}\, y\} & \end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, \mathrm{O} \equiv t_{\mathrm{O}} : P\{x := \mathrm{O}\}}$$

$$\frac{\begin{array}{ll} \Gamma, x : \mathbb{N} \vdash P & \Gamma \vdash t_{\mathrm{O}} : P\{x := \mathrm{O}\} \\ \Gamma \quad \vdash t_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x := y\} \to P\{x := \mathrm{S}\, y\} & \Gamma \vdash n : \mathbb{N} \end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, (\mathrm{S}\, n) \equiv t_{\mathrm{S}}\, n\, (\mathbb{N}_{\mathbf{ind}}\, P\, t_{\mathrm{O}}\, t_{\mathrm{S}}\, n) : P\{x := \mathrm{S}\, n\}}$$

Figure 1.4.: Conversion rules of MLTT

1.3. CIC

$$A, B, t, u ::= \square_i \mid x \mid t\ u \mid \lambda x : A.t \mid \Pi x : A.t$$
$$\Gamma, \Delta ::= \cdot \mid \Gamma, x : A$$

$$\frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash A : \square_i}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \square_i : \square_{i+1}}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma \vdash t : B}{\Gamma, x : A \vdash t : B} \qquad \frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Pi x : A.B : \square_{\max(i,j)}}$$

$$\frac{\Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B\{x := u\}} \qquad \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.B : \square_i}{\Gamma \vdash \lambda x : A.t : \Pi x : A.B}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \square_i \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Sigma x : A.B : \square_{\max(i,j)}} \qquad \frac{\Gamma \vdash t : \Sigma x : A.B}{\Gamma \vdash t.\pi_1 : A} \qquad \frac{\Gamma \vdash t : \Sigma x : A.B}{\Gamma \vdash t.\pi_2 : B\{x := t.\pi_1\}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B\{x := t\} \qquad \Gamma \vdash \Sigma x : A.B : \square_i}{\Gamma \vdash (t, u) : \Sigma x : A.B}$$

(conversion omitted)

Figure 1.5.: Syntax of CC_ω extended with Σ-types

As we pointed out in the previous chapter, MLTT stems from a religious overhaul, turning a pagan System T into a monotheist society. Our next system, the *Calculus of Inductive Constructions* (CIC), concludes the spiritual revolution by stripping the last god □ of its divine ornaments and building a complete impious world. Alas, a society without an absolute ruler is by no means egalitarian, and a new hierarchy soon surfaces, more stratified than ever, with countless layers of ambitious, relentless, smaller dictators, always bickering and confronting their respective powers.

The ever-growing Babel tower It is customary to separate CIC in two parts: the *Calculus of Constructions* (CC_ω) on the one hand, and the *Inductives* on the other hand. The CC_ω part is displayed in Figure 1.5, and presents some changes with respect to MLTT. Let us wander through them one by one.

The fact that $\square$ is the only term in MLTT to not have a type feels a bit *ad-hoc* and, even though the rule

$$\square : \square$$

is inconsistent, we have not lost hope to solve this problem. A first observation is that if we take two versions of $\square$, for instance

$$\square_0 \qquad \text{and} \qquad \square_1, \qquad \text{such that} \qquad \square_0 : \square_1,$$

then the resulting system is consistent. Of course, if we try and write

$$\square_1 : \square_1$$

then we face Girard's paradox once again, so $\square_1$ cannot be typed. We can try and add $\square_2$ to the family to solve this issue but of course the problem now lies a few goalposts away. We soon end up with

$$\square_0 : \square_1 : \ldots : \square_n$$

but as soon as we stop, trouble resurfaces. Well, the answer is easy: we simply *never stop*, and build an infinite, ever-growing tower of $\square_i$, relevantly named the *universe hierarchy*. Now universes $\square_i$ are indexed by variables i, j and for any i, we have:

$$\square_i : \square_{i+1}$$

This rule is displayed in Table 1.23. The natural idea is to take i, j to be natural numbers but this is not necessary, and other choices might make our system more flexible. One was introduced by *Harper et al [71]* under the name *typical ambiguity*. We will silently rely on it for the rest of this thesis.

$$\frac{\vdash \Gamma}{\Gamma \vdash \square_i : \square_{i+1}}$$

Table 1.23.: Hierarchy of universes in CC_ω

[71]: Harper et al. (1991), "Type Checking with Universes"

There are many tweaks one can make to this infinite tower of universes. One is called *cumulativity*, presented in Table 1.24. It says that if A is a term of type $\square_i$ for some i then it is also a term of type $\square_j$ for any greater j. Cumulativity is an important feature of Coq [136], a proof assistant base on CIC. Every formalized result of this thesis will be written in Coq, which means that our meta-theory will effectively be Coq's version of CIC. Therefore, it will feature cumulativity.

$$\frac{\Gamma \vdash A : \square_i \qquad i < j}{\Gamma \vdash A : \square_j}$$

Table 1.24.: Cumulativity in CC_ω

[136]: Timany et al. (2018), "Cumulative Inductive Types In Coq"

We will however not use `Prop`, the impredicative universe of propositions present in Coq and often associated with CIC.

In MLTT, we duplicated the well-formation rules for dependent products, to avoid Girard's paradox. When facing an infinite hierarchy, this technique scales to the rule displayed in Table 1.25. It formalizes the intuition that a dependent product

$$\Pi x : A.\, B$$

has to live at the same level or higher than both A and B.

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Pi x : A.\, B : \square_{\max(i,j)}}$$

Table 1.25.: Dependent products in CC_ω

No existential crisis The last enhancement of CC_ω is the addition of *dependent sums* (also called Σ-types), which are the computational counterpart of the existential quantifier $\exists$, in the same way dependent products are the computational counterpart of the universal quantifier $\forall$. Rules for Σ-types are displayed in Table 1.26. To summarize:

- $\Sigma x : A.B$ is well-formed when A is well-formed and B is a well-formed type, dependent on A;
- To build an element of
$$\Sigma x : A.B,$$
one has to provide an element
$$t : A \qquad \text{and a proof of} \qquad B\{x \coloneqq t\},$$
mimicking what a mathematician would do to prove the existence of some mathematical object validating some property.
- Given an element
$$t : \Sigma x : A\,B,$$
one can recover an element of A thanks to the first projection
$$t.\pi_1 : A.$$
The second projection
$$t.\pi_2 : B\{x \coloneqq t.\pi_1\}$$
is a proof of B instantiated with $t.\pi_1$.

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Sigma x : A.B : \square_{\max(i,j)}}$$

$$\frac{\Gamma \vdash t : \Sigma x : A.B}{\Gamma \vdash t.\pi_1 : A}$$

$$\frac{\Gamma \vdash t : \Sigma x : A.B}{\Gamma \vdash t.\pi_2 : B\{x \coloneqq t.\pi_1\}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B\{x \coloneqq t\} \qquad \Gamma \vdash \Sigma x : A.B : \square_i}{\Gamma \vdash (t,u) : \Sigma x : A.B}$$

Table 1.26.: Dependent sums in CC_ω

This is it for CC_ω, the *negative* part of CIC. We can now turn to the last letter of this grandiose acronym: *I for inductive types*.

All your base types are belong to us Until now, whether it was System T or MLTT, our theories only featured one base type, $\mathbb{N}$. This is rather poor, as there is more to life than counting stuff. In CIC, this limitation is removed by the addition of a general pattern to derive *inductive types*. Let us present them through a series of portraits. The syntax we will use in this thesis is the following:

```
Inductive ℕ : □_i :=
| O : ℕ
| S (n : ℕ) : ℕ
```

From this, Coq automatically derives $\mathbb{N}_{\mathbf{ind}}$, the induction principle we already encountered in MLTT. With the same syntax, we could also define booleans:

```
Inductive 𝔹 : □_i :=
| true : 𝔹
| false : 𝔹
```

We would then recover another induction principle, called $\mathbb{B}_{\mathbf{ind}}$, with the following rules:

$$\frac{\Gamma, x : \mathbb{B} \vdash P \quad \Gamma \vdash t_{\mathsf{true}} : P\{x := \mathsf{true}\} \quad \Gamma \vdash b : \mathbb{B} \quad \Gamma \vdash t_{\mathsf{false}} : P\{x := \mathsf{false}\}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b : P\{x := b\}}$$

$$\frac{\Gamma, x : \mathbb{B} \vdash P \quad \Gamma \vdash t_{\mathsf{true}} \equiv t'_{\mathsf{true}} : P\{x := \mathsf{true}\} \quad \Gamma \vdash b \equiv b' : \mathbb{B} \quad \Gamma \vdash t_{\mathsf{false}} \equiv t'_{\mathsf{false}} : P\{x := \mathsf{false}\}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b \equiv \mathbb{B}_{\mathbf{ind}}\ P\ t'_{\mathsf{true}}\ t'_{\mathsf{false}}\ b' : P\{x := b\}}$$

$$\frac{\begin{array}{lll}\Gamma, x : \mathbb{B} & \vdash & P \\ \Gamma & \vdash & t_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma & \vdash & t_{\mathsf{false}} : P\{x := \mathsf{false}\}\end{array}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{true} \equiv t_{\mathsf{true}} : P\{x := \mathsf{true}\}}$$

$$\frac{\begin{array}{lll}\Gamma, x : \mathbb{B} & \vdash & P \\ \Gamma & \vdash & t_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma & \vdash & t_{\mathsf{false}} : P\{x := \mathsf{false}\}\end{array}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{false} \equiv t_{\mathsf{false}} : P\{x := \mathsf{false}\}}$$

The reader may have recognized in

$$\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b$$

a dependent variant of

$$\texttt{if}\ b\ \texttt{then}\ t_{\mathsf{true}}\ \texttt{else}\ t_{\mathsf{false}},$$

widespread in programming languages.

Induction principles are often written using Π-types for the predicate, as follows:

$$\frac{\Gamma \vdash P : \mathbb{B} \to \square \quad \Gamma \vdash t_{\mathsf{true}} : P\ \mathsf{true} \quad \Gamma \vdash b : \mathbb{B} \quad \Gamma \vdash t_{\mathsf{false}} : P\ \mathsf{false}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b : P\ b}$$

However, due to bad interactions with cumulativity this wording led to a bug in Coq, and Π-types were removed from pattern-matching implementation, as explained by *Sozeau et al* [127] and in *Lennon-Bertrand* [96]. We follow their lead and display dependent elimination without dependent products for the predicate. However, to ease the reading we will make a slight abuse of notation when defining functions using dependent eliminators in the rest of the thesis, and we will write

[127]: Sozeau et al. (2022), "The Curious Case of Case: Correct & Efficient Representation of Case Analysis in Coq and MetaCoq"

[96]: Lennon-Bertrand (2022), "Bidirectional Typing for the Calculus of Inductive Constructions"

$$\mathbb{B}_{\mathbf{ind}}\ (\lambda x : \mathbb{B}.\ P)\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b \qquad \text{to mean} \qquad \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b$$

when x appears as a free variable in P.

Another quite straightforward inductive construction is the type of *falsity*, aptly dubbed the *empty type*:

$$\texttt{Inductive}\ \bot : \square_i := .$$

Its recursor is quite interesting:

$$\frac{\begin{array}{lll}\Gamma, x : \bot & \vdash & P \\ \Gamma & \vdash & e : \bot\end{array}}{\Gamma \vdash \bot_{\mathbf{ind}}\ P\ e : P\{x := e\}}$$

Indeed, as $\bot$ is empty there is no base case for its induction principle: as soon as we have an inhabitant of $\bot$, we can produce inhabitants

of any type P. This is the logical rule for falsity, often called *proof by contradiction*, sometimes quoted in latin: *ex falso quodlibet*, literally *out of false, what you want*.

At that point, one should start to get an intuition of how inductive types work: to build a term of a given inductive I, one starts with the base constructors (constructors that do not ask for an argument of type I, such as O for $\mathbb{N}$ or true and false for $\mathbb{B}$) then stacks any number of recursive constructors one wants (constructors that ask for an argument of type I, such as S for $\mathbb{N}$). The number of recursive constructors must however be finite: the infinite string

$$\mathsf{S}\ (\mathsf{S}\ (\ldots))$$

is *not* an element of $\mathbb{N}$. When there is no base constructor, as in $\bot$, then the type is empty in the empty context.

In an extended context, however, one can always suppose $x : I$ to inhabit I.

Conversely, this intuition applies to recursors: as a general pattern, if one is able to provide a proof of

$$P\{x := c\}$$

for any base constructor c of I and if one is able to propagate these proofs through the recursive constructors, then one is able to recover a proof of

$$P\{x := i\}$$

for any term $i : I$.

Parameters by the meter We can however go beyond simple types such as $\mathbb{B}$, $\bot$ and $\mathbb{N}$. Indeed, *dependent type theory* would not be true to its name if inductive types could not depend on other types. This is the case of *lists*, inductively defined in Figure 1.6.

$$\begin{array}{l}\texttt{Inductive}\ \mathsf{list}\ (A : \square_i) : \square_i :=\\ \mid \mathsf{nil} : \mathsf{list}\ A\\ \mid \mathsf{cons}\ (a : A)\ (l : \mathsf{list}\ A) : \mathsf{list}\ A.\end{array}$$

$$\frac{\begin{array}{lll}\Gamma, x : \mathsf{list}\ A & \vdash & P\\ \Gamma & \vdash & t_{\mathsf{nil}} : P\{x := \mathsf{nil}\}\\ \Gamma & \vdash & t_{\mathsf{cons}} : \Pi(a : A)\,(l : \mathsf{list}\ A).\, P\{x := l\} \to P\{x := \mathsf{cons}\ a\ l\}\\ \Gamma & \vdash & l : \mathsf{list}\ A\end{array}}{\Gamma \vdash \mathsf{list}_{\mathbf{ind}}\ A\ P\ t_{\mathsf{nil}}\ t_{\mathsf{cons}}\ l : P\{x := l\}}$$

Figure 1.6.: Lists in CIC

Lists will be used throughout this thesis, often with the abbreviation

$$a :: l \qquad \text{to designate} \qquad \mathsf{cons}\ a\ l.$$

For the sake of conciseness, we skip computation rules for $\mathsf{list}_{\mathbf{ind}}$, as they are rather close to those of $\mathbb{N}_{\mathbf{ind}}$, albeit with an additional parameter A.

Lists are rather intuitive and programmers should be familiar with them. However, they do not make use of the full power of *dependence*.

Indeed, A is fixed at the beginning of the inductive definition, and constructors have no say in it. In such cases, we say that A is a *parameter* of the inductive list A. Could we imagine a type that changes depending on the constructors we use ? A first example would be the one of *vectors*, defined in Figure 1.7. Vectors are essentially lists whose length is disclosed by the type.

$$\begin{array}{l}\texttt{Inductive}\ \mathsf{vec}\ (A : \square_i) : \mathbb{N} \to \square_i :=\\ |\ \mathsf{vnil} : \mathsf{vec}\ A\ \mathsf{O}\\ |\ \mathsf{vcons}\ (n : \mathbb{N})\ (a : A)\ (l : \mathsf{vec}\ A\ n) : \mathsf{vec}\ A\ (\mathsf{S}\ n).\end{array}$$

$$\frac{\begin{array}{lll}\Gamma, x : \mathbb{N}, y : \mathsf{vec}\ A\ x & \vdash & P\\ \Gamma & \vdash & t_{\mathsf{vnil}} : P\{x := \mathsf{O}; y := \mathsf{vnil}\}\\ \Gamma & \vdash & t_{\mathsf{vcons}} : \Pi(n : \mathbb{N})\ (a : A)\ (v : \mathsf{vec}\ A\ n).\\ & & \qquad P\{x := n; y := v\} \to\\ & & \qquad P\{x := \mathsf{S}\ n; y := \mathsf{vcons}\ n\ a\ v\}\\ \Gamma & \vdash & n : \mathbb{N}\\ \Gamma & \vdash & v : \mathsf{vec}\ A\ n\end{array}}{\Gamma \vdash \mathsf{vec}_{\mathbf{ind}}\ A\ P\ t_{\mathsf{vnil}}\ t_{\mathsf{vcons}}\ n\ v : P\{x := n; y := v\}}$$

Figure 1.7.: Vectors in CIC

Here, constructors specify at which value of $\mathbb{N}$ they inhabit vec, and the natural number they harbour is called an *index*. The constructor vnil, for instance, can only ever live in vec A O.

When writing functions such as hd (which returns the head of a non-empty list) or tl (which returns the tail of a non-empty list), this allows the user to make sure that said function will never be applied to an empty list, by mere virtue of typing.

Are you my equal? However, vectors are not the end of our trip, and we have yet to reach the pinnacle of dependency, the paramount *equality type*.

$$\begin{array}{l}\texttt{Inductive}\ \mathsf{eq}\ (A : \square_i)\ (x : A) : A \to \square_i :=\\ |\ \mathsf{refl} : \mathsf{eq}\ A\ x\ x\end{array}$$

$$\frac{\begin{array}{lll}\Gamma, x : A, y : \mathsf{eq}\ A\ a\ x & \vdash & P\\ \Gamma & \vdash & t_{\mathsf{refl}} : P\{x := a; y := \mathsf{refl}\ a\}\\ \Gamma & \vdash & a' : A\\ \Gamma & \vdash & e : \mathsf{eq}\ A\ a\ a'\end{array}}{\Gamma \vdash \mathsf{eq}_{\mathbf{ind}}\ A\ a\ P\ t_{\mathsf{refl}}\ a'\ e : P\{x := a'; y := e\}}$$

Figure 1.8.: Equality type in CIC

In the remainder of this thesis, we will write

$$x = y \qquad \text{to mean} \qquad \mathsf{eq}\ A\ x\ y,$$

omitting A because it can be retrieved from x and y.

Equality is a tricky type that raises many questions, the first of which being, why do we consider this particular inductive to be the type of equality ? The first properties one might expect from equality are those of an equivalence relation:

- equality should be reflexive: x should be equal to itself;
- equality should be symmetric: if $x = y$ then $y = x$;
- equality should be transitive: if $x = y$ and $y = z$ then $x = z$.

Reflexivity is exactly refl. Moreover, making good use of $\mathsf{eq}_{\mathbf{ind}}$, we can show the last two properties. For instance, symmetry is proven by the following term, while transitivity is very similar:

$$\begin{aligned}&\lambda(A : \square_i)\,(x\ y : A)\,(e : x = y).\\ &\quad \mathsf{eq}_{\mathbf{ind}}\ A\ x\ (\lambda(z : A)\,(\_ : x = z).\, z = x)\ (\mathsf{refl}\ x)\ y\ e :\\ &\qquad \Pi(A : \square_i)\,(x\ y : A).\, x = y \to y = x.\end{aligned}$$

Moreover, eq should respect *Leibniz*'s definition of equality [95]: two objects are equal iff they satisfy the same properties. Intuitively, it entails that two objects x and y should be said *equal* when the system is unable to distinguish them. This effectively means that given

[95]: Leibniz (1686), "Discourse on Metaphysics"

$$f : A \to B \qquad \text{and} \qquad x, y : A \qquad \text{such that} \qquad x = y$$

then we can deduce

$$f\ x = f\ y.$$

This is easily derivable from $\mathsf{eq}_{\mathbf{ind}}$.

On the other hand, there are things that should *not* be equal: true and false, O and S O, and so on. In CIC, negation of a property P is interpreted as a function from P to $\bot$. Once more, $\mathsf{eq}_{\mathbf{ind}}$ proves useful and allows us to build a function

$$f : \mathsf{true} = \mathsf{false} \to \bot,$$

as follows: first, we define

$$\begin{array}{lcl} P & : & \Pi(b : \mathbb{B})\,(e : \mathsf{true} = b).\, \square_i \\ P\ \mathsf{true}\ \_ & \coloneqq & \top \\ P\ \mathsf{false}\ \_ & \coloneqq & \bot \end{array}$$

where $\top$ is the always inhabited type with only one element $\star : \top$. Then we make use of our proof that true = false in the following way:

$$f \coloneqq \lambda e : \mathsf{true} = \mathsf{false}.\ \mathsf{eq}_{\mathbf{ind}}\ \mathbb{B}\ \mathsf{true}\ P\ \star\ \mathsf{false}\ e.$$

Unfolding the definition of $\mathsf{eq}_{\mathbf{ind}}$ and P, we can see that

$$P\ \mathsf{true}\ (\mathsf{refl}\ \mathsf{true}) \equiv \top : \square_i$$

hence $\star : \top$ is a proof of P true (refl true). Moreover, we have a proof

$$e : \mathsf{true} = \mathsf{false},$$

thus we indeed retrieve a proof of

$$P\ \mathsf{false}\ e \equiv \bot : \square_i.$$

All of this advocates for eq being a good candidate to interpret equality. Still, its formulation can be unsettling. We only have one constructor,

$$\mathsf{refl} : \mathsf{eq}\ A\ a\ a.$$

Does this mean that all we are ever going to prove are equalities between some term $a : A$ and itself? This would be rather poor. Fortunately, this is not the case, thanks to two pillars of type theory: conversion and contexts.

Firstly, conversion already equates some terms that are syntactically different but computationally the same. Let us recall for instance our definition of addition from Section 1.1:

$$\begin{array}{lcl} \mathsf{add} & : & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \mathsf{add}\ \mathsf{O}\ m & \coloneqq & m \\ \mathsf{add}\ (\mathsf{S}\ k)\ m & \coloneqq & \mathsf{add}\ k\ (\mathsf{S}\ m) \end{array}$$

In CIC, it is still true that $4+3 \equiv 7$. Now, already in MLTT we explained that computation exists at the level of types. This means that

$$\mathsf{eq}\ \mathbb{N}\ (4+3)\ 7 \equiv \mathsf{eq}\ \mathbb{N}\ 7\ 7.$$

Then, since

$$\mathsf{refl}\ 7 \qquad \text{is of type} \qquad \mathsf{eq}\ \mathbb{N}\ 7\ 7,$$

thanks to the conversion rule of CIC it is also an inhabitant of

$$\mathsf{eq}\ \mathbb{N}\ (4+3)\ 7, \qquad \text{which means that} \qquad 4+3=7.$$

Thus, equality *encompasses* conversion.

Still, looking at our definition of addition, we notice that it computes by recursion on its first argument. This means that given a variable $n : \mathbb{N}$ in the context,

$$\mathsf{O} + n \equiv n \qquad \text{but} \qquad n + \mathsf{O} \not\equiv n.$$

Indeed, when facing a variable, computation is *blocked*. This entails that

$$\mathsf{refl}\ n : n = n \qquad \text{is } \textit{not} \text{ a proof of} \qquad n + \mathsf{O} = n.$$

However, reflexivity is not the only way to inhabit equality. Using $\mathbb{N}_{\mathbf{ind}}$, we can *prove* that $n + \mathsf{O} = n$. All we need is a proof

$$\Gamma \vdash t_{\mathsf{O}} : \mathsf{O} + \mathsf{O} = \mathsf{O}$$

which follows from

$$\mathsf{refl}\ \mathsf{O}$$

and computation, and a proof

$$\Gamma, n : \mathbb{N}, H_n : n + \mathsf{O} = n \vdash (\mathsf{S}\ n) + \mathsf{O} = \mathsf{S}\ n$$

which can be obtained through $\mathsf{eq}_{\mathbf{ind}}$ applied to H_n. Thus, as soon as we have assumptions in the context, refl stops being the only way to derive equality.

Equating equalities How many proofs of equalities do we want? If we find many ways to prove equality between two terms, should those ways be equal? Convertible? For simple enough types, such as $\mathbb{B}$ or $\mathbb{N}$, it is possible to prove *uniqueness of identity proofs* (UIP). For $\mathbb{N}$, this means:

$$\Pi(n\ m : \mathbb{N})\,(e_1\ e_2 : n = m).\ e_1 = e_2.$$

More generally, as observed by *Hedberg* [72], UIP is derivable for any type where we can build a *decision procedure* for equality. This result was later extended by *Kraus et al* [92]. The idea is, given two booleans $b_1, b_2 : \mathbb{B}$, they are equal if and only if the two of them are true or the two of them are false; no matter what fancy proof we find, it will at the end reduce to this simple fact.

[72]: Hedberg (1998), "A coherence theorem for Martin-Löf's type theory"

Hedberg's result is formalized in the Standard Library of Coq, it can be found here.

[92]: Kraus et al. (2013), "Generalizations of Hedberg's Theorem"

However, for functional types such as

$$\mathbb{N} \to \mathbb{B}$$

such a decision procedure is out of reach. Given two functions

$$f_1, f_2 : \mathbb{N} \to \mathbb{B},$$

a mathematician's intuition would be to check whether f_1 and f_2 are equal on every input, but:

- This would mean applying f_1 and f_2 to every possible natural number, which would take infinitely long, thus can hardly be called a reasonable procedure;
- Even assuming we manage this impossible feat, at the end of eternity we still only get

  $$\Pi(n : \mathbb{N}).\ f_1\ n = f_2\ n,$$

  and in CIC there is no way to retrieve $f_1 = f_2$ from this. The fact that *two functions are equal iff they are equal on every argument* is called *function extensionality* (funext) and is not provable in CIC.

Hoffman and Streicher [76, 131] proved that there is no hope to prove uniqueness of identity proofs or its close relative, the *K axiom*, as a general principle in CIC. Some would argue that this is a serious limitation: two objects are either equal, thus perfectly identical, or not equal, hence there is no way to prove their equality, but the fact that *different ways of being equal* can cohabit is philosophically wrong.

[76]: Hofmann et al. (1994), "The Groupoid Model Refutes Uniqueness of Identity Proofs"
[131]: Streicher (1993), "Investigations into intensional type theory"

This is part of a wider debate on the status of *proofs*: suppose we have a predicate

$$P : \mathbb{N} \to \square$$

such that $P\ n$ means that n is a prime number. Then should there be many different ways to prove $P\ 5$ or should all these proofs be considered equal? The latter option is called *proof irrelevance*. Coq, for instance, features a universe of propositions

$$* : \square$$

where the user can add as an axiom that

$$\text{if } A : * \quad \text{and} \quad x, y : A \quad \text{then} \quad x = y.$$

Some, as *Gilbert et al* [59], go even further and ask that

$$\text{if } A : * \qquad \text{and} \qquad x, y : A \qquad \text{then} \qquad x \equiv y;$$

a feature called *definitional proof-irrelevance* and implemented in the Agda proof-assistant, and in Coq under the name `SProp`.

[59]: Gilbert et al. (2019), "Definitional Proof-Irrelevance without K"

Given proof-irrelevance, one can basically ask for the equality type to live in $*$ and get UIP. This line of thought led to *Observational Type Theory*, advocated for by *Altenkirch et al* [8] and recently implemented by *Pujet et al* [119, 120].

[8]: Altenkirch et al. (2007), "Observational equality, now!"

[119]: Pujet et al. (2023), "Impredicative Observational Equality"

[120]: Pujet et al. (2022), "Observational Equality: Now for Good"

This is not the only path, though, and one can be even more brutal by adding *equality reflection*:

$$\frac{\Gamma \vdash e : x = y : A}{\Gamma \vdash x \equiv y : A}$$

Now every proof of equality can be turned into a conversion rule, collapsing all proofs of equality until refl is the only one left standing. The resulting theory is called *Extensional Type Theory* (ETT). This is a quite barbarian way of dealing with the issue, and it cannot go without some casualties. Indeed, assuming for instance we have a type A such that if some Turing machine M terminates on some input t then

$$A \coloneqq \mathbb{N} \qquad \text{else} \qquad A = \mathbb{B}$$

then with equality reflection we get A such that if some Turing machine M terminates on some input t then

$$A \equiv \mathbb{N} \qquad \text{else} \qquad A \equiv \mathbb{B}$$

which means that

$$\mathrm{O} : A \qquad \text{iff some Turing-machine } M \text{ terminates on some input } t.$$

Now, to automatically check whether

$$\mathrm{O} : A$$

or not, Coq or any proof assistant based on type theory would have to automatically prove or disprove termination of Turing-machines. However, this problem is known to be undecidable and, using a variant of our informal argument, *Hoffman* [75] formally proved *undecidability of type-checking in* ETT. *Type-checking* is the task of ascertaining whether a given term t is or is not of a given type A and, in most proof assistants, such as Coq, it is considered crucial to feature an automatic type-checker, meaning decidability of type-checking is necessary.

[75]: Hofmann (1997), *Extensional constructs in intensional type theory*

Still, some adventurers have followed the tortuous path of undecidability, building proof assistants without a *type-checker* verifying that terms are well-typed, but rather a *derivation-checker* verifying whole proof trees. Two salient examples of such endeavours are `NuPRL` [6], a proof assistant based on realisability where equality reflection is derivable, and `Andromeda` [19], a flexible tool where the user can add equality reflection and write their own algorithm to typecheck part of the resulting theory.

[6]: Allen et al. (2000), "The Nuprl Open Logical Environment"

[19]: Bauer et al. (2018), "Design and Implementation of the Andromeda Proof Assistant"

On a less adventurous note, *Winterhalter et al* [144] provided a translation from ETT to *Intensional Type Theory* (ITT, understood here as CC_ω with dependent sums and equality) extended with funext and UIP. Their translation builds upon previous work by *Hofmann [74, 75]* and *Oury* [111]. Finally, let us say that a restriction of the reflection rule retaining decidability of type checking can be found, which led to the inclusion of *rewrite rules* in the Agda proof assistant [34].

[144]: Winterhalter et al. (2019), "Eliminating reflection from type theory"

[74]: Hofmann (1995), "Conservativity of Equality Reflection over Intensional Type Theory"
[75]: Hofmann (1997), *Extensional constructs in intensional type theory*

[111]: Oury (2005), "Extensionality in the Calculus of Constructions"
[34]: Cockx et al. (2021), "The Taming of the Rew: A Type Theory with Computational Assumptions"

Paved with good intensions We just mentioned two concepts that are often used but rarely defined in type theory: *extensionality* and *intensionality*. As a general principle, the more a theory or its interpretation describes its objects as black-boxes only caracterized by their input/output interactions, the more we will call it *extensional*. Function extensionality, which states that two functions that return the same output when fed the same input, is an obvious example of an extensional principle. Another one is *proposition extensionality* (propext), which states that any two propositions that are equivalent are equal:

$$\mathsf{propext} \coloneqq \Pi(A\ B : *).\ (A \to B) \to (B \to A) \to A = B.$$

Usual set theory such as *ZF* set theory [148, 149] is very extensional, as functions are defined as relations between inputs and outputs. In type theory and in the proof-as-programs world, things are not so clear and different interpretations cohabit. The more a theory or its interpretation attaches importance to the internal, computational entrails of such objects, the more it will be worthy of the name *intensional*. As we try to give computational content to classical principles, we are naturally inclined to prefer the latter. We will consider intensional interpretations of CIC later in this Chapter, and Chapter 2 will be devoted to studying different notions of continuity, some more intensional than others.

[148]: Zermelo (1904), "Beweis, dass jede Menge wohlgeordnet werden kann: Aus einem an Herrn Hilbert gerichteten Briefe"
[149]: Zermelo (1908), "Untersuchungen über die Grundlagen der Mengenlehre. I"

A concurrent and complementary line of work turns the table and explains that terms can be seen as *points* in a space, where equality of terms is merely a *path* between two points. Depending on the space we consider, there can then be many different paths between two points, or even between a point and itself. A relaxed version of equality arises, where equality between functions validates funext and proofs of equality between types are akin to *isomorphisms of types*, close to the mathematical intuition that different objects sharing the same structure should be identified. More than a decade ago, *univalent type theory* was born [140], a research field still prolific as of today.

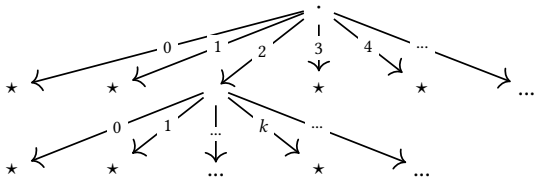
[140]: Univalent Foundations Program (2013), *Homotopy Type Theory: Univalent Foundations of Mathematics*

Higher, deeper, larger, stronger Going back to our zoo of inductives, we should stress that *higher-order types* can live among the arguments of constructors of inductive types. For instance, Coq accepts definitions such as

$$\begin{array}{l}\texttt{Inductive}\ \mathbb{K} :=\\ \mid \star : \mathbb{K}\\ \mid \beta_{\mathbb{K}} : (\mathbb{N} \to \mathbb{K}) \to \mathbb{K}\end{array}$$

where $\beta_{\mathbb{K}}$ takes as argument a function from $\mathbb{N}$ to $\mathbb{K}$. Intuitively, this means that elements of $\mathbb{K}$ are infinitely wide trees: as an example,

is a element of $\mathbb{K}$, as well as



A variant of this tree-like structure will be all the rage in Chapter 2.

Stay positive Allowing such quantifications is quite powerful and can quickly lead to inconsistencies if not done with care. For instance, the similar following inductive burns our system down to ashes:

$$\begin{array}{l}\texttt{Inductive}\ \mathbb{K}' :=\\ \mid \Lambda_{\mathbb{K}'} : (\mathbb{K}' \to \mathbb{K}') \to \mathbb{K}'\end{array}$$

Should its recursor $\mathbb{K}'_{\mathbf{ind}}$ exists, it would satisfy the following rule:

$$\frac{\begin{array}{lll}\Gamma, x : \mathbb{K}' & \vdash & P : \square_i\\ \Gamma, & \vdash & p_\Lambda : \Pi(f : \mathbb{K}' \to \mathbb{K}')\,(k : \mathbb{K}').\, P\{x := f\ k\} \to P\{x := \Lambda_{\mathbb{K}'}\ f\}\\ \Gamma & \vdash & k : \mathbb{K}'\end{array}}{\Gamma \vdash \mathbb{K}'_{\mathbf{ind}}\ P\ p_\Lambda\ k : P\{x := k\}}$$

From this, we can derive falsity in the empty context. Indeed, taking

$$P := \bot, \qquad \text{we first have} \qquad x : \mathbb{K}' \vdash \bot,$$

which is the first premise of $\mathbb{K}'_{\mathbf{ind}}$. The second premise is direct:

$$\cdot \vdash \lambda(\_ : \mathbb{K}' \to \mathbb{K}')\,(\_ : \mathbb{K}')\,(H : \bot).\, H : (\mathbb{K}' \to \mathbb{K}') \to \mathbb{K}' \to \bot \to \bot.$$

Finally, we can inhabit $\mathbb{K}'$ with the following:

$$\cdot \vdash \Lambda_{\mathbb{K}'}\ (\lambda x : \mathbb{K}'.\, x) : \mathbb{K}',$$

which concludes of proof of $\bot$.

All of this advocates for a touchstone, able to separate "harmless" inductive types from inconsistent ones. Such a rule does exist, in the shape of a *positivity criterion*. To summarize, it states that when defining an inductive I, the type I should *never appear on the left side of an arrow*. This means that for any given type A,

$$c_i : (A \to I) \to I$$

is a valid constructor of I while

$$c_i' : (I \to A) \to I$$

is not. The intuition is the following:

- in the former case, to build an element of I using c_i the user needs to produce *a function that produces elements of type I*. Quantifying over functionals does not relieve the user from his need to provide an element of I;
- in the latter case, to build an element of I using c_i' the user need to produce *a function that consumes elements of type I*. This is overstepping his prerogatives: instead of providing an element of I, he can now simply ask for it. This dangerous behaviour is excluded from CIC.

A proper account of the positivity criterion can be found in *Paulin-Mohring* [112].

[112]: Paulin-Mohring (1993), "Inductive Definitions in the system Coq - Rules and Properties"

1.4. BTT

We now leave CIC behind, godless hive of dictatorial, shape-shifting bees, bossing and buzzing and swarming around to keep the nest running. In this curious maze, the many tunnels of typing cross the countless roads of computation, and cumulativity ladders give access to an infinite stack of identical floors, where everyone despises those who live below. A single stair taken downwards could make this gigantic, well-founded skyscraper collapse, yet term workers still turn into type queens, then into terms again,in a positive, well-crafted, perfectly timed choreography, with everyone fitting and everything clicking. Every term abides by the thirty-three Commandments of type theory, law is enforced at every corner case of the hive, but in the shadows of the over-disciplined, surgically clean world, some harbour a different kind of dream. Marginals, impures, non-standards: they bear many names; they are the *deviants*, those who live outside the norm, who failed to fit in the overarching mold of CIC and were pushed back in the growing slums of untyped terms, away from the authoritarian utopia, alone and longing for a place they could call home.

Baclofen Type Theory (BTT) was born from the desire of *Pédrot and Tabareau* [121] to accomodate some of those bohemians in the cold world of CIC. Said bohemians are called *effects*. Effects are an evading notion, whose precise limits are hard to outline. We thus will not try and give an overarching definition in this thesis, but will rather restrict ourselves to *observable effects*, a definition of which was given by *Pédrot and Tabareau [116]*.

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

[116]: Pédrot et al. (2020), "The fire triangle: how to mix substitution, dependent elimination, and effects"

Definition 1.4.1: Observable effects
Given a type theory $\mathcal{T}$, $\mathcal{T}$ features *observable effects* when it is possible to build a closed term

$$b : \mathbb{B} \qquad \text{and a function} \qquad C : \mathbb{B} \to \square$$

such that

$$\begin{aligned} C\ \mathsf{true} &\equiv \top \\ C\ \mathsf{false} &\equiv \top \\ C\ b &\equiv \bot \end{aligned}$$

Exceptions with handlers, backtracking, interaction with a user... Observable effects are everywhere in programming languages. Yet they are very difficult to deal with in CIC as adding them breaks the consistency of the system. Indeed, as *Pédrot and Tabareau* explain, assuming an observable effect

$$b : \mathbb{B} \qquad \text{together with a discriminating function} \qquad C : \mathbb{B} \to \square$$

then we can easily build a proof of $\bot$ using $\mathbb{B}_{\mathbf{ind}}$. All we need to do is set

$$\begin{aligned} P &:= C \\ t_{\mathsf{true}} &:= \star : C\ \mathsf{true} \\ t_{\mathsf{false}} &:= \star : C\ \mathsf{false} \end{aligned}$$

then we can derive

$$\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b : C\ b \equiv \bot.$$

We need some leeway here. At this crossroads, three paths lie before us:

- restrict substitution so that

  $$\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b$$

  is of type $C\ b$ but we do not translate this into a proof of $\bot$. This can be done by introducing a value restriction, meaning that the $C\ b$ computation is blocked because b is not a value. This is the path followed by *Lepigre* in [97];
- accept inconsistency as a necessary evil for more expressiveness and look at effectful CIC not as a logic theory but rather as an effectful, dependently-typed programming language, as in *Pédrot and Tabareau* [122];
- restrict dependent elimination so that

  $$C\ b \equiv \bot$$

  but

  $$\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b$$

  is *not* of type $C\ b$. This is *Baclofen Type Theory* [121];
- a fourth path also lies behind us, the one we came from: abandoning effects and going back to our beloved, unsullied CIC.

[97]: Lepigre (2016), "A Classical Realizability Model for a Semantical Value Restriction"

[122]: Pédrot et al. (2018), "Failure is Not an Option An Exceptional Type Theory"

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

The name *Baclofen Type Theory* is not very informative. It comes from a play on words involving *an effectful way to restrict addiction to dependence*, the title of the paper introducing this particular theory, and Baclofen, a drug sometimes used to treat opioid withdrawal symptoms.

Drugstore operator As the name of the Section heavily hints at, we will choose the *Baclofen* road.

Like its parent CIC, BTT is based on the predicative calculus of constructions CC_ω. It however stands out on the inductive side, for its relaxed version of *dependent elimination*, another name for the induction principles we already encountered.

Indeed, contrarily to CIC which has a single dependent eliminator $\mathscr{I}_{\mathbf{ind}}$ for any given inductive type $\mathscr{I}$, BTT has two eliminators: a non-dependent one $\mathscr{I}_{\mathbf{rec}}$, identical to the one we encountered when dealing with System T, and a strict dependent one $\mathscr{I}_{\mathbf{sind}}$. These three eliminators enjoy the same computational ι-rules, i.e. they reduce on constructors. The difference lies in their typing rules: the predicate of $\mathscr{I}_{\mathbf{rec}}$ does not depend on its inductive argument, i.e. it is basically simply-typed, while the predicate of $\mathscr{I}_{\mathbf{sind}}$ is wrapped in a *storage operator* $\mathscr{I}_{\mathbf{str}}$, similar to the ones used by *Krivine* [94], that locally evaluates its argument in a by-value fashion.

[94]: Krivine (1990), "Opérateurs de mise en mémoire et traduction de Gödel"

To make things clear, let us get back to our usual example of natural numbers. As already pointed out, the non-dependent one $\mathbb{N}_{\mathbf{rec}}$ is the same as in System T:

$$\frac{\Gamma \vdash P : \square \qquad \Gamma \vdash t_{\mathrm{O}} : P \qquad \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}} : P}$$

Using this non-dependent eliminator, we can define the *storage operator* $\mathbb{N}_{\mathbf{str}}$ and the dependent eliminator $\mathbb{N}_{\mathbf{sind}}$ in a systematic way, as done in Figure 1.9.

$$\begin{aligned}
&\mathbb{N}_{\mathbf{str}}\ (n : \mathbb{N})\ (P : \mathbb{N} \to \square) : \square :=\\
&\quad \mathbb{N}_{\mathbf{rec}}\ ((\mathbb{N} \to \square) \to \square)\ (\lambda Q : \mathbb{N} \to \square.\, Q\ \mathrm{O})\\
&\qquad (\lambda(m : \mathbb{N})\,(_ : (\mathbb{N} \to \square) \to \square)\,(Q : \mathbb{N} \to \square).\, Q\ (\mathrm{S}\ m))\ n\ P.
\end{aligned}$$

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{N} \vdash & P\\ \Gamma \vdash & t_{\mathrm{O}} : P\{x := \mathrm{O}\}\\ \Gamma \vdash & t_{\mathrm{S}} : \Pi y : \mathbb{N}.\, P\{x := y\} \to P\{x := \mathrm{S}\ y\}\\ \Gamma \vdash & n : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{sind}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : \mathbb{N}_{\mathbf{str}}\ n\ P}$$

Figure 1.9.: Dependent eliminator for $\mathbb{N}$ in BTT

The syntax of $\mathbb{N}_{\mathbf{str}}$ and $\mathbb{N}_{\mathbf{rec}}$ is quite a bit abstruse, so let us pause here, to try and grasp the meaning of it all. Let us first notice that $\mathbb{N}_{\mathbf{rec}}$ is equipped with the usual conversion rules:

$$\frac{\begin{array}{l}\Gamma \vdash P\\ \Gamma \vdash t_{\mathrm{O}} : P\\ \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ \mathrm{O} \equiv t_{\mathrm{O}} : P\{x := \mathrm{O}\}} \qquad \frac{\begin{array}{l}\Gamma \vdash P\\ \Gamma \vdash t_{\mathrm{S}} : \mathbb{N} \to P \to P\\ \Gamma \vdash t_{\mathrm{O}} : P\\ \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ (\mathrm{S}\ n) \equiv t_{\mathrm{S}}\ n\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n) : P\{x := \mathrm{S}\ n\}}$$

This gives us some hints when unfolding the definition of $\mathbb{N}_{\mathbf{str}}$. We have the following:

$$\begin{array}{lcl} \mathbb{N}_{\mathbf{str}}\ \mathrm{O}\ P & \equiv & P\{x \coloneqq \mathrm{O}\} \\ \mathbb{N}_{\mathbf{str}}\ (\mathrm{S}\ n)\ P & \equiv & \mathbb{N}_{\mathbf{str}}\ n\ (\lambda x : \mathbb{N}.P\ (\mathrm{S}\ x)) \end{array}$$

From this, we can deduce that for any term of the form

$$\mathrm{S}^n\ \mathrm{O}, \qquad \text{we have that} \qquad \mathbb{N}_{\mathbf{str}}\ (\mathrm{S}^n\ \mathrm{O})\ P \equiv P\ (\mathrm{S}^n\ \mathrm{O}).$$

Yet this is not the case for all terms, as once again variables block computation and

$$\mathbb{N}_{\mathbf{str}}\ x\ P \not\equiv P\ x.$$

Moreover, going back to our observable effect

$$b : \mathbb{B} \qquad \text{and its discriminating function} \qquad C : \mathbb{B} \to \square,$$

we can still set

$$\begin{array}{lcl} P & \coloneqq & C \\ t_{\mathsf{true}} & \coloneqq & \star : C\ \mathsf{true} \\ t_{\mathsf{false}} & \coloneqq & \star : C\ \mathsf{false} \end{array}$$

but this time we get

$$\mathbb{B}_{\mathbf{sind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b : \mathbb{B}_{\mathbf{str}}\ b\ C$$

where $\mathbb{B}_{\mathbf{str}}$ is similar to $\mathbb{N}_{\mathbf{str}}$. Its definition is the following:

$$\begin{array}{l} \mathbb{B}_{\mathbf{str}}\ (b : \mathbb{B})\ (P : \mathbb{B} \to \square) : \square \coloneqq \\ \quad \mathbb{B}_{\mathbf{rec}}\ ((\mathbb{B} \to \square) \to \square)\ (\lambda Q : \mathbb{B} \to \square.\ Q\ \mathsf{true})\ (\lambda Q : \mathbb{B} \to \square.\ Q\ \mathsf{false})\ b\ P \end{array}$$

This entails

$$\begin{array}{lcl} \mathbb{B}_{\mathbf{str}}\ \mathsf{true}\ P & \equiv & P\ \mathsf{true} \\ \mathbb{B}_{\mathbf{str}}\ \mathsf{false}\ P & \equiv & P\ \mathsf{false} \end{array}$$

but computation on b is *not specified by* BTT. This means that when dealing with a specific effect, we can choose the computational content of $\mathbb{B}_{\mathbf{str}}$ and make sure it is harmless. In Chapter 3, we will encounter a model featuring effects where for every effectful $b : \mathbb{B}$ we will have

$$\mathbb{B}_{\mathbf{str}}\ b\ P \equiv \top.$$

Then of course, $\mathbb{B}_{\mathbf{sind}}$ will act accordingly and for the same effectful $b : \mathbb{B}$ we will get:

$$\mathbb{B}_{\mathbf{sind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b \equiv \star : \top.$$

From this we will retrieve consistency of this particular model of BTT.

One might notice that, working from within CIC, using $\mathbb{N}_{\mathbf{ind}}$ it is possible to prove

$$\Pi(n : \mathbb{N})\ (P : \mathbb{N} \to \square).\ \mathbb{N}_{\mathbf{str}}\ n\ P = P\ n.$$

In that sense, BTT can be said finer-grained than CIC, as it allows to study objects that are collapsed together by CIC rules. It is similar to the way intuitionistic logic is finer-grained than classical logic where $\neg\neg A \leftrightarrow A$.

Classical logic does not not exist All the rage so far in this Section has been around effects as a computational tool. Is it to say that the *proof as programs* motto falls short of providing an interpretation for effects? Not at all, and for many years now effects have been linked to *classical principles*.

Classical logic is the system overwhelmingly used in mathematics, and the fact that it is *not* the logic implemented by CIC is worth mentioning. The most salient feature missing here is the infamous *double-negation elimination* (DNE), used by mathematicians when they derive proofs "by contradiction". It is displayed in Table 1.27. Roughly speaking, logics featuring this principle (such as usual *ZF* set theory [148, 149]) are deemed *classical*, and logics that do not (like CIC) are dubbed *constructive*. However, this distinction is not set in stone and, as we already explained in the beginning of the chapter, the purpose of this thesis is precisely to help bridge the gap between classical and constructive logic.

$$\frac{\Gamma \vdash t : \neg\neg A}{\Gamma \vdash \mathsf{DNE}\ t : A}$$

Table 1.27.: Double-negation elimination

[148]: Zermelo (1904), "Beweis, dass jede Menge wohlgeordnet werden kann: Aus einem an Herrn Hilbert gerichteten Briefe"
[149]: Zermelo (1908), "Untersuchungen über die Grundlagen der Mengenlehre. I"

We should first explain what we mean when we say that DNE is not part of CIC. Formally, this signifies that using the rules of CIC, it is not possible to build a term t such that

$$t : \Pi A : \square_i. \neg\neg A \to A.$$

Crucially, it is also not possible to prove that DNE is false in *CIC*, that is, build a term t such that

$$t : (\Pi A : \square_i. \neg\neg A \to A) \to \bot.$$

When such case arises with a type A such that neither A nor $\neg A$ are provable in a theory $\mathscr{T}$, we say that A is *independent* from $\mathscr{T}$.

We cannot prove double-negation elimination in CIC. Nonetheless, maybe we can add it as an axiom? As a matter of fact, yes. Since there is no proof of $\neg\mathsf{DNE}$ in CIC, adding this principle leads to a consistent system, which could be used to reason with.

A list of usual axioms one can safely add to Coq without breaking consistency is provided by the Coq team here.

However, if we do not provide *computation rules* for DNE we quickly end up in troubles. Indeed, it is for instance very easy to build a proof of

$$\neg\neg\mathbb{N}$$

using terms of type $\mathbb{N}$, like O. As an example, we can define:

$$t : (\mathbb{N} \to \bot) \to \bot \coloneqq \lambda P : \mathbb{N} \to \bot.\ P\ \mathsf{O}.$$

Then, appplying DNE we recover

$$\mathsf{DNE}\ t : \mathbb{N}.$$

Sadly, even though there is no variable in sight, this term does not compute and we cannot retrieve an actual number out of it. It is well-typed, we can add it, substract it, recurse on it but nothing will ever compute anymore: our system is effectively frozen.

Backtrack A This is an aporia. We built a system that is both a logic and a programming language, where computation relieves us from tedious work by giving us *for free* proofs of equality such as

$$0 + 7 \equiv 7,$$

yet if we want to reach classical logic's expressiveness we have to add an axiom that disrupts its computational endeavour. We need a way out. Could we add computation rules to DNE? This is not trivial: even if someone proves that it is impossible that a unicorn does not exist, computing said unicorn is easier said than done! Still, looking at our example

$$t : (\mathbb{N} \to \bot) \to \bot \coloneqq \lambda P : \mathbb{N} \to \bot.\ P\ \mathsf{O},$$

an idea springs to mind: to build a proof of

$$\bot \qquad \text{out of} \qquad P : \mathbb{N} \to \bot,$$

t applies P to O. Could we then say that

$$\mathsf{DNE}\ t \equiv \mathsf{O},$$

and generalize this strategy to every type

$$A \qquad \text{and every term} \qquad t : \neg\neg A?$$

This would make sense: hopefully our system is consistent, meaning that there is no inhabitant of $\bot$. Thus, to build a term of type $\bot$ out of a function

$$P : A \to \bot$$

there seems to be only one way: first get a term

$$a : A \qquad \text{then apply } P \text{ to get} \qquad P\ a : \bot.$$

Hence, by looking deep enough in the structure of the term, DNE should be able to extract and return $a : A$, thereby unfreezing our computational engine.

This intuition did in fact lead to computational interpretations of DNE, in the form of *control operators*. One of them, *Felleisen's* $\mathscr{C}$ operator, precisely validates the following rules:

$$\frac{}{\Gamma, H : \neg\neg A \vdash \mathscr{C}_A\ H : A} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathscr{C}_A\ (\lambda k : \neg A.\ k\ a) \equiv a : A}$$

Using $\mathscr{C}$, *Griffin* [68] gave computational content to DNE in simply-typed λ-calculus. Unfortunately, *Herbelin* [73] proved that this operator breaks consistency of any dependent type theory featuring dependent sums and equality. A few years earlier, a similar result was proven for *CPS*-translations, another way of providing computational content to DNE, by *Barthe and Uustalu* [15]. There are many hurdles along this sinous path, and explorers have yet to see the end of the road.

[68]: Griffin (1990), "A Formulae-as-Types Notion of Control"

[73]: Herbelin (2005), "On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic"

[15]: Barthe et al. (2002), "CPS translating inductive and coinductive types"

Independence is an exception This is not to say we are still stuck at bay, though, and some islands have already been reached by type theorists' ships. Indeed, there are many principles, called *semi-classical*, both weaker than DNE, yet not derivable in CIC, that have been given computational content through an effectful interpretation. One is the so-called *independence of premise* (IP). It states the following:

$$\mathsf{IP} : \square \coloneqq \Pi(A : \square)\,(B : \mathbb{N} \to \square).$$
$$(\neg A \to \Sigma(n : \mathbb{N}).\ B\,n) \to \Sigma(n : \mathbb{N}).\ \neg A \to B\,n$$

Many semi-classical principles are of this form, mixing functions, dependent sums and negation. *Double-negation shift*, a principle we will encounter in Chapter 2, is of the same kind.

In plain CIC, IP is not provable. Yet, by adding exceptions to CIC, *Pédrot and Tabareau* [122] managed to build a theory where IP is inhabited by a computing term. Delving into the nitty-gritty of their paper would be out of our scope, but we can give an intuition of their result. The main steps are the following:

[122]: Pédrot et al. (2018), "Failure is Not an Option An Exceptional Type Theory"

- They add a type $\mathbb{E}$ of exceptions to CIC, together with a function
$$\texttt{raise} : \Pi(A : \square).\ \mathbb{E} \to A.$$
This reflects the intended meaning of exceptions: whenever a program fails, it can raise an exception and still be well-typed.
- They assume that $\mathbb{E}$ features an inhabitant $e : \mathbb{E}$;
- The former means that any type A is now inhabited by $\texttt{raise}\ e$; the resulting theory is thus inconsistent. To deal with this issue, they restrict the use of exceptions through a layer of *parametricity*, a proof technique we will not explain here. Let us simply say that the resulting *exceptionally parametric* extension of CIC is consistent;
- Given a function
$$f : \neg A \to \Sigma(n : \mathbb{N}).\ B\,n,$$
they apply it to
$$\texttt{fail} : \neg A \coloneqq \lambda\_.\texttt{raise}\ \bot\ e,$$
recovering
$$f\ \texttt{fail} : \Sigma(n : \mathbb{N}).\ B\,n.$$
There are then two options: either f tries to apply `fail` to an argument $a : A$, or it does not.
If it does, then `fail` raises an exception and there is work to be done to justify why this exception is allowed to happen despite parametricity. We will keep this part under the rug and refer the interested reader to the original paper.
The other case is when f does not use `fail`. This means that the proof of $\neg A$ was unnecessary to build a term of type
$$\Sigma(n : \mathbb{N}).\ B\,n.$$

Knowing whether f applies `fail` or not is decidable.

Then it is fairly easy to transform this into a term of type

$$\Sigma(n : \mathbb{N}).\ \neg A \to B\ n$$

and conclude.

A broader intuition of what is happening there is the following. We need to extract a global natural number from a function

$$f : \neg A \to \Sigma(n : \mathbb{N}).\ B\ n.$$

Unfortunately, as long as f stays a query-answer, black-box function, giving us no access to its internal computation, there is no way for us to achieve this feat. Exceptions plays the role of a dynamite stick, blowing the black box open and letting us search its ruins for hints of f's internal functioning. This is rather primitive, though, and some finer-tuned effects can give us more information without blowing everything up in smoke. *Continuity* is of that kind, as we will argue in this thesis.

1.5. Syntactic models

Since *Gödel*'s incompleteness theorem [64], we know that an expressive enough theory cannot prove its own consistency. Said consistency thus has to be proven in another, stronger theory. Of course this other theory cannot prove its own consistency either, so we need another one to do so. We could forge ahead and build an infinite tower of stronger and stronger theories

$$\mathcal{T}_{n+1} \coloneqq \mathcal{T}_n + (\mathcal{T}_n \texttt{ is consistent})$$

but this would be an idol with feet of clay: if $\mathcal{T}_0$ is consistent then all of them are, but if $\mathcal{T}_0$ is inconsistent then all of them are, too!

An inconsistent theory could in fact prove its own consistency, since everything is true in an inconsistent world. This is unfortunately not very helpful to us.

[64]: Gödel (1931), "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I"

Welcome to the metaverse At the end of the day, we have to choose a theory whose consistency we do not challenge, and prove every other theory consistent using this first one. In this thesis, we will call *object theory* or *source theory* the theory we are studying, and trying to justify; we will call *target theory* the glorified, unchallenged theory in which the proof is written. Finally, we will call *meta-theory* the ambient, ethereal theory we use to think about the proof, and discuss its validity. Formally, this means that from the meta-theory, we are able to ascertain that our target theory justifies our source theory.

Historically, consistency of type theories have been proven using some variant of set theory as a target, and classical logic in the meta. Recent years, however, have seen a rise in endeavours to justify type theory with another type theory, using another version of type theory (often implemented in a proof assistant) as meta-theory [3, 7, 31, 86, 120, 121]. In this thesis, we will go with the flow and mostly use CIC as a target and Coq's version of CIC as our meta-theory.

Given a target type theory $\mathcal{T}$ and an object type theory $\mathcal{S}$, the way to go is then to build an *interpretation* of $\mathcal{S}$ in $\mathcal{T}$.

[3]: Abel et al. (2017), "Decidability of Conversion for Type Theory in Type Theory"
[7]: Altenkirch et al. (2016), "Type theory in type theory using quotient inductive types"
[31]: Chapman (2009), "Type Theory Should Eat Itself"
[86]: Kaposi et al. (2019), "Shallow Embedding of Type Theory is Morally Correct"
[120]: Pujet et al. (2022), "Observational Equality: Now for Good"
[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

Definition 1.5.1: Interpretation
An *interpretation* of a type theory $\mathcal{S}$ into another type theory $\mathcal{T}$ consists of:

- for any context Γ of $\mathcal{S}$, an object $[\![\Gamma]\!]$ in $\mathcal{T}$ called the interpretation of Γ;
- for any type A of $\mathcal{S}$, an object $[\![A]\!]$ in $\mathcal{T}$ called the interpretation of A ;
- for any term t of $\mathcal{S}$, an object $[t]$ called the interpretation of t;
- a predicate P_C in $\mathcal{T}$ interpreting well-formation of contexts;
- a predicate P_T in $\mathcal{T}$ interpreting well-formation of types;
- a relation $\mathsf{R}_{\vdash}$ in $\mathcal{T}$ interpreting typing;
- a relation $\mathsf{R}_{\equiv}$ in $\mathcal{T}$ interpreting conversion.

Using these notations, we can now describe *soundness* of an interpretation.

Definition 1.5.2: Soundness
An interpretation of $\mathcal{S}$ in $\mathcal{T}$ is *sound* when every rule of $\mathcal{S}$ is correctly interpreted in $\mathcal{T}$. This means for instance that given a typing judgment in the object theory

$$\Gamma \vdash^{\mathcal{S}} t : A,$$

given

$$[\![\Gamma]\!], [t] \text{ and } [\![A]\!] \qquad \text{the interpretations of} \qquad \Gamma, t \text{ and } A$$

and $\mathsf{R}_{\vdash}$ the relation interpreting typing, we have

$$\vdash^{\mathcal{T}} \mathsf{R}_{\vdash}\ [\![\Gamma]\!]\ [t]\ [\![A]\!]$$

in the target theory.

A sound interpretation of $\mathcal{S}$ in $\mathcal{T}$ is also called a *model* of $\mathcal{S}$ in $\mathcal{T}$.

The converse of soundness is called *completeness*, and is defined as follows.

Definition 1.5.3: Completeness
An interpretation of $\mathcal{S}$ in $\mathcal{T}$ is *complete* when every interpreted judgment in $\mathcal{T}$ corresponds to a correct judgment in $\mathcal{S}$. This means for instance that given

$$\Gamma, t \text{ and } A \qquad \text{interpreted as} \qquad [\![\Gamma]\!], [t] \text{ and } [\![A]\!],$$

given $\mathsf{R}_{\vdash}$ the relation interpreting typing, if

$$\vdash^{\mathcal{T}} \mathsf{R}_{\vdash}\ [\![\Gamma]\!]\ [t]\ [\![A]\!]$$

in the target theory then

$$\Gamma \vdash^{\mathcal{S}} t : A$$

in the source.

Never listen to the classics We emphasize that the *model* lives in $\mathscr{T}$ but the *proof that it is sound and complete* lives in the meta-theory. This means that the choice of the meta-theory has an impact on whether an interpretation *is indeed* a model: there could be cases where the use of double-negation elimination is necessary for an interpretation to be proven sound, which would not be available in CIC. As an example, if we take:

- as source theory $\mathscr{S}$, an extension of System T featuring an empty type $\bot$, non-dependent sum types and higher-order quantification over types. The expert reader may recognize System F enhanced with sums, $\mathbb{N}$ and $\bot$ in this picture;
- as target theory $\mathscr{T}$, a very weak theory featuring only $\top$ and $\bot$;
- as interpretation function for types, the function that takes a type A and sends it to $\top$ if A is inhabited in the empty context, and $\bot$ if it is not. This is doable only by assuming that any type is either inhabited or not, a reasoning called *excluded middle*, displayed in Table 1.28 and equivalent to DNE;
- as interpretation function for terms, the constant function that sends a term t of $\mathscr{S}$ is to $\star : \top$ in $\mathscr{T}$;
- as relation $\mathsf{R}_{\vdash}$ interpreting typing of $\mathscr{S}$ in $\mathscr{T}$, the typing relation of $\mathscr{T}$, with only one rule: $\star : \top$;
- as relation $\mathsf{R}_{\equiv}$ interpreting relation of $\mathscr{S}$ in $\mathscr{T}$, the conversion relation of $\mathscr{T}$ with only one rule: $\star \equiv \star : \top$.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee \neg A}$$

Table 1.28.: Excluded middle

Finally, if

$$\cdot \vdash^{\mathscr{S}} t : A$$

then A is inhabited, hence

$$[\![A]\!] \coloneqq \top.$$

As $[t] \coloneqq \star$, we indeed have

$$\vdash^{\mathscr{T}} [t] : [\![A]\!],$$

hence

$$\mathsf{R}_{\vdash}\ [t]\ [\![A]\!].$$

Our interpretation is thus *sound for typing*. In fact, as empty types are sent to $\bot$, our interpretation exactly captures typing in $\mathscr{S}$, hence it is also *complete for typing*. It is also sound but not complete for conversion, as all terms are convertible in the model.

Of course, this model is not very informative on $\mathscr{S}$. We basically asked God if every type was or was not inhabited, and we know the answer is *somewhere* but we do not have access to it. In this thesis, we try to live in the weakest possible meta-theory, so that this scenario does not happen.

The Top Model In fact, a basic information we would like to get from a model is *consistency of the source theory*. Indeed, if for any t,

$$\mathsf{R}_{\vdash}\ [\![\cdot]\!]\ [t]\ [\![\bot]\!]$$

is empty, then there is no proof

$$\cdot \vdash^{\mathcal{S}} t : \bot$$

and $\mathcal{S}$ is consistent. This is typically what was lacking in our previous example, as we used classical logic as an escape route to evade the question of consistency of the source theory.

A more rigorous way of phrasing it is to say that $\mathcal{S}$ is consistent *assuming both $\mathcal{T}$ and the meta-theory are consistent*. Indeed, assuming $\bot$ in $\mathcal{T}$ or in the meta-theory we can recover a proof of

$$\mathsf{R}_{\vdash}\ [\![\Gamma]\!]\ [t]\ [\![\bot]\!]$$

for any Γ, t from the source.

However, classical logic in the meta-theory is not the only way to build models that do not provide information on consistency of the source. An example of a sound interpretation where this is the case is the degenerate *top model*, where every object is interpreted by $\top$ and every relation (typing and conversion) is interpreted by the *full relation*, *i.e.*

$$\mathsf{R}_{\vdash}\ [\![\Gamma]\!]\ [t]\ [\![A]\!] \qquad \text{for any} \qquad \Gamma, t \text{ and } A.$$

What we call *top model* is mostly known as *terminal model* in the literature.

The top model is trivially *sound*, but unless $\mathcal{S}$ is also degenerate it will not be complete. In particular, any term is bound to $\bot$ by $\mathsf{R}_{\vdash}$ so we cannot hope to recover consistency of $\mathcal{S}$ from the top model. In the rest of the thesis we will only consider models where consistency of the target theory entails consistency of the source theory.

Syntax haven Most of the models in the literature are *semantics* models. They are so called because they make sense of the syntax of the source theory by interpreting it as objects of another, sometimes quite different, theory. The target theory is often *ZFC* set theory as in *Werner* [142], or category theory as it is the case in *categories with families* [49] or *comprehension categories* [83]. *Hoffman* [75] gives a general overview of such historical models. More recently, *Winterhalter and Bauer* [145] presented a semantic model of CIC with *cardinals* to derive independence of some principles. Notably, their model validates

$$\mathbb{N} \to \mathbb{N} = \mathbb{N} \to \mathbb{B},$$

meaning that it is not possible in *CIC* to prove that they are apart. Building such a model can be a long and tedious task, as one has to make sure that every rule of the source theory is correctly interpreted in the target theory. Moreover, it often involves many proofs about conversion, so much that in the end it can be hard to get a computational understanding of what is happening. This is problematic: if we want to stay true to our *proofs as programs* slogan, we need to be able to tell how our model computes, which is difficult in semantic models where proofs obfuscate computational content, even more so when classical logic is at the wheel. To this end, in Chapter 3 we will make use of a specific kind of model: *program translations*, as presented by *Boulier et al [25, 126]*. Such translations form a subset of *syntactic models* (so-called because they revolve around syntax) and are defined as follows.

[142]: Werner (1997), "Sets in Types, Types in Sets"

[49]: Dybjer (1995), "Internal Type Theory"

[83]: Jacobs (1993), "Comprehension Categories and the Semantics of Type Dependency"

[75]: Hofmann (1997), *Extensional constructs in intensional type theory*

[145]: Winterhalter (2020), "Formalisation and meta-theory of type theory"

[25]: Boulier (2018), "Extending type theory with syntactic models. (Etendre la théorie des types à l'aide de modèles syntaxiques)"

[126]: Simon Boulier et al. (2017), "The next 700 syntactical models of type theory"

Definition 1.5.4: Program translations
A *program translation* of a type theory $\mathcal{S}$ into another type theory $\mathcal{T}$ is an interpretation of $\mathcal{S}$ into $\mathcal{T}$ defined by induction on the syntax. In summary:

- a *context* Γ of $\mathcal{S}$ is translated as a *context* $[\![\Gamma]\!]$ of $\mathcal{T}$;
- a *type* A of $\mathcal{S}$ is translated as a *type* $[\![A]\!]$ of $\mathcal{T}$;
- a *term* t of $\mathcal{S}$ is translated as a *term* $[t]$ of $\mathcal{T}$;
- *typing* in $\mathcal{S}$ is translated as *typing* in $\mathcal{T}$;
- *conversion* in $\mathcal{S}$ is translated as *conversion* in $\mathcal{T}$;
- if the source and the target theory feature universes, an explicit function
$$\texttt{El} : [\![\square]\!] \to \square$$
is provided to turn the translation $[A]$ of a term $A : \square$ into the translation
$$[\![A]\!] \coloneqq \texttt{El}\ [A]$$
of the corresponding type A.

It is easy to give computational content to program translations: we are translating the syntax of a programming language into the syntax of another, in such a way that computation in the latter simulates computation in the former. This is compilation!

As program translations are defined on raw syntax, conversion is often considered untyped, which allows for less intricate proofs. We will do so in Chapter 3. When conversion is untyped, the proof that a program translation is sound will always follow the same steps. First comes *substitution soundness*:

Definition 1.5.5: Substitution soundness
A program translation enjoys *substitution soundness* when for any terms t and u of the source theory, the following holds in the target theory:
$$[t\{x \coloneqq u\}] \equiv_{\mathcal{T}} [t]\{x \coloneqq [u]\}.$$

Once substitution is covered, general soundness follows, divided into *typing* and *conversion soundness*. Since conversion is untyped, we can prove first conversion soundness, then typing soundness.

Definition 1.5.6: Conversion soundness
A program translation enjoys *conversion soundness* when for any terms t and u of the source theory, the following holds:
$$\text{if} \quad t \equiv_{\mathcal{S}} u \quad \textit{then} \quad [t] \equiv_{\mathcal{T}} [u],$$
where $\equiv_{\mathcal{S}}$ denotes conversion in the source theory and $\equiv_{\mathcal{T}}$ conversion in the target.

Definition 1.5.7: Typing soundness
A program translation enjoys *typing soundness* when for any context Γ, any term t and any type A of the source theory, the following holds:

$$\text{if} \quad \Gamma \vdash^{\mathcal{S}} t : A \quad \textit{then} \quad [\![\Gamma]\!] \vdash^{\mathcal{T}} [t] : [\![A]\!],$$

where $\vdash^{\mathcal{S}}$ denotes typing in the source theory and $\vdash^{\mathcal{T}}$ typing in the target.

Finally, one last ingredient is needed to derive consistency of the source theory from consistency of the target.

Definition 1.5.8: Consistency preservation
A program translation enjoys *consistency preservation* when there is a map

$$\cdot \vdash^{\mathcal{T}} [\![\bot_{\mathcal{S}}]\!] \to \bot_{\mathcal{T}}$$

where $\bot_{\mathcal{S}}$ is the empty type in the source theory and $\bot_{\mathcal{T}}$ the empty type in the target theory.

It is easy to see how consistency preservation indeed preserves consistency (as its name suggests): the only way to build an inconsistent model of $\mathcal{S}$ is when $\mathcal{T}$ is itself inconsistent.

Changing times All of this is well and good, but do program translations really exist? Their requirements are quite harsh, and this concept would not be very useful if the only program translation available was the identity. A sufficient criterion to turn a syntactic model into a program translation is given by *Altenkirch et al* [10], but an existence is best proven by providing an example, so let us introduce the *times-bool translation*, introduced by *Boulier et al* [126].

[10]: Altenkirch et al. (2019), "Setoid type theory - a syntactic translation"

[126]: Simon Boulier et al. (2017), "The next 700 syntactical models of type theory"

$$\begin{array}{lcl}
[\square_i] & := & \square_i \\
[\Pi x : A.\, B] & := & (\Pi x : [\![A]\!].\, [\![B]\!]) \times \mathbb{B} \\
[\Sigma x : A.\, B] & := & \Sigma x : [\![A]\!].\, [\![B]\!] \\
[\![A]\!] & := & [A] \\
[x] & := & x \\
[\lambda x : A.\, t] & := & (\lambda x : [\![A]\!].\, [t], \mathrm{true}) \\
[t\ u] & := & [t].\pi_1\ [u] \\
[(t, u)] & := & ([t], [u]) \\
[t.\pi_1] & := & [t].\pi_1 \\
[t.\pi_2] & := & [t].\pi_2
\end{array}$$

$$\begin{array}{lcl}
[\![\cdot]\!] & := & \cdot \\
[\![\Gamma,\ x : A]\!] & := & [\![\Gamma]\!], x : [\![A]\!]
\end{array}$$

Figure 1.10.: Times-bool translation of CC_ω

As we already explained, models are a tool to study type theories and their possible extensions. Here, the concept under the microscope is *function extensionality*, a principle we briefly met in Section 1.3.

Its precise wording is the following:

$$\begin{aligned}\mathsf{funext} \quad &:= \quad \Pi(A : \square)(B : A \to \square)(f\ g : \Pi x : A.\ B\ x).\\ &\qquad (\Pi x : A. f\ x = g\ x) \to f = g\end{aligned}$$

The times-bool translation is a model *negating* funext in a concise and elegant manner: we simply add a label (true or false) to every function; then any function f exists in two versions,

$$(f, \mathsf{true}) \qquad \text{and} \qquad (f, \mathsf{false}).$$

These two versions compute in the same way, yet they are different because of the label we put on them: funext is negated.

The translation is formally displayed in Figure 1.10, with the syntax for the source theory on the left and its translation on the right. We implicitly extend this translation to any term t by recursion on the structure of t.

Recall that $A \times B$ is a notation for

$$\Sigma x : A.\ B$$

when B does not depend on $x : A$, the same way $A \to B$ is used to denote

$$\Pi x : A.\ B$$

when B does not depend on $x : A$.

In this translation:

- The source theory $\mathcal{S}$ is a weak version of CC_ω, where η-conversion was stripped off. We recall this missing rule in Table 1.29;
- The target theory $\mathcal{T}$ is the same version of CC_ω, extended with booleans;
- As *Boulier et al* formalized the translation in Coq, the meta-theory is Coq's CIC implementation.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash f : \Pi x : A.B \qquad \Gamma \vdash g : \Pi x : A.B \qquad \Gamma, x : A \vdash f\ x \equiv g\ x : B}{\Gamma \vdash f \equiv g : \Pi x : A.B}$$

Table 1.29.: η-conversion in CC_ω

Note that we can extend the interpretation to inductive types: most of them are simply translated by themselves, giving us a translation from CIC without η-conversion into itself. We give the translations of $\mathbb{N}$ and eq as examples and refer the interested reader to the original paper for more information.

$$\begin{aligned}
[\mathbb{N}] &:= \mathbb{N}\\
[\mathsf{O}] &:= \mathsf{O}\\
[\mathsf{S}] &:= \mathsf{S}\\
[\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n] &:= \mathbb{N}_{\mathbf{ind}}\ [P]\ [t_{\mathsf{O}}]\ [t_{\mathsf{S}}]\ [n]\\
\\
[\mathsf{eq}\ A\ t\ u] &:= \mathsf{eq}\ [\![A]\!]\ [t]\ [u]\\
[\mathsf{refl}\ A\ t] &:= \mathsf{refl}\ [\![A]\!]\ [t]\\
[\mathsf{eq}_{\mathbf{ind}}\ A\ a\ P\ t_{\mathsf{refl}}\ a'\ e] &:= \mathsf{eq}_{\mathbf{ind}}\ [\![A]\!]\ [a]\ [P]\ [t_{\mathsf{refl}}]\ [a']\ [e]
\end{aligned}$$

Figure 1.11.: Times-bool translation of some inductive types

The times-bool translation is a model of CIC without η-conversion.

Lemma 1.5.1:
The times-bool translation validates substitution soundness.

Proof. By induction on the term at hand. ∎

Proposition 1.5.2: Soundness
The times-bool translation validates computational soundness as well as typing soundness.

Proof. Computational soundness follows from induction on the conversion derivation, typing soundness from induction on the typing derivation. ∎

Now comes the interesting part.

Theorem 1.5.3: Negation of functional extensionality
In the target theory, there is a term of type

$$[\![\mathsf{funext} \to \bot]\!],$$

thereby negating functional extensionality.

Proof. Assuming $[\![\mathsf{funext}]\!]$, we will in fact prove

$$\mathsf{true} = \mathsf{false},$$

from which falsity is easily retrieved using $\mathsf{eq}_{\mathbf{ind}}$. To do so, we set

$$\begin{array}{lcl} A : \square & \coloneqq & \mathbb{N} \\ B : A \to \square & \coloneqq & \lambda\_ : \mathbb{N}.\, \mathbb{N} \end{array}$$

in the source theory. The translated dependent product then becomes

$$[\![\Pi x : A.\, B\, x]\!] \coloneqq (\mathbb{N} \to \mathbb{N}) \times \mathbb{B}.$$

We then provide two functions:

$$\begin{array}{lcl} f : (\mathbb{N} \to \mathbb{N}) \times \mathbb{B} & \coloneqq & (\lambda x : \mathbb{N}.\, x, \mathsf{true}) \\ g : (\mathbb{N} \to \mathbb{N}) \times \mathbb{B} & \coloneqq & (\lambda x : \mathbb{N}.\, x, \mathsf{false}) \end{array}$$

Then, since application in the model only considers the first projection of functions, we can prove

$$\Pi(x : \mathbb{N}).\, f\, x = g\, x.$$

Finally, applying funext we get

$$(\lambda x : \mathbb{N}.\, x, \mathsf{true}) = (\lambda x : \mathbb{N}.\, x, \mathsf{false}),$$

from which we deduce

$$\mathsf{true} = \mathsf{false}$$

then the desired falsity. ∎

For a more detailed version of the proof, we refer the interested reader to *Boulier's* PhD thesis [25] once again, where it is formalized in Coq.

[25]: Boulier (2018), "Extending type theory with syntactic models. (Etendre la théorie des types à l'aide de modèles syntaxiques)"

Now that we have a term

$$H_{\mathcal{T}} : [\![\mathsf{funext} \to \bot]\!]$$

in the target theory, we can assume a term

$$H_{\mathcal{S}} : \mathsf{funext} \to \bot$$

in the source theory, simply defining

$$[H_{\mathcal{S}}] \coloneqq H_{\mathcal{T}}.$$

This is the usual point of models: whenever we inhabit the interpretation of a type, we can assume that this type is inhabited in the source theory, and the model guarantees that the resulting extended source theory is consistent. However, consistency is no panacea: as we saw in Section 1.4, double-negation elimination is consistent with CIC, yet adding it as an inert axiom with no conversion rule destroys a lot of good properties of our system.

Fortunately, a model built through a program translation not only provides consistency, but also the computational content of this additional axiom: we simply have to look at the target theory to see what is happening.

Note that η-conversion is also negated by the times-bool model. Indeed, taking the function

$$(\lambda x : \mathbb{N}.\ x, \mathsf{false}),$$

we can reflect it as a special term

$$g : \mathbb{N} \to \mathbb{N}$$

in the source theory. Then we have

$$[g] \equiv (\lambda x : \mathbb{N}.\ x, \mathsf{false}),$$

yet unfolding the definition of λ-abstraction in the model we get

$$[\lambda x : \mathbb{N}.\ g\ x] \equiv (\lambda x : \mathbb{N}.\ x, \mathsf{true})$$

which is not convertible to the former.

It is nonetheless possible to build program translations of CIC negating funext while retaining η-conversion. This is the case of *Pédrot and Tabareau's* exceptional model [122], which we already presented in Section 1.4.

[122]: Pédrot et al. (2018), "Failure is Not an Option An Exceptional Type Theory"

The logical history tour Here comes the end of our guided visit in dependent type theory. As we explained in the beginnning of the Chapter, this thesis is devoted to the study of a particular kind of effect, *continuity*, encoded by the *dialogue monad*, in the context of dependent type theory.

Chapter 2 focuses on dialogue trees as an inductive representation of *continuous functionals*. This way of envisioning continuity is one among many others, hence we survey different definitions of continuity that exist in the literature, and describe their interactions. In particular, we highlight the closeness between the dialogue monad and sheafification.

Chapter 3 displays a first attempt at mingling continuity with dependent type theory. In this Chapter, we extend a proof by *Escardó* [51] that every functional definable in System T is continuous. As the proof is fairly technical, we first go through *Escardó*'s proof in Section 3.1, albeit rephrasing it as a program translation. We finally enhance the proof to BTT in Section 3.2.

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

Unfortunately, this proof technique does not scale well to MLTT, so we change tack to try and gain more control over computation by building a *normalization model* for an extension of MLTT. This extension, dubbed ꟻTT (read "Digamma TT" or "Split TT") was already described by *Coquand and Jaber* [42, 43] and features a specific function

$$\mathfrak{f} : \mathbb{N} \to \mathbb{B}$$

whose calls are recorded in *forcing conditions*. The idea is to recover a modified *canonicity* theorem for ꟻTT, essentially stating that every term $t : \mathbb{N}$ potentially mentioning $\mathfrak{f}$ is continuous. This proof is however not fully formalized, though we hope to provide a certified proof in the coming months. We discuss this endeavour in Chapter 4.

[42]: Coquand et al. (2012), "A Computational Interpretation of Forcing in Type Theory"
[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"

2. A world made of trees

Children continuously ask questions It is a universal observation that children are prone to ask their parents many questions, and scenes are quite common where no matter what answer parents can come up with, their child will ask another "Why" in what looks like an endless streams of queries. However, this is just an impression, and were we to watch parents of extreme patience, we would see the questions' flow dry up after a while, be it through tiredness, boredom or hunger. Mathematically speaking, the property of a function (or, in our case, a child) that only asks a finite number of queries before reaching the end of its computation is called *continuity*.

Continuity is a concept that has been in the toolbox of mathematicians for centuries, and the principle that *every function is continuous* is present in the constructive realm since the days of *Brouwer* [27, 28], as a principle close to *bar induction*, later studied in detail by *Howard and Kreisel* [78] , *Tait* [133] or *Dummett* [48]. In this setting, it is common to consider continuity of functionals of the form

$$f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \qquad \text{or} \qquad f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N},$$

where $\mathbb{N} \to \mathbb{B}$ is called the *Cantor space* and $\mathbb{N} \to \mathbb{N}$ is called the *Baire space*, and continuity on such spaces has been largely studied [20–22, 55, 138, 139] from a mathematical point of view.

As we already hinted at, on a intuitive level *continuity* of a functional

$$f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

describes the fact that f is a computation that returns a value in a finite amount of time. As such, given an argument

$$\alpha : \mathbb{N} \to \mathbb{B},$$

f can only ask α a finite number of queries before returning a value. In usual mathematics, this is expressed by saying that if another argument

$$\beta : \mathbb{N} \to \mathbb{B}$$

give the same answers as α to those queries, then f cannot distinguish them and $f\,\alpha = f\,\beta$. However, as we will see later on, more intensional definitions of continuity require that we get access to the *trace of calls* that f does to its argument, an interpretation close to the notion of computation advocated by *Kleene* [89, 90].

In this Chapter, we will suppose given two types

$$\vdash^{\mathscr{T}} \mathbf{I} : \square_0 \qquad \text{and} \qquad \vdash^{\mathscr{T}} \mathbf{O} : \mathbf{I} \to \square_0,$$

[27]: Brouwer (1981), *Brouwer's Cambridge lectures on intuitionism*
[28]: Brouwer (1925), "Zur Begründung der intuitionistischen Mathematik. I."
[78]: Howard et al. (1966), "Transfinite Induction and Bar Induction of Types Zero and One, and the Role of Continuity in Intuitionistic Analysis"
[133]: Tait (1968), "Constructive reasoning"
[48]: Dummett (2000), *Elements of intuitionism*
[20]: Berger (2012), "Aligning the weak König lemma, the uniform continuity theorem, and Brouwer's fan theorem"
[21]: Berger (2005), "The Fan Theorem and Uniform Continuity"
[22]: Berger (2006), "The Logical Strength of the Uniform Continuity Theorem"
[55]: Fujiwara et al. (2021), "Characterising Brouwer's continuity by bar recursion on moduli of continuity"
[138]: Troelstra (1988), "Constructivism in mathematics"
[139]: Troelstra (1973), *Metamathematical investigation of intuitionistic arithmetic and analysis*

[89]: Kleene (1978), "Recursive Functionals and Quantifiers of Finite Types Revisited I"
[90]: Kleene (1959), "Recursive functionals and quantifiers of finite types. I"

and we will study continuity of functionals on the form

$$f : (\Pi i : \mathbf{I}.\, \mathbf{O}\, i) \to A.$$

On the computational side, the type $\mathbf{I}$ is to be understood as a type of input or questions to a black-box, called an oracle in reference to the impenetrable nature of its answers. Dually, $\mathbf{O}$ is the type of output or answers from the oracle. Since $\mathbf{O}$ depends on $\mathbf{I}$, we can encode pretty much arbitrary interactions. Finally, we define the type of oracles as

$$\mathbf{Q} := \Pi(i : \mathbf{I}).\, \mathbf{O}\, i.$$

Carrying our metaphor further, the child in our example is the functional

$$f : \mathbf{Q} \to A$$

and oracles are his parents answering his questions. A reader more inclined towards computer science could also consider that $\mathbf{I}$ and $\mathbf{O}$ describe an interface for system calls, and $\mathbf{Q}$ is the type of operating systems implementing these calls.

On the proof-theoretic side, an analog of continuity can be found in the form of *compactness theorems* [69, 102], which state that even in a theory with infinite axiom schemes, a concrete proof of a theorem will only use a finite amount of those.

[69]: Gödel (1929), *Über die Vollständigkeit des Logikkalküls*
[102]: Malcev (1936), "Untersuchungen aus dem Gebiete der mathematischen Logik"

Please continue We emphasize that in the rest of this thesis we will study continuity both as a property of functions and as an axiom stating that *all functions of a certain type are continuous*. Since the general purpose of this thesis is to help bring classical and constructive logic closer, let us already mention that

$$(\mathbf{I} \to \bot) \to \bot$$

is also of the form

$$\mathbf{Q} \to A,$$

hence we hope that studying continuity can help give computational content to semi-classical principles, if not full-blown DNE. In particular, as we will see in Section 2.4, assuming that every function

$$f : (\mathbf{I} \to \bot) \to \bot$$

is continuous entails double-negation elimination.

This last argument could be surprising, as the axiom of continuity of all functions is widely known to be an *anti-classical* principle. For instance, *Troelstra* [137] showed that choice, continuity of all functions of the form

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

and extensionality principles are incompatible in intuitionistic finite-type arithmetic, improving a previous result by *Kreisel* [93]. However, it may be possible to unite choice and continuity in a more intensional setting, where extensionality principles are dismissed. Indeed, as we

[137]: Troelstra (1977), "A note on non-extensional operations in connection with continuity and recursiveness"

[93]: Kreisel (1962), "On Weak Completeness of Intuitionistic Predicate Logic"

have seen in the previous Chapter in Section 1.5, effectful computation often comes at the cost of extensionality principles such as funext and in this thesis we will precisely consider an encoding of continuity through some kind of effect. As such, this thesis is quite close to the line of work of *Cohen*, *Rahli*, *Bickford et al* [24, 35–37, 123, 124] who retrieve continuity results in some variants of type theory using effects, like stateful computations, references or named exceptions.

[24]: Bickford et al. (2018), "Computability Beyond Church-Turing via Choice Sequences"
[35]: Cohen et al. (2022), "Constructing Unprejudiced Extensional Type Theories with Choices via Modalities"
[36]: Cohen et al. (2023), "Realizing Continuity Using Stateful Computations"
[37]: Cohen et al. (2023), "Inductive Continuity via Brouwer Trees"
[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"
[124]: Rahli et al. (2019), "Bar Induction is Compatible with Constructive Type Theory"

Overview In our case, the effect under scrutiny will be the *dialogue monad* $\mathfrak{D}$, an inductive operator taking a type A and returning the type of inductive trees with leaves in A. We describe $\mathfrak{D}$ in detail in Section 2.1.

In Section 2.2, we present some definitions of continuity that, although equated in classical logic, remain distinct in type theory. We present them in increasing order of logical power, each new definition implying the former one.

In Section 2.3, we investigate which logical principles (such as function extensionality or bar induction) are needed for the implications described in Section 2.2 to become equivalences.

In Section 2.4 we explain how, in a Curry-Howard perspective, continuity can be seen as the computational counterpart of semi-classical principles such as the *double-negation-shift*.

Finally, in Section 2.5, we rephrase unpublished notes by Pédrot [113, 114] about sheaves, and highlight the strong similarity between sheafification and the dialogue monad.

[113]: Pédrot (2021), "Debunking Sheaves"
[114]: Pédrot (2023), "Pursuing Shtuck"

Most definitions and results of this Chapter are formalized in Coq. When it is the case, we will provide hyperlinks to the relevant part of the code in the margins.

2.1. Talking trees

As we hinted at in the introduction, for the remaining of this Chapter we will consider two types

$$\vdash^{\mathcal{T}} \mathbf{I} : \square_0 \qquad \text{and} \qquad \vdash^{\mathcal{T}} \mathbf{O} : \mathbf{I} \to \square_0.$$

We set them in the lowest universe level for simplicity, but all of the constructions to come can handle an arbitrary base level by bumping them by an appropriate amount.

2.1.1. Dialogue is the key

We start by defining the Dialogue operatore $\mathfrak{D}$, which will be the main character of this Chapter.

Definition 2.1.1: Dialogue operator
We consider an operator $\mathfrak{D} : \square \to \square$, which given a type $A : \square$, associates the type of I-labelled, O-branching, well-founded trees, with leaves labelled in A. Each inner node is labelled with a certain

$$i : \mathrm{I} \qquad \text{and has} \qquad \mathrm{O}\, i$$

children. In $\mathscr{T}$, this amounts to the following inductive definition:

$$\begin{array}{l}\texttt{Inductive}\ \mathfrak{D}\ (A : \square_i) : \square_i := \\ \mid \eta : A \to \mathfrak{D}\ A \\ \mid \beta : \Pi(i : \mathrm{I}).\, (\mathrm{O}\, i \to \mathfrak{D}\ A) \to \mathfrak{D}\ A.\end{array}$$

The dialogue operator is formally defined here.

As an example, let us set

$$\begin{array}{l}\mathrm{I} := \mathbb{B} \\ \mathrm{O} := \lambda i : \mathrm{I}.\ \texttt{if}\ i\ \texttt{then}\ \top\ \texttt{else}\ \mathbb{N}\end{array}$$

where $\top$ is the always inhabited type with only one element $\star : \top$. Table 2.1 displays a typical dialogue tree in that case.

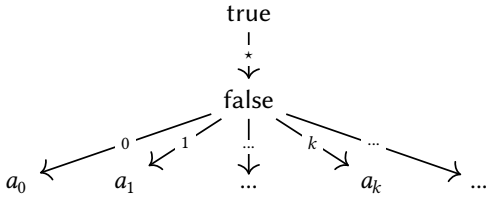


Table 2.1.: Example of dialogue tree.

2.1.2. Monadic labs

In an extensional enough setting, the $\mathfrak{D}$ type former turns out to be a monad: the

$$\eta : \Pi A : \square.\, A \to \mathfrak{D}\ A$$

natural transformation is already part of the definition, and we can recursively define a `bind` function:

$$\begin{array}{lcl}\texttt{bind} & : & \Pi\{A\ B : \square\}(f : A \to \mathfrak{D}\ B)(d : \mathfrak{D}\ A).\, \mathfrak{D}\ B \\ \texttt{bind}\ f\ (\eta\ x) & := & f\ x \\ \texttt{bind}\ f\ (\beta\ i\ k) & := & \beta\ i\ (\lambda(o : \mathrm{O}\ i).\, \texttt{bind}\ f\ (k\ o))\end{array}$$

Lemma 2.1.1:
Assuming function extensionality, $(\mathfrak{D}, \eta, \texttt{bind})$ is a monad.

Function extensionality is needed to prove commutation laws between η and `bind`. Without funext, we only end up with what *Pédrot and Tabareau* call a *proto-monad* [121].

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

Freedom of speech However, even by categorical standards, $\mathfrak{D}$ is a very particular monad.

Definition 2.1.2: Free monad
A free monad in CIC is a parameterized inductive type $\mathscr{M} : \square \to \square$ with a dedicated constructor

$$\eta : \Pi(A : \square).\, A \to \mathscr{M}\ A$$

and a finite set of constructors

$$\mathrm{c}_i : \Pi(A : \square).\, \Phi_i\ (\mathscr{M}\ A) \to \mathscr{M}\ A$$

where

$$\Phi_i : \square \to \square$$

is a type former syntactically strictly positive in its argument.

Note that the formal definition of free monad from category theory requires a forgetful functor to specify against what the monad would be free. The closest thing to our definition would be a free monad relatively to pointed functors, but even there our definition is stricter. A free monad can be thought of as a way to extend a type with unspecified, inert side-effects, a trivial form of algebraic effects [5, 18, 118]. In the field of algebraic effects, a free monad as defined here is also known as a monad induced by a *signature* [17]; the c_i constructors are called *operation symbols* and the Φ_i type formers are dubbed *arities* (in our setting, however, arities are typed as operations may mention other types than the carrier of the algebraic theory). Since we have neither QITs [9] nor HITs [140] in CIC, we cannot enforce equations on these effects to get an *algebraic theory* but we can still go a long way.

[5]: Ahman (2018), "Handling fibred algebraic effects"
[18]: Bauer et al. (2015), "Programming with algebraic effects and handlers"
[118]: Plotkin et al. (2013), "Handling Algebraic Effects"
[17]: Bauer (2018), "What is algebraic about algebraic effects and handlers?"
[9]: Altenkirch et al. (2018), "Quotient Inductive-Inductive Types"
[140]: Univalent Foundations Program (2013), *Homotopy Type Theory: Univalent Foundations of Mathematics*

Free monads in CIC enjoy a lot of interesting properties. As the name implies, they are indeed monads, once again up to function extensionality. Again, the η function is given by definition, and `bind` can be defined functorially by induction similarly to the $\mathfrak{D}$ case. Furthermore, algebras of a free monad can be described in an intensionally-friendly way.

Definition 2.1.3: Intensional algebras of a free monad
Given $\mathscr{M}$ as above, the type of *intensional $\mathscr{M}$-algebras* is the record type

$$\square^{\mathscr{M}} := \{A : \square; \ldots; \mathrm{p}_i : \Phi_i\ A \to A; \ldots\}$$

In the field of algebraic effects, intensional algebras of a free monad are called *models* of an algebraic theory. The following result, stating that models of a signature are algebra of the induced monad, is well-known, and for instance presented in *Bauer* [17].

[17]: Bauer (2018), "What is algebraic about algebraic effects and handlers?"

Theorem 2.1.2: Free algebras are algebras
Assuming funext, $\square^{\mathscr{M}}$ is isomorphic to the usual definition of $\mathscr{M}$-algebras.

Said otherwise, the p_i functions are equivalent to the usual morphism

$$h_A : \mathscr{M}\ A \to A$$

preserving the monadic structure, except that this presentation does not require any equation. This results in the main advantage of intensional algebras, namely that they are closed under product type in a purely intensional setting. That is, if

$$A : \square \qquad \text{and} \qquad B : A \to \square^{\mathscr{M}},$$

then

$$\Pi x : A.\, B.\pi_1$$

can be equipped with an intensional algebra structure defined pointwise. This solves a similar issue encountered in [130].

[130]: Sterling (2021), "Higher order functions and Brouwer's thesis"

As $\mathfrak{D}$ is a free monad, we can define similarly intensional $\mathfrak{D}$-algebras.

Definition 2.1.4: Pythias
A pythia for $A : \square$ is a term

$$p_A : \Pi(i : \mathrm{I}).\, (\mathrm{O}\ i \to A) \to A.$$

We call such terms pythias as they are here to deal with oracles. Per the above theorem, pythias for A are extensionally in one-to-one correspondence with $\mathfrak{D}$-algebra structures over A, but are much better behaved intensionally. This will be the crux of the branching translation for BTT in Section 3.2.3 from Chapter 3.

In ancient Greece, *pythia* was another name for the *Oracle of Delphi*.

Speech of freedom In fact, $\mathfrak{D}$ is able to encode *any* free monad.

Theorem 2.1.3:
Given A a type and $\mathscr{M}$ a free algebra as above, there are instances of I and O such that $\mathfrak{D}\ A$ is extensionally isomorphic to $\mathscr{M}\ A$.

Proof. We make use of the *containers'* encoding, as described by *Abbott et al* [1]. We do not formally prove this theorem, but give an intuition by considering the specific case where $\mathscr{M}$ is the following:

[1]: Abbott et al. (2005), "Containers: Constructing strictly positive types"

$$\begin{aligned}
&\texttt{Inductive}\ \mathscr{M}\ (A : \square) : \square :=\\
&\mid \eta : A \to \mathscr{M}\ A\\
&\mid c_1 : \Pi\, \Gamma^1.\, (\Delta_1^1 \to \mathscr{M}\ A) \to (\Delta_2^1 \to \mathscr{M}\ A) \to \mathscr{M}\ A\\
&\mid c_2 : \Pi\, \Gamma^2.\, (\Delta_1^2 \to \mathscr{M}\ A) \to (\Delta_2^2 \to \mathscr{M}\ A) \to \mathscr{M}\ A,
\end{aligned}$$

where

$$\Gamma^i, \Delta^i_j$$

are telescopes of types that do not mention $\mathscr{M}\ A$ (although they potentially mention A).

Let us first notice that, assuming funext, for any types A, B, C, X, we have the following isomorphism:

$$(A \to X) \to (B \to X) \to C \simeq ((A + B) \to X) \to C,$$

where $A + B$ is the disjoint union of A and B, defined in Coq as

$$\begin{array}{l}\texttt{Inductive sum}\ (A\ B : \square) : \square :=\\ \mid \texttt{inl} : A \to \texttt{sum}\ A\ B\\ \mid \texttt{inr} : B \to \texttt{sum}\ A\ B.\end{array}$$

Then $\mathscr{M}$ is pointwise isomorphic to the following operator:

$$\begin{array}{l}\texttt{Inductive}\ \mathscr{M}'\ (A : \square) : \square :=\\ \mid \eta' : A \to \mathscr{M}'\ A\\ \mid c_1' : \Pi\, \Gamma^1.\ (\Delta_1^1 + \Delta_2^1 \to \mathscr{M}'\ A) \to \mathscr{M}'\ A\\ \mid c_2' : \Pi\, \Gamma^2.\ (\Delta_1^2 + \Delta_2^2 \to \mathscr{M}'\ A) \to \mathscr{M}'\ A.\end{array}$$

Using containers encoding, we can now factorize c_1' and c_2' as

$$\begin{array}{l}c : \Pi\, (s : \Gamma^1 + \Gamma^2).\ (\texttt{match}\ s\ \texttt{with}\\ \qquad\qquad \mid \texttt{inl}\ \gamma_1 \longrightarrow \Delta_1^1 + \Delta_2^1\\ \qquad\qquad \mid \texttt{inr}\ \gamma_2 \longrightarrow \Delta_1^2 + \Delta_2^2\\ \qquad\qquad \texttt{end} \to \mathscr{M}'\ A)\\ \qquad\qquad \to \mathscr{M}'\ A,\end{array}$$

leading to the following operator:

$$\begin{array}{l}\texttt{Inductive}\ \mathscr{M}''\ (A : \square) : \square :=\\ \mid \eta'' : A \to \mathscr{M}''\ A\\ \mid c'' : \Pi\, (s : \Gamma^1 + \Gamma^2).\ (\texttt{match}\ s\ \texttt{with}\\ \qquad\qquad \mid \texttt{inl}\ \gamma_1 \longrightarrow \Delta_1^1 + \Delta_2^1\\ \qquad\qquad \mid \texttt{inr}\ \gamma_2 \longrightarrow \Delta_1^2 + \Delta_2^2\\ \qquad\qquad \texttt{end} \to \mathscr{M}''\ A)\\ \qquad\qquad \to \mathscr{M}''\ A.\end{array}$$

Finally, taking

$$\mathbf{I} := \Gamma^1 + \Gamma^2 \qquad \text{and} \qquad \begin{array}{l}\mathbf{O} := \lambda i : \mathbf{I}.\ \texttt{match}\ i\ \texttt{with}\\ \qquad \mid \texttt{inl}\ \gamma_1 \longrightarrow \Delta_1^1 + \Delta_2^1\\ \qquad \mid \texttt{inr}\ \gamma_2 \longrightarrow \Delta_1^2 + \Delta_2^2\\ \qquad \texttt{end},\end{array}$$

we recover $\mathscr{M}''$ as an instance of $\mathfrak{D}$. ∎

This proof is quite straightforward to extend to an arbitrary number of constructors c_i featuring an arbitrary number of telescopes Δ_i^j. Moreover, making use of *Hugunin*'s more involved encoding of W-types [80], it should be possible to remove the need for funext.

[80]: Hugunin (2021), "Why Not W?"

All of this advocates for the study of the $\mathfrak{D}$ monad, as a quite powerful tool able to encode many side-effects. In this thesis we will focus on *continuity*, a logical principle that happens to be deeply connected with $\mathfrak{D}$.

2.2. Every tree will die a log

For further questions please call the oracle The type of *dialogue trees* is known under several other names and has a lot of close relatives [88, 105, 117, 132, 146]. However, until we provide a way for it to talk, this shrub will hardly be befitting of the *dialogue* name. The key idea is, a dialogue tree can be seen as a delayed computation, waiting for an oracle to answer its calls, so that it can collapse into a value. In that way of thinking, dialogue trees are interpreted as *functionals* of type

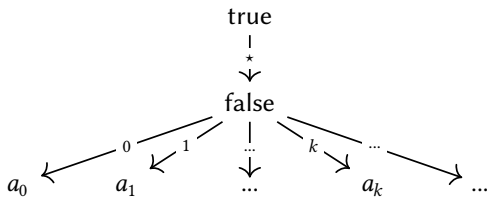
$$\mathrm{Q} \to A,$$

where every inner node is an inert call to an oracle $\alpha : \mathrm{Q}$, and the answer is the label of the leaf. This interpretation is implemented by a recursively defined *decode function*. In the field of algebraic effect, this function would be called a *handler*.

$$\begin{array}{lcl} \partial & : & \Pi\{A : \square\}\,(d : \mathfrak{D}\,A)\,(\alpha : \mathrm{Q}).\,A \\ \partial\,(\eta\,x)\,\alpha & := & x \\ \partial\,(\beta\,i\,k)\,\alpha & := & \partial\,(k\,(\alpha\,i))\,\alpha. \end{array}$$

The decode function is formally defined here.

Representing functions as trees is a well-known way to extract intensional content from them [58, 82, 89]. Pictorially, looking back at our previous example



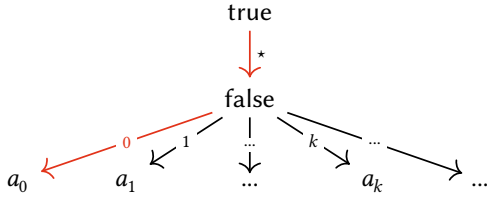
where

$$\mathrm{I} := \mathbb{B} \qquad \text{and} \qquad \mathrm{O} := \lambda i : \mathrm{I}.\ \texttt{if}\ i\ \texttt{then}\ \top\ \texttt{else}\ \mathbb{N},$$

if we take as oracle

$$\alpha : \mathrm{Q} := \lambda i : \mathrm{I}.\quad \texttt{if}\ i\ \texttt{then}\ \star : \top\ \texttt{else}\ \mathrm{O} : \mathbb{N},$$

then α describes the following path in d:



At the end, we recover:

$$\partial\ d\ \alpha = a_0.$$

Functionals that can be encoded as dialogue trees are called *dialogue continuous*.

[88]: Kiselyov et al. (2015), "Freer monads, more extensible effects"
[105]: McBride (2015), "Turing-Completeness Totally Free"
[117]: Piróg et al. (2014), "The Coinductive Resumption Monad"
[132]: Swierstra (2008), "Data types à la carte"
[146]: Xia et al. (2020), "Interaction trees: representing recursive and impure programs in Coq"

[58]: Ghani et al. (2009), "Representations of Stream Processors Using Nested Fixed Points"
[82]: Hyland et al. (2000), "On Full Abstraction for PCF: I, II, and III"
[89]: Kleene (1978), "Recursive Functionals and Quantifiers of Finite Types Revisited I"

Definition 2.2.1: Dialogue continuity

A function $f : \mathbb{Q} \to A$ validates *dialogue continuity* if the following holds:

$$\mathscr{C}_{\mathfrak{D}} := \Sigma d : \mathfrak{D}\, A. \Pi \alpha : \mathbb{Q}.\ f\, \alpha = \partial\, d\, \alpha.$$

Dialogue continuity is formally defined here.

It might look surprising to a reader inclined towards mathematics to find the name *continuity* here, with a definition that does not transparently recall the usual, by-the-book definition of continuity. We will however see in Section 2.3 that dialogue continuity coincides with standard continuity in a classical and extensional enough setting. In fact, there are many different definitions of continuity in the literature, some stronger than others. The purpose of this Section is to provide a small survey of continuity definitions, and their relative implications.

We will encounter many inductive definitions in this Section, some for predicates, some for data types. As we think it is useful to mentally distinguish the two, we will write inductive data types in a Coq or Agda-like syntax, and inductive predicates with inference rules. However, for readers who would want an uniform way of presenting both, we will also provide in the margin Coq-like definitions of inductive predicates.

At the end of this Section, we will end up with a map of continuity principles presented in Figure 2.1. All implications displayed in this Figure are internal to MLTT here, meaning for instance that when we state as a theorem that *intensional dialogue continuity implies dialogue continuity*, it effectively means that there is a term

$$\cdot \vdash^{\mathsf{MLTT}} \_ : \Pi(f : \mathbb{Q} \to A).\, \mathscr{C}_{\mathfrak{I}}\, f \to \mathscr{C}_{\mathfrak{D}}\, f$$

in MLTT.

Finally, all results presented in this Section are proven in Coq.

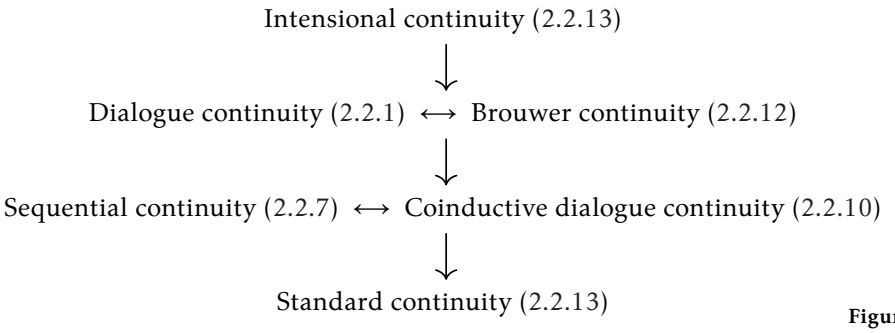


Figure 2.1.: Projected map of continuity principles

2.2.1. Standard definition

The most common definition of continuity is what we call the *standard definition*. It is also called *pointwise continuity* in constructive mathematics, where it is usually described in a setting where

$$\mathrm{I} \coloneqq \mathbb{N} \qquad \text{and} \qquad \mathrm{O}\ i \coloneqq \mathbb{B} \qquad \text{or} \qquad \mathrm{O}\ i \coloneqq \mathbb{N},$$

meaning that Q is the *Cantor space* or the *Baire space*. In that case, in mathematical terms, the standard definition comes from taking the product topology on Q. However, it is straightforward to extend this definition to arbitrary I and O, as follows:

Definition 2.2.2: Finite equality
Given $\alpha_1, \alpha_2 : \mathrm{Q}$ and $l : \mathrm{list}\ \mathrm{I}$, we say that α_1 and α_2 are *finitely equal* on l, written $\alpha_1 \approx_l \alpha_2$ when the following inductively defined predicate holds:

$$\frac{}{\alpha_1 \approx_{[]} \alpha_2} \qquad \frac{\alpha_1\ i = \alpha_2\ i \qquad \alpha_1 \approx_l \alpha_2}{\alpha_1 \approx_{(i::l)} \alpha_2}$$

This definition is equivalent to the following Coq-like syntax:

```
Inductive EqFin (α1 α2 : Q) :
  list I → □i :=
| Eqnil :  EqFin α1 α2 []
| Eqcons : Π(i : I)(l : list I).
             α1 i = α2 i →
             EqFin α1 α2 l →
             EqFin α1 α2 (i :: l).
```

Standard continuity is formally defined here.

Definition 2.2.3: Standard continuity
A function validates *standard continuity* when it satisfies the following predicate:

$$\begin{aligned} \mathscr{C} \quad &: \quad \Pi\{A : \square\}.\,(\mathrm{Q} \to A) \to \square \\ \mathscr{C}\ f \quad &:= \quad \Pi(\alpha : \mathrm{Q}).\,\Sigma(l : \mathrm{list}\ \mathrm{I}).\,\Pi(\beta : \mathrm{Q}).\,\alpha \approx_l \beta \to f\ \alpha = f\ \beta. \end{aligned}$$

Note that the equality $f\ \alpha = f\ \beta$ means that we implicitly take the discrete topology on A in this definition. The usual case will be $A \coloneqq \mathbb{N}$. This definition captures in a generic way the intuitive notion that a computable functional only needs a finite amount of information from its argument to produce an output. Depending on the expressivity of the theory, one can also consider weaker variants where the existential is squashed with various proof-irrelevant modalities [52, 123, 124]. We emphasize that the list of points l where the function is evaluated depends on the argument α, so this notion of continuity is weaker than uniform continuity, where the two quantifiers for l and α are swapped.

In fact, we have the following:

Definition 2.2.4: Uniform continuity
A function validates *uniform continuity* when it satisfies the following predicate:

$$\begin{aligned} \mathscr{C} \quad &: \quad \Pi\{A : \square\}.\,(\mathrm{Q} \to A) \to \square \\ \mathscr{C}\ f \quad &:= \quad \Sigma(l : \mathrm{list}\ \mathrm{I}).\,\Pi(\alpha\ \beta : \mathrm{Q}).\,\alpha \approx_l \beta \to f\ \alpha = f\ \beta. \end{aligned}$$

[52]: Escardó et al. (2015), "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation"
[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"
[124]: Rahli et al. (2019), "Bar Induction is Compatible with Constructive Type Theory"

Uniform continuity is formally defined here.

Lemma 2.2.1: Uniform is dialogue
Given $f : \mathbb{Q} \to A$, if f is uniformly continuous, if $\mathbb{I}$ has decidable equality and if $\mathbb{Q}$ is inhabited, then f is dialogue continuous.

Implication of dialogue continuity by uniform continuity is formally proved here in the specific case of the Cantor space.

Proof. Let us assume $\alpha : \mathbb{Q}$ and that $\mathbb{I}$ has decidable equality.

Given a uniformly continuous function f together with its witness

$$l : \mathsf{list}\ \mathbb{I},$$

we have an easy way to build the structure of the tree: we can simply build the complete dialogue tree of height $\mathtt{len}\ l$ (where $\mathtt{len}\ l$ is the length of l), making sure that every node $\beta\ i\ k$ is such that i is in l.

The difficulty is to provide a leaf $\eta\ a$. Our only hope is to provide $f\ \beta$ for some suitable β, which means we have to build it. The idea is then to maintain a list of pairs to keep track of what answers α had to give to choose the path we are in. Formally, we first define:

$$\begin{array}{lcl}
\mathtt{evallist} & : & \mathsf{list}\ (\Sigma i : \mathbb{I}.\ \mathbb{O}\ i) \to \mathbb{Q} \\
\mathtt{evallist}\ \mathsf{nil}\ i & := & \alpha\ i \\
\mathtt{evallist}\ ((i,o) :: l)\ i' & := & \mathtt{if}\ (i == i')\ \mathtt{then}\ o\ \mathtt{else}\ \mathtt{evallist}\ l\ i'
\end{array}$$

Here, $i == i'$ is a notation for decidable equality on $\mathbb{I}$. Intuitively, `evallist` evaluates the partial function defined by a list

$$q : \mathsf{list}\ (\Sigma i : \mathbb{I}.\ \mathbb{O}\ i).$$

When some argument is not in l, it uses α to return a default value.

Now, assuming $f : \mathbb{Q} \to A$ uniformly continuous, the function used to build a dialogue tree is the following:

$$\begin{array}{lcl}
\mathtt{LtoD} & : & \mathsf{list}\ \mathbb{I} \to \mathsf{list}\ (\Sigma i : \mathbb{I}.\ \mathbb{O}\ i) \to \mathfrak{D}\ A \\
\mathtt{LtoD}\ \mathsf{nil}\ q & := & \eta\ (f\ (\mathtt{evallist}\ q)) \\
\mathtt{LtoD}\ (i :: l)\ q & := & \beta\ i\ (\lambda o : \mathbb{O}\ i.\ \mathtt{LtoD}\ l\ ((i,o) :: q))
\end{array}$$

Given

$$l : \mathsf{list}\ \mathbb{I}$$

the witness of uniform continuity for f,

$$\mathtt{LtoD}\ l\ \mathsf{nil}$$

is the required dialogue tree. All that is left is to show that for any

$$\alpha : \mathbb{Q},$$

we indeed get

$$f\ \alpha = \partial\ (\mathtt{LtoD}\ l\ \mathsf{nil})\ \alpha.$$

The proof is maybe a bit technical, but in summary continuity of f is what we need. We encourage the interested reader to take a look at the formalization to get more details. ∎

However, uniform continuity is too strict a requirement in most settings. As pointed out by *Fujiwara et al* [57], uniform continuity mainly

[57]: Fujiwara et al. (2019), "Equivalence of bar induction and bar recursion for continuous functions with continuous moduli"

works when Q is the *Cantor space*, which has the specific property of being *compact*, which is not the case of other spaces. For instance, taking

$$\mathrm{I} \coloneqq \mathbb{N}, \qquad \mathrm{O}\ i \coloneqq \mathbb{N} \qquad \text{and} \qquad A \coloneqq \mathbb{N},$$

the function

$$f \coloneqq \lambda\alpha : \mathbb{N} \to \mathbb{N}.\ \alpha\ (\alpha\ \mathrm{O})$$

fails to be uniformly continuous, even though it is fairly simple. Moreover, in the specific case of the Cantor space, we have the following result:

We do not define *compactness* here, as it is fairly technical. Let us simply state that when O i is finite for all i, then Q is compact.

Proposition 2.2.2: Cantor dialogue is uniform
In the case when Q is the Cantor space, then a function

$$f : \mathrm{Q} \to A$$

is uniformly continuous if and only if it is dialogue continuous.

Implication of uniform continuity by dialogue continuity in the specific case of the Cantor space is formally proved here.

Proof. Thanks to Lemma 2.2.1, we only need to prove the converse implication. Assuming a dialogue continuous function f together with its witness $d : \mathfrak{D}\ A$, we notice that since

$$\mathrm{O}\ i \coloneqq \mathbb{B}$$

is finite for every i, then d is also finite. By induction on d, we can thus recover a list l : list I and derive uniform continuity. The same proof applies for any Q such that O i is finite for all i. ∎

This result was already pointed out by *Escardó* [51] and more recently by *Cohen et al* [37] but the links between uniform continuity and tree-like structures have long been observed [20–22]. This has led *Fujiwara et Kawai* [57] to consider inductively-defined notions of continuity (such as dialogue continuity) as a generalization of uniform continuity at all types.

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

[37]: Cohen et al. (2023), "Inductive Continuity via Brouwer Trees"

[20]: Berger (2012), "Aligning the weak König lemma, the uniform continuity theorem, and Brouwer's fan theorem"

[21]: Berger (2005), "The Fan Theorem and Uniform Continuity"

[22]: Berger (2006), "The Logical Strength of the Uniform Continuity Theorem"

[57]: Fujiwara et al. (2019), "Equivalence of bar induction and bar recursion for continuous functions with continuous moduli"

Finally, let us emphasize that the above lemma only works because we use the discrete topology on A, which entails that pointwise continuous functions with A as codomain are in fact *locally constant*. This is not the case in general: for instance, taking

$$\begin{aligned} \mathrm{I} &\coloneqq \mathbb{N} \\ \mathrm{O} &\coloneqq \lambda\_.\ \mathbb{B} \\ A &\coloneqq \mathbb{N} \to \mathbb{B} \end{aligned}$$

then the function

$$f \coloneqq \lambda(\alpha : \mathbb{N} \to \mathbb{B}).\ \alpha$$

is neither dialogue continuous nor locally constant, although it would be pointwise continuous in constructive mathematics. In the remainder of this thesis, A will always implicitly be considered equipped with the discrete topology (in fact, we will only encounter examples where A is either $\mathbb{N}$, $\mathbb{B}$ or $\bot$).

2.2.2. Sequential continuity

An intermediate step between standard and dialogue continuity comes from a more extensional way of defining trees, seeing them as *predicates over lists*. We call them *extensional trees* and take their definition from *van Oosten* [109, 141], recently rephrased by *Forster et al* [53] in the setting of synthetic computability theory [16]. We however slightly simplify the setting, as we only care about total functions while *van Oosten's* definition is geared towards partial functions. *Forster et al* use the name *sequential continuity*; we stick with their notation but warn the reader not to be confused, as sequential continuity sometimes mean a different concept in constructive mathematics.

[109]: Oosten (2011), "Partial Combinatory Algebras of Functions"
[141]: Van Oosten (1999), "A combinatory algebra for sequential functionals of finite type"
[53]: Forster et al. (2023), "Oracle Computability and Turing Reducibility in the Calculus of Inductive Constructions"
[16]: Bauer (2005), "First Steps in Synthetic Computability Theory"

In this Section, let us assume that $\mathbf{O}$ does not depend on $\mathbf{I}$. That is, we simply have

$$\mathbf{I}, \mathbf{O} : \square_0.$$

Definition 2.2.5: Extensional tree
We consider the `Result` operator, defined by the following inductive predicate:

$$\begin{aligned}&\texttt{Inductive Result}\ (A : \square_i) : \square_i :=\\ &\mid \texttt{ret} : A \to \texttt{Result}\ A\\ &\mid \texttt{ask} : \mathbf{I} \to \texttt{Result}\ A.\end{aligned}$$

Then the type of *extensional trees* is the following:

$$\mathsf{Extree}\ (A : \square_i) : \square_i := \mathsf{list}\ \mathbf{O} \to \texttt{Result}\ A.$$

Extensional trees are formally defined here.

Since we only deal with total functions, and not partial ones as *Forster et al*, we skip the `reject` constructor used to implement partiality. Like dialogue trees, extensional trees can be interpreted as functionals, using another *decode* function.

Definition 2.2.6: Extensional decode
Extensional trees can be interpreted as functionals of type $\mathbf{Q} \to A$. As for dialogue trees, every `ask` constructor is an inert call to an oracle $\alpha : \mathbf{Q}$, and the answer is a result given by `ret`. This interpretation is implemented by a recursively defined *extensional decode function*:

$$\begin{aligned}&\partial_{\mathsf{Ext}} && : && \mathsf{Extree} \to \mathbf{Q} \to \mathsf{list}\ \mathbf{O} \to \mathbb{N} \to \texttt{Result}\ A\\ &\partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ \mathrm{O} && := && \tau\ l\\ &\partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ (\mathrm{S}\ k) && := && \texttt{match}\ (\tau\ l)\ \texttt{with}\\ &&&&& \quad \mid \texttt{ret}\ a \longrightarrow \texttt{ret}\ a\\ &&&&& \quad \mid \texttt{ask}\ i \longrightarrow \partial_{\mathsf{Ext}}\ \tau\ \alpha\ ((\alpha\ i) :: l)\ k\\ &&&&& \quad \texttt{end}\end{aligned}$$

Evaluation of extensional trees is formally defined here.

As we are not certain that τ will return a value after a finite number of steps, we use a natural number n as fuel to make the recursion well-founded. A similar definition can be found in *van Oosten's* book [110] in the proof of Theorem 1.7.5 under the name *f-dialogue*.

[110]: Oosten (2008), "Realizability: an introduction to its categorical side. Studies in Logic and the Foundations of Mathematics, vol. 152. Elsevier Science, Amsterdam, 2008, 328 pp."

Definition 2.2.7: Sequential continuity
A function $f : \mathbb{Q} \to A$ is sequentially continuous if the following holds:

$$\mathscr{C}_{\mathsf{Ext}}\ f := \Sigma(\tau : \mathsf{Extree}).\ \Pi(\alpha : \mathbb{Q}).\ \Sigma(n : \mathbb{N}).\ \partial_{\mathsf{Ext}}\ \tau\ \alpha\ []\ n = \mathtt{ret}\ (f\ \alpha)$$

Sequential continuity is formally defined here.

Lemma 2.2.3: Sequential implies standard continuity
Sequential continuity implies standard continuity.

Implication of standard continuity by sequential continuity is formally proved here.

Proof. Let us assume $f : \mathbb{Q} \to A$ and $\tau : \mathsf{Extree}$ its witness of sequential continuity. Given $\alpha : \mathbb{Q}$, we have

$$n : \mathbb{N} \qquad \text{such that} \qquad \partial_{\mathsf{Ext}}\ \tau\ \alpha\ \mathsf{nil}\ n = \mathtt{ret}\ (f\ \alpha).$$

We want to produce a list $l : \mathsf{list}\ \mathbb{I}$. Intuitively, this list is exactly the one we reach at the end of the ∂_{Ext} recursion. Hence, we define an auxiliary function

$$\begin{array}{lcl}
\mathsf{aux} & : & \Pi(\tau : \mathsf{Extree})\,(\alpha : \mathbb{Q})\,(l : \mathsf{list}\ \mathbb{O})\,(n : \mathbb{N}).\ \mathsf{list}\ \mathbb{I} \\
\mathsf{aux}\ \tau\ \alpha\ l\ \mathsf{O} & := & \mathsf{nil} \\
\mathsf{aux}\ \tau\ \alpha\ l\ (\mathsf{S}\ k) & := & \mathtt{match}\ (\tau\ l)\ \mathtt{with} \\
 & & \quad \mid \mathtt{ret}\ a \ \longrightarrow\ \mathsf{nil} \\
 & & \quad \mid \mathtt{ask}\ i \ \longrightarrow\ i :: (\mathsf{aux}\ \tau\ \alpha\ ((\alpha\ i) :: l)\ k) \\
 & & \quad \mathtt{end}
\end{array}$$

We now need to prove the following:

$$\Pi(\beta : \mathbb{Q}).\ \alpha \approx_{\mathsf{aux}\ \tau\ \alpha\ \mathsf{nil}\ n} \beta \to f\ \alpha = f\ \beta.$$

Given

$$\beta \qquad \text{and} \qquad H : \alpha \approx_{\mathsf{aux}\ \tau\ \alpha\ \mathsf{nil}\ n} \beta,$$

by sequential continuity of f we recover

$$m : \mathbb{N} \qquad \text{such that} \qquad \partial_{\mathsf{Ext}}\ \tau\ \beta\ \mathsf{nil}\ m = \mathtt{ret}\ (f\ \beta).$$

By injectivity of `ret`, it is sufficient to prove that

$$\partial_{\mathsf{Ext}}\ \tau\ \alpha\ \mathsf{nil}\ n = \partial_{\mathsf{Ext}}\ \tau\ \beta\ \mathsf{nil}\ m.$$

In fact, we need to slightly generalize that goal and prove

$$\Pi(l : \mathsf{list}\ \mathbb{I}).\ \alpha \approx_{\mathsf{aux}\ \tau\ \alpha\ l\ n} \beta \to \partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ n = \partial_{\mathsf{Ext}}\ \tau\ \beta\ l\ m.$$

We first prove an auxiliary lemma stating that

$$\Pi(n\ m : \mathbb{N})\,(a : A).\ n \le m \to \partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ n = \mathtt{ret}\ a \to \partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ m = \mathtt{ret}\ a.$$

This follows from double induction on n and m. The key point is that once ∂_{Ext} reaches a value, it stays constant. All that is left is to prove

$$\partial_{\mathsf{Ext}}\ \tau\ \alpha\ l\ (\mathsf{max}\ n\ m) = \partial_{\mathsf{Ext}}\ \tau\ \beta\ l\ (\mathsf{max}\ n\ m),$$

which is straightforward by induction on $(\mathsf{max}\ n\ m)$. ∎

2.2.3. Interaction Trees

The key difference between dialogue trees and extensional trees is that dialogue trees are inductive concepts while extensional trees are possibly infinite trees that we restrict *a posteriori* by asking a natural number in the continuity definition. However, functions are not the only way to encode potentially infinite data structures in type theory, another option is to use *coinductive types*. In this setting, *Xia et al* [146] describe what they call *interaction trees*, a data structure behaving similarly to extensional trees. We give a slightly different version of them, fitting into our setting with I and O.

[146]: Xia et al. (2020), "Interaction trees: representing recursive and impure programs in Coq"

Definition 2.2.8: Interaction trees
Interaction trees are inhabitants of the following coinductive type:

$$\begin{aligned}&\texttt{Coinductive}\ \mathfrak{I}\ (A : \square_i) : \square_i :=\\ &\mid \texttt{Ret} : A \to \mathfrak{I}\ A\\ &\mid \texttt{Tau} : \mathfrak{I}\ A \to \mathfrak{I}\ A\\ &\mid \texttt{Vis} : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to \mathfrak{I}\ A) \to \mathfrak{I}\ A.\end{aligned}$$

Interaction trees are formally defined here.

Here, `Vis` stands for *visible event*, where the computation explicitly asks the oracle for an answer, contrarily to the *silent transition* `Tau` where the program silently computes. In Xia's setting, `Tau` transitions are also used to model non-termination. This actually means that interaction trees are the composition of two effects:

1. Potentially endless interaction with an external oracle $\alpha : \mathrm{Q}$, encoded by a coinductive version of $\mathfrak{D}$;
2. Silent non-termination, encoded via *Capretta*'s *delay monad* [30].

[30]: Capretta (2005), "General Recursion via Coinductive Types"

See you later, operator Interestingly, if we remove the delay monad from interaction trees we recover a definition of continuity equivalent to the one with extensional trees.

Definition 2.2.9: Coinductive dialogue trees
Coinductive dialogue trees are inhabitants of the following coinductive type:

$$\begin{aligned}&\texttt{Coinductive}\ \mathfrak{E}\ (A : \square_i) : \square_i :=\\ &\mid \texttt{Ret}_{\mathfrak{E}} : A \to \mathfrak{E}\ A\\ &\mid \texttt{Vis}_{\mathfrak{E}} : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to \mathfrak{E}\ A) \to \mathfrak{E}\ A.\end{aligned}$$

Coinductive dialogue trees are formally defined here.

They come with their own decoding function, interpreting them as functionals of type $\mathrm{Q} \to A$. As they are coinductives, the decoding function is parametrized by a natural number n that we use as fuel.

$$\begin{aligned}&\partial_{\mathfrak{E}} && : && \Pi\{A : \square_i\}.\ \mathfrak{E}\ A \to \mathrm{Q} \to \mathbb{N} \to \texttt{Result}\ A\\ &\partial_{\mathfrak{E}}\ (\texttt{Ret}_{\mathfrak{E}}\ a)\ \alpha\ n && \coloneqq && \texttt{ret}\ a\\ &\partial_{\mathfrak{E}}\ (\texttt{Vis}_{\mathfrak{E}}\ i\ k)\ \alpha\ \mathrm{O} && \coloneqq && \texttt{ask}\ i\\ &\partial_{\mathfrak{E}}\ (\texttt{Vis}_{\mathfrak{E}}\ i\ k)\ \alpha\ (\mathrm{S}\ n) && \coloneqq && \partial_{\mathfrak{E}}\ (k\ (\alpha\ i))\ \alpha\ n\end{aligned}$$

Definition 2.2.10: Coinductive dialogue continuity
A function $f : \mathbb{Q} \to A$ is *coinductive dialogue continuous* if the following holds:

$$\mathscr{C}_{\mathfrak{E}}\ f \coloneqq \Sigma(c : \mathfrak{E}).\ \Pi(\alpha : \mathbb{Q}).\ \Sigma(n : \mathbb{N}).\ \partial_{\mathfrak{E}}\ c\ \alpha\ n = \mathtt{ret}\ (f\ \alpha)$$

Coinductive dialogue continuity is formally defined here.

Proposition 2.2.4: Coinductive dialogue and sequential continuity coincide
Sequential continuity and coinductive dialogue continuity are equivalent.

The first implication is formally proved here, the converse is formally proved here.

Proof. This Proposition of course only makes sense in a setting where $\mathbb{O}$ does not depend on $\mathbb{I}$, hence we switch back to it.

To go from sequential continuity to coinductive dialogue continuity, we make use of the following conversion function:

$$\begin{array}{lcl}
\mathtt{EtoC} & : & \Pi\{A : \square_i\}.\ \mathsf{Extree} \to \mathsf{list}\ \mathbb{O} \to \mathfrak{E}\ A \\
\mathtt{EtoC}\ \tau\ l & \coloneqq & \mathtt{match}\ \tau\ l\ \mathtt{with} \\
& & \quad \mid \mathtt{ret}\ a \longrightarrow \mathtt{Ret}_{\mathfrak{E}}\ a \\
& & \quad \mid \mathtt{ask}\ i \longrightarrow \mathtt{Vis}_{\mathfrak{E}}\ i\ (\lambda o : \mathbb{O}.\ \mathtt{EtoC}\ \tau\ (o :: l)) \\
& & \quad \mathtt{end}
\end{array}$$

Remember that the type of extensional trees is

$$\mathsf{Extree} \coloneqq \mathsf{list}\ \mathbb{O} \to \mathtt{Result}\ A$$

Note that, since we are building an element of a coinductive type, there is no question of termination here, only a question of productivity. We are effectively building a possibly infinite chain of queries to the oracle.

Then, given a sequentially continuous functional

$$f : \mathbb{Q} \to A$$

and its witness of continuity

$$\tau : \mathsf{Extree},$$

we provide

$$\mathtt{EtoC}\ \tau\ \mathsf{nil}$$

as a witness of coinductive dialogue continuity for f. What we have to prove then amounts to

$$\Pi(l : \mathsf{list}\ A)\,(n : \mathbb{N}).\ \partial_{\mathtt{Ext}}\ \tau\ \alpha\ l\ n = \partial_{\mathfrak{E}}\ (\mathtt{EtoC}\ \tau\ l)\ \alpha\ n.$$

The proof follows from a straightforward induction on n.

To go from coinductive dialogue continuity to sequential continuity, we proceed the same way: first, we define the following conversion function:

$$\begin{array}{lcl}
\mathtt{CtoE} & : & \Pi\{A : \square_i\}.\ \mathfrak{E}\ A \to \mathsf{Extree} \\
\mathtt{CtoE}\ (\mathtt{Ret}_{\mathfrak{E}}\ a)\ _ & \coloneqq & \mathtt{ret}\ a \\
\mathtt{CtoE}\ (\mathtt{Vis}_{\mathfrak{E}}\ i\ k)\ \mathsf{nil} & \coloneqq & \mathtt{ask}\ i \\
\mathtt{CtoE}\ (\mathtt{Vis}_{\mathfrak{E}}\ i\ k)\ (l\,;;\,o) & \coloneqq & \mathtt{CtoE}\ (k\ o)\ l
\end{array}$$

Note that the function is defined by induction on the list l, but from right to left and not from left to right. This is to avoid problems when ∂_{Ext} adds elements to the list.

Then, given a coinductive dialogue continuous functional $f : \mathrm{Q} \to A$ and its witness of continuity

$$c : \mathfrak{C}\ A,$$

we provide

$$\mathtt{CtoE}\ c$$

as a witness of sequential continuity for f. Once again, we end up having to prove that our conversion procedure preserves the behaviour of decoding functions:

$$\Pi(l : \mathsf{list}\ A)(n : \mathbb{N}).\ \partial_{\mathfrak{C}}\ c\ \alpha\ n = \partial_{\mathsf{Ext}}\ (\mathtt{CtoE}\ c)\ \alpha\ l\ n.$$

The proof follows from a straightforward induction on n. ∎

Talking is a good way to interact Looking at interaction trees, the link with dialogue trees is quite obvious, as one is simply the coinductive counterpart of the other. It is pretty easy to turn an element of an inductive type into an element of its coinductive dual, so the following result is hardly surprising.

Proposition 2.2.5: Dialogue is interaction
Dialogue continuous functions are coinductive-dialogue continuous.

Proof. We use the following conversion function from dialogue trees to coinductive dialogue trees:

$$\begin{array}{lcl} \mathtt{DtoC} & : & \Pi\{A : \square_i\}.\ \mathfrak{D}\ A \to \mathfrak{C}\ A \\ \mathtt{DtoC}\ (\eta\ a) & \coloneqq & \mathtt{Ret}_{\mathfrak{C}}\ a \\ \mathtt{DtoC}\ (\beta\ i\ k) & \coloneqq & \mathtt{Vis}_{\mathfrak{C}}\ i\ (\lambda o : \mathrm{O}\ i.\ \mathtt{DtoC}\ (k\ o)) \end{array}$$

It is then sufficient to prove

$$\Pi(d : \mathfrak{D}\ A)(\alpha : \mathrm{Q}).\Sigma n : \mathbb{N}.\partial\ d\ \alpha = \partial_{\mathfrak{C}}\ (\mathtt{DtoC}\ d)\ \alpha\ n.$$

Given $\alpha : \mathrm{Q}$, the natural number n we return is intuitively the length of the path taken by α in d before reaching a leaf. This intuition is formalized in the following function:

$$\begin{array}{lcl} \mathsf{aux} & : & \Pi\{A : \square_i\}.\ \mathfrak{D}\ A \to \mathrm{Q} \to \mathbb{N} \\ \mathsf{aux}(\eta\ a)\ \alpha & \coloneqq & \mathrm{O} \\ \mathsf{aux}\ (\beta\ i\ k)\ \alpha & \coloneqq & \mathrm{S}\ (\mathsf{aux}\ (k\ (\alpha\ i))\ \alpha) \end{array}$$

The proof that

$$\partial\ d\ \alpha = \partial_{\mathfrak{C}}\ (\mathtt{DtoC}\ d)\ \alpha\ (\mathsf{aux}\ d\ \alpha)$$

is straightforward by induction on d. ∎

2.2.4. Monologuing Trees

One question at a time, please Dialogue trees can ask their queries in whatever order they choose to, or even ask the same question an arbitrary number of times. Building on previous work by *Escardó and Oliva*, *Sterling* [130] decides to address this unproper behaviour and defines the type of *Brouwer trees*. Inspired by the work of Brouwer [28], they are a specific kind of dialogue trees where queries must be made in order. They are defined in a setting where

$$\mathrm{I} := \mathbb{N}$$

and O does not depend on I.

Escardó and Oliva's specific work on Brouwer trees did not lead to a paper, but their Agda code can be browsed online here.

[130]: Sterling (2021), "Higher order functions and Brouwer's thesis"

[28]: Brouwer (1925), "Zur Begründung der intuitionistischen Mathematik. I."

Definition 2.2.11: Brouwer Trees
Given A a type, *Brouwer trees on* A are inductively defined as follows:

$$\begin{aligned}&\texttt{Inductive}\ \mathfrak{B}\,(A : \square_i) : \square_i := \\ &\mid \texttt{spit} : A \to \mathfrak{B}\,A \\ &\mid \texttt{bite} : (\mathrm{O} \to \mathfrak{B}\,A) \to \mathfrak{B}\,A.\end{aligned}$$

Brouwer trees are formally defined here.

The purpose of Brouwer trees in Sterling's work is to study continuity on *streams of value*. As such, the decoding function for dialogue trees makes use of usual functions for streams, such as *head* (`hd`) or *tail* (`tl`). In our case, we will simply consider that streams of type A are functions of type $\mathbb{N} \to A$. Hence, in the following, we have:

$$\begin{aligned}\texttt{hd}\ \alpha &:= \alpha\ \mathrm{O} \\ \texttt{tl}\ \alpha &:= \lambda(x : \mathbb{N}).\ \alpha\ (\mathrm{S}\ x) \\ \mathrm{Q} &:= \mathbb{N} \to \mathrm{O}\end{aligned}$$

These notations being set, we get:

$$\begin{aligned}\partial_{\mathfrak{B}} &: \Pi\{A : \square\}\,(d : \mathfrak{B}\,A)\,(\alpha : \mathrm{Q}).\ A \\ \partial_{\mathfrak{B}}\,(\texttt{spit}\ x)\,\alpha &:= x \\ \partial_{\mathfrak{B}}\,(\texttt{bite}\ k)\,\alpha &:= \partial_{\mathfrak{B}}\,(k\,(\texttt{hd}\ \alpha))\,(\texttt{tl}\ \alpha).\end{aligned}$$

Similarly to the dialogue case, we can define Brouwer continuity.

Definition 2.2.12: Brouwer continuity
A function $f : \mathrm{Q} \to A$ validates *Brouwer continuity* if there exists a Brouwer tree $b : \mathfrak{B}\,A$ and a proof that

$$\Pi\alpha : \mathrm{Q}.\ f\,\alpha = \partial_{\mathfrak{B}}\,d\,\alpha$$

Brouwer continuity is formally defined here.

Brouwer trees can be seen as normal forms of dialogue trees. Actually, *Sterling* exhibits a normalization procedure to turn a dialogue tree into a Brouwer tree and proves the following:

Proposition 2.2.6: Dialogue can be done in order
For $f : (\mathbb{N} \to \mathrm{O}) \to A$, Brouwer and dialogue continuity are equivalent.

Equivalence of Brouwer and dialogue continuity is formally proved here.

Proof. It is enough to provide a procedure to turn a Brouwer tree into a dialogue one (and the other way around) such that the decoding functions compute the same way.

Going from a Brouwer tree to a dialogue tree is the easy part, we simply keep track of the number of queries we have already made:

$$\begin{array}{lcl} \texttt{BtoD} & : & \Pi\{A : \square\}.\ \mathfrak{B}\ A \to \mathbb{N} \to \mathfrak{D}\ A \\ \texttt{BtoD}\ (\texttt{spit}\ a)\ \_ & \coloneqq & \eta\ a \\ \texttt{BtoD}\ (\texttt{bite}\ k)\ n & \coloneqq & \beta\ n\ (\lambda o : \mathrm{O}.\ \texttt{BtoD}\ (k\ o)\ (\mathrm{S}\ n)) \end{array}$$

Then, given $b : \mathfrak{B}\ A$, it is straightforward to prove by induction on b that

$$\Pi(k : \mathbb{N})\ (\alpha : \mathrm{Q}).\ \partial_{\mathfrak{B}}\ b\ (\lambda n.\ \alpha\ (k + n)) = \partial\ (\texttt{BtoD}\ b\ k)\ \alpha.$$

Intanciating with $k \coloneqq \mathrm{O}$, we recover

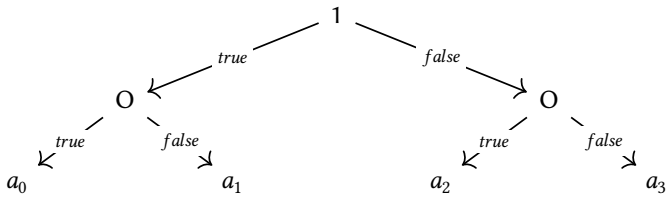
$$\Pi\alpha : \mathrm{Q}.\ \partial_{\mathfrak{B}}\ b\ \alpha = \partial\ (\texttt{BtoD}\ b\ \mathrm{O})\ \alpha.$$

The other way around is a bit more tricky. Indeed, we need to know how to deal with dialogue trees such as the following one:

In this picture, we set

$$\mathrm{O} \coloneqq \mathbb{B},$$

but in the general case the tree might be infinitely large.



Given

$$\alpha : \mathrm{Q},$$

the only way for us to have access to the value of α on 1 is by doing two successive `bite` nodes, but if we are not cautious we might lose access to the value of α on O in the process. *Sterling* finds a way out *via* an auxiliary list

$$l : \mathsf{list}\ \mathrm{I}$$

keeping track of all queries already asked, and their corresponding answer in the current branch. Morally, it means something like this:

$$\begin{array}{lcl} \texttt{DtoB} & : & \Pi\{A : \square\}.\ \mathfrak{D}\ A \to \mathsf{list}\ \mathrm{O} \to \mathfrak{B}\ A \\ \texttt{DtoB}\ (\eta\ a)\ \_ & \coloneqq & \texttt{spit}\ a \\ \texttt{DtoB}\ (\beta\ i\ k)\ l & \coloneqq & \texttt{if}\ i < \texttt{len}\ l \\ & & \quad \texttt{then}\ \texttt{DtoB}\ (k\ (l.\texttt{nth}\ i)) \\ & & \quad \texttt{else}\ \texttt{bite}\ (\lambda o.\ \texttt{DtoB}\ (\beta\ i\ k)\ (o :: l)) \end{array}$$

where

$$\texttt{len}\ l$$

is the length of the list l and

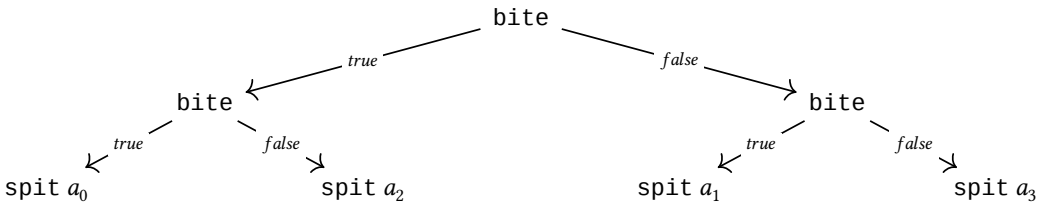
$$l.\texttt{nth}$$

is the function returning the nth element of the list l (if it exists).

However, no proof assistant would accept this function as is, since its termination is not obvious. To circumvent this issue, *Sterling* describes the following procedure, making use of an auxiliary function:

$$
\begin{array}{lcl}
\texttt{DtoB} & : & \Pi\{A : \square\}.\, \mathfrak{D}\ A \to \text{list}\ \mathbf{O} \to \mathfrak{B}\ A \\
\texttt{DtoB}\ (\eta\ a)\ \_ & := & \texttt{spit}\ a \\
\texttt{DtoB}\ (\beta\ i\ k)\ l & := & \text{aux}\ l\ i\ k\ l \\
 & & \text{where} \\
\text{aux} & : & \Pi\{A : \square\}.\, \text{list}\ \mathbf{O} \to \mathbb{N} \to (\mathbf{O} \to \mathfrak{D}\ A) \to \\
 & & \quad \text{list}\ \mathbf{O} \to \mathfrak{B}\ A \\
\text{aux}\ l\ i\ k\ \text{nil} & := & \texttt{bite}\ (\lambda o.\ \text{aux}\ (o :: l)\ i\ k\ [o]) \\
\text{aux}\ l\ \text{O}\ k\ (o :: \_) & := & \texttt{DtoB}\ (k\ o)\ l \\
\text{aux}\ l\ (S\ i)\ k\ (\_ :: q) & := & \text{aux}\ l\ i\ k\ q
\end{array}
$$

Applying `DtoB` to our pictorial example, we get:



To prove the correctness of this function, *Sterling* also provides two mutually defined inductive predicates. Unfortunately, even though this code type-checks in Agda, Coq does not accept this definition either, as it is not immediate that the fixpoint is structurally decreasing.

However, as we already mentioned, another proof exists, provided by *Escardó and Oliva*, expanding on *Escardó*'s previous work [51]. Rather than keeping a list of all previous queries, they directly prune the tree on the fly, by updating the branching function k of `bite` k. Formally, their function looks like this:

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

The Agda code of *Escardó and Oliva*'s proof can be found here.

$$
\begin{array}{lcl}
\texttt{follow} & : & \Pi\{A : \square\}.\, \mathbf{O} \to \mathfrak{B}\ A \to \mathfrak{B}\ A \\
\texttt{follow}\ \_\ (\texttt{spit}\ a) & := & \texttt{spit}\ a \\
\texttt{follow}\ o\ (\texttt{bite}\ k) & := & k\ o \\
 & & \\
\beta' & : & \Pi\{A : \square\}.\, \mathbb{N} \to (\mathbf{O} \to \mathfrak{B}\ A) \to \mathfrak{B}\ A \\
\beta'\ \text{O}\ k & := & \texttt{bite}\ (\lambda o : \mathbf{O}.\ \texttt{follow}\ o\ (k\ o)) \\
\beta'\ (\text{S}\ i)\ k & := & \texttt{bite}\ (\lambda o : \mathbf{O}.\ \beta'\ i\ (\lambda o' : \mathbf{O}.\ \texttt{follow}\ o\ (k\ o'))) \\
 & & \\
\texttt{DtoB}' & : & \Pi\{A : \square\}.\, \mathfrak{D}\ A \to \mathfrak{B}\ A \\
\texttt{DtoB}'\ (\eta\ a) & := & \texttt{spit}\ a \\
\texttt{DtoB}'\ (\beta\ i\ k) & := & \beta'\ i\ (\lambda o : \mathbf{O}.\ \texttt{DtoB}'\ (k\ o))
\end{array}
$$

Its meaning might be a bit harder to grasp than Sterling's version. The intuition is, when facing

$$\beta\, i\, k,$$

by calling `DtoB'` recursively we can get a family of Brouwer trees

$$\lambda o : \mathrm{O}.\ \texttt{DtoB'}\ (k\ o).$$

What we need then is a clever way of mimicking β. Sterling's way of doing it is to store every answer from the oracle in a list. Here,

$$\beta' \qquad \text{and} \qquad \texttt{follow}$$

change the branching function itself, collapsing on the fly the queries we already made. We encourage the reader to look at the formalized proof of correctness for this version, which we find enlightening. ■

2.2.5. Intensional dialogue continuity

All the different definitions of continuity encountered so far are quite extensional, as we only consider outputs of the function

$$F : \mathrm{Q} \to A$$

under scrutiny. We could argue that dialogue and Brouwer continuity are more intensional than standard and sequential continuity, as they provide a concrete, inductive witness of continuity and not just a function waiting for an oracle

$$\alpha : \mathrm{Q}$$

to output a list (in the case of standard continuity) or a result (in the case of sequential continuity). Still, even in dialogue continuity F is only pointwise equal to $\partial\, d$ and we know nothing of the internal computation of F.

In some sense, continuity itself enforces a bit of extensionality. Indeed, the intuitive depiction of continuity is "a function is continuous if it asks a finite number of queries before returning a value", which means that the only way

$$F : \mathrm{Q} \to A$$

can use its argument

$$\alpha : \mathrm{Q}$$

is by providing input to α and computing on its output.

On the other hand, continuity is not a purely extensional notion, as two extensionally equal functions

$$F, G : \mathrm{Q} \to A$$

could have different witnesses of continuity. Still, we can go further down the intentional path and define a more intensional version of continuity.

Definition 2.2.13: Intensional dialogue continuity
A function f validates *intensional dialogue continuity* when the following inductive predicate holds:

$$\frac{a : A}{\eta_{\mathfrak{J}}\ a : \mathscr{C}_{\mathfrak{J}}\ A\ (\lambda\alpha : \mathrm{Q}.\ a)} \qquad \frac{i : \mathrm{I} \qquad k : \mathrm{O}\ i \to \mathrm{Q} \to A \qquad k_\epsilon : \Pi(o : \mathrm{O}\ i).\ \mathscr{C}_{\mathfrak{J}}\ A\ (k\ o)}{\beta_{\mathfrak{J}}\ i\ k\ k_\epsilon : \mathscr{C}_{\mathfrak{J}}\ A\ (\lambda\alpha : \mathrm{Q}.\ k\ (\alpha\ i)\ \alpha)}$$

This definition is equivalent to the following Coq-like syntax:

```
Inductive 𝒞𝔍 (A : □ᵢ) :
  (Q → A) → □ᵢ :=
| η𝔍 : Π (a : A). 𝒞𝔍 A (λx. a)
| β𝔍 : Π (i : I)
       (k : O i → Q → A)
       (kε : Π(o : O i). 𝒞𝔍 A (k o)).
       𝒞𝔍 A (λα. k (α i) α).
```

Perhaps unsurprinsingly, an intensional dialogue continuous function is dialogue continuous.

Lemma 2.2.7: Intensional dialogue is dialogue
Intensional dialogue continuity implies dialogue continuity.

Implication of dialogue continuity by intensional dialogue continuity is formally proved here.

Proof. Direct by induction, using the fact that

$$\mathfrak{D} \qquad \text{and} \qquad \mathscr{C}_{\mathfrak{J}}$$

have the same structure. ∎

Conversely, the decoding of a dialogue tree is intensional dialogue continuous.

Lemma 2.2.8: Intensional continuity of dialogue trees
Let d be a dialogue tree. Then

$$\partial\ d$$

validates intensional dialogue continuity.

Intensional continuity of dialogue trees is formaly proved here.

Proof. By induction on d.

- If
  $$d = \eta\ x$$
  then
  $$\partial\ (\eta\ x) = \lambda\_.\ x$$
  and we conclude by $\eta_{\mathfrak{J}}$.
- If
  $$d = \beta\ i\ k$$
  then we instantiate $\beta_{\mathfrak{J}}$ with i and
  $$\lambda(o : \mathrm{O}\ i).\ \partial\ (k\ o).$$
  The induction hypothesis gives us k_ϵ.

∎

2.3. The zoo of continuity and logical principles

At the end of our stroll in the continuity landscape, we indeed ended up with the map of definitions and implications we hinted at at the beginning of the previous Section. However, this picture depends on the strength of the ambient theory. If we had the full power of classical logic and lived in *ZFC* set theory, for instance, we would end up with the following result:

Proposition 2.3.1: Continuities are classically equivalent
In classical logic, with the axiom of choice and function extensionality, all definitions of continuity from Section 2.2 are equivalent.

This is the point of view of a jackhammer, though, and we can be more subtle. In this Section, we investigate what logical principles are needed to turn the implications of Figure 2.2 into equivalences. At the end, we recover three principles that are sufficient, although maybe not necessary. They are displayed near curvy arrows in Figure 2.2.

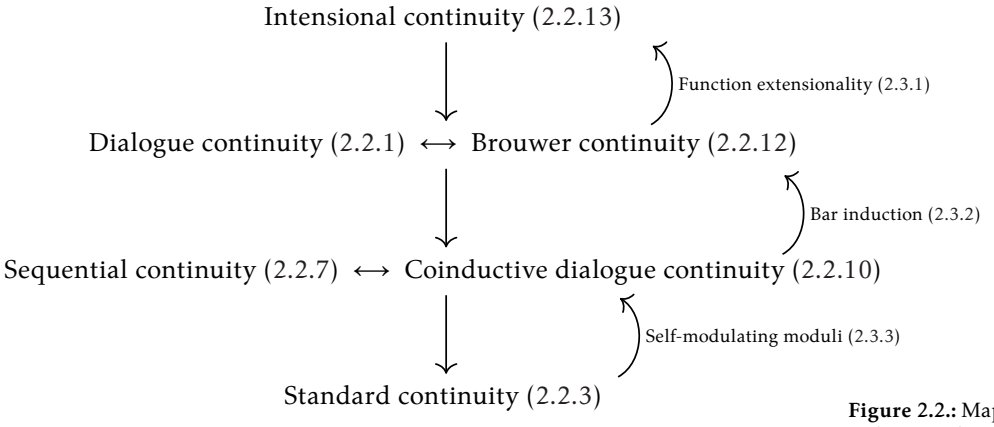


Figure 2.2.: Map of continuity and logical principles

2.3.1. Dialogue continuity is extensionally intensional dialogue continuity

Let us start at the top of the picture, and look at the difference between intensional dialogue continuity and dialogue continuity. As we already noticed, for any dialogue tree d, the function $\partial\, d$ is intensionally dialogue continuous. The difference between intensional and simple dialogue continuity lies in the fact that dialogue continuity only requires pointwise equality between a function and its dialogue witness, while intensional dialogue continuity is a predicate on the function itself. The following result should then be no surprise.

Proposition 2.3.2: Dialogue is extensionally intensional
If the ambient theory features funext for $\mathbb{Q} \to A$ then dialogue continuity implies intensional dialogue continuity.

Implication of intensional dialogue continuity by dialogue continuity and function extensionality is formally proved here.

Proof. Let f be a dialogue continuous function, and let d be its dialogue tree witness. We have

$$\Pi\alpha : \mathbf{Q}.\ f\ \alpha = \partial\ d\ \alpha.$$

Using funext this is enough to derive

$$f = \partial\ d.$$

We conclude with Lemma 2.2.8. ∎

2.3.2. Dialogue trees are barred sequences

In this section, we set

$$\mathbf{I} := \mathbb{N}$$

and $\mathbf{O}$ is taken as an arbitrary type that does not depend on $\mathbf{I}$. We investigate in that setting what logical principles allow to prove dialogue continuity and extensional dialogue continuity to be equivalent. As it turns out, a variant of *bar induction* does the trick.

Following *Brede and Herbelin* [26], we first define *barred* predicates on lists.

[26]: Brede et al. (2021), "On the logical structure of choice and bar induction principles"

Definition 2.3.1: Barred predicate
A predicate

$$T : \mathsf{list}\ \mathbf{O} \to \square$$

is *barred* if the following holds:

$$\Pi(\alpha : \mathbf{Q}).\ \Sigma(l : \mathsf{list}\ \mathbf{O}).(l \sqsubseteq \alpha) \times T\ l,$$

where $l \sqsubseteq \alpha$ means that

$$l \equiv [\alpha\ \mathrm{O}; \ldots; \alpha\ ((\mathtt{len}\ l) - 1)].$$

Otherwise said, every infinite branch has a finite prefix that is in T.

Formal definition of barred predicates, hereditary closure and bar induction can be found here.

A specific subset of barred predicates consists of *inductively barred* predicates.

Definition 2.3.2: Hereditary closure
Given A a type,

$$T : \mathsf{list}\ \mathbf{O} \to \square$$

a predicate on lists and $l : \mathsf{list}\ \mathbf{O}$, T is *hereditary closed at* l, written $T \triangleleft l$, if the following inductive predicate holds:

$$\frac{l : \mathsf{list}\ \mathbf{O} \qquad e : T\ l}{\eta_{\triangleleft}\ e : T \triangleleft l} \qquad \frac{\begin{array}{c} l : \mathsf{list}\ \mathbf{O} \\ l_{\varepsilon} : \Pi(o : \mathbf{O}).\ T \triangleleft (o :: l) \end{array}}{\beta_{\triangleleft}\ l\ l_{\varepsilon} : T \triangleleft l}$$

T is *inductively barred* if it is hereditary closed at the empty list nil.

In Coq-like syntax, this inductive predicates is written like this:

```
Inductive Her (T : list O → □) :
  list O → □ :=
| η◁ : Π(l : list O). T l → Her T l
| β◁ : Π(l : list O).
       (Π(a : O). Her T (a :: l)) →
       Her T l.
```

Lemma 2.3.3: Inductively barred is barred
If a predicate T is inductively barred, then it is barred.

Proof. Direct by induction on the proof of hereditary closure. ∎

The converse is exactly bar induction.

Definition 2.3.3: Bar induction
The axiom of *bar induction* states that any barred predicate T is inductively barred.

Otherwise said, bar induction states that any tree such that any of its branches is finite can be inductively specified. As extensional trees are potentially infinite trees, and since sequential continuity implies that every branch is finite, we can precisely use bar induction to recover a dialogue tree from it.

Proposition 2.3.4: Dialogue is barred extensional dialogue
If bar induction holds, then any function

$$F : \mathbb{Q} \to A$$

that is sequentially continuous is dialogue continuous.

Formal proof that Bar Induction imply equivalence between dialogue and sequential continuity can be found here.

Proof. As it turns out, for our proof to work, we first need to define an extensional analogue to *Brouwer trees*. That is, given

$$F : \mathbb{Q} \to A \qquad \text{and} \qquad \tau : \mathsf{Extree}$$

its witness of sequential continuity, there is some τ' that asks its questions in order. Given such τ', we define the predicate

$$T : \mathsf{list}\ \mathbb{O} \to \square := \lambda l : \mathsf{list}\ \mathbb{O}.\, \Sigma a : A.\, \tau'\ l = \mathtt{ret}\ a.$$

Going from τ to τ' is fairly tedious and technical, and took many lines of Coq code. Its formal proof can be found here.

Unfolding the definition of sequential continuity, we get:

$$\Pi(\alpha : \mathbb{Q}).\, \Sigma(n : \mathbb{N}).\, \partial_{\mathtt{Ext}}\ \tau'\ \alpha\ \mathsf{nil}\ n = \mathtt{ret}\ (F\ \alpha)$$

As it turns out, it is essentially stating that T is barred. Indeed, given

$$\alpha : \mathbb{Q},$$

if $n : \mathbb{N}$ is the first projection of the above statement, it is possible to prove that

$$l := [\alpha\ \mathbb{O}; \ldots; \alpha\ (n-1)]$$

is such that

$$\tau'\ l = \mathtt{ret}\ (F\ \alpha).$$

Hence T is barred with witnesses l and $F\ \alpha$. Applying bar induction, we recover a proof H that T is inductively barred. We recover a Brouwer tree by induction on H. We conclude by equivalence between Brouwer tree continuity and dialogue continuity. ∎

2.3.3. Reflecting on oneself before speaking

In this Section, we set

$$\mathrm{I} := \mathbb{N}$$

and take O independent from I. Moreover, we assume an element

$$o : \mathrm{I}.$$

This particular setting allows us to talk about *continuous ways of being continuous*.

Definition 2.3.4: Moduli of continuity
Let us recall the definition of standard continuity from Section 2.2.1, adapted to the natural number setting:

$$\begin{aligned} \mathscr{C} \quad &: \quad (\mathrm{Q} \to A) \to \square \\ \mathscr{C}\ f \quad &:= \quad \Pi(\alpha : \mathrm{Q}).\, \Sigma l : \mathrm{list}\ \mathrm{O}.\, \Pi(\beta : \mathrm{Q}).\, \alpha \approx_l \beta \to f\ \alpha = f\ \beta. \end{aligned}$$

Given $f : \mathrm{Q} \to A$, from a proof $P : \mathscr{C}\ f$ we can build a function

$$M_f : \mathrm{Q} \to \mathrm{list}\ \mathrm{I} := \lambda \alpha : \mathrm{Q}.\, (P\ \alpha).\pi_1$$

together with a proof

$$\begin{aligned} M_\varepsilon : \Pi(\alpha\ \beta : \mathrm{Q}).\, \alpha \approx_{M_f\ \alpha} \beta \to f\ \alpha = f\ \beta := \\ \lambda \alpha : \mathrm{Q}.\, (P\ \alpha).\pi_2 \end{aligned}$$

Such a function M_f is called a *modulus of continuity* for f.

Moduli of continuity are formally defined here.

Note that sometimes, for a given

$$\alpha : \mathrm{Q}, \qquad \text{the list} \qquad M_f\ \alpha$$

will also be called a modulus of continuity for f on α.

Definition 2.3.5: Self-modulating moduli
Given $f : \mathrm{Q} \to A$ a standard continuous function, given

$$M_f : \mathrm{Q} \to \mathrm{list}\ \mathrm{I}$$

and M_ε a proof that M_f is a modulus of continuity for f, M_f is a *self-modulating modulus* if there is a proof

$$M'_\varepsilon : \Pi(\alpha\ \beta : \mathrm{Q}).\, \alpha \approx_{M_f\ \alpha} \beta \to M_f\ \alpha = M_f\ \beta.$$

Self-modulating moduli are formally defined here.

The *self-modulating modulus* terminology is present in *Konecný and Steinberg* and *Steinberg et al* [91, 129]. Interestingly, self modulating moduli can be used to recover sequential continuity.

[91]: Konecný et al. (2020), "Continuous and monotone machines"
[129]: Steinberg et al. (2021), "Computable analysis and notions of continuity in Coq"

Proposition 2.3.5: Self-modulating standard implies sequential continuity
Let $f : \mathrm{Q} \to A$ be standard continuous, with a modulus M_f that is self-modulating. Then f is sequentially continuous.

Implication of sequential continuity by self-modulating moduli is formaly proved here.

Proof. Given

$$f : \mathbb{Q} \to A$$

a standard continuous function and M_f a self-modulating modulus of continuity for f, we define an extensional tree τ_M as follows: first, given a list

$$l : \mathsf{list}\ \mathbb{O},$$

we define the function

$$\uparrow l : \mathbb{N} \to \mathbb{O} \coloneqq \lambda n : \mathbb{N}.\ \mathtt{if}\ n < \mathtt{len}\ l\ \mathtt{then}\ l.\mathtt{nth}\ n\ \mathtt{else}\ o$$

where o is the inhabitant of $\mathbb{O}$ we assumed at the beginning of the Section. Then we define:

$$\begin{aligned}\tau_M : \mathsf{Extree} \coloneqq \lambda l : \mathsf{list}\ \mathbb{O}.\quad & \mathtt{if}\ \mathtt{max}\ (M_f\ (\uparrow\ l)) \leq (\mathtt{len}\ l)\\ & \mathtt{then}\ \mathtt{ret}\ (f\ (\uparrow\ l))\\ & \mathtt{else}\ \mathtt{ask}\ (\mathtt{len}\ l)\end{aligned}$$

where max here denotes the function that returns the maximum element of a list. To prove that f is sequentially continuous with witness τ_M, we need to show that

$$\Pi(\alpha : \mathbb{Q}).\ \Sigma(n : \mathbb{N}).\ \partial_{\mathtt{Ext}}\ \tau_M\ \alpha\ \mathsf{nil}\ n = \mathtt{ret}\ (f\ \alpha).$$

The way we defined τ_M, given how $\partial_{\mathtt{Ext}}$ computes, if

$$\partial_{\mathtt{Ext}}\ \tau_M\ \alpha\ \mathsf{nil}\ n$$

ever outputs a value, it will be

$$(f\ (\uparrow\ l))$$

for some list l such that

$$l \coloneqq [\alpha\ \mathrm{O}, ..., \alpha(\mathtt{len}\ l)].$$

Moreover, l will be such that

$$\mathtt{max}\ (M_f\ (\uparrow\ l)) \leq (\mathtt{len}\ l),$$

hence

$$\uparrow\ l \approx_{M_f\ (\uparrow\ l)} \alpha$$

and

$$f\ (\uparrow\ l) = f\ \alpha$$

since M_f is a modulus of continuity for f. All that is left is to find an n such that

$$\partial_{\mathtt{Ext}}\ \tau_M\ \alpha\ \mathsf{nil}\ n$$

returns a value. We cannot prove this in general, and it could be so that M_f outputs different lists when called on

$$\uparrow\ [\alpha\ \mathrm{O}] \qquad \text{compared to} \qquad \uparrow\ [\alpha\ \mathrm{O},\ \alpha\ 1],$$

then another one when called on

$$\uparrow\ [\alpha\ \mathrm{O},\ \alpha\ 1, \alpha\ 2],$$

and so on, so much that τ_M never returns a value.

What saves us from the pithole is the fact that M_f itself is continuous. Hence, it cannot keep outputing new values on functions that coincide on bigger and bigger prefixes. Using this argument, and after a bit of technical work, we finally find an l such that

$$\texttt{max}\ (M_f\ (\uparrow\ l)) \leq \texttt{len}\ l$$

and we conclude. ∎

There are some similarities between the proof of this lemma and the definition of a *barred* predicate from Section 2.3.2, and the fact that M is self-modulating allows us to extract what looks like a proof that it is barred. There are actually results about the links between self-modulating moduli and bar induction, as studied for instance by *Fujiwara and Kawai* [55–57, 87], following previous work by *Troelstra* [138]. In particular, *Fujiwara and Kawai* [57] prove the following result:

Proposition 2.3.6: Continuous moduli are self-modulating
Given $f : \mathrm{Q} \to A$ a standard continuous function with a modulus M_f. If M_f is itself continuous, then it is self modulating.

Their result holds in

$$\mathsf{HA}^\omega + \mathsf{QF} - \mathsf{AC}^{1,0}$$

as defined in *Troelstra* [139], meaning intuitionistic arithmetic at all finite types, together with the quantifier-free axiom of choice of degrees 1 and 0, *i.e.*

$$\mathsf{QF} - \mathsf{AC}^{1,0} \coloneqq \forall x : \sigma. \exists y : \tau. A(x,y) \to \exists f : \sigma \to \tau. A(x, f\ x)$$

where A is quantifier-free, σ is of degree at most 1 (meaning

$$\sigma \coloneqq \mathbb{N} \to \mathbb{N}$$

at most) and τ of degree 0.

This means that if we were to build a model where every function of the form

$$f : (\mathrm{I} \to \mathrm{O}) \to A$$

is standard continuous, we would immediately get that every modulus M_f of continuity for f would be itself continuous. Then this lemma entails that M_f would be self-modulating and, by Lemma 2.3.5, this would mean that every function of the form

$$f : (\mathrm{I} \to \mathrm{O}) \to A$$

is sequentially continuous. We leave the study of the translation of *Fujiwara and Kawai*'s theorem to Coq for future work.

[55]: Fujiwara et al. (2021), "Characterising Brouwer's continuity by bar recursion on moduli of continuity"
[56]: Fujiwara et al. (2021), "Decidable fan theorem and uniform continuity theorem with continuous moduli"
[57]: Fujiwara et al. (2019), "Equivalence of bar induction and bar recursion for continuous functions with continuous moduli"
[87]: Kawai (2019), "Principles of bar induction and continuity on Baire space"
[138]: Troelstra (1988), "Constructivism in mathematics"
[57]: Fujiwara et al. (2019), "Equivalence of bar induction and bar recursion for continuous functions with continuous moduli"
[139]: Troelstra (1973), *Metamathematical investigation of intuitionistic arithmetic and analysis*

2.4. The Continuous Hypothesis

We now have a clearer picture of the different continuity definitions that can be found in the realm of type theory, and the technical subtleties needed to compare them. We are however yet to answer two basic questions:

1. What kind of functions do we expect to be continuous? For which I, which O, which A do we mean to build models where every function is continuous?
2. How useful a principle continuity of all functions even is?

These two questions are of course linked: if we find out that some functions are not continuous, then continuity of all functions will not be relevant as a principle since it will entain inconsistency of our theory. In this Section we will mainly tackle the latter question. We will argue that continuity is an observable effect in the sense of Section 1.4, and that, when instantiated with well-chosen I and O, it can help recover semi-classical principles. Chapter 3 and Chapter 4 will focus on models of dependent type theory where every function is dialogue continuous, once again for some specific I, O and A.

2.4.1. Continuity is a classic

Assuming we live in a perfect world where every function at every type is dialogue continuous, what would we gain from this? In that kind of setting, recovering the whole power of classical logic would be no sweat.

Proposition 2.4.1: Continuity implies double-negation elimination
Let P be a type, and F a proof of $\neg\neg P$. If F is continuous in the dialogue sense, then there is a proof of P.

Implication of DNE by continuity is formally proved here.

Proof. Let us assume that F is continuous in the dialogue sense. That is, there is a dialogue tree

$$d : \mathfrak{D}\,\bot, \qquad \text{with} \qquad \mathbf{I} \coloneqq P, \qquad \mathbf{O} \coloneqq \lambda\_.\,\bot \qquad \text{and} \qquad A \coloneqq \bot,$$

such that

$$\Pi(\mathfrak{f} : P \to \bot).\, F\,\mathfrak{f} = \partial\, d\,\mathfrak{f}.$$

Let us extract a witness of P from d. We proceed by induction on d:

- If $d \coloneqq \eta\, a$ with $a : \bot$, then we conclude by absurdity.
- If $d \coloneqq \beta\,(i : P)\,(k : \bot \to \mathfrak{D}\,\bot)$ then we simply return $i : P$.

Note that the same proof would work with coinductive dialogue continuity. Indeed, we are not doing a proof by induction on d here, but simply a case analysis on the head constructor, hence the difference

between inductive and coinductive does not matter. It would however not work with standard continuity, as we would have to provide

$$\mathfrak{f} : P \to \bot$$

before using the fact that F is continuous.

We emphasize that this proof is internal to the theory here. Otherwise said, if we were able to build a model validating the axiom

$$\vdash^{\mathcal{S}} \mathsf{All}_{\mathscr{C}} : \Pi\, \mathbf{I}\, \mathbf{O}\, A\, (F : (\Pi i : \mathbf{I}.\, \mathbf{O}\, i) \to A).\, \mathscr{C}_{\mathfrak{D}}\, F,$$

we would be able to internally inhabit

$$\vdash^{\mathcal{S}} \mathsf{DNE} : \Pi A.\, \neg\neg A \to A.$$

2.4.2. Absurdly continuous

If we could build a model that computed a proof of continuity at every type of the form

$$(\Pi i : \mathbf{I}.\, \mathbf{O}\, i) \to A,$$

our grand goal of giving computational content to DNE would hence be within reach. This is too good to be true, though, and the world always hits hard delusional daydreamers who mistake their hopes for reality. In our case, reality strikes back through *Escardó and Xu*'s hand *[52]*, in the form of an inconsistency result.

[52]: Escardó et al. (2015), "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation"

Proposition 2.4.2: Continuity on the Baire space is inconsistent Assuming that every function

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

is continuous in the standard sense is inconsistent in MLTT.

Inconsistency of continuity of all functions

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

is formally proved here.

Proof. The proof of *Escardó and Xu* makes use of the standard version of continuity, albeit with a natural number instead of a list. That is, given

$$\alpha, \beta : \mathbb{N} \to \mathbb{N},$$

they write

$$\alpha \approx_n \beta$$

to mean that α and β coincide on the first n natural numbers. This definition is equivalent to Definition 2.2.3. To go from the natural number definition to the list one, we simply by take

$$l : \mathsf{list}\, \mathbb{N}$$

to be made of the first n natural numbers. Conversely, by taking as n the maximum element of

$$l : \mathsf{list}\, \mathbb{N},$$

we can go from the list definition to the natural number one.

Then, assuming

$$\begin{aligned}&\mathfrak{H} : \Pi\ (f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N})\ (\alpha : \mathbb{N} \to \mathbb{N}).\\ &\qquad \Sigma(n : \mathbb{N}).\, \Pi(\beta : \mathbb{N} \to \mathbb{N}).\, \alpha \approx_n \beta \to f\ \beta = f\ \alpha\end{aligned}$$

they derive $0 = 1$.

To do this, they first define

$$0^\omega := \lambda(n : \mathbb{N}).\ 0$$

the infinite sequence of 0 and

$$0^n k^\omega := \lambda(m : \mathbb{N}).\texttt{if}\ m < n\ \texttt{then}\ 0\ \texttt{else}\ k$$

the sequence of n many zeros followed by infinitely many k.

By projection of $\mathfrak{H}$ they recover

$$\begin{aligned}&M : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N} \qquad \text{and}\\ &M_\epsilon : \Pi(f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N})\ (\beta : \mathbb{N} \to \mathbb{N}).\ 0^\omega =_{M\ f} \beta \to f\ \beta = f\ 0^\omega\end{aligned}$$

The proof is a diagonalization argument, making use of M to build a function

$$f \qquad \text{such that} \qquad M\ f$$

cannot be a modulus of continuity for f, which leads to absurdity.

They start by defining

$$f := \lambda\alpha.\, M\ (\lambda\beta.\, \alpha\ (\beta\ m)) \qquad \text{where} \qquad m := M\ (\lambda\alpha.\ 0)$$

This definition leads to two important properties. First,

$$f\ 0^\omega = M\ (\lambda\beta.\ 0^\omega\ (\beta m)) = M\ (\lambda\beta.\ 0) = m$$

Then, using this equality and M_ϵ, they prove:

$$\Pi(\beta : \mathbb{N} \to \mathbb{N}).0^\omega =_{M\ f} \beta \to f\ \beta = m.$$

Escardó and Xu then notice that, given any fixed $\alpha : \mathbb{N} \to \mathbb{N}$,

$$f\ \alpha = M(\lambda\beta.\ \alpha\ (\beta\ m))$$

is itself a modulus of continuity for

$$\lambda\beta.\ \alpha\ (\beta\ m)$$

Using M_ϵ, this means that

$$\Pi(\beta : \mathbb{N} \to \mathbb{N}).\, 0^\omega =_{f\ \alpha} \beta \to \alpha\ (\beta\ m) = \alpha\ 0$$

The idea is then to pick suitable α and β and make use of that last property to derive

$$0 = 1.$$

They choose

$$\alpha := 0^{M\ f+1}1^\omega \qquad \text{and} \qquad \beta := 0^m(M\ f + 1)^\omega.$$

These two definitions validate the conditions of the moduli of continuity: first,

$$\alpha =_{M\ f} 0^\omega$$

which means that

$$f\ \alpha = f\ 0^\omega = m.$$

Then,

$$\beta =_{f\ \alpha} 0^\omega \qquad \text{which means that} \qquad \alpha\ (\beta\ m) = \alpha\ 0.$$

Expanding the definitions, they conclude:

$$0 = \alpha\ 0 = \alpha\ (\beta\ m) = 0^{Mf+1}1^\omega\ (0^m(Mf+1)^\omega\ m) = 0^{Mf+1}1^\omega\ (Mf+1) = 1$$

hence continuity of all functionals from the Baire space to natural numbers is inconsistent in MLTT. ∎

However, as *Escardó and Xu* point out, this proof works because continuity is expressed as a proof-relevant statement, with the use of Σ-type. Indeed, if we add propositional truncation to MLTT, then continuity of all functions

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

expressed propositionally is consistent with the resulting system. It is for instance validated by *Johnstone's topological topos model* [84]. Somehow we are walking a very thin line here and

$$(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

looks like a plausible border between consistent and inconsistent continuity axioms.

[84]: Johnstone (1979), "On a topological topos"

2.4.3. The Shift Project

Definition 2.4.1: Double-negation shift
The *double-negation shift principle* (DNS) states the following:

$$\mathsf{DNS} :\equiv \Pi(\mathbf{I} : \square_0)\,(\mathbf{O} : \mathbf{I} \to \square_0).\,(\Pi(i : \mathbf{I}).\,\neg\,\neg(\mathbf{O}\ i)) \longrightarrow \neg\,\neg(\Pi(i : \mathbf{I}).\,\mathbf{O}\ i)$$

Given $\mathbf{I}$ and $\mathbf{O}$, the double-negation shift principle is intuitionnistically equivalent to the following:

$$\neg(\Pi(i : \mathbf{I}).\,\mathbf{O}\ i) \longrightarrow \neg(\Pi(i : \mathbf{I}).\,\neg\,\neg(\mathbf{O}\ i)).$$

Once again, the first type is of the form

$$(\Pi(i : \mathbf{I}).\,\mathbf{O}\ i) \to A$$

and we can consider continuity over proofs of this type.

Proposition 2.4.3: Continuity implies DNS
Let us assume that we have a functional

$$F : \neg(\Pi(i : \mathbf{I}).\, \mathbf{O}\, i)$$

continuous in the dialogue sense. Then there is a proof of

$$\neg(\Pi(i : \mathbf{I}).\, \neg\, \neg(\mathbf{O}\, i)).$$

Implication of DNS by continuity is formally proved here.

Proof. Let us assume

$$F : (\Pi(i : \mathbf{I}).\, \mathbf{O}\, i) \to \bot$$

continuous in the dialogue sense. Then there exists a dialogue tree

$$d : \mathfrak{D}\, \mathbf{I}\, \mathbf{O}\, \bot$$

such that

$$\Pi(\mathfrak{f} : \Pi(i : \mathbf{I}).\, \mathbf{O}\, i).\, F\, \mathfrak{f} = \partial\, d\, \mathfrak{f}.$$

Let us now take a function

$$f : \Pi(i : \mathbf{I}).\, \neg\, \neg(\mathbf{O}\, i)$$

and prove falsity. We proceed by induction on d:

1. If
$$d \coloneqq \eta\, a \qquad \text{with} \qquad a : \bot,$$
then we simply return a.
2. If
$$d = \beta\, (i : \mathbf{I})\, (k : \mathbf{O}\, i \to \mathfrak{D}\, \mathbf{I}\, \mathbf{O}\, \bot)$$
with the induction hypothesis
$$H : \mathbf{O}\, i \to \bot,$$
specializing f to i we get a term
$$f\, i : \neg\, \neg(\mathbf{O}\, i).$$
By applying it to H, we get the desired absurdity.

Interestingly, through double-negation translation the axiom of choice is precisely interpreted as DNS. This is why, in classical realizability [106, 107], realisers of the axiom of choice are exactly realisers of DNS, as can be seen in *Miquey* [108]. Furthermore, the memoization technique used by *Miquey* in the latter is similar to the one used by *Rahli et al* [36, 123] to enforce continuity principles.

[106]: Miquel (2011), "A Survey of Classical Realizability"
[107]: Miquel (2011), "Forcing as a Program Transformation"
[108]: Miquey (2019), "A constructive proof of dependent choice in classical arithmetic via memoization"
[36]: Cohen et al. (2023), "Realizing Continuity Using Stateful Computations"
[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"

2.5. Sheaves and ShTT

One last facet of $\mathfrak{D}$ is its link to sheaves. Indeed, in a note later turned into a draft, *Pédrot* [113, 114] gives a type theoretic account of the internal model of sheaves in the presheaf topos. Unfolding the definitions, it turns out that the sheafification operator is surprisingly close to the dialogue monad. Moreover, we believe this type theoretic unravelling may help dispel the fog around what can constitute, from a computer scientist's point of view, a rather misty concept. For both reasons, we take the liberty of providing here a reformulated plagiarism of *Pédrot*'s note.

[113]: Pédrot (2021), "Debunking Sheaves"
[114]: Pédrot (2023), "Pursuing Shtuck"

For the remainder of this Section, we fix a small category $\mathbb{C}$ that we call the *base category*.

2.5.1. Set setting

We recall in this section the usual definitions of presheaves and sheaves in a set-theoretic metatheory [101]. Let us first define *presheaves*.

Definition 2.5.1: Presheaves
A *presheaf* on $\mathbb{C}$ is a functor of type

$$\mathbb{C}^{\mathrm{op}} \to \mathrm{Set},$$

where Set is the category of sets.

Theorem 2.5.1: Presheaf topos
The category of presheaves on $\mathbb{C}$, with natural transformations as morphisms, forms a topos.

This essentially means that presheaves feature an *internal language* able to interpret most constructions of type theory. We will write

$$\mathrm{Pshf}(\mathbb{C})$$

to denote the presheaf topos.

Definition 2.5.2: Sieves
Given p an object of $\mathbb{C}$, a *sieve* on p is a set of arrows with codomain p closed under precomposition.

In the following, we will write $\mathfrak{s}_p$ for the set of sieves on p.

Let us now set our eyes on *Grothendieck topologies*.

Definition 2.5.3: Grothendieck topology
A *Grothendieck topology* $\mathfrak{T}$ on $\mathbb{C}$ is a $\mathbb{C}$-indexed family of collections of sieves, meaning

$$\Pi p \in \mathbb{C}.\, \mathfrak{T}_p \subseteq \mathfrak{s}_p,$$

which satisfy the following properties:

- the maximal sieve on p is in $\mathfrak{T}_p$;
- if $P \in \mathfrak{T}_p$ and $\alpha \in \mathrm{Hom}_{\mathbb{C}}(q, p)$ then the pullback of P along α, noted $\alpha^* P$, is in $\mathfrak{T}_p$;
- if $P \in \mathfrak{T}_p$, if $Q \in \mathfrak{s}_p$ and if

$$P \subseteq \bigcup_{q \in \mathbb{C}} \{(q, \alpha) \mid \alpha \in \mathrm{Hom}_{\mathbb{C}}(q, p) \wedge \alpha^* Q \in \mathfrak{T}_q\},$$

then $Q \in \mathfrak{T}_p$.

For the remainder of this section, we fix a Grothendieck topology $\mathfrak{T}$.

Definition 2.5.4: Compatible families
Let $p \in \mathbb{C}$, $P \in \mathfrak{s}_p$ and A a presheaf on $\mathbb{C}$. A *compatible family of A on P* is given by:

- for every $(q, \alpha) \in P$, an element $x_{q,\alpha} \in A_q$;
- for any $(q, \alpha) \in P$, for any $\beta \in \mathrm{Hom}_{\mathbb{C}}(r, q)$, a proof that

$$A[\beta]\, x_{q,\alpha} = x_{r,\beta \circ \alpha},$$

where

$$A[\beta]$$

is the morphism in Set obtained by applying A to β.

Definition 2.5.5: Sheaves
A presheaf A on $\mathbb{C}$ is a $\mathfrak{T}$-sheaf if for any $p \in \mathbb{C}$, for any $P \in \mathfrak{T}_p$ and any compatible family x of A on P, there exists a unique element

$$\hat{x} \in A_p \qquad \text{such that} \qquad \Pi(q, \alpha) \in P.\, A[\alpha]\, \hat{x} = x_{q,\alpha}.$$

Theorem 2.5.2: Sheaf topos
The full subcategory of $\mathfrak{T}$-sheaves on $\mathbb{C}$ forms a topos.

We will write

$$\mathrm{Shf}(\mathbb{C}, \mathfrak{T})$$

to denote the sheaf topos.

Interestingly, the sheaf topos can be described in the internal language of the presheaf topos, meaning the presheaf topos can be taken as a target theory for an interpretation of the sheaf topos. As the presheaf topos itself is interpreted in set theory, this means that the historical sheaf model can be factorized in two:

$$\mathrm{Shf}(\mathbb{C}, \mathfrak{T}) \xrightarrow{\text{internal sheaf model}} \mathrm{Pshf}(\mathbb{C}) \xrightarrow{\text{presheaf model}} \mathrm{Set}$$

Actually, the internal sheaf model can be described in the internal language of *any* topos, and not only the presheaf one. In Section 2.5.2, we will give a description of the internal sheaf model in an arbitrary type theory.

Theorem 2.5.3: Free sheaves
The inclusion of the category of sheaves on $\mathbb{C}$ into the category of presheaves on $\mathbb{C}$ has a left adjoint, called sheafification.

This adjonction gives rise to a monad, which we will see is very close to the dialogue one.

2.5.2. Type setting

We describe once again from scratch the sheaf construction from Section 2.5.1. However, this time we choose the internal sheaf model route, and switch to type theory. This actually means that we will work in a type theory named

$$\mathsf{PshTT}$$

and recover concepts from Section 2.5.1 through a syntactic translation. The target type theory of this translation will be an extension of CIC dubbed

$$\mathsf{SetTT}.$$

It is basically CIC enhanced with quotient types, and where equality is interpreted as a proposition. All in all, SetTT can be interpreted inside any presheaf topos. *Observational Type Theory* [8, 119, 120] would provide everything we need, and a bit more, since only a few lemmas will require proof irrelevance or function extensionality. We will explicitly say so when it is the case.

[8]: Altenkirch et al. (2007), "Observational equality, now!"
[119]: Pujet et al. (2023), "Impredicative Observational Equality"
[120]: Pujet et al. (2022), "Observational Equality: Now for Good"

To make things clearer, let us consider the translation $[\![\_]\!]$ of types from PshTT to SetTT. A type

$$\vdash^{\mathsf{PshTT}} A : \square$$

is translated as a presheaf

$$[A] \coloneqq ([\![A]\!], \theta_A)$$

over $\mathbb{C}$ in SetTT, where $[\![A]\!]$ is the action on objects and θ_A the action on morphisms. Their types are the following:

$$\begin{array}{lcl} [\![A]\!] & : & \Pi(p : \mathbb{C}).\, \square_i \\ \theta_A & : & \Pi(p\, q : \mathbb{C}).\, q \leq p \to [\![A]\!]\ p \to [\![A]\!]\ q \end{array}$$

where $q \leq p$ is a shorthand for $\mathsf{Hom}_{\mathbb{C}}(q, p)$.

This comes with naturality conditions on θ_A but we do not delve in it here. Note that we can replace $\square_i$ with `Prop` in this definition to recover what we call *propositional presheaves*, which will be the translation of propositions.

We then have an $\mathtt{El}$ function defined as follows:

$$\mathtt{El}\ (A, \theta_A) \coloneqq \left\{ \begin{array}{l} x : \Pi(p : \mathbb{C}).\, A\ p \quad ; \\ \_ : \Pi(p\ q : \mathbb{C})\,(\alpha : q \leq p).\, \theta_A\ \alpha\ (x\ p) = x\ q. \end{array} \right\}$$

where $\{ \ \ldots\ ;\ \ldots\ \}$ is a notation for records, which constitute a more readable way of writing

$$\Sigma(x : \Pi(p : \mathbb{C}).\, A\ p).\, \Pi(p\ q : \mathbb{C})\,(\alpha : q \leq p).\, \theta_A\ \alpha\ (x\ p) = x\ q.$$

The $\mathtt{El}$ function essentially maps a presheaf to the type of its *global elements*. In the case of $\mathtt{Prop}$, we have:

$$[\![\mathtt{Prop}]\!]_p \coloneqq \left\{ \begin{array}{l} \mathtt{typ} : \Pi(q : \mathbb{C})\,(\alpha : q \leq p).\, \mathtt{Prop} \quad ; \\ \mathtt{hom} : \Pi(q\ r : \mathbb{C})\,(\alpha : q \leq p)\,(\beta : r \leq q). \\ \qquad \mathtt{typ}\ q\ \alpha \to \mathtt{typ}\ r\ (\beta \circ \alpha) \end{array} \right\}$$

As for $\theta_{\mathtt{Prop}}$, it is defined as follows:

$$\theta_{\mathtt{Prop}}\ p\ q\ (\alpha : q \leq p)\ (A, \theta_A) \coloneqq \left(\begin{array}{l} \lambda(r : \mathbb{C})\,(\beta : r \leq q).\, A\ r\ (\beta \circ \alpha) \quad , \\ \lambda(r\ s : \mathbb{C})\,(\beta : r \leq q)\,(\gamma : s \leq r)\,(x : A\ r\ (\beta \circ \alpha)).\, \theta_A\ r\ s\ (\beta \circ \alpha)\ \gamma\ x \end{array} \right)$$

This definition of $[\![\mathtt{Prop}]\!]$ leads to the following result:

Lemma 2.5.4: Propositions as sieves
The type of global elements of $[\![\mathtt{Prop}]\!]$ is isomorphic to the type of propositional presheaves, which means that a proposition

$$\vdash^{\mathsf{PshTT}} A : \mathtt{Prop}$$

is translated in SetTT to a propositional presheaf.

Moreover, given $p : \mathbb{C}$, a sieve on p is simply an element of $[\![\mathtt{Prop}]\!]_p$.

Proof. The first isomorphism is straightforward. For the second part, recall that a sieve on p is a set of arrows with codomain p that is closed under precomposition. Then, looking at

$$[\![\mathtt{Prop}]\!]_p,$$

we see that

$$\mathtt{typ} : \Pi(q : \mathbb{C})\,(\alpha : q \leq p).\, \mathtt{Prop}$$

can be seen as a set of arrows with codomain p, while

$$\mathtt{hom} : \Pi(q\ r : \mathbb{C})\,(\alpha : q \leq p)\,(\beta : r \leq q).\, \mathtt{typ}\ q\ \alpha \to \mathtt{typ}\ r\ (\beta \circ \alpha)$$

describes the closure under precomposition. ∎

It is now time to describe another kind of topology.

Definition 2.5.6: Lawvere-Tierney topology
A *Lawvere-Tierney topology* is a triple

- $\mathfrak{T} : \mathtt{Prop} \to \mathtt{Prop}$;
- $\eta^{\mathfrak{T}} : \Pi(P : \mathtt{Prop}).\ P \to \mathfrak{T}\ P$;
- $\mathtt{bind}^{\mathfrak{T}} : \Pi(P\ Q : \mathtt{Prop}).\ (P \to \mathfrak{T}\ Q) \to \mathfrak{T}\ P \to \mathfrak{T}\ Q$.

Let us already state the obvious: Lawvere-Tierney topologies are exactly monads on `Prop`. Interestingly, once translated into SetTT, they coincide with Grothendieck topologies.

Lemma 2.5.5: Lawvere-Tierney is Grothendieck
Lawvere-Tierney topologies in PshTT are translated as Grothendieck topologies in SetTT.

Moreover, assuming proposition extensionality in SetTT, Lawvere-Tierney topologies exactly capture Grothendieck topologies in the presheaf translation.

Proof. To prove this, we first need to describe the presheaf translation of functions. Up to isomorphism, we have:

$$\mathtt{El}\,[A \to B] \simeq \left\{ \begin{array}{l} f : \Pi(p : \mathbb{C}).\ [\![A]\!]_p \to [\![B]\!]_p \quad ; \\ \_ : \Pi(p\ q : \mathbb{C})\,(\alpha : q \leq p)\,(x : [\![A]\!]_p).\ \theta_B\ \alpha\ (f\ p\ x) = f\ q\ (\theta_A\ \alpha\ x) \end{array} \right\}$$

Then, given $(\mathfrak{T}, \eta^{\mathfrak{T}}, \mathtt{bind}^{\mathfrak{T}})$, given $p : \mathbb{C}$, we have

$$[\mathfrak{T}].\pi_1 : \Pi(p : \mathbb{C}).\ [\![\mathtt{Prop}]\!]_p \to [\![\mathtt{Prop}]\!]_p.$$

From this, we can extract a function

$$\lambda p\,(P : [\![\mathtt{Prop}]\!]_p).([\mathfrak{T}].\pi_1\ p\ P).\mathtt{typ}\ p\ \mathtt{id}_p : \Pi(p : \mathbb{C})\,(P : [\![\mathtt{Prop}]\!]_p).\ \mathtt{Prop}$$

which is exactly a $\mathbb{C}$-indexed collection of sieves on p, as required by the Grothendieck topology definition.

The return $\eta^{\mathfrak{T}}$ of the $\mathfrak{T}$ monad asserts that any inhabited proposition P is sent to an inhabited proposition $\mathfrak{T}\ P$. Assuming proposition extensionality, which says that equivalent propositions are equal, we can prove that any inhabited proposition is equal to $\top$. In that case, the return of the monad can be reformulated by simply asking that $\mathfrak{T}\ \top$ is inhabited, leading to the maximal sieve condition. Without proposition exensionality, we can still prove that $\eta^{\mathfrak{T}}$ implies that the maximal sieve is in $\mathfrak{T}$, but not the converse.

Finally, the pullback requirement is reflected by

$$[\mathfrak{T}].\pi_2 : \Pi p\ q\,(\alpha : q \leq p)\,(x : [\![\mathtt{Prop}]\!]_p).\ \theta_{\mathtt{Prop}}\ \alpha\ (f\ p\ x) = f\ q\ (\theta_{\mathtt{Prop}}\ \alpha\ x)$$

while $\mathtt{bind}^{\mathfrak{T}}$ captures the $\bigcup$ condition. ∎

It turns out that compatible families also have a direct interpretation in PshTT.

Lemma 2.5.6: Compatible families are `fun`
Given

$$\vdash^{\mathsf{PshTT}} A : \square_i$$

translated as a presheaf (A, θ_A) in SetTT, given

$$p : \mathbb{C} \qquad \text{and} \qquad P : [\![\mathtt{Prop}]\!]_p$$

a sieve on p, a compatible family x of A on P is simply an element of the local functional type $[\![P \to A]\!]_p$.

Proof. Once again, the first projection of

$$x : [\![P \to A]\!]_p$$

gives the matching family itself, while the second projection is equivalent to the naturality condition. ∎

Note that we did a slight abuse of notations here: since

$$P : [\![\mathtt{Prop}]\!]_p$$

is only a local element of $[\![\mathtt{Prop}]\!]$, it cannot be reflected in PshTT, thus $[\![P \to A]\!]$ does not really exist. It only makes sense because we consider a local element of $[\![P \to A]\!]_p$.

Definition 2.5.7: Is sheaf
Given $\mathfrak{T}$ a Lawvere-Tierney topology, the predicate isSh is defined as the following record type:

$$\mathsf{isSh}_{\mathfrak{T}}\ (A : \square_i) : \square_i \coloneqq \left\{ \begin{array}{l} \beta : \Pi(P : \mathtt{Prop}).\, \mathfrak{T}\ P \to (P \to A) \to A \\ \varepsilon : \Pi P\, (p : \mathfrak{T}\ P)\, (x : A).\, \beta\ P\ p\ (\lambda\_.\ x) = x \end{array} \right\}$$

When there is a proof of $\mathsf{isSh}_{\mathfrak{T}}\ A$, we will say that A is a $\mathfrak{T}$-sheaf. This name is justified by the following results:

Lemma 2.5.7: One way to be a sheaf
Assuming function extensionality and proof irrelevance in PshTT, there is a proof

$$\vdash^{\mathsf{PshTT}} \_ : \Pi(A : \square_i)\, (p\ q : \mathsf{isSh}_{\mathfrak{T}}\ A).\, p = q.$$

Otherwise said, isSh A is a *mere proposition*.

Theorem 2.5.8: Is sheaf is sheaf
Let $A : \square_i$ in PshTT. There is a proof

$$\vdash^{\mathsf{PshTT}} \_ : \mathsf{isSh}_{\mathfrak{T}}\ A$$

exactly when, once translated in SetTT, A is a $\mathfrak{T}$-sheaf in the sense of Definition 2.5.5.

Proof. A sieve on p is a local element

$$P : [\![P]\!]_p.$$

The fact that it is in $\mathfrak{T}_p$ is interpreted by the fact that $[\![\mathfrak{T}\ P]\!]_p$ is inhabited. The compatible family is once again a local element

$$x : [\![P \to A]\!]_p,$$

and unicity of $\hat{x}$ makes sure that we have a function. All in all, this is captured by the type of β through the presheaf translation. As for naturality conditions, they are equivalent to ε. ∎

Looking at $\mathrm{isSh}_{\mathfrak{T}}$, let us notice that up to curryfication, the type of β is equivalent to

$$\Pi(H : \Sigma P : \mathtt{Prop}.\, \mathfrak{T}\ P).\ (H.\pi_1 \to A) \to A.$$

Then, if we set

$$\mathrm{I} \coloneqq \Sigma P : \mathtt{Prop}.\, \mathfrak{T}\ P \qquad \text{and} \qquad \mathrm{O}\ i \coloneqq i.\pi_1,$$

we recover

$$\beta : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to A) \to A,$$

which should ring a bell.

We can generalize isSh to arbitrary I and O and get:

$$\mathrm{isSh}\ (A : \square_i) : \square_i \coloneqq \left\{ \begin{array}{l} \beta : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to A) \to A \\ \varepsilon : \Pi(i : \mathrm{I})(x : A).\ \beta\ i\ (\lambda\_.\ x) = x \end{array} \right\}$$

Recall that we assumed the existence of quotient types. This means we can tweak our usual dialogue operator and recover another monad.

Definition 2.5.8: Efficient dialogues
The sheafification operator $\mathfrak{S}$ is inductively defined as follows:

$$\begin{array}{l} \mathtt{Inductive}\ \mathfrak{S}\ (A : \square_i) : \square_i := \\ \mid \eta : A \to \mathfrak{S}\ A \\ \mid \beta : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to \mathfrak{S}\ A) \to \mathfrak{S}\ A \\ \mid \varepsilon : \Pi(i : \mathrm{I})\ (x : \mathfrak{S}\ A).\beta\ i\ (\lambda\_ : \mathrm{O}\ i.\ x) = x. \end{array}$$

Intuitively, elements of $\mathfrak{S}\ A$ are dialogue trees where every *pointless questions* have been expunged. Moreover, sheaves are easily captured by this monad.

Proposition 2.5.9: Sheaves as algebras
Sheaves are isomorphic to algebras of the $\mathfrak{S}$ monad.

Actually, sheaves are the $\mathfrak{S}$ equivalent of *intensional algebras* as described in Section 2.1.2.

Part II.

Present

3. Gardening with the Pythia

Having studied the dialogue operator and glimpsed the different definitions of continuity in Chapter 2, it is now time to try and build a continuous world, an ideal place where every function quietly relaxes by the shadow of its own dialogue tree. However, when the time comes to lay the first stone of our continuous utopia, a question arises: where do we start?

Assuming we pick λ-calculus as our favourite computational system, a modern proof would boil down to building a semantic model, typically some flavour of complete partial orders (cpos). By construction, cpos are a specific kind of topological spaces, and all functions are interpreted as continuous functions in the model. For some simple enough types, cpo-continuity implies continuity in the standard sense, thus proving the claim.

Instead of going down the semantic route, *Escardó* made good use of the dialogue operator and developed an alternative syntactic technique called *effectful forcing* [51] to prove the continuity of all functionals

$$\vdash^{\mathsf{T}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

definable in System T. While semantic models such as cpos are historically defined inside a non-computational metatheory (although recent work such as *Tom de Jong*'s PhD [85] try and provide constructive foundations to cpos), *Escardó*'s technique amounts to building a model of System T inside the dependent type theory MLTT, which is intrinsically a programming language with a built-in notion of computation. The *effectful* epithet is justified by the fact that the model construction extends System T with two different kinds of side-effects, and constrains those two extensions by a logical relation.

A clear advantage of this approach is that there is a clear computational explanation for why continuity holds in terms of elementary side-effects, which is not immediately apparent in cpos. This computational aspect is reminiscent of similar realizability models built by *Cohen et al* [36, 123], internalizing continuity with various side-effects. But contrarily to the latter, the purely syntactic nature of *Escardó*'s argument can actually be leveraged to interpret much richer languages than System T while preserving desirable properties that would be lost with a semantic realizability model, such as decidability of typechecking.

Indeed, it happens that this technique can be formulated pretty much straightforwardly as a program translation as presented in Section 1.5, a point we discuss in Section 3.1. From this initial observation, we show in this Chapter how *Escardó*'s argument can be generalized to *Baclofen Type Theory* (BTT) [121], a rich dependent type we displayed in Section 1.4. Unfortunately, as we pointed out in the same Section, since *Escardó*'s model introduces observable side-effects in the sense of *Pédrot and Tabareau* [116], we cannot interpret large dependent elimination in our model, thus MLTT is out of reach.

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

[85]: Jong (2023), "Domain Theory in Constructive and Predicative Univalent Foundations"

[36]: Cohen et al. (2023), "Realizing Continuity Using Stateful Computations"
[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

[116]: Pédrot et al. (2020), "The fire triangle: how to mix substitution, dependent elimination, and effects"

The model is given by three program translations, each bringing its own computational effect: the *axiom translation*, that adds an oracle to the context; the *branching translation*, based on the dialogue monad, turning every type into a tree; and finally, a layer of *algebraic binary parametricity*, binding together the two translations. Since this construction is a bit of a mouthful, we start in Section 3.1 by rephrasing *Escardó*'s model as a program translation from System T to MLTT. This helps us point out some quirks of the model that will become problematic when enhancing the interpretation to dependent types.

We then present our model of BTT in Section 3.2, replaying *Escardó*'s construction in a pedestrian way, in the context of dependent types.

We prove continuity of BTT-definable functionals in Section 3.3. In the end we recover the continuity result of *Escardó* applied to BTT rather than System T. That is, from any

$$\vdash^{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N},$$

we get a proof that it is dialogue continuous. This proof however lives in the meta-theory: we call it *external continuity*, as opposed to *internal continuity*, where it would be reflected in BTT as a term

$$\vdash^{\mathsf{BTT}} H : \Pi(f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}).\ \mathscr{C}_{\mathfrak{D}}\ f.$$

Even though it is not completely clear whether *Escardó and Xu*'s diagonalization argument [52], as presented in Section 2.4.2, is still valid in the case of BTT, internal continuity is harder to retrieve than external one, a fact we discuss in Section 3.4. In the same Section we also describe related work and the different hurdles we need to overcome to derive continuity of all MLTT-definable functionals.

[52]: Escardó et al. (2015), "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation"

Our model has been formalized in Coq using a presentation close to category-with-families. The code can be found at `https://gitlab.inria.fr/mbaillon/gardening-with-the-pythia`.

3.1. Escardó's model

Escardó proves continuity of all functionals

$$\vdash^{\mathsf{T}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

that are definable in System T, a theory we presented in Section 1.1. To do this, he builds two models of System T and binds the two interpretations together with a binary logical relation. Since System T is a simpler theory than BTT and as the crux of the argument is the same, it is worth looking at his proof in details before chewing up our (rather indigest) model.

The main objective of the proof is continuity of $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ functionals, but almost every construction used in the model can be generalized to arbitrary types

$$\vdash^{\mathscr{T}} \mathbf{I} : \square_0 \qquad \text{and} \qquad \vdash^{\mathscr{T}} \mathbf{O} : \mathbf{I} \to \square_0,$$

as in Chapter 2. Using the same notation, we will thus write

$$\mathbf{Q} := \Pi(i : \mathbf{I}).\, \mathbf{O}\, i$$

to denote the type of oracles. That being said, *Escardó*'s model stands on three legs:

1. An axiom model, where System T is extended with a black-box oracle $\alpha : \mathbf{Q}$;
2. A dialogue model, where types are interpreted as algebras of the dialogue monad $\mathfrak{D}$, and every term consequently becomes a tree;
3. A logical relation binding the two previous models together. On the base type, this relation ensures that for any System T definable term
$$\vdash^{\mathsf{T}} t : \mathbb{N},$$
we get
$$[t]_a = \partial\, [t]_d\, \alpha$$
where $[t]_a$ is the axiom translation of t and $[t]_d$ its dialogue translation. This equation lets us derive dialogue continuity for every term.

The dialogue monad was presented in Section 2.1.

This Section differs from *Escardó* [51] on one key component. Indeed, our approach is to build a *syntactic model* of System T through a *program translation* as presented in Section 1.5, while *Escardó*'s model does not really qualify as such. Rather, it is a model in a type-theoretic metatheory. The difference is subtle, and lies in the fact that in *Escardó*'s model the source language is deeply embedded as an AST in the target theory, while there will be no such thing in sight in our variant. The reason is, to generalize *Escardó*'s approach to BTT, we would need to internalize type theory inside itself, a notoriously difficult feat which we try to avoid. Morally, the program translation paradigm is a way to evade the kind of headache that led *Altenkirch and Kaposi* to use quotient inductive types to describe MLTT from within Agda [7].

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

Escardó uses the K and S combinators, but we find this presentation more readable, especially for a program translation.

In Chapter 4, however, we will internalize type theory inside itself, as we found no other way to enhance our proof from BTT to MLTT.

[7]: Altenkirch et al. (2016), "Type theory in type theory using quotient inductive types"

3.1.1. A for Axiom

Let us fix a reserved variable $\alpha : \mathrm{Q}$. The first translation from System T to CIC simply consists in adding α as the first variable of the context. Everywhere else, this translation is the standard embedding of System T into MLTT. Reserving a variable has no technical consequence, if we were to use de Bruijn indices it would just amount to shifting them all by one. We will also annotate both free and bound variables with an a subscript for readability of the future parts of the Chapter, where we mix together different translations. We call this translation the *axiom translation* and formally describe it in Figure 3.1.

$$
\begin{array}{lcl}
[\![\mathbb{N}]\!]_a & := & \mathbb{N} \\
[\![A \to B]\!]_a & := & [\![A]\!]_a \to [\![B]\!]_a \\
[x]_a & := & x_a \\
[\lambda x : A.\, t]_a & := & \lambda x_a : [\![A]\!]_a.\, [t]_a \\
[t\ u]_a & := & [t]_a\ [u]_a \\
[\mathrm{O}]_a & := & \mathrm{O} \\
[\mathrm{S}]_a & := & \mathrm{S} \\
[\mathbb{N}_{\mathbf{rec}}]_a\ P & := & \mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\ P)
\end{array}
$$

$$
\begin{array}{lcl}
[\![\cdot]\!]_a & := & \alpha : \mathrm{Q} \\
[\![\Gamma,\ x : A]\!]_a & := & [\![\Gamma]\!]_a, x_a : [\![A]\!]_a
\end{array}
$$

Figure 3.1.: Axiom translation of System T

Since our translation is mostly harmless, the following result should be no surprise:

Lemma 3.1.1:
The axiom translation is a model of System T.

3.1.2. Dialogue is maybe not the key

$$
\begin{array}{lcl}
[\![\mathbb{N}]\!]_d & := & \mathfrak{D}\,\mathbb{N} \\
[\![A \to B]\!]_d & := & [\![A]\!]_d \to [\![B]\!]_d \\
[x]_d & := & x_d \\
[\lambda x : A.\, t]_d & := & \lambda x_d : [\![A]\!]_d.\, [t]_d \\
[t\ u]_d & := & [t]_d\ [u]_d \\
[\mathrm{O}]_d & := & \eta\ \mathrm{O} \\
[\mathrm{S}]_d & := & \mathtt{bind}\ (\lambda(x : \mathbb{N}).\, \eta\ (\mathrm{S}\ x)) \\
[\mathbb{N}_{\mathbf{rec}}]_d\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n & := & \mathtt{gbind}\ \mathbb{N}\ P\ (\mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\, [\![P]\!]_d)\ p_{\mathrm{O}}\ p_{\mathrm{S}})\ n
\end{array}
$$

$$
\begin{array}{lcl}
[\![\cdot]\!]_d & := & \cdot \\
[\![\Gamma,\ x : A]\!]_d & := & [\![\Gamma]\!]_d, x_a : [\![A]\!]_d
\end{array}
$$

As a reminder, the definition of $\mathfrak{D}$ is the following:

$$
\begin{array}{l}
\mathtt{Inductive}\ \mathfrak{D}\ (A : \square_i) : \square_i := \\
\mid \eta : A \to \mathfrak{D}\ A \\
\mid \beta : \Pi(i : \mathrm{I}).\, (\mathrm{O}\ i \to \mathfrak{D}\ A) \to \mathfrak{D}\ A.
\end{array}
$$

Figure 3.2.: Dialogue translation of System T

Taking advantage of the fact that $\mathfrak{D}$ is a monad, something we discussed in Section 2.1, *Escardó* defines the second translation, displayed in Figure 3.2. *Escardó* calls it the *dialogue interpretation*, because of its use of the $\mathfrak{D}$ operator. The idea is to interpret types of System T by algebras of $\mathfrak{D}$. *Escardó* starts by picking the free algebra of $\mathfrak{D}$ for his translation of $\mathbb{N}$.

$$[\![\mathbb{N}]\!]_d := \mathfrak{D}\,\mathbb{N}$$

This non-standard interpretation for $\mathbb{N}$ is followed by pointwise translation for functional types:

$$[\![A \to B]\!]_d := [\![A]\!]_d \to [\![B]\!]_d$$

The functional type $A \to B$ is an algebra of $\mathfrak{D}$ as long as B is one. This property allows *Escardó* to define a generalized `bind` operator:

The `bind` operator is called *Kleisli extension* in *Escardó's* paper, but both names refer to the same concept.

$$
\begin{array}{lcl}
\mathtt{gbind} & : & \Pi\, A\, B\, (f : A \to [\![B]\!]_d)\, (d : \mathfrak{D}\ A).\, \mathfrak{D}\ B \\
\mathtt{gbind}\ A\ \mathbb{N}\ f\ d & := & \mathtt{bind}\ f\ d \\
\mathtt{gbind}\ A\ (X \to Y)\ f\ d & := & \lambda x.\, \mathtt{gbind}\ A\ Y\ (\lambda a.\, f\ a\ x)\ d
\end{array}
$$

Regarding terms, translations for zero and successor are quite straightforward:

$$
\begin{array}{lcl}
[\mathrm{O}]_d & := & \eta\ \mathrm{O} \\
[\mathrm{S}]_d & := & \mathtt{bind}\ (\lambda(x : \mathbb{N}).\, \eta\ (\mathrm{S}\ x))
\end{array}
$$

Finally, to define the interpretation of $\mathbb{N}_{\mathbf{rec}}$, *Escardó* simply makes use of the `gbind` operator defined above:

$$[\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_d := \mathtt{gbind}\ \mathbb{N}\ P\ (\mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\, [\![P]\!]_d)\ [p_{\mathrm{O}}]_d\ [p_{\mathrm{S}}]_d)\ [n]_d.$$

At this point, we can already highlight a key problem that needs solving if we want to turn *Escardó*'s model into a model of dependent type theory. The reason for that has been already briefly observed by *Sterling* [130] but it is worth elaborating here. Said bluntly, we have the following:

[130]: Sterling (2021), "Higher order functions and Brouwer's thesis"

Lemma 3.1.2:
The dialogue interpretation is *not* a model of System T

Proof. Indeed, while

$$[\mathbb{N}_{\mathbf{rec}}]_d$$

has the right type, it does not enjoy the correct computational behaviour. Namely, in general

$$[\mathbb{N}_{\mathbf{rec}}]_d\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ ([\mathrm{S}]_d\ n) \quad \not\equiv \quad p_{\mathrm{S}}\ n\ ([\mathbb{N}_{\mathbf{rec}}]_d\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n).$$

A simple counter-example is the following: given $i : \mathbf{I}$, let us pick

$$\begin{array}{lcl} P & := & \mathbb{N} \\ p_{\mathrm{O}} & := & \eta\ \mathrm{O} \\ p_{\mathrm{S}} & := & \lambda\_\ \_.\ \eta\ \mathrm{O} \\ n & := & \beta\ i\ (\lambda\_.\ \eta\ \mathrm{O}) \end{array}$$

Then on the one hand we have

$$[\mathrm{S}]_d\ n \equiv \mathtt{bind}\ (\lambda(x : \mathbb{N}).\ \eta\ (\mathrm{S}\ x))\ (\beta\ \mathrm{O}\ (\lambda\_.\ \eta\ \mathrm{O}) \equiv \beta\ i\ (\lambda\_.\ \eta\ (\mathrm{S}\ \mathrm{O}))$$

which leads to

$$\begin{array}{rcl} [\mathbb{N}_{\mathbf{rec}}\ P]_d\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ ([\mathrm{S}]_d\ n) & \equiv & \mathtt{gbind}\ \mathbb{N}\ \mathbb{N}\ (\mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\ \mathfrak{D}\ \mathbb{N})\ p_{\mathrm{O}}\ p_{\mathrm{S}})\ (\beta\ i\ (\lambda\_.\ \eta\ (\mathrm{S}\ \mathrm{O}))) \\ & \equiv & \mathtt{bind}\ (\mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\ \mathfrak{D}\ \mathbb{N})\ p_{\mathrm{O}}\ p_{\mathrm{S}})\ (\beta\ i\ (\lambda\_.\ \eta\ (\mathrm{S}\ \mathrm{O}))) \\ & \equiv & \beta\ i\ (\lambda\_.\ \eta\ (\mathbb{N}_{\mathbf{ind}}\ (\lambda\_.\ \mathfrak{D}\ \mathbb{N})\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\mathrm{S}\ \mathrm{O}))) \\ & \equiv & \beta\ i\ (\lambda\_.\ \eta\ (p_{\mathrm{S}}\ \mathrm{O}\ p_{\mathrm{O}})) \\ & \equiv & \beta\ i\ (\lambda\_.\ \eta\ \mathrm{O}) \end{array}$$

while on the other hand

$$p_{\mathrm{S}}\ ([\mathbb{N}_{\mathbf{rec}}]_d\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n) \equiv \mathrm{O}$$

by definition of p_{S}. ∎

The wrong call. This can be explained by the fact that recursive constructors in effectful call-by-name need to thunk their arguments, i.e., pattern-matching on the head of an inductive term must not evaluate the subterms of the constructor. This is not the case for *Escardó*'s intepretation, which is closer to a call-by-value embedding of $\mathbb{N}$ in call-by-name.

This drawback does not prevent *Escardó*'s proof to go through for System T. Indeed, $[\mathbb{N}_{\mathbf{rec}}]_d$ correctly computes on pure terms, which is enough to derive that any System T definable functional is continuous. However, in our case we are trying to extend this proof to *dependent* type theory. Hence, if this equation does not hold, typing rules themselves collapse. We thus need to pick the right interpretation of $\mathbb{N}$.

Since we want to build a model of dependent type theory, we need to preserve a call-by-name equational theory, i.e., generated by the unrestricted β-rule. Following *Pédrot and Tabareau* [121], this means that we need to interpret types as some kind of $\mathfrak{D}$-algebras. Unfortunately, for this interpretation to work we would need to preserve naturality laws, which fundamentally rely on funext, a principle not available in CIC. Thankfully, as we pointed out in Section 2.1.2, $\mathfrak{D}$ is not only a monad, it is a *free monad*, allowing intensional algebras, *i.e.* types with *pythias*. As we explained in the same Section, intensional algebras are extensionnally equivalent to usual $\mathfrak{D}$-algebras, but allow us to build dependent products *intensionally*. As we try and provide a System T model with all the necessary tools to scale to dependent type theory, we should now be looking for an interpretation of $\mathbb{N}$ featuring pythias.

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

3.1.3. System Trees

$$
\begin{array}{lcll}
[\![\mathbb{N}]\!]_b & := & \mathbb{N}_b & \\
[\![A \to B]\!]_b & := & [\![A]\!]_b \to [\![B]\!]_b & \\
[x]_b & := & x_b & \\
[\lambda x : A.\, t]_b & := & \lambda x_b : [\![A]\!]_b.\, [t]_b & \\
[t\ u]_b & := & [t]_b\ [u]_b & \\
[\mathrm{O}]_b & := & \mathrm{O}_b & \\
[\mathrm{S}]_b & := & \mathrm{S}_b & \\
[\mathbb{N}_{\mathbf{rec}}]_b & := & [\mathbb{N}_{\mathbf{rec}}]_b & \text{as defined below} \\
 & & & \\
[\![\cdot]\!]_b & := & \cdot & \\
[\![\Gamma, x : A]\!]_b & := & [\![\Gamma]\!]_b, x_b : [\![A]\!]_b &
\end{array}
$$

Figure 3.3.: Branching translation of System T

As it turns out, the solution can already be found in *Pédrot and Tabareau* [121]. Given an inductive type $\mathcal{I}$, we need to create an inductive type $\mathcal{I}_b$ whose constructors are the pointwise translation of the constructors of $\mathcal{I}$, together with an additional $\beta_{\mathcal{I}}$ pythia, turning $\mathcal{I}_b$ into an intensional $\mathfrak{D}$-algebra as presented in Section 2.1.2. For $\mathbb{N}$, we get:

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}_b : \square_i := \\
\mid \mathrm{O}_b : \mathbb{N}_b \\
\mid \mathrm{S}_b : \mathbb{N}_b \to \mathbb{N}_b \\
\mid \beta_{\mathbb{N}} : \Pi(i : \mathrm{I}).\, (\mathrm{O}\ i \to \mathbb{N}_b) \to \mathbb{N}_b
\end{array}
$$

We can now define what we call the *branching translation*, displayed in Figure 3.3. The translation of $\mathbb{N}$ is

$$[\![\mathbb{N}]\!]_b := \mathbb{N}_b$$

while O and S are immediately sent to their $\mathbb{N}_b$ counterpart. Translation for functional types is once again pointwise:

$$[\![A \to B]\!]_b := [\![A]\!]_b \to [\![B]\!]_b.$$

Our main concern regarding the translation of $\mathbb{N}_{\mathbf{rec}}$ is already addressed. The following definition is enough to retrieve a model of System T:

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_b & : & \Pi\, P\, (p_{\mathrm{O}} : [\![P]\!]_b)\, (p_{\mathrm{S}} : [\![P]\!]_b \to [\![P]\!]_b)\, (n : \mathbb{N}_b).\, [\![P]\!]_b \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ \mathrm{O}_b & \coloneqq & p_{\mathrm{O}} \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\mathrm{S}_b\ k) & \coloneqq & p_{\mathrm{S}}\ ([\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ k) \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\beta_{\mathbb{N}}\ i\ k) & \coloneqq & p_{\mathrm{O}}
\end{array}
$$

This time we satisfy our computational needs:

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ \mathrm{O}_b & \equiv & p_{\mathrm{O}} \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ ([\mathrm{S}]_b\ n) & \equiv & p_{\mathrm{S}}\ ([\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n)
\end{array}
$$

As there is no requirement regarding the computation on $\beta_{\mathbb{N}}$, we can simply return p_{O} as a dummy value. However, this choice would prevent our proof to go through later on, when we build a *generic element* to derive continuity of all functionals. Instead of a dummy value, we thus provide a branching function at any type:

$$
\begin{array}{lcl}
\mathtt{branch} & : & \Pi\, A\, (i : \mathrm{I})\, (k : \mathbf{O}\ i \to [\![A]\!]_b).\, [\![A]\!]_b \\
\mathtt{branch}\ \mathbb{N}\ i\ k & \coloneqq & \beta_{\mathbb{N}} \\
\mathtt{branch}\ (A \to B)\ i\ k & \coloneqq & \lambda x.\, \mathtt{branch}\ B\ i\ (\lambda o.\, k\ o\ x)
\end{array}
$$

We can then define:

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_b & : & \Pi\, P\, (p_{\mathrm{O}} : [\![P]\!]_b)\, (p_{\mathrm{S}} : [\![P]\!]_b \to [\![P]\!]_b)\, (n : \mathbb{N}_b).\, [\![P]\!]_b \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ \mathrm{O}_b & \coloneqq & p_{\mathrm{O}} \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\mathrm{S}_b\ k) & \coloneqq & p_{\mathrm{S}}\ ([\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ k) \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\beta_{\mathbb{N}}\ i\ k) & \coloneqq & \mathtt{branch}\ P\ i\ (\lambda o.\, [\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (k\ o))
\end{array}
$$

In some sense, this definition is linked to our discussion on free monads in Section 2.1.2. Indeed, branch is actually a function providing a *pythia* at every type, making sure that both

$$\mathbb{N} \qquad \text{and} \qquad A \to B$$

are actually *intensional algebras for the* $\mathfrak{D}$ *free monad.* Note that branch recurses on types of System T, a feature available to us because System T is simply-typed. In dependent type theory, having a recursor on $\square$ is however non trivial; we will circumvent the issue by interpreting $\square$ as $\square^b$, the type of intensional algebras for the $\mathfrak{D}$ monad, defined as follows:

$$\square^b := \Sigma(A : \square).\, \Pi(i : \mathrm{I}).\, (\mathbf{O}\ i \to A) \to A.$$

Hence every type $A : \square^b$ in the model will carry a proof that it is an intensional algebra, and branch will be replaced by the second projection $A.\pi_2$.

Going back to our System T model, we still retain the following:

Lemma 3.1.3:
The branching translation is a model of System T.

3.1.4. The logical song

What is left is to bind together the axiom and branching translations. To that effect, *Escardó* defines a binary logical relation by recursion on the types of System T. Through the lenses of program translations this comes down to interpreting a type A as a binary relation

$$[\![A]\!]_\ell \ : \ [\![A]\!]_a \to [\![A]\!]_b \to \square.$$

This leads to a syntactic model akin to parametricity [23]. We call it the *logical translation*. Notice that we stick with our branching translation $[\![A]\!]_b$ here. This means that what follows parts ways with *Escardó*'s proof, which relies on the dialogue one. However, both translations being quite similar, for System T the switch from one to the other is relatively painless.

[23]: Bernardy et al. (2011), "Realizability and Parametricity in Pure Type Systems"

For instance, for $\mathbb{N}_b$ we can define a dialogue function $\partial^{\mathbb{N}}$ similar to ∂ defined in Section 2.1:

$$\begin{array}{lcl} \partial^{\mathbb{N}} & : & \mathbb{N}_b \to \mathrm{Q} \to \mathbb{N} \\ \partial^{\mathbb{N}}\ \mathrm{O}_b\ \alpha & := & \mathrm{O} \\ \partial^{\mathbb{N}}\ (\mathrm{S}_b\ n_b)\ \alpha & := & \mathrm{S}\ (\partial^{\mathbb{N}}\ n_b\ \alpha) \\ \partial^{\mathbb{N}}\ (\beta_{\mathbb{N}}\ i\ k)\ \alpha & := & \partial^{\mathbb{N}}\ \ (k\ (\alpha\ i))\ \alpha. \end{array}$$

The binary relation for natural numbers then becomes the following:

$$[\![\mathbb{N}]\!]_\ell \coloneqq \Pi(n_a \ : \ [\![\mathbb{N}]\!]_a)\,(n_b \ : \ [\![\mathbb{N}]\!]_b).\, n_a = \partial^{\mathbb{N}}\ n_b\ \alpha$$

where

$$\alpha \ : \ \mathrm{Q}$$

is a parameter of the translation, as for the axiom translation.

As usual with logical relation, functional types are interpreted as preservation of the relation:

$$[\![A \to B]\!]_\ell\ f_a\ f_b \coloneqq \Pi(x_a \ : \ [\![A]\!]_a)\,(x_b \ : \ [\![A]\!]_b)\,(x_e \ : \ [\![A]\!]_\ell\ x_a\ x_b).\, [\![B]\!]_\ell\ (f_a\ x_a)\ (f_b\ x_b)$$

As for terms, the translation of zero is the constructor refl of equality.

Using transport, it is possible to derive a lemma

$$\mathrm{S}_\ell \ : \ \Pi(n_a \ : \ [\![\mathbb{N}]\!]_a)\,(n_b \ : \ [\![\mathbb{N}]\!]_b).\, n_a = \partial^{\mathbb{N}}\ n_b\ \alpha \to \mathrm{S}\ n_a = \partial^{\mathbb{N}}\ (\mathrm{S}_b\ n_b)\ \alpha$$

translating S.

Finally, for $\mathbb{N}_{\mathbf{rec}}$ we first need an auxiliary function to take care of the branching case:

$$\begin{array}{lcl} \mathtt{branch}_\ell & : & \Pi\, A\,(a \ : \ [\![A]\!]_a)\,(i \ : \ \mathrm{I})\,(k \ : \ \mathrm{O}\ i \to [\![A]\!]_b). \\ & & \quad [\![A]\!]_\ell\ a\ (k\ (\alpha\ i)) \to [\![A]\!]_\ell\ a\ (\beta\ i\ k) \\ \mathtt{branch}_\ell\ \mathbb{N}\ a\ i\ k\ k_\ell & := & k_\ell \\ \mathtt{branch}_\ell\ (A \to B)\ f_a\ i\ k\ k_\ell & := & \lambda x_a\ x_b\ x_\ell.\, \mathtt{branch}_\ell\ B\ (f_a\ x_a) \\ & & \qquad\qquad\qquad\qquad i\ (\lambda o.\, k\ o\ x) \\ & & \qquad\qquad\qquad\qquad (k_\ell\ x_a\ x_b\ x_\ell) \end{array}$$

$$
\begin{array}{lcl}
[\![\mathbb{N}]\!]_\ell\ n_a\ n_b & := & n_a = \partial^{\mathbb{N}}\ n_b\ \alpha \\
[\![A \to B]\!]_\ell\ f_a\ f_b & := & \Pi(x_a : [\![A]\!]_a)(x_b : [\![A]\!]_b)(x_\ell : [\![A]\!]_\ell\ x_a\ x_b).\ [\![B]\!]_\ell\ (f_a\ x_a)\ (f_b\ x_b) \\
[x]_\ell & := & x_\ell \\
[\lambda x : A.\, t]_\ell & := & \lambda(x_a : [\![A]\!]_a)(x_b : [\![A]\!]_b)(x_\ell : [\![A]\!]_\ell\ x_a\ x_b).\ [t]_\ell \\
[t\ u]_\ell & := & [t]_\ell\ [u]_a\ [u]_b\ [u]_\ell \\
[\mathrm{O}]_\ell & := & \texttt{refl} \\
[\mathrm{S}]_\ell & := & \mathrm{S}_\ell \\
[\mathbb{N}_{\mathbf{rec}}]_\ell & := & [\mathbb{N}_{\mathbf{rec}}]_\ell \qquad \text{as defined below}
\end{array}
$$

$$
\begin{array}{lcl}
[\![\cdot]\!]_\ell & := & \alpha : \mathrm{Q} \\
[\![\Gamma, x : A]\!]_\ell & := & [\![\Gamma]\!]_\ell, x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\ell : [\![A]\!]_\ell\ x_a\ x_b
\end{array}
$$

Figure 3.4.: Logical translation for System T

The syntactic burden is at its heaviest at this point since we now have everything repeated three times. To enhance readability, we will use the following shorthand for binders:

$$\langle x : A\rangle := x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\ell : [\![A]\!]_\ell\ x_a\ x_b$$

and similarly for application to variables.

We then have to define

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_\ell & : & \Pi\, P\langle p_{\mathrm{O}} : P\rangle\langle p_{\mathrm{S}} : \mathbb{N} \to P \to P\rangle\langle n : \mathbb{N}\rangle. \\
 & & \quad [\![P]\!]_\ell\ ([\mathbb{N}_{\mathbf{rec}}]_a\ p_{\mathrm{O}a}\ p_{\mathrm{S}a}\ n_a)\ ([\mathbb{N}_{\mathbf{rec}}]_b\ p_{\mathrm{O}b}\ p_{\mathrm{S}b}\ n_b)
\end{array}
$$

However, as

$$n_\ell : [\![\mathbb{N}]\!]_\ell\ n_a\ n_b$$

is in fact a proof of

$$n_a = \partial^{\mathbb{N}}\ n_b\ \alpha,$$

we can transport along this equality. The following auxiliary function is then enough, and completes our translation:

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]'_\ell & : & \Pi\, P\langle p_{\mathrm{O}} : P\rangle\langle p_{\mathrm{S}} : \mathbb{N} \to P \to P\rangle(n_b : [\![\mathbb{N}]\!]_b). \\
 & & \quad [\![P]\!]_\ell\ ([\mathbb{N}_{\mathbf{rec}}]_a\ p_{\mathrm{O}a}\ p_{\mathrm{S}a}\ (\partial^{\mathbb{N}}\ n_b\ \alpha))\ ([\mathbb{N}_{\mathbf{rec}}]_b\ p_{\mathrm{O}b}\ p_{\mathrm{S}b}\ n_b) \\
[\mathbb{N}_{\mathbf{rec}}]'_\ell\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \mathrm{O}_b & := & p_{\mathrm{O}\ell} \\
[\mathbb{N}_{\mathbf{rec}}]'_\ell\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ (\mathrm{S}_b\ n_b) & := & p_{\mathrm{S}\ell}\ n_\ell\ ([\mathbb{N}_{\mathbf{rec}}]'_\ell\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ n_b) \\
[\mathbb{N}_{\mathbf{rec}}]'_\ell\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ (\beta\ i\ k) & := & \mathtt{branch}_\ell\ P\ ([\mathbb{N}_{\mathbf{rec}}]_a\ p_{\mathrm{O}a}\ p_{\mathrm{S}a}\ (\partial^{\mathbb{N}}\ (k\ (\alpha\ i))\ \alpha)) \\
 & & \qquad\qquad i\ (\lambda o.\ ([\mathbb{N}_{\mathbf{rec}}]_b\ p_{\mathrm{O}b}\ p_{\mathrm{S}b}\ (k\ o)) \\
 & & \qquad\qquad ([\mathbb{N}_{\mathbf{rec}}]'_\ell\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ (k\ (\alpha\ i)))
\end{array}
$$

However, we now face a similar problem as with the dialogue translation:

Lemma 3.1.4:
The logical translation is *not* a model of System T.

Proof. Once again, trouble lies within computation rules for $\mathbb{N}_{\mathbf{rec}}$. Indeed, the way we defined it,

$$\mathsf{S}_\ell : \Pi(n_a : [\![\mathbb{N}]\!]_a)\,(n_b : [\![\mathbb{N}]\!]_b).\, n_a = \partial^{\mathbb{N}}\, n_b\, \alpha \to \mathsf{S}\, n_a = \partial^{\mathbb{N}}\, (\mathsf{S}_b\, n_b)\, \alpha$$

is not a constructor: it is a function computing on the proof of equality

$$n_\ell : n_a = \partial^{\mathbb{N}}\, n_b\, \alpha.$$

Consequently, when

$$\langle n : \mathbb{N} \rangle$$

is a variable, computation is blocked,and we have:

$$[\mathbb{N}_{\mathbf{rec}}]_\ell\; p_{\mathsf{O}}\; p_{\mathsf{S}}\; ([\mathsf{S}]_\ell\; n) \not\equiv p_{\mathsf{S}}\; ([\mathbb{N}_{\mathbf{rec}}]_\ell\; p_{\mathsf{O}}\; p_{\mathsf{S}}\; n).$$

There are at least two ways out of the pithole:

1. We can switch to a target theory validating *equality reflection* so that any proof
   $$n_\ell : n_a = \partial^{\mathbb{N}}\, n_b\, \alpha$$
   becomes definitionally equal to refl, thereby unblocking computation. This can be achieved for instance by composing our model with *Winterhalter et al*'s translation of ETT to ITT [144]. As we pointed out in Section 1.3, by doing so we would lose decidability of type-checking, which would seriously hinder our formalization effort. Still, it is a price we might find acceptable if this were our only option, since it would not completely block our proof of continuity.
2. The same way we dealt with computation rules for the branching translation, we can turn our translation of $\mathbb{N}$ into an inductive one, while ensuring that any proof of our new logical relation still entails
   $$n_a = \partial^{\mathbb{N}}\, n_b\, \alpha.$$
   As with the branching translation, it is highly likely that our new translation of $\mathbb{N}$ will have to feature some kind of β pythia constructor, turning it into an intensional algebra for the $\mathfrak{D}$ monad.

We choose the latter option and define our fifth and last translation of System T.

[144]: Winterhalter et al. (2019), "Eliminating reflection from type theory"

3.1.5. For a handful of models

Once again, in this last translation types will be interpreted as binary predicates over the axiom and branching translation. This time, however, the translation sticks more closely to the branching one, in that it makes sure that every type is *algebraic* with respect to $\mathfrak{D}$. We call it the *algebraic parametricity translation*. Formally, this means that the translation of $\mathbb{N}$ is the following inductive:

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}_\varepsilon\ (\alpha : \mathrm{Q}) : \mathbb{N} \to \mathbb{N}_b \to \square := \\
\quad | \ \mathrm{O}_\varepsilon \quad : \quad \mathbb{N}_\varepsilon\ \alpha\ \mathrm{O}\ \mathrm{O}_b \\
\quad | \ \mathrm{S}_\varepsilon \quad : \quad \Pi(n_a : \mathbb{N})\,(n_b : \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ n_b).\,\mathbb{N}_\varepsilon\ \alpha\ (\mathrm{S}\ n_a)\ (\mathrm{S}_b\ n_b) \\
\quad | \ \beta^\varepsilon_{\mathbb{N}} \quad : \quad \Pi(n_a : \mathbb{N})\,(i : \mathrm{I})\,(k : \mathrm{O}\ i \to \mathbb{N}_b). \\
\qquad\qquad \mathbb{N}_\varepsilon\ \alpha\ n_a\ (k\ (\alpha\ i)) \to \mathbb{N}_\varepsilon\ \alpha\ n_a\ (\beta_{\mathbb{N}}\ i\ k)
\end{array}
$$

Note that $\alpha : \mathrm{Q}$ is implicitly part of the context as in the axiom model. Similarly to previous translations, function types are translated pointwise:

$$[\![A \to B]\!]_\varepsilon\ f_a\ f_b \coloneqq \Pi(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[\![B]\!]_\varepsilon\ (f_a\ x_a)\ (f_b\ x_b)$$

and we need to provide a proof of algebraicity at every type:

$$
\begin{array}{lcl}
\texttt{branch}_\varepsilon & : & \Pi\,A\,(a : [\![A]\!]_a)\,(i : \mathrm{I})\,(k : \mathrm{O}\ i \to [\![A]\!]_b). \\
& & \quad [\![A]\!]_\varepsilon\ a\ (k\ (\alpha\ i)) \to [\![A]\!]_\varepsilon\ a\ (\beta\ i\ k) \\
\texttt{branch}_\varepsilon\ \mathbb{N}\ n_a\ i\ k\ n_\varepsilon & \coloneqq & \beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon \\
\texttt{branch}_\varepsilon\ (A \to B)\ f_a\ i\ k\ k_\varepsilon & \coloneqq & \lambda x_a\ x_b\ x_\varepsilon.\,\texttt{branch}_\varepsilon\ B\ (f_a\ x_a) \\
& & \qquad\qquad i\ (\lambda o.\ k\ o\ x) \\
& & \qquad\qquad (k_\varepsilon\ x_a\ x_b\ x_\varepsilon)
\end{array}
$$

This time, zero and successor are translated as constructors of $\mathbb{N}_\varepsilon$.

$$[\mathrm{O}]_\varepsilon \coloneqq \mathrm{O}_\varepsilon\ \alpha \qquad [\mathrm{S}]_\varepsilon \coloneqq \mathrm{S}_\varepsilon\ \alpha$$

Since the syntactic pain has not been alleviated when switching translations, we keep our shorthand but change its meaning to

$$\langle x : A \rangle \coloneqq x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b.$$

Finally, the definition of $[\mathbb{N}_{\mathbf{rec}}]_\varepsilon$ is the following:

$$
\begin{array}{lcl}
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon & : & \Pi\,P\,\langle p_{\mathrm{O}} : P\rangle\,\langle p_{\mathrm{S}} : \mathbb{N} \to P \to P\rangle\,\langle n : \mathbb{N}\rangle. \\
& & \quad [\![P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_b \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \_\ \_\ \mathrm{O}_\varepsilon & \coloneqq & p_{\mathrm{O}\varepsilon} \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \_\ \_\ (\mathrm{S}_\varepsilon\ \langle n\rangle) & \coloneqq & p_{\mathrm{S}\varepsilon}\ n_\varepsilon\ ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \langle n\rangle) \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \_\ \_\ (\beta^\varepsilon n_a\ i\ k\ n_\varepsilon) & \coloneqq & \texttt{branch}_\varepsilon\ P\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a \\
& & \qquad\qquad i\ (\lambda o.\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (k\ o)]_b) \\
& & \qquad\qquad ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ P\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ n_a\ (k\ (\alpha\ i))\ n_\varepsilon)
\end{array}
$$

Lemma 3.1.5:
The algebraic parametricity translation is a model of System T.

3.1.6. A generic proof

We are almost there. To prove the main theorem, what is left is to define a generic element

$$\gamma : \mathbb{N} \to \mathbb{N},$$

a clever instance of the model described above. Before getting to the nitty-gritty, we will fix henceforth the oracular type parameters for the remainder of this Section as

$$\mathbf{I} := \mathbb{N} \qquad \text{and} \qquad \mathbf{O} := \lambda(\_ : \mathbf{I}).\, \mathbb{N}.$$

Some results exposed in this Section are still independent from this precise choice of oracle. When this is the case, we will stick to the $\mathbf{Q}$ notation to highlight this fact.

Let us state one last time the main result we are trying to prove:

Theorem 3.1.6:
Any System T-definable functional $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is dialogue continuous.

The main argument is the following: let us assume we have a generic element

$$\gamma : \mathbb{N} \to \mathbb{N}$$

such that its axiom translation is the $\alpha : \mathbb{N} \to \mathbb{N}$ oracle added in the context. That is, we have $\gamma_b, \gamma_\varepsilon$ in the target theory such that:

$$\begin{array}{rcl} \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & \alpha : [\![\mathbb{N} \to \mathbb{N}]\!]_a \\ & \vdash^{\mathcal{T}} & \gamma_b : [\![\mathbb{N} \to \mathbb{N}]\!]_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & \gamma_\varepsilon : [\![\mathbb{N} \to \mathbb{N}]\!]_\varepsilon\ \alpha\ \gamma_b \end{array}$$

Then given $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ in the source theory, we can consider the term

$$\vdash^{\mathsf{T}} f\ \gamma : \mathbb{N}.$$

By soundness, it results in the three terms below.

$$\begin{array}{rcl} \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & [f]_a\ \alpha : \mathbb{N} \\ & \vdash^{\mathcal{T}} & [f]_b\ \gamma_b : \mathbb{N}_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & [f]_\varepsilon\ \alpha\ \gamma_b\ \gamma_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ ([f]_a\ \alpha)\ ([f]_b\ \gamma_b) \end{array}$$

Crucially, the term $[f]_b\ \gamma_b : \mathbb{N}_b$ does not depend on α, meaning it can be our witness of dialogue continuity. All that is left is notice the following:

Proposition 3.1.7: Unicity of specification
There is a proof

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbf{Q})\, \langle n : \mathbb{N} \rangle.\, n_a = \partial^{\mathbb{N}}\ n_b\ \alpha.$$

Proof. By induction on n_ε. ∎

Applying Proposition 3.1.7 to

$$[f]_a\ \alpha, \qquad [f]_b\ \gamma_b \qquad \text{and} \qquad [f]_\varepsilon\ \alpha\ \gamma_b\ \gamma_\varepsilon,$$

we get

$$\vdash^{\mathcal{T}} \Pi(\alpha : \mathbb{N} \to \mathbb{N}).\, [f]_a\ \alpha = \partial^{\mathbb{N}}\ ([f]_b\ \gamma_b)\ \alpha.$$

Since f is a term in System T that does not use any impure extension of the model, it is easy to check that

$$[f]_a \equiv f.$$

We thus retain

$$\vdash^{\mathcal{T}} \Pi(\alpha : \mathbb{N} \to \mathbb{N}).\, f\ \alpha = \partial^{\mathbb{N}}\ ([f]_b\ \gamma_b)\ \alpha,$$

meaning that f is dialogue continuous, from the target theory perspective, which concludes our proof.

Fixing a hole The argument we just presented critically relies on the existence of γ, which we are yet to demonstrate. Let us now tackle this last missing bit in our proof.

Proposition 3.1.8: Generic parametricity
There is a proof

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbb{Q})\,(n_b : \mathbb{N}_b).\, \mathbb{N}_\varepsilon\ \alpha\ (\partial^{\mathbb{N}}\ n_b\ \alpha)\ n_b.$$

Proof. By induction on n_b. ∎

Definition 3.1.1: Generic tree
We define in $\mathcal{T}$ the *generic tree* $\mathbf{t}$ as

$$\begin{array}{lcl} \mathbf{t} & : & \mathbb{N} \to \mathbb{N}_b \\ \mathbf{t} & := & \lambda(n : \mathbb{N}).\, \beta_{\mathbb{N}}\ n\ \eta_{\mathbb{N}} \end{array}$$

where

$$\begin{array}{lcl} \eta_{\mathbb{N}} & : & \mathbb{N} \to \mathbb{N}_b \\ \eta_{\mathbb{N}}\ \mathrm{O} & := & \mathrm{O}_b \\ \eta_{\mathbb{N}}\ (\mathrm{S}\ n) & := & \mathrm{S}_b\ (\eta_{\mathbb{N}}\ n) \end{array}$$

Graphically, $\mathbf{t}$ looks like the tree represented at Tab 3.1. It has the following property:

Lemma 3.1.9: Fundamental property of the generic tree
We have a proof

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n : \mathbb{N}).\, \partial^{\mathbb{N}}\ (\mathbf{t}\ n)\ \alpha = \alpha\ n.$$

Proof. Immediate by the definition of the $\partial^{\mathbb{N}}$ function. ∎

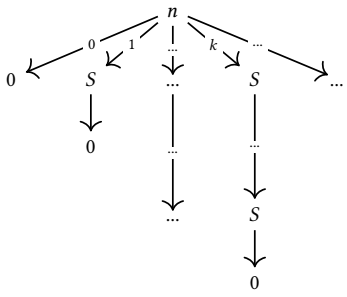


Table 3.1.: Generic Trees

Definition 3.1.2: Generic element
We define the generic element $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$ as follows.

$$\gamma_b\ n_b \quad := \quad \gamma_0\ \mathrm{O}\ n_b$$

where

$$\begin{array}{lcl} \gamma_0 & : & \mathbb{N} \to \mathbb{N}_b \to \mathbb{N}_b \\ \gamma_0\ a\ \mathrm{O}_b & := & \mathsf{t}\ a \\ \gamma_0\ a\ (\mathrm{S}_b\ n_b) & := & \gamma_0\ (\mathrm{S}\ a)\ n_b \\ \gamma_0\ a\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_{\mathbb{N}}\ i\ (\lambda o : \mathbb{N}.\, \gamma_0\ a\ (k\ o)). \end{array}$$

Intuitively, γ_b adds a layer to its argument, replacing each leaf by a $\mathsf{t}\ n$, where n is the number of S_b encountered in the branch. It has the following property:

Lemma 3.1.10: Fundamental property of the generic element
We have a proof

$$\vdash^{\mathscr{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n_b : \mathbb{N}_b).\, \partial^{\mathbb{N}}\ (\gamma_b\ n_b)\ \alpha = \alpha\ (\partial^{\mathbb{N}}\ n_b\ \alpha).$$

Proof. By induction on n_b, using Lemma 3.1.9 for the O_b case. ∎

Proposition 3.1.11:
The γ_b term can be lifted in the source theory to a function

$$\vdash^{\mathsf{T}} \gamma : \mathbb{N} \to \mathbb{N} \qquad \text{such that} \qquad [\gamma]_a := \alpha : [\![\mathbb{N} \to \mathbb{N}]\!]_a.$$

Proof. It is enough to derive the following sequents, the first two being trivial.

$$\begin{array}{lcl} \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathscr{T}} & \alpha : [\![\mathbb{N} \to \mathbb{N}]\!]_a \\ & \vdash^{\mathscr{T}} & \gamma_b : [\![\mathbb{N} \to \mathbb{N}]\!]_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathscr{T}} & \gamma_\varepsilon : [\![\mathbb{N} \to \mathbb{N}]\!]_\varepsilon\ \alpha\ \gamma_b \end{array}$$

For γ_ε, assuming $\langle n : \mathbb{N} \rangle$ we have to prove

$$[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ n_a)\ (\gamma_b\ n_b).$$

By Proposition 3.1.7, this is the same as

$$[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ (\partial^{\mathbb{N}}\ n_b\ \alpha))\ (\gamma_b\ n_b).$$

By Proposition 3.1.10, this is the same as

$$[\![\mathbb{N}]\!]_\varepsilon\ (\partial^{\mathbb{N}}\ (\gamma_b\ n_b)\ \alpha)\ (\gamma_b\ n_b).$$

We conclude by Proposition 3.1.8. ∎

This concludes our proof of continuity for all System T definable functionals. We can now set our sight on dependent type theory, and build a model of *Baclofen Type Theory*.

3.2. Our model gains weight

3.2.1. Overview

We prove that all BTT functions are continuous using a generalization of *Escardó*'s model. While the latter only provides a model of System T, a simply-typed language, our model accomodates not only dependent types, but also universes and inductive types equipped with a strict form of dependent elimination. It is given as a program translation, and thus belongs to the class of syntactic models [25, 75]. The final model is built in three stages, namely

1. An axiom model (Section 3.2.2),
2. A branching model (Section3.2.3),
3. An algebraic parametricity model (Section3.2.4).

[25]: Boulier (2018), "Extending type theory with syntactic models. (Etendre la théorie des types à l'aide de modèles syntaxiques)"
[75]: Hofmann (1997), *Extensional constructs in intensional type theory*

The first two models are standalone, and the third one glues them together. Each model can be explained computationally. The axiom model adds a blackbox oracle as a global variable. Asking the oracle is just function application, so there is no internal way to observe calls to the oracle. The branching model does the exact converse, as it provides an oracle in a purely inert way. Every single call to the branching oracle is tracked as a node of a dialogue tree, a representation that is reminiscent of game semantics. Finally, the algebraic parametricity model internalizes the fact that these two interpretations are computing essentially the same thing, behaving like a proof-relevant logical relation.

The results from this Section have been formalized in Coq using a presentation similar to category with families. It is a shallow embedding in the style of *Kaposi et al* [86], hence in particular all conversions are interpreted as definitional equalities. The development relies on universe polymorphism to implement universes in the model, but it could have been avoided at the cost of duplicating the code for every level existing in the hierarchy. As usual, we use negative pairs to handle context extensions in a definitional way. Apart from this, the development does not make use of any fancier feature from the Coq kernel. Throughout the Section, we provide hyperlinks in margins to the relevant parts of the development. The code as a whole can be found at

[86]: Kaposi et al. (2019), "Shallow Embedding of Type Theory is Morally Correct"

`https://gitlab.inria.fr/mbaillon/gardening-with-the-pythia`.

3.2.2. Axiom Translation

$$
\begin{array}{lcl}
[x]_a & := & x_a \\
[\lambda x : A.\, t]_a & := & \lambda x_a : [\![A]\!]_a.\, [t]_a \\
[t\ u]_a & := & [t]_a\ [u]_a \\
[\Pi x : A.\, B]_a & := & \Pi x_a : [\![A]\!]_a.\, [\![B]\!]_a \\
[\square_i]_a & := & \square_i \\
[\![A]\!]_a & := & [A]_a \\
 & & \\
[\![\cdot]\!]_a & := & \alpha : \mathbf{Q} \\
[\![\Gamma,\ x : A]\!]_a & := & [\![\Gamma]\!]_a, x_a : [\![A]\!]_a
\end{array}
$$

Figure 3.5.: Axiom translation (negative fragment)

There is not much to say, as it is exactly the same translation as in Section 3.1.1: we simply add $\alpha : \mathbf{Q}$ as the first variable of the context, nothing more. We formally give the translation of the negative fragment in Figure 3.5. Inductive types pose no difficulty.

Theorem 3.2.1:
The axiom translation is a syntactic model of CIC and hence of BTT.

Formalization of the axiom translation can be found here.

3.2.3. Branching Translation

$$
\begin{array}{lcl}
[x]_b & := & x_b \\
[\lambda x : A.\, t]_b & := & \lambda x_b : [\![A]\!]_b.\, [t]_b \\
[t\ u]_b & := & [t]_b\ [u]_b \\
[\![A]\!]_b & := & [A]_b.\pi_1 \\
[\![\square]\!]_b & := & \square^b \\
\beta_\square & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \square^b).\, \mho_b \\
[\![\Pi x : A.\, B]\!]_b & := & \Pi x_b : [\![A]\!]_b.\, [\![B]\!]_b \\
\beta_{\Pi x : A.\, B} & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \Pi x : [\![A]\!]_b.\, [\![B]\!]_b)\,(x : [\![A]\!]_b).\, \beta_B\ i\ (\lambda o : \mathbf{O}\ i.\, k\ o\ x) \\
 & & \\
[\![\cdot]\!]_b & := & \cdot \\
[\![\Gamma, x : A]\!]_b & := & [\![\Gamma]\!]_b, x_b : [\![A]\!]_b
\end{array}
$$

Figure 3.6.: Branching translation (negative fragment)

Using results from Section 2.1.2, we can use a simplified form of *weaning construction* [121] to define the *branching translation*. It all boils down to interpreting types as intensional $\mathfrak{D}$-algebras, whose type is

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

$$\vdash^{\mathscr{T}} \square^b := \Sigma(A : \square).\, \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to A) \to A.$$

Figure 3.6 defines the negative branching translation, translating

$$\vdash^{\mathscr{S}} A : \square \qquad \text{as} \qquad \vdash^{\mathscr{T}} [A]_b : \square^b,$$

i.e. a pair

$$([\![A]\!]_b, \beta_A) \qquad \text{where} \qquad [\![A]\!]_b : \square$$

and β_A is a pythia for $[\![A]\!]_b$.

For readability, we give the translation of types as these two components through a slight abuse of notation.

The main difficulty is to endow $\square^b$ with a $\mathfrak{D}$-algebra structure. However, to get a model of BTT the only constraint a pythia on $\square$ must satisfy is its own existence, and nothing more. This means that any function

$$\beta_\square : \Pi(i : \mathrm{I}).\,(\mathrm{O}\ i \to \square^b) \to \square^b$$

will do. Hence, we simply assume as a parameter of the translation a dummy $\mathfrak{D}$-algebra

$$\mathfrak{V}_b : \square^b \qquad \text{together with} \qquad \omega_b : \mathfrak{V}_b.\pi_1,$$

used to define dependent elimination. There are many possible choices for $\mathfrak{V}_b$, the simplest one being the unit type which is trivially inhabited and algebraic. As an instance of weaning, we get the following:

Proposition 3.2.2: CC_ω Soundness
We have the following:

- If $t \equiv_{\mathrm{CC}_\omega} u$ then $[t]_b \equiv_{\mathscr{T}} [u]_b$.
- If $\Gamma \vdash^{\mathrm{CC}_\omega} t : A$ then $[\![\Gamma]\!]_b \vdash^{\mathscr{T}} [t]_b : [\![A]\!]_b$.

Formalization of the branching translation can be found here.

The interpretation of inductive types is fairly straightforward. Given an inductive type $\mathscr{I}$, we create an inductive type $\mathscr{I}_b$ whose constructors are the pointwise translation of the constructors of $\mathscr{I}$, together with an additional $\beta_{\mathscr{I}}$ constructor turning it into a free $\mathfrak{D}$-algebra. We give as an example below the translation of $\mathbb{N}$, which will be the running example for the remainder of this Chapter. Parameters and indices present no additional difficulty and we refer to *Pédrot and Tabareau* [121] for more details.

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

$$\begin{array}{l}\texttt{Inductive}\ \mathbb{N}_b : \square_i := \\ |\ \mathrm{O}_b : \mathbb{N}_b \\ |\ \mathrm{S}_b : \mathbb{N}_b \to \mathbb{N}_b \\ |\ \beta_{\mathbb{N}} : \Pi(i : \mathrm{I}).\,(\mathrm{O}\ i \to \mathbb{N}_b) \to \mathbb{N}_b\end{array}$$

Theorem 3.2.3:
For any inductive type $\mathscr{I}$, its branching translation $\mathscr{I}_b$ is well-typed and satisfies the strict positivity criterion.

Formalization of the branching translation for some inductive types can be found here.

Corollary 3.2.4: Typing soundness
Typing soundness holds for the translation of inductive types and their constructors.

We must now implement eliminators. In the same way as for their System T counterpart, we first define the non-dependent ones.

$$\begin{array}{lcl}[\mathbb{N}_{\mathbf{rec}}]_b & : & \Pi P : \square^b.\ [\![P]\!]_b \to (\mathbb{N}_b \to [\![P]\!]_b \to [\![P]\!]_b) \to \\ & & \quad \mathbb{N}_b \to [\![P]\!]_b \\ [\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ \mathrm{O}_b & := & p_{\mathrm{O}} \\ [\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\mathrm{S}_b\ n) & := & p_{\mathrm{S}}\ n\ ([\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n) \\ [\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_P\ i\ (\lambda(o : \mathrm{O}\ i).\ [\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (k\ o))\end{array}$$

As $P : [\![\Box]\!]_b$, it has a pythia

$$\beta_P : \Pi(i : \mathrm{I}).\,(\mathrm{O}\ i \to [\![P]\!]_b) \to [\![P]\!]_b.$$

Every time we encounter a branching occurence of $\beta_{\mathbb{N}}$, we can thus use β_P and propagate the call recursively in the branches. This is the usual by-name semantics of recursors.

However, as pythias are a typical example of observable effects as we described in Section 1.4, problems arise with dependent elimination: assuming $\mathscr{T}$ is consistent, given

$$P : \mathbb{N}_b \to \Box^b$$

and subproofs for O_b and S_b, there is no clear way to produce a term of type

$$(P\ (\beta_{\mathbb{N}}\ i\ k)).\pi_1.$$

We therefore restrict ourselves to a strict dependent elimination, relying on the storage operator $\mathbb{N}_{\mathbf{str}}$ from Section 1.4. We recall its definition below. Since it is given in direct style, its translation is systematic.

$$\begin{array}{l}\mathbb{N}_{\mathbf{str}}\ (n : \mathbb{N})\ (P : \mathbb{N} \to \Box) : \Box := \\ \quad \mathbb{N}_{\mathbf{rec}}\ ((\mathbb{N} \to \Box) \to \Box)\ (\lambda(Q : \mathbb{N} \to \Box).\,Q\ \mathrm{O}) \\ \qquad (\lambda(m : \mathbb{N})\ (\_ : (\mathbb{N} \to \Box) \to \Box)\ (Q : \mathbb{N} \to \Box).\,Q\ (\mathrm{S}\ m))\ n\ P.\end{array}$$

Lemma 3.2.5:
We have the following conversions.

1. $[\mathbb{N}_{\mathbf{str}}]_b\ \mathrm{O}_b\ P \equiv P\ \mathrm{O}_b$
2. $[\mathbb{N}_{\mathbf{str}}]_b\ (\mathrm{S}_b\ n)\ P \equiv P\ (\mathrm{S}_b\ n)$
3. $[\mathbb{N}_{\mathbf{str}}]_b\ (\beta_{\mathbb{N}}\ i\ k)\ P \equiv \mathfrak{O}_b$

Note that the two first equations above are a consequence of the conversion rules of $\mathbb{N}_{\mathbf{rec}}$ and thus hold in any model of BTT. Only the last one is specific to the current model at hand. Using this, we define the dependent eliminator below. Thanks to the fact that the predicate is wrapped in a storage operator, it is able to return a dummy term when applied to an effectful argument.

$$\begin{array}{lcl}[\mathbb{N}_{\mathbf{sind}}]_b & : & \Pi P : \mathbb{N} \to \Box.\,P\ \mathrm{O} \to \\ & & \quad (\Pi n : \mathbb{N}.\,\mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathrm{S}\ n)\ P) \to \\ & & \quad \Pi n : \mathbb{N}.\,\mathbb{N}_{\mathbf{str}}\ n\ P \\ [\mathbb{N}_{\mathbf{sind}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ \mathrm{O}_b & := & p_{\mathrm{O}} \\ [\mathbb{N}_{\mathbf{sind}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\mathrm{S}_b\ n) & := & p_{\mathrm{S}}\ n\ ([\mathbb{N}_{\mathbf{sind}}]_b\ \ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n) \\ [\mathbb{N}_{\mathbf{sind}}]_b\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (\beta_{\mathbb{N}}\ i\ k) & := & \omega_b\end{array}$$

Theorem 3.2.6:
The branching translation provides a syntactic model of BTT.

3.2.4. Algebraic Parametricity Translation

Following *Escardó*, we now have to relate the two translations. We achieve this through a third layer of *algebraic parametricity*. Intuitively, every type $A : \square$ is translated as a predicate

$$[\![A]\!]_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square.$$

As it was the case for parametricity in Section 3.1, $\alpha : \mathbb{Q}$ is implicitly part of the context, like in the axiom model. As explained above, we also ask for the predicate to be $\mathfrak{D}$-algebraic in the sense that it must be equipped with a proof

$$\begin{aligned}\beta^\varepsilon_A : \quad & \Pi(x_a : [\![A]\!]_a)\,(i : \mathbb{I})\,(k : \mathbb{O}\,i \to [\![A]\!]_b).\\ & [\![A]\!]_\varepsilon\ x_a\ (k\ (\alpha\ i)) \to [\![A]\!]_\varepsilon\ x_a\ (\beta_A\ i\ k).\end{aligned}$$

We will write the type of such algebraic parametricity predicates as

$$\begin{aligned}\square^\varepsilon\ (A_a : [\![\square]\!]_a)\ (A_b : [\![\square]\!]_b) := \Sigma(A_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square).\\ \Pi(x_a : [\![A]\!]_a)\,(i : \mathbb{I})\,(k : \mathbb{O}\,i \to [\![A]\!]_b).\\ A_\varepsilon\ x_a\ (k\ (\alpha\ i)) \to A_\varepsilon\ x_a\ (\beta_A\ i\ k)\end{aligned}$$

Just as we did for the branching translation, given $A : \square_i$ we define separately the predicate $[\![A]\!]_\varepsilon$ and the proof of parametric algebraicity β^ε_A. We define the translation in Figure 3.7. As before we also ask for a dummy algebraic predicate

$$\mho_\varepsilon : \Pi(A : \square).\, \square^\varepsilon\ A\ \mho_b$$

which can be taken to be always a trivially inhabited predicate, together with an arbitrary proof

$$\omega_\varepsilon : \Pi(A : \square)\,(x : A).\,(\mho_\varepsilon\ A).\pi_1\ x\ \omega_b.$$

Formalization of the algebraic parametricity translation can be found here.

Theorem 3.2.7: CC_ω Soundness
We have the following.

1. If $t \equiv_{\mathrm{CC}_\omega} u$ then $[t]_\varepsilon \equiv_{\mathcal{T}} [u]_\varepsilon$.
2. If $\Gamma \vdash^{\mathrm{CC}_\omega} t : A$ then $[\![\Gamma]\!]_\varepsilon \vdash^{\mathcal{T}} [t]_\varepsilon : [\![A]\!]_\varepsilon\ [t]_a\ [t]_b$.

The algebraic parametricity translation of inductive types sticks closely to the branching one. Given an inductive type $\mathcal{I}$, we create an inductive type $\mathcal{I}_\varepsilon$ whose constructors are the pointwise $[\![\cdot]\!]_\varepsilon$ translation of those of $\mathcal{I}$. An additional constructor $\beta^\varepsilon_\mathcal{I}$ freely implements the algebraicity requirement. Since $\alpha : \mathbb{Q}$ is implicitly part of the translated context, we have to take it as a parameter of the translated inductive type and explicitly pass it as an argument when interpreting those types and their proof of algebraicity. We give the translation on our running example in Figure 3.8. Once again, parameters and indices present no particular problem and are handled similarly to *Pédrot and Tabareau* [121].

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

$$
\begin{array}{lcl}
[\![\square]\!]_\varepsilon & := & \lambda(A_a : [\![\square]\!]_a)\,(A_b : [\![\square]\!]_b).\,\square^\varepsilon\ A_a\ A_b \\
\beta^\varepsilon_\square & := & \lambda(A_a : [\![\square]\!]_a)\,(i : \mathrm{I})\,(k : \mathrm{O}\ i \to [\![\square]\!]_b)\,(A_\varepsilon : [\![\square]\!]_\varepsilon\ A_a\ (k\ (\alpha\ i))).\,\mho_\varepsilon\ A_a \\
[x]_\varepsilon & := & x_\varepsilon \\
[\lambda x : A.\,t]_\varepsilon & := & \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[t]_\varepsilon \\
[t\ u]_\varepsilon & := & [t]_\varepsilon\ [u]_a\ [u]_b\ [u]_\varepsilon \\
[\![\Pi x : A.\,B]\!]_\varepsilon & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(f_b : [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \Pi(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[\![B]\!]_\varepsilon\ (f_a\ x_a)\ (f_b\ x_b) \\
\beta^\varepsilon_{\Pi x:A.\,B} & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(i : \mathrm{I})\,(k : \mathrm{O}\ i \to [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \lambda(f_\varepsilon : [\![\Pi x : A.\,B]\!]_\varepsilon\ f_a\ (k\ (\alpha\ i))). \\
 & & \quad\quad \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b). \\
 & & \quad\quad\quad \beta^\varepsilon_B\ (f_a\ x_a)\ i\ (\lambda(o : \mathrm{O}\ i).\,k\ o\ x_b)\ (f_\varepsilon\ x_a\ x_b\ x_\varepsilon) \\
[\![A]\!]_\varepsilon & := & [A]_\varepsilon.\pi_1
\end{array}
$$

$$
\begin{array}{lcl}
[\![\cdot]\!]_\varepsilon & := & \alpha : \mathrm{Q} \\
[\![\Gamma, x : A]\!]_\varepsilon & := & [\![\Gamma]\!]_\varepsilon,\ x_a : [\![A]\!]_a,\ x_b : [\![A]\!]_b,\ x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b
\end{array}
$$

Figure 3.7.: Algebraic parametricity translation (negative fragment)

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}_\varepsilon\ (\alpha : \mathrm{Q}) : \mathbb{N} \to \mathbb{N}_b \to \square := \\
\quad |\ \mathrm{O}_\varepsilon \ :\ \mathbb{N}_\varepsilon\ \alpha\ \mathrm{O}\ \mathrm{O}_b \\
\quad |\ \mathrm{S}_\varepsilon \ :\ \Pi(n_a : \mathbb{N})\,(n_b : \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ n_b).\,\mathbb{N}_\varepsilon\ \alpha\ (\mathrm{S}\ n_a)\ (\mathrm{S}_b\ n_b) \\
\quad |\ \beta^\varepsilon_{\mathbb{N}} \ :\ \Pi(n_a : \mathbb{N})\,(i : \mathrm{I})\,(k : \mathrm{O}\ i \to \mathbb{N}_b). \\
\qquad\quad \mathbb{N}_\varepsilon\ \alpha\ n_a\ (k\ (\alpha\ i)) \to \mathbb{N}_\varepsilon\ \alpha\ n_a\ (\beta_{\mathbb{N}}\ i\ k)
\end{array}
$$

Figure 3.8.: Algebraic parametricity translation for $\mathbb{N}$

Theorem 3.2.8:
For any inductive type $\mathscr{I}$, its algebraic parametricity translation $\mathscr{I}_\varepsilon$ is well typed and satisfies the positivity criterion.

Formalization of the algebraic parametricity translation for some inductive types can be found here.

Corollary 3.2.9: Typing soundness
Typing soundness holds for the translation of inductive types and their constructors.

As for the branching translation, we retrieve a restricted form of dependent elimination based on storage operators. The argument is virtually the same, but now at the level of parametricity, which makes the syntactic burden even heavier since we now have everything repeated three times. Once again, we rely on the following shorthand for binders:

$$\langle x : A \rangle := x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b$$

and similarly for application to variables.

We give the eliminators for our running example in this lighter syntax,

which is already the limit of what can be done on paper.

$$\begin{aligned}[\mathbb{N}_{\mathbf{rec}}]_\varepsilon \quad : \quad & \Pi\langle P : \square\rangle\langle p_{\mathrm{O}} : P\rangle\langle p_{\mathrm{S}} : \mathbb{N} \to P \to P\rangle\langle n : \mathbb{N}\rangle. \\ & [\![P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_b\end{aligned}$$

$$\begin{aligned}
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ \mathrm{O}_\varepsilon & := p_{\mathrm{O}\varepsilon} \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ (\mathrm{S}_\varepsilon\ \langle n\rangle) & := p_{\mathrm{S}\varepsilon}\ n_\varepsilon\ ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \langle n\rangle) \\
[\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ (\beta^\varepsilon n_a\ i\ k\ n_\varepsilon) & := \beta^\varepsilon_P\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a \\
& \qquad i\ (\lambda o.\ [\mathbb{N}_{\mathbf{rec}}\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ (k\ o)]_b) \\
& \qquad ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ n_a\ (k\ (\alpha\ i))\ n_\varepsilon)
\end{aligned}$$

Note that the $\beta^\varepsilon_{\mathbb{N}}$ case explicitly calls the global axiom α to relate the oracular term with the branching one. This is one of the few places that introduce an actual use of the oracle in the translation, by opposition to merely passing it around.

We define $[\mathbb{N}_{\mathbf{str}}]_\varepsilon$ as before, using the fact it is given directly in the source in terms of $\mathbb{N}_{\mathbf{rec}}$. In particular we do not have to write its translation explicitly. Finally, we can define the dependent eliminators, following the same structure as before.

$$\begin{aligned}[\mathbb{N}_{\mathbf{sind}}]_\varepsilon \quad : \quad & \Pi\langle P : \mathbb{N} \to \square\rangle\langle p_{\mathrm{O}} : P\ \mathrm{O}\rangle\langle p_{\mathrm{S}} : \Pi(n : \mathbb{N}).\ \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathrm{S}\ n)\ P\rangle. \\ & \Pi\langle n : \mathbb{N}\rangle.\ [\![\mathbb{N}_{\mathbf{str}}\ n\ P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{sind}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a\ [\mathbb{N}_{\mathbf{sind}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_b\end{aligned}$$

$$\begin{aligned}
[\mathbb{N}_{\mathbf{sind}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ \mathrm{O}_\varepsilon & := p_{\mathrm{O}\varepsilon} \\
[\mathbb{N}_{\mathbf{sind}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ (\mathrm{S}_\varepsilon\ \langle n\rangle) & := p_{\mathrm{S}\varepsilon}\ \langle n\rangle\ ([\mathbb{N}_{\mathbf{sind}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ \langle n\rangle) \\
[\mathbb{N}_{\mathbf{sind}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathrm{O}}\rangle\ \langle p_{\mathrm{S}}\rangle\ _\ _\ (\beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon) & := \omega_\varepsilon\ (P_a\ n_a)\ [\mathbb{N}_{\mathbf{sind}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n]_a
\end{aligned}$$

Following the results from *Pédrot and Tabareau* [121], this translation can be generalized to any inductive type, potentially with parameters and indices. Indeed, it is very similar to the composition of weaning with binary parametricity (the difference coming from the fact that we do not encode the same effect in the axiom translation and the branching translation).

[121]: Pédrot et al. (2017), "An effectful way to eliminate addiction to dependence"

Theorem 3.2.10:
The algebraic parametricity translation is a syntactic model of BTT.

3.3. Continuity of functionals

This Section is dedicated to the proof of the main theorem which we formally state below.

Theorem 3.3.1:
If

$$\vdash^{\mathcal{S}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

in the source theory $\mathcal{S} \coloneqq \mathsf{BTT}$ then

$$\vdash^{\mathcal{T}} \_ : \mathscr{C}_{\mathfrak{D}}\, f$$

in the target theory $\mathcal{T} \coloneqq \mathsf{CIC}$.

Continuity of functionals is formally proven here.

Proof. The proof follows the same structure as *Escardó*'s proof for System T, as presented in Section 3.1.6. Once again, we will fix henceforth the oracular type parameters for the remainder of this Section as

$$\mathbf{I} \coloneqq \mathbb{N} \qquad \text{and} \qquad \mathbf{O} \coloneqq \lambda(\_ : \mathbf{I}).\, \mathbb{N}.$$

As our parametricity translation is essentially the same as in Section 3.1.6, we can derive the same lemmas:

Proposition 3.3.2: Unicity of specification
There is a proof

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbf{Q})\, \langle n : \mathbb{N} \rangle.\, n_a = \partial^{\mathbb{N}}\, n_b\, \alpha.$$

Unicity of specification is formally proven here.

Proof. By induction on n_ε. ∎

Proposition 3.3.3: Generic parametricity
There is a proof

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbf{Q})\, (n_b : \mathbb{N}_b).\, \mathbb{N}_\varepsilon\, \alpha\, (\partial^{\mathbb{N}}\, n_b\, \alpha)\, n_b.$$

Generic parametricity is formally proven here.

Proof. By induction on n_b. ∎

We now define the generic element. Once again, it is the same definition as in Section 3.1.6.

Definition 3.3.1: Generic tree
We define in $\mathscr{T}$ the *generic tree* $\mathfrak{t}$ as

$$\begin{array}{lll} \mathfrak{t} & : & \mathbb{N} \to \mathbb{N}_b \\ \mathfrak{t} & := & \lambda(n : \mathbb{N}).\, \beta_{\mathbb{N}}\, n\, \eta_{\mathbb{N}} \end{array}$$

where

$$\begin{array}{lll} \eta_{\mathbb{N}} & : & \mathbb{N} \to \mathbb{N}_b \\ \eta_{\mathbb{N}}\, \mathrm{O} & := & \mathrm{O}_b \\ \eta_{\mathbb{N}}\, (\mathrm{S}\, n) & := & \mathrm{S}_b\, (\eta_{\mathbb{N}}\, n) \end{array}$$

Generic trees still retain the following property:

Lemma 3.3.4: Fundamental property of the generic tree
We have a proof

$$\vdash^{\mathscr{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\, (n : \mathbb{N}).\, \partial^{\mathbb{N}}\, (\mathfrak{t}\, n)\, \alpha = \alpha\, n,$$

where $\partial^{\mathbb{N}}$ was defined in Section 3.1.4.

Proof. Immediate by the definition of the $\partial^{\mathbb{N}}$ function. ∎

Definition 3.3.2: Generic element
We define the generic element $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$ as follows.

$$\gamma_b\, n_b \quad := \quad \gamma_0\, \mathrm{O}\, n_b$$

where

$$\begin{array}{lll} \gamma_0 & : & \mathbb{N} \to \mathbb{N}_b \to \mathbb{N}_b \\ \gamma_0\, a\, \mathrm{O}_b & := & \mathfrak{t}\, a \\ \gamma_0\, a\, (\mathrm{S}_b\, n_b) & := & \gamma_0\, (\mathrm{S}\, a)\, n_b \\ \gamma_0\, a\, (\beta_{\mathbb{N}}\, i\, k) & := & \beta_{\mathbb{N}}\, i\, (\lambda o : \mathbb{N}.\, \gamma_0\, a\, (k\, o)). \end{array}$$

The generic element γ_b is formally defined here, making use of an auxiliary function γ_0 formally defined here.

The generic element still retains its crucial property:

Lemma 3.3.5: Fundamental property of the generic element
We have a proof

$$\vdash^{\mathscr{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\, (n_b : \mathbb{N}_b).\, \partial^{\mathbb{N}}\, (\gamma_b\, n_b)\, \alpha = \alpha\, (\partial^{\mathbb{N}}\, n_b\, \alpha).$$

A slight generalization of the fundamental property of the generic element (proving it for γ_0) is formalized here.

Proof. Straightforward by induction on n_b, using Lemma 3.3.4 for the O_b case. ∎

Proposition 3.3.6:
The γ_b term can be lifted to a function $\gamma : \mathbb{N} \to \mathbb{N}$ in the source theory.

Parametricity for the generic element is formalized here.

Proof. The proof is the same as in Section 3.1.6. ∎

We can now get to the proof of the main result.

This last bit of proof is formalized here.

Let

$$\vdash^{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.$$

Since $\gamma : \mathbb{N} \to \mathbb{N}$ can be reflected from the model into BTT, we can consider the term

$$\vdash^{\mathsf{BTT}} f\,\gamma : \mathbb{N}.$$

By soundness, it results in the three terms below.

$$\begin{array}{lll} \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & [f]_a\,\alpha : \mathbb{N} \\ & \vdash^{\mathcal{T}} & [f]_b\,\gamma_b : \mathbb{N}_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash^{\mathcal{T}} & [f]_\varepsilon\,\alpha\,\gamma_b\,\gamma_\varepsilon : \mathbb{N}_\varepsilon\,\alpha\,([f]_a\,\alpha)\,([f]_b\,\gamma_b) \end{array}$$

Applying Proposition 3.3.2 to $[f]_a$, $[f]_b$ and $[f]_\varepsilon$, we get:

$$\vdash^{\mathcal{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N}).\,[f]_a\,\alpha = \partial^{\mathbb{N}}\,([f]_b\,\gamma_b)\,\alpha$$

Since f is a term in BTT that does not use any impure extension of the model, it is easy to check that $[f]_a \equiv f$. Therefore, f is dialogue continuous, which concludes our proof. ∎

3.4. Discussion and Related Work

Before discussing other models that can be found in the literature, let us already stress the fact that our model can be generalized to prove continuity of many functionals, not only of the form

$$\vdash^{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.$$

For instance, dialogue continuity for functionals

$$\vdash^{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

can be easily derived by instantiating our model with

$$\mathbf{I} := \mathbb{N} \qquad \text{and} \qquad \mathbf{O} := \lambda\_ : \mathbb{N}.\,\mathbb{B}.$$

This was already pointed out by *Escardó* [51] who makes use of his model to retain uniform continuity of System T definable functionals

$$\vdash^{\mathsf{T}} f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}.$$

Indeed, as we showed in Section 2.2.1, uniform continuity and dialogue continuity are equivalent on the Cantor space.

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

3.4.1. Comparison with Similar Models

As already stated, our proof follows the argument given by *Escardó* [51] for System T, which can also be found as a close variant by Sterling that uses streams instead of trees [130]. Yet, in order to scale to BTT there are a few non-trivial technical differences in our version. We already discussed them along the way but they ought to be highlighted and summarized here.

[51]: Escardó (2013), "Continuity of Gödel's System T Definable Functionals via Effectful Forcing"

[130]: Sterling (2021), "Higher order functions and Brouwer's thesis"

The first obvious one is that our model is a program translation of BTT into CIC, while *Escardó*'s is a model in a type-theoretic metatheory. Using our definitions from Chapter 1, this means that in our case the source theory is BTT, the target theory is CIC and the meta-theory can be very weak, as proving the soundness of the translation does not require powerful logical principles. In *Escardó*'s case, however, the source language is embedded as an AST in the meta-theory, meaning there is not clear distinction between the target theory and the meta-theory. If we wanted to use this technique for BTT, we would need strong principles to internalize type theory inside itself.

Another major divergence is that parametricity predicates must be compatible with the $\mathfrak{D}$-algebra structure of the underlying types. This is needed to interpret large elimination, which is absent from System T. This requirement did appear in *Escardó*'s proof, but only as a tool to prove existence of the generic element, and not as a requirement of the model. Algebraicity of predicates was a surprising part of the model design, but in hindsight it is obvious that it would pop up eventually. Furthermore, both to preserve conversion and to scale to richer inductive types, the parametricity predicates need to be given in an inductive way following the underlying source type, rather than as an ad-hoc equality between two terms.

Escardó and Xu [50, 147] also gave related models to internalize uniform continuity. Contrarily to the above one, they build these models out of sheaves, which have also been used similarly by *Coquand and Jaber* [42, 43]. *Escardó and Xu* emphasize that since the universe of sheaves is not a sheaf in general, they only implement a small fragment of MLTT. In more recent work, *Gratzer et al* [67] proved the existence of universes in every sheaf category, albeit using classical logic in the meta-theory. We have several remarks to make. First, *Rijke et al* [125] showed that, assuming univalence and HITs in the target theory, one can build a syntactic sheaf model of MLTT. Univalence is typically needed to relax the strict uniqueness requirement of sheaves.

[50]: Escardó et al. (2016), "A constructive manifestation of the Kleene-Kreisel continuous functionals"

[147]: Xu et al. (2013), "A Constructive Model of Uniform Continuity"

[42]: Coquand et al. (2012), "A Computational Interpretation of Forcing in Type Theory"

[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"

[67]: Gratzer et al. (2022), "Strict universes for Grothendieck topoi"

[125]: Rijke et al. (2020), "Modalities in homotopy type theory"

Moreover, as we showed in Section 2.5, the sheafification operator

$$
\begin{aligned}
&\texttt{Inductive}\ \mathfrak{S}\ (A : \square_i) : \square_i := \\
&\mid \eta : A \to \mathfrak{S}\ A \\
&\mid \beta : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to \mathfrak{S}\ A) \to \mathfrak{S}\ A \\
&\mid \varepsilon : \Pi(i : \mathrm{I})\ (x : \mathfrak{S}\ A).\beta\ i\ (\lambda\_ : \mathrm{O}\ i.\ x) = x.
\end{aligned}
$$

is very close to $\mathfrak{D}$, which leads us to challenge *Escardó*'s claim that the dialogue model is not a sheaf model. Otherwise said, the dialogue monad is an impure variant of the sheafification monad, giving a curious and unexpected double entendre to the phrase *effectful forcing*.

Rahli et al [123] give another proof of uniform continuity for NuPRL using a form of delimited exceptions. Computationally, their model tracks accesses to arguments of functions by passing them exception-raising placeholders. The control flow is inverted w.r.t. our model, as it requires non-terminating realizers, but we believe that the fundamental mechanism is similar. In the same context *Rahli et al* [124] define a sheaf model with bar induction in mind, but this principle is inextricably tied to uniform continuity [22, 24].

[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"

[124]: Rahli et al. (2019), "Bar Induction is Compatible with Constructive Type Theory"

[22]: Berger (2006), "The Logical Strength of the Uniform Continuity Theorem"

[24]: Bickford et al. (2018), "Computability Beyond Church-Turing via Choice Sequences"

3.4.2. Internalization

In this Chapter we have constructed a model of BTT that associates to every closed term

$$\vdash^{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

a proof in CIC that it is continuous. Can we do better? First, we know that there is a major limitation. Indeed, as explained in Section 2.4.2, MLTT extended with the internal statement

$$\Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, \mathscr{C}\ f,$$

stating that all functionals on the Baire type are standard continuous, results in an inconsistent theory. We will call this property *internal continuity* below. The proof crucially relies on two ingredients, namely congruence of conversion and large dependent elimination. Thus, there might be hope for BTT where the latter is restricted. However, we have the following:

Theorem 3.4.1:
Through our program translation, internal continuity in the source theory $\mathscr{S}$ implies internal continuity in the target theory $\mathscr{T}$.

Proof. Let us assume a term

$$\vdash^{\mathscr{S}} M : \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, \mathscr{C}\ f$$

in the source theory. Then its axiom translation $[M]_a$ is a proof of

$$\alpha : \mathbb{N} \to \mathbb{N} \vdash^{\mathscr{T}} [M]_a : [\![\Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, \mathscr{C}\ f]\!]_a.$$

However, the type of M does not mention α. Hence, if we substitute α by a concrete function, as for instance $\lambda n : \mathbb{N}.\, n$, we get

$$\vdash^{\mathscr{T}} [M]_a\{\alpha := \lambda n : \mathbb{N}.\, n\} : \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\, \mathscr{C}\ f,$$

which is exactly internal continuity in the target theory. ∎

This is obviously disappointing, since it implies that $\mathscr{T}$ is inconsistent. One can then wonder if it is possible to aim for a middle ground, where we keep the computation of the modulus in the target, but reflect in the source theory a proof of continuity for every concrete functional. That is, construct in the target theory a term

$$\vdash^{\mathscr{T}} M : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle. [\![\mathscr{C}\ f]\!],$$

where $[\![A]\!]$ stands for the triple

$$\Sigma(x_a : [\![A]\!]_a)(x_b : [\![A]\!]_b). [\![A]\!]_\varepsilon\ x_a\ x_b.$$

Then, given a term

$$\vdash^{\mathscr{S}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N},$$

from the source, we would recover

$$\vdash^{\mathscr{T}} M\ f : [\![\mathscr{C}\ f]\!]$$

in the target, which would be reflected as a symbol

$$\vdash^{\mathscr{S}} M_f : \mathscr{C}\ f$$

in the source, resulting in a theory where every function f can be given a proof of continuity M_f, without building a term

$$\vdash^{\mathscr{S}} \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}. \mathscr{C}\ f.$$

The implication regarding the target theory is a bit more subtle.

Proposition 3.4.2:
If we have a term

$$M : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle. [\![\mathscr{C}\ f]\!]$$

in the target theory, then the target theory negates funext.

This perhaps surprising result comes from the following lemma:

Lemma 3.4.3:
If we have

$$\vdash^{\mathscr{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle. [\![\mathscr{C}\ f]\!]$$

then we can also get a proof that

$$\vdash^{\mathscr{T}} \_ : \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}. f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f \to \mathscr{C}\ f,$$

where

$$f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} g \quad := \quad \Pi(u\ v : \mathbb{N} \to \mathbb{N}). \\ (\Pi n : \mathbb{N}. u\ n = v\ n) \to f\ u = f\ v$$

is the canonical setoid equality on the functional type.

Proof. Let $\alpha : \mathbb{N} \to \mathbb{N}$ in $\mathscr{T}$ and let

$$\vdash^{\mathscr{T}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \qquad \text{such that} \qquad f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f.$$

We define the following:

$$\begin{array}{lcl} \tilde{f} & : & [\![(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}]\!] \qquad \text{such that} \\ [\tilde{f}]_a & := & f \\ [\tilde{f}]_b & := & \lambda(u_b : [\![\mathbb{N}]\!]_b \to [\![\mathbb{N}]\!]_b). \\ & & \quad \eta_{\mathbb{N}}\ (f\ (\lambda n : \mathbb{N}.\, \partial^{\mathbb{N}}\ (u_b\ (\eta_{\mathbb{N}}\ n))\ \alpha)) \end{array}$$

To implement parametricity $[\tilde{f}]_\varepsilon$, we need to prove that

$$\Pi\langle u : \mathbb{N} \to \mathbb{N}\rangle.\ f\ u_a = f\ (\lambda n : \mathbb{N}.\, \partial^{\mathbb{N}}\ (u_b\ (\eta_{\mathbb{N}}\ n))\ \alpha).$$

Since $f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f$, it is sufficient to prove that

$$\Pi n : \mathbb{N}.\, u_a\ n = \partial^{\mathbb{N}}\ (u_b\ (\eta_{\mathbb{N}}\ n))\ \alpha,$$

which is done by induction on $u_\varepsilon\ (n, \eta_{\mathbb{N}}\ n, \_)$.

Finally, if we have a term of type

$$\Pi(\alpha : \mathbb{N} \to \mathbb{N})\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle.\, [\![\mathscr{C}\ f]\!],$$

then we have $[\![\mathscr{C}\ \tilde{f}]\!]$ and thus $\mathscr{C}\ f$ by projection. ∎

Hence, if our target theory features funext, such a way to reflect the modulus of continuity in the source implies continuity of all

$$f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

in the target. Thus, by the above theorem, we get a proof of false.

Otherwise said, if we were able to build such a model, we would be able to negate funext in our target theory. As funext is independent from CIC, it is in particular impossible to negate funext in CIC. Therefore, such a model is out of reach with CIC as target theory.

Hence, *Escardó et al*'s diagonalization argument in CIC prevents us from extending this model to an internal modulus of continuity. However, it is still unclear whether it is possible to construct a similar paradox for BTT, or if there exists another model of it with internal modulus of continuity. This is still an open question. Still, we conjecture that adding an additional layer of presheaves to allow a varying number of oracles in the context could be key to build such a model.

Indeed, adding a modal type of exceptions to MLTT is precisely what permits to go from the external Markov's rule [122] to the internal Markov's principle [115]. If we were able to locally create a fresh generic element independent from all the previously allocated ones, it seems that we could turn the external continuity rule into an internal one, mimicking what happens when

Fresh exceptions are precisely used by *Rahli et al* [123] to get what amounts to an independent generic element at every call, so this argument does not seem far-fetched. We leave this to future work.

[122]: Pédrot et al. (2018), "Failure is Not an Option An Exceptional Type Theory"

[115]: Pédrot (2020), "Russian Constructivism in a Prefascist Theory"

[123]: Rahli et al. (2018), "Validating Brouwer's continuity principle for numbers using named exceptions"

3.4.3. Extension to MLTT

Our model provides some insights about what kind of properties we would need to build a model of MLTT proving continuity. Indeed, in the process of proving continuity for full-blown MLTT, the same arch-enemy always stands in our way: how do we derive the following sequent ?

$$\frac{\Gamma, x : \mathbb{B} \vdash P \qquad \Gamma \vdash t_{\mathsf{true}} : P\{x \coloneqq \mathsf{true}\} \qquad \Gamma \vdash t_{\mathsf{false}} : P\{x \coloneqq \mathsf{false}\} \qquad \Gamma \vdash i : \mathbf{I} \qquad \Gamma \vdash k : \mathbf{O}\, i \to \mathbb{B}}{\Gamma \vdash \mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, (\beta_{\mathbb{B}}\, i\, k) : P\{x \coloneqq \beta_{\mathbb{B}}\, i\, k\}}$$

What makes our world go wrong and led us to building a model of BTT is unjustified discrimination of $\beta_{\mathbb{B}}$ by predicates. To circumvent this issue, we could try and ask that P simply propagates the effect in the following way:

$$P\{x \coloneqq \beta_{\mathbb{B}}\, i\, k\} \equiv \beta_{\square}\, i\, (\lambda o : \mathbf{O}\, i.\, P\, (k\, o))$$

We would call such a predicate *strict*.

Now, if we had at our disposal a model where every predicate were strict, then $\beta_{\square}$ would no more need to return a dummy value

$$\mho : \square,$$

but something more meaningful. For instance, we could take

$$\beta_{\square}\, i\, k \coloneqq \Pi o : \mathbf{O}\, i.\, k\, o.$$

Then again, given such a *strict predicate* P and such a definition for $\beta_{\square}$, a simple way to inhabit

$$P\{x \coloneqq \beta_{\mathbb{B}}\, i\, k\}$$

would be the following:

$$\mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, (\beta_{\mathbb{B}}\, i\, k) \coloneqq \lambda o : \mathbf{O}\, i.\, \mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, (k\, o) : \Pi o : \mathbf{O}\, i.\, P\, (k\, o).$$

All of this makes sense:

$$\beta_{\mathbb{B}}\, i\, k : \mathbb{B}$$

is a *stored computation* waiting for an oracle to collapse to a boolean value. Therefore, P should not be able to distinguish between $\beta_{\mathbb{B}}\, i\, k$ and either of its branches. Consequently, if we were able to inhabit any branch the oracle may choose, we should be able to inhabit the node. Hence our definition of $\beta_{\square}\, i\, k$ as a Π-type.

In fact, this is essentially what allows sheaf models to feature dependent elimination. Indeed, let us look once again at the sheafification operator

$$
\begin{aligned}
&\texttt{Inductive}\ \mathfrak{S}\ (A : \square_i) : \square_i :=\\
&\mid \eta : A \to \mathfrak{S}\ A\\
&\mid \beta : \Pi(i : \mathrm{I}).\ (\mathrm{O}\ i \to \mathfrak{S}\ A) \to \mathfrak{S}\ A\\
&\mid \varepsilon : \Pi(i : \mathrm{I})\ (x : \mathfrak{S}\ A).\beta\ i\ (\lambda\_ : \mathrm{O}\ i.\ x) = x.
\end{aligned}
$$

Note that the ε constructor enables us to prove that

$$P\ (\beta_{\mathrm{B}}\ i\ k) = \beta_{\square}\ i\ (\lambda o : \mathrm{O}\ i.\ P\ (k\ o)).$$

Indeed, we are in a setting where $\mathrm{O}\ i$ is proof-irrelevant. Hence, we have:

$$
\begin{aligned}
P\ (\beta_{\mathrm{B}}\ i\ k) &= \beta_{\square}\ i\ (\lambda o : \mathrm{O}\ i.\ P\ (\beta_{\mathrm{B}}\ i\ k))\\
&= \beta_{\square}\ i\ (\lambda o : \mathrm{O}\ i.\ P\ (\beta_{\mathrm{B}}\ i\ (\lambda\_ : \mathrm{O}\ i.\ k\ o))\\
&= \beta_{\square}\ i\ (\lambda o : \mathrm{O}\ i.\ P\ (k\ o)).
\end{aligned}
$$

However, even delving as deep as we could in the arcane of *strict parametricity* and making use of the various capabilities of *SProp* [59], we were not able to build a program translation of MLTT into CIC satisfying these rules. The best asset of program translations, the fact that *conversion in the source theory is interpreted as conversion in the target theory*, shows its inner weakness: when we need more *control over computation*, we cannot achieve it by that means.

[59]: Gilbert et al. (2019), "Definitional Proof-Irrelevance without K"

Part III.

Future

4. The cone of possibilities

In this chapter, we face our desire for control and take a whack at bridging the authoritarian gap. Building upon *Coquand and Jaber*'s work on continuity [42, 43], later followed by *Coquand and Mannaa* [44], we define a theory where typing and conversion judgments are definitionally sheaves over the Cantor space.

However, as we are doing a type-theoretical variant of sheaves that is more intensional than what can be found in the categorical world, we believe there is need for another name than sheaves to describe our theory. A similar endeavour was done by *Pédrot* [115] to give a type-theoretic version of *presheaves*. In this work, *Prédrot* notices that sheaves were first introduced in French as "faisceaux", itself derived from the latin word *fascis*. Sticking closely to the absolute laws of etymology, *Pédrot* names *prefascist* theory his type-theoretical account of presheaves. Walking in his footsteps and acknowledging that the political overtone is in tune with our need for control on terms, we devise the following battle plan:

[42]: Coquand et al. (2012), "A Computational Interpretation of Forcing in Type Theory"
[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"
[44]: Coquand et al. (2017), "The Independence of Markov's Principle in Type Theory"
[115]: Pédrot (2020), "Russian Constructivism in a Prefascist Theory"

Theoretical fascism We describe in Section 4.1 a theory dubbed ꟻTT (read "split TT" or "digamma TT"). This theory displays a formal oracle

$$\mathfrak{f} : \mathbb{N} \to \mathbb{B},$$

as in the *axiom translation* of the previous chapter. In ꟻTT however, typing and conversion rules are moreover indexed with *forcing conditions [38]*

$$\ell, \ell' : \mathsf{list}\,(\mathbb{N} \times \mathbb{B}),$$

so as to make them *sheaves* over the Cantor space. ꟻTT was already described in a note by *Coquand and Jaber* [43], later developed by the same authors on the particular case of System T [42], then by *Coquand and Mannaa* [44] to show independence of Markov's principle in type theory.

[38]: Cohen (1963), "The independence of the continuum hypothesis"
[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"
[42]: Coquand et al. (2012), "A Computational Interpretation of Forcing in Type Theory"
[44]: Coquand et al. (2017), "The Independence of Markov's Principle in Type Theory"

Continuous fascism In Section 4.2, we explain how, assuming normalization of ꟻTT, we recover continuity of all MLTT-definable functionals

$$\vdash^{\mathsf{MLTT}} f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}.$$

Mainly, the normalization result we hope for entails that for any term

$$\vdash^{\mathsf{ꟻTT}} t : \mathbb{N},$$

potentially making use of $\mathfrak{f}$, the list ℓ of all the queries that t asks $\mathfrak{f}$ is finite. This is exactly uniform continuity as described in Section 2.2.1, which is equivalent to dialogue continuity in the case of the Cantor space, as shown in Proposition 2.2.2.

Normal fascism In Section 4.4, we prove normalization for System $\digamma\mathsf{T}$, an extension of System T with the same oracle

$$\mathfrak{f} : \mathbb{N} \to \mathbb{B}$$

and the same forcing conditions

$$\ell, \ell' : \mathsf{list}\,(\mathbb{N} \times \mathbb{B})$$

as $\digamma\mathsf{TT}$. System $\digamma\mathsf{T}$ serves as a simple example of the normalization technique we are going to use for $\digamma\mathsf{TT}$, and results of Section 4.4 are formalized in Coq.

Finally, in Section 4.5 we describe an unfinished proof of normalization for $\digamma\mathsf{TT}$. We make use of a Coq development of *Adjedj et al* [4], itself inspired by *Abel et al [3]*. At the time of writing, the formalization is still an ongoing work.

[4]: Adjedj et al. (2023), "Martin-Löf à la Coq"

[3]: Abel et al. (2017), "Decidability of Conversion for Type Theory in Type Theory"

4.1. Why you should buy $\digamma\mathsf{TT}$

The aim of this Section is to explain and justify $\digamma\mathsf{TT}$ as a type theory worthy of our attention.

Section 4.1.1 provides a birdeye's view of the theory, as well as the intuition behind its definition.

To make things more formal, a complete list of rules for $\digamma\mathsf{TT}$ is displayed in Section 4.1.2.

4.1.1. A brief tour around $\digamma\mathsf{TT}$

We define $\digamma\mathsf{TT}$, a variant of MLTT extended with a formal oracle

$$\mathfrak{f} : \mathbb{N} \to \mathbb{B}.$$

As in the *axiom translation*, $\mathfrak{f}$ is a black-box, akin to a variable, and when applied to some term $t : \mathbb{N}$ it usually blocks computation. However, in $\digamma\mathsf{TT}$, we are able to gradually *unblock* computation, and store the resulting values in *forcing conditions* ℓ, ℓ'.

Forcing conditions are lists of pairs of natural numbers and booleans:

$$\ell : \mathsf{list}\,(\mathbb{N} \times \mathbb{B}).$$

They represent calls to the $\mathfrak{f}$ oracle, and the answers it already produced. We use lists in the formalization as they are available in Coq and easy to work with, but the order of pairs will not matter. Our reader may thus think of forcing conditions as *finite sets*. Backing this perception, we write

$$(n, b) \in \ell$$

to mean that a pair (n, b) is present in ℓ.

Forcing conditions, as their elements, live in the meta-theory. We write n and b in the meta-theory and add an overline bar $\overline{\mathsf{n}}$ and $\overline{\mathsf{b}}$ when we inject them into the object theory.

As lists represent finite sets, any sequent derivable with a list ℓ can also be derived with any other list ℓ' containing the same elements at different positions. Hence, our order $\ell' \preceq \ell$ will be defined as the *inclusion* of ℓ in ℓ':

$$\ell' \preceq \ell := \Pi \mathsf{n}\, \mathsf{b}.\, (\mathsf{n}, \mathsf{b}) \in \ell \longrightarrow (\mathsf{n}, \mathsf{b}) \in \ell'.$$

The fact that our order is the converse of inclusion comes from the fact that the smaller a list ℓ is, the coarser a forcing condition it is. For instance, the empty list nil defines the bigger world of all, one that encompasses every other. Hence, for any ℓ, we get $\ell \preceq \mathsf{nil}$.

If we have a pair

$$(\mathsf{n}, \mathsf{b}) \in \ell,$$

that means that $\mathfrak{f}$ has already been called on $\overline{\mathsf{n}}$, and we live in a world where it answered $\overline{\mathsf{b}}$.

Hence, any future call to $\overline{\mathsf{n}}$ will entail

$$\Gamma \vdash_\ell \mathfrak{f}\, \overline{\mathsf{n}} \equiv \overline{\mathsf{b}} : \mathbb{B}.$$

This conversion rule for $\mathfrak{f}$ is presented in Tab 4.1. To give a simple example,

$$\text{if} \qquad (\mathsf{O}, \mathsf{true}) \in \ell \qquad \text{then} \qquad \Gamma \vdash_\ell \mathfrak{f}\, \mathsf{O} \equiv \mathsf{true} : \mathbb{B}.$$

When this is the first time the $\overline{\mathsf{n}}$ query is asked, computation is temporarily blocked. The way out of the *cul-de-sac* is through the *splitting rules* of $\mathfrak{f}$TT. As their name hints at, they allow us to *split computation* in two parallel worlds, extending ℓ with $(\mathsf{n}, \mathsf{true})$ on the one hand, and $(\mathsf{n}, \mathsf{false})$ on the other hand. These rules are presented in Tab 4.2.

A first intuition of the logical power of these rules can be given through a quick example: in $\mathfrak{f}$TT the sequent

$$\Gamma \vdash_{\mathsf{nil}} (\mathtt{if}\ \mathfrak{f}\, 3\ \mathtt{then}\ \mathsf{true}\ \mathtt{else}\ \mathsf{true}) \equiv \mathsf{true}$$

is derivable, which is of course not the case for a simple variable

$$\vdash^{\mathsf{MLTT}} f : \mathbb{N} \to \mathbb{B}$$

in plain MLTT.

Proof. We can split on 3, making use of the splitting rule for conversion of terms. Then we have to prove

$$\mathtt{if}\ \mathsf{true}\ \mathtt{then}\ \mathsf{true}\ \mathtt{else}\ \mathsf{true} \equiv \mathsf{true}$$

and

$$\mathtt{if}\ \mathsf{false}\ \mathtt{then}\ \mathsf{true}\ \mathtt{else}\ \mathsf{true} \equiv \mathsf{true}.$$

Both cases are direct using the conversion rules of $\mathbb{B}_{\mathbf{ind}}$. ∎

$$\frac{\mathsf{WF}\ \ell \qquad \vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathfrak{f}\, \overline{\mathsf{n}} \equiv \overline{\mathsf{b}} : \mathbb{B}}\ (\mathsf{n}, \mathsf{b}) \in \ell$$

Table 4.1.: $\mathfrak{f}$-conversion rule

As lists ℓ, ℓ' can be seen as *partial maps*, in the rest of the Section, we write $\mathsf{n} \notin \mathrm{dom}(\ell)$ to mean

$(\mathsf{n}, \mathsf{true}) \notin \mathrm{dom}(\ell) \wedge (\mathsf{n}, \mathsf{false}) \notin \mathrm{dom}(\ell)$.

$$\frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} A \quad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} A}{\Gamma \vdash_\ell A}\ \mathsf{n} \notin \mathrm{dom}(\ell)$$

$$\frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} t : A \quad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} t : A}{\Gamma \vdash_\ell t : A}\ \mathsf{n} \notin \mathrm{dom}(\ell)$$

$$\frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} A \equiv B \quad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} A \equiv B}{\Gamma \vdash_\ell A \equiv B}\ \mathsf{n} \notin \mathrm{dom}(\ell)$$

$$\frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} t \equiv u : A \quad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} t \equiv u : A}{\Gamma \vdash_\ell t \equiv u : A}\ \mathsf{n} \notin \mathrm{dom}(\ell)$$

Table 4.2.: Splitting rules of $\mathfrak{f}$TT

This is reminiscent of η-rule for booleans. However, our setting is much weaker, as the splitting rules are restricted to the particular case of $\mathfrak{f}$ applied to numerals.

Splitting heirs However, we do not accept any list as forcing condition; they need to be well-mannered and validate some properties. First, let us notice that if we allowed the user to split several times on the same natural number n, we would end up with a tree of derivations where *every branch but two* proved

$$\Gamma \vdash_\ell \mathsf{true} \equiv \mathsf{f}\,\overline{\mathsf{n}} \equiv \mathsf{false} : \mathbb{B},$$

and are thus inconsistent. The system as a whole would stay consistent, though, as the extreme-left (resp. extreme-right) branch would only contain copies of $(\mathsf{n}, \mathsf{true})$ (resp. copies of $(\mathsf{n}, \mathsf{false})$), escaping the fateful conversion. Still, for peace of mind we discard that kind of boisterous behaviour, with a well-formation WF judgment, presented in Tab 4.3.

In the rest of the chapter, we will implicitly only consider forcing conditions validating that predicate, meaning that our forcing conditions actually implement *partial functional relations* on natural numbers and booleans.

$$\frac{}{\mathsf{WF}\ \mathsf{nil}} \qquad \frac{\mathsf{WF}\ \ell \qquad \mathsf{n} \notin \mathrm{dom}(\ell)}{\mathsf{WF}\ (\mathsf{n}, \mathsf{b}) :: \ell}$$

Table 4.3.: Well-formation of ℓ

Very meta Moreover, we ask that forcing conditions only contain numerals and canonical booleans (*i.e.* true or false). To enforce this property, we do not give them access to terms of TT but to *meta-natural numbers* and *meta-booleans*. This is the reason we make a distinction between

$$\mathsf{n} \quad \text{and} \quad \mathsf{b}$$

in the meta-theory and

$$\overline{\mathsf{n}} \quad \text{and} \quad \overline{\mathsf{b}}$$

when we inject them into the object theory. This saves us from bothering with computation inside ℓ. Indeed, had we allowed every term of the object theory to creep into ℓ, we would have had to deal with terms such as

$$(\mathsf{S}\ x, \mathsf{true}) \qquad \text{or} \qquad (\mathsf{S}\ \mathsf{O}, y)$$

in ℓ. This would put us at the mercy of a treacherous substitution σ, suddenly turning x into O and y into false and making us inconsistent all over again.

Thankfully, checking whether a term $t : \mathbb{N}$ is a numeral is an easily decidable property.

As a consequence, this means that in our proof of normalization in Section 4.3, the term

$$\mathsf{f}\,(\mathsf{S}\ x)$$

will for instance be considered *neutral*, although this is not obvious when looking at its head constructor. This is a slight drift away from the usual *weak-head reduction* paradigm.

4.1.2. Undressed code

We now strip ʃTT of its its mysterious gauze and stare directly at its naked set of rules. Formally, our theory features the following judgments:

1. $\mathsf{WF}\ \ell$, meaning that ℓ is a well-formed *forcing condition*;
2. $\vdash_\ell \Gamma$, meaning that Γ is a well-typed context;
3. $\Gamma \vdash_\ell A$, meaning that A is a type under context Γ;
4. $\Gamma \vdash_\ell t : A$, meaning that t is of type A under context Γ;
5. $\Gamma \vdash_\ell A \equiv B$, meaning that A and B are convertible types under context Γ;
6. $\Gamma \vdash_\ell t \equiv u : A$, meaning that t and u are convertible terms at type A under context Γ.

The most unusual judgment is the first one, $\mathsf{WF}\ \ell$, which we already mentioned. Its definition is given in Tab 4.3. Other judgments are rather standard.

Extension is our intention We make sure that every rule of MLTT gets a corresponding one in ʃTT. Every typing judgement

$$\Gamma \vdash^{\mathsf{MLTT}} t : A$$

can thus be translated into a judgment

$$\Gamma \vdash^{\mathsf{ʃTT}}_{\mathsf{nil}} t : A.$$

The same is true for

- $\vdash^{\mathsf{MLTT}} \Gamma$
- $\Gamma \vdash^{\mathsf{MLTT}} A$
- $\Gamma \vdash^{\mathsf{MLTT}} A \equiv B$
- $\Gamma \vdash^{\mathsf{MLTT}} t \equiv u : A$

We will call ʃTT an *extension* of MLTT.

Fifty shades of rules The typing rules of our theory are displayed in Figure 4.1, its conversion rules in Figure 4.2.

In the previous Section, we presented almost every rule regarding ♭. The only remaining one is a congruence rule over natural numbers. For the sake of exhaustiveness, we present it in Tab .4.4 All these unusual rules are written in blue in Figure 4.1 and 4.2.

$$\frac{\Gamma \vdash_\ell n \equiv m : \mathbb{N}}{\Gamma \vdash_\ell ♭\, n \equiv ♭\, m : \mathbb{B}}$$

Table 4.4.: Congruence rule for ♭

$$A, B, t, u := \square \mid x \mid t\ u \mid \lambda x : A.t \mid \Pi x : A.t \mid \mathbb{N} \mid \mathrm{O} \mid \mathrm{S}\ t \mid \mathbb{N}_{\mathbf{ind}}\ P\ p_{\mathrm{O}}\ p_{\mathrm{S}}\ n \mid \mathbb{B} \mid \mathsf{true} \mid \mathsf{false} \mid \mathfrak{f}\ t \mid \mathbb{B}_{\mathbf{ind}}\ P\ p_{\mathsf{true}}\ p_{\mathsf{false}}\ b$$

$$\Gamma, \Delta := \cdot \mid \Gamma, x : A$$

$$\frac{}{\mathsf{WF}\ \mathsf{nil}} \qquad \frac{\mathsf{WF}\ \ell \qquad \mathsf{n} \in \mathbb{N} \qquad \mathsf{b} \in \mathbb{B}}{\mathsf{WF}\ (\mathsf{n}, \mathsf{b}) :: \ell}\ \mathsf{n} \notin \mathrm{dom}(\ell) \qquad \frac{\mathsf{WF}\ \ell}{\vdash_\ell \cdot} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell A}{\vdash_\ell \Gamma, x : A}$$

$$\frac{\vdash_\ell \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash_\ell x : A} \qquad \frac{\Gamma \vdash_\ell A \qquad \Gamma \vdash_\ell t : B}{\Gamma, x : A \vdash_\ell t : B} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \square} \qquad \frac{\Gamma \vdash_\ell A : \square}{\Gamma \vdash_\ell A}$$

$$\frac{\Gamma \vdash_\ell A : \square \qquad \Gamma, x : A \vdash_\ell B : \square}{\Gamma \vdash_\ell \Pi x : A.B : \square} \qquad \frac{\Gamma \vdash_\ell A \qquad \Gamma, x : A \vdash_\ell B}{\Gamma \vdash_\ell \Pi x : A.B}$$

$$\frac{\Gamma \vdash_\ell t : \Pi x : A.B \qquad \Gamma \vdash_\ell u : A}{\Gamma \vdash_\ell t\ u : B\{x := u\}} \qquad \frac{\Gamma \vdash_\ell A \qquad \Gamma, x : A \vdash_\ell t : B}{\Gamma \vdash_\ell \lambda x : A.t : \Pi x : A.B}$$

$$\frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{N} : \square} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{N}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{B} : \square} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{B}}$$

$$\frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathrm{O} : \mathbb{N}} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell n : \mathbb{N}}{\Gamma \vdash_\ell \mathrm{S}\ n : \mathbb{N}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathsf{true} : \mathbb{B}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathsf{false} : \mathbb{B}}$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{B} \vdash_\ell P \qquad \Gamma \vdash_\ell t_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma \vdash_\ell t_{\mathsf{false}} : P\{x := \mathsf{false}\} \qquad \Gamma \vdash_\ell b : \mathbb{B}\end{array}}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b : P\{x := b\}} \qquad \frac{\begin{array}{c}\Gamma, x : \mathbb{N} \vdash_\ell P \qquad \Gamma \vdash_\ell t_{\mathrm{O}} : P\{x := \mathrm{O}\} \\ \Gamma \vdash_\ell t_{\mathrm{S}} : \Pi y : \mathbb{N}.P\{x := y\} \to P\{x := \mathrm{S}\ y\} \qquad \Gamma \vdash_\ell n : \mathbb{N}\end{array}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : P\{x := n\}}$$

$$\frac{\Gamma \vdash_\ell t : A \qquad \Gamma \vdash_\ell B \qquad \Gamma \vdash_\ell A \equiv B}{\Gamma \vdash_\ell t : B} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell n : \mathbb{N}}{\Gamma \vdash_\ell \mathfrak{f}\ n : \mathbb{B}}$$

$$\mathfrak{f}\mathrm{WFT}\ \frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} A \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} A}{\Gamma \vdash_\ell A}\ \mathsf{n} \notin \mathrm{dom}(\ell) \qquad \mathfrak{f}\mathrm{TY}\ \frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} t : A \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} t : A}{\Gamma \vdash_\ell t : A}\ \mathsf{n} \notin \mathrm{dom}(\ell)$$

Figure 4.1.: Syntax and typing rules of $\mathfrak{f}\mathsf{TT}$

$$\frac{\Gamma \vdash_\ell t : A}{\Gamma \vdash_\ell t \equiv t : A} \qquad \frac{\Gamma \vdash_\ell A}{\Gamma \vdash_\ell A \equiv A} \qquad \frac{\Gamma \vdash_\ell t \equiv u : A}{\Gamma \vdash_\ell u \equiv t : A} \qquad \frac{\Gamma \vdash_\ell A \equiv B}{\Gamma \vdash_\ell B \equiv A} \qquad \frac{\Gamma \vdash_\ell A \equiv B : \square}{\Gamma \vdash_\ell A \equiv B}$$

$$\frac{\Gamma \vdash_\ell t \equiv u : A \qquad \Gamma \vdash_\ell A \equiv B}{\Gamma \vdash_\ell t \equiv u : B} \qquad \frac{\Gamma \vdash_\ell t \equiv u : A \qquad \Gamma \vdash_\ell u \equiv v : A}{\Gamma \vdash_\ell t \equiv v : A} \qquad \frac{\Gamma \vdash_\ell A \equiv B \qquad \Gamma \vdash_\ell B \equiv C}{\Gamma \vdash_\ell A \equiv C}$$

$$\frac{\Gamma \vdash_\ell A \equiv B : \square \qquad \Gamma, x : A \vdash_\ell C \equiv D : \square}{\Gamma \vdash_\ell \Pi x : A. C \equiv \Pi x : B. D : \square} \qquad \frac{\Gamma \vdash_\ell A \equiv B \qquad \Gamma, x : A \vdash_\ell C \equiv D}{\Gamma \vdash_\ell \Pi x : A. C \equiv \Pi x : B. D}$$

$$\frac{\Gamma \vdash_\ell A \qquad \Gamma, x : A \vdash_\ell t : B \qquad \Gamma \vdash_\ell u : A}{\Gamma \vdash_\ell (\lambda x : A. t)\, u \equiv t\{x := u\} : B\{x := u\}} \qquad \frac{\Gamma \vdash_\ell f \equiv g : \Pi x : A. B \qquad \Gamma \vdash_\ell u \equiv v : A}{\Gamma \vdash_\ell f\ u \equiv g\ v : B\{x := u\}}$$

$$\frac{\Gamma \vdash_\ell A \qquad \Gamma \vdash_\ell f : \Pi x : A. B \qquad \Gamma \vdash_\ell g : \Pi x : A. B \qquad \Gamma, x : A \vdash_\ell f\ x \equiv g\ x : B}{\Gamma \vdash_\ell f \equiv g : \Pi x : A. B}$$

$$\frac{\Gamma \vdash_\ell n \equiv m : \mathbb{N}}{\Gamma \vdash_\ell \mathsf{S}\, n \equiv \mathsf{S}\, m : \mathbb{N}} \qquad \frac{\begin{array}{l}\Gamma, x : \mathbb{N} \vdash_\ell P \qquad \Gamma \vdash_\ell t_\mathsf{O} : P\{x := \mathsf{O}\} \\ \Gamma \vdash_\ell t_\mathsf{S} : \Pi y : \mathbb{N}. P\{x := y\} \to P\{x := \mathsf{S}\, y\}\end{array}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{ind}}\, P\, t_\mathsf{O}\, t_\mathsf{S}\, \mathsf{O} \equiv t_\mathsf{O} : P\{x := \mathsf{O}\}}$$

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{N} \vdash_\ell P & \Gamma \vdash_\ell t_\mathsf{O} \equiv t'_\mathsf{O} : P\{x := \mathsf{O}\} \\ \Gamma \vdash_\ell t_\mathsf{S} \equiv t'_\mathsf{S} : \Pi y : \mathbb{N}. P\{x := y\} \to P\{x := \mathsf{S}\, y\} & \Gamma \vdash_\ell n \equiv n' : \mathbb{N}\end{array}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{ind}}\, P\, t_\mathsf{O}\, t_\mathsf{S}\, n \equiv \mathbb{N}_{\mathbf{ind}}\, P\, t'_\mathsf{O}\, t'_\mathsf{S}\, n' : P\{x := n\}}$$

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{N} \vdash_\ell P & \Gamma \vdash_\ell t_\mathsf{O} : P\{x := \mathsf{O}\} \\ \Gamma \vdash_\ell t_\mathsf{S} : \Pi y : \mathbb{N}. P\{x := y\} \to P\{x := \mathsf{S}\, y\} & \Gamma \vdash_\ell n : \mathbb{N}\end{array}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{ind}}\, P\, t_\mathsf{O}\, t_\mathsf{S}\, (\mathsf{S}\, n) \equiv t_\mathsf{S}\{y := n; p_y := \mathbb{N}_{\mathbf{ind}}\, P\, t_\mathsf{O}\, t_\mathsf{S}\, n\} : P\{x := \mathsf{S}\, n\}}$$

$$\frac{\begin{array}{ll}\Gamma, x : \mathbb{B} \vdash_\ell P & \Gamma \vdash_\ell t_{\mathsf{true}} \equiv t'_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma \vdash_\ell b \equiv b' : \mathbb{B} & \Gamma \vdash_\ell t_{\mathsf{false}} \equiv t'_{\mathsf{false}} : P\{x := \mathsf{false}\}\end{array}}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, b \equiv \mathbb{B}_{\mathbf{ind}}\, P\, t'_{\mathsf{true}}\, t'_{\mathsf{false}}\, b' : P\{x := b\}}$$

$$\frac{\Gamma, x : \mathbb{B} \vdash_\ell P \qquad \begin{array}{l}\Gamma \vdash_\ell t_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma \vdash_\ell t_{\mathsf{false}} : P\{x := \mathsf{false}\}\end{array}}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, \mathsf{true} \equiv t_{\mathsf{true}} : P\{x := \mathsf{true}\}} \qquad \frac{\Gamma, x : \mathbb{B} \vdash_\ell P \qquad \begin{array}{l}\Gamma \vdash_\ell t_{\mathsf{true}} : P\{x := \mathsf{true}\} \\ \Gamma \vdash_\ell t_{\mathsf{false}} : P\{x := \mathsf{false}\}\end{array}}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{ind}}\, P\, t_{\mathsf{true}}\, t_{\mathsf{false}}\, \mathsf{false} \equiv t_{\mathsf{false}} : P\{x := \mathsf{false}\}}$$

$$\frac{\Gamma \vdash_\ell n \equiv m : \mathbb{N}}{\Gamma \vdash_\ell \mathsf{f}\, n \equiv \mathsf{f}\, m : \mathbb{B}} \qquad \frac{\mathsf{WF}\ \ell \qquad \vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathsf{f}\, \overline{\mathsf{n}} \equiv \overline{\mathsf{b}} : \mathbb{B}}\ (\mathsf{n}, \mathsf{b}) \in \ell$$

$$\frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} A \equiv B \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} A \equiv B}{\Gamma \vdash_\ell A \equiv B}\ \mathsf{n} \notin \mathsf{dom}(\ell) \qquad \frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} t \equiv u : A \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} t \equiv u : A}{\Gamma \vdash_\ell t \equiv u : A}\ \mathsf{n} \notin \mathsf{dom}(\ell)$$

Figure 4.2.: Conversion rules of ɾTT

4.2. Canonizing continuity

Let us now explain how we plan to prove continuity of all

$$\vdash^{\mathsf{FTT}} f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

functionals. As continuity will be essentially deduced from normalization of the theory, we need to introduce a few concepts usually linked to the latter. We will however refrain from exposing every technical detail in this Section, and we will deliberately keep some definitions informal. Everything will be made precise in Section 4.3. Moreover, as it is our main goal, in this Section we only consider normalization at the $\mathbb{N}$ type, under the empty context.

In this Section, we implicitly assume that our proof of normalization for FTT goes through. As such, every result presented here should rather be understood as a strongly claimed conjecture.

Shrunken heads To prove normalization, we first define a reduction strategy for FTT. In our case, it is the usual weak-head reduction, extended with the rule

$$\mathsf{f}\ \overline{n} \longrightarrow_\ell \overline{b} \qquad \text{if } (n, b) \in \ell.$$

We will write $t \longrightarrow_\ell u$ for one-step reduction and $t \longrightarrow_\ell^* u$ for its reflexive, transitive closure.

In most normalization proofs, we end up with a *canonicity theorem* for $\mathbb{N}$. This means that any term

$$\cdot \vdash t : \mathbb{N}$$

can be turned into a numeral

$$\mathsf{S}^n\ \mathsf{O}$$

by recursively applying weak-head reduction. In our setting, this is not true anymore: even in the empty context, if $3 \notin \mathrm{dom}(\ell)$, the term

$$\mathit{if}\ \mathsf{f}\ 3\ \mathit{then}\ \mathsf{O}\ \mathit{else}\ \mathsf{O}$$

cannot be reduced anymore.

Numerous numerals This leads us to make a distinction between what we call *reducible* and *split-reducible* terms:

A term $t : \mathbb{N}$ is *reducible* in the empty context under ℓ when it can be recursively reduced to a numeral $\mathsf{S}^n\ \mathsf{O}$. We write

$$\cdot \Vdash^{\mathbf{s}}_\ell t \in \mathbb{N}.$$

A term $t : \mathbb{N}$ is *split-reducible* in the empty context under ℓ, written

$$\cdot \Vdash^{\mathbf{w}}_\ell t : \mathbb{N},$$

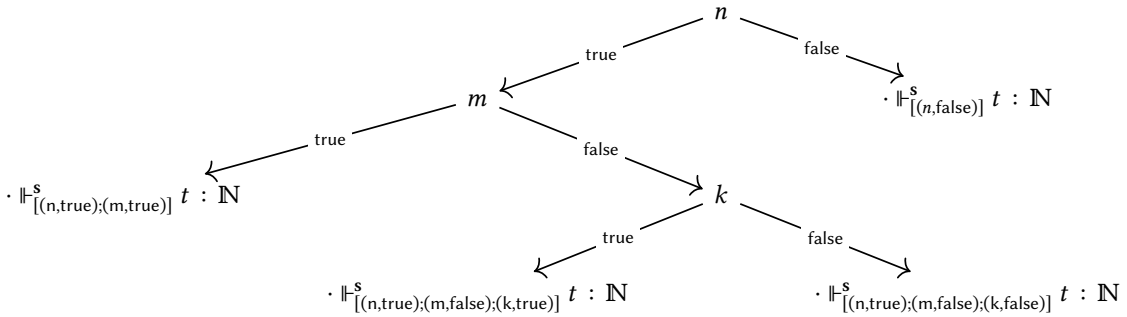
if:

- It is reducible in the empty context under ℓ, or

- There exists a split on some natural number n such that t is split-reducible in the empty context under both

$$(\mathsf{n}, \mathsf{true}) :: \ell \qquad \text{and} \qquad (\mathsf{n}, \mathsf{false}) :: \ell.$$

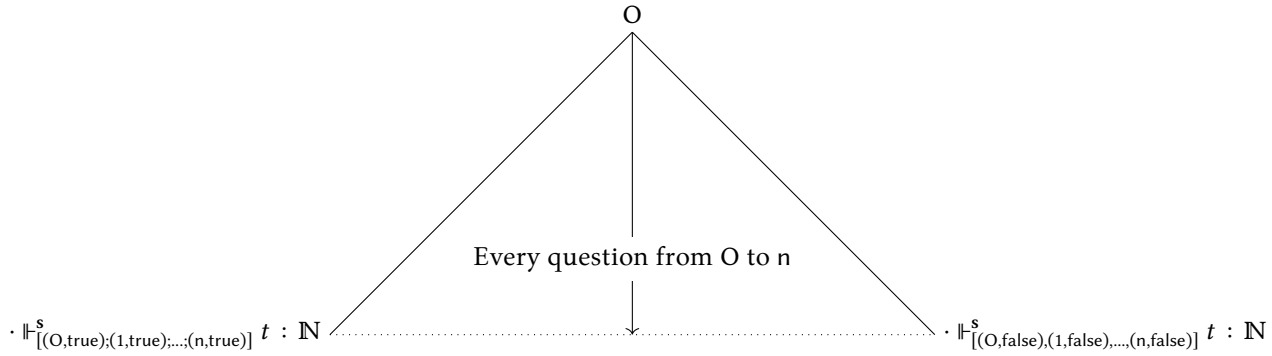
Intuitively, this means that a term t is *split-reducible* if there is a well-founded tree of splits such that t is *reducible* at every leaf. Visually, a split-reducibility proof looks like this:



The above definition is equivalent to the following one:

$$\begin{aligned} \cdot \Vdash^{\mathbf{w}}_{\ell} t : \mathbb{N} \coloneqq \Sigma\, \mathsf{n} : \mathbb{N}.\, \Pi\, \ell'.\ & (\ell' \preceq \ell) \longrightarrow \\ & (\Pi \mathsf{m} : \mathbb{N}.\, \mathsf{m} \leq \mathsf{n} \to \mathsf{m} \in \ell') \longrightarrow \\ & \cdot \Vdash^{\mathbf{s}}_{\ell} t : \mathbb{N} \end{aligned}$$

This wording is coarser than the previous definition, in the sense that it builds a complete tree of height n, giving it the shape of a cone. Visually, it looks like this:



For practical reasons, we will use the latter in Section 4.3, as it allows us to reuse as much code as possible. However, with the former definition the link with dialogue continuity is more blatant. Providing we prove that every term

$$\cdot \vdash^{\mathsf{f}\,\mathsf{TT}}_{\mathsf{nil}} t : \mathbb{N}$$

is split-reducible, we can deduce a theorem of *split-canonicity*: for every term $t : \mathbb{N}$, there is a finite tree of splits such that t can be recursively reduced to a true numeral $\mathsf{S}^n\,\mathsf{O}$ at every leaf.

The true numeral $\mathsf{S}^n\mathsf{O}$ can of course be different at every leaf, as different values for f n lead to different computations.

Fruitful trees Finally, taking a function

$$\cdot \vdash_{\mathsf{nil}} h : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N},$$

we can apply it to

$$\lambda x.\, \tilde{\mathsf{f}}\, x$$

and get a closed term of type $\mathbb{N}$. Split-canonicity applies, from which we retrieve a tree of splits. Then, still in the meta-theory we can convert this tree of splits into a dialogue tree

$$d : \mathfrak{D}\ (I \coloneqq \mathbb{N})\ (O \coloneqq \lambda\_.\ \mathbb{B})\ \mathbb{N}$$

and retrieve dialogue continuity:

$$\mathbf{\Pi}(\alpha : \mathbb{N} \to \mathbb{B}).\ h\ \alpha = \partial\ d\ \alpha.$$

Notice that as in Chapter 3, this proof is external to the theory, and we do not retrieve a fTT term of type

$$\Pi(h : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}).\, \Sigma(d : \mathfrak{D}\ \mathbb{N}\ (\lambda\_.\ \mathbb{B})\ \mathbb{N}).\, \Pi(\alpha : \mathbb{N} \to \mathbb{B}).\ h\ \alpha = \partial\ d\ \alpha.$$

Still, as fTT is an extension of MLTT, we have our theorem: every

$$\cdot \vdash^{\mathsf{MLTT}} h : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

functional is dialogue continuous.

As our goal is to prove that every *closed term* of type

$$(\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$

is continuous, we could restrict the scope of our proof and only care about normalization in the empty context. This is in fact the path followed by *Coquand and Jaber* in their paper: they define their logical relation in the empty context and take care of arbitrary context by closing under substitution of reducible terms. This way they for instance never have to worry about neutrals.

We however believe that normalization under any context is a necessary first step if we want to *internalize* the continuity theorem. Indeed, there might be hope in iterating the forcing extension, in a way alluded to by *Coquand and Jaber* at the end of their paper [43], but we would need to specify the computational content of the resulting axiom under any context, not only the empty one.

[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"

Let us leave this for future work, and for now let us focus on proving normalization for fTT.

4.3. Normalizing normalization

Our emphasis on syntactic models in the previous chapters owed a lot to the fact that it saved us from normalization proofs. Indeed, proving in dependent type theory that every well-typed term is normalizing is a notoriously difficult feat. Thankfully, now that we can no longer escape it, there are some giants' shoulders to stand on. *Abel and al* [3] present a model of normalization for MLTT with dependent products, natural numbers and one predicative universe. Their proof is a variant of the historical *Tait*'s argument [134], and is formalized in Agda, making use of induction recursion. This model has since been extended by various authors, such as *Gilbert et al* [59] with a universe of *strict propositions* or *Pujet et al [119, 120]* with observational equality. Recently, the development was ported to Coq by *Adjedj et al* [4].

[3]: Abel et al. (2017), "Decidability of Conversion for Type Theory in Type Theory"

[134]: Tait (1967), "Intensional Interpretations of Functionals of Finite Type I"

[59]: Gilbert et al. (2019), "Definitional Proof-Irrelevance without K"

[119]: Pujet et al. (2023), "Impredicative Observational Equality"

[120]: Pujet et al. (2022), "Observational Equality: Now for Good"

[4]: Adjedj et al. (2023), "Martin-Löf à la Coq"

Normalization has historically often been linked to reduction, since an easy way to define normal forms is simply to say that they are terms that cannot be further reduced. However, the paradigm over reduction has shifted a bit over time. In historic work, as in the proof of strong reduction for System T presented by *Girard* [63], reduction is a non-deterministic relation, close to our notion of *conversion*. For instance, β-reduction can be triggered on any subterm $(\lambda x.\, t)\ u$ of a term v. Consequently, normal forms of the λ-calculus, presented in Tab 4.7, are normal from head to tail: we shall call them *deep normal forms*.

To make things even blurrier, in the same paper *Girard* calls $t \longrightarrow u$ *reduction* and $t \longrightarrow^* u$ *conversion.*

$$\frac{}{\mathsf{ne}\ x} \qquad \frac{\mathsf{ne}\ f \qquad \mathsf{nf}\ u}{\mathsf{ne}\ (f\ u)}$$

$$\frac{\mathsf{nf}\ t}{\mathsf{nf}\ (\lambda x.\, t)} \qquad \frac{\mathsf{ne}\ t}{\mathsf{nf}\ t}$$

Table 4.7.: (Deep) normal forms for λ-calculus

In a proof based on weak-head reduction,normal forms are terms that do not further weak-head reduce: the *weak-head normal forms*. *Abel et al*'s normalization theorem is thus weaker than *Girard*'s, as it only entails that any well-typed term reduces to a weak-head normal form. As any subterm v of a well-typed term $t : A$ is also well-typed, we could iterate weak-head reduction to reach *iterated weak-head normal forms*. Most models show that it is possible for base types like natural numbers, where it is called *canonicity*.

[63]: Girard et al. (1989), "Proofs and Types, volume 7 of"

In that setting, it is sometimes called a *reduction strategy.*

The fact that iterated normal forms exist at all types, and that they coincide with *deep normal forms*, is not that trivial. It can however be recovered from *Abel*'s and *Adjedj et al*'s endeavours, as in both cases they prove equivalence of declarative and algorithmic conversion, the latter being a procedure that recursively weak-head normalizes the two terms it compares. Using the fact that a term is convertible to itself, one can recover existence of iterated weak-head normal forms.

Let us also mention Normalization by Evaluation (NbE) models, as in *Wieczorek et al* [143], that *directly* prove existence of deep normal forms.

[143]: Wieczorek et al. (2018), "A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory"

Still, as we care about deep normal forms only for terms of type $\mathbb{N}$ and $\mathbb{B}$, and can accomodate weak-head normal forms for the rest, we will not need to prove equivalence of declarative and algorithmic conversion; normalization will fit our needs.

4.4. Fascism in the system

Before presenting a normalization model for the whole of ɾTT, we describe a simpler version, adapted to the case of our ever-running example, System T, and its extension dubbed System ɾT. We divide this Section in two parts. First, we will prove normalization for System T using Abel's model. Then, we will adapt it to prove normalization for our extension, System ɾT. Typing rules of ɾT are presented in Figure 4.3, conversion rules in Figure 4.4. Once again, rules relevant to our extension are displayed in blue. Removing them, together with $\mathfrak{f}$ and ℓ subscripts for judgments, one recovers usual System T as presented in Section 1.1, only this time with natural numbers and booleans.

System ɾT is exactly the system described by *Coquand and Jaber* in their 2012 paper [42]. It is an extension of System T in the same way as ɾTT is an extension of MLTT.

[42]: Coquand et al. (2012), "A Computational Interpretation of Forcing in Type Theory"

The reader might notice that forcing conditions ℓ, ℓ' only have an impact on conversion rules for $\mathfrak{f}$. In the absence of computation in types, they play no role whatsoever in judgments $\Gamma \vdash A$ and $\Gamma \vdash t : A$, hence we could remove ℓ subscripts there. They will however become important in our definition of reducibility, and crucial when we deal with MLTT, thus we choose to keep them in this example.

$$
\begin{array}{lcl}
A, B & \coloneqq & \mathbb{N} \mid \mathbb{B} \mid A \to B \\
t, u & \coloneqq & x \mid t\ u \mid \lambda x : A.t \mid \mathrm{O} \mid \mathrm{S}\ t \mid \mathbb{N}_{\mathrm{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n \mid \mathrm{true} \mid \mathrm{false} \mid \mathbb{B}_{\mathrm{rec}}\ P\ t_{\mathrm{true}}\ t_{\mathrm{false}}\ b \mid \mathfrak{f}\ t \\
\Gamma, \Delta & \coloneqq & \cdot \mid \Gamma, x : A
\end{array}
$$

$$
\frac{\mathrm{WF}\ \ell}{\vdash_\ell \cdot} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell A}{\vdash_\ell \Gamma, x : A} \qquad \frac{}{\mathrm{WF\ nil}} \qquad \frac{\mathrm{WF}\ \ell \qquad \mathrm{n} \notin \mathrm{dom}(\ell)}{\mathrm{WF}\ (\mathrm{n}, \mathrm{b}) :: \ell} \qquad \frac{\vdash_\ell \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash_\ell x : A}
$$

$$
\frac{\Gamma \vdash_\ell A \qquad \Gamma \vdash_\ell B}{\Gamma \vdash_\ell A \to B} \qquad \frac{\Gamma \vdash_\ell t : A \to B \qquad \Gamma \vdash_\ell u : A}{\Gamma \vdash_\ell t\ u : B} \qquad \frac{\Gamma \vdash_\ell A \qquad \Gamma, x : A \vdash_\ell t : B}{\Gamma \vdash_\ell \lambda x : A.t : A \to B}
$$

$$
\frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{N}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathrm{O} : \mathbb{N}} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell n : \mathbb{N}}{\Gamma \vdash_\ell \mathrm{S}\ n : \mathbb{N}}
$$

$$
\frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathbb{B}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathrm{true} : \mathbb{B}} \qquad \frac{\vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathrm{false} : \mathbb{B}} \qquad \frac{\vdash_\ell \Gamma \qquad \Gamma \vdash_\ell n : \mathbb{N}}{\Gamma \vdash_\ell \mathfrak{f}\ n : \mathbb{B}}
$$

$$
\frac{\begin{array}{cc}\Gamma \vdash_\ell P & \Gamma \vdash_\ell t_{\mathrm{true}} : P \\ \Gamma \vdash_\ell t_{\mathrm{false}} : P & \Gamma \vdash_\ell b : \mathbb{B}\end{array}}{\Gamma \vdash_\ell \mathbb{B}_{\mathrm{rec}}\ P\ t_{\mathrm{true}}\ t_{\mathrm{false}}\ b : P} \qquad \frac{\begin{array}{cc}\Gamma \vdash_\ell P & \Gamma \vdash_\ell t_{\mathrm{O}} : P \\ \Gamma \vdash_\ell t_{\mathrm{S}} : \mathbb{N} \to P \to P & \Gamma \vdash_\ell n : \mathbb{N}\end{array}}{\Gamma \vdash_\ell \mathbb{N}_{\mathrm{rec}}\ P\ t_{\mathrm{O}}\ t_{\mathrm{S}}\ n : P}
$$

$$
\mathfrak{f}\mathrm{TY}\ \frac{\Gamma \vdash_{(\mathrm{n},\mathrm{true})::\ell} t : A \qquad \Gamma \vdash_{(\mathrm{n},\mathrm{false})::\ell} t : A}{\Gamma \vdash_\ell t : A}\ \mathrm{n} \notin \mathrm{dom}(\ell)
$$

Figure 4.3.: Typing rules of System ɾT

$$\frac{\Gamma \vdash_\ell t : A}{\Gamma \vdash_\ell t \equiv t : A} \qquad \frac{\Gamma \vdash_\ell t \equiv u : A}{\Gamma \vdash_\ell u \equiv t : A} \qquad \frac{\Gamma \vdash_\ell t \equiv u : A \quad \Gamma \vdash_\ell u \equiv v : A}{\Gamma \vdash_\ell t \equiv v : A} \qquad \frac{\Gamma \vdash_\ell f \equiv g : A \to B \quad \Gamma \vdash_\ell u \equiv v : A}{\Gamma \vdash_\ell f\ u \equiv g\ v : B}$$

$$\frac{\Gamma \vdash_\ell A \qquad \Gamma, x : A \vdash_\ell t : B \qquad \Gamma \vdash_\ell u : A}{\Gamma \vdash_\ell (\lambda x : A.\ t)\ u \equiv t\{x := u\} : B} \qquad \frac{\Gamma \vdash_\ell A \qquad \Gamma \vdash_\ell f : A \to B \quad \Gamma \vdash_\ell g : A \to B \qquad \Gamma, x : A \vdash_\ell f\ x \equiv g\ x : B}{\Gamma \vdash_\ell f \equiv g : A \to B}$$

$$\frac{\Gamma \vdash_\ell n \equiv m : \mathbb{N}}{\Gamma \vdash_\ell \mathsf{S}\ n \equiv \mathsf{S}\ m : \mathbb{N}} \qquad \frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_\mathsf{O} \equiv t'_\mathsf{O} : P \quad \Gamma \vdash_\ell t_\mathsf{S} \equiv t'_\mathsf{S} : \mathbb{N} \to P \to P \qquad \Gamma \vdash_\ell n \equiv n' : \mathbb{N}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{rec}}\ P\ t_\mathsf{O}\ t_\mathsf{S}\ n \equiv \mathbb{N}_{\mathbf{rec}}\ P\ t'_\mathsf{O}\ t'_\mathsf{S}\ n' : P}$$

$$\frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_\mathsf{O} : P \quad \Gamma \vdash_\ell t_\mathsf{S} : \mathbb{N} \to P \to P}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{rec}}\ P\ t_\mathsf{O}\ t_\mathsf{S}\ \mathsf{O} \equiv t_\mathsf{O} : P} \qquad \frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_\mathsf{O} : P \quad \Gamma \vdash_\ell t_\mathsf{S} : \mathbb{N} \to P \to P \qquad \Gamma \vdash_\ell n : \mathbb{N}}{\Gamma \vdash_\ell \mathbb{N}_{\mathbf{rec}}\ P\ t_\mathsf{O}\ t_\mathsf{S}\ (\mathsf{S}\ n) \equiv t_\mathsf{S}\ n\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_\mathsf{O}\ t_\mathsf{S}\ n) : P}$$

$$\frac{\Gamma \vdash_\ell n \equiv m : \mathbb{N}}{\Gamma \vdash_\ell \mathsf{f}\ n \equiv \mathsf{f}\ m : \mathbb{B}} \qquad \frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_{\mathsf{true}} \equiv t'_{\mathsf{true}} : P \quad \Gamma \vdash_\ell t_{\mathsf{false}} \equiv t'_{\mathsf{false}} : P \qquad \Gamma \vdash_\ell b \equiv b : \mathbb{B}}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}} \equiv \mathbb{B}_{\mathbf{rec}}\ P\ t'_{\mathsf{true}}\ t'_{\mathsf{false}} : P}$$

$$\frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_{\mathsf{true}} : P \quad \Gamma \vdash_\ell t_{\mathsf{false}} : P}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{true} \equiv t_{\mathsf{true}} : P} \qquad \frac{\Gamma \vdash_\ell P \qquad \Gamma \vdash_\ell t_{\mathsf{true}} : P \quad \Gamma \vdash_\ell t_{\mathsf{false}} : P}{\Gamma \vdash_\ell \mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{false} \equiv t_{\mathsf{false}} : P}$$

$$\frac{\mathsf{WF}\ \ell \qquad \vdash_\ell \Gamma}{\Gamma \vdash_\ell \mathsf{f}\ \overline{\mathsf{n}} \equiv \overline{\mathsf{b}} : \mathbb{B}}\ (\mathsf{n}, \mathsf{b}) \in \ell \qquad \frac{\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} t \equiv u : A \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} t \equiv u : A}{\Gamma \vdash_\ell t \equiv u : A}\ \mathsf{n} \notin \mathsf{dom}(\ell)$$

Figure 4.4.: Conversion rules of System $\mathsf{f}\mathsf{T}$

4.4.1. Normalizing System T

Before entering the proof, let us quickly mention two technicalities: *weakenings* and *substitutions*. They will become important later on, as some predicates will quantify over them.

Definition 4.4.1: Weakenings and substitutions
Weakenings are inductively defined as follows:

$$\frac{}{\mathtt{id}_w : \Gamma \subseteq \Gamma} \qquad \frac{\rho : \Delta \subseteq \Gamma}{\uparrow_w \rho : (\Delta, x : A) \subseteq \Gamma} \qquad \frac{\rho : \Delta \subseteq \Gamma}{\Uparrow_w \rho : (\Delta, x : A) \subseteq (\Gamma, x : A)}$$

Substitutions are then defined as follows:

$$\frac{}{\mathtt{id}_s : \Gamma \longrightarrow \Gamma} \qquad \frac{\sigma : \Gamma \longrightarrow \Delta \qquad \Gamma \vdash t : A}{(\sigma, t) : \Gamma \longrightarrow (\Delta, x : A)}$$

$$\frac{\sigma : \Gamma \longrightarrow \Delta}{\uparrow_s \sigma : (\Gamma, x : A) \longrightarrow \Delta} \qquad \frac{\sigma : \Gamma \longrightarrow \Delta}{\Uparrow_s \sigma : (\Gamma, x : A) \longrightarrow (\Delta, x : A)}$$

We will write $t[\rho]$ or $t[\sigma]$ to denote the action of some weakening ρ or some substitution σ on some term t. When we consider the action of

$$(\mathtt{id}_s, u) : \Gamma \longrightarrow (\Gamma, x : A)$$

on t, we will simply write $t\{x := u\}$.

Formalization of weakenings (under the name "lifts") and substitutions is available here.

Let us notice that, contrarily to many pen-and-paper presentations, this definition of substitutions does not depend on weakenings.

We will refrain from explaining in detail here how theses actions are defined. The interested reader may refer to the formalization to get their precise behaviour. Let us juste assume that we have well-behaved notions of weakenings and substitution that validate the following lemma:

Lemma 4.4.1: Well formedness of weakenings and substitutions
If

$$\Gamma \vdash t : A$$

and

$$\rho : \Delta \subseteq \Gamma \qquad \text{or} \qquad \sigma : \Delta \longrightarrow \Gamma$$

then

$$\Delta \vdash t[\rho] : A \qquad \text{or} \qquad \Delta \vdash t[\sigma] : A$$

Proof that well formed weakenings preserve typing can be found here. Regarding substitutions, proof can be found here.

To prove normalization for System T, we first need to specifiy our reduction strategy. As discussed, it is the usual *weak-head reduction*.

Definition 4.4.2: Reduction
Iterated reduction for System T

$$t \longrightarrow^* u$$

is the reflexive, transitive closure of one-step reduction

$$t \longrightarrow u,$$

presented in Tab 4.8. Then

$$\Gamma \vdash t :\longrightarrow^* : u : A$$

is the conjonction of the following four conditions:

1. $\Gamma \vdash t : A$
2. $\Gamma \vdash u : A$
3. $t \longrightarrow^* u$
4. $\Gamma \vdash t \equiv u : A$

$(\lambda x.\, t)\, u$	$\longrightarrow$	$t\{x := u\}$
$\mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{true}$	$\longrightarrow$	t_{true}
$\mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{false}$	$\longrightarrow$	t_{false}
$\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ \mathsf{O}$	$\longrightarrow$	t_{O}
$\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ (\mathsf{S}\ n)$	$\longrightarrow$	$t_{\mathsf{S}}\ n\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ n)$

Table 4.8.: One-step reduction

Let us now define *neutral terms*. As explained, their definition is different from the one given in Tab 4.7. In particular, they are no more mutually defined with normal forms.

Definition 4.4.3: Neutral term
A term n is *neutral* when the following holds:

$$\frac{}{\mathsf{ne}\ x} \qquad \frac{\mathsf{ne}\ f}{\mathsf{ne}\ (f\ u)} \qquad \frac{\mathsf{ne}\ n}{\mathsf{ne}\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)} \qquad \frac{\mathsf{ne}\ b}{\mathsf{ne}\ (\mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b)}$$

Substituting a variable simply by a normal form, like a λ-term, however, might trigger a β-reduction.

Neutral terms are so called because they do not fire any reduction rule, when substituted with a variable in a term. They are moreover a particular case of *normal form*.

Definition 4.4.4: Weak-head normal forms
W*eak-head normal form* are defined as the following:

$$\frac{}{\mathtt{whnf}\ \mathsf{O}} \qquad \frac{}{\mathtt{whnf}\ \mathsf{S}\ t} \qquad \frac{\mathsf{ne}\ t}{\mathtt{whnf}\ t}$$

$$\frac{}{\mathtt{whnf}\ \mathsf{true}} \qquad \frac{}{\mathtt{whnf}\ \mathsf{false}} \qquad \frac{}{\mathtt{whnf}\ (\lambda\ x.\, t)}$$

As explained, our goal is to prove that every well-typed term

$$\Gamma \vdash^{\mathsf{T}} t : A$$

reduces to some whnf t'. We will say that every such term is *normalizing*.

To that end, we define the *reducibility predicates*, starting with base types.

Definition 4.4.5: Reducibility predicates for $\mathbb{N}$ and $\mathbb{B}$
The *reducibility predicates* $\_ \Vdash \_ \in \mathbb{N}$ and $\_ \Vdash \_ \in \mathbb{B}$ for base types $\mathbb{N}$ and $\mathbb{B}$ are defined as follows:

$$\frac{\Gamma \vdash t :\longrightarrow^* : \mathsf{O} : \mathbb{N}}{\Gamma \Vdash t \in \mathbb{N}} \qquad \frac{\Gamma \vdash t :\longrightarrow^* : u : \mathbb{N} \qquad \mathsf{ne}\ u}{\Gamma \Vdash t \in \mathbb{N}}$$

$$\frac{\Gamma \vdash t :\longrightarrow^* : \mathsf{S}\ u : \mathbb{N} \qquad \Gamma \Vdash u \in \mathbb{N}}{\Gamma \Vdash t \in \mathbb{N}}$$

$$\frac{\Gamma \vdash t :\longrightarrow^* : \mathsf{true} : \mathbb{B}}{\Gamma \Vdash t \in \mathbb{B}} \qquad \frac{\Gamma \vdash t :\longrightarrow^* : \mathsf{false} : \mathbb{B}}{\Gamma \Vdash t \in \mathbb{B}}$$

$$\frac{\Gamma \vdash t \longrightarrow^* u : \mathbb{B} \qquad \mathsf{ne}\ u}{\Gamma \Vdash t \in \mathbb{B}}$$

We can already state the following lemma:

Lemma 4.4.2: Reducibility implies canonicity
If

$$\cdot \Vdash t \in \mathbb{N}$$

then t can be recursively reduced to a true numeral

$$\mathsf{S}^n\ \mathsf{O}.$$

Similarly, for $\mathbb{B}$, either

$$t \longrightarrow^* \mathsf{true} \qquad \text{or} \qquad t \longrightarrow^* \mathsf{false}.$$

Proof. Direct as there is no neutral term in the empty context. ∎

We then define reducibility at the function type, which is mainly pointwise preservation of reducibility, up to weakening.

Definition 4.4.6: Reducibility predicate for the function type
A term t is reducible at a function type $A \to B$, if the following two conditions hold:

1. That $t \longrightarrow^* t'$ where t' is either a neutral or a λ-abstraction
2. That t preserves reducibility under any weakening $\rho : \Delta \subseteq \Gamma$:

$$\mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(a : A)\,([a] : \Delta \Vdash a \in A).\,\Delta \Vdash (t[\rho]\ a) \in B$$

The first condition simply ensures that t is weakly normalizing; the universal quantification over all weakenings is needed later to prove that reducibility is stable under weakening, a property we will use to deal with variables in the fundamental lemma.

Definition 4.4.7: Reducible type
A type equipped with a reducible predicate is called a *reducible type*. Formally, the $\Gamma \Vdash \_$ judgment is inductively defined as follows:

$$\frac{\vdash \Gamma}{\Gamma \Vdash \mathbb{N}} \qquad \frac{\vdash \Gamma}{\Gamma \Vdash \mathbb{B}} \qquad \frac{\Gamma \Vdash A \qquad \Gamma \Vdash B}{\Gamma \Vdash A \to B}$$

In MLTT, with computation in types, we will have to take reduction at type level into account. For instance, if

$$\Gamma : A \longrightarrow^* \mathbb{N}$$

then A will be a reducible type. For now, in System T it is easy to show the following:

Lemma 4.4.3: Fundamental lemma for types
$\Gamma \vdash A$ entails $\Gamma \Vdash A$.

We can also already prove that reducibility effectively entails normalization.

Lemma 4.4.4: Reducibility implies normalization
Reducible terms are normalizing.

Proof. Direct by induction on the type and definition of reducibility. ∎

We then have a few useful lemmas.

Lemma 4.4.5: Weakening of reduction
If

$$\Gamma \vdash t :\longrightarrow^* : u \qquad \text{and} \qquad \rho : \Delta \subseteq \Gamma$$

then

$$\Delta \vdash t[\rho] \longrightarrow^* u[\rho].$$

Proof. By induction on the reduction path. ∎

Lemma 4.4.6: Weakening of neutrals and whnfs
If

$$\mathsf{ne}\ n$$

then

$$\mathsf{ne}\ n[\rho].$$

The same goes for `whnf` t.

Proof. By induction on the proof of neutrality of n. For `whnf`, we make use of the following facts:

$$\begin{array}{lcllcl} \mathsf{O}[\rho] & = & \mathsf{O} & (\mathsf{S}\ t)[\rho] & = & \mathsf{S}\ (t[\rho]) \\ \mathsf{true}[\rho] & = & \mathsf{true} & (\lambda x.\ t)[\rho] & = & \lambda x.\ (t[\Uparrow_w \rho]) \\ \mathsf{false}[\rho] & = & \mathsf{false} & & & \end{array}$$

Lemma 4.4.7: Monotonicity under weakenings
If

$$\Gamma \Vdash t \in A \qquad \text{and} \qquad \rho : \Delta \subseteq \Gamma$$

then

$$\Delta \Vdash t[\rho] \in A.$$

Proof. By induction on the type A:

1. If $t : \mathbb{N}$ or $t : \mathbb{B}$ then we go by induction on the reducibility proof of t. We then apply weakening of reduction and weakening of whnfs and conclude.
2. If
$$t : A \to B$$
, then weakening of reduction and weakening of whnf get us through the first condition. For the second condition, getting a weakening
$$\rho' : \Xi \subseteq \Delta$$
we specialize the reducibility condition with $\rho' \circ \rho$ and conclude.

Note that weakenings form a pre-order. Moreoever, we have that

$$t[\rho][\rho'] = t[\rho' \circ \rho].$$

Lemma 4.4.8: Reducibility of neutrals
Neutral terms are reducible.

Proof. Let $n : A$ be a neutral term. By induction on the type A:

1. If $n : \mathbb{N}$ or $n : \mathbb{B}$, we apply the neutral rule of the corresponding reducibility predicate, with the trivial reduction $n \longrightarrow^* n$;
2. If $n : A \to B$ then for any term $a : A$, $n\ a$ is neutral. We conclude by applying the induction hypothesis.

Lemma 4.4.9: Weak-head expansion
If

$$t \longrightarrow^* u \qquad \text{and} \qquad \Gamma \Vdash u \in A$$

then

$$\Gamma \Vdash t \in A.$$

Proof. Direct by induction on the type A.

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u \in B}$$

Table 4.9.: Application rule

Are reducible terms a model of System T? To prove it, we need to check that they satisfy every rule of the theory. For instance, regarding the application rule (reminded in Tab 4.9), knowing that

$$\Gamma \Vdash t \in A \to B \qquad \text{and} \qquad \Gamma \Vdash u \in A,$$

we have to prove that

$$\Gamma \Vdash t\ u \in B.$$

It is indeed the case, as reducibility at the function type was specifically taylored to manage this rule.

However, reducible terms fail at validating another rule of System T: *λ-abstraction* (reminded in Tab 4.10).

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}$$

Table 4.10.: λ-abstraction rule

Indeed, to prove that

$$\lambda x : A.t : A \to B$$

is reducible, we need to assume a term

$$a : A \qquad \text{such that} \qquad \Gamma \Vdash a \in A$$

and prove that

$$t\{x \coloneqq a\} : B$$

is reducible at type B. This is not true *a priori*, as reducible terms are not closed under substitution. The way out of the conundrum is by two-steps bruteforcing. First, we define *valid substitutions* as substitutions that only substitute reducible terms.

Definition 4.4.8: Valid substitutions
A substitution $\sigma : \Gamma \to \Delta$ is *valid*, written

$$\Vdash^v \sigma : \Gamma \to \Delta,$$

if the following holds:

$$\frac{}{\Vdash^v \mathtt{id}_s : \Gamma \longrightarrow \Gamma} \qquad \frac{\Vdash^v \sigma : \Gamma \longrightarrow \Delta}{\Vdash^v (\Uparrow_s \sigma) : (\Gamma, x : A) \longrightarrow (\Delta, x : A)}$$

$$\frac{\Vdash^v \sigma : \Gamma \to \Delta}{\Vdash^v (\uparrow_s \sigma) : (\Gamma, A) \to \Delta} \qquad \frac{\Gamma \Vdash t : A \qquad \Vdash^v \sigma : \Gamma \to \Delta}{\Vdash^v (\sigma, t) : \Gamma \to (\Delta, A)}$$

We then define *valid terms* as the closure of reducible terms under valid substitutions. Composing the two definitions, this effectively means that valid terms are the closure of reducible terms under substitution by reducible terms, exactly what we needed.

Definition 4.4.9: Valid terms
A term $t : A$ is *valid at type* A, written $\Gamma \Vdash^v t \in A$, if the following holds:

$$\Gamma \Vdash^v t \in A \coloneqq \mathbf{\Pi}(\sigma : \Gamma \to \Delta).\ \Vdash^v \sigma : \Gamma \to \Delta \longrightarrow \Delta \Vdash t[\sigma] \in A$$

This time, we get our desired result:

Theorem 4.4.10: Fundamental lemma
If $\Gamma \vdash t : A$, then $\Gamma \Vdash^v t \in A$.

Proof. We do it by induction on the typing judgment:

- The base cases of true, false : $\mathbb{B}$ and $\mathsf{O} : \mathbb{N}$ are trivial by definition of the reducibility predicates. The case of $\mathsf{S}\ t$ follows from definition of $\_ \Vdash \_ \in \mathbb{N}$ and the induction hypothesis;
- The variable rule $\Gamma, x : A \vdash x : A$ is proven by induction on the substitution σ, using some computation rules and monotonicity of reducibility under weakening;
- Application is direct by definition of the reducibility predicate for functions, along with the induction hypothesis;
- λ-abstraction follows from the definition of validity, taylored for this case;
- $\mathbb{N}_{\mathbf{rec}}$ and $\mathbb{B}_{\mathbf{rec}}$ are a bit more involved, the trickiest being $\mathbb{N}_{\mathbf{rec}}$. Let us assume

  $$\Delta \qquad \text{and} \qquad \sigma : \Gamma \to \Delta.$$

  By induction hypothesis, we have a type P, three terms t, p_{O} and p_{S} and their validity proofs

  $$\mathbf{H_t} : \Gamma \Vdash^v t \in \mathbb{N}, \quad \mathbf{H}_{\mathsf{O}} : \Gamma \Vdash^v p_{\mathsf{O}} \in P \quad \text{and} \quad \mathbf{H}_{\mathsf{S}} : \Gamma \Vdash^v p_{\mathsf{S}} \in P \to P.$$

  By applying $\mathsf{H_t}$ to σ, we get a reducibility proof $\mathbf{R_t}$ for t. We then go by induction on $\mathbf{R_t}$. We have three cases:

  1. If
     $$t \longrightarrow^* \mathsf{O}$$
     then
     $$\mathbb{N}_{\mathbf{rec}}\ t\ p_{\mathsf{O}}\ p_{\mathsf{S}} \longrightarrow^* p_{\mathsf{O}}$$
     and we conclude with $\mathbf{H}_{\mathsf{O}}$.
  2. If
     $$t \longrightarrow^* u \qquad \text{with} \qquad \mathsf{ne}\ u$$
     then
     $$\mathbb{N}_{\mathbf{rec}}\ t\ p_{\mathsf{O}}\ p_{\mathsf{S}} \longrightarrow^* \mathbb{N}_{\mathbf{rec}}\ u\ p_{\mathsf{O}}\ p_{\mathsf{S}}$$
     which is neutral. We conclude by reducibility of neutrals.
  3. Finally, if
     $$t \longrightarrow^* \mathsf{S}\ u$$
     where u is reducible, we can apply $\mathbf{H}_{\mathsf{S}}$ and conclude.

Corollary 4.4.11: Normalization
System T is normalizing.

Proof. Let $t : A$ a term of System T. By the previous theorem, t is valid. By applying the identity substitution, we get that t is reducible. Finally, as every reducible term is normalizing, t is normalizing.

4.4.2. Domain extension

It is now time to prove normalization for System $\digamma\mathsf{T}$. The interested reader can find a formalization of all the results of this Section here:
https://gitlab.inria.fr/mbaillon/Manuscript/-/tree/main/reducibility
For every definition and lemma, we will provide hyperlinks to the relevant line in the development.

Let us already highlight a difference between the pen-and-paper proof and the formalization: as we already explained, typing rules of System $\digamma\mathsf{T}$ do not depend on the forcing conditions ℓ, ℓ'; hence, in the formalization we do not mention them for typing judgments. To keep consistent notations with the following Section, we choose nonetheless to keep them on paper.

That being said, let us start our proof. We follow the same steps as previously, first defining a one-step reduction strategy $t \longrightarrow_\ell u$.

Definition 4.4.10: Reduction for System $\digamma\mathsf{T}$
For a given ℓ, one-step $\digamma$-reduction

$$t \longrightarrow_\ell u$$

is the previous one-step reduction extended with the following rules:

$$\frac{}{\mathfrak{f}\,\overline{n} \longrightarrow_\ell \overline{b}}\ (n, b) \in \ell \qquad \frac{n \longrightarrow_\ell m}{\mathfrak{f}\,n \longrightarrow_\ell \mathfrak{f}\,m} \qquad \frac{\mathfrak{f}\,n \longrightarrow_\ell \mathfrak{f}\,m}{\mathfrak{f}\,(\mathsf{S}\,n) \longrightarrow_\ell \mathfrak{f}\,(\mathsf{S}\,m)}$$

Iterated reduction for System $\digamma\mathsf{T}$

$$t \longrightarrow_\ell^* u$$

is then the reflexive, transitive closure of $t \longrightarrow_\ell u$. Finally,

$$\Gamma \vdash_\ell t : \longrightarrow^* : u : A$$

is the conjonction of the following four conditions:

1. $\Gamma \vdash_\ell t : A$
2. $\Gamma \vdash_\ell u : A$
3. $t \longrightarrow_\ell^* u$
4. $\Gamma \vdash_\ell t \equiv u : A$

One-step $\digamma$-reduction can be found here.

The first rule we add is the core of our approach: when $\mathfrak{f}$ is applied to a true numeral

$$\overline{n} \qquad \text{such that} \qquad (n, b) \in \ell, \qquad \text{then} \qquad \mathfrak{f}\,\overline{n} \longrightarrow_\ell \overline{b}.$$

However, to know when to apply this rule, we need to part from the weak-head approach: when facing

$$\mathfrak{f}\,t,$$

we recursively reduce t to check whether it is a true numeral, leading to something reminiscent of an *iterated weak-head normal form* for t.

It is not iterated weak-head normal form *per se*, though: we do not reduce subterms of neutrals, we simply reduce until we reach a neutral of a true numeral. This is what the last two rules are all about.

Despite this act of insubordination towards the grand weak-head strategy, we still retain the following:

Lemma 4.4.12:
One-step reduction for System $\mathfrak{f}\mathsf{T}$ is deterministic.

Determinism of reduction can be found here.

Proof. It is sufficient to prove that given a reduction $t \longrightarrow_\ell u$, only one rule applies. To prove this, we go by induction on the reduction structure. The only potential critical pair is when facing

$$\mathfrak{f}\,(\mathsf{S}\,n).$$

Indeed, two rules could potentially trigger a reduction there: either

$$\mathfrak{f}\,n \longrightarrow \mathfrak{f}\,m \qquad \text{hence} \qquad \mathfrak{f}\,(\mathsf{S}\,n) \longrightarrow \mathfrak{f}\,(\mathsf{S}\,m),$$

or

$$\mathsf{S}\,n \longrightarrow k \qquad \text{hence} \qquad \mathfrak{f}\,(\mathsf{S}\,n) \longrightarrow \mathfrak{f}\,k.$$

Thankfully, there is no rule to reduce $\mathsf{S}\,n$ so the second case is impossible. ∎

Changing our notion of reduction immediately gives birth to a new notion of *neutrals*.

Definition 4.4.11: Neutral terms of System $\mathfrak{f}\mathsf{T}$
Neutral terms are defined together with *natural numbers containing a neutral* as follows:

Neutral terms and weak-head normal forms are part of the same inductive, presented here.

$$\frac{}{\mathsf{ne}\ x} \qquad \frac{\mathsf{ne}\ f}{\mathsf{ne}\ (f\ u)} \qquad \frac{\mathsf{ne}\ n}{\mathsf{ne}\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)}$$

$$\frac{\mathsf{ne}\ b}{\mathsf{ne}\ (\mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b)} \qquad \frac{\mathtt{contne}\ u}{\mathsf{ne}\ (\mathfrak{f}\ u)}$$

with

$$\frac{\mathsf{ne}\ n}{\mathtt{contne}\ n} \qquad \frac{\mathtt{contne}\ n}{\mathtt{contne}\ (\mathsf{S}\ n)}$$

The fact that we deeply reduce arguments of $\mathfrak{f}$ has consequences at the level of neutrals. Indeed, once we have reduced an argument t into its simili-iterated weak-head normal form u, there are three possible cases:

- it u is of the form

$$\mathsf{S}^m\ v \qquad \text{with} \qquad v \text{ neutral}$$

At this point, we do not now that these three cases are the only reasons why computation would stop. Alternatively, it could be blocked on

$$\mathfrak{f}\ \mathsf{true}$$

or some other wrongly typed term. Ruling out these cases when

$$\mathfrak{f}\ t$$

is well-typed is the job of the logical relation.

then $\mathfrak{f}\ u$ is neutral. This is the reason behind our `contne` predicate;

- if u is a numeral

$$\overline{\mathsf{n}} \qquad \text{with} \qquad (\mathsf{n}, \mathsf{b}) \in \ell$$

then $\mathfrak{f}\ u$ will reduce to $\overline{\mathsf{b}}$;

- if u is a numeral

$$\overline{\mathsf{n}} \qquad \text{with} \qquad \mathsf{n} \notin \mathrm{dom}(\ell)$$

then computation is blocked.

In the formalization, we did not use the `contne` predicate. Rather, we defined ne in one swoop like this:

$$\frac{\mathsf{ne}\ u}{\mathsf{ne}\ (\mathfrak{f}\ (\mathsf{S}^{\mathsf{n}}\ u))}$$

where S^{n} is a way from the meta-theory to add n successors to a term (with n a meta-natural number). This phrasing is more compact, but we find the former one more telling. Both versions are equivalent.

When u is a numeral

$$\overline{\mathsf{n}} \qquad \text{with} \qquad \mathsf{n} \notin \mathrm{dom}(\ell),$$

$\mathfrak{f}\ \overline{\mathsf{n}}$ behaves like a neutral, in the sense that it does not trigger any reduction when substituted in place of a variable. We call such terms *$\mathfrak{f}$-neutrals*.

Definition 4.4.12: $\mathfrak{f}$-neutral
A term n is *$\mathfrak{f}$-neutral* when the following holds:

$$\frac{}{\mathfrak{f}\text{–ne}\ (\mathfrak{f}\ \overline{\mathsf{n}})}\ \mathsf{n} \notin \mathrm{dom}(\ell) \qquad \frac{\mathsf{ne}_{\mathfrak{f}}\ f}{\mathsf{ne}_{\mathfrak{f}}\ (f\ u)}$$

$$\frac{\mathsf{ne}_{\mathfrak{f}}\ n}{\mathsf{ne}_{\mathfrak{f}}\ (\mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)} \qquad \frac{\mathsf{ne}_{\mathfrak{f}}\ b}{\mathsf{ne}_{\mathfrak{f}}\ (\mathbb{B}_{\mathbf{rec}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b)}$$

$\mathfrak{f}$-neutrals will not appear in the rest of the proof. We define them nonetheless, for the sake of completeness.

The question of whether $\mathfrak{f}$-neutrals should be part of *normal forms* is an important choice of the formalization. We choose to *not* consider $\mathfrak{f}$-neutrals as normal. Thus, apart from neutrals, our definition of normal forms does not change.

Definition 4.4.13: Normal forms of System $\mathfrak{f}\mathsf{T}$
Normal forms are defined as follows:

$$\frac{}{\mathsf{nf}\ \mathsf{O}} \qquad \frac{}{\mathsf{nf}\ \mathsf{S}\ t} \qquad \frac{\mathsf{ne}\ t}{\mathsf{nf}\ t}$$

$$\frac{}{\mathsf{nf}\ \mathsf{true}} \qquad \frac{}{\mathsf{nf}\ \mathsf{false}} \qquad \frac{}{\mathsf{nf}\ (\lambda\ x.\ t)}$$

Let us already prove the following lemma:

Lemma 4.4.13: Monotonicity of judgments
Any judgment

$$\Gamma \vdash_\ell t : A, \qquad \Gamma \vdash_\ell t \equiv u : A \qquad \text{or} \qquad t \longrightarrow_\ell^* u$$

is still valid under any extension ℓ' of ℓ.

In the formalization, typing judgments do not depend on ℓ so monotonicity is immediate.

Proof. Direct by induction on the derivation tree. ∎

Let us also notice that this lemma is trivially true for $\mathsf{ne}\ t$, $\mathtt{contne}\ t$ and $\mathtt{nf}\ t$, as they don't mention ℓ at all.

Having updated our definitions to System fT, we can start our proof. As discussed, we now make a distinction between *reducible* and *split-reducible* terms. Reducibility at base types is usual reducibility.

Definition 4.4.14: Reducibility predicates for $\mathbb{N}$ and $\mathbb{B}$
The *reducibility predicates*

$$\_ \Vdash^{\mathbf{s}}_\ell \_ \in \mathbb{N} \qquad \text{and} \qquad \_ \Vdash^{\mathbf{s}}_\ell \_ \in \mathbb{B}$$

for $\mathbb{N}$ and $\mathbb{B}$ are defined as follows:

$$\frac{\Gamma \vdash_\ell t :\longrightarrow^* : \mathsf{O} : \mathbb{N}}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{N}} \qquad \frac{\Gamma \vdash_\ell t :\longrightarrow^* : u : \mathbb{N} \qquad \mathsf{ne}\ u}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{N}}$$

$$\frac{\Gamma \vdash_\ell t :\longrightarrow^* : \mathsf{S}\ u : \mathbb{N} \qquad \Gamma \Vdash^{\mathbf{s}}_\ell u \in \mathbb{N}}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{N}}$$

$$\frac{\Gamma \vdash_\ell t :\longrightarrow^* : \mathsf{true} : \mathbb{B}}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{B}} \qquad \frac{\Gamma \vdash_\ell t :\longrightarrow^* : \mathsf{false} : \mathbb{B}}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{B}}$$

$$\frac{\Gamma \vdash_\ell t :\longrightarrow^* : u : \mathbb{B} \qquad \mathsf{ne}\ u}{\Gamma \Vdash^{\mathbf{s}}_\ell t \in \mathbb{B}}$$

Reducibility predicates for base types can be found here.

We still have the same lemma regarding canonicity:

Lemma 4.4.14: Reducibility implies canonicity
If

$$\cdot \Vdash^{\mathbf{s}}_\ell t \in \mathbb{N},$$

then t can be recursively reduced to a true numeral $\overline{\mathsf{n}}$. Similarly, for $\mathbb{B}$, either

$$t \longrightarrow_\ell^* \mathsf{true} \qquad \text{or} \qquad t \longrightarrow_\ell^* \mathsf{false}.$$

Things however become more involved on the functional side of things, as reducibility for the function type makes use of *split-reducibility*. We thus start by defining the latter.

Definition 4.4.15: Split-reducibility
Split-reducibility under ℓ, written

$$\_ \Vdash^{\mathbf{w}}_{\ell} \_ \in \_$$

is defined as follows:

$$\frac{\Gamma \Vdash^{\mathbf{s}}_{\ell} t \in A}{\Gamma \Vdash^{\mathbf{w}}_{\ell} t \in A} \qquad \frac{\Gamma \Vdash^{\mathbf{w}}_{(n,\mathsf{true})::\ell} \; t \in A \qquad \Gamma \Vdash^{\mathbf{w}}_{(n,\mathsf{false})::\ell} \; t \in A}{\Gamma \Vdash^{\mathbf{w}}_{\ell} \; t \in A} \; n \notin \mathsf{dom}(\ell)$$

This definition is a generalized, formal version of the one we briefly discussed in Section 4.2. Intuitively, a term t is *split-reducible* if there is a *finite tree of splits* such that t is *reducible* at every leaf. This inductive definition is structurally very close the the $\mathfrak{D}$ operator we encountered in Chapter 3, which will be key to retrieve dialogue continuity at the end of the normalization proof.

As already hinted at, we however do not make use of this version of split-reducibility in the development, but of another equivalent one, which allows to reuse as many code from *Adjedj et al* as possible.

Lemma 4.4.15: Alternative definition for split-reducibility
A term $t : A$ is split-reducible under ℓ iff the following holds:

$$\Sigma\, n : \mathbb{N}.\, \mathbf{\Pi}\, \ell'.\, (\ell' \preceq \ell) \longrightarrow [n] \sqsubseteq \ell' \longrightarrow \Gamma \Vdash^{\mathbf{s}}_{\ell'} t \in A$$

where we write

$$[n] \sqsubseteq \ell$$

to mean

$$\mathbf{\Pi} m : \mathbb{N}.\, m \leq n \to m \in \ell.$$

This version of split-reducibility can be found here.

Proof. Both directions of the equivalence are quite direct.

- First direction is proven by induction on the split-reducibility proof.
  For the base case, we return O and make use of Lemma 4.4.16.
  For the splitting case, with k the number on which we split, we return
  $$\mathsf{max}(k, l, m)$$
  where l and m are the two numbers recovered by the induction hypotheses.
- For the second direction, let n be the witness of alternative split-reducibility. We simply split as many times as necessary to ensure that any
  $$m \leq n$$
  ends up in ℓ.

∎

We can now define reducibility for the function type:

Definition 4.4.16: Reducibility for the function type
For a function t to be reducible under ℓ at $A \to B$, we require the following:

- That
$$\Gamma \vdash_\ell t :\longrightarrow^* : t'$$
where t' is either a neutral or a λ-abstraction;
- That t' preserves reducibility under any weakening and any extension of ℓ. Formally:
$$\begin{aligned}&\mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\, \ell'\ (a : A).\\ &\quad (\ell' \preceq \ell) \longrightarrow (\Delta \Vdash^{\mathbf{s}}_{\ell'} a \in A) \longrightarrow \Gamma \Vdash^{\mathbf{w}}_{\ell'} (t'[\rho]\ a) \in B\end{aligned}$$

Reducibility for the function type can be found here.

As for weakenings, universal quantification over extensions of ℓ is necessary to ensure that reducible types form a presheaf family over forcing conditions.

Lemma 4.4.16: Monotonicity of reducibility
If t is reducible (resp. split-reducible) at type A under ℓ, then it is also reducible (resp. split-reducible) at type A under any extension ℓ' of ℓ.

Monotonicity of reducibility and split-reducibility under ℓ can be found here.

Proof. We prove monotonicity for reducibility and split-reducibility at the same time, by induction on the type A.

- For $\mathbb{N}$ and $\mathbb{B}$, let us assume
$$\ell' \qquad \text{such that} \qquad \ell' \preceq \ell.$$
Monotonicity of reducibility follows immediately from Lemma 4.4.13, while split-reducibility is direct by induction on the split-reducibility proof.
- For $A \to B$, the first condition of reducibility follows from Lemma 4.4.13. Second condition follows from transitivity of $\ell' \preceq \ell$ and the induction hypothesis. As before, split-reducibility is direct by induction on the split-reducibility proof.

We then have the same list of lemmas as for System T; proofs do not vary much.

Lemma 4.4.17: Reducibility implies normalization
Reducible terms are normalizing.

Lemma 4.4.18: Reducibility of neutrals
Neutral terms are reducible.

Both lemmas are proven at the same time in the formalization. They can be found here.

Lemma 4.4.19: Weak-head expansion
If $t \longrightarrow^*_\ell u$ and $\Gamma \Vdash^{\mathbf{s}}_\ell u \in A$ then $\Gamma \Vdash^{\mathbf{s}}_\ell t \in A$.

Weak-head expansion can be found here.

Note that reducible terms validate fewer rules than split-reducible ones. The obvious case is that of application: given a reducible function

$$f : A \to B$$

and a reducible argument

$$a : A,$$

we only recover a split-reducible output

$$f\ a : B.$$

However, even split-reducible terms fail to validate λ-abstraction, like reducible terms previously. We thus still need to define valid substitutions and valid terms:

Definition 4.4.17: Valid substitutions
A substitution $\sigma : \Gamma \to \Delta$ is *valid* under ℓ, written

$$\Vdash^v_\ell \sigma : \Gamma \to \Delta$$

if the following holds:

$$\frac{}{\Vdash^v_\ell \mathtt{id}_s : \Gamma \longrightarrow \Gamma} \qquad \frac{\Vdash^v_\ell \sigma : \Gamma \longrightarrow \Delta}{\Vdash^v_\ell (\Uparrow_s \sigma) : (\Gamma, x : A) \longrightarrow (\Delta, x : A)}$$

$$\frac{\Vdash^v_\ell \sigma : \Gamma \to \Delta}{\Vdash^v_\ell (\uparrow_s \sigma) : (\Gamma, A) \to \Delta} \qquad \frac{\Gamma \Vdash^s_\ell t : A \qquad \Vdash^v_\ell \sigma : \Gamma \to \Delta}{\Vdash^v_\ell (\sigma, t) : \Gamma \to (\Delta, A)}$$

Validity of substitutions can be found here.

Definition 4.4.18: Valid terms
A term $t : A$ is *valid* at type A under ℓ , written

$$\Gamma \Vdash^v_\ell t \in A$$

if the following holds:

$$\Gamma \Vdash^v_\ell t \in A \coloneqq \mathbf{\Pi}(\sigma : \Gamma \to \Delta).\, (\Vdash^v_\ell \sigma : \Gamma \to \Delta) \longrightarrow \Delta \Vdash^{\mathbb{w}}_\ell t[\sigma] \in A$$

We can then prove an updated version of the fundamental lemma:

Theorem 4.4.20: Fundamental lemma
If

$$\Gamma \vdash t : A$$

then

$$\Gamma \Vdash^v t \in A.$$

The fundamental lemma is also called *soundness*. Its proof can be found here.

Proof. We do it by induction on the typing judgment, with the same reasoning as before. The only new term is $\mathfrak{f}$; we need to prove that it is split-reducible at type $\mathbb{N} \to \mathbb{B}$.

Let us thus assume a split-reducible term

$$\Gamma \Vdash^{\mathrm{w}}_{\ell} t \in \mathbb{N}.$$

We go by induction on the split-reducibility proof of t (using definition 4.4.15):

- If t is reducible then it can be recursively reduced either to a true numeral
$$\overline{\mathsf{n}} \quad \text{or to} \quad \mathsf{S}^k\, u \text{ with } u \text{ neutral.}$$
In the latter case,
$$\mathsf{f}\,(\mathsf{S}^k\, u)$$
is also neutral, thus split-reducible. In the former case, we split on n and
$$\Gamma \vdash_{(\mathsf{n},\mathsf{true})::\ell} \mathsf{f}\,\overline{\mathsf{n}} \longrightarrow^* \mathsf{true} \qquad \text{or} \qquad \Gamma \vdash_{(\mathsf{n},\mathsf{false})::\ell} \mathsf{f}\,\overline{\mathsf{n}} \longrightarrow^* \mathsf{false}$$
in both extensions of ℓ. Thus f is split-reducible.
- If t is split-reducible in both branches of a split on some numeral n, then we simply split on n and use the induction hypothesis to conclude.

All other cases of System $\mathsf{F}\mathsf{T}$ are the same as for System T. ∎

Corollary 4.4.21: Normalization
Any term
$$\Gamma \vdash_{\ell} t : A$$
of system $\mathsf{F}\mathsf{T}$ is split-reducible.

Proof. We instantiate the fundamental lemma with the identity substitution. ∎

Corollary 4.4.22: Continuity
Any closed term
$$\cdot \vdash_{\mathsf{nil}} t : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$$
of system $\mathsf{F}\mathsf{T}$ is uniformly continuous.

Proof. Given
$$\cdot \vdash_{\mathsf{nil}} t : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N},$$
we apply
$$t \quad \text{to} \quad \mathsf{f}$$
to get a term of type $\mathbb{N}$. This term is split-reducible. The definition of split-reducibility then gives us the desired result. ∎

4.5. Everything is normal

Rules for $\mathfrak{f}$TT were displayed in the previous Section, in Figure 4.1.

It is now time to apply the same strategy to MLTT and its extension, $\mathfrak{f}$TT. In MLTT, as explained in Section 1.2, judgments are more intertwined than in System T. Especially:

- We add another judgment $\Gamma \vdash A \equiv B$, meaning that *computation exists at the level of types*;
- If
$$\Gamma \vdash t : A \qquad \text{or} \qquad \Gamma \vdash t \equiv u : A$$
and
$$\Gamma \vdash A \equiv B$$
then
$$\Gamma \vdash t : B \qquad \text{or} \qquad \Gamma \vdash t \equiv u : B.$$
We say that *judgments on terms depend on computation in types*;
- We add another type $\square$ such that if
$$\Gamma \vdash A : \square \qquad \text{or} \qquad \Gamma \vdash A \equiv B : \square$$
then
$$\Gamma \vdash A \qquad \text{or} \qquad \Gamma \vdash A \equiv B.$$
We say that *judgments on types depend on judgments on terms*.

This means that we cannot hope to prove normalization with a single reducibility predicate

$$\_ \Vdash \_ : A.$$

We need to ensure for instance that if

$$\Gamma \vdash A \equiv B$$

then A and B have compatible reducibility predicates. *Abel et al* solved the problem by defining four predicates at the same time, through induction-recursion:

- An inductive predicate $\Gamma \Vdash A$ meaning that *A is a reducible type;*
- Three predicates
$$\Gamma \Vdash A \equiv B, \qquad \Gamma \Vdash t \in A \qquad \text{and} \qquad \Gamma \Vdash t \equiv u : A$$
mutually defined by recursion on $\Gamma \Vdash A$. They respectively mean that *B is reducibly equal to A*, that *t is a reducible term at type A* and that *t and u are reducibly equal at type A.*

To avoid induction-recursion, these four predicates are defined as part of an inductive relation in *Adjedj et al*'s development. In short, they define an inductive relation LR such that

$$\mathsf{LR}\ A\ A_{\equiv}\ A_t\ A_{t\equiv}$$

means *A is a reducible type with associated predicates $A_{\equiv}$, A_t,* and *$A_{t\equiv}$*. The reasoning is however the same, and the predicates have same meaning.

4.5.1. Back to one-step

Before diving head first into the proof, let us talk about substitutions once again. In Section 4.3, as System T is rather small, we could deal with weakenings and subtitutions manually. Unfortunately, the burden grows heavier when facing MLTT. To alleviate the syntactic millstone, *Adjedj et al* make use of *Autosubst* [128] which automatically derives boilerplate lemmas on untyped weakenings and substitutions. This is an interesting aspect of mechanized proofs, that makes full use of the automation features that come with proof assistants.

The syntax of ꟻTT and lemmas generated by *Autosubst* can be found here.

[128]: Stark et al. (2019), "Autosubst 2: reasoning with multi-sorted de Bruijn terms and vector substitutions"

Technicalities aside, we can start unrolling our now well-polished strategy. We begin by devising a notion of *reduction*.

One-step reduction is exactly the same in System ꟻT and in ꟻTT.

Definition 4.5.1: Reduction for ꟻTT
Given ℓ, one-step reduction $t \longrightarrow_\ell u$ for ꟻTT is defined in Tab 4.11.

Iterated reduction for ꟻTT $t \longrightarrow_\ell^* u$ is then the reflexive, transitive closure of $t \longrightarrow_\ell u$. Finally,

$$\Gamma \vdash_\ell A :\longrightarrow^* : B \qquad (\text{resp. } \Gamma \vdash_\ell t :\longrightarrow^* : u : A)$$

is the conjonction of the following four conditions:

1. $\Gamma \vdash_\ell A$ (resp. $\Gamma \vdash_\ell t : A$)
2. $\Gamma \vdash_\ell B$ (resp. $\Gamma \vdash_\ell t : B$)
3. $A \longrightarrow_\ell^* B$ (resp. $t \longrightarrow_\ell^* u$);
4. $\Gamma \vdash_\ell A \equiv B$ (resp $\Gamma \vdash_\ell t \equiv u \in A$).

Even though our judgments are now more complex, they still validate monotonicity under weakenings and forcing conditions.

Lemma 4.5.1: Monotonicity of judgments
If

$$\rho : \Delta \subseteq \Gamma \qquad \text{and} \qquad \ell' \preceq \ell$$

then

If	t	$\longrightarrow_\ell^*$	u	then	$t[\rho]$	$\longrightarrow_{\ell'}^*$	$u[\rho]$
If	Γ	$\vdash_\ell$	A	then	Δ	$\vdash_{\ell'}$	$A[\rho]$
If	Γ	$\vdash_\ell$	$t : A$	then	Δ	$\vdash_{\ell'}$	$t[\rho] : A[\rho]$
If	Γ	$\vdash_\ell$	$A \equiv B$	then	Δ	$\vdash_{\ell'}$	$A[\rho] \equiv B[\rho]$
If	Γ	$\vdash_\ell$	$t \equiv u : A$	then	Δ	$\vdash_{\ell'}$	$t[\rho] \equiv u[\rho] : A[\rho]$

Proof. Reduction is done by simple induction. The other four judgments are proven simultaneously, through mutual induction. ∎

$$\frac{}{(\lambda x.\, t)\, u \longrightarrow t\{x := u\}}$$

$$\frac{}{\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{true} \longrightarrow t_{\mathsf{true}}}$$

$$\frac{}{\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ \mathsf{false} \longrightarrow t_{\mathsf{false}}}$$

$$\frac{}{\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ \mathsf{O} \longrightarrow t_{\mathsf{O}}}$$

$$\frac{}{\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ (\mathsf{S}\ n) \longrightarrow t_{\mathsf{S}}\ (\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{S}}\ t_{\mathsf{O}}\ n)}$$

$$\frac{n \longrightarrow_\ell m}{\mathfrak{f}\ n \longrightarrow_\ell \mathfrak{f}\ m}$$

$$\frac{\mathfrak{f}\ n \longrightarrow_\ell \mathfrak{f}\ m}{\mathfrak{f}\ (\mathsf{S}\ n) \longrightarrow_\ell \mathfrak{f}\ (\mathsf{S}\ m)}$$

$$\frac{}{\mathfrak{f}\ \overline{\mathsf{n}} \longrightarrow_\ell \overline{\mathsf{b}}}\ (\mathsf{n}, \mathsf{b}) \in \ell$$

Table 4.11.: One-step reduction for ꟻTT

Definition 4.5.2: Neutral terms of ꟻTT
Neutral terms are defined together with *natural numbers containing a neutral* as follows:

Neutral terms are exactly the same in System ꟻT and in ꟻTT

$$\frac{}{\mathsf{ne}\ x} \qquad \frac{\mathsf{ne}\ f}{\mathsf{ne}\ (f\ u)} \qquad \frac{\mathsf{ne}\ n}{\mathsf{ne}\ (\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)} \qquad \frac{\mathsf{ne}\ b}{\mathsf{ne}\ (\mathbb{B}_{\mathbf{ind}}\ P\ t_{\mathsf{true}}\ t_{\mathsf{false}}\ b)}$$

$$\frac{\mathtt{contne}\ u}{\mathsf{ne}\ (\mathfrak{f}\ u)}$$

with

$$\frac{\mathsf{ne}\ n}{\mathtt{contne}\ n} \qquad \frac{\mathtt{contne}\ n}{\mathtt{contne}\ (\mathsf{S}\ n)}$$

As types and terms are now mingled, we need to add $\mathbb{N}$ and the rest to our definition of *normal forms*.

Definition 4.5.3: Normal forms for ꟻTT
Normal forms are defined as follows:

$$\frac{}{\mathtt{nf}\ \square} \qquad \frac{}{\mathtt{nf}\ \mathbb{N}} \qquad \frac{}{\mathtt{nf}\ \mathbb{B}} \qquad \frac{}{\mathtt{nf}\ (\Pi\ A.\ B)} \qquad \frac{}{\mathtt{nf}\ \mathsf{O}}$$

$$\frac{}{\mathtt{nf}\ \mathsf{S}\ t} \qquad \frac{\mathsf{ne}\ t}{\mathtt{nf}\ t} \qquad \frac{}{\mathtt{nf}\ \mathsf{true}} \qquad \frac{}{\mathtt{nf}\ \mathsf{false}} \qquad \frac{}{\mathtt{nf}\ (\lambda\ x.\ t)}$$

Everything is in place; we can start building our normalization model.

One inductive to bind them As explained, *Adjedj et al*'s reducibility model (and thus ours) is based on an inductive relation `LR`, which is displayed in Figure 4.5.

Formal definition of the `LR` inductive can be found here.

We start by defining the type of *reducibility relations* $\mathfrak{R}$. Under ℓ and Γ, a reducibility relation binds together a term A and three predicates on terms. `LR` will be of that type.

However, to deal with universes we need to define our reducibility relation in a stratified way. In their development, *Abel et al* introduced to that effect a type `Lvl` of universe levels, to parametrize the relation. *Adjedj et al* followed their lead, and so do we.

As a consequence, `LR` will take as argument a universe level i and a proof *rec* that a reducibility relation is available at every universe level $j < i$. As the name hints at, *rec* effectively implements an induction on universe levels.

$$\texttt{Definition}\ \mathfrak{R} \coloneqq \mathbf{\Pi}\ \ell\ \Gamma\ A.\ (\texttt{term} \to \square) \to (\texttt{term} \to \square) \to (\texttt{term} \to \texttt{term} \to \square).$$

$$\begin{aligned}
&\texttt{Inductive}\ \mathsf{LR}\ \{i : \texttt{Typelvl}\}\ \{rec : \Pi\ j.\ j < i \to \mathfrak{R}\} : \mathfrak{R} \coloneqq \\
&\mid \mathsf{LR}_{\square} : \mathbf{\Pi}\ \ell\ \Gamma\ A\ (H : \Gamma \Vdash^{\square}_{\ell,i} A). \\
&\qquad \mathsf{LR}\ rec\ \ell\ \Gamma\ A\ \mathrm{H} \\
&\qquad\quad (\lambda B.\ \Gamma \Vdash^{\square}_{\ell,i} A \equiv B/\mathrm{H}) \\
&\qquad\quad (\lambda t.\ \Gamma \Vdash^{\square}_{\ell,i} t \in A/\mathrm{H}) \\
&\qquad\quad (\lambda t\ u.\ \Gamma \Vdash^{\square}_{\ell,i} t \equiv u \in A/\mathrm{H}) \\
&\mid \mathsf{LR}_{\mathbb{N}} : \mathbf{\Pi}\ \ell\ \Gamma\ A\ (\mathrm{H} : \Gamma \Vdash^{\mathbb{N}}_{\ell} A). \\
&\qquad \mathsf{LR}\ rec\ \ell\ \Gamma\ A\ \mathrm{H} \\
&\qquad\qquad (\lambda B.\ \Gamma \Vdash^{\mathbb{N}}_{\ell} A \equiv B/\mathrm{H}) \\
&\qquad\qquad (\lambda t.\ \Gamma \Vdash^{\mathbb{N}}_{\ell} t \in A/\mathrm{H}) \\
&\qquad\qquad (\lambda t\ u.\ \Gamma \Vdash^{\mathbb{N}}_{\ell} t \equiv u \in A/\mathrm{H}) \\
&\mid \mathsf{LR}_{\mathbb{B}} : \mathbf{\Pi}\ \ell\ \Gamma\ A\ (\mathrm{H} : \Gamma \Vdash^{\mathbb{B}}_{\ell} A). \\
&\qquad \mathsf{LR}\ rec\ \ell\ \Gamma\ A\ \mathrm{H} \\
&\qquad\qquad (\lambda B.\ \Gamma \Vdash^{\mathbb{B}}_{\ell} A \equiv B/\mathrm{H}) \\
&\qquad\qquad (\lambda t.\ \Gamma \Vdash^{\mathbb{B}}_{\ell} t \in A/\mathrm{H}) \\
&\qquad\qquad (\lambda t\ u.\ \Gamma \Vdash^{\mathbb{B}}_{\ell} t \equiv u \in A/\mathrm{H}) \\
&\mid \mathsf{LR}_{\mathrm{ne}} : \mathbf{\Pi}\ \ell\ \Gamma\ A\ (\mathrm{H} : \Gamma \Vdash^{\mathrm{ne}}_{\ell} A). \\
&\qquad \mathsf{LR}\ rec\ \ell\ \Gamma\ A\ \mathrm{H} \\
&\qquad\qquad (\lambda B.\ \Gamma \Vdash^{\mathrm{ne}}_{\ell} A \equiv B/\mathrm{H}) \\
&\qquad\qquad (\lambda t.\ \Gamma \Vdash^{\mathrm{ne}}_{\ell} t \in A/\mathrm{H}) \\
&\qquad\qquad (\lambda t\ u.\ \Gamma \Vdash^{\mathrm{ne}}_{\ell} t \equiv u \in A/\mathrm{H}) \\
&\mid \mathsf{LR}_{\Pi} : \mathbf{\Pi}\ \ell\ \Gamma\ A\ (\mathrm{H} : \Gamma \Vdash^{\Pi}_{\ell} A). \\
&\qquad \mathsf{LR}\ rec\ \ell\ \Gamma\ A\ \mathrm{H} \\
&\qquad\qquad (\lambda B.\ \Gamma \Vdash^{\Pi}_{\ell} A \equiv B/\mathrm{H}) \\
&\qquad\qquad (\lambda t.\ \Gamma \Vdash^{\Pi}_{\ell} t \in A/\mathrm{H}) \\
&\qquad\qquad (\lambda t\ u.\ \Gamma \Vdash^{\Pi}_{\ell} t \equiv u \in A/\mathrm{H})
\end{aligned}$$

Figure 4.5.: Inductive presentation of reducibility

In our case, we only have two distinct levels, 0 and 1, thus our universe type $\square$ is in fact $\square_0$. The Coq development is however written in such a way that it should be relatively straightforward to go to an arbitrary (albeit finite) hierarchy of universes. Getting an infinite hierarchy will probably still take time and hard work, though.

Pujet et al did add an infinite hierarchy of universes to their pen-and-paper proof, but their Agda development, which is built on top of *Abel's*, only features 3 levels: 0, 1 and ∞.

We will write

$$\Gamma \Vdash^{\mathbf{s}}_{\ell,i} A$$

to denote that there exist predicates $A_{\equiv}$, A_t, and $A_{t\equiv}$ such that

$$\mathsf{LR}\ \{i\}\ rec\ \ell\ \Gamma\ A\ A_{\equiv}\ A_t\ A_{t\equiv}.$$

Given a proof

$$H : \Gamma \Vdash^{\mathbf{s}}_{\ell,i} A,$$

we will write:

$$\begin{array}{llll}
\Gamma & \Vdash^{\mathbf{s}}_{\ell,i} & A \equiv _ & \text{to mean} \quad A_{\equiv}; \\
\Gamma & \Vdash^{\mathbf{s}}_{\ell,i} & _ : A & \text{to mean} \quad A_t; \\
\Gamma & \Vdash^{\mathbf{s}}_{\ell,i} & _ \equiv _ : A & \text{to mean} \quad A_{t\equiv}.
\end{array}$$

We now turn to defining the reducibility predicates that compose LR.

4.5.2. Back to basics

We start with the base types of ꟻTT, $\mathbb{B}$ and $\mathbb{N}$.

Booleans

Definition 4.5.4: Reducibility for $\mathbb{B}$
The

$$\_ \Vdash^{\mathbb{B}}_{\ell} \_$$

predicate is defined as follows:

$$\frac{\Gamma \vdash_{\ell} A :\longrightarrow^{*}: \mathbb{B}}{\Gamma \Vdash^{\mathbb{B}}_{\ell} A}$$

Then, given

$$\Gamma \Vdash^{\mathbb{B}}_{\ell} A,$$

the following predicates are defined:

- $\Gamma \Vdash^{\mathbb{B}}_{\ell} A \equiv \_$ simply as $\Gamma \vdash_{\ell} B :\longrightarrow: \mathbb{B}$;
- $\Gamma \Vdash^{\mathbb{B}}_{\ell} \_ \in A$ as follows:

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathsf{true} : A}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \in A} \qquad \frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathsf{false} : A}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \in A}$$

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: n : \mathbb{B} \qquad \mathsf{ne}\ n}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \in A}$$

- $\Gamma \Vdash^{\mathbb{B}}_{\ell} \_ \equiv \_ \in A$ as follows:

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathsf{true} : \mathbb{B} \quad \Gamma \vdash_{\ell} u :\longrightarrow^{*}: \mathsf{true} : \mathbb{B}}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \equiv u \in A} \qquad \frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathsf{false} : \mathbb{B} \quad \Gamma \vdash_{\ell} u :\longrightarrow^{*}: \mathsf{false} : \mathbb{B}}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \equiv u \in A}$$

$$\frac{\begin{array}{lll}\Gamma & \vdash_{\ell} & t :\longrightarrow^{*}: n : A \\ \Gamma & \vdash_{\ell} & u :\longrightarrow^{*}: m : A \\ \Gamma & \vdash_{\ell} & n \equiv m : A\end{array} \qquad \begin{array}{l}\mathsf{ne}\ n \\ \mathsf{ne}\ m\end{array}}{\Gamma \Vdash^{\mathbb{B}}_{\ell} t \equiv u \in A}$$

Reducibility for booleans can be found here.

For a type A to be reducible as type $\mathbb{B}$, it needs to reduce to $\mathbb{B}$. Similarly, when

$$\Gamma \Vdash^{\mathbb{B}}_{\ell} A,$$

for a type B to be reducibly equal to A, B itself needs to reduce to $\mathbb{B}$.

When

$$\Gamma \Vdash^{\mathbb{B}}_{\ell} A,$$

t is reducible at type A when it reduces to true, false or a neutral $n : \mathbb{B}$.

Finally, two terms t and u are reducibly equal at type A when they both reduce to true or false, or when they reduce to convertible neutrals of type $\mathbb{B}$.

Natural numbers

Definition 4.5.5: Reducibility for $\mathbb{N}$
The

$$\_ \Vdash^{\mathbb{N}}_{\ell} \_$$

predicate is defined as follows:

$$\frac{\Gamma \vdash_{\ell} A :\longrightarrow^{*}: \mathbb{N}}{\Gamma \Vdash^{\mathbb{N}}_{\ell} A}$$

Then, given

$$\Gamma \Vdash^{\mathbb{N}}_{\ell} A,$$

the following predicates are defined:

- $\Gamma \Vdash^{\mathbb{N}}_{\ell} A \equiv \_$ simply as $\Gamma \vdash_{\ell} B :\longrightarrow: \mathbb{N}$;
- $\Gamma \Vdash^{\mathbb{N}}_{\ell} \_ \in A$ as follows:

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathrm{O} : \mathbb{N}}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \in A} \qquad \frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: n : \mathbb{N} \qquad \mathsf{ne}\ n}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \in A}$$

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathrm{S}\, k : \mathbb{N} \qquad \Gamma \Vdash^{\mathbb{N}}_{\ell} t \in A}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \in A}$$

- $\Gamma \Vdash^{\mathbb{N}}_{\ell} \_ \equiv \_ : A$ as follows:

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: \mathrm{O} : \mathbb{N} \qquad \Gamma \vdash_{\ell} u :\longrightarrow^{*}: \mathrm{O} : \mathbb{N}}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \equiv u \in A} \qquad \frac{\begin{array}{lll}\Gamma & \vdash_{\ell} & t :\longrightarrow^{*}: \mathrm{S}\, k : \mathbb{N} \\ \Gamma & \vdash_{\ell} & u :\longrightarrow^{*}: \mathrm{S}\, k' : \mathbb{N} \\ \Gamma & \Vdash^{\mathbb{N}}_{\ell} & k \equiv k' : A\end{array}}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \equiv u \in A}$$

$$\frac{\begin{array}{lll}\Gamma & \vdash_{\ell} & n \equiv m : \mathbb{N} \\ \Gamma & \vdash_{\ell} & t :\longrightarrow^{*}: n : \mathbb{N} \\ \Gamma & \vdash_{\ell} & u :\longrightarrow^{*}: m : \mathbb{N}\end{array} \qquad \begin{array}{l}\mathsf{ne}\ n \\ \mathsf{ne}\ m\end{array}}{\Gamma \Vdash^{\mathbb{N}}_{\ell} t \equiv u \in A}$$

Reducibility for natural numbers can be found here.

It is pretty similar to $\mathbb{B}$: for a type A to be reducible as type $\mathbb{N}$, it needs to reduce to $\mathbb{N}$. Similarly, when

$$\Gamma \Vdash^{\mathbb{N}}_{\ell} A,$$

for a type B to be reducibly equal to A, B itself needs to reduce to $\mathbb{N}$.

When

$$\Gamma \Vdash^{\mathbb{N}}_{\ell} A,$$

a term t is reducible at type A when it reduces to O, a neutral $n : \mathbb{N}$ or S k with k reducible at type A.

Finally, two terms t and u are reducibly equal at type A when they both reduce to O, when they reduce to convertible neutrals of type $\mathbb{N}$ or when they reduce to S k and S k' with k and k' reducibly equal at type A.

Neutral types Contrarily to System T, we need to accomodate for neutral terms of type $\square$, which quickly become *neutral types*.

Definition 4.5.6: Reducibility for neutral types
The

$$\_ \Vdash^{\mathsf{ne}}_{\ell} \_$$

predicate is defined as follows:

$$\frac{\Gamma \vdash_{\ell} A :\longrightarrow^{*}: N \qquad \mathsf{ne}\ N}{\Gamma \Vdash^{\mathsf{ne}}_{\ell} A}$$

Then, given

$$\Gamma \Vdash^{\mathsf{ne}}_{\ell} A,$$

the following predicates are defined:

- $\Gamma \Vdash^{\mathsf{ne}}_{\ell} A \equiv \_$ as the following:

$$\frac{\Gamma \vdash_{\ell} B :\longrightarrow^{*}: M \qquad \mathsf{ne}\ M \qquad \Gamma \vdash_{\ell} N \equiv M}{\Gamma \Vdash^{\mathsf{ne}}_{\ell} A \equiv B}$$

- $\Gamma \Vdash^{\mathsf{ne}}_{\ell} \_ \in A$ as the following:

$$\frac{\Gamma \vdash_{\ell} t :\longrightarrow^{*}: n : N \qquad \mathsf{ne}\ n}{\Gamma \Vdash^{\mathsf{ne}}_{\ell} t \in A}$$

- $\Gamma \Vdash^{\mathsf{ne}}_{\ell} \_ \equiv \_ \in A$ as the following:

$$\frac{\begin{array}{lll} \Gamma & \vdash_{\ell} & n \equiv m : N \\ \Gamma & \vdash_{\ell} & t :\longrightarrow^{*}: n : N \\ \Gamma & \vdash_{\ell} & u :\longrightarrow^{*}: m : N \end{array} \qquad \begin{array}{l} \mathsf{ne}\ n \\ \mathsf{ne}\ m \end{array}}{\Gamma \Vdash^{\mathsf{ne}}_{\ell} t \equiv u \in A}$$

Reducibility for neutral types can be found here.

A type A is reducible as neutral when it reduces to a neutral type N. Similarly, when

$$\Gamma \Vdash^{\mathsf{ne}}_{\ell} A,$$

for a type B to be reducibly equal to A, B itself needs to reduce to a neutral M that is convertible to N.

When

$$\Gamma \Vdash^{\mathsf{ne}}_{\ell} A,$$

t is reducible at type A when it reduces to a neutral $n : N$.

Finally, two terms t and u are reducibly equal at type A when they reduce to convertible neutrals of type N.

4.5.3. Universes

Definition 4.5.7: Reducibility for the universe
The

$$\_ \Vdash^{\square}_{\ell,i} \_$$

predicate is defined as the following:

$$\frac{\Gamma \vdash_\ell A :\longrightarrow^* : \square_j \qquad j < i}{\Gamma \Vdash^{\square}_{\ell,i} A}$$

Then, given

$$\Gamma \Vdash^{\square}_{\ell,i} A,$$

the following predicates are defined:

- $\Gamma \Vdash^{\square}_{\ell,i} A \equiv B$ simply as $\Gamma \vdash_\ell B :\longrightarrow^* \square_j$;
- $\Gamma \Vdash^{\square}_{\ell,i} \_ \in A$ as the following:

$$\frac{\Gamma \vdash_\ell t :\longrightarrow^* : a : \square_j \qquad \mathsf{nf}\ n \qquad \Gamma \Vdash^{\mathbf{s}}_{\ell,j} a \qquad j < i}{\Gamma \Vdash^{\square}_{\ell,i} t \in A}$$

- $\Gamma \Vdash^{\square}_{\ell,i} \_ \equiv \_ \in A$ as the following:

$$\frac{\begin{array}{llll} \Gamma & \vdash_\ell & a \equiv b & : \square_j \\ \Gamma & \vdash_\ell & t :\longrightarrow^* : a & : \square_j \\ \Gamma & \vdash_\ell & u :\longrightarrow^* : b & : \square_j \\ \mathsf{nf} & a & & \\ \mathsf{nf} & b & & \end{array} \qquad \begin{array}{rlll} & i & < & j \\ H_a\colon & \Gamma & \Vdash^{\mathbf{s}}_{\ell,j} & a \\ H_b\colon & \Gamma & \Vdash^{\mathbf{s}}_{\ell,j} & b \\ & \Gamma & \Vdash^{\mathbf{s}}_{\ell,j} & a \equiv b \ /\ H_a \end{array}}{\Gamma \Vdash^{\square}_{\ell,i} t \equiv u \in A}$$

Reducibility for the universe type can be found here.

The requirement that $j < i$ in the rule for

$$\_ \Vdash^{\square}_{\ell,i} \_$$

could be replaced in our setting by $j := 0$ and $i := 1$. It is phrased that way to ease scalability to a system with more universes.

For a type A to be reducible as universe, it needs to reduce to $\square$. Similarly, when

$$\Gamma \Vdash^{\square}_{\ell} A,$$

for a type B to be reducibly equal to A, B itself needs to reduce to $\square$.

As one would expect, reducibility as a term of type $\square$ is essentially reducibility as a type, at a lower universe level.

The same is true for reducible equality between terms of type $\square$, which is basically reducible equality as types at a lower level. Note that this definition only makes sense because *rec* ensures that reducibility is defined at lower levels.

Abel et al use typed reduction, which enables them to prove that nothing reduces to $\square$.
They can then define

$$\_ \Vdash^{\square}_{\ell,i} \_$$

simply as being *equal* to $\square$, rather than reducing to it.
As we use untyped reduction, this is not the case in our setting.

4.5.4. Split-reducibility

A function

$$f : \Pi x : A.\, B$$

will be reducible if it sends *reducible inputs* to *split-reducible outputs*. Thus, before defining reducibility for Π-types, we need to describe *split-reducibility*. As explained, we pick the definition from Lemma 4.4.15 because it allows us to reuse code from *Adjedj et al* more freely.

Definition 4.5.8: Split-reducibility
A type A is *split-reducible under* ℓ *in context* Γ when the following holds:

$$\Gamma \Vdash^{\mathbf{w}}_{\ell,i} A \coloneqq \Sigma\, \mathsf{n} : \mathbb{N}.\, \mathbf{\Pi}\, \ell'.\, (\ell' \preceq \ell) \longrightarrow [\mathsf{n}] \sqsubseteq \ell' \longrightarrow \Gamma \Vdash^{\mathbf{s}}_{\ell'} A$$

Given $H_A : \Gamma \Vdash^{\mathbf{w}}_{\ell,i} A$, the following predicates are defined:

$$\begin{aligned}
\Gamma \Vdash^{\mathbf{w}}_{\ell,i} A \equiv B \,/\, H_A \quad &\coloneqq \quad \Sigma\, \mathsf{m} : \mathbb{N}.\, \mathbf{\Pi}\, \ell'\, (\alpha : \ell' \preceq \ell)\, (\mathsf{n}_\epsilon : [H_A.\pi_1] \sqsubseteq \ell').\\
&\qquad [\mathsf{m}] \sqsubseteq \ell' \to \Gamma \Vdash^{\mathbf{s}}_{\ell,i} A \equiv B \,/\, (H_A.\pi_2\ \alpha\ \mathsf{n}_\epsilon)\\
\Gamma \Vdash^{\mathbf{w}}_{\ell,i} t \in A \,/\, H_A \quad &\coloneqq \quad \Sigma\, \mathsf{m} : \mathbb{N}.\, \mathbf{\Pi}\, \ell'\, (\alpha : \ell' \preceq \ell)\, (\mathsf{n}_\epsilon : [H_A.\pi_1] \sqsubseteq \ell').\\
&\qquad [\mathsf{m}] \sqsubseteq \ell' \to \Gamma \Vdash^{\mathbf{s}}_{\ell,i} t \in A \,/\, (H_A.\pi_2\ \alpha\ \mathsf{n}_\epsilon)\\
\Gamma \Vdash^{\mathbf{w}}_{\ell,i} t \equiv u \in A \,/\, H_A \quad &\coloneqq \quad \Sigma\, \mathsf{m} : \mathbb{N}.\, \mathbf{\Pi}\, \ell'\, (\alpha : \ell' \preceq \ell)\, (\mathsf{n}_\epsilon : [H_A.\pi_1] \sqsubseteq \ell').\\
&\qquad [\mathsf{m}] \sqsubseteq \ell' \to \Gamma \Vdash^{\mathbf{s}}_{\ell,i} t \equiv u \in A \,/\, (H_A.\pi_2\ \alpha\ \mathsf{n}_\epsilon)
\end{aligned}$$

Formal definition of split-reducibility can be found here.

We write $A.\pi_1$ and $A.\pi_2$ to designate the first and second projection of a Σ-type.

In this alternative version, a type A is split-reducible at ℓ if there exists a uniform bound

$$\mathsf{n} : \mathbb{N}$$

such that, for any ℓ' extending ℓ, if every query between O and n has an answer in ℓ', then A is reducible at ℓ'.

A visual intuition is that we build the *complete tree of splits of height* n at ℓ (which we call the *cone* of height n at ℓ). We then ask A to be reducible at the leaves of that tree, and for any ℓ' beyond.

Reducible equality as types, reducibility as terms and reducible equality for terms all bring their own bound $\mathsf{m} : \mathbb{N}$ with them. We then simply consider the cone of height

$$\mathtt{max}(\mathsf{n}, \mathsf{m})$$

before applying the corresponding definition of the predicate.

4.5.5. Functional types

Definition 4.5.9: Reducibility for functional types
Given ℓ and i, the

$$\_ \Vdash^{\Pi}_{\ell,i} \_$$

predicate is defined through the following rules:

$$\frac{\begin{array}{lll}
\Gamma & \vdash_\ell & A :\longrightarrow^* : \Pi x : F.\, G \\
\Gamma & \vdash_\ell & F \\
\Gamma, x : F & \vdash_\ell & G \\
N_F : & \mathbb{N} & \\
H_F : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma).\, (\ell' \preceq \ell) \to ([N_F] \sqsubseteq \ell') \to & \Delta \Vdash^{\mathbf{s}}_{\ell,i} F[\rho] \\
H_G : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\, (\alpha : \ell' \preceq \ell)\, (n_\epsilon : [N_F] \sqsubseteq \ell'). & \\
 & \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \to & \Delta \Vdash^{\mathbf{w}}_{\ell,i} G[a,\rho] \\
H_{G\equiv} : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\, (\alpha : \ell' \preceq \ell)\, (n_\epsilon : [N_F] \sqsubseteq \ell'). & \\
 & \mathbf{\Pi}(H_a : \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon)). & \\
 & \Delta \Vdash^{\mathbf{s}}_{\ell,i} b \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \to & \\
 & \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \equiv b \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \to & \\
 & \Delta \Vdash^{\mathbf{w}}_{\ell,i} G[a,\rho] \equiv G[b,\rho] \,/\, (H_G\ \rho\ \alpha\ n_\epsilon\ H_a) &
\end{array}}{\Gamma \Vdash^{\Pi}_{\ell,i} A}$$

Reducibility for functional types can be found here.

Let us go over the premises one by one:

- We first ask A to reduce to a well-formed Π-type

$$\Pi x : F.\, B.$$

- We make sure that any weakening

$$F[\rho]$$

of F is split-reducible. For readability, we divide this split-reducibility proof into a natural number N_F and a proof H_F.

- Given

$$\rho : \Delta \subseteq \Gamma \qquad \text{and} \qquad \ell' \preceq \ell$$

such that $F[\rho]$ is reducible under ℓ', given any reducible term

$$\Delta \Vdash^{s}_{\ell,i} a \in F[\rho],$$

we ask

$$G[a,\rho]$$

to be split-reducible under ℓ'.

- Finally, we also require that given two reducibly equal terms

$$a, b : F[\rho],$$

applying G to both leads to split-reducibly equal types

$$G[a,\rho] \qquad \text{and} \qquad G[b,\rho].$$

Definition 4.5.10: Derived predicates for functional types
Given

$$H_A : \Gamma \Vdash^{\Pi}_{\ell,i} A,$$

the following predicates are defined:

- $\Gamma \Vdash^{\Pi}_{\ell,i} A \equiv \_ \,/\, H_A$ as the following:

$$\frac{\begin{array}{ll}\Gamma & \vdash_\ell \; B :\longrightarrow^* : \Pi x : F'.G' \\ \Gamma & \vdash_\ell \; \Pi x : F.G \equiv \Pi x : F'.G' \\ N'_F : & \mathbb{N} \\ H'_F : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(\alpha : \ell' \preceq \ell)\,(n_\epsilon : [N_F] \sqsubseteq \ell'). \\ & \quad [N'_F] \sqsubseteq \ell' \to \Delta \Vdash^{\mathbf{s}}_{\ell,i} F[\rho] \equiv F'[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \\ H'_G : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(\alpha : \ell' \preceq \ell)\,(n_\epsilon : [N_F] \sqsubseteq \ell'). \\ & \quad \mathbf{\Pi}(n'_\epsilon : [N'_F] \sqsubseteq \ell')\,(H_a : \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon)). \\ & \quad \Delta \Vdash^{\mathbf{w}}_{\ell,i} G[a,\rho] \equiv G'[a,\rho] \,/\, (H_G\ \rho\ \alpha\ n_\epsilon\ H_a)\end{array}}{\Gamma \Vdash^{\Pi}_{\ell,i} A \equiv B \,/\, H_A}$$

Reducible equality for functional types can be found here.

- $\Gamma \Vdash^{\Pi}_{\ell,i} \_ \in A \,/\, H_A$ as the following:

$$\frac{\begin{array}{ll}\Gamma & \vdash_\ell t :\longrightarrow^* : f : \Pi x : F.G \\ \mathtt{nf} & f \\ N_t : & \mathbb{N} \\ H_t : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(\alpha : \ell' \preceq \ell)\,(n_\epsilon : [N_F] \sqsubseteq \ell'). \\ & \quad \mathbf{\Pi}(n'_\epsilon : [N_t] \sqsubseteq \ell')\,(H_a : \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon)). \\ & \quad \Delta \Vdash^{\mathbf{w}}_{\ell,i} f[\rho]\ a \in G[a,\rho] \,/\, (H_G\ \rho\ \alpha\ n_\epsilon\ H_a) \\ H_{t\equiv} : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(\alpha : \ell' \preceq \ell)\,(n_\epsilon : [N_F] \sqsubseteq \ell'). \\ & \quad \mathbf{\Pi}(n'_\epsilon : [N_t] \sqsubseteq \ell')\,(H_a : \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon)). \\ & \quad \Delta \Vdash^{\mathbf{s}}_{\ell,i} b \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \to \\ & \quad \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \equiv b \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon) \to \\ & \quad \Delta \Vdash^{\mathbf{w}}_{\ell,i} f[\rho]\ a \equiv f[\rho]\ b \in G[a,\rho] \,/\, (H_G\ \rho\ \alpha\ n_\epsilon\ H_a)\end{array}}{\Gamma \Vdash^{\Pi}_{\ell,i} t \in A \,/\, H_A}$$

Reducibility for terms of functional types can be found here.

- $\Gamma \Vdash^{\Pi}_{\ell,i} \_ \equiv \_ \in A \,/\, H_A$ as the following:

$$\frac{\begin{array}{ll}\Gamma & \Vdash^{\Pi}_{\ell,i} t \in A/H_A \\ \Gamma & \Vdash^{\Pi}_{\ell,i} u \in A/H_A \\ N_\equiv : & \mathbb{N} \\ H_\equiv : & \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma)\,(\alpha : \ell' \preceq \ell)\,(n_\epsilon : [N_F] \sqsubseteq \ell'). \\ & \quad \mathbf{\Pi}(n'_\epsilon : [N_\equiv] \sqsubseteq \ell')\,(H_a : \Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho] \,/\, (H_F\ \rho\ \alpha\ n_\epsilon)). \\ & \quad \Delta \Vdash^{\mathbf{w}}_{\ell,i} f_t[\rho]\ a \equiv f_u[\rho]\ a \in G[a,\rho] \,/\, (H_G\ \rho\ \alpha\ n_\epsilon\ H_a)\end{array}}{\Gamma \Vdash^{\Pi}_{\ell,i} t \equiv u \in A \,/\, H_A}$$

Reducible equality for terms of functional types can be found here.

where f_t and f_u designate the normal forms t and u reduce to, as described in their respective proof of

$$\Gamma \Vdash^{\Pi}_{\ell,i} \_ \in A \,/\, H_A.$$

Here,

$$\Gamma \Vdash^{\mathbf{s}}_{\ell,i} A \equiv B$$

states that B reduces to

$$\Pi x : F'.G'$$

which should be convertible to

$$\Pi x : F.G.$$

Moreoever, for any weakening

$$\rho : \Delta \subseteq \Gamma, \qquad F'[\rho]$$

should be split-reducibly equal to $F[\rho]$ (once again, we separate this split-reducibility proof into a natural number N'_F and a proof H'_F).

Similarly, for any

$$\ell' \preceq \ell$$

such that $F[\rho]$ is reducible under ℓ', given any reducible term

$$\Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho],$$

we ask

$$G'[a, \rho]$$

to be split-reducibly equal to $G[a, \rho]$ under ℓ'.

The predicate $\Gamma \Vdash^{\mathbf{s}}_{\ell,i} t \in A$ states that t reduces to a normal form f. Then, similarly to G, given any

$$\rho : \Delta \subseteq \Gamma \qquad \text{and any} \qquad \ell' \preceq \ell$$

such that $F[\rho]$ is reducible under ℓ', given any reducible term

$$\Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho],$$

$f[\rho]\ a$ should be split-reducible at $G[a, \rho]$. Moreover, it should send reducibly equal inputs

$$a \equiv b : F[\rho]$$

to split-reducibly equal outputs

$$f[\rho]\ a \qquad \text{and} \qquad f[\rho]\ b.$$

Finally, $\Gamma \Vdash^{\mathbf{s}}_{\ell,i} t \equiv u \in A$ first states that t and u are split-reducible at type A. Then, given any

$$\rho : \Delta \subseteq \Gamma \qquad \text{and any} \qquad \ell' \preceq \ell$$

such that $F[\rho]$ is reducible under ℓ', given any reducible term

$$\Delta \Vdash^{\mathbf{s}}_{\ell,i} a \in F[\rho],$$

we require a split-reducible equality between the normal forms of t and u, once applied to a.

4.5.6. Lemmas about reducibility

All the following lemmas go by mutual induction on the reducibility judgments.

Lemma 4.5.2: Escape
We have:

If	$\Gamma \Vdash^{\mathbf{w}}_{\ell,i} A$	then	$\Gamma \vdash_\ell A$
If	$\Gamma \Vdash^{\mathbf{w}}_{\ell,i} t \in A$	then	$\Gamma \vdash_\ell t \in A$
If	$\Gamma \Vdash^{\mathbf{w}}_{\ell,i} A \equiv B$	then	$\Gamma \vdash_\ell A \equiv B$
If	$\Gamma \Vdash^{\mathbf{w}}_{\ell,i} t \equiv u \in A$	then	$\Gamma \vdash_\ell t \equiv u : A$

Escape is what we called *completeness* in Section 1.5. Its formal proof can be found here.

As reducibility immediately entails split-reducibility, we also have escape for reducible terms.

Lemma 4.5.3: Monotonicity of reducibility
If t is reducible (resp. split-reducible) at type A under ℓ, then it is also reducible (resp. split-reducible) at type A under any extension ℓ' of ℓ.

Formal proof of monotonicity can be found here.

Lemma 4.5.4: Weak-head expansion

1. If
$$A \longrightarrow^*_\ell B \qquad \text{and} \qquad \Gamma \Vdash^{\mathbf{s}}_\ell B$$
then there exists
$$H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A$$
such that
$$\Gamma \Vdash^{\mathbf{s}}_\ell A \equiv B \,/\, H_A$$

2. Given
$$H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A,$$
if
$$t \longrightarrow^*_\ell u \qquad \text{and} \qquad \Gamma \Vdash^{\mathbf{s}}_\ell u \in A \,/\, H_A$$
then
$$\Gamma \Vdash^{\mathbf{s}}_\ell t \in A \,/\, H_A \qquad \text{and} \qquad \Gamma \Vdash^{\mathbf{s}}_\ell t \equiv u \in A \,/\, H_A.$$

Formal proof of weak-head expansion for types can be found here.

Formal proof of weak-head expansion for terms can be found here.

Lemma 4.5.5: Reflexivity

1. If $H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A$ then $\Gamma \Vdash^{\mathbf{s}}_\ell A \equiv A \,/\, H_A$;
2. If
$$H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A \qquad \text{and} \qquad \Gamma \Vdash^{\mathbf{s}}_\ell t \in A \,/\, H_A$$
then
$$\Gamma \Vdash^{\mathbf{s}}_\ell t \equiv t \in A \,/\, H_A.$$

The same two properties are also true for split-reducibility.

Formal proof of reflexivity for types can be found here.

Formal proof of reflexivity for terms can be found here.

To prove some of the next properties of reducibility, we need to recurse on two or more reducibility proofs. However, many cases will simply be discarded by inversion. For instance, if we have

$$H_A : \Gamma \Vdash^{\mathbb{N}}_{\ell} A \qquad \text{and} \qquad \Gamma \Vdash^{\mathbf{s}}_{\ell} A \equiv B \,/\, (\mathtt{LR}_{\mathbb{N}}\ H_A)$$

then any proof

$$\Gamma \Vdash^{\mathbf{s}}_{\ell} B$$

will be of the form

$$\mathtt{LR}_{\mathbb{N}}\ H_B \qquad \text{with} \qquad H_B : \Gamma \Vdash^{\mathbb{N}}_{\ell} B$$

and any other case would be absurd.

To enforce this in a systematic way, *Abel et al* define an inductive predicate

$$\mathtt{ShapeView} : \mathbf{\Pi}\{A\ B\}\ (H_A : \Gamma \Vdash^{\mathbf{s}}_{\ell} A)\ (H_B : \Gamma \Vdash^{\mathbf{s}}_{\ell} B).\ \square$$

binding together reducibility proofs when they are of the same kind. For instance, there is a constructor

$$\mathtt{Shp}_{\mathbb{N}} : \mathbf{\Pi}(H_A : \Gamma \Vdash^{\mathbb{N}}_{\ell} A)\ (H_B : \Gamma \Vdash^{\mathbb{N}}_{\ell} B).\ \mathtt{ShapeView}\ (\mathtt{LR}_{\mathbb{N}}\ H_A)\ (\mathtt{LR}_{\mathbb{N}}\ H_B)$$

for reducibility proof for $\mathbb{N}$. It goes similarly for $\mathbb{B}$, Π-types, etc.In the Coq development from *Adjedj et al*, however, this work is done by pattern-matching. Following their lead, we define a `ShapeView` function as follows:

Definition of ShapeView by *Abel et al* can be found here.

This definition is directly taken from the Coq development here.

```
Definition ShapeView : Π{A B} (H_A : Γ ⊩ˢ_ℓ A) (H_B : Γ ⊩ˢ_ℓ B). □ :=
  match H_A, H_B with
  | LR□   _ , LR□   _  ⇒ ⊤
  | LRℕ   _ , LRℕ   _, ⇒ ⊤
  | LR𝔹   _ , LR𝔹   _, ⇒ ⊤
  | LRne  _ , LRne  _, ⇒ ⊤
  | LRΠ   _ , LRΠ   _, ⇒ ⊤
  | _       , _        ⇒ ⊥
  end.
```

Lemma 4.5.6: Reducible equality implies ShapeView
If

$$H_A : \Gamma \Vdash^{\mathbf{s}}_{\ell} A, \quad H_B : \Gamma \Vdash^{\mathbf{s}}_{\ell} B \quad \text{and} \quad \Gamma \Vdash^{\mathbf{s}}_{\ell} A \equiv B \,/\, H_A$$

then

$$\mathtt{ShapeView}\ H_A\ H_B.$$

Proof that reducible equality implies ShapeView can be found here.

In the following proofs, `ShapeView` allows us to focus on the diagonal cases and not bother with absurd ones. In Agda, *Abel et al* and *Pujet et al* had to write an elimination of $\bot$ for every non-diagonal case; in Coq however every one of those is taken care of by the `solve` tactic, which makes the proofs more readable and scalable.

Lemma 4.5.7: Irrelevance
Given two proofs

$$H_A, H'_A : \Gamma \Vdash^{\mathbf{s}}_{\ell} A,$$

the following holds:

If $\Gamma \Vdash^{\mathbf{s}}_{\ell} A \equiv B \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} A \equiv B \,/\, H'_A$
If $\Gamma \Vdash^{\mathbf{s}}_{\ell} t \in A \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} t \in A \,/\, H'_A$
If $\Gamma \Vdash^{\mathbf{s}}_{\ell} t \equiv u \in A \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} t \equiv u \in A \,/\, H'_A$

The main irrelevance theorem can be found here.

Proof. We go by induction on H_A, then do each case by induction on H'_A, using `ShapeView` to take care of the non-diagonal cases. The diagonal cases are pretty straightforward. ∎

Irrelevance is necessary to prove *Monotonicity under weakenings* and *Transitivity* of the reducibility relation.

Lemma 4.5.8: Weakening
If

$$\Gamma \Vdash^{\mathbf{s}}_{\ell} A \qquad \text{and} \qquad \rho : \Delta \subseteq \Gamma$$

then

$$\Delta \Vdash^{\mathbf{s}}_{\ell} A[\rho].$$

Then, given

$$H_A : \Gamma \Vdash^{\mathbf{s}}_{\ell} A \qquad \text{and} \qquad H^{\rho}_A : \Delta \Vdash^{\mathbf{s}}_{\ell} A[\rho],$$

the following holds:

If $\Gamma \Vdash^{\mathbf{s}}_{\ell} A \equiv B \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} A[\rho] \equiv B[\rho] \,/\, H^{\rho}_A$
If $\Gamma \Vdash^{\mathbf{s}}_{\ell} t \in A \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} t[\rho] \in A[\rho] \,/\, H^{\rho}_A$
If $\Gamma \Vdash^{\mathbf{s}}_{\ell} t \equiv u \in A \,/\, H_A$	then	$\Gamma \Vdash^{\mathbf{s}}_{\ell} t[\rho] \equiv u[\rho] \in A[\rho] \,/\, H^{\rho}_A$

Weakening of the logical relation for types can be found here.

Proof of monotonicity under weakenings for reducible equality on types can be found here. Regarding terms, it can be found here.

Let us give an example where irrelevance comes in handy when proving monotonicity of the logical relation under weakening. Given a proof

$$\Gamma \Vdash^{\Pi}_{\ell,i} A$$

we need to build a proof of

$$\Gamma \Vdash^{\Pi}_{\ell,i} A[\rho].$$

In particular, knowing

$$H_F : \mathbf{\Pi}(\rho : \Delta \subseteq \Gamma).\, (\ell' \preceq \ell) \to ([N_F] \sqsubseteq \ell') \to \Delta \Vdash^{\mathbf{s}}_{\ell,i} F[\rho]$$

we need to prove

$$H^{\rho}_F : \mathbf{\Pi}(\rho' : \Delta \subseteq \Gamma).\, (\ell' \preceq \ell) \to ([N_F] \sqsubseteq \ell') \to \Delta \Vdash^{\mathbf{s}}_{\ell,i} F[\rho][\rho'].$$

Of course, we know that

$$A[\rho][\rho'] = A[\rho' \circ \rho]$$

and we could rewrite along this equality, then conclude instantiating

$$H_F \qquad \text{with} \qquad \rho' \circ \rho.$$

However, we would then need to take care of this rewriting in every subsequent proof making use of H_F^ρ. Moreover, proofs regarding G would need their own rewriting as well, and everything would become quite painful. Here, we can simply evade these bothersome thoughts and call the (judiciously named) `irrelevance` tactic.

Lemma 4.5.9: Transitivity
Given

$$H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A, \quad H_B : \Gamma \Vdash^{\mathbf{s}}_\ell B \quad \text{and} \quad H_C : \Gamma \Vdash^{\mathbf{s}}_\ell C$$

such that

$$\Gamma \Vdash^{\mathbf{s}}_\ell A \equiv B \,/\, H_A, \quad \text{and} \quad \Gamma \Vdash^{\mathbf{s}}_\ell B \equiv C \,/\, H_B,$$

then

$$\Gamma \Vdash^{\mathbf{s}}_\ell A \equiv C \,/\, H_A.$$

Similarly, if we have

$$H_A : \Gamma \Vdash^{\mathbf{s}}_\ell A$$

such that

$$\Gamma \Vdash^{\mathbf{s}}_\ell t \equiv u \in A \,/\, H_A \quad \text{and} \quad \Gamma \Vdash^{\mathbf{s}}_\ell u \equiv v \in A \,/\, H_A,$$

then

$$\Gamma \Vdash^{\mathbf{s}}_\ell t \equiv v \in A \,/\, H_A.$$

The same is true for split-reducibility.

Transitivity for reducible equality on types can be found here.

Transitivity for reducible equality on terms can be found here.

At the time of writing, this is where the formalization stops. Following the proof structure from *Adjedj et al* [4] as presented in Section 4.4, what is left is to define validity and show that it is indeed a model of ɾTT. The fact that the Coq development of *Adjedj et al* offers a complete proof of normalization for MLTT, together with the fact that *Coquand and Mannaa* [44] derived a pen-and-paper proof of normalization for ɾTT and that we successfully formalized a proof for System ɾT, provide hints that our endeavour should succeed. We thus hope to be able to complete this formalization in the coming months. For the time being, this is left as future work.

[4]: Adjedj et al. (2023), "Martin-Löf à la Coq"

[44]: Coquand et al. (2017), "The Independence of Markov's Principle in Type Theory"

Conclusion

In this thesis, we highlighted a particular operator, the dialogue monad, and provided two attempts at bringing together dependent type theory and continuity encoded with dialogue trees.

Chapter 2 surveys the different notions of continuity, and assesses their respective strength. In particular, we show that some logical principles such as function extensionality or bar induction make some definitions equivalent, but our analysis is incomplete as we have not studied in detail the return direction, *i.e.*, what logical principle can be derived by postulating that two definitions are equivalent. There is thus room for further investigation there.

Chapter 3 shows how far we can go in the realm of program translations and provides a purely syntactic proof that functionals of *Baclofen Type Theory* are continuous (albeit from the target theory only, internalization stays out of reach). Not only is the argument syntactic, it is also expressed as a program translation into another dependent type theory. Thus, everything computes by construction and conversion in the source is interpreted as conversion in the target. Despite being a generalization of a simpler proof by Escardó, the dependently-typed presentation gives more insight about the constraints one has to respect for it to work properly, and highlights a few hidden flaws of the original version. Finally, the model gives empirical foothold to the claim that BTT is a natural setting for dependently-typed effects. We believe it is not merely an ad-hoc set of rules, but a system that keeps appearing in various contexts, and thus a generic effectful type theory. It would hence be meaningful to make an in-depth study of BTT. For instance, the problem of building a model of CIC in BTT is, to our knowledge, still open. It is also unclear whether we can provide models of BTT for any effect, as some prove challenging, like the global state effect. We might need more restrictions on dependent elimination for it to work. Finally, that we did *not* manage to build a program translation for CIC ought to be highlighted, too, as it provides insight on what can, and cannot be done, through program translations.

Chapter 4 presents ꟻTT, an extension of MLTT, together with an incomplete proof of normalization. From this proof, we should be able to recover the fact that any MLTT-definable functional is continuous. That this proof is not fully formalized yet offers immediate future work materials. In the longer run, we hope to modify ꟻTT in order to internalize the proof of continuity, as was already suggested by *Coquand and Jaber* [43]. Since *Coquand and Mannaa*'s [44] proof of independence of Markov's principle makes use of the same theory, we should be able to provide a formalization of this result too. More globally, we think that the existence of modular, formalized normalization proofs such as *Abel et al*'s [3] and *Adjedj et al*'s [4] paves the way for many type theory experiments in the coming years. Some fruits have already been plucked [59, 119, 120], but we believe there are many to come, such as η-rule for booleans (which is quite close to our ꟻTT setting), explicit subtyping, or even internal Church thesis in MLTT.

[43]: Coquand et al. (2010), "A Note on Forcing and Type Theory"

[44]: Coquand et al. (2017), "The Independence of Markov's Principle in Type Theory"

[3]: Abel et al. (2017), "Decidability of Conversion for Type Theory in Type Theory"

[4]: Adjedj et al. (2023), "Martin-Löf à la Coq"

[59]: Gilbert et al. (2019), "Definitional Proof-Irrelevance without K"

[119]: Pujet et al. (2023), "Impredicative Observational Equality"

[120]: Pujet et al. (2022), "Observational Equality: Now for Good"

Bibliography

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: Constructing strictly positive types". In: *Theoretical Computer Science* 342.1 (2005). Applied Semantics: Selected Topics, pp. 3–27. DOI: https://doi.org/10.1016/j.tcs.2005.06.002 (cit. on p. 68).

[2] Andreas Abel, Klaus Aehlig, and Peter Dybjer. "Normalization by Evaluation for Martin-Löf Type Theory with One Universe". In: *Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS 2007, New Orleans, LA, USA, April 11-14, 2007*. Ed. by Marcelo Fiore. Vol. 173. Electronic Notes in Theoretical Computer Science. Elsevier, 2007, pp. 17–39. DOI: 10.1016/j.entcs.2007.02.025 (cit. on p. 31).

[3] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. "Decidability of Conversion for Type Theory in Type Theory". In: *Proc. ACM Program. Lang.* 2.POPL (2017). DOI: 10.1145/3158111 (cit. on pp. 11, 53, 137, 146, 180).

[4] Arthur Adjedj et al. "Martin-Löf à la Coq". working paper or preprint. Sept. 2023 (cit. on pp. 11, 137, 146, 179, 180).

[5] Danel Ahman. "Handling fibred algebraic effects". In: *Proc. ACM Program. Lang.* 2.POPL (2018), 7:1–7:29. DOI: 10.1145/3158095 (cit. on p. 67).

[6] Stuart F. Allen et al. "The Nuprl Open Logical Environment". In: *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings*. Ed. by David A. McAllester. Vol. 1831. Lecture Notes in Computer Science. Springer, 2000, pp. 170–176. DOI: 10.1007/10721959_12 (cit. on p. 43).

[7] Thorsten Altenkirch and Ambrus Kaposi. "Type theory in type theory using quotient inductive types". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by Rastislav Bodík and Rupak Majumdar. ACM, 2016, pp. 18–29. DOI: 10.1145/2837614.2837638 (cit. on pp. 53, 106).

[8] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. "Observational equality, now!" In: *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*. Ed. by Aaron Stump and Hongwei Xi. ACM, 2007, pp. 57–68. DOI: 10.1145/1292597.1292608 (cit. on pp. 43, 98).

[9] Thorsten Altenkirch et al. "Quotient Inductive-Inductive Types". In: *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. Ed. by Christel Baier and Ugo Dal Lago. Vol. 10803.

Lecture Notes in Computer Science. Springer, 2018, pp. 293–310. DOI: 10.1007/978-3-319-89366-2_16 (cit. on p. 67).

[10] Thorsten Altenkirch et al. "Setoid type theory - a syntactic translation". In: *MPC 2019 - 13th International Conference on Mathematics of Program Construction*. Vol. 11825. LNCS. Porto, Portugal: Springer, Oct. 2019, pp. 155–196. DOI: 10.1007/978-3-030-33636-3_7 (cit. on p. 58).

[11] K. Appel and W. Haken. "Every planar map is four colorable. Part I: Discharging". In: *Illinois Journal of Mathematics* 21.3 (1977), pp. 429 –490. DOI: 10.1215/ijm/1256049011 (cit. on p. 17).

[12] Ali Assaf. "A Calculus of Constructions with Explicit Subtyping". In: *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*. Ed. by Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau. Vol. 39. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 27–46. DOI: 10.4230/LIPICS.TYPES.2014.27 (cit. on p. 31).

[13] Brian E. Aydemir et al. "Mechanized Metatheory for the Masses: The PoplMark Challenge". In: *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*. Ed. by Joe Hurd and Thomas F. Melham. Vol. 3603. Lecture Notes in Computer Science. Springer, 2005, pp. 50–65. DOI: 10.1007/11541868_4 (cit. on p. 25).

[14] Henk Barendregt. "Introduction to Generalized Type Systems". In: *J. Funct. Program.* 1.2 (1991), pp. 125–154. DOI: 10.1017/s0956796800020025 (cit. on p. 31).

[15] Gilles Barthe and Tarmo Uustalu. "CPS translating inductive and coinductive types". In: *Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '02), Portland, Oregon, USA, January 14-15, 2002*. Ed. by Peter Thiemann. ACM, 2002, pp. 131–142. DOI: 10.1145/503032.503043 (cit. on pp. 18, 51).

[16] Andrej Bauer. "First Steps in Synthetic Computability Theory". In: *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2005, Birmingham, UK, May 18-21, 2005*. Ed. by Martín Hötzel Escardó, Achim Jung, and Michael W. Mislove. Vol. 155. Electronic Notes in Theoretical Computer Science. Elsevier, 2005, pp. 5–31. DOI: 10.1016/J.ENTCS.2005.11.049 (cit. on p. 75).

[17] Andrej Bauer. "What is algebraic about algebraic effects and handlers?" In: *CoRR* abs/1807.05923 (2018) (cit. on p. 67).

[18] Andrej Bauer and Matija Pretnar. "Programming with algebraic effects and handlers". In: *J. Log. Algebraic Methods Program.* 84.1 (2015), pp. 108–123. DOI: 10.1016/J.JLAMP.2014.02.001 (cit. on p. 67).

[19] Andrej Bauer et al. "Design and Implementation of the Andromeda Proof Assistant". In: *CoRR* abs/1802.06217 (2018) (cit. on p. 43).

[20] Josef Berger. "Aligning the weak König lemma, the uniform continuity theorem, and Brouwer's fan theorem". In: *Ann. Pure Appl. Log.* 163.8 (2012), pp. 981–985. DOI: 10.1016/J.APAL.2011.12.021 (cit. on pp. 63, 74).

[21] Josef Berger. "The Fan Theorem and Uniform Continuity". In: *New Computational Paradigms, First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, June 8-12, 2005, Proceedings*. Ed. by S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet. Vol. 3526. Lecture Notes in Computer Science. Springer, 2005, pp. 18–22. DOI: 10.1007/11494645_3 (cit. on pp. 63, 74).

[22] Josef Berger. "The Logical Strength of the Uniform Continuity Theorem". In: *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*. Ed. by Arnold Beckmann et al. Vol. 3988. Lecture Notes in Computer Science. Springer, 2006, pp. 35–39. DOI: 10.1007/11780342_4 (cit. on pp. 63, 74, 130).

[23] Jean-Philippe Bernardy and Marc Lasson. "Realizability and Parametricity in Pure Type Systems". In: *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by Martin Hofmann. Vol. 6604. Lecture Notes in Computer Science. Springer, 2011, pp. 108–122. DOI: 10.1007/978-3-642-19805-2_8 (cit. on p. 112).

[24] Mark Bickford et al. "Computability Beyond Church-Turing via Choice Sequences". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. Ed. by Anuj Dawar and Erich Grädel. ACM, 2018, pp. 245–254. DOI: 10.1145/3209108.3209200 (cit. on pp. 65, 130).

[25] Simon Pierre Boulier. "Extending type theory with syntactic models. (Etendre la théorie des types à l'aide de modèles syntaxiques)". PhD thesis. Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire, France, 2018 (cit. on pp. 18, 56, 61, 119).

[26] Nuria Brede and Hugo Herbelin. "On the logical structure of choice and bar induction principles". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470523 (cit. on p. 86).

[27] Luitzen Egbertus Jan Brouwer. *Brouwer's Cambridge lectures on intuitionism*. Cambridge University Press, 1981 (cit. on p. 63).

[28] Luitzen Egbertus Jan Brouwer. "Zur Begründung der intuitionistischen Mathematik. I." In: *Mathematische Annalen* 93.1 (1925), pp. 244–257 (cit. on pp. 63, 80).

[29] Cesare Burali-Forti. "Una questione sui numeri transfiniti". In: *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 11.1 (1897), pp. 154–164 (cit. on p. 32).

[30] Venanzio Capretta. "General Recursion via Coinductive Types". In: *Logical Methods in Computer Science* Volume 1, Issue 2 (2005). DOI: 10.2168/lmcs-1(2:1)2005 (cit. on p. 77).

[31] James Chapman. "Type Theory Should Eat Itself". In: *Electronic Notes in Theoretical Computer Science* 228 (2009). Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008), pp. 21–36. DOI: https://doi.org/10.1016/j.entcs.2008.12.114 (cit. on p. 53).

[32] Alonzo Church. "The calculi of lambda-conversion." In: *The Journal of Symbolic Logic* (1941) (cit. on p. 19).

[33] Jesper Cockx. "Dependent Pattern Matching and Proof-Relevant Unification". PhD thesis. Katholieke Universiteit Leuven, Belgium, 2017 (cit. on p. 27).

[34] Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. "The Taming of the Rew: A Type Theory with Computational Assumptions". In: *Proceedings of the ACM on Programming Languages*. POPL 2021 (2021). DOI: 10.1145/3434341 (cit. on p. 44).

[35] Liron Cohen and Vincent Rahli. "Constructing Unprejudiced Extensional Type Theories with Choices via Modalities". In: *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*. Ed. by Amy P. Felty. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 10:1–10:23. DOI: 10.4230/LIPICS.FSCD.2022.10 (cit. on p. 65).

[36] Liron Cohen and Vincent Rahli. "Realizing Continuity Using Stateful Computations". In: *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*. Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 15:1–15:18. DOI: 10.4230/LIPICS.CSL.2023.15 (cit. on pp. 65, 95, 104).

[37] Liron Cohen et al. "Inductive Continuity via Brouwer Trees". In: *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*. Ed. by Jérôme Leroux, Sylvain Lombardy, and David Peleg. Vol. 272. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 37:1–37:16. DOI: 10.4230/LIPICS.MFCS.2023.37 (cit. on pp. 65, 74).

[38] Paul J Cohen. "The independence of the continuum hypothesis". In: *Proceedings of the National Academy of Sciences* 50.6 (1963), pp. 1143–1148 (cit. on p. 136).

[39] Thierry Coquand. "A new paradox in type theory". In: *Logic, Methodology and Philosophy of Science IX*. Ed. by Dag Prawitz, Brian Skyrms, and Dag Westerståhl. Vol. 134. Studies in Logic and the Foundations of Mathematics. Elsevier, 1995, pp. 555–570. DOI: https://doi.org/10.1016/S0049-237X(06)80062-5 (cit. on p. 32).

[40] Thierry Coquand. "An Analysis of Girard's Paradox". In: *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*. IEEE Computer Society, 1986, pp. 227–236 (cit. on pp. 31, 32).

[41] Thierry Coquand and Gérard P. Huet. "The Calculus of Constructions". In: *Inf. Comput.* 76.2/3 (1988), pp. 95–120. DOI: 10.1016/0890-5401(88)90005-3 (cit. on p. 17).

[42] Thierry Coquand and Guilhem Jaber. "A Computational Interpretation of Forcing in Type Theory". In: *Epistemology versus Ontology - Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*. Ed. by Peter Dybjer et al. Vol. 27. Logic, Epistemology, and the Unity of Science. Springer, 2012, pp. 203–213. DOI: 10.1007/978-94-007-4435-6_10 (cit. on pp. 62, 129, 136, 147).

[43] Thierry Coquand and Guilhem Jaber. "A Note on Forcing and Type Theory". In: *Fundam. Informaticae* 100.1-4 (2010), pp. 43–52. DOI: 10.3233/FI-2010-262 (cit. on pp. 62, 129, 136, 145, 180).

[44] Thierry Coquand and Bassel Mannaa. "The Independence of Markov's Principle in Type Theory". In: *Log. Methods Comput. Sci.* 13.3 (2017). DOI: 10.23638/LMCS-13(3:10)2017 (cit. on pp. 136, 179, 180).

[45] Thierry Coquand and Christine Paulin. "Inductively defined types". In: *International Conference on Computer Logic*. Springer. 1988, pp. 50–66 (cit. on p. 17).

[46] Haskell B Curry. "Functionality in combinatory logic". In: *Proceedings of the National Academy of Sciences* 20.11 (1934), pp. 584–590. DOI: 10.1073/pnas.20.11.584 (cit. on p. 17).

[47] N.G. de Bruijn. "The Mathematical Language Automath, its Usage, and Some of its Extensions". In: *Selected Papers on Automath*. Ed. by R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer. Vol. 133. Studies in Logic and the Foundations of Mathematics. Elsevier, 1994, pp. 73–100. DOI: https://doi.org/10.1016/S0049-237X(08)70200-3 (cit. on pp. 17, 25).

[48] Michael Dummett. *Elements of intuitionism*. Vol. 39. Oxford University Press, 2000 (cit. on p. 63).

[49] Peter Dybjer. "Internal Type Theory". In: *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*. Ed. by Stefano Berardi and Mario Coppo. Vol. 1158. Lecture Notes in Computer Science. Springer, 1995, pp. 120–134. DOI: 10.1007/3-540-61780-9_66 (cit. on p. 56).

[50] Martín Escardó and Chuangjie Xu. "A constructive manifestation of the Kleene-Kreisel continuous functionals". In: *Ann. Pure Appl. Log.* 167.9 (2016), pp. 770–793. DOI: 10.1016/j.apal.2016.04.011 (cit. on p. 129).

[51] Martin Hötzel Escardó. "Continuity of Gödel's System T Definable Functionals via Effectful Forcing". In: *Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2013, New Orleans, LA, USA, June 23-25, 2013* (2013). DOI: 10.1016/j.entcs.2013.09.010 (cit. on pp. 10, 62, 74, 82, 104, 106, 128, 129).

[52] Martin Hötzel Escardó and Chuangjie Xu. "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation". In: *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland* (2015). DOI: 10.4230/LIPIcs.TLCA.2015.153 (cit. on pp. 9, 72, 92, 105).

[53] Yannick Forster, Dominik Kirst, and Niklas Mück. "Oracle Computability and Turing Reducibility in the Calculus of Inductive Constructions". In: *CoRR* abs/2307.15543 (2023). DOI: 10.48550/arXiv.2307.15543 (cit. on p. 75).

[54] Gottlob Frege. *Grundgesetze der Arithmetik: begriffsschriftlich abgeleitet*. Vol. 1. H. Pohle, 1893 (cit. on p. 32).

[55] Makoto Fujiwara and Tatsuji Kawai. "Characterising Brouwer's continuity by bar recursion on moduli of continuity". In: *Arch. Math. Log.* 60.1-2 (2021), pp. 241–263. DOI: 10.1007/S00153-020-00740-9 (cit. on pp. 63, 90).

[56] Makoto Fujiwara and Tatsuji Kawai. "Decidable fan theorem and uniform continuity theorem with continuous moduli". In: *Math. Log. Q.* 67.1 (2021), pp. 116–130. DOI: 10.1002/MALQ.202000028 (cit. on p. 90).

[57] Makoto Fujiwara and Tatsuji Kawai. "Equivalence of bar induction and bar recursion for continuous functions with continuous moduli". In: *Ann. Pure Appl. Log.* 170.8 (2019), pp. 867–890. DOI: 10.1016/J.APAL.2019.04.001 (cit. on pp. 73, 74, 90).

[58] Neil Ghani, Peter G. Hancock, and Dirk Pattinson. "Representations of Stream Processors Using Nested Fixed Points". In: *Log. Methods Comput. Sci.* 5.3 (2009) (cit. on p. 70).

[59] Gaëtan Gilbert et al. "Definitional Proof-Irrelevance without K". In: *Proc. ACM Program. Lang.* 3.POPL (2019). DOI: 10.1145/3290316 (cit. on pp. 11, 43, 134, 146, 180).

[60] Eduarde Giménez. "Codifying guarded definitions with recursive schemes". In: *Types for Proofs and Programs*. Ed. by Peter Dybjer, Bengt Nordström, and Jan Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 39–59 (cit. on p. 27).

[61] Eduardo Giménez. "Structural recursive definitions in type theory". In: *Automata, Languages and Programming*. Ed. by Kim G. Larsen, Sven Skyum, and Glynn Winskel. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 397–408 (cit. on p. 27).

[62] Jean-Yves Girard. "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur". Theses. Université Paris VII, 1972 (cit. on p. 32).

[63] Jean-Yves Girard, Yves Lafont, and Paul Taylor. "Proofs and Types, volume 7 of". In: *Cambridge tracts in theoretical computer science* 7 (1989) (cit. on p. 146).

[64] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I". In: *Monatshefte für mathematik und physik* 38 (1931), pp. 173–198 (cit. on p. 53).

[65] Georges Gonthier. *A computer-checked proof of the Four Color Theorem*. Tech. rep. Inria, Mar. 2023 (cit. on p. 17).

[66] Georges Gonthier. "The Four Colour Theorem: Engineering of a Formal Proof". In: *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*. Ed. by Deepak Kapur. Vol. 5081. Lecture Notes in Computer Science. Springer, 2007, p. 333. DOI: 10.1007/978-3-540-87827-8_28 (cit. on p. 17).

[67] Daniel Gratzer, Michael Shulman, and Jonathan Sterling. "Strict universes for Grothendieck topoi". In: *CoRR* abs/2202.12012 (2022) (cit. on p. 129).

[68] Timothy Griffin. "A Formulae-as-Types Notion of Control". In: *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*. Ed. by Frances E. Allen. ACM Press, 1990, pp. 47–58. DOI: 10.1145/96709.96714 (cit. on pp. 18, 51).

[69] Kurt Gödel. *Über die Vollständigkeit des Logikkalküls*. ger. 1929 (cit. on p. 64).

[70] Von Kurt Gödel. "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes". In: *Dialectica* 12.3-4 (1958), pp. 280–287. DOI: https://doi.org/10.1111/j.1746-8361.1958.tb01464.x (cit. on pp. 18, 19).

[71] Robert Harper and Robert Pollack. "Type Checking with Universes". In: *Theor. Comput. Sci.* 89.1 (1991), pp. 107–136. DOI: 10.1016/0304-3975(90)90108-T (cit. on p. 35).

[72] Michael Hedberg. "A coherence theorem for Martin-Löf's type theory". In: *Journal of Functional Programming* 8.4 (July 1998), pp. 413–436. DOI: 10.1017/S0956796898003153 (cit. on p. 42).

[73] Hugo Herbelin. "On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic". In: *Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings*. Ed. by Pawel Urzyczyn. Vol. 3461. Lecture Notes in Computer Science. Springer, 2005, pp. 209–220. DOI: 10.1007/11417170_16 (cit. on pp. 18, 51).

[74] Martin Hofmann. "Conservativity of Equality Reflection over Intensional Type Theory". In: *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*. Ed. by Stefano Berardi and Mario Coppo. Vol. 1158. Lecture Notes in Computer Science. Springer, 1995, pp. 153–164. DOI: 10.1007/3-540-61780-9_68 (cit. on p. 44).

[75] Martin Hofmann. *Extensional constructs in intensional type theory*. CPHC/BCS distinguished dissertations. Springer, 1997 (cit. on pp. 43, 44, 56, 119).

[76] Martin Hofmann and Thomas Streicher. "The Groupoid Model Refutes Uniqueness of Identity Proofs". In: *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*. IEEE Computer Society, 1994, pp. 208–212. DOI: 10.1109/LICS.1994.316071 (cit. on p. 42).

[77] William A Howard. "The formulae-as-types notion of construction". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* 44 (1980), pp. 479–490 (cit. on p. 17).

[78] William A. Howard and Georg Kreisel. "Transfinite Induction and Bar Induction of Types Zero and One, and the Role of Continuity in Intuitionistic Analysis". In: *J. Symb. Log.* 31.3 (1966), pp. 325–358. DOI: 10.2307/2270450 (cit. on p. 63).

[79] Gérard P. Huet. "The Constructive Engine". In: *A Perspective in Theoretical Computer Science - Commemorative Volume for Gift Siromoney*. Ed. by R. Narasimhan. Vol. 16. World Scientific Series in Computer Science. World Scientific, 1989, pp. 38–69. DOI: 10.1142/9789814368452_0004 (cit. on p. 17).

[80] Jasper Hugunin. "Why Not W?" In: *26th International Conference on Types for Proofs and Programs (TYPES 2020)*. Ed. by Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch. Vol. 188. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 8:1–8:9. DOI: 10.4230/LIPIcs.TYPES.2020.8 (cit. on p. 69).

[81] Antonius J. C. Hurkens. "A Simplification of Girard's Paradox". In: *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*. Ed. by Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin. Vol. 902. Lecture Notes in Computer Science. Springer, 1995, pp. 266–278. DOI: 10.1007/BFb0014058 (cit. on p. 32).

[82] J. M. E. Hyland and C.-H. Luke Ong. "On Full Abstraction for PCF: I, II, and III". In: *Inf. Comput.* 163.2 (2000), pp. 285–408. DOI: 10.1006/inco.2000.2917 (cit. on p. 70).

[83] Bart Jacobs. "Comprehension Categories and the Semantics of Type Dependency". In: *Theor. Comput. Sci.* 107.2 (1993), pp. 169–207. DOI: 10.1016/0304-3975(93)90169-T (cit. on p. 56).

[84] Peter T Johnstone. "On a topological topos". In: *Proceedings of the London mathematical society* 3.2 (1979), pp. 237–271 (cit. on p. 94).

[85] Tom de Jong. "Domain Theory in Constructive and Predicative Univalent Foundations". In: *CoRR* abs/2301.12405 (2023). DOI: 10.48550/ARXIV.2301.12405 (cit. on p. 104).

[86] Ambrus Kaposi, András Kovács, and Nicolai Kraus. "Shallow Embedding of Type Theory is Morally Correct". In: *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*. Ed. by Graham Hutton. Vol. 11825. Lecture Notes in Computer Science. Springer, 2019, pp. 329–365. DOI: 10.1007/978-3-030-33636-3_12 (cit. on pp. 53, 119).

[87] Tatsuji Kawai. "Principles of bar induction and continuity on Baire space". In: *J. Log. Anal.* 11 (2019) (cit. on p. 90).

[88] Oleg Kiselyov and Hiromi Ishii. "Freer monads, more extensible effects". In: *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell, Haskell 2015, Vancouver, BC, Canada, September 3-4, 2015*. Ed. by Ben Lippmeier. ACM, 2015, pp. 94–105. DOI: 10.1145/2804302.2804319 (cit. on p. 70).

[89] S.C. Kleene. "Recursive Functionals and Quantifiers of Finite Types Revisited I". In: *Generalized Recursion Theory II*. Ed. by J.E. Fenstad, R.O. Gandy, and G.E. Sacks. Vol. 94. Studies in Logic and the Foundations of Mathematics. Elsevier, 1978, pp. 185–222. DOI: https://doi.org/10.1016/S0049-237X(08)70933-9 (cit. on pp. 63, 70).

[90] Stephen Cole Kleene. "Recursive functionals and quantifiers of finite types. I". In: *Transactions of the American Mathematical Society* 91.1 (1959), pp. 1–52 (cit. on p. 63).

[91] Michal Konecný, Florian Steinberg, and Holger Thies. "Continuous and monotone machines". In: *CoRR* abs/2005.01624 (2020) (cit. on p. 88).

[92] Nicolai Kraus et al. "Generalizations of Hedberg's Theorem". In: *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer, 2013, pp. 173–188. DOI: 10.1007/978-3-642-38946-7_14 (cit. on p. 42).

[93] Georg Kreisel. "On Weak Completeness of Intuitionistic Predicate Logic". In: *J. Symb. Log.* 27.2 (1962), pp. 139–158. DOI: 10.2307/2964110 (cit. on p. 64).

[94] Jean-Louis Krivine. "Opérateurs de mise en mémoire et traduction de Gödel". In: *Arch. Math. Log.* 30.4 (1990), pp. 241–267. DOI: 10.1007/BF01792986 (cit. on p. 48).

[95] Gottfried Wilhelm Leibniz. "Discourse on Metaphysics". In: (1686) (cit. on p. 40).

[96] Meven Lennon-Bertrand. "Bidirectional Typing for the Calculus of Inductive Constructions". Theses. Nantes Université, June 2022 (cit. on pp. 24, 37).

[97] Rodolphe Lepigre. "A Classical Realizability Model for a Semantical Value Restriction". In: *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April*

2-8, 2016, Proceedings. Ed. by Peter Thiemann. Vol. 9632. Lecture Notes in Computer Science. Springer, 2016, pp. 476–502. DOI: 10.1007/978-3-662-49498-1_19 (cit. on p. 47).

[98] Zhaohui Luo. "An extended calculus of constructions". PhD thesis. University of Edinburgh, UK, 1990 (cit. on p. 17).

[99] Zhaohui Luo. "ECC, an Extended Calculus of Constructions". In: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 1989, pp. 386–395. DOI: 10.1109/LICS.1989.39193 (cit. on p. 17).

[100] Zhaohui Luo. "Notes on universes in type theory". In: *Lecture notes for a talk at Institute for Advanced Study, Princeton* (2012), p. 16 (cit. on p. 31).

[101] Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 2012 (cit. on p. 96).

[102] Anatolii Malcev. "Untersuchungen aus dem Gebiete der mathematischen Logik". In: *Matematicheskiĭ Sbornik* 1.3 (1936), pp. 323–336 (cit. on p. 64).

[103] Per Martin-Löf. *Intuitionistic type theory*. Vol. 1. Studies in proof theory. Bibliopolis, 1984 (cit. on pp. 28, 31, 32).

[104] Per Martin-Löf. "A Theory of Types". In: *Technical report 71-3, University of Stockholm* (1971) (cit. on pp. 17, 28, 31, 32).

[105] Conor McBride. "Turing-Completeness Totally Free". In: *Mathematics of Program Construction - 12th International Conference, MPC 2015, Königswinter, Germany, June 29 - July 1, 2015. Proceedings*. Ed. by Ralf Hinze and Janis Voigtländer. Vol. 9129. Lecture Notes in Computer Science. Springer, 2015, pp. 257–275. DOI: 10.1007/978-3-319-19797-5_13 (cit. on p. 70).

[106] Alexandre Miquel. "A Survey of Classical Realizability". In: *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*. Ed. by C.-H. Luke Ong. Vol. 6690. Lecture Notes in Computer Science. Springer, 2011, pp. 1–2. DOI: 10.1007/978-3-642-21691-6_1 (cit. on p. 95).

[107] Alexandre Miquel. "Forcing as a Program Transformation". In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*. IEEE Computer Society, 2011, pp. 197–206. DOI: 10.1109/LICS.2011.47 (cit. on p. 95).

[108] Étienne Miquey. "A constructive proof of dependent choice in classical arithmetic via memoization". In: *CoRR* abs/1903.07616 (2019) (cit. on p. 95).

[109] Jaap van Oosten. "Partial Combinatory Algebras of Functions". In: *Notre Dame Journal of Formal Logic* 52.4 (2011), pp. 431 – 448. DOI: 10.1215/00294527-1499381 (cit. on p. 75).

[110] Jaap van Oosten. "Realizability: an introduction to its categorical side. Studies in Logic and the Foundations of Mathematics, vol. 152. Elsevier Science, Amsterdam, 2008, 328 pp." In: *The Bulletin of Symbolic Logic* 16.3 (2008). DOI: 10.1017/S1079898600000858 (cit. on p. 75).

[111] Nicolas Oury. "Extensionality in the Calculus of Constructions". In: *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*. Ed. by Joe Hurd and Thomas F. Melham. Vol. 3603. Lecture Notes in Computer Science. Springer, 2005, pp. 278–293. DOI: 10.1007/11541868_18 (cit. on p. 44).

[112] Christine Paulin-Mohring. "Inductive Definitions in the system Coq - Rules and Properties". In: *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*. Ed. by Marc Bezem and Jan Friso Groote. Vol. 664. Lecture Notes in Computer Science. Springer, 1993, pp. 328–345. DOI: 10.1007/BFb0037116 (cit. on pp. 17, 46).

[113] Pierre-Marie Pédrot. "Debunking Sheaves". note. 2021 (cit. on pp. 65, 96).

[114] Pierre-Marie Pédrot. "Pursuing Shtuck". working paper or preprint. Oct. 2023 (cit. on pp. 65, 96).

[115] Pierre-Marie Pédrot. "Russian Constructivism in a Prefascist Theory". In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns et al. ACM, 2020, pp. 782–794. DOI: 10.1145/3373718.3394740 (cit. on pp. 6, 18, 132, 136).

[116] Pierre-Marie Pédrot and Nicolas Tabareau. "The fire triangle: how to mix substitution, dependent elimination, and effects". In: *Proc. ACM Program. Lang.* 4.POPL (2020), 58:1–58:28. DOI: 10.1145/3371126 (cit. on pp. 10, 46, 104).

[117] Maciej Piróg and Jeremy Gibbons. "The Coinductive Resumption Monad". In: *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014, Ithaca, NY, USA, June 12-15, 2014*. Ed. by Bart Jacobs, Alexandra Silva, and Sam Staton. Vol. 308. Electronic Notes in Theoretical Computer Science. Elsevier, 2014, pp. 273–288. DOI: 10.1016/j.entcs.2014.10.015 (cit. on p. 70).

[118] Gordon D. Plotkin and Matija Pretnar. "Handling Algebraic Effects". In: *Log. Methods Comput. Sci.* 9.4 (2013). DOI: 10.2168/LMCS-9(4:23)2013 (cit. on p. 67).

[119] Loïc Pujet and Nicolas Tabareau. "Impredicative Observational Equality". In: *POPL 2023 - 50th ACM SIGPLAN Symposium on Principles of Programming Languages*. Vol. 7. Proceedings of the ACM on programming languages. Boston, United States, Jan. 2023, p. 74. DOI: 10.1145/3571739 (cit. on pp. 43, 98, 146, 180).

[120] Loïc Pujet and Nicolas Tabareau. "Observational Equality: Now for Good". In: *Proc. ACM Program. Lang.* 6.POPL (2022). DOI: 10.1145/3498693 (cit. on pp. 11, 43, 53, 98, 146, 180).

[121] Pierre-Marie Pédrot and Nicolas Tabareau. "An effectful way to eliminate addiction to dependence". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017* (2017). DOI: 10.1109/LICS.2017.8005113 (cit. on pp. 6, 10, 18, 46, 47, 53, 66, 104, 110, 120, 121, 123, 125).

[122] Pierre-Marie Pédrot and Nicolas Tabareau. "Failure is Not an Option An Exceptional Type Theory". In: *ESOP 2018 - 27th European Symposium on Programming* (2018). DOI: https://doi.org/10.1007/978-3-319-89884-1_9 (cit. on pp. 47, 52, 61, 132).

[123] Vincent Rahli and Mark Bickford. "Validating Brouwer's continuity principle for numbers using named exceptions". In: *Math. Struct. Comput. Sci.* 28.6 (2018), pp. 942–990. DOI: 10.1017/S0960129517000172 (cit. on pp. 65, 72, 95, 104, 130, 132).

[124] Vincent Rahli et al. "Bar Induction is Compatible with Constructive Type Theory". In: *J. ACM* 66.2 (2019), 13:1–13:35. DOI: 10.1145/3305261 (cit. on pp. 65, 72, 130).

[125] Egbert Rijke, Michael Shulman, and Bas Spitters. "Modalities in homotopy type theory". In: *Logical Methods in Computer Science* Volume 16, Issue 1 (Jan. 2020). DOI: 10.23638/LMCS-16(1:2)2020 (cit. on p. 129).

[126] Pierre-Marie Pédrot Simon Boulier and Nicolas Tabareau. "The next 700 syntactical models of type theory". In: *Certified Programs and Proofs (CPP 2017), Jan 2017, Paris, France. pp.182 - 194* (2017). DOI: 10.1145/3018610.3018620 (cit. on pp. 18, 56, 58).

[127] Matthieu Sozeau, Meven Lennon-Bertrand, and Yannick Forster. "The Curious Case of Case: Correct & Efficient Representation of Case Analysis in Coq and MetaCoq". In: *Talk. 1st Workshop on the Implementation of Type Systems*. 2022 (cit. on p. 37).

[128] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. "Autosubst 2: reasoning with multi-sorted de Bruijn terms and vector substitutions". In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*. 2019, pp. 166–180. DOI: 10.1145/3293880.3294101 (cit. on p. 165).

[129] Florian Steinberg, Laurent Théry, and Holger Thies. "Computable analysis and notions of continuity in Coq". In: *Log. Methods Comput. Sci.* 17.2 (2021) (cit. on p. 88).

[130] Jonathan Sterling. "Higher order functions and Brouwer's thesis". In: *Journal of Functional Programming* 31 (2021). *Bob Harper Festschrift Collection*, e11. DOI: 10.1017/S0956796821000095 (cit. on pp. 68, 80, 108, 129).

[131] Thomas Streicher. "Investigations into intensional type theory". PhD thesis. Habilitationsschrift, Ludwig-Maximilians-Universität München, 1993 (cit. on p. 42).

[132] Wouter Swierstra. "Data types à la carte". In: *J. Funct. Program.* 18.4 (2008), pp. 423–436. DOI: 10.1017/S0956796808006758 (cit. on p. 70).

[133] William W Tait. "Constructive reasoning". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 52. Elsevier, 1968, pp. 185–199 (cit. on p. 63).

[134] William W. Tait. "Intensional Interpretations of Functionals of Finite Type I". In: *J. Symb. Log.* 32.2 (1967), pp. 198–212. DOI: 10.2307/2271658 (cit. on p. 146).

[135] "The Coq Proof Assistant (8.17)". In: (2023). DOI: 10.5281/zenodo.1003420 (cit. on p. 17).

[136] Amin Timany and Matthieu Sozeau. "Cumulative Inductive Types In Coq". In: *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*. Ed. by Hélène Kirchner. Vol. 108. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 29:1–29:16. DOI: 10.4230/LIPIcs.FSCD.2018.29 (cit. on p. 35).

[137] Anne S Troelstra. "A note on non-extensional operations in connection with continuity and recursiveness". In: *Indagationes Mathematicae (Proceedings)*. Vol. 80. 5. Elsevier. 1977, pp. 455–462 (cit. on p. 64).

[138] Anne Sjerp Troelstra. "Constructivism in mathematics". In: *An Introduction* 2 (1988) (cit. on pp. 63, 90).

[139] Anne Sjerp Troelstra. *Metamathematical investigation of intuitionistic arithmetic and analysis*. Springer, 1973 (cit. on pp. 63, 90).

[140] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cit. on pp. 44, 67).

[141] Jaap Van Oosten. "A combinatory algebra for sequential functionals of finite type". In: *LONDON MATHEMATICAL SOCIETY LECTURE NOTE SERIES* (1999), pp. 389–406 (cit. on p. 75).

[142] Benjamin Werner. "Sets in Types, Types in Sets". In: *Theoretical Aspects of Computer Software, Third International Symposium, TACS '97, Sendai, Japan, September 23-26, 1997, Proceedings*. Ed. by Martín Abadi and Takayasu Ito. Vol. 1281. Lecture Notes in Computer Science. Springer, 1997, pp. 530–346. DOI: 10.1007/BFB0014566 (cit. on p. 56).

[143] Paweł Wieczorek and Dariusz Biernacki. "A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2018. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 266–279. DOI: 10.1145/3167091 (cit. on p. 146).

[144] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. "Eliminating reflection from type theory". In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*. Ed. by Assia Mahboubi and Magnus O. Myreen. ACM, 2019, pp. 91–103. DOI: 10.1145/3293880.3294095 (cit. on pp. 44, 114).

[145] Théo Winterhalter. "Formalisation and meta-theory of type theory". PhD thesis. Université de Nantes, 2020 (cit. on p. 56).

[146] Li-yao Xia et al. "Interaction trees: representing recursive and impure programs in Coq". In: *Proc. ACM Program. Lang.* 4.POPL (2020), 51:1–51:32. DOI: 10.1145/3371119 (cit. on pp. 70, 77).

[147] Chuangjie Xu and Martín Hötzel Escardó. "A Constructive Model of Uniform Continuity". In: *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer, 2013, pp. 236–249. DOI: 10.1007/978-3-642-38946-7_18 (cit. on p. 129).

[148] Ernst Zermelo. "Beweis, dass jede Menge wohlgeordnet werden kann: Aus einem an Herrn Hilbert gerichteten Briefe". In: *Mathematische Annalen* 59.4 (1904), pp. 514–516 (cit. on pp. 17, 44, 50).

[149] Ernst Zermelo. "Untersuchungen über die Grundlagen der Mengenlehre. I". In: *Mathematische Annalen* 65.2 (1908), pp. 261–281 (cit. on pp. 17, 44, 50).

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

Titre : Continuité en théorie des types

Mots clés : Continuité, Théorie des types, Assistants à la preuve, Preuve de normalisation, modèles syntaxiques.

Résumé : Dans cette thèse, j'étudie l'interaction entre la théorie des types et la continuité, un concept mathématique formalisant l'intuition qu'une fonction ne peut interroger qu'une partie finie de son argument avant de renvoyer une valeur
Dans le premier chapitre, je décris la théorie des types et mon prisme de lecture : la correspondance preuve-programme, ou isomorphisme de Curry-Howard, qui affirme que calcul et preuve sont deux faces d'une même pièce. J'y explique comment la théorie des types bute encore sur l'intégration de principes dits classiques, comme l'axiome du choix ou le tiers-exclu.
Dans le deuxième chapitre, j'analyse les différentes définition de la continuité, et comment celles-ci diffèrent les unes des autres d'un point de vue logique.
Dans le troisième chapitre, je présente une première tentative d'intégration de la continuité en théorie des types, à travers un modèle syntaxique d'une théorie des types particulières, appelée *Baclofen Type Theory.*
Enfin, dans le dernier chapitre, je détaille une théorie des types où toutes les fonctions sont continues, et présente des résultats préliminaires de normalisation de cette théorie.

Title : Continuity in Type Theory

Keywords : Continuity, Type Theory, Proof Assistants, Normalisation Procedure, Syntactic Models

Abstract : In this thesis, I study the interaction between type theory and continuity, a mathematical concept describing the intuition that a function can only query a finite part of its argument before returning a value.
In the first Chapter, I describe type theory and my working paradigm: the proof-program correspondence, or Curry-Howard isomorphism, which asserts that computation and proof are two sides of the same coin. I explain how type theory is still struggling with the integration of so-called classical principles, such as the axiom of choice or the excluded-middle.
In a second part I survey the different definitions of continuity, and how they differ from each other from a logical point of view.
In the third Chapter I present a first attempt to mingle continuity and type theory, through a syntactic model of a particular type theory, dubbed *Baclofen Type Theory*.
Finally, in the last Chapter I define a type theory where all functions are continuous, and present preliminar results on the proof of its normalisation.