On Semantic and Type-Theoretic Aspects of Polynomial-Time Computability

Andrzej S. Murawski

St Hugh's College, Oxford

Submitted for the degree of Doctor of Philosophy
Trinity Term 2001

Oxford University Computing Laboratory
Programming Research Group

Abstract

This thesis strives to combine two traditions in theoretical computer science: complexity theory and denotational semantics. The recently proposed setting of game semantics, which provided first syntax-independent fully abstract models of many programming languages, is used to characterize the polynomial-time computable functions as those definable by a certain kind of strategies in two-player games. In particular, players must comply with a network protocol that structures the exchange of moves between players and restricts the way games can be modified by them during play.

The tight correspondence is achieved through a faithful characterization of the space of proofs in light affine logic, which itself captures polynomial-time computation by the process of cut-elimination. A representation theory of proofs, extending essential nets for intuitionistic linear logic, is developed for light affine logic and serves as a link between proofs and strategies. The nets corresponding to proofs distinguish themselves by satisfying such graph-theoretic properties as acyclicity, domination and well-bracketing. Proofs in normal form are then identified as canonical nets and related to strategies by a full and faithful completeness result.

We also investigate the algorithmic content of light affine logic in the form of a simple programming language incorporating a form of safe recursion. The language is demonstrated to express all polynomial-time computable functions and to be compositionally interpretable in the logic which guarantees that it can be executed in polynomial-time using the cut-elimination procedure and also interpreted in the game model.

The results of the thesis constitute a first successful attempt to construct an abstract denotational semantics capturing a complexity-theoretic concept. It is hoped that they will initiate an alternative approach to complexity based on the analysis of the semantic content of complexity classes.

Acknowledgements

First and foremost, I would like to thank my supervisor, Luke Ong, for inspiring supervision and invaluable advice on research and life in academia in general. I am grateful to all other members of the Foundations of Computer Science group at Oxford for a stimulating atmosphere over the three years. The EU TMR Network on Linear Logic in Computer Science was also an excellent forum for discussion about the topics related to this thesis; I have benefited a great deal from interactions with the network's members. For help at various stages of writing I am indebted to Patrick Baillot, Keye Martin, Jeff Sarnat and Ben Worrell. The diagrams were prepared using Paul Taylor's macro package.

My studies would have been impossible without the University of Oxford Scatcherd European Scholarship and the Overseas Research Students Award. In my third year I was also supported by the Searle Graduate Scholarship of St Hugh's College.

I owe an incalculable debt of gratitude to my family, especially to my Mother, who has been helping me during the studies in every conceivable way; to my Aunt Barbara, who has always been there for me, offering timely words of support; and to my cousin Justyna, who has been doing her best to keep in touch. Finally, I want to thank my fiancée, Aleksandra, for thousands of e-mails, a hundred letters and, much more importanly, for her understanding and love.

To my Mother

Contents

Abstract **i**

Acknowledgments **ii**

1 Introduction **1**

2 The many facets of PTIME **7**

2.1 Machine-dependent definitions . . . 7
2.2 Logical characterizations . . . 10
2.3 Programming language characterizations . . . 13
2.4 Lambda calculi . . . 17
2.5 Light Affine Logic . . . 18
2.5.1 Term calculus . . . 20
2.5.2 Commuting conversions . . . 23
2.5.3 Reduction . . . 24
2.6 ELL and SLL . . . 28

3 Programming with IMLAL2 **30**

3.1 Iteration . . . 30
3.2 Safe and normal variables in IMLAL2 . . . 32
3.3 Linear safe recursion . . . 33
3.4 Completing BC^- . . . 37
3.4.1 **perm-rec**: recursion on *safe* arguments . . . 38
3.4.2 Terms of type BIN $\multimap$ BIN. . . . 39
3.5 $BC^\pm$ characterizes FP . . . 40
3.5.1 $BC^\pm$ is FP closed . . . 40
3.5.2 Any FP function is representable in $BC^\pm$ . . . 44
3.6 Another FP completion of BC^- . . . 47
3.7 Further directions . . . 52

4 Essential nets **54**

4.1 IMLL . . . 55
4.1.1 Polarization . . . 55

4.1.2 Nets for polarized sequents 57
4.1.3 Interpretation of proofs 60
4.1.4 Sequentializability 61
4.1.5 Dominator trees 62
4.1.6 A sequentialization algorithm 65
4.2 IMLLL 69
4.2.1 Correctness 71
4.2.2 Soundness and sequentialization 74
4.3 IMLLL2 77
4.3.1 IMLLL2 nets 78
4.3.2 Protonets 84
4.4 IMLAL2 87
4.4.1 Correctness 88
4.4.2 Sequentialization 91
4.4.3 Collected and canonical nets 92
4.5 Extensions 96
4.6 Terms and nets 97

5 Some category theory 98
5.1 Autonomous categories 98
5.2 Symmetric monoidal functors 104
5.3 Weakening and garbage collection 108
5.4 Commutative comonoid 110
5.5 Terminal object 111
5.6 Light affine categories 112
5.7 Type variables and universal quantification 112
5.8 Light affine hyperdoctrines 114

6 Game models 116
6.1 Games and strategies 117
6.2 IMAL 121
6.2.1 IMAL games 121
6.2.2 IMAL strategies 122
6.2.3 Affine nets defined by strategies 126
6.3 IMLAL 130
6.3.1 ! and § games 131
6.3.2 IMLAL games 132
6.3.3 Network protocol 133
6.3.4 Networked strategies 139
6.3.5 Equivalence of positions 144
6.3.6 The model 149
6.3.7 Full Completeness 151

6.4 IMLAL2 . . . 155
6.4.1 Static games . . . 155
6.4.2 Symbolic and expanded strategies . . . 163
6.4.3 Local strategies . . . 166
6.4.4 Bounded strategies . . . 168
6.4.5 Consistent strategies . . . 174
6.4.6 Winning strategies . . . 175
6.4.7 Uniformity . . . 175
6.4.8 A light affine hyperdoctrine . . . 176
6.4.9 Full completeness . . . 179

7 Future work **183**
7.1 Semantics . . . 183
7.2 Syntax . . . 184

Bibliography **185**

Index **194**

Chapter 1

Introduction

Complexity theory and denotational semantics have tended to evolve without much interaction. Semantics has traditionally been concerned with a sound mathematical interpretation of computational phenomena. Complexity theory has dealt with quantitative aspects of computation and seemed to favour more concrete analysis. This thesis is an attempt to bridge the gap.

Complexity classes were originally defined using machine models. These low-level descriptions might have led to the suspicion that complexity theory would be confined to combinatorial manipulations with Turing machines and would escape abstract analysis. However, it turned out that many of the definitions are robust and coincide for various machine models. For a large number of complexity classes even machine-independent characterizations have been found, revealing challenges for other theories.

A potential area for application of semantic methods stems from the Curry-Howard correspondence—the idea of considering proofs as programs, types as formulas and modelling computation by the process of cut-elimination in logic. In order to exploit this paradigm in complexity theory, an appropriate logical system taking into account the consumption of resources must be considered, for only then can one hope to be in control of conditions relevant to complexity. Indeed, Light Linear Logic proposed by Girard [42] and its simplification by Asperti [11] called Light Affine Logic (LAL) both build upon the sensitivity of Girard's Linear Logic [39] to the use of premises. By ensuring that duplicable entities have at most one input, they rule out the possibility of exponential explosion. Natural numbers in these logics can be represented by proofs of the formula

$$\textsc{bin} = \forall\alpha.!(\alpha \multimap \alpha) \multimap (!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha))\,,$$

which correspond to binary strings. Crucially, any proof of $\textsc{bin} \multimap \S^k\ \textsc{bin}$ for some k defines a polynomial-time computable function (soundness) and, conversely, any such function can be expressed by a proof of $\text{BIN} \multimap \S^l\ \text{BIN}$ for some l (completeness). Besides, LAL is not only a machine-independent characterization of a complexity class, but also an intrinsic one: the process that produces the polynomial bounds is the logical counterpart of computation: cut-elimination.

Light Affine Logic opens up the possibility of constructing an abstract model of polynomial-time computability. Its formal structure facilitates the specification of requirements needed to interpret its syntax and thus polynomial-time algorithms. Nevertheless, to be of value to complexity theory, the model should not only interpret the syntax, but also do so in an undistorted way, as specified by the notions of full and faithful completeness [3]. It ought to be free from undefinable elements and, ideally, different proofs of the same formula ought to have different denotations. It has turned out very difficult to discover syntax-independent constructions of such models and only the recent advances in game semantics seem to have provided a methodology allowing one to try to address the issue.

A variety of game models for fragments of linear logic [3, 63, 1, 14, 15, 10, 95, 100] and intuitionistic logic [62, 60, 102, 61] have been proposed, a large majority of which are fully and faithfully complete. These early developments have stimulated progress in other areas and some models based on domain theory [84, 85] and Chu-spaces [36] emerged shortly afterwards. However, the full completeness problem for full linear logic still remains open. Game semantics also gained recognition for solving the long-standing full abstraction problem for the paradigmatic functional programming language PCF [6, 64, 101]. Since then the observational behaviour of many languages with more advanced features has been accounted for in [7, 8, 59, 93, 8, 2, 78, 79]. Lately game semantics has found its first applications in program analysis [88, 89], security [90] and verification of program equivalence [38].

The main result of this thesis is a fully and faithfully complete model for Light Affine Logic (more precisely, for its second-order intuitionistic multiplicative fragment $(\forall, \S, !, \otimes, \multimap)$ called IMLAL2). Hence, the model captures polynomial-time computability in a semantic fashion and is a machine-independent framework whose elements denote polynomial-time computable functions. No such model has been constructed thus far, although the semantics of LAL has been studied in the setting of phase [71] and coherence [13] spaces. The former provide a strong completeness result, i.e. provability in LAL is characterized faithfully rather than the space of proofs. The latter do interpret proofs, but the model is not fully complete, does not interpret polymorphism and the type of natural numbers is introduced as a base type. Therefore, it is not clear what functions are really definable. Semantic methods based on realizability have also been used extensively in Hofmann's work [55, 56, 57] to study the relationships between complexity classes and programming languages. However, their use consisted in structuring the machine-dependent universe rather than replacing it with more abstract notions.

Overview

In Chapter 2 we survey the many ways in which polynomial-time computability is present in computer science and logic. We begin with the standard definitions of the complexity class P [27], the associated functional class FP and the widely accepted claims as to their suitability for formal definitions of feasible and inherently sequential computation. But P also arises in other contexts in complexity theory even though the polynomial bounds are not specified explicitly: in connection with special kinds of automata [29] or alternating Turing machines [26]. Further characterizations come from finite model theory and can be formulated in first-order logic [114, 66, 111, 48, 49], second-order logic [46], their bounded versions [21] (this result uses a single infinite model for a change) and the simply typed lambda calculus of order 4 [53]. Analogous results have also been obtained by taking advantage of a properly calibrated functional programming language (e.g. without *cons*) either with first-order unlimited recursion [48, 45, 69, 70], or second-order primitive recursion [45, 70]. A subset of PROLOG without functional constants will do as well [103, 67]. However, to compute FP functions the availability of *cons* ought to be reinstated, which calls for new methods of reducing complexity. Cobham's paper [27] on bounded recursion was the first contribution to this goal. The idea has been further developed in a logical context in [30, 24, 82], the last of which has in turn suggested the stratification (tiering) technique [81, 16]. The method has inspired many researchers [25, 92, 23], was shown to equivalent to a modal typing system [54] and extended to higher order functions [55]. Moreover, the stratification regime turned out unnecessary in frameworks that prevent functions from increasing the size of input [56].

Another series of results that take advantage of reduction in lambda calculi was started in [83] (in which the spirit of tiering is still perceptible) and resulted in subsequent improvements motivated by linear logic in the form of bounded linear logic [44] and light linear logic [42]. Chapter 2 culminates in a thorough introduction to light affine logic [11], its syntax, the rewrite system and detailed explanations of its properties. At the very end, we shortly discuss Soft Linear Logic [77]—the latest development in the area capturing P rather than FP.

Chapter 3 investigates the scope for programming in Light Affine Logic. In particular, we seek connections with the Bellantoni-Cook algebra BC [16, 81] of FP functions, in which input arguments are divided into normal and safe ones. Having identified a related concept of normal and safe in the logic, we show that a subsystem of BC, called BC^-, can be translated into LAL so that normal and safe arguments correspond to normal and safe variables. In BC^- safe arguments are not duplicable, and many FP functions become undefinable. For that reason we try to extend BC^- with new constructs aiming for FP-complete systems that still admit a compositional translation into LAL. We propose two such by strengthening the power of safe variables. The first one, named $BC^\pm$, is BC^- enriched with a case construct $\mathbf{case}_K$ that replaces a fixed number of bits at both ends of a safe argument (depending on a fixed number of least significant bits) and a recursion scheme **perm-rec** over *safe variables* (in contrast to BC) in which at each step a new bit can be added and a fixed number of the least significant bits of the intermediate result permuted. The latter functionality is no longer necessary to capture FP if one replaces **perm-rec** with the base function **e-shift** that shifts all the even-numbered bits to the left. We also show that safe variables are not contractible in LAL, because they could then be used to define superpolynomial computations. The results of this chapter have been presented in [96].

In Chapter 4 we introduce graph-theoretic representations of LAL proofs, which will provide a link between the sequent calculus and strategies in our game models, and simplify proofs of full and faithful completeness. The most important results of the chapter are the correctness criteria for various kinds of nets that characterize precisely the nets that arise from proofs by referring to properties of the underlying directed graphs. Our work builds upon Lamarche's essential nets [80]: we adopt the original definitions for IMLL $(\otimes, \multimap)$ and independently reprove the adequacy of his correctness criterion. For IMLL it boils down to acyclicity and a domination property for $⅋^+$-nodes: all paths from the root to the left premise of a $⅋^+$-node must pass through the $⅋^+$-node in question. Then we extend the framework to handle § and ! by requiring the existence of a global matching between the negative nodes $§^-, ?^-$ and $§^+, !^+$ such that the matching nodes induce well-formed sequences of 'brackets' in any path of the net starting from the root. $!^+$-nodes can be matched only by single $?^-$-nodes (if any) and all paths between the premises of $⅋^+$-nodes must be well-bracketed in the above sense as well as paths connecting the root of the net with any of conclusions of the net. For second-order quantification another domination condition is necessary, this time for $\forall^+$-nodes: any path from the root to a $\exists^-$-node whose eigentype has a free occurrence of X visits the $\forall^+_X$-node if such a node exists, and all such paths must be well-formed. Finally, when weakening is taken into account, a correct net is one that can be transformed to a standard correct net with the aid of additional links. We also define canonical forms of such nets for which the criterion about extra links collapses to the old form. Canonical nets represent terms with collected garbage, which ensures that the canonical proofs of BIN correspond exactly to binary strings. These nets will be shown to correspond to strategies of our game models in Chapter 6.

Chapter 5 is an interlude on category theory in which a categorical semantics for Light Affine Logic is derived. The categorical structures needed to model the quantifier-free part of Light Affine Logic are called *light affine categories*, their second-order counterparts are *light affine hyperdoctrines*. In addition, we examine briefly the special case of the tensor unit being a terminal object with which we have to deal in our game semantics.

Chapter 6 presents a series of game models for increasing fragments of light affine logic. They follow the AJM-style of play [1, 6] and involve two players O and P who play games defined by logical formulas by making moves alternately. The first model is fully and faithfully complete for the $(\otimes, \multimap)$-fragment and is based on total, injective history-free, token-reflecting strategies. For ! and § we introduce a special protocol of conduct which is obeyed by the players. The occurrences of ! and § are regarded as opening threads of play, which are then organized into networks:

- A network comprises an O-thread (which can only be created by O) and finitely many P-threads (which can only be created by P).
- A network whose O-thread arises from a !-game can have at most one P-thread which must also arise from a !-game.
- No thread can belong to more than one network.
- Only O can switch between networks, and only P can switch between threads within the same network.

A description of the game model for $(\otimes, \multimap)$ arising from our work (accompanied by an accurate account of the theory of the tensor unit) has appeared as [95]. An extended abstract presenting the model for § and ! was presented in [97]. Our first attempt to introduce quantification into the $(\otimes, \multimap)$-framework was also reported in [100]. Its basic idea follows the evolving game paradigm [60, 102] which assumes that players can change the game when making moves. These special evolutionary moves are called *evolution arguments*. Our framework incorporates them into ordinary moves in contrast to previous game models that isolated evolution arguments from moves. All the game models are the first fully and faithfully complete models for the respective fragments.

To capture full Light Affine Logic, two further developments turned out to be necessary. Firstly, it is much more difficult to describe the evolution of the game when it can concern just some single threads of the !-game so a generalized !-game (written $!_{\{\,G_i\,|\,i \in I\,\}}G$) had to be introduced. For the same reason it becomes harder to pin down the finitely representable strategies as it is too strict to require that the history-free function be finite and it is insufficient to stipulate that all networks have a finite number of threads. Instead, more parameters have to be uniformly bounded: the maximal size of the game that can be developed, the size of games that players can use as evolution arguments and the number of threads occurring in networks as specified in the definition of *bounded* strategies. Only then, assuming that the strategy behaves consistently, can we extract its finite representation and hope to relate it with proofs. Secondly, the evolution of a game must be compatible with the network structure: evolution arguments provided by O in a network, can only be used by P *locally*, i.e. inside the same network, though possibly at a different level. The faithful and full completeness results are then established by relating a special kind of positions (*short-sighted* and *symbolic*) with paths in the associated net. Thus we have succeeded in characterizing polynomial-time computability by a class of strategies.

Finally, we discuss the potential applications of our work in Chapter 7 as well as some future directions for research.

Chapter 2

The many facets of PTIME

PTIME is one of the most intensively studied complexity classes. It was introduced in the 1960s by Alan Cobham, under the name $\mathcal{L}^*$, as the smallest class containing natural decision problems which could be characterized in terms of time-bounded computers. Since then PTIME has been the subject of numerous open problems in complexity theory, many of which still remain unsolved. It is also regarded as a sensible formalization of feasibility (Cook's Thesis). The hardest among PTIME problems (the P-complete problems) are considered inherently sequential and commonly thought to set the limits for efficient parallel computing (with $(\log n)^{O(1)}$ time complexity on $n^{O(1)}$ processors) [47]. In this chapter we give a survey of non-standard definitions of the class PTIME (P) of decision problems and the associated class FPTIME (FP) of numeric functions.

2.1 Machine-dependent definitions

Cobham's original definition of $\mathcal{L}^*$ [27] read:

> A set A of strings is in the class $\mathcal{L}^*$ if and only if A is accepted by a computer within time $p(n)$, for some polynomial p.

It was purposely stated without reference to a specific model of 'machine' as it had already been observed that $\mathcal{L}^*$ was independent of that choice. Any class of computational models known at that time (e.g. random-access machines, Turing machines, Shepherdson-Sturgis machines, iterative arrays of finite-state machines), when substituted for the word 'computer' in the above definition, would give rise to the same class of decision problems. This has remained true until now, though many other classes of machine models have emerged in the meantime. In fact, a more general principle, justifying this stability, is believed to hold—a quantitative form of Church's Thesis [104]—stating that any model of computer algorithms can be simulated by a Turing machine with a polynomial loss in time complexity [22].

Let us recall one of the simplest standard definitions of P and FP based on a single-tape (deterministic) Turing machine with a binary alphabet. A single-tape Turing machine is a triple $\langle \textsc{states}, \mu, s_{\mathrm{in}}, s_{\mathrm{fin}} \rangle$, where:

- $\textsc{states}$ is a finite set,
- $\mu : \textsc{states} \times \{ 0, 1, \Box \} \longrightarrow \textsc{states} \times \{ 0, 1, \Box \} \times \{ L, R, S \}$ is the transition function,
- $s_{\mathrm{in}}, s_{\mathrm{fin}} \in S$ are the start and finish states respectively.

$\Box$ is the blank symbol and S, L, R describe the movements of the head of the machine. The actions of the machine are specified with respect to a tape represented by a triple $t = \langle L, h, R \rangle$, where $L = \Box^\infty l$, $R = r\Box^\infty$, $l, r \in \{ 0, 1, \Box \}^*$, $h \in \{ 0, 1, \Box \}$ and $\Box^\infty$ is the countably infinite string of $\Box$'s signifying the fact that only a finite part of the tape will ever be overwritten by the computation and the rest will always be blank. The character h is to denote the symbol which is being scanned by the machine at a given point in time.

The configuration of a machine consists of a state and the contents of the tape. The transition function can be used to modify a configuration $\langle s, \langle Lc_L, h, c_R R \rangle \rangle$ (for $c_L, c_R \in \{ 0, 1, \Box \}$) as follows: if $\mu(s, h) = \langle s', h', d \rangle$, the new configuration is

$$\begin{array}{ll} \langle s', \langle Lc_L, h', c_R R \rangle \rangle & \text{if } d = S, \\ \langle s', \langle L, c_L, h' c_R R \rangle \rangle & \text{if } d = L, \\ \langle s', \langle Lc_L h', c_R, R \rangle \rangle & \text{if } d = R. \end{array}$$

The initial configuration is $\langle s_{\mathrm{in}}, \langle \Box^\infty, \Box, i\Box^\infty \rangle \rangle$, where i is a binary string—an instance of the decision problem we would like to solve or an encoding of the input. Afterwards, the computation proceeds as indicated above, and as soon as the machine reaches a configuration with the final state s_{fin}, the computation is deemed to terminate. If we are interested in solving decision problems, the result is the symbol scanned once the machine has reached its final state. For this purpose we interpret 1 as YES and 0 as NO. In order to compute numeric functions we require the output—a string of 0's and 1's—to be placed to the right of the head. These conventions are immaterial to a large extent, but are indispensable in a precise definition.

Definition 2.1. Under the view that a decision problem is a set of binary strings, PTIME (P) is the class of decision problems which can be solved within a number of steps polynomial in the length of the (binary) input. FPTIME (FP) is the class of numeric functions that, interpreted as functions on binary strings, can be computed within the same bounds.

Many parameters in the above definition can be changed and it will still give rise to the same class of problems or functions. For example, even if the machine makes use of a richer alphabet throughout the computation, no new problems will be solved or functions computed in poly-time. We could also try to add more tapes and designate one of them to hold the input and another the output, but only a quadratic speed-up can then be expected and no quantitative advances are gained. Moreover, when the number of heads increases and at each step the transition depends on all the symbols scanned by the heads, we can simulate the new conditions by the simple single-head machines with a polynomial loss in complexity.

Only a non-deterministic variation on the machines is believed to lead beyond P (FP respectively). Then, at each step the computation is allowed to branch into several configurations, and once a single branch reaches an answer, it is treated as an answer to the whole computation. In addition, only the time used for that particular branch is counted. The problems solvable in polynomial time by non-deterministic Turing machines form the class (NPTIME (NP)). The question whether P is a proper subset of NP is still unanswered, but in what follows we will be able to see how it can reappear in various theories under different guises.

A number of characterizations of P based on space complexity are also available. By definition, for multi-tape Turing machines, the space occupied by the input and output tapes does not count towards the overall complexity and the two tapes are respectively read- and write-only. What is monitored is the number of cells of the other tapes (called work tapes) that are being used in computations. For example, LOGSPACE is the class of problems solvable in space bounded by $c \cdot \log(|i|)$, where c is a constant that does not depend on the input i. In the presence of non-determinism, the class of solvable problems is called NLOGSPACE. It is known that LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ PTIME and it is conjectured that the inclusions are all strict.

Logarithmic space can be related to PTIME by considering the notion of alternating computation—a generalization of non-determinism. A non-deterministic Turing machine in configuration c accepts a string if there is a next configuration leading to acceptance. A dual definition would say that all next configurations must be successful and that there is a next configuration. An alternating Turing machine combines the two standards of acceptance by dividing the set of states into the *existential* and the *universal*. A machine in an existential state accepts according to the 'non-deterministic' canon, but for universal states the stronger, universal property is needed. For such hybrid machines, we can also measure space complexity as before. The class of problems that take logarithmic space to solve in this setting is called ALOGSPACE. Chandra, Kozen and Stockmeyer [26] have proved that ALOGSPACE = PTIME. (They have also shown that APTIME = PSPACE, APSPACE = EXPTIME and AEXPTIME = EXPSPACE.)

The potential gap between LOGSPACE, NLOGSPACE and PTIME can also be accounted for by the addition of a special work tape to Turing machines. The new tape, called the pushdown tape, does not contribute to the monitored space consumption, but can be used only in a rather restrictive way:

- it is bounded at one end (e.g. on the left),
- initially its head (called the pushdown head) is positioned above the leftmost cell,
- at each step, as the other heads move, read and write (if possible) in the standard way, the pushdown head can only move to the right if it overwrites the previously scanned cell with a non-blank symbol, and if the head is to move to the left while scanning a cell with a non-blank symbol, it must erase the symbol (overwrite it with the blank symbol) before doing so.

Of course, the head cannot move left if its current position is the end of the tape. As shown by Cook [29], Turing machines running in LOGSPACE or NLOGSPACE enriched with such a new tape (and head) can solve exactly the PTIME problems.

The ability to make use of an additional unlimited pushdown tape allows a class of automata, known as the multi-head two-way automata [52], to capture PTIME. An automaton like this comprises a single input tape with one marker at each end of the input string. The n-heads (for $n \geqslant 1$) scan the input starting from the left end, then move left or right independently, entering only a finite number of states. The automaton has no writing facility and the computation terminates when one of the heads leaves the tape. The input string is accepted (i.e. it is a YES-instance of the problem) if the state at termination is some distinguished state called final. By adding a pushdown tape operating as before, we get a class of multi-head two-way pushdown automata. Cook [29] has shown that they solve precisely the problems in PTIME. Here, unlike for Turing machines, the number of heads *does* matter: the more heads there are, the more problems become solvable. Note that no resource bounds have been specified here, though this characterization is still based on machine models.

2.2 Logical characterizations

It is not obvious at all that complexity theory can be expressed in a machine-independent, resource-free way. The very existence of alternative definitions of complexity classes lends credence to their importance and suggests novel approaches to the whole theory. Implicit characterizations of complexity classes can potentially provide novel insights and serve to relate problems in complexity to issues relevant to other theories.

Decision problems are amenable to logical treatment. Intuitively, it seems plausible that properties that are harder to check will be harder to express. Indeed, this intuition is validated in a precise way in that restrictions placed on various syntactic parameters, e.g. the quantifier pattern, can be shown to correspond to standard complexity classes. For example, the well-known NP-complete problem 3-COLOURABILITY is expressible in second-order logic:

$$\begin{aligned}\exists R \exists G \exists B. \forall x \forall y.\quad & [(E(x,y) \Rightarrow \neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \\ & \wedge \neg(B(x) \wedge B(y))) \wedge (R(x) \vee G(x) \vee B(x))]\end{aligned}$$

The first clause states that no two adjacent vertices have the same colour, and the second says that every vertex has some colour. In this way, messy codings can be avoided. Here graphs are represented by pairs $\langle V, E, \leq \rangle$, where $E \subseteq V \times V$, and $\leq$ is a linear order on the vertices. In fact, any problem in NP can be defined by a formula of existential second-order logic (SO$\exists$):

$$\exists R_1 \cdots \exists R_n . \phi$$

where ϕ is a first-order formula. The converse is also true: any SO$\exists$ formula defines a property which can be verified in non-deterministic polynomial time. Consequently, it can be said that existential second-order logic characterizes precisely the problems in NP. This result by Fagin [37] initiated a new discipline, now called descriptive complexity theory [67].

In order to account for decision problems phrased in terms of strings, we should represent binary strings as finite models:

$$\langle\, S = \{\, 0, 1, \cdots, max \,\}, BIT \subseteq S, 0, \leq \,\rangle.$$

$BIT(i)$ holds iff the ith bit is to be set, and $\leq$ is the significance ordering on bits. For example, the numbers $2^n - 1$ $(n \geq 0)$ can be specified by the formula $\forall x. BIT(x)$.

What then is the logical characterization of polynomial-time computability and how is it different from that of NP? Surprisingly, it suffices to restrict the shape of ϕ to a (universally quantified) conjunction of Horn clauses (a result by Grädel [46]). Note that the formula used for 3-COLOURABILITY does not consist of Horn clauses. Otherwise, the results mentioned so far would imply a positive answer to the P=NP problem. The formula below is Horn and specifies the unreachability of a vertex y from x in a graph, a problem known to be in P:

$$\exists R \forall a \forall b \forall c. [R(a,a) \wedge (E(a,b) \Rightarrow R(a,b)) \wedge (R(a,b) \wedge R(b,c) \Rightarrow R(a,c)) \wedge \neg R(x,y)].$$

If we do not insist on having only existential second-order quantifiers at the front, we will still get yet another PTIME logic—SO-Horn.

Second-order definitions resemble inductive definitions formalized via least fixed points (LFP). However, the former are much weaker in that they only contain the fixed point: the clause above says that R contains the reachability relation of the graph and could not be used to define reachability itself. Anyway, the reachability of y from x is expressed by

$$R(x, y) = E(x, y) \vee \exists z.(R(x, z) \wedge R(z, y)).$$

It turns out that the properties expressible in first-order logic (FO) with least fixed point (FO+LFP) are exactly the problems in P [111, 114, 66]. To ensure that the least fixed points exist, one should stipulate that the inductively defined relational symbol occurs in positive positions inside definitions. This guarantees that the corresponding operator on relations will be monotone and the least fixed point can be computed by iteration. One can relax the monotonicity condition and use another kind of fixed point, called inflationary (IFP), which are calculated by taking the least fixed point of the monotone operator $\Phi(R) = R \cup \phi(R)$, where ϕ is the not necessarily monotone operator. As Gurevich and Shelah showed [49], FO+IFP=P.

There exist equally satisfying descriptions of decision problems and graph-theoretic properties in LOGSPACE, NLOGSPACE, PSPACE and many other complexity classes. Hence, they can be characterized by basic logical notions without any mention of computation. A notable feature of all these descriptions is the presence of an explicit ordering on the carrier of the model. This is considered undesirable and a great deal of research has been devoted to finding logics in which this element could be left out.

Analogous characterizations not only exist for finite models representing instances of problems, but they can also be carried over to a single infinite model—the set of all binary strings:

$$\langle \{0, 1\}^*, \mathsf{S}_0, \mathsf{S}_1, \leq \rangle$$

equipped with two successor functions $\mathsf{S}_i(x) = xi$ and the prefix order $\leq$. For this model, it is necessary to consider bounded versions of the previously mentioned logics, in which each quantification and fixed-point operation is guarded and consequently finitary. By a guard we mean a quantifier-free formula $G(\vec{x}, \vec{y})$ such that for any tuple $\vec{y}$ of carrier elements the set $\{\vec{x} \mid G(\vec{x}, \vec{y})\}$ is finite. It is required that each occurrence of a quantifier $Q \in \{\forall, \exists\}$ be of the form $(Q\,\vec{z} \in \{\vec{x} \mid G(\vec{x}, \vec{y})\}).F$ for some guard G. Additionally, the least fixed point computations may only involve formulas of the form $F \wedge G$, where G is a guard. These bounded versions of FO+LFP, SO-Horn and SO$\exists$-Horn have been proved to capture exactly PTIME [21].

Decision problems can be seen as database queries, which are modelled formally using finite model theory. Queries are then regarded as functions from finite models to finite models. In that context, the queries computable in polynomial time define the class QPTIME. The logical characterizations of PTIME extend easily to QPTIME, but the problem of specifying QPTIME queries can also be revisited in the typed lambda calculus.

Given a countable set of atomic constants representing the universe of the database, relations are encoded as lists of tuples using standard methods of representation. Queries correspond then to λ-terms that, when applied to encodings of input relations, reduce to the encoding of the output relation. Provided that there is a built-in equality test for the constants, it is possible to translate all QPTIME queries into the fragment of order 4. It turns out that all queries expressible in this restricted fragment can be handled in PTIME, so we obtain yet another way of describing polynomial-time computability [53].

2.3 Programming language characterizations

A related strand of research concerns recursive programs interpreted in finite models. Its results can be expressed in the setting of modern programming languages and used to associate complexity classes with various fragments of such languages. The first result of this kind by Gurevich [48] linked LOGSPACE with first-order primitive recursion.

Let us consider a typed functional programming language with the following type system:

$$T ::= Bool \quad | \quad [Bool] \quad | \quad T \times T \quad | \quad T \to T.$$

The order of a type is defined by

$$\begin{aligned} order(Bool) &= order([Bool]) = 0\,, \\ order(T_1 \times T_2) &= max(order(T_1), order(T_2))\,, \\ order(T_1 \to T_2) &= max(1 + order(T_1), order(T_2)). \end{aligned}$$

Programs consist simply of n function definitions:

$$P = \mathrm{def}_1; \cdots; \mathrm{def}_n$$

where $\mathrm{def}_1 : [Bool] \to Bool$ is the main function (representing a decision problem). Each function definition is of the form:

$$f x_1 \ldots x_n = expr$$

where *expr* is generated from typed variables, constants (True, False) and the empty list [ ] by function application, λ-abstraction, if-then-else, head, tail, pairing and projections. A notable feature is the lack of *cons*. *expr* can also contain occurrences of f to implement various kinds of recursion. f is said to be of order k, if its type is of that order. Programs have order k iff all functions defined inside them have order at most k.

Gurevich's result says that first-order *cons*-free programs with primitive recursion constructed according to the above rules will solve precisely the LOGSPACE problems. Unsurprisingly, an increase of the order of arguments leads to larger complexity classes. Second-order primitive recursive programs characterize PTIME as well as first-order general recursive programs [45, 70]. In the latter case, because general recursion is involved, we need to consider the programs that terminate for all inputs. The above results can also be formulated for imperative families of languages —the crucial element is again the absence of *cons* [69]. A suitable subset of PROLOG without function symbols (called DATALOG [67])) will do as well [103]: if P is to be captured, one is not allowed to negate the relations defined in a program, though negation of the built-in relation representing the input is in order.

It does not suffice to consider *cons*-free programs if we want to characterize FPTIME, because values of functions need to be constructed in some way. *cons* must therefore be added to the language, but this makes even primitive recursion too powerful. This brings us back to the original paper of Cobham [27]. Because primitive recursive functions need not be in P, one considers a bounded recursion scheme on the binary representation of a number.

Starting with the constant ϵ, successors $\mathsf{S}_0, \mathsf{S}_1$, projections and the *smash* function $\#$ ($x\#y = 2^{|x|\cdot|y|}$) let us generate new functions by composition and ***bounded recursion*** with bound h:

$$\begin{aligned} f(\epsilon, \vec{y}) &= g(\vec{x}), \\ f(\mathsf{S}_i(x), \vec{y}) &= h_i(x, \vec{y}, f(x, \vec{y})) \text{ for } i = 0, 1\,, \end{aligned}$$

where g, h, h_0, h_1 must have already been defined independently of f (using the same rules) and $f(x, \vec{y}) \leqslant_{\mathbb{N}} h(x, \vec{y})$ for all arguments $x, \vec{y}$. Cobham [27] proved that exactly the functions in FP are definable this way. The rather mysterious smash function is provided to give enough expressibility to initial definitions.

Cook [30] used Cobham's work to construct an equational theory PV (Polynomially Verifiable) in which the set of provably definable functions coincides with FP. A novelty of the logic is the proviso that the adequacy of a bound must be provable in PV itself. Later Buss came up with a system S_2^1 of [24] which was an extension of PV but still characterized FP. The induction scheme therein was restricted to formulas $(\exists\, x \leqslant t)A$, where only sharply bounded quantifiers $(\exists\, x \leqslant |t|)$ (i.e. on size rather than value) can occur in A. By counting alternations of bounded quantifiers (ignoring sharply bounded ones) it is also possible to obtain characterizations of other complexity classes from the Polynomial Time Hierarchy.

A related result by Leivant gives 'a foundational delineation' of FP [82]. He uses second-order logic instead in which the algebra of binary words can be defined from first principles. The equational programs computing FP functions are those whose convergence can be verified in constructive second-order logic with the comprehension (set-existence) scheme restricted to positive (without negation or implication) quantifier-free (or existential) formulas.

Finally, we mention a result due to Muchnik [94], who proves that bounded recursion is dispensable at the cost of a finite number of base functions i.e. FP has a finite basis with respect to composition. Unfortunately, this interesting fact does not give any clues as to how poly-time functions can be programmed.

The following example suggests some modifications to primitive recursion which do not force the programmer to look for explicit bounds. First we define a function *double* such that $double(x) = \underbrace{1 \cdots 1}_{2|x|}$:

$$\begin{array}{rcl} double([\,]) & = & [\,]\,, \\ double(i : x) & = & 1 : 1 : double(x)\,. \end{array}$$

Next we use *double* as a step function in

$$\begin{array}{rcl} f([\,]) & = & [1]\,, \\ f(i : x) & = & double(f(x)) \end{array}$$

to get a function f satisfying $|f(x)| = 2^{|x|}$. Thus, step functions cannot be allowed to process recursive calls via recursive arguments.

An abstract presentation of that idea relies on stratification (tiering) of inputs and outputs [81]. One introduces a countable number of levels, each of them containing a copy $\mathbb{N}_i$ of the set of natural numbers (binary strings) along with the successors $\mathsf{S}^0_k, \mathsf{S}^1_k : \mathbb{N}_k \to \mathbb{N}_k$ and empty strings $\epsilon_k : \mathbb{N}_k$ for $k \geqslant 0$. There are global projections for all combinations of levels, e.g.

$$\pi_2 : \mathbb{N}_4 \times \mathbb{N}_6 \times \mathbb{N}_2 \to \mathbb{N}_6\,.$$

Composition must respect the level labels and primitive recursion on binary notation follows the scheme:

$$\begin{array}{rcl} f(\epsilon, \vec{y}) & = & g(\vec{y})\,, \\ f(\mathsf{S}^i_k(x), \vec{y}) & = & h_i(x, \vec{y}, f(x, \vec{y}))\,, \end{array}$$

where $f : \mathbb{N}_k \times \vec{\mathbb{N}} \to \mathbb{N}_l$, $h : \mathbb{N}_k \times \vec{\mathbb{N}} \times \mathbb{N}_l \to \mathbb{N}_l$ and $k > l$. The final condition applies only if the recursive call is actually used in computation. For example, $double : \mathbb{N}_1 \to \mathbb{N}_0$ can be defined by taking $g = \epsilon_0$, $h = \mathsf{S}^1_0 \circ \mathsf{S}^1_0 \circ \pi_2 : \mathbb{N}_1 \times \mathbb{N}_0 \to \mathbb{N}_0$. Clearly, it cannot be used further in recursion because of the stratifying conventions. In return, each function $f : \mathbb{N}_m \to \mathbb{N}_n$ is polynomial-time computable and any FPTIME function can be defined as $f : \mathbb{N}_1 \to \mathbb{N}_0$ or indeed $f : \mathbb{N}_k \to \mathbb{N}_l$ for any k, l such that $k > l$.

A similar idea has independently appeared in Bellantoni and Cook's work [16] (motivated by a proof in [82]). They distinguish two kinds of arguments, called ***normal*** and ***safe*** respectively, and define functions $f(\vec{n}; \vec{s})$ in which the arguments are grouped in two zones. The safe ones can then be used to handle recursive calls, but recursive arguments must be normal. This results in the following ***safe recursion*** scheme:

$$\begin{aligned} f(\epsilon, \vec{n}; \vec{s}) &= g(\vec{n}, \vec{s})\,, \\ f(\mathsf{S}_i(x), \vec{n}; \vec{s}) &= h_i(x, \vec{n}; \vec{s}, f(x, \vec{n}; \vec{s})) \end{aligned}$$

resembling the previously considered stratified scheme. Composition prevents confusion between the two kinds of variables: safe variables cannot appear in normal positions. To begin, we decree that only normal variables can be substituted in normal slots:

$$f(\vec{n}; \vec{s}) = h(n_{i_1}, \ldots, n_{i_k}; f_{j_1}(\vec{n}; \vec{s}), \ldots, f_{j_l}(\vec{n}; \vec{s})).$$

The functions $f(\vec{n}; \vec{s})$ generated using these schemes and some basic functions (to be mentioned later) are polynomial-time computable. Moreover, they can be translated into the stratified system as functions

$$\mathbb{N}_1^{|\vec{n}|} \times \mathbb{N}_0^{|\vec{s}|} \to \mathbb{N}_0\,.$$

Conversely, any FPTIME function is expressible inside this system [50] as a function of a normal argument. One might wonder what flexibility is added by the higher levels of the stratification. The addition of the rest of levels allows for a more liberal composition scheme in which functions of normal arguments can be substituted in normal positions rather than just plain variables:

$$f(\vec{n}; \vec{s}) = h(g_{i_1}(\vec{n};), \ldots, g_{i_k}(\vec{n};); f_{j_1}(\vec{n}; \vec{s}), \ldots, f_{j_l}(\vec{n}; \vec{s})).$$

This is the original ***safe composition*** scheme of the Bellantoni and Cook's formalism BC, which consists of the safe composition and recursion schemes and the following base functions: $\epsilon(;)$, successors $\mathsf{S}_0(;), \mathsf{S}_1(;n)$ and the case construct:

$$\mathbf{cond}(;n, a, b, c) = \begin{cases} a & n = \epsilon\,, \\ b & n \text{ ends in } 0\,, \\ c & n \text{ ends in } 1\,. \end{cases}$$

BC definitions can be stratified as $\mathbb{N}_{i_1} \times \cdots \times \mathbb{N}_{i_k} \to \mathbb{N}_i$, where the safe arguments correspond to indices i_s that are equal to i.

The distinction between safe and normal inputs has inspired many researchers. Caseiro [25] investigated equational definitions on arbitrary data structures and generalized the notion of safe arguments to what she calls ***critical arguments***. When constructors of arity greater than one are used, the right-hand side of each definition must be linear in the critical arguments from the left-hand side. This prevents doubling of critical arguments and her system is called DDC (Don't Double Criticals) accordingly.

Hofmann [55] extended BC to a higher-order programming language SLR using a type system with a modality to forbid unsafe recursion by typability. Linear safe recursion for values of type

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

is also allowed in SLR and it turned out not to lead beyond FP. In another paper [56] Hofmann proposes a language in which one can write only non-size-increasing functions. The distinction between safe and normal input can be relaxed in such a framework and all definable functions are nonetheless poly-time computable. Although it is clear that most FP functions cannot be programmed in a non-size-increasing framework, it has been shown by the same author [58] that all those representing problems in P can be.

Returning to stratification, Marion [92] applied the methodology to termination proofs using multiset path orderings [35]. Programs that can be proved to terminate by his *light* ordering compute exactly the FP functions. Before that FP was also characterized as a class of first-order programs admitting termination proofs by an appropriate assignment of polynomials with positive coefficients to function symbols [23] (to be of any use for termination analysis, the polynomial interpretation must generate a assignment of polynomials to terms, which is monotone with respect to the reduction rules). The condition corresponding to poly-time computability stipulates that successors be interpreted by linear polynomials $X + c$ (by taking $aX + b$ instead one gets all functions in EXPTIME).

2.4 Lambda calculi

We have mentioned the simply typed lambda calculus as a tool to encode and examine database queries. However, there is a more traditional approach to expressibility in the lambda-calculus setting. Typically, one associates appropriate types T_D and T_C with respectively the domain and co-domain of the functions to be represented, and investigates all terms of type $T_D \to T_C$. For instance, for numeric functions, the natural choice is

$$T_D = T_C = (\alpha \to \alpha) \to (\alpha \to \alpha),$$

the type of the Church numerals. Unfortunately, typed terms with type $T_D \to T_C$ define only the extended polynomials (functions generated by 0 and 1 using addition, multiplication and conditional). Clearly, this is insufficient to capture all FPTIME functions and there are still many basic functions that are not computable by the typed terms, e.g. the predecessor.

A first step towards closing this gap is to change the representation of values and consider numbers in binary notation encoded as binary lists of type

$$L = (\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha).$$

Leivant and Marion added the free algebra of words $\{0, 1\}^*$ to the calculus (as the constant $\epsilon : \mathsf{o}$, constructors $0, 1 : \mathsf{o} \to \mathsf{o}$, the destructor and discriminator functions). Under more liberal input-output conventions (i.e. we no longer require the terms to be of type $L \to L$ but consider $L[\mathsf{o}^k/\alpha] \to \mathsf{o}$ or $L[\mathsf{o}^k/\alpha] \to L[\mathsf{o}^l/\alpha]$

instead), they were able to show that the set of expressible functions is precisely FP [83]. Many researchers took up the challenge of purifying this result and finding an intrinsically poly-time system in which reduction for all types would be of polynomial complexity. A perfect formalism of this kind ought not to involve external elements like superimposed words.

According to the proposition-as-types paradigm, standard lambda terms can be regarded as proof-terms for intuitionistic logic. Therefore, the programme of finding a calculus for P (or FP) can be restated equivalently as a quest for a suitable logic. Indeed, this direction has been most fruitful and resulted in two logics: bounded linear logic and light linear logic. Both are inspired by linear logic, which is itself a refinement of classical logic. Linear Logic makes it possible to control duplication of premises and so the sharing of data throughout computation. The duplicable objects always have type $!A$ for some A and are managed (in the intuitionistic fragment) through the four rules:

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A} \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}.$$

The first rule (called *promotion*) says that the inputs of an object to be certified as copiable must be duplicable too. The second (contraction rule) introduces the duplication itself, whereas the remaining two (dereliction and weakening) take account of the two special cases of copying zero times and annihilating the object. So, $!A$ really means an arbitrary number of copies of A including 0.

Bounded linear logic (BLL) [44] achieves the desired complexity by introducing explicit bounds on the use of inputs. They are special kinds of polynomials (called *resource polynomials*) added to the duplicable formulas ($!_{x<p}A$) or to type variables ($\alpha(p_1, \cdots, p_n)$). Although related to space complexity of the computation, they restrict the input-output behaviour in such a way that only polynomial-time computations become legal. The rules defining BLL are presented in Figure 2.1.

Binary lists of length at most n can be encoded as BLL proofs of

$$N_n = \forall\alpha.!_{y<n}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<n}(\alpha(y) \multimap \alpha(y+1)) \multimap (\alpha(0) \multimap \alpha(n)).$$

Proofs of $N_x \multimap N_{p(x)}$ (where p ranges over resource polynomials) represent exactly the FP functions. The computational mechanism that is used for poly-time evaluation is based on intuitions from the process of cut-elimination, but the irreducible forms are not necessarily cut-free (only certain cuts are eliminated). Therefore, there is still scope for improvement and light linear logic addresses this issue.

2.5 Light Affine Logic

Light linear logic (LLL) [42] does away with explicit bounds and is based on the simple observation that duplicable entities may not depend on more than one input.

$$
\begin{array}{llll}
\textbf{(var)} & A \vdash A' \text{ for } A \sqsubseteq A' & \textbf{(exch)} & \dfrac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C} \\[2ex]
\textbf{($\otimes$-l)} & \dfrac{A, B, \Gamma \vdash C}{A \otimes B, \Gamma \vdash C} & \textbf{($\otimes$-r)} & \dfrac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \\[2ex]
\textbf{($\multimap$-l)} & \dfrac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C} & \textbf{($\multimap$-r)} & \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \\[2ex]
\textbf{($\forall$-l)} & \dfrac{\Gamma, A[B/\alpha] \vdash C}{\Gamma, \forall\alpha.A \vdash C} & \textbf{($\forall$-r)} & \dfrac{\Gamma \vdash A}{\Gamma \vdash \forall\alpha.A} \quad \alpha \notin \mathrm{FV}(\Gamma) \\[2ex]
\textbf{(!-l)} & \dfrac{\Gamma, A[x := 0] \vdash B}{\Gamma, !_{x<1+w} A \vdash B} & & \\[2ex]
\textbf{(!-r)} & \multicolumn{3}{l}{\dfrac{\cdots, !_{z<q_i(x)}\, A_i[y := q_i(0) + \cdots + q_i(x-1) + z], \cdots \vdash B[x]}{\cdots, !_{y<q_i(0)+\cdots+q_i(p-1)+w}\, A_i[y], \cdots \vdash !_{x<p} B}} \\[2ex]
\textbf{(contr)} & \multicolumn{3}{l}{\dfrac{\Gamma, !_{x<p}\, A,\ !_{y<q}\, A[x := p + y] \vdash C \qquad x \text{ does not occur in } p}{\Gamma, !_{x<p+q+w}\, A \vdash C}} \\[2ex]
\textbf{(weak)} & \dfrac{\Gamma \vdash B}{\Gamma, !_{x<w} A \vdash B} & \textbf{(cut)} & \dfrac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}
\end{array}
$$

$\mathrm{FV}(\Gamma)$ is the set of free type variables in elements of Γ. $A \sqsubseteq A'$ if A and A' differ only in their choice of polynomials and for any polynomial p occurring positively (respectively negatively) in A, the corresponding p' in A' is such that $p \leq p'$ (respectively $p' \leq p$).

Figure 2.1: The rules defining valid BLL sequents

Otherwise, their nesting could lead to exponential explosion. Let us look at the following untyped term

$$(\boldsymbol{\lambda} x_3.(\boldsymbol{\lambda} x_2.(\boldsymbol{\lambda} x_1.x_1x_1)(x_2x_2))(x_3x_3))(x_4x_4)$$

and its β-normal form. Because at each step the number of occurrences of some variable doubles, the normal form (reachable in 3 steps) contains $2^{3+1} = 16$ occurrences of x_4. The same problem can be recast in the typed framework (application associates to the left by convention, so txy means $(tx)y$):

$$(\boldsymbol{\lambda} x_3.(\boldsymbol{\lambda} x_2.(\boldsymbol{\lambda} x_1.t_1x_1x_1)(t_2x_2x_2))(t_3x_3x_3))(t_4x_4x_4).$$

In linear logic we have control over the types of copiable items: for each i ($1 \leqslant i \leqslant 4$) $x_i :!T_i$ for some T_i, hence $t_i :!T_i \multimap (!T_i \multimap !T_{i-1})$ (for some type T_0) and then $t_ix_ix_i : T_{i-1}$. Cut-elimination in linear logic proceeds by waiting for $t_ix_ix_i$ to reach its canonical form (a box in the proof-net terminology) before duplication. Thus, if we are certain that the canonical form depends on one input, there is still a chance of avoiding an exponential blow-up. Fortunately, such a condition can be enforced syntactically via the **(!)** rule in Figure 2.2. Now, if $t_i x_i x_i$ is typable in the new framework, one of the variables will disappear during reduction (e.g. when t_i is a projection $\boldsymbol{\lambda} xy.x$). Afterwards, when a box depending on a single variable is copied k times, there will be k more occurrences of the variable rather than $2k$. Consequently, the size will grow polynomially, but not exponentially.

Nevertheless, this restriction on the use of ! is rather severe in absence of other opportunities for sharing. It is impossible to type the Church numerals uniformly in the $(\otimes, \multimap, !)$-fragment with the **(!)** rule and each formula will only have a finite number of proofs. A new connective § is introduced to strengthen the logic. Binary lists are then encoded as proofs of

$$\textsc{bin} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha).$$

The inference rules for § and all the other connectives are shown in Figure 2.2 together with associated proof-terms. Note that the weakening rule **(weak)** applies to all formulas, not only to those of shape $!A$ unlike in linear logic proper. The syntax extends the work of Koh and Ong [76] in which IMLL and MLL with units were handled. The logic we are concerned with is the intuitionistic fragment of Light Affine Logic (LAL) [11], which in turn is a significant simplification of light linear logic. By convention, the logic will be referred to as Second-Order Intuitionistic Multiplicative Light Affine Logic (IMLAL2)). Its sublogics have names dependent on the connectives they include and the presence of weakening. We show them in Figure 2.3. For instance, IMLLL stands for intuitionistic multiplicative light linear logic. When writing formulas we assume the standard precedence: $\forall, \S, !, \otimes, \multimap$.

There have already been quite a few attempts to design syntax for LAL [108, 110, 12]. Some constructs of those languages are simpler than ours, but they were to provide a simple programming notation rather than a tool to analyze the space of proofs. For instance, the presence of notational ambiguities in [110, 12] makes it possible to define the reduction procedure with just three rules!

2.5.1 Term calculus

The convention for the term language is that variables which appear in the denominator position (i.e. on the right of "-/-") are binders. Our aim here is simply to use the term language as a compact notation to express IMLAL2 derivations. We introduce some shorthand notation to interpret some familiar operations from the λ-calculus.

Remark 2.2. The conventional application of two suitably typed terms can be defined by

$$s : A \multimap B, t : A \vdash (\langle z^{A\multimap B}, t/x^B \rangle x)\{s / z^{A\multimap B}\} : B$$

$$\textbf{(ax)} \quad x : A \ \vdash \ x^A : A$$

$$\textbf{(exch)} \quad \frac{\Gamma, x : A, y : B, \Delta \ \vdash \ s : C}{\Gamma, y : B, x : A, \Delta \ \vdash \ s : C}$$

$$\textbf{(}\otimes\textbf{-l)} \quad \frac{\Gamma, x : A, y : B \ \vdash \ s : C}{\Gamma, z : A \otimes B \ \vdash \ \langle z^{A\otimes B} / x^A \otimes y^B \rangle s : C}$$

$$\textbf{(}\otimes\textbf{-r)} \quad \frac{\Gamma \ \vdash \ s : A \qquad \Delta \ \vdash \ t : B}{\Gamma, \Delta \ \vdash \ s \otimes t : A \otimes B}$$

$$\textbf{(}\multimap\textbf{-l)} \quad \frac{\Gamma \ \vdash \ s : A \qquad \Delta, y : B \ \vdash \ t : C}{\Gamma, z : A \multimap B, \Delta \ \vdash \ \langle z^{A\multimap B}, \, s/y^B \rangle t : C}$$

$$\textbf{(}\multimap\textbf{-r)} \quad \frac{\Gamma, x : A \ \vdash \ s : B}{\Gamma \ \vdash \ \boldsymbol{\lambda} x^A.s : A \multimap B}$$

$$\textbf{(!)} \quad \frac{x' : A \ \vdash \ s : B}{x : !A \ \vdash \ !\langle x/x' \rangle(s) : !B}$$

$$\textbf{(!}_0\textbf{)} \quad \frac{\vdash \ s : B}{\vdash \ !(s) : !B}$$

$$\textbf{(\S)} \quad \frac{x_1 : A_1, \cdots, x_k : A_k \ \vdash \ s : B \qquad \Box_i = \S, ! \quad 1 \leqslant i \leqslant k}{x'_1 : \Box_1 A_1, \cdots, x'_k : \Box_k A_k \ \vdash \ \S\langle x'_1/x_1, \cdots, x'_k/x_k \rangle(s) : \S B}$$

$$\textbf{(\S}_0\textbf{)} \quad \frac{\vdash \ s : B}{\vdash \ \S(s) : \S B}$$

$$\textbf{(}\forall\textbf{-l)} \quad \frac{\Gamma, y : A[B/\alpha] \ \vdash \ t : C}{\Gamma, z : \forall\alpha.A \ \vdash \ \langle z^{\forall\alpha.A}, \, B/y \rangle t : C}$$

$$\textbf{(}\forall\textbf{-r)} \quad \frac{\Gamma \ \vdash \ s : A}{\Gamma \ \vdash \ \boldsymbol{\Lambda}\alpha.s : \forall\alpha.A} \qquad \alpha \notin \mathrm{FV}(\Gamma)$$

$$\textbf{(contr)} \quad \frac{x_1 : !A, x_2 : !A, \Gamma \ \vdash \ s : B}{x_1 : !A, \Gamma \ \vdash \ s[x_1/x_2] : B}$$

$$\textbf{(weak)} \quad \frac{\Gamma \ \vdash \ s : A}{\Gamma, y : B \ \vdash \ s : A}$$

$$\textbf{(cut)} \quad \frac{\Gamma, x : A \ \vdash \ s : B \qquad \Delta \ \vdash \ t : A}{\Gamma, \Delta \ \vdash \ s\{t/x^A\} : B}$$

N.B. $s[x_1/x_2]$ denotes meta-substitution.

Figure 2.2: Rules that define valid IMLAL2 sequents.

NAME	CONNECTIVES	WEAKENING FOR ARBITRARY FORMULAS
IMLAL2	$\otimes, \multimap, \S, !, \forall$	Yes
IMLLL2	ditto	No
IMAL2	$\otimes, \multimap, \forall$	Yes
IMLL2	ditto	No
IMLAL	$\otimes, \multimap, \S, !$	Yes
IMLLL	ditto	No
IMAL	$\otimes, \multimap$	Yes
IMLL	ditto	No

Figure 2.3: IMLAL2 and its sublogics.

and we write st to mean the right-hand side of the sequent. Often we do not differentiate between various parsings of the tensor and feel free to use

$$\langle z\,/z_1 \otimes \cdots \otimes z_n\,\rangle(t)$$

for any possible way of disambiguating z's type and extracting its components. Besides, $\langle s\,/z_1 \otimes \cdots \otimes z_n\,\rangle(t)$ will serve as an abbreviation of

$$(\langle z\,/z_1 \otimes \cdots \otimes z_n\,\rangle(t))\{s\,/z\}.$$

We replace $(\langle z^{\forall\alpha.A},\ B/x\,\rangle x)\{t\,/z\}$ with the standard type application $t[B]$ and write $\boldsymbol{\lambda}(z_1 \otimes z_2).t$ for $\boldsymbol{\lambda}z.\langle z\,/z_1 \otimes z_2\,\rangle t$. Further, $!\langle\, x'/x\,\rangle(s)\{t\,/x'\}$ is abbreviated to $!\langle\, t/x\,\rangle(s)$ and similarly for $\S\langle\, \overline{z'}/\overline{z}\,\rangle(s)\{t\,/z'\}$. For instance, $\S\langle\, z_1'/z_1, t/z_2, z_3'/z_3\,\rangle(s)$ stands for $\S\langle\, z_1'/z_1, z_2'/z_2, z_3'/z_3\,\rangle(s)\{t\,/z_2'\}$.

In $!\langle\text{-}/\text{-}\rangle(\text{-})$ and $\S\langle\text{-}\rangle(\text{-})$ terms, the $\langle\text{-}\rangle$ part is called the ***interface*** of the term and is intended to represent a set of variable pairs i.e. $\S\langle\, a'/a, b'/b\,\rangle(\text{-}) = \S\langle\, b'/b, a'/a\,\rangle(\text{-})$. We omit the brackets for empty interfaces.

Binary strings (and so natural numbers via their binary representation) are represented in LAL as closed terms of type

$$\text{BIN} \;=\; \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha).$$

For instance, the empty string ϵ is represented by $\boldsymbol{\Lambda}\alpha.\boldsymbol{\lambda}f_0 f_1.\S(\boldsymbol{\lambda}x.x)$ (also called ϵ) and 100 by

$$\boldsymbol{\Lambda}\alpha.\boldsymbol{\lambda}f_0^{!(\alpha\multimap\alpha)} f_1^{!(\alpha\multimap\alpha)}.\S\langle\, f_0/f_0^0, f_0/f_0^1, f_1/f_1^2\,\rangle(\boldsymbol{\lambda}x.f_0^0(f_0^1(f_1^2 x)))).$$

Note that the characters (bits) are stored in reverse order. Thanks to this, recursion and iteration are easier to encode, as it suffices to activate the pattern of the term. The two successors S_i $(i = 0, 1)$ are defined as

$$n : \text{ BIN } \vdash \boldsymbol{\Lambda}\alpha.\boldsymbol{\lambda} f_0 f_1.\S\langle\, n[\alpha] f_0 f_1 / n', f_i / f_i' \,\rangle(\boldsymbol{\lambda} x. f_i'(n'x)) : \text{ BIN}.$$

Remark 2.3. Our definition of S_0 signifies that we focus on implementing the algebra of binary strings instead of 'true' natural numbers. (For the latter we would have to make sure that $\mathsf{S}_0(\epsilon) = \epsilon$ and not $\mathsf{S}_0(\epsilon) = 0$. This additional trouble has been taken in [96].) The difference is that binary strings provide ambiguous representations of natural numbers, when the standard binary encoding is used. For instance, 00100 and 0100 represent the same number but, as binary strings, they induce different courses of iteration. The shortest encoding of a number can always be obtained by erasing all leading zeros and may be regarded as canonical for the purpose of various encodings. The normalization, which would have to follow each operation then, is rather cumbersome and we prefer to omit it. It should be stressed though that this does not affect the class of representable functions, but rather the way they are defined.

2.5.2 Commuting conversions

The sequent calculus conserves the order in which various rules are applied. Sometimes it is possible to apply two rules and any order of their application will result in the derivation of the same sequent. Besides, the preservation of the order prevents the calculus from acquiring good mathematical properties and complicates proofs of cut-elimination. A solution to this problem, in the form of proof-nets, has already been proposed at the very start of linear logic [39], but it fulfils its role ideally only for the multiplicative fragment of linear logic (the treatment of exponentials, let alone the units, is still problematic). We revisit the problems and try alleviate them to some extent in one of the chapters to come, where we focus on extending essential nets [80] to handle IMLAL2.

All the deficiencies of sequent calculus are unfortunately 'inherited' by the term calculus and we have to impose equivalences between proofs explicitly.

Let π range over the let-constructs, namely

$$\langle z^{A\otimes B} / x \otimes y \,\rangle(\text{-}), \quad \langle z^{A\multimap B}, s/y \,\rangle(\text{-}) \quad \text{and} \quad \langle z^{\forall\alpha.A}, B/y \,\rangle(\text{-}),$$

and σ over the explicit substitution constructs

$$(\text{-})\{t / x^A\}$$

(so that π is viewed as a prefix and σ a postfix operator). We let C range over one-holed contexts, or simply *contexts*, defined by recursion as follows:

$$C \ ::= \ [\text{-}] \mid s\{C\,/x^A\} \mid C\{s\,/x^A\} \mid s \otimes C \mid C \otimes s \mid \boldsymbol{\lambda} x^A.C \mid \boldsymbol{\Lambda}\alpha.C \mid$$

$$\langle z^{A\otimes B}\,/x\otimes y\rangle C \mid \langle z^{A\multimap B}\,,\, C/y^B\rangle s \mid \langle z^{A\multimap B}\,,\, s/y^B\rangle C \mid \langle z^{\forall\alpha.A}\,,\, B/y\rangle C,$$

where s is an arbitrary term. We write $C[t]$ to mean 'the capture-permitting substitution of t for the hole in C'.

Now, the congruence $\sim$ characterizing terms that are the same up to commuting conversions is the reflexive, symmetric, transitive and contextual closure of:

$$(\pi\textbf{-cong}) \quad \Gamma \vdash \pi C[t] \sim C[\pi t] : A,$$

$$(\sigma\textbf{-cong}) \quad \Gamma \vdash C[t]\sigma \sim C[t\sigma] : A.$$

Each congruence axiom above is required to satisfy a side condition, called *strong typability*, which ensures that the expressions on both sides of $\sim$ must be well-typed terms of the same declared type. The effect of the axioms is simply that the let-construct π- or the explicit substitution construct -σ, viewed as a variable binder, may 'float across the term' and is free to occupy any position in the term *provided* that typability is maintained.

For example, the binder $(\text{-})\{s\,/x^A\}$ in $(\boldsymbol{\lambda} y^B.x \otimes y)\{s\,/x^A\}$ is permitted to park itself adjacent to x, as in $\boldsymbol{\lambda} y^B.(x\{s\,/x^A\} \otimes y)$ (as the two terms are both well-typed, they are congruent to each other by (σ**-cong**), but not adjacent to y, as in $\boldsymbol{\lambda} y^B.(x \otimes (y\{s\,/x^A\}))$ which is not well-typed.

In the rewriting system, defined in Figure 2.4, the reduction will concern $\pi\sigma$-equivalence classes of terms rather than terms. The rules simply implement the process of cut-elimination in the sequent calculus [42] like the systems presented in [11, 110, 12, 113][1].

2.5.3 Reduction

Observe that replication will usually have to take place before the execution of the first two exponential rules. Thus it makes sense to introduce a stratified reduction strategy that will perform replication before its results are needed.

By the ***depth of a formula*** (type), we mean the maximum nesting depth of ! and § that occur in it; e.g.

$$\S(a \multimap !b), \quad a \multimap b, \quad \S(\forall\alpha.(\alpha \multimap !\S\alpha))$$

[1]Note that the contexts in which reduction can take place are not the same as these for $\pi\sigma$-congruence and in contrast reduction inside boxes is allowed.

β-redexes

(rename-β)	$s\{y^A\,/x^A\} \to s[y/x]$
(var-β)	$z^A\{s\,/z^A\} \to s$
($\otimes$-β)	$(\langle z^{A\otimes B}\,/x\otimes y\,\rangle s)\{u\otimes v\,/z\} \to (s\{u\,/x^A\})\{v\,/y^B\}$
($\multimap$-β)	$(\langle z^{A\multimap B}\,,\,s/y^B\,\rangle t)\{\boldsymbol{\lambda} x^A.u\,/z\} \to t\{u\{s\,/x^A\}\,/y^B\}$
($\forall$-β)	$(\langle z^{\forall\alpha.A}\,,\,B/y\,\rangle s)\{\boldsymbol{\Lambda}\alpha.t\,/z\} \to s\{t[B/\alpha]\,/y\}$

Exponential redexes

(!-!)	$!\langle\, x/x_0\,\rangle(s)\{!\langle\, y/y_0\,\rangle(t)\,/x\} \to\, !\langle\, y/y_0\,\rangle(s\{t\,/x_0\})$
(§-!)	$\S\langle\,\cdots, x/x_0, \cdots\rangle(s)\{!\langle\, y/y_0\,\rangle(t)\,/x\} \to \S\langle\,\cdots, y/y_0, \cdots\rangle(s\{t\,/x_0\})$
(§-§)	$\S\langle\,\cdots, x/x_0, \cdots\rangle(s)\{\S\langle\,\overline{y/y_0}\,\rangle(t)\,/x\} \to \S\langle\,\cdots, \overline{y/y_0}, \cdots\rangle(s\{t\,/x_0\})$

Replication redexes

Suppose there are $n \geqslant 2$ free occurrences of z in s.

$$s\{!\langle\, x/x_0\,\rangle(t)\,/z\} \to s'\{!\langle\, x/x_0\,\rangle(t)\,/z_1\}\cdots\{!\langle\, x/x_0\,\rangle(t)\,/z_n\}$$

where s' is obtained from s by distinguishing the n occurrences of z as $z_1, \cdots, z_n$ respectively.

Garbage collection (provided x, y do not occur in s)

(var-γ)	$s\{t\,/y^A\} \to s$	($\otimes$-γ)	$\langle z^{A\otimes B}\,/x\otimes y\,\rangle s \to s$
($\multimap$-γ)	$\langle z^{A\multimap B}\,,\,t/y^B\,\rangle s \to s$	($\forall$-γ)	$\langle z^{\forall\alpha.A}\,,\,B/y\,\rangle s \to s$
(!-γ)	$!\langle\, x'/x\,\rangle(s) \to !(s)$	(§-γ)	$\S\langle\,\cdots, y'/y, \cdots\rangle(s) \to \S\langle\,\cdots,\cdots\rangle(s)$

Figure 2.4: Rewriting rules for IMLAL2 proofs.

have depths 2,0 and 3 respectively. We will say that a redex is at depth i, if it is inside i boxes (by a box we mean terms of the shape $\Box\langle\text{-}\rangle(\text{-})$ for $\Box = \S, !$). Then in round i ($i \geqslant 0$) one reduces all exponential redexes at depth $i-1$ and all non-exponential redexes at depth i. In this way, round i enables round $i+1$ to be implemented eagerly. Another nice feature is that round i can only contribute new redexes at depth i. These facts facilitate the estimation of changes in the size of terms (without type annotations) during reduction and lead to the following result:

Theorem 2.4 (Girard, Asperti). The stratified reduction strategy applied to a term $t : T$ terminates after $O(|t|^{2^{d+2}})$ steps, where d is the depth of T.

which was recently strengthened in [113] to:

Theorem 2.5 (Terui). Any reduction path of a term $t : T$ has $O(|t|^{2^{d+2}})$ steps.

Hence, locally (for types of fixed depth), cut-elimination is a poly-step procedure. We should mention that Girard's result [42] concerns classical light linear logic (with additives). He proves that if T does not contain any occurrences of additive connectives, then any proof of T (possibly using the additive rules) can be brought to a cut-free form by a lazy procedure with the above bound on the number of steps.

The Theorems above rely on the fact that the size of terms during reduction will be bounded by a polynomial, but 'size' refers to the underlying untyped terms. When type annotations are present, they may still become exponentially large! This is because all the restrictions concern duplication and sharing of term variables and not that of type variables. In type applications one might still copy type variables and, for example, substitute $\alpha \multimap \alpha$ for α.

Example 2.6. [2] Let $T_0 = \alpha$ and $T_{n+1} = T_n \multimap T_n$ for $n \geqslant 0$. Clearly, the size of T_i is exponential in i.

We are going to construct a term $t : \textsc{bin} \multimap \S T$ such that, for $u : \textsc{bin}$, tu is the identity term for $T_{|u|}$ (here $|u|$ is the length of the string corresponding to u), and therefore represents a function that is not in FP, if our rewriting system with type annotations is to be used.

Let T be the existential type for encoding terms with types of form $\beta \multimap \beta$ [43]:

$$T = \forall\alpha.((\forall\beta.((\beta \multimap \beta) \multimap \gamma)) \multimap \gamma).$$

Then the identity term for T_n ($n \geqslant 0$) is represented by

$$\mathbf{id}_n = \mathbf{\Lambda}\alpha.\boldsymbol{\lambda} v^{\forall\beta.((\beta\multimap\beta)\multimap\gamma)}.v[T_n](\boldsymbol{\lambda} x^{T_n}.x).$$

[2] The possibility of constructing this example in IMLAL2 was suggested by Kazushige Terui.

It is possible to convert this representation of the identity on T_n to one for T_{n+1} using

$$\mathbf{next} = \boldsymbol{\lambda} t^T.\boldsymbol{\Lambda}\alpha.t[\alpha \multimap \alpha] : T \multimap T.$$

Then

$$\boldsymbol{\lambda} s^{\text{ BIN}}.\S\langle\, s[T]\,!(\mathbf{next})\,!(\mathbf{next})/s'\,\rangle(s'\,\mathbf{id}_0) : \ \textsc{bin} \multimap \S T$$

applied to some u : BIN reduces to $\S(\mathbf{id}_{T_{|u|}})$.

This example shows that cut-elimination in IMLAL2 (or even IMAL2) is *not* exactly a (locally) polynomial-time algorithm. It is, however, in its untyped form, because it is then both poly-step and poly-space: the potentially exponential space to store type labels is no longer needed. In many cases, untyped cut-elimination suffices to reveal the full (typed) cut-free form, namely, when there is no need for existential quantification in the final cut-free term i.e. provided the $\forall$ quantifier never occurs in the negative position of the result type (like in BIN, but not T). The above example also demonstrates that if η-expansion was admitted into the system (like in the original proposal [76]), the poly-space bounds would be violated. To make up for the exclusion and preserve the property of cut-elimination, the rule (rename-β) has to be added as well as (var-β) (for all types rather than just base types like in [76]). The two rules handle the case of an axiom link being 'cut' with a node that need not be an axiom link (or the other way round). If we had η-expansion (see Figure 2.5), the axiom link in question could be decomposed into another axiom link (or links) between smaller formulas, and at the end only axiom-links for base types would interact:

$$(\text{atom-}\beta) \qquad z^b\{s\,/z^b\} \to s \quad (b \text{ is a base type (atom or variable)}).$$

When this 'decompression' is not available, one can still eliminate the cut as detailed in the (var-β) and (rename-β) rules. It is worth stressing that even if η-rules are allowed, the set of definable functions remains the same.

Theorem 2.7. Terms of type BIN $\multimap \S^n$ BIN ($n \in \mathbb{N}$) correspond exactly to FP functions.

The first complete account of how poly-time Turing machines can be encoded in LAL was given in [109]. Our Chapter 3 provides an alternative.

In conclusion, LLL and IMLAL2 characterize FP not only extensionally, but also dynamically, provided we do not insist on η-expansion and introduce the lazy rules ((rename-β) and (var-β)).

$$
\begin{array}{ll}
(\otimes\text{-}\eta) & z^{A\otimes B} \to \langle z \,/ x^A \otimes y^B \,\rangle x \otimes y \\
(\multimap\text{-}\eta) & z^{A\multimap B} \to \boldsymbol{\lambda} x^A.\langle z \,,\, x^A / y^B \,\rangle y \\
(!\text{-}\eta) & z^{!A} \to\, !\langle\, z / z_0 \,\rangle (z_0) \\
(\S\text{-}\eta) & z^{\S A} \to \S\langle\, z / z_0 \,\rangle (z_0) \\
(\forall\text{-}\eta) & z^{\forall\alpha.A} \to \boldsymbol{\Lambda}\alpha.\langle z \,,\, \alpha / y^A \,\rangle y
\end{array}
$$

Figure 2.5: η-redexes.

2.6 ELL and SLL

By allowing contexts with multiple formulas on the left-hand side of the **(!)** rule, we obtain yet another logical system capturing Kalmar elementary complexity [107]. In our terminology it should be called IMEAL2 (without additives), because we then get the intuitionistic affine variant of Elementary Linear Logic (ELL).

The principles underlying ELL were first sketched in the paper introducing LLL [42] and developed in more detail in [105]. A different formulation based on characterizing the elementary derivations among the intuitionistic ones was presented in [33]. The latter also addresses a problem with the combination of the additives and the units in the previous framework.

It is not difficult to modify our syntax so that it handles cut-elimination in ELL. Similarly, the game model for IMLAL2 that we present in one of the following chapters cuts down to a model of ELL, when the notion of suitably networked strategies is tuned to the ELL **(!)** rules.

The latest development, also succeeding in capturing polynomial-time computability is Lafont's Soft Linear Logic (SLL) [77]. Recall the four intuitionistic rules regarding formulas of the shape $!A$. They can be replaced with an alternative (countably infinite) set of rules:

$$
\frac{\Gamma \;\vdash\; A}{!\Gamma \;\vdash\; !A}
\qquad
\frac{\Gamma, !!A \;\vdash\; B}{\Gamma, !A \;\vdash\; B}
\qquad
\frac{\Gamma, \overbrace{A, \cdots, A}^{n} \;\vdash\; B}{\Gamma, !A \;\vdash\; B}\; n \geqslant 0
$$

SLL2 is then obtained by excluding the middle rule (called *digging*). Several observations are already apparent. In ILL, $!!A$ and $!A$ are provably equivalent and therefore the nesting of occurrences of $!$ is inessential. By striking the digging rule out we can make this information meaningful. In ILL $!A \multimap !A \otimes !A$ is provable and witnesses the duplicable nature of $!A$. However, a duplicated copy of $!A$ retains this character and can be cloned again. Having made a copy once, we do not have any control over the number of future copies. In SLL2, $!A \multimap !A \otimes !A$ cannot be proved, though we have $!A \multimap \underbrace{A \otimes \cdots \otimes A}_{n}$ $(n \geqslant 0)$ instead. Consequently, the nesting of $!$'s in a formula like $\underbrace{! \cdots !}_{k} A$ allows one for k-fold nested duplication, which yields an elementary proof of poly-time cut-elimination (where the degree of nesting is fixed). The representability of polynomial-time computations is very subtle in this system and the original proof only accounts for P. The natural 'soft' variant of our type for binary lists

$$\forall \alpha . !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha),$$

cannot be used to encode numeric functions, because even the successor functions would not be definable. However, there are still proofs interpreting binary strings, which are used to represent the input.

Chapter 3

Programming with IMLAL2

Although it has been proved that any FP function is represented in IMLAL2 by a term with appropriate type [109], it is not clear what algorithmic capabilities IMLAL2 offers. The type used to encode Turing machines is far from anything one would use in modern programming. In this chapter we take a more systematic approach and try to identify programming constructs which are both interpretable in IMLAL2 and expressible enough to enable us to represent poly-time Turing machines. Our analysis reveals some connexions with safe recursion [16, 81] on the one hand and with non-size-increasing computation [56] on the other.

3.1 Iteration

Many numeric functions can be translated into IMLAL2 with the help of an ***iteration principle***. Suppose we have

$$\begin{array}{rcl} [f' : A] & \vdash & b : \mathbb{I}\,, \\ [f'_0 : A_0] & \vdash & g_0 : \mathbb{I} \multimap \mathbb{I}\,, \\ [f'_1 : A_1] & \vdash & g_1 : \mathbb{I} \multimap \mathbb{I}\,, \end{array}$$

and $\Box \in \{!, \S\}$ ($[\cdots]$ denotes an optional part). The iterative application of the step functions g_0 and g_1 to the seed b, following the pattern of the term $s : \textsc{bin}$, is represented by

$$\begin{array}{l} s : \textsc{bin},\ [f : \Box A],\ f_0 : !A_0,\ f_1 : !A_1 \vdash \\ \qquad \S\langle\, s[\mathbb{I}]\, !\langle\, [f_0/f'_0]\,\rangle(g_0)\, !\langle\, [f_1/f'_1]\,\rangle(g_1)/h,\ \Box\langle\, [f/f']\,\rangle(b)/x\,\rangle(hx) : \S\mathbb{I}\ . \end{array}$$

Note that each time the iteration principle is invoked, the result type is 'lifted' by a $\S$.

Example 3.1. We can apply the iteration principle to show that $\textsc{bin}$ is embeddable in $\S\Box^k\ \textsc{bin}$, where $\Box^k$ is any combination of ! and $\S$ with length k ($k \geqslant 0$). To that end we use $\vdash \Box^k\epsilon : \Box^k$ BIN as the seed and and $\vdash \Box^k\mathsf{S}_i : \Box^k$ BIN $\multimap \Box^k$ BIN as the step functions (the terms are constructed analogously to $\epsilon, \mathsf{S}_0, \mathsf{S}_1 :$ BIN). By an appeal to the iteration principle for $\mathbb{I} = \Box^k$ BIN we get a closed term:

$$\mathbf{coerce}_{\S\Box^k} \ : \ \text{BIN} \multimap \S\Box^k \ \text{BIN}\,.$$

Such terms are called ***coercions*** in [42].

As a more complicated illustration of the principle we define a term **strip** : BIN $\multimap$ BIN whose effect on a string is iterated erasure of the leftmost symbol as long as it is 0:

$$\begin{aligned} \mathbf{strip}(0000) &= \epsilon\,, \\ \mathbf{strip}(00100) &= 100\,. \end{aligned}$$

The term plays an important role if one cares about working with natural numbers instead of binary strings: it can be used to convert binary representations of natural numbers to the canonical form: the empty string or a string beginning with 1 [96].

Lemma 3.2. **strip** : BIN $\multimap$ BIN is definable in IMLAL2.

Proof. Let us use iteration for $\mathbb{I} = P \otimes (\alpha \multimap \alpha)$, where $P = \forall\alpha.\alpha \otimes \alpha \multimap \alpha$. Let $\pi_i = \mathbf{\Lambda}\alpha.\boldsymbol{\lambda}(x_1 \otimes x_2).x_i : P$ for $i = 0, 1$. The first component of the iterated pair will be π_0 as long as the iteration should proceed according to the offending zeros. Once the first 1 is encountered, it will be set to π_1. From then on the binary symbols may be appended to the result. Depending on the projection, different actions will be taken. When processing 0's, the projection from the previous stage will be copied and the current bit ignored. In contrast, 1's will fix the projection at π_1 and add the current bit to the aggregate. Moreover, the projections will always be used to select the prescribed actions. Note that this means that we have to duplicate the projection when processing 0's. With this aim we exploit their polymorphic type and set $\mathbf{dup}\,p = p[P \otimes P]((\pi_0 \otimes \pi_0) \otimes (\pi_1 \otimes \pi_1))$. The step functions are then:

$$f_0 : (\alpha \multimap \alpha) \ \vdash \ \boldsymbol{\lambda}(p \otimes c).\langle \mathbf{dup}\,p\,/p_1 \otimes p_2\,\rangle(p_1 \otimes (p_2[G](g_0 \otimes g_1))(f_0 \otimes c))$$

and

$$f_1 : (\alpha \multimap \alpha) \ \vdash \ \boldsymbol{\lambda}(p \otimes c).\pi_1 \otimes ((p[G](g_0 \otimes g_1))(f_1 \otimes c)),$$

where $g_i : G$ $(i = 0, 1)$ are functions that manipulate f_i and c so that the bit f_i is appended to c when the current projection is π_1 and ignored in the other case. Hence $G = (\alpha \multimap \alpha)^2 \multimap (\alpha \multimap \alpha)$,

$$g_0 = \boldsymbol{\lambda}(f^{(\alpha\multimap\alpha)} \otimes c^{(\alpha\multimap\alpha)}).c$$

and

$$g_1 = \boldsymbol{\lambda}(f^{(\alpha\multimap\alpha)} \otimes c^{(\alpha\multimap\alpha)}).\boldsymbol{\lambda}x^{\alpha}.f(cx)\,.$$

The iteration principle with $b = \pi_0 \otimes I_\alpha$ gives a proof of:

$$\text{BIN}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)$$

after we project the result onto $(\alpha \multimap \alpha)$. By lambda and universal abstractions we arrive at the term **strip**. □

3.2 Safe and normal variables in IMLAL2

The first step in understanding safe recursion in IMLAL2 is to type safe and normal arguments. We have already seen that the type constructor § is a mark of iteration. Since safe arguments are the (only) places where recursive calls may occur, they should have the same result type as that of the recursive function being defined.

Definition 3.3. Suppose the sequent

$$x_1 : \ \text{BIN}, \cdots, x_m : \ \text{BIN}, y_1 : \S^k \ \text{BIN}, \cdots, y_n : \S^k \ \text{BIN} \ \vdash \ t : \S^k \ \text{BIN}$$

is provable. If $k \geqslant 1$, we say that the variables x_i $(1 \leqslant i \leqslant m)$ are ***normal*** and y_j $(1 \leqslant j \leqslant n)$ are ***safe*** in t; if $k = 0$ we say that all variables are safe. To save writing, we shall write $t(x_1, \cdots, x_m : y_1, \cdots, y_n \mid k)$ as shorthand for the sequent. In an obvious manner, the sequent represents an $m+n$-unary function on binary strings.

Safe variables can always be made normal: if $t(\vec{x} : \vec{u}, y, \vec{v} \mid k)$ is provable then there exists some t' such that $t'(\vec{x}, y : \vec{u}, \vec{v} \mid k)$ is provable and t' represents the same numeric function (it suffices to use the cut rule with $\mathbf{coerce}_{\S^k}$ for the appropriate formula on the left). We state two further useful principles:

Proposition 3.4. (i) **(Lifting Principle)** If $t(\vec{x} : \vec{y} \mid k)$ is provable, then for each $l > k$ there exists t' such that $t'(\vec{x} : \vec{y} \mid l)$ is provable and represents the same numeric function.

(ii) **(Normal contractibility)** If $t(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \cdots, x_m : \vec{y} \mid k)$ represents $f : \mathbb{N}^m \times \mathbb{N}^n \to \mathbb{N}$ then there exists $t'(x_1, \cdots, x_{i-1}, x_i, x_{i+2}, \cdots, x_m : \vec{y} \mid k+2)$ representing $f' : \mathbb{N}^{m-1} \times \mathbb{N}^n \to \mathbb{N}$ such that

$$f'(x_1, \cdots, x_{i-1}, x_i, x_{i+2} \cdots, x_m : \vec{y}) \ = \ f(x_1, \cdots, x_{i-1}, x_i, x_i, x_{i+2}, \cdots, x_m : \vec{y})\,.$$

That is to say, normal variables in IMLAL2 may be contracted.

Proof. For (i), by assumption, we have a derivation of

$$(\ \mathrm{BIN})^m, (\S^k\ \mathrm{BIN})^n \ \vdash\ \S^k\ \mathrm{BIN}\,,$$

to which we apply the §-rule $(l-k)$ times to get a derivation of

$$(\S^{l-k}\ \mathrm{BIN})^m, (\S^{l-k}(\S^k\ \mathrm{BIN}))^n \ \vdash\ \S^{l-k}(\S^k\ \mathrm{BIN})\,.$$

Finally, we invoke the cut rule with $(\mathbf{coerce}_{\S^{l-k}})^m \otimes (I_{\S^l\ \mathrm{BIN}})^n$ to end up with the expected proof of $(\ \mathrm{BIN})^m, (\S^l\ \mathrm{BIN})^n \ \vdash\ \S^l\ \mathrm{BIN}$.

For (ii), we start from the given proof of $(\ \mathrm{BIN})^m, (\S^k\ \mathrm{BIN})^n \ \vdash\ \S^k\ \mathrm{BIN}$ and use the §-rule with !'s for normal arguments to get $(!\ \mathrm{BIN})^m, (\S\S^k\ \mathrm{BIN})^n \ \vdash\ \S\S^k\ \mathrm{BIN}$. Next we perform contraction for appropriate copies of $!\ \mathrm{BIN}$, which results in

$$(!\ \mathrm{BIN})^{m-1}, (\S\S^k\ \mathrm{BIN})^n \ \vdash\ \S\S^k\ \mathrm{BIN}\,,$$

and apply the §-rule, this time using only §'s, to obtain

$$(\S!\ \mathrm{BIN})^{m-1}, (\S^{k+2}\ \mathrm{BIN})^n \ \vdash\ \S^{k+2}\ \mathrm{BIN}\,.$$

Finally, a cut with $(\mathbf{coerce}_{\S!})^{m-1}$ gives

$$(\ \mathrm{BIN})^{m-1}, (\S^{k+2}\ \mathrm{BIN})^n \ \vdash\ \S^{k+2}\ \mathrm{BIN}\,.$$

□

In the following section we show that safe variables in IMLAL2 cannot be similarly contracted. However, it is possible to normalize safe variables first and then contract them as normal variables.

3.3 Linear safe recursion

We introduce the function algebra BC^- as the fragment of BC defined by all of BC's rules, except that safe variables that occur in safe recursion and safe composition must do so in a *linear* fashion. Consequently, the two schemes have to be weakened respectively to (henceforth we write $f(\vec{x} : \vec{y})$ to mean functions definable in the function algebra BC^-):

$$\begin{aligned} f(\epsilon, \vec{x} : \vec{y}) &= g(\vec{x} : \vec{y})\,, \\ f(\mathsf{S}_i(z), \vec{x} : \vec{y}) &= h_i(z, \vec{x} : f(z, \vec{x} : \vec{y}))\,, \end{aligned}$$

and

$$\begin{aligned} &f(\vec{x} : y_1, \cdots, y_k) = \\ &\quad h(g_1(\vec{x} :), \cdots, g_a(\vec{x} :) : h_1(\vec{x} : y_{\sigma(1)}, \cdots, y_{\sigma(n_1)}), \cdots, h_b(\vec{x} : y_{\sigma(\sum_{j=1}^{b-1} n_j + 1)}, \cdots, y_{\sigma(k)})), \end{aligned}$$

where σ is a permutation of $\{\,1, \ldots, k\,\}$ and n_i $(1 \leqslant i \leqslant b)$ is the arity of h_i with respect to safe arguments (so $\sum_{j=1}^{b} n_j = k$).

Example 3.5. Concatenation of binary strings ($concat(x, y) = y \cdot x$) is definable in BC^- (and hence also in BC with one normal and one safe argument) by

$$\begin{aligned}
\mathbf{concat}(\epsilon : y) &= \pi_1^{0,1}(: y), \\
\mathbf{concat}(\mathsf{S}_i(x) : y) &= \mathsf{S}_i(\pi_2^{1,1}(x : \mathbf{concat}(x : y))).
\end{aligned}$$

Our first result is that BC^- can be interpreted in IMLAL2 in the following sense:

Theorem 3.6. For each $f(\vec{x} : \vec{y})$ in BC^- there exist $k \geqslant 0$ and $t_f(\vec{x} : \vec{y} \mid k)$ representing f in the sense of Definition 3.3.

Proof. The successors have already been dealt with in the previous chapter, so here we show the derivation for the predecessor. It uses iteration for $\mathbb{I} = (\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha)$, in which the first component holds the rightmost symbol and the second component holds all preceding characters. Finally, all we need to do is to use the second projection. The whole term is

$$\begin{aligned}
&n : \text{ BIN } \vdash \\
&\qquad \mathbf{\Lambda}\alpha.\boldsymbol{\lambda} f_0 f_1.\S\langle\, n[\mathbb{I}]\,!\langle\, f_0/f_0'\,\rangle(h_0)\,!\langle\, f_1/f_1'\,\rangle(h_1)/n'\,\rangle(\langle n'(I_\alpha \otimes I_\alpha)\,/n_1 \otimes n_2\,\rangle n_2)
\end{aligned}$$

where for $i = 0, 1$:

$$h_i = \boldsymbol{\lambda}(z_1 \otimes z_2).(f_i' \otimes \boldsymbol{\lambda} x.z_1(z_2 x)) : \mathbb{I} \multimap \mathbb{I}\,.$$

For the conditional we use iteration for

$$\mathbb{I} = (\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

and $b = \boldsymbol{\lambda}(x_\epsilon \otimes x_0 \otimes x_1).x_\epsilon : \mathbb{I}$ with the value meaning the appropriate projection that should be applied to the three potential results. The step functions p_i are constant and return the respective projections:

$$p_i = \boldsymbol{\lambda} x^\alpha.(\boldsymbol{\lambda}(x_\epsilon \otimes x_0 \otimes x_1).x_i) : \mathbb{I} \multimap \mathbb{I}$$

for $i = 0, 1$. At the end only the one corresponding to the least significant bit (or its absence) should be used:

$$\begin{aligned}
&n, n_\epsilon, n_0, n_1 : \text{ BIN } \vdash \\
&\quad \mathbf{\Lambda}\alpha.\boldsymbol{\lambda} f_0 f_1.\S\langle\, n[I]!(p_0)!(p_1)/n',\ n_\epsilon[\alpha] f_0 f_1/n_\epsilon',\ n_0[\alpha] f_0 f_1/n_0',\ n_1[\alpha] f_0 f_1/n_1'\,\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (n' b(n_\epsilon' \otimes n_0' \otimes n_1'))\,.
\end{aligned}$$

Projections for safe arguments are simply the canonical projections, whereas for normal arguments they have to be 'cut' with a suitable coercion.

The restricted version of safe composition is interpreted in the following way. By the Lifting Principle it is possible to find terms h,g_i $(1 \leqslant i \leqslant a)$ and h_j $(1 \leqslant j \leqslant b)$ representing the functions to be composed and typable as follows (we omit the names of variables in the contexts):

$$\text{BIN}^a, (\S^k\ \text{BIN})^b \ \vdash\ h : \S^k\ \text{BIN}\,,$$

$$\text{BIN}^m \ \vdash\ g_i : \S^l\ \text{BIN}\,,$$
$$\text{BIN}^m, (\S^{k+l}\ \text{BIN})^{n_j} \ \vdash\ h_j : \S^{k+l}\ \text{BIN}$$

for some $k, l \in \mathbb{N}$. After applying the ($\otimes$-r) rule several times to get $(\bigotimes_{i=1}^a g_i) \otimes (\bigotimes_{j=1}^b h_j)$ we use the (**exch**) rule as specified by the permutation σ to reach a derivation of

$$(\ \text{BIN}^m)^a,\ (\ \text{BIN}^m)^{n_1+\cdots+n_b},\ (\S^{k+l}\ \text{BIN})^{n_1+\cdots+n_b} \ \vdash\ (\S^l\ \text{BIN})^a \otimes (\S^{k+l}\ \text{BIN})^b\,,$$

which we 'cut' with

$$(\S^l\ \text{BIN})^a, (\S^{k+l}\ \text{BIN})^b \ \vdash\ \S^l h : \S^{k+l}\ \text{BIN}$$

and arrive at

$$(\ \text{BIN}^m)^a, (\ \text{BIN}^m)^{n_1+\cdots+n_b}, (\S^{k+l}\ \text{BIN})^{n_1+\cdots+n_b} \ \vdash\ \S^{k+l}\ \text{BIN}\,.$$

Now contractions on appropriate normal variables (performed in order to identify the occurrences of $\vec{x}$ in g_i and h_j) produce a proof of

$$\text{BIN}^m, (\S^{k+l+2}\ \text{BIN})^{n_1+\cdots+n_b} \ \vdash\ \S^{k+l+2}\ \text{BIN}\,,$$

which corresponds to the composite function.

For safe recursion with linear safe variables we can assume (by the Lifting Principle) that g, h_0, h_1 are interpreted by

$$\text{BIN}^m, (\S^k\ \text{BIN})^n \ \vdash\ g : \S^k\ \text{BIN}$$

and

$$\text{BIN},\ \text{BIN}^m, \S^k\ \text{BIN} \vdash\ h_i : \S^k\ \text{BIN}$$

respectively. Then we apply the ($\S$) rule to both:

$$(!\ \text{BIN})^m, (\S^{k+1}\ \text{BIN})^n \ \vdash\ g' : \S^{k+1}\ \text{BIN}\,,$$

$$!\ \text{BIN}, (!\ \text{BIN})^m, \S^{k+1}\ \text{BIN} \ \vdash\ h'_i : \S^{k+1}\ \text{BIN}\,.$$

In what follows, we also write g', h'_i for the curried versions of these proofs. Next we use iteration for $\mathbb{I} = (!\ \text{BIN})^m \otimes (\S^{k+1}\ \text{BIN})^n \multimap\ !\ \text{BIN} \otimes \S^{k+1}\ \text{BIN}$. The value is to represent $\boldsymbol{\lambda}(\vec{x} \otimes \vec{y}).(z \otimes f(z, \vec{x}; \vec{y}))$ during the iteration according to z. The step functions are:

$$g_i = \boldsymbol{\lambda} f(\vec{x} \otimes \vec{y}).\langle f(\vec{x} \otimes \vec{y}) \,/\, z \otimes r \,\rangle((!\langle\, z/z' \,\rangle(\mathsf{S}_i z')) \otimes h'_i z\vec{x}r) : \mathbb{I} \multimap \mathbb{I}$$

with the initial value $b = \boldsymbol{\lambda}(\vec{x} \otimes \vec{y}).((!\epsilon) \otimes g\vec{x}\vec{y})$. The iteration principle defines a derivation of

$$\text{BIN} \;\vdash\; \S\mathbb{I}\,.$$

Observe that

$$\S(A^m \otimes B^n \multimap A \otimes B) \;\vdash\; (\S A)^m \otimes (\S B)^n \multimap \S B$$

is provable and apply the **(cut)** rule to its obvious proof and the previous derivation to obtain

$$\text{BIN} \;\vdash\; (\S!\ \text{BIN})^m \otimes (\S^{k+2}\ \text{BIN})^n \multimap \S^{k+2}\ \text{BIN}\,.$$

Finally, uncurry and apply $\mathbf{coerce}_{\S!}$ to prove

$$\text{BIN}^{m+1}, (\S^{k+2}\ \text{BIN})^n \;\vdash\; \S^{k+2}\ \text{BIN}\,,$$

which interprets the safe recursion scheme of BC^-. □

In light of the theorem, we say that a numeric function $f(\vec{x}; \vec{y})$ (whose arguments are partitioned into those that are normal $\vec{x}$ and those that are safe $\vec{y}$ in a certain putative sense) is ***interpretable in IMLAL2*** just in case there are $k \geqslant 0$ and a sequent $t_f(\vec{x} : \vec{y}\,|\,k)$ representing f in the sense of Definition 3.3. That is to say, the safe and normal arguments of f must be compatible with safe and normal variables in IMLAL2.

Remark 3.7. There are numeric functions, representable as terms of solely safe variables in IMLAL2, which cannot be defined as BC-functions with solely safe arguments. For instance, *concat* (Example 3.5) is an example of such a function. The following, which we shall refer to as $\mathbf{concat}(: x, y \mid 0)$, represents *concat*:

$$\begin{aligned} &n_1 : \ \text{BIN},\ n_2 : \ \text{BIN} \ \vdash \\ &\qquad \boldsymbol{\Lambda}\alpha.\boldsymbol{\lambda} f_0 f_1.\S\langle\, n_1[\alpha] f_0 f_1 / n'_1, n_2[\alpha] f_0 f_1 / n'_2 \,\rangle(\boldsymbol{\lambda} x.n'_1(n'_2 x)) : \ \text{BIN}\ . \end{aligned}$$

That *concat* is not definable in BC with two safe arguments is a consequence of the following invariance of BC (see [16]): for each $f(\vec{x}; \vec{y})$ in BC there exists a polynomial p_f such that

$$|f(\vec{x}; \vec{y})| \;\leqslant\; p_f(|x_1|, \cdots, |x_m|) + \max\{\, |y_j| : 1 \leqslant j \leqslant n \,\}.$$

A natural question to ask is whether BC^- is the largest subalgebra of BC that is interpretable in IMLAL2. The central issue in determining this is whether safe variables in IMLAL2 are contractible (i.e. whether the analogous version of Proposition 3.4(ii) is valid for safe variables). The answer is no, for if they were, the numeric function $dup(s) = concat(s, s)$ would be representable by a term $\mathbf{dup}(: x \mid 0)$ with a safe variable. It then follows that the following function would also be representable in IMLAL2:

$$\begin{array}{rcl} f(\epsilon) & = & 1\,, \\ f(\mathsf{S}_i(x)) & = & dup(f(x))\,, \end{array}$$

but it is easy to check that $f(s) = \underbrace{1 \cdots 1}_{2^{|s|}}$.

We highlight another difference between BC and BC^-: the step functions of the safe recursion scheme in BC^- are restricted to those that have at most one safe argument. It turns out that this restriction is also necessary. Indeed, were the full safe recursion scheme interpretable in IMLAL2 (in a way that is consistent with our definition of safe and normal variables in the logic), then we could define a function $g(x, y)$ that produces a string of length $|y|^{|x|}$ as follows. First, set f to

$$\begin{array}{rcl} f(\epsilon, y) & = & y\,, \\ f(\mathsf{S}_i(x), y) & = & concat(y, f(x, y))\,, \end{array}$$

which keeps y safe in IMLAL2, and then define g by

$$\begin{array}{rcl} g(\epsilon, y) & = & 1\,, \\ g(\mathsf{S}_i(x), y) & = & f(y, g(x, y))\,. \end{array}$$

We can interpret *concat* as $\mathbf{concat}(: x, y \mid 0)$ in Remark 3.7, so both definition are translatable into IMLAL2. $f(x, y)$ has the effect of copying y in binary $|x|$ times and g produces a string of $|y|^{|x|}$ 1's.

We can infer from the preceding that if BC is augmented by a version of *concat* that has two safe arguments (which we know is not BC-definable), we violate the property of FP closure. By contrast, there are functions which must be recursively defined in BC (and so have at least one normal argument), but which can be added to BC^- as functions with solely safe arguments (provided they are interpretable in IMLAL2) without violating its FP closure. We explore this possibility in the sections to follow.

3.4 Completing BC^-

In this section we enrich BC^- with new constructs that are not definable in it, though they are interpretable in IMLAL2.

A definition-by-cases construct

As safe variables are not contractible in BC^-, it is impossible to define branching constructs of solely safe arguments such that the selector expression (x say) can still be manipulated after the choice has been made, like in

$$\mathbf{cond}(: x, \mathsf{S}_0(x), x, \mathsf{P}(x)),$$

which is definable in BC. Fortunately, it turns out that some such functions are interpretable in IMLAL2, though the range of actions that may be defined on the selector expression is rather limited.

We introduce a pattern-matching case construct: for $K, m \geqslant 0$, and for strings $p_1, \cdots, p_m$ such that $|p_i| \leqslant K$

$$\mathbf{case}_K(: u)[p_1 : f_1 | \cdots | p_m : f_m | \mathsf{else} : f_{m+1}] \;=\; \begin{cases} f_i(u) & \text{if the least sig. } K \text{ bits} \\ & \text{of } u \text{ match } p_i\,, \\ f_{m+1}(u) & \text{otherwise}\,. \end{cases}$$

We stipulate that any pattern p_i shorter than K can only be matched by itself (this also applies to the empty string). Moreover, the actions f_j $(1 \leqslant j \leqslant m+1)$ are required to be of a certain form:

$$f_j(n) \;=\; concat(s_j^1, concat(\mathsf{P}^{k_j}(n), s_j^2)),$$

where $k_j \geqslant 0$ and s_j^1, s_j^2 are binary strings. That is, each f_j may delete a fixed number of the least significant bits and then append some bits in their place (s_j^1) and also some bits at the other end (s_j^2).

A construct related to our $\mathbf{case}_K$ (based on remainders modulo 2^K) is definable in BC (see [50]); in that case construct any BC-definable function may be chosen as the action after the selection. This does not seem possible in IMLAL2 under our regime of safe and normal variables. For this reason, we believe that functions definable in the function algebra $BC^- + \mathbf{case}_K$ are a proper subalgebra of BC.

3.4.1 perm-rec: recursion on *safe* arguments

Surprisingly, BC^- can be extended by a form of definition-by-recursion scheme on *safe* arguments which is interpretable in IMLAL2, and so the extension preserves FP closure. Certainly not all functions can be used as the corresponding step functions of the recursion scheme. Here we extract the first such scheme from IMLAL2.

The functions that will be permissible as step functions are instances of a branching construct $\mathbf{perm}_L(: u)$ that permutes the last L bits of the input u: for $k \geqslant 0, L > 0$, strings $p_1, \cdots, p_k$ (all of length L):

$$\mathbf{perm}_L(: u)[p_1 : f_1 \mid \cdots \mid p_k : f_k] = \begin{cases} f_i(: u) & \text{if the } L \text{ least sig. bits of } u \text{ are } p_i, \\ u & \text{otherwise.} \end{cases}$$

Each action f_i ($1 \leqslant i \leqslant k$) permutes the L least significant bits of the argument i.e. $f_i = \mathsf{S}_{r_i} \circ \mathsf{P}^L$, where r_i is a permutation of p_i and $\mathsf{S}_s(x) = concat(s, x)$ for any string s. The construct returns u, if $|u| < L$, or if the L least significant bits of u do not match any of the patterns. For example, let $p(: u)$ be

$$\mathbf{perm}_3(: u)[101 : \mathsf{S}_{011} \circ \mathsf{P}^3 \mid 001 : \mathsf{S}_{100} \circ \mathsf{P}^3 \mid 010 : \mathsf{S}_{001} \circ \mathsf{P}^3] \,.$$

For $u = 1010, 1001, 10$ we have $p(: u) = 1001, 1100$ and 10 respectively. Note that the above construct is non-size-increasing in the sense of Hofmann [56].

The new definition-by-recursion scheme over safe arguments, called **perm-rec**, has the form:

$$\begin{aligned} f(\vec{x} : \epsilon, \vec{y}) &= h(\vec{x} : \vec{y})\,, \\ f(\vec{x} : \mathsf{S}_i(z), \vec{y}) &= \mathbf{step}_i(: f(\vec{x} : z, \vec{y}))\,, \end{aligned}$$

where the step functions $\mathbf{step}_i(: u) = p_i(: \mathsf{S}_{j_i}(u))$ for $i, j_i \in \{0, 1\}$, and $p_i(: x)$ is some fixed instance of the **perm** construct. Note the special case of $\mathbf{step}_i = \mathsf{S}_{j_i}$ and $\mathbf{step}_i = \mathsf{S}_i$.

Example 3.8. We define

$$\begin{aligned} f(: \epsilon) &= \epsilon\,, \\ f(: \mathsf{S}_i(z)) &= \mathbf{step}_i(: f(: z)) \end{aligned}$$

using the **perm** construct in the preceding example as $p_i(: n)$ and taking i as j_i. Let us compute $f(: 110101)$. Since $f(: 110) = 110$, we have $f(: 1101) = p_1(: \mathsf{S}_1 f(: 110)) = p_1(: 1101) = 1011$. And so $f(: 11010) = p_0(: \mathsf{S}_0 f(1101)) = p_0(: 10110) = 10110$. Finally $f(: 110101) = p_1(: \mathsf{S}_1 f(: 11010)) = p_1(: 101101) = 101011$.

3.4.2 Terms of type bin $\multimap$ bin.

An interesting class of functions can be defined using terms of type $\text{BIN} \multimap \text{BIN}$. They will turn out not to change the size of the input by more than constant.

The argument $n : \text{BIN}$ in such terms can be instantiated at some type $n[T]$ and then applied to two terms of type $!(T \multimap T)$. η-expanded terms of this type are always of the form $!\langle\,\text{-}/\text{-}\,\rangle(\text{-})$, so each term of type $\text{BIN} \multimap \text{BIN}$ has the shape

$$\begin{aligned} &\boldsymbol{\lambda} n \boldsymbol{\Lambda}\alpha.\boldsymbol{\lambda} f_0^{!(\alpha \multimap \alpha)} f_1^{!(\alpha \multimap \alpha)}. \\ &\qquad \S\langle\, n[T]\, !\langle\, f_i/f_i' \,\rangle(F_0 f_i')\, !\langle\, f_j/f_j' \,\rangle(F_1 f_j')/n', \{f_0/f_0'\}, \{f_1/f_1'\} \,\rangle(F n' \{f_0'\}\{f_1'\}), \end{aligned}$$

where $\{f_k/f_k'\}$ denotes some finite number of imports of f_0 or f_1 via the interface of $\S\langle\,\cdot\,\rangle(\cdot)$. We can glean several useful facts from the shape:

- The step functions $!\langle\, f_k/f_k' \,\rangle(F_i f_k')$ use single copies of f_k. Therefore at each step the size can increase by at most 1. This resembles Hofmann's [56].

- It follows from the previous remark that $|f(x)| \leqslant c + |x|$.
- The kind of contribution of each step function is fixed (they always contribute only 0's or only 1's, if at all).
- Any fixed number of f_k's can be accessed through the interface, so the initial value can be arbitrary. These bits can also be concatenated to the result of iteration.

3.5 $BC^{\pm}$ characterizes FP

Definition 3.9 ($BC^{\pm}$)**.** The function algebra $BC^{\pm}$ is defined by augmenting BC^{-} by **case** and **perm-rec**.

Observe that $\mathbf{concat}(: x, y)$ (with the obvious semantics of *concat*) becomes definable in $BC^{\pm}$ thanks to **perm-rec**:

$$\begin{array}{rcl} \mathbf{concat}(: \epsilon, y) & = & \pi_1^{0,1}(: y)\,, \\ \mathbf{concat}(: \mathsf{S}_i(x), y) & = & \mathsf{S}_i(: \mathbf{concat}(: x, y))\,. \end{array}$$

Theorem 3.10. $BC^{\pm}$ is FP sound and complete: $BC^{\pm}$-functions $f(x :)$ are exactly the FP computable functions.

First we shall show that the two new constructs **case** and **perm-rec** are interpretable in IMLAL2. Thus, we can infer that $BC^{\pm}$-definable functions are in FP. The rest of the section is devoted to a proof of the completeness direction.

3.5.1 $BC^{\pm}$ is FP closed

Interpreting $\mathbf{case}_{K,m}$ in IMLAL2

Recall that

$$f_j(: n) = concat(s_j^1, concat(\mathsf{P}^{k_j}(n), s_j^2)).$$

Let $k = \max(k_1, \cdots, k_m, k_{m+1})$. Hence, k denotes how many characters have to be erased from the input in the worst case. Our method is to erase those k characters and later, if necessary, attach an appropriate number of them back. We are forced to do this, because in our representation it is easy to add characters, but difficult to delete them (just compare the definitions of the successor and predecessor functions!).

We shall use the iteration principle for

$$\mathbb{I} = (P_3)^K \otimes (\alpha \multimap \alpha)^k \otimes (\alpha \multimap \alpha)$$

where $P_3 = \forall\alpha.\alpha \otimes \alpha \otimes \alpha \multimap \alpha$, and the three associated projections are called π_0, π_1, π_2. The first K-tuple is used to store K projections corresponding to the K least significant bits (π_0 or π_1 is stored then) or their absence (signalled by π_2). The following k-tuple will contain variables representing the k least significant bits of the selector value. In the last component we store $\mathsf{P}^k(n)$. To preserve the meaning of the tuple, the step functions are defined by

$$\begin{aligned}
&f_i : (\alpha \multimap \alpha) \vdash \\
&\quad \boldsymbol{\lambda}((\textstyle\bigotimes_{j=0}^{K-1} p_j) \otimes (\bigotimes_{j=0}^{k-1} b_j) \otimes p). \\
&\qquad ((\pi_i \otimes \textstyle\bigotimes_{j=0}^{K-2} p_j) \otimes (f_i \otimes \bigotimes_{j=0}^{k-2} b_j) \otimes (\boldsymbol{\lambda} x^\alpha . b_{k-1}(px)))
\end{aligned}$$

with the initial value of $(\pi_2)^K \otimes (I_\alpha)^k \otimes I_\alpha$. The iteration principle yields then a derivation of

$$\text{BIN}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \S\mathbb{I}.$$

To complete the translation, we define a proof of

$$!(\alpha \multimap \alpha), !(\alpha \multimap \alpha), \S\mathbb{I} \vdash \S(\alpha \multimap \alpha)\,,$$

whose role is to select the appropriate functions to be applied to $\mathsf{P}^k(n)$ and the k least significant bits. The derivation will be of the shape

$$\S\langle \{f_0/f_0'\}, \{f_1/f_1'\}, x/x' \rangle(\langle x' /(p_0 \otimes \cdots \otimes p_{K-1}) \otimes (b_0 \otimes \cdots \otimes b_{k-1}) \otimes p \rangle \ \mathbf{handle}),$$

where $\mathbf{handle} : (\alpha \multimap \alpha)$ and $\{f_i/f_i'\}$ $(i = 0, 1)$ denotes $f_i/f_{i1}', \cdots, f_i/f_{ij_i}'$ for $j_i \geqslant 0$. For simplicity, each f_{ik}' will be referred to as f_i'. It will become clear soon that j_0, j_1 should be chosen as sufficiently large numbers after examination of all s_j^1 and s_j^2 $(j = 1, \ldots, m+1)$. Our next aim is the definition of **handle**.

Suppose $f(: n) = \mathit{concat}(s^1, \mathit{concat}(\mathsf{P}^h(n), s^2))$ is to be implemented. f can be realized by applying

$$\begin{aligned}
&\mathbf{act} = \boldsymbol{\lambda}(b_0 \otimes \cdots \otimes b_{k-1} \otimes p). \\
&\qquad \boldsymbol{\lambda} x. f_{1,|s^1|}'(\cdots f_{1,1}'(b_h \cdots (b_{k-1}(p(f_{2,|s^2|}'(\cdots (f_{2,1}' x) \cdots)))))) : \text{ACTION}
\end{aligned}$$

to the (representations of the) k least significant bits of n and $\mathsf{P}^k(n)$ as implied by the names of variables. We have $\text{ACTION} = (\alpha \multimap \alpha)^{k+1} \multimap (\alpha \multimap \alpha)$. Besides, each variable $f_{i,j}'$ for $i = 1, 2$ and $j = 1, \ldots, |s^i|$ should be identical to some f_0' if $s_j^i = 0$, or some f_1' if $s_j^i = 1$. Therefore, the numbers j_i for $i = 0, 1$ must be large enough to accommodate all possibilities for s_j^1 and s_j^2. In addition, the term for the default case of s_{m+1}^1, s_{m+1}^2 will be used several times as will be explained below.

Note that **act** works correctly even if $P^l(n) = \epsilon$ for some $l < h$ (then we have $b_m = I_\alpha$ for $m \geqslant l$).

For each $j = 1, \ldots, m+1$ we prepare a version of **act** depending on s_j^1, s_j^2 and h_j. Such terms are then arranged in a table **table** : ROW_K, where:

$$\begin{array}{rcll} \text{ROW}_1 & = & \text{ACTION}^3 , & \\ \text{ROW}_{i+1} & = & (\text{ROW}_i)^3 & \text{for } i = 1, \cdots, K-1 . \end{array}$$

As many as 3^K such pairs of terms can be stored inside the table, which covers all patterns built from 0,1 and 'lack of bit' (obviously some combinations will never be needed). Copies of the term **act** corresponding to f_{m+1} (from the **case** construct) must be cloned with separation of variables to account for the default case. This is the final factor on which j_0, j_1 depend. As there are 3^K entries, the choice of appropriate j_0, j_1 is always possible.

Once **table** is constructed, the K projections (stored as the first component of the tuple used for iteration) are employed to select the appropriate action:

$$\begin{array}{l} \textbf{handle} = \\ \quad p_0[\text{ACTION}](\cdots(p_{K-2}[\text{ROW}_{K-2}](p_{K-1}[\text{ROW}_{K-1}]\ \textbf{table})))(b_0 \otimes \cdots \otimes b_{k-1} \otimes b) \ . \end{array}$$

We have **handle** : $(\alpha \multimap \alpha)$. To find a term representing the **case** construct as a proof of $\text{BIN} \vdash \text{BIN}$ it remains to combine the two derivations of

$$\text{BIN}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \ \vdash \ \S\mathbb{I}$$

and

$$!(\alpha \multimap \alpha), !(\alpha \multimap \alpha), \S\mathbb{I} \ \vdash \ \S(\alpha \multimap \alpha)$$

with a cut and to perform two contractions for $!(\alpha \multimap \alpha)$ followed by two lambda abstractions and universal quantification.

Interpreting perm-rec in IMLAL2

Recall that the **perm** construct involved in **perm-rec** makes use of $L-1$ bits of the interim result—the Lth bit is provided by the step function. Hence, we only need to store $L-1$ bits to make the next step.

Accordingly, we take advantage of iteration for

$$\mathbb{I} = (P_3)^{L-1} \otimes (\alpha \multimap \alpha)^{L-1} \otimes (\alpha \multimap \alpha) .$$

The first $L-1$ elements of the tuple are projections $p_1, \cdots, p_{L-1} : P_3$ (as in the previous encoding), which correspond to the $L-1$ least significant bits of the interim result. The projections π_0, π_1 correspond to 0 and 1 respectively and π_2 means that the corresponding bit is not available. The next $L-1$ elements $b_1, \cdots, b_{L-1}$ are the actual last $L-1$ bits of the interim result in the form of imported f_0 or f_1 'filtered' through the interface (as in the previous example). The last component p contains P^{L-1} of the current iteration result as a term of type $(\alpha \multimap \alpha)$.

The step functions will add f_{j_i} and π_i to the current tuple as b_0 and p_0 respectively. Then the L projections will be used to select the right action to be taken with respect to $b_0, \cdots, b_{L-1}$, $p_0, \cdots, p_{L-1}$ and p: the bits and projections should be permuted and the most significant bit of the result appended to p. Projections are therefore needed twice and we duplicate them as follows:

$$\mathbf{copy}\, p = p[P_3 \otimes P_3]((\pi_0 \otimes \pi_0) \otimes (\pi_1 \otimes \pi_1) \otimes (\pi_2 \otimes \pi_2))\,.$$

The actions will have type:

$$\textsc{Action} = (P_3)^L \otimes (\alpha \multimap \alpha)^L \otimes (\alpha \multimap \alpha) \multimap (P_3)^{L-1} \otimes (\alpha \multimap \alpha)^{L-1} \otimes (\alpha \multimap \alpha).$$

For instance, if 100 is to be converted into 010, the term:

$$\boldsymbol{\lambda}((p_0 \otimes p_1 \otimes p_2) \otimes (b_0 \otimes b_1 \otimes b_2) \otimes p).(p_0 \otimes p_2) \otimes (b_0 \otimes b_2) \otimes \boldsymbol{\lambda} x.b_1(px) : \textsc{Action}$$

effects the required transformation.

Terms of this sort for all necessary permutations (as specified by the **perm** construct) will be arranged in $\mathbf{table} : \textsc{row}_L$, where:

$$\begin{array}{rcll} \textsc{row}_1 & = & \textsc{Action}^3\,, & \\ \textsc{row}_{i+1} & = & (\textsc{row}_i)^3 & \text{for } i = 1, \cdots, L-1\,. \end{array}$$

Assuming the current projections are $p'_0, \cdots, p'_{L-1}$ the corresponding action can be chosen by

$$\mathbf{select} = p'_0[\textsc{Action}](p'_1[\textsc{row}_1](\cdots p'_{L-1}[\textsc{row}_{L-1}]\,\mathbf{table})) : \textsc{Action}\,.$$

The step functions for $i = 0, 1$ can then be defined as

$$\begin{array}{l} !\langle\, f_{j_i}/b_0\,\rangle(\boldsymbol{\lambda}((\otimes p_k) \otimes (\otimes b_l) \otimes p). \\ \qquad \langle \otimes(\mathbf{copy}\, p_k)\,/\otimes(p'_k \otimes p''_k)\,\rangle\, \mathbf{select}\,((\pi_i \otimes (\otimes p''_k)) \otimes (b_0 \otimes (\otimes b_l)) \otimes p)) \end{array}$$

of type $\mathbb{I} \multimap \mathbb{I}$. The iteration principle for $\Box = \S$ and the initial value $x : \mathbb{I} \vdash x : \mathbb{I}$ returns a derivation of

$$\textsc{bin}, \S\mathbb{I}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \S\mathbb{I}\,.$$

To complete the definition, we still need to extract the result from $\mathbb{I}$ using

$$\boldsymbol{\lambda}((\otimes p_k) \otimes (b_1 \otimes \cdots \otimes b_{L-1}) \otimes p).\boldsymbol{\lambda} x^{\alpha}.b_1(\cdots(b_{L-1}(px))\cdots) : \mathbb{I} \multimap (\alpha \multimap \alpha)$$

to get a proof of

$$\textsc{bin}, \S\mathbb{I}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)\,.$$

After applying three abstraction rules, a derivation of

$$\text{BIN}, \S\mathbb{I} \vdash \text{BIN}.$$

will be reached. Then we have to 'initialize' the value of type $\S\mathbb{I}$ and for this purpose we define a proof of $\text{BIN} \vdash \S\mathbb{I}$ by iteration for $\mathbb{I}$ with the seed $(\pi_3)^{L-1} \otimes (I_\alpha)^{L-1} \otimes I_\alpha$ and step functions:

$$f_i : (\alpha \multimap \alpha) \vdash$$
$$\boldsymbol{\lambda}(\bigotimes_{j=0}^{L-2} p_j \otimes \bigotimes_{j=0}^{L-2} b_j \otimes p).(\pi_i \otimes \bigotimes_{j=0}^{L-3} p_j) \otimes (f_i \otimes \bigotimes_{j=0}^{L-3} b_j) \otimes (\boldsymbol{\lambda}x.b_{L-2}(px)) \ .$$

After a cut with the previously obtained proof of $\text{BIN}, \S\mathbb{I} \vdash \text{BIN}$ we get a derivation of

$$\text{BIN},\ \text{BIN} \vdash \text{BIN}.$$

As the initial value of the recursion is $g(\vec{x} : \vec{y})$, we consider its IMLAL2 interpretation:

$$\text{BIN}^m, (\S^k\ \text{BIN})^n \vdash t_g : \S^k\ \text{BIN}.$$

We $\S^k$-lift the proof of $\text{BIN},\ \text{BIN} \vdash \text{BIN}$ to one of

$$\S^k\ \text{BIN}, \S^k\ \text{BIN} \vdash \S^k\ \text{BIN}$$

and use the cut rule for the right copy of $\S^k$ BIN on the left and for t_g to get the final derivation of

$$(\ \text{BIN})^m, (\S^k\ \text{BIN})^{n+1} \vdash \S^k\ \text{BIN},$$

which interprets the **perm-rec** construct.

We have shown that all $BC^\pm$ definitions are interpretable in IMLAL2, which completes our proof that functions definable in $BC^\pm$ are in FP.

3.5.2 Any FP function is representable in $BC^\pm$

Roversi's completeness proof [109] for IMLAL2 cannot be repeated in our setting, because it uses a carefully selected type for coding machine configurations which is related to but different from BIN. Handley's proof [50] for BC does use $\mathbb{N}$ to encode configurations, but it relies critically on the contractibility of safe arguments. So does Bellantoni and Cook's which demonstrates how Cobham-style definitions can be translated into BC. Here we propose a new way of encoding poly-time Turing machines using the constructs of $BC^\pm$.

Polynomials

First we need a way of representing polynomials for defining the polynomial clock. For each polynomial p with positive integer coefficients we define

$$f_p(n:) = \underbrace{1\cdots 1}_{p(|n|)}$$

by induction on (representations of) polynomials with one indeterminate x:

$$\begin{aligned} f_1(n:) &= 1\,, \\ f_{p_1+p_2}(n:) &= \mathbf{concat}(f_{p_1}(n:) : f_{p_2}(n:))\,, \\ f_{x\cdot p}(n:) &= f'(n, n:)\,, \end{aligned}$$

where $f'(n_1, n_2 :) = \underbrace{1\cdots 1}_{|n_1|\cdot p(|n_2|)}$ is defined as follows:

$$\begin{aligned} f'(\epsilon, x:) &= \epsilon\,, \\ f'(\mathsf{S}_i(z), x:) &= \mathbf{concat}(f_p(x:) : f'(z, x:))\,. \end{aligned}$$

Configurations

Let us fix a Turing machine with the initial state q_0. Suppose its alphabet includes $0, 1$ and the blank symbol $\sqcup$. We require the input to be placed to the right of the head with the most significant bit below the head. The same convention applies to the output. Furthermore, the rest of the tape should be blank at the end of the computation.

Let us choose an even number L which is sufficiently large to allow for an encoding of the alphabet and the set of states as binary strings of L bits, half of which are zeros. Clearly, the code of each symbol can be converted into that of another by permutation and the same applies to the states. We write $\ulcorner\theta\urcorner$ for the code of θ where θ ranges over alphabet-symbols and states, and we require the function $\ulcorner - \urcorner$ to be injective.

The configuration of the machine is coded by the code of the current state followed by L zeros and L ones between (codes for) the left and right part of the tape. The L zeros and ones will later enable us to distinguish the position of the head. Hence, if the current state has code $s_0 \cdots s_{L-1}$ and the symbol under the head has code $r_0 \cdots r_{L-1}$, the configuration is coded by the string

$$\cdots l_0 \cdots l_{L-1} s_0 \cdots s_{L-1} \underbrace{0\cdots 0}_{L}\underbrace{1\cdots 1}_{L}\overline{r_0 \cdots r_{L-1}} r_L \cdots r_{2L-1} \cdots$$

where $\cdots l_0 \cdots l_{L-1}$ refers to the string that codes the part of the tape which is to the left of the head. The string consisting of the overlined bits codes the symbol under the head.

Transition

To make it clear that each possible transition corresponds to a permutation of the above $6L$ bits, we show the code of the new configuration if the next state is coded by $s'_0 \cdots s'_{L-1}$, $r'_0 \cdots r'_{L-1}$ is written to the tape and the head moves to the right:

$$\cdots l_0 \cdots l_{L-1} r'_0 \cdots r'_{L-1} s'_0 \cdots s'_{L-1} \underbrace{0 \cdots 0}_{L} \underbrace{1 \cdots 1}_{L} \overline{r_L \cdots r_{2L-1}} \cdots$$

or to the left:

$$\cdots s'_0 \cdots s'_{L-1} \underbrace{0 \cdots 0}_{L} \underbrace{1 \cdots 1}_{L} \overline{l_0 \cdots l_{L-1}} r'_0 \cdots r'_{L-1} r_L \cdots r_{2L-1} \cdots$$

To mark the fact that the transition has taken place, we change the $2L$ bits $\underbrace{0 \cdots 0}_{L} \underbrace{1 \cdots 1}_{L}$ to $\underbrace{1 \cdots 1}_{L} \underbrace{0 \cdots 0}_{L}$. Therefore, each transition with the change of $2L$ bits can be implemented using **perm-rec** with $\mathbf{perm}_{6L}$. Next we can use **perm-rec** with $\mathbf{perm}_{2L}$ just to change $\underbrace{1 \cdots 1}_{L} \underbrace{0 \cdots 0}_{L}$ back to $\underbrace{0 \cdots 0}_{L} \underbrace{1 \cdots 1}_{L}$ to enable the next step (we stipulate $\ulcorner \theta \urcorner \neq \underbrace{1 \cdots 1}_{\frac{L}{2}} \underbrace{0 \cdots 0}_{\frac{L}{2}}$ to avoid matching of the special segment by some symbol). Thus, a complete step can be performed by a function

$$\mathbf{transition}(: n)\,.$$

Suppose the initial configuration (assuming the input is n) is given by $\mathbf{init}(n :)$. Then the configuration after $p(|n|)$ steps is $\mathbf{iterate}(f_p(n :) : \mathbf{init}(n :))$, where:

$$\begin{aligned} \mathbf{iterate}(\epsilon : y) &= y\,, \\ \mathbf{iterate}(\mathsf{S}_i(z) : y) &= \mathbf{transition}(: \mathbf{iterate}(z : y))\,. \end{aligned}$$

Initial configuration

Because **perm** cannot increase the size of the tape, the initial tape has to be long enough for the whole computation, whose duration is controlled by a fixed polynomial clock. For this purpose, we simply supply $p(|n|)$ blank cells on both sides of the input. For $|x|$ blank cells we call $\mathbf{bl}(x :)$:

$$\begin{aligned} \mathbf{bl}(\epsilon :) &= \epsilon\,, \\ \mathbf{bl}(\mathsf{S}_i(x) :) &= \mathsf{S}_{\ulcorner \sqcup \urcorner}(: \mathbf{bl}(x :))\,. \end{aligned}$$

rep should be used to encode the input string:

$$\begin{aligned} \mathbf{rep}(\epsilon :) &= \epsilon\,, \\ \mathbf{rep}(\mathsf{S}_i(x) :) &= \mathsf{S}_{\ulcorner i \urcorner}(\mathbf{rep}(x :))\,. \end{aligned}$$

Finally, the whole initial configuration for input n is given by:

$$\begin{aligned} &\mathbf{init}(n :) = \\ &\quad \mathbf{concat}(\mathbf{bl}(f_p(n :) :) : \mathbf{concat}(\mathbf{concat}(\mathbf{rep}(n :) : \mathsf{S}_{0^L 1^L}(\ulcorner q_0 \urcorner)) : \mathbf{bl}(f_p(n :) :)))\,. \end{aligned}$$

Extraction

To convert the final configuration to an output string, we have to erase the left-hand side of the tape, the state and the auxiliary $2L$ bits, decode the non-blank string on the right and erase the rest of the representation of the right tape. We will process these bits in groups of L using $L-1$ extra bits for counting. The $L-1$ bits will be used to recognize with which bit we deal: the ith bit ($i = 1, \cdots, L$) is represented by

$$\bar{i} = \underbrace{0 \cdots 0}_{L-i}\underbrace{1 \cdots 1}_{i-1} .$$

Then $\mathbf{result}(n :)$ extracts the output string accompanied by the $L-1$ auxiliary bits:

$$\begin{aligned} \mathbf{result}(\epsilon :) &= \bar{1}\,, \\ \mathbf{result}(\mathsf{S}_i(x) :) &= \mathbf{case}_{2L-1}(: \mathsf{S}_i(: \mathbf{result}(x :)))\, Q\,, \end{aligned}$$

where

$$Q = \left[\begin{array}{lcll} t\bar{j}b & : & \mathsf{S}_{b\,\overline{j+1}} \circ \mathsf{P}^L & |t| \leqslant L-1,\ b = 0,1,\ 1 \leqslant j < L \quad | \\ t\bar{L}b & : & \mathsf{S}_{c\bar{1}} \circ \mathsf{P}^L & |t| = L-1,\ b,c = 0,1,\ tb = \ulcorner c \urcorner \quad | \\ t\bar{L}b & : & \mathsf{S}_{\bar{1}} \circ \mathsf{P}^L & |t| = L-1,\ b,c = 0,1,\ tb \neq \ulcorner c \urcorner \quad | \\ \mathsf{else} & : & \mathsf{P}^0. & \end{array}\right]$$

The output can now be obtained as follows:

$$\mathbf{final}(n :) = \mathsf{P}^{L-1}(\mathbf{result}(n :))\,.$$

The whole computation

It follows that every numeric function $g(n)$ computable by a Turing machine running in time $p(|n|)$ can be simulated by:

$$\mathbf{final}(\mathbf{iterate}(f_p(n :) : \mathbf{init}(n :)) :)$$

which wraps up our proof of $BC^{\pm}$'s FP completeness. Note that this also constitutes a new proof of the FP completeness of IMLAL2.

3.6 Another FP completion of BC^-

We have seen that $BC^{\pm}$ is FP complete. There are also other ways to complete BC^-.

Another function which we can add to BC^- without breaking polynomial-time computability is $\mathbf{e\text{-}shift}(: n)$, which "shifts even bits to the left":

$$\begin{aligned} \mathbf{e\text{-}shift}(: s_{2n+1} \cdots s_1 s_0) &= s_{2n} s_{2n+1} \cdots s_4 s_5 s_2 s_3 s_0 s_1 0\,, \\ \mathbf{e\text{-}shift}(: s_{2n} \cdots s_1 s_0) &= s_{2n} 0 s_{2n-2} s_{2n-1} \cdots s_2 s_3 s_0 s_1 0\,. \end{aligned}$$

For instance, **e-shift** : $1\underline{1} \mapsto \underline{1}10$, $\underline{1}1\underline{1} \mapsto \underline{1}0\underline{1}10$ and $\underline{1}0\underline{1}1\underline{1} \mapsto \underline{1}0\underline{1}0\underline{1}10$ (we underlined the even bits to indicate their movements). Note that **e-shift**$(: n)$ does not violate the invariance in Remark 3.7, but despite that it fails to satisfy its strengthened version proposed by Handley (Claim 3 in [50]):

Proposition 3.11. Each $f(\vec{x} : \vec{y})$ is computable in time $p_f(\vec{x})$ for some polynomial p_f on multi-tape Turing machines that can copy the entire content of one tape to another in one step.

Were **e-shift**$(; n)$ definable in BC, one could compute it in constant time (with copying), which is impossible. This highlights yet another difference between safe variables of IMLAL2 and BC, since we are going to demonstrate that **e-shift**$(: n)$ is interpretable in IMLAL2 (with safe input!).

In the rest of this section we show that the function algebra $BC^- +$ **case** $+$ **e-shift** is also FP complete.

Representation of e-shift in IMLAL2

Here we focus on the construction of an IMLAL2 term **e-shift** : $\text{BIN} \multimap \text{BIN}$ representing **e-shift**$(: n)$. We begin with a function that reverses a binary list.

The function arises by iteration for $\mathbb{I} = \alpha \multimap \alpha$ starting from I_α with step functions:

$$f_i : (\alpha \multimap \alpha) \vdash \boldsymbol{\lambda} f^{(\alpha \multimap \alpha)}.\boldsymbol{\lambda} x.f(f_i x)) : (\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)\,.$$

The resultant derivation of $\text{BIN}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)$ defines **rev** : $\text{BIN} \multimap \text{BIN}$ in three steps.

Shift

The function to be defined in this section takes a list and shifts every other element to the left beginning with the last:

$$\begin{array}{rcr} b_{2k}b_{2k-1}b_{2k-2}\cdots b_1b_0 & \mapsto & b_{2k}0b_{2k-2}b_{2k-1}\cdots b_0b_1 0\,, \\ b_{2k+1}b_{2k}b_{2k-1}b_{2k-2}\cdots b_1b_0 & \mapsto & b_{2k}b_{2k+1}b_{2k-2}b_{2k-1}\cdots b_0b_1 0\,. \end{array}$$

We take advantage of the iteration principle for $\mathbb{I} = P_2 \otimes (\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha)$. The intended meaning of interim values throughout the iteration will be:

$$\pi_0 \otimes I_\alpha \otimes \boldsymbol{\lambda} x.b_{2k}(b_{2k+1}(\cdots b_0(b_1(f_0 x))))$$

after processing an even number of step functions (i.e. for $b_0, \cdots, b_{2k+1}$) and

$$\pi_1 \otimes b_{2k} \otimes \boldsymbol{\lambda} x.b_{2k-2}(b_{2k-1}(\cdots b_0(b_1(f_0 x))))\,,$$

if the pattern processed so far has odd length (i.e. $b_0, \cdots, b_{2k}$ have been processed). The first component of the triple is a projection indicating the case we deal with. Thus, the initial value must be $\pi_0 \otimes I_\alpha \otimes I_\alpha : \mathbb{I}$, because the first step is the odd-length case. In this case the current bit f needs to be kept for future use by the other case, so the triple should be modified by

$$F_1 = \boldsymbol{\lambda}(p \otimes b \otimes c).(\pi_1 \otimes f \otimes c) : \mathbb{I} \multimap \mathbb{I}\,.$$

For the even-length case we use

$$F_2 = \boldsymbol{\lambda}(p \otimes b \otimes c).\pi_0 \otimes I_\alpha \otimes \boldsymbol{\lambda}x.b(f(cx)) : \mathbb{I} \multimap \mathbb{I}\,.$$

Note that f occurs in both, but we are allowed to use it at most once. For this reason, we shall consider

$$\boldsymbol{\lambda}f.F_i : (\alpha \multimap \alpha) \multimap (\mathbb{I} \multimap \mathbb{I})$$

for $i = 1, 2$ and we write T to mean their type. f will be fed to one of these terms, after it has been decided which applies. The right one will be selected by the projection from the triple as follows ($i = 0, 1$):

$$f_i : (\alpha \multimap \alpha) \;\vdash\; \boldsymbol{\lambda}x.\langle x\,/p \otimes b \otimes c\,\rangle\,(p[T](\boldsymbol{\lambda}f.F_0 \otimes \boldsymbol{\lambda}f.F_1))f_i(p \otimes b \otimes c) : \mathbb{I} \multimap \mathbb{I}\,.$$

With these step functions we invoke the iteration principle to obtain a derivation of:

$$\text{BIN}, !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \;\vdash\; \S\mathbb{I}\,.$$

To complete the definition, the output must be synthesized from the triple. If the input was of even length, all we need is the third component, so

$$H_1 = \boldsymbol{\lambda}(p \otimes b \otimes c).c : H = \mathbb{I} \multimap (\alpha \multimap \alpha)$$

applied to the triple yields the correct result. Otherwise, the following transformation does the trick:

$$H_2 = \boldsymbol{\lambda}(p \otimes b \otimes c).\boldsymbol{\lambda}x.b(f_0(cx)) : H\,.$$

The first component—an appropriate projection—tells us which is the case. To finish the definition we cut the previous derivation with:

$$\begin{aligned} &f_0 : !(\alpha \multimap \alpha), x : \S\mathbb{I} \;\vdash \\ &\qquad \S\langle\, f_0/f_0', x/p \otimes b \otimes c\,\rangle((p[H](H_0 \otimes H_1))(p \otimes b \otimes c)) : \S(\alpha \multimap \alpha)\,, \end{aligned}$$

contract the two copies corresponding to f_0 and use lambda and universal abstractions to construct the final term **shift** : BIN $\multimap$ BIN.

Even-bits-shift

The term **shift** shifts the bits starting from the most significant one, but to find a translation for **e-shift**$(: n)$ we would like the procedure to begin with the least significant bit. To this end, we simply reverse the representation first, apply **shift** and reverse it again:

$$n : \text{ BIN } \vdash \textbf{e-shift} = \textbf{rev}(\textbf{shift}(\textbf{rev}\, n)) : \text{ BIN }.$$

Encoding poly-time Turing Machines

Turing Machines

W.l.o.g. we assume that both states and elements of the alphabet are represented by binary strings of even length L. Besides, we want the code of each state to end in 1 for a technical reason.

Tape and input

The contents of the tape as well as the state will be encoded as a single binary string whose L least significant correspond to the code of the state.

Suppose, the left and right parts of the tape (after the binary encoding of symbols) are $\ldots l_2 l_1 l_0$ and $r_0 r_1 r_2 \ldots$ respectively and the head is positioned above the symbol $r_0 \ldots r_{L-1}$. We shall represent the tape by a string which is an interleaving of (representations of) the left and right tapes e.g.

$$l_5 0 l_4 r_4 l_3 r_3 l_2 r_2 l_1 r_1 l_0 r_0$$

corresponds to

$$l_5 l_4 l_3 l_2 l_1 l_0 r_0 r_1 r_2 r_3 r_4 \,.$$

If one part of the tape is longer than the other, 0's can be used to make up for the shortfall. This is consistent with the fact that the tape is potentially infinite and that is why we insisted on representing the blank symbol with a string of zeros. When the state is appended at the front we get:

$$l_5 0 l_4 r_4 l_3 r_3 l_2 r_2 l_1 r_1 l_0 r_0 s_{L-1} \cdots s_0 \,.$$

In addition, we stipulate that the input and output strings always be placed to the right of the head and that the rest of the tape be left blank before the computation begins and after it terminates. Clearly, these conventions do not affect polynomial-time computability.

Transition

In order to make a step, the state and the head symbol (represented by $r_{L-1} \cdots r_0$) will have to be accessed i.e. the $3L$ least significant bits of our representation suffice to determine the next configuration.

First we are going to use $\mathbf{case}_{3L}$ to replace the bits corresponding to the current state with the bits representing the new state $s'_{L-1} \cdots s'_0$. If the head is to be moved to the right, we append one additional 0 to the representation. After this preprocessing we get either

$$\mathbf{tempL} = l_5 0 l_4 r_4 l_3 r_3 l_2 r_2 l_1 r_1 l_0 r_0 s'_{L-1} \cdots s'_0$$

or

$$\mathbf{tempR} = l_5 0 l_4 r_4 l_3 r_3 l_2 r_2 l_1 r_1 l_0 r_0 s'_{L-1} \cdots s'_0 0\,.$$

What other modifications must be made to the tape will become clear soon, when we look at how the representation of the tape should change once the head moves to the left or right. We consider shifts by one bit; note however that to simulate what the original Turing machine does, we will have to repeat such a step L times so that the head scans the first bit of the representation of the new 'true' cell. The table below shows how the representation ought to be modified, if the head changes its position.

	tape	representation + state
left	$l_5 l_4 l_3 l_2 l_1 \overline{l_0} r_0 r_1 r_2 r_3 r_4$	$r_4 l_5 r_3 l_4 r_2 l_3 r_1 l_2 r_0 l_1 l_0 s'_{L-1} \cdots s'_0$
right	$l_5 l_4 l_3 l_2 l_1 l_0 r_0 \overline{r_1} r_2 r_3 r_4$	$l_5 0 l_4 0 l_3 0 l_2 r_4 l_1 r_3 l_0 r_2 r_0 r_1 s'_{L-1} \cdots s'_0 0$

Observe that the changes to the tape resemble the effect of $\mathbf{e\text{-}shift}^2$:

$$\mathbf{e\text{-}shift}^2(:\mathbf{tempL}) \qquad 00 r_4 l_5 r_3 l_4 r_2 l_3 r_1 l_2 r_0 l_1 s'_{L-2} l_0 s'_{L-4} s'_{L-1} \cdots s'_0 s'_3 0 s'_1 0$$
$$\mathbf{e\text{-}shift}^2(:\mathbf{tempR}) \quad l_5 0 l_4 0 l_3 0 l_2 r_4 l_1 r_3 l_0 r_2 s'_{L-1} r_1 s_{L-3} r_0 s'_{L-5} s_{L-2} \cdots s'_1 s'_4 0 s'_2 0 s'_0 0$$

For the right shift this is due to the addition of the extra 0 bit (otherwise, the effect would be similar to '**odd-shift**'). We would still like to be able to read the direction, in which the representation 'moved', from the representation and that is why it was assumed at the beginning that s'_0 must be 1. Now we can simply look at the fifth bit from the right.

The results of $\mathbf{e\text{-}shift}^2$ differ from what should be modelled in the last $L+5$ bits. Hence, the correction can be made using $\mathbf{case}_{L+5}$. Let us call this correcting term $\mathbf{correct}(:u)$. The move of the head is then mimicked by L-fold application of $\mathbf{correct}(:\mathbf{e\text{-}shift}^2(:u))$ to **tempL** or **tempR**. Afterwards, we must not forget to erase the extra bit added for the move to the right. That case can be recognized by looking at the first bit. If it is 0, we should remove it (a single use of $\mathbf{case}_1$ does that). What results is a term $\mathbf{transition}(:u)$ which can be iterated $|n|$ times from some initial value $\mathbf{init}(:\mathbf{n})$ by $\mathbf{iterate}(n:\mathbf{init}(n:))$ (we can use **iterate** from the previous section on $BC^{\pm}$, because it was definable in BC^-).

Input conversion

But what is the value of **init**$(n :)$? The function **bint2tape**$(n :)$ is defined by safe recursion:

$$\begin{aligned} \textbf{bint2tape}(\epsilon :) &= \epsilon\,, \\ \textbf{bint2tape}(\mathsf{S}_i(x) :) &= \mathsf{S}_{w_i}(: \textbf{bint2tape}(x :))\,, \end{aligned}$$

where w_i $(i = 0, 1)$ is an interleaving of the representation of i with $\underbrace{0 \cdots 0}_{L}$. Assuming $s_{L-1} \cdots s_0$ represents the initial state, we set

$$\textbf{init}(: n) = \textbf{concat}(s_{L-1} \cdots s_0 : \textbf{bint2tape}(n :))\,.$$

Extraction of the result

To strip the first L bits describing the state we use P^L. After that, the bits corresponding to the right tape should be extracted i.e. every other bit must be ignored. The following function performs that by attaching a flag-bit to the intermediate result. When the flag is 1 we append the currently processed bit and change the flag to 0. Otherwise, we only replace it with 1 thus ignoring the corresponding bit. Because L is even, the base case is consistent with the case of 0.

$$\begin{aligned} \textbf{pick-right}(\epsilon :) &= 0\,, \\ \textbf{pick-right}(\mathsf{S}_i(x) :) &= \textbf{case}_1(: \textbf{pick-right}(x :))[0 : \mathsf{S}_1 \circ \mathsf{P} \mid 1 : \mathsf{S}_{i0} \circ \mathsf{P}]\,. \end{aligned}$$

To sum up, the following term provides the representation of the right tape, given the representation u of the whole tape:

$$\textbf{extract-right}(u :) = \mathsf{P}(: \textbf{pick-right}(\mathsf{P}^L(u :) :))\,.$$

Note that $\mathsf{P}(u :)$ must be used in the composition instead of the canonical $\mathsf{P}(: u)$ ($\mathsf{P}(u :)$ can be defined by composition with $\mathsf{P}(: u)$). It remains to use a similar trick to that from the definition of **result**$(u :)$ (of the previous section) to convert the representation into a binary output string.

3.7 Further directions

A different approach to completing BC^- is to embed it in a type theory and add suitable higher-type operators. Hofmann [55] has shown how BC can be embedded in the typed system SLR. The subsystem of SLR that corresponds to (and extends) BC^- is what we call SLR^-, which is SLR less the axiom (S-AX) (Remark 3.2.1, [55]), so that N is non-duplicable. It has been shown [99] that SLR^- is FP complete: closed SLR^--terms of type $\Box\mathsf{N} \to \mathsf{N}$ define exactly the FP functions $\mathbb{N} \to \mathbb{N}$.

$$\begin{tikzcd} BC \arrow[dr, dash] & SLR^{-} \arrow[d, dash] & BC^{\pm} \arrow[dl, dash] \\ & BC^{-} & \end{tikzcd}$$

In exploring FP completions of BC^- we have gone to IMLAL2 in search of constructs interpretable by IMLAL2 terms of safe arguments, which can then be used as step functions. The supply of such functions seems unlimited, and as the encoding of **perm-rec** and of **e-shift** shows, there are many IMLAL2 coding tricks one can exploit. For future work, we would like to understand how our notion of safe and normal variables in IMLAL2 may be extended to higher types. If successful, we can then investigate higher-order extensions of BC^-, using the methodology of interpretability in IMLAL2 to obtain FP-sound systems.

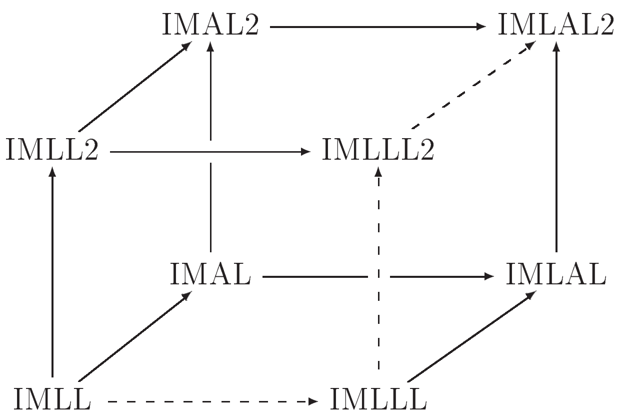
Chapter 4

Essential nets

Essential nets are directed graphs representing sequent-calculus proofs. They succeed at identifying proofs that are equal up to the commuting conversions defined by π- and σ-congruence. The treatment of those involving weakening is not entirely satisfactory and, to define the canonical forms, a procedure akin to garbage collection needs to be introduced even in the absence of cuts. Nevertheless, the sole ability to accommodate the multiplicative commutation rules allows the net-based syntax to factor out some of the redundant sequentiality present in the sequent calculus.

The idea underlying the construction of essential nets is the treatment of inference rules as building blocks for directed graphs. The identity axiom contributes an oriented link and other rules add nodes corresponding to the connectives they introduce. The orientation of new edges depends on what connective is being introduced and on which side of the sequent. In that way each derivation defines a directed graph. The properties that single out exactly the graphs arising from proofs will be the subject of this chapter.

Essential nets have been introduced by François Lamarche [80] as an intuitionistic variant of proof nets [39]. Lamarche's original paper concerned intuitionistic linear logic. Here we revisit the original nets for IMLL, give a new algorithmic proof of the correctness criterion. Our approach takes advantage of dominator trees and results in a sequentialization algorithm which has linear-time complexity; it can also be adapted to sequentialize classical MLL proofs with the same linear-time complexity [98]. Subsequently, we propose essential nets and correctness criteria for IMLLL, IMLLL2 and finally IMLAL2, but in fact our methods and definitions are immediately applicable to all vertices of the cube shown on the next page. There are three conceptual stages in our development: extensions to the second order (marked by vertical arrows), the addition of the light connectives (horizontal arrows) and 'weakenings' to the corresponding affine fragments (diagonal arrows). The path we decided to take is mapped out with dashed arrows.



To our knowledge, this is the first attempt to extend essential nets to these fragments. Proof nets for first-order MLL which extend the Danos-Régnier (DR) criterion [34] have already been proposed in [40], from which it is an easy step to derive a correctness criterion for MLL2 (i.e. second-order *classical* MLL). Although IMLL2 is a sublogic of MLL2, essential nets are quite a different kind of structure from proof nets; knowing the corresponding DR-criterion is not much of a help in finding a correctness criterion for essential nets. Besides, in the intuitionistic case, one would like to find a criterion which is easier to verify, does not use boxes, $\parr$-switches or the notion of jump for $\forall$-links. In the affine case, we propose a notion of canonical nets with respect to which the full and faithful completeness results for our game models will be established in Chapter 6. For that reason, we will focus on cut-free nets representing η-expanded proofs (i.e. the identity axiom can be applied to atoms only). However, the framework can be extended to handle proofs of any kind along with cut-elimination as we explain at the end.

4.1 IMLL

IMLL can be regarded as a polarized fragment of multiplicative linear logic (MLL), in which the two MLL connectives $\otimes$ and $\parr$ are allowed to occur with either positive or negative annotations and can be applied only if their arguments are appropriately polarized. This view originates from the Danos-Régnier system of polarities [91] and Bellin and van de Wiele's work [17, 18].

4.1.1 Polarization

For a start, we show how to transform an IMLL formula A to a ***positive polarized form*** $\ulcorner A \urcorner$ and a ***negative polarized form*** $\llcorner A \lrcorner$. The polarized forms are constructed from ***polarized atoms*** $(a^+, a^-, b^+, b^-, \cdots)$ and ***polarized connectives***

$(\otimes^+, \otimes^-, \parr^+, \parr^-)$ by mutual recursion:

$$\begin{array}{rclcrcl}
\ulcorner a \urcorner & = & a^+ & \qquad & \llcorner a \lrcorner & = & a^- \\
\ulcorner A \multimap B \urcorner & = & \llcorner A \lrcorner \parr^+ \ulcorner B \urcorner & & \llcorner A \multimap B \lrcorner & = & \ulcorner A \urcorner \otimes^- \llcorner B \lrcorner \\
\ulcorner A \otimes B \urcorner & = & \ulcorner A \urcorner \otimes^+ \ulcorner B \urcorner & & \llcorner A \otimes B \lrcorner & = & \llcorner A \lrcorner \parr^- \llcorner B \lrcorner
\end{array}$$

Note that $\otimes$ translates to $\otimes^+$ in a positive context and to $\parr^-$ otherwise. $\parr^+$ and $\otimes^-$ respectively correspond to $\multimap$. For example,

$$\begin{array}{rcl}
\ulcorner (b \otimes (c \multimap c)) \otimes a \urcorner & = & (b^+ \otimes^+ (c^- \parr^+ c^+)) \otimes^+ a^+ \,, \\
\llcorner f \otimes e \multimap (d \multimap a \otimes b) \lrcorner & = & (f^+ \otimes^+ e^+) \otimes^- (d^+ \otimes^- (a^- \parr^- b^-))\,.
\end{array}$$

For any unpolarized IMLL formula A, the polarized formulas $\ulcorner A \urcorner$ and $\llcorner A \lrcorner$ are dual to one another: $\ulcorner A \urcorner = \llcorner A \lrcorner^\perp$ and $\llcorner A \lrcorner = \ulcorner A \urcorner^\perp$ where the $A \mapsto A^\perp$ duality, corresponding to negation in *classical* linear logic, is defined by:

$$\begin{array}{rclcrcl}
(a^+)^\perp & = & a^- & \qquad & (a^-)^\perp & = & a^+ \\
(A \parr^+ B)^\perp & = & A^\perp \otimes^- B^\perp & & (A \parr^- B)^\perp & = & A^\perp \otimes^+ B^\perp \\
(A \otimes^+ B)^\perp & = & A^\perp \parr^- B^\perp & & (A \otimes^- B)^\perp & = & A^\perp \parr^+ B^\perp
\end{array}$$

It is easy to verify that $(A^\perp)^\perp = A$ for an arbitrary polarized formula A.

Analogously, one can translate IMLL sequents $C_1, \cdots, C_n \vdash A$ into one-sided polarized form $\Vdash \llcorner C_1 \lrcorner, \cdots, \llcorner C_n \lrcorner, \ulcorner A \urcorner$. The polarized counterparts always contain one positively polarized formula. For instance,

$$\begin{array}{l}
\Vdash e^- \parr^- (d^- \parr^- f^-),\;\; (f^+ \otimes^+ e^+) \otimes^- (d^+ \otimes^- (a^- \parr^- b^-)), \\
\qquad\qquad\qquad\qquad\qquad\qquad (b^+ \otimes^+ (c^- \parr^+ c^+)) \otimes^+ a^+
\end{array}$$

is a polarized version of

$$e \otimes (d \otimes f), f \otimes e \multimap (d \multimap a \otimes b) \;\vdash\; (b \otimes (c \multimap c)) \otimes a.$$

Following this pattern, one can also polarize all inference rules. We show them in Figure 4.1. We say that a formula is ***positive***, if it is a positive atom a^+ or its outermost connective is $\otimes^+$ or $\parr^+$; we shall write A^+ for such formulas. Similarly, ***negative formulas*** can be defined, which we write as A^-.

In the polarized logic formulas that are being connected must have suitable polarizations. For example, a new $\parr^+$ can only be introduced if its left argument is negative and the right one positive. The following table illustrates the relationship between the polarized rules and the standard IMLL rules:

IMLL rule	polarized rule
$\otimes$-r	$\otimes^+$
$\multimap$ -l	$\otimes^-$
$\multimap$ -r	$\parr^+$
$\otimes$-l	$\parr^-$

$$
\textbf{(atom)}\ \vdash a^-, a^+ \qquad\qquad \textbf{(exch)}\ \frac{\vdash \Gamma^-, A^-, B^-, \Delta^-, C^+}{\vdash \Gamma^-, B^-, A^-, \Delta^-, C^+}
$$

$$
(\otimes^+)\ \frac{\vdash \Delta^-, A^+ \qquad \vdash \Gamma^-, B^+}{\vdash \Delta^-, \Gamma^-, A^+ \otimes^+ B^+} \qquad (\otimes^-)\ \frac{\vdash \Delta^-, A^+ \qquad \vdash B^-, \Gamma^-, C^+}{\vdash \Delta^-, A^+ \otimes^- B^-, \Gamma^-, C^+}
$$

$$
(\parr^+)\ \frac{\vdash \Delta^-, A^-, B^+}{\vdash \Delta^-, A^- \parr^+ B^+} \qquad (\parr^-)\ \frac{\vdash A^-, B^-, \Delta^-, C^+}{\vdash A^- \parr^- B^-, \Delta^-, C^+}
$$

Γ^-, Δ^- are sets of negative formulas.

Figure 4.1: Polarized IMLL rules.

At first it may seem surprising that ($\otimes$**-r**) and ($\multimap$ **-l**) are related, but linear logic provides an explanation: they are just polarized versions of the same rule, namely, the classical tensor rule:

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

Indeed, in both two disjoint sequents are necessary. Similarly, the rules ($\otimes$-l) and ($\multimap$ -r) are instances of the $\parr$ rule:

$$
\frac{\vdash\ \Gamma, A, B}{\vdash\ \Gamma, A \parr B},
$$

but now two formulas from the same sequent are being combined. By an easy induction on the structure of derivations, we can obtain the following result:

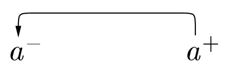
Proposition 4.1. For any IMLL formulas $C_1, \cdots, C_n$ and A, we have $C_1, \cdots, C_n \vdash A$ is IMLL-provable if and only if $\vdash \llcorner C_1 \lrcorner, \cdots, \llcorner C_n \lrcorner, \ulcorner A \urcorner$ is provable using the polarized rules.

4.1.2 Nets for polarized sequents

Essential nets are best explained in the polarized context. This is not a limitation: by Proposition 4.1 any fact about the polarized IMLL rules can easily be translated into the standard setting.

Definition 4.2. Essential nets ***for IMLL*** are directed graphs constructed from:

- horizontal links between negatively and positively polarized copies of atoms (called ***atomic nodes***) by orienting them from a^+ to a^-:



$$e \otimes (d \otimes f), f \otimes e \multimap (d \multimap a \otimes b) \ \vdash\ (b \otimes (c \multimap c)) \otimes a$$

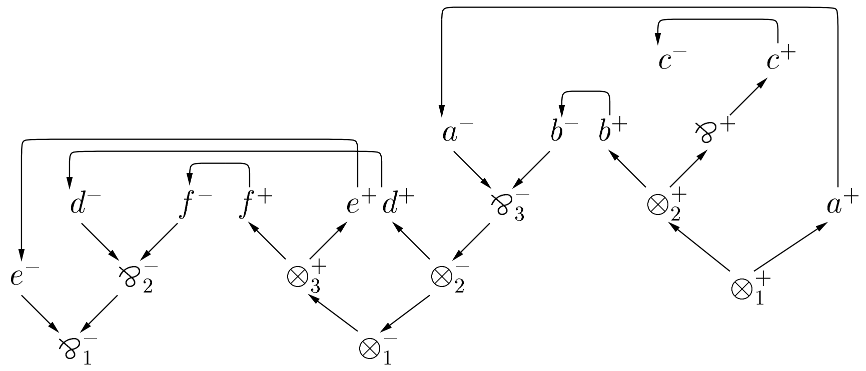
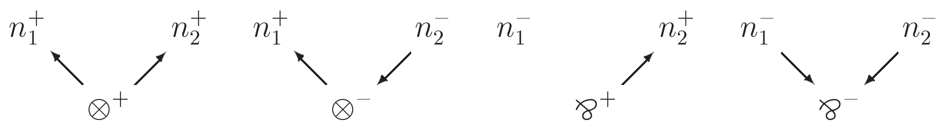


Figure 4.2: An IMLL essential net.

- links consisting of ***polarized nodes*** $\otimes^+, \otimes^-, \parr^+, \parr^-$ with directed edges drawn in according to the following rules:



The links will be referred to as $\otimes^+$-, $\otimes^-$-, $\parr^+$-, $\parr^-$-links respectively.

Note that the polarized nodes reflect the constraints present in the polarized IMLL rules. The nodes n_1^x and n_2^y ($x, y \in \{+,-\}$) are called ***premises*** of the links; the $\otimes^+$-,$\otimes^-$-, $\parr^+$-,$\parr^-$-nodes are their ***conclusions***—the conclusion of an x-link is always an x-node. By convention, atomic nodes are regarded as conclusions of the axiom link that connects them.

We require that each node be the premise of at most one link. Nodes which are not premises of any links will be called ***conclusions of the net***. Links whose conclusions are conclusions of the net are ***terminal***. Any node must be the conclusion of exactly one link, which we call its ***canonical link*** (or simply its link). We stipulate that exactly one conclusion—the ***root of the net*** —be annotated positively. Note that in a $\parr^+$-link there is no edge between the $\parr^+$-node and the negative left premise, which we call the ***sink*** of that $\parr^+$-node. If p is a $\parr^+$-node, we shall write s_p to denote its sink.

A node n is a ***hereditary premise of a link*** whose premises are n_1^x and n_2^y if and only if:

$$b \otimes e, e \multimap ((c \multimap d) \otimes a) \ \vdash \ (a \otimes b) \otimes (c \multimap d)$$

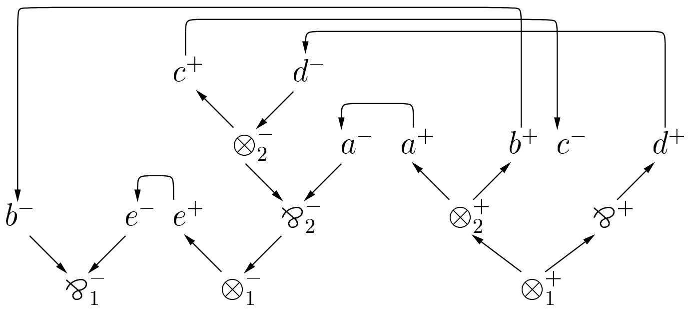


Figure 4.3: A correct IMLL essential net

- either $n = n_1^x$ or $n = n_2^y$,
- or n is a hereditary premise of n_1^x's or n_2^y's canonical link.

Each node n of an essential net defines a polarized IMLL formula $\phi(n)$ whose syntactic tree is simply the tree consisting of n and the hereditary premises of n's link: for atomic nodes $\phi(n) = n$ and for any other node $\Box$ we have $\phi(\Box) = \phi(n_1^x)\Box\phi(n_2^y)$ where n_1^x, n_2^y $(x, y \in \{+, -\})$ are the respective premises of $\Box$'s (canonical) link. The conclusions of an essential net naturally define a polarized sequent: the unique positive conclusion corresponds to the unique positive formula of the sequent, whereas the other conclusions represent the negative formulas. Thus, an essential net with conclusions $n_1^-, \cdots, n_k^-, n^+$ represents the sequent $\Vdash \phi(n_1^-), \cdots, \phi(n_k^-), \phi(n^+)$. Two nets along with associated sequents (in unpolarized form) are shown in Figures 4.2 and 4.3.

While each essential net defines a sequent via its conclusions, not all sequents can be represented in that way. In any case, in order to construct an essential net representing a sequent, it suffices to find a matching between the same atoms of opposite polarities. As we shall see later, this is possible for all provable sequents. The lack of a matching like this is a simple necessary condition for provability. For instance, we immediately see that neither $\Vdash a^-, a^-, a^+$ (equivalently $a, a \vdash a$) nor $\Vdash a^-, a^+ \otimes^+ a^+$ $(a \ \vdash \ a \otimes a)$ is provable. Nor is a variant of Peirce's law $(a \multimap b) \multimap a \ \vdash \ a$. Nevertheless, sequents which have corresponding essential nets can be unprovable. Figure 4.4 shows two such examples. Our aim will be to characterize the nets that do arise from proofs. Obviously the two nets just mentioned should then be classified as incorrect.

Definition 4.3. An essential net is ***correct*** if and only if:

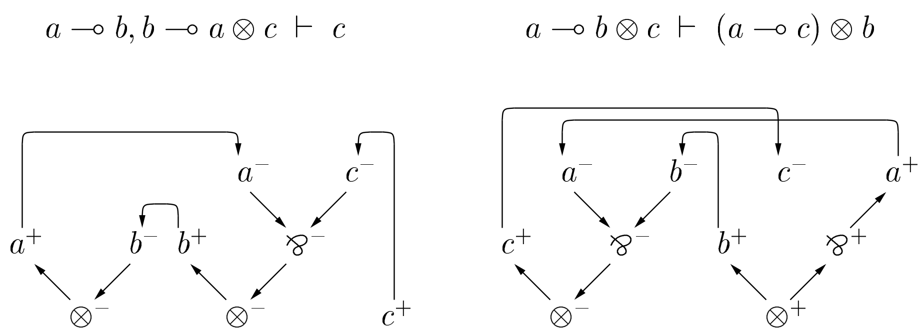


Figure 4.4: Incorrect IMLL essential nets.

(i) it is acyclic,

(ii) for any $\wp^+$-node p, each path from the root to s_p visits p.

Returning to our examples, the net on the left in Figure 4.4 does not satisfy Condition (i) and the right one breaks Condition (ii). The two nets in Figures 4.2,4.3 are correct.

Remark 4.4. In all acyclic essential nets, each node is reachable from the root. This is a consequence of the fact that backtracking from any node must end in the root, as it is the only node without an incoming edge. In particular, this is must be true of all correct nets (cf. Figures 4.2, 4.3).

4.1.3 Interpretation of proofs

Here we focus on how each sequent calculus proof in polarized form defines a correct essential net. The atomic axiom rule $\Vdash a^-, a^+$ is interpreted by an axiom link between a^+ and a^- and the introduction of a polarized connective is modelled by the addition of a link with the same name. The new node always links the two conclusions of the net(s) that represent the two arguments of the new connective.

For the $(\otimes^+)$ rule, the new link will connect the roots of the two essential nets corresponding to the sequents that are premises of that rule. For $(\otimes^-)$, we link the root of one net with the appropriate negative conclusion of the other. For the remaining two rules, the associated link will simply be added to the net representing the premise of the rule. As we do not assume any particular order on negative terminal links, sequents which are equivalent modulo the **(exch)** rule are represented by the same net.

Theorem 4.5 (Soundness). Each (polarized) IMLL sequent-calculus proof gives rise to a correct essential net.

Proof. We prove the theorem by induction on the structure of sequent-calculus derivations following the rules from Figure 4.1. Obviously, the identity axiom gives rise a correct essential net—a single axiom link. The exchange rule has no effect on the net.

The addition of a $⅋^-$-link clearly preserves (i) and (ii). A new $⅋^+$-node cannot create a cycle in a correct net, but we need to check (ii) for the new node. This is rather straightforward, as the condition then states that each path from the root to the sink of the root must visit the root.

Recall that the $(\otimes^+)$ rule is interpreted by linking the roots of two disjoint (correct) nets with a $\otimes^+$-node, which becomes the new root. By induction hypothesis, (i) and (ii) hold for the two initial graphs and this ensures that the composite graph will also satisfy them.

Finally, if there were any cycles after the $(\otimes^-)$-rule is applied, there would have to be an edge between the two graphs that are being connected. However, the rule assumes that they are disjoint. (ii) will also hold in the new net, because all paths from the root are either paths in one of the nets (then we can appeal to (ii) for that net) or they pass through the root of the other net and never leave it (then we use (ii) for the other net). $\square$

4.1.4 Sequentializability

Our next aim is to show the converse of the Soundness Theorem, namely, that correct essential nets can be *sequentialized* i.e. deconstructed in steps corresponding to IMLL inference rules. Accordingly, we define the notion of eliminability of nodes.

Definition 4.6 (Eliminable nodes). A link of an (IMLL) essential net is ***eliminable*** if and only if:

- it is terminal, and
- if it is a $\otimes$-link, its removal will result in two disjoint essential nets.

Definition 4.7 (Sequentializability).

- An essential net, which is an axiom link, is ***sequentializable***.
- If the removal of an eliminable link from a net yields sequentializable essential nets (by eliminability, one for a $⅋$-link and two for a $\otimes$-link), then the net is sequentializable.

Sequentializable nets can thus be reduced to axiom links by iterated removal of terminal links. A concrete order of decomposition leading to a set of axiom links for

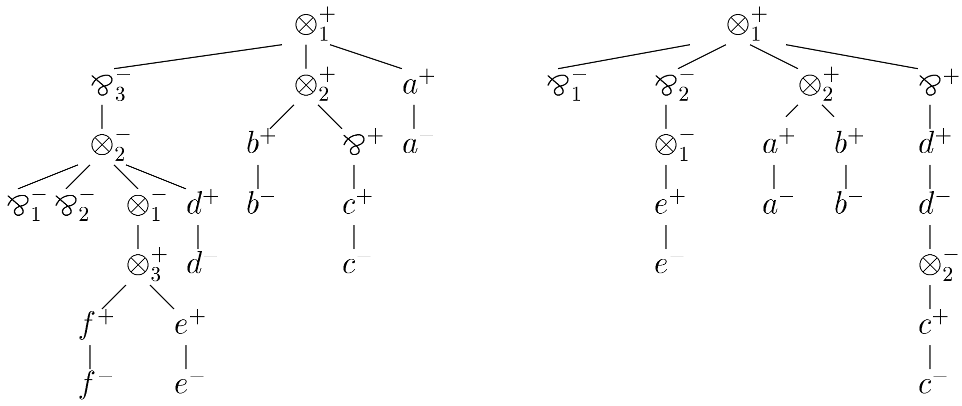


Figure 4.5: The dominator trees of essential nets in Figures 4.2, 4.3.

a given net will be called its ***sequentialization***. Because of the way eliminability of links is defined, it is easy to see that each sequentialization of an essential net corresponds to a sequent-calculus proof of the sequent represented by the net. The Soundness Theorem simply states that sequentializable essential nets are correct. Therefore, there can be no way of sequentializing the nets in Figure 4.4 as the corresponding sequents are not provable. In general, guessing an order in which a net could be sequentialized is not easy. The source of the difficulties are the $\otimes$-links, which are eliminable only if they split the net in two.

In the next section we present an algorithm which, given a correct essential net, produces its sequentialization. A prominent role in our analysis is played by the notion of the dominator tree of a graph. Its importance has already been observed by Lamarche, though his use of the ordering is different from ours.

4.1.5 Dominator trees

The nodes of an acyclic essential net $\mathcal{N}$ can be ordered as follows: we say that node v ***dominates*** w if and only if every path from the root to w passes through v. In view of Remark 4.4, this condition is never vacuous i.e. if $v \neq w$ there will always be a path to check. The ordering has a tree structure, which is called the ***dominator tree*** of $\mathcal{N}$ and will be denoted by $T_{\mathcal{N}}$. Note that $T_{\mathcal{N}}$ is induced by the edges $(idom(v), v)$, where v ranges over the non-root nodes of $\mathcal{N}$ and where $idom(v)$ is the *immediate dominator* of v i.e. $idom(v)$ dominates v and every other node that dominates v also dominates $idom(v)$. Hence, the root of the tree is the root of $\mathcal{N}$ and the parent of any other node in $T_{\mathcal{N}}$ is its immediate dominator.

Condition (ii) from the correctness criterion states that each $\parr^+$-node dominates its sink. This is indeed the case for the nets in Figures 4.2 and 4.3, whose dominator trees are shown in Figure 4.5. Dominator trees of acyclic essential nets exhibit the following structure:

- each positive occurrence of an atom is the immediate dominator of the corresponding negative one,
- a $\otimes^+$-node has exactly two positive successors (which are the premises of its canonical link) and a finite number of negative successors, each of which is a $⅋^-$-node,
- a $\otimes^-$-node has exactly one positive successor (the left premise of its canonical link) and a finite number of negative ones, only at most one of which can be a $\otimes^-$-node; besides, its immediate dominator is the right premise of its canonical link,
- a $⅋^+$-node always has one successor, which is the positive right premise of its canonical link,
- a $⅋^-$-node can have at most one successor, which must be a $\otimes^-$-node.

All these possibilities can be observed in Figure 4.5. The only nodes with multiple successors are the $\otimes$-nodes. Moreover, the parent of a $⅋^-$-node in $T_{\mathcal{N}}$ is the *nearest common ancestor* of the premises of that $⅋^-$-node. This is because the immediate dominator of a $⅋^-$-node will dominate both premises of the $⅋^-$-link. The following result generalizes this observation.

Lemma 4.8. In a correct essential net, if x dominates y (if y is negative we also require $x \neq y$), then x dominates both premises of y's canonical link.

Proof. If y is a $⅋^+$-node, we appeal to the second correctness condition stating that each sink of a $⅋^+$-node is dominated by that $⅋^+$-node. For the rest of cases it suffices to take advantage of the above remarks concerning the structure of dominator trees. □

Remark 4.9. For positive x Lemma 4.8 implies that all nodes dominated by x form a subnet of $\mathcal{N}$. That subnet is traditionally called the ***empire*** of x [39, 80].

The last lemma helps to specify the conditions under which $\otimes$-links become eliminable.

Proposition 4.10. A terminal $\otimes^+$-link with premises p_1, p_2 and conclusion p—which is then the root—is eliminable if and only if p has no negative children in $T_{\mathcal{N}}$ (equivalently, for $i = 1, 2$ any node reachable from p_i is dominated by p_i).

Proof. Surely eliminable $\otimes^+$-links meet the property. To prove the converse, we need to show that the net will be split, if we remove the $\otimes^+$-link. Consider the empires of p_1 and p_2. By Remark 4.4 and the assumption, each node of the net different from the root will be in one of the empires. Besides, the empires must be disjoint as p_1 is not dominated by p_2 or vice versa. This shows that the $\otimes^+$-link is eliminable. □

For instance, in Figure 4.2 the $\otimes_3^+$-link becomes eliminable, if the links $⅋_1^-$, $⅋_2^-$ and $\otimes_1^-$ are erased. In Figure 4.3 the $\otimes_1^+$-link would be eliminable if $⅋_1^-, \otimes_1^-, e^+$ and e^- were removed. A kind of dual property to that from Lemma 4.8 will help to characterize eliminable $\otimes^-$-links.

Lemma 4.11. In a correct essential net, if x dominates y, $x \neq y$ and y is a premise of l, then x also dominates l's conclusion, provided:

(i) x dominates each node reachable from x,

(ii) for each sink s_p reachable from x, x dominates p.

Proof. The condition $x \neq y$ is all we need for positive y. (i) ensures that the lemma holds if y is a premise of a $⅋^-$-link. No assumptions are necessary if y is the negative premise of a $\otimes^-$-node. (ii) is vital, if y is the sink of a $⅋^+$-node. □

It turns out that Lemma 4.11 specifies exactly the conditions under which $\otimes^-$-nodes can be eliminated.

Proposition 4.12. In a correct essential net, a terminal $\otimes^-$-link is eliminable, if its conclusion satisfies the two conditions of Lemma 4.11.

Proof. Let v, w be the left and right premise of the $\otimes^-$-link (call it l) in question. Clearly, if the link is eliminable, v is the root of a correct essential net, so both (i) and (ii) hold.

Suppose l satisfies (i), (ii) and consider v's empire. By Lemma 4.11 whenever the empire contains a hereditary premise of a terminal link, the conclusion of that terminal link will also be part of the empire. Consequently, no hereditary premise of the root's canonical link or w's canonical link can be present in v's empire. Therefore, the $\otimes^-$-node is eliminable. □

For example, in the first net from Figure 4.3 the $\otimes_1^-$-link becomes eliminable if $⅋_1^-$ and $⅋_2^-$ are erased. In the second net, $\otimes_1^-$ will be eliminable, provided $⅋_1^-$ is deleted. The $\otimes^-$-link in Figure 4.6 is not eliminable, because (ii) does not hold.

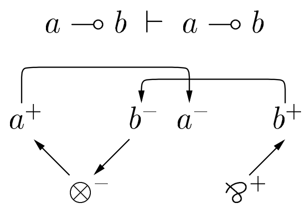


Figure 4.6: An uneliminable $\otimes^-$-link.

Remark 4.13. The removal of an eliminable link from a correct net results in one or two *correct* essential nets. Their dominator trees are subtrees of the dominator tree of the initial net. This means that even after successive eliminable links have been removed and the original net has been split into smaller correct nets, for two vertices v_1, v_2 belonging to one of the nets we have v_1 is the immediate dominator of v_2 if and only if the same holds in the original net.

4.1.6 A sequentialization algorithm

From now on, we shall focus on an ordered form of dominator trees. We require that the children of each node be linearly ordered subject to the following conditions:

- the negative children of a node are ordered topologically with respect to the essential net $\mathcal{N}$ i.e. $c_i < c_j$ if c_i is reachable from c_j in $\mathcal{N}$,
- if c is a negative node, then $c < d$ for any positive node d.

By remarks about the structure of dominator trees, these requirements are non-trivial only if the parent-node is a $\otimes$-node. A dominator tree satisfying the above properties will be called an ***ordered dominator tree***. For acyclic essential nets, each dominator tree can be converted into an ordered one, but there may be many ways of doing this. In particular, no order on positive children is assumed. In figures, if c_i and c_j are children of the same node, we shall draw c_i to the left of c_j we indicate that $c_i < c_j$. We have already followed this convention in Figure 4.5.

The goal of the rest of this section is to prove:

Theorem 4.14 (Lamarche). Correct essential nets are sequentializable.

We give a proof based on depth-first search of an ordered dominator tree of a correct essential net. The traversal will be guided by the order on the children of nodes: first all negative children of a node are visited in topological order, then the node is eliminated, and finally the procedure visits its positive children. The

Sequentialization Algorithm

Input: an ordered dominator tree $T_{\mathcal{N}}$ of a correct essential net $\mathcal{N}$
Output: a sequentialization of $\mathcal{N}$ given by a sequence of nodes

- Run $\boldsymbol{visit}(r^+)$, where r^+ is $\mathcal{N}$'s root:

$\boldsymbol{visit}(v : Vertex)$

1. call $\boldsymbol{visit}(n)$ for each negative $T_{\mathcal{N}}$-child of v according to the linear order between them
2. *ELIMINATE* v
3. run $\boldsymbol{visit}(p)$ for each positive $T_{\mathcal{N}}$ child of v

Figure 4.7:

algorithm is presented in Figure 4.7. To start off, we call $\boldsymbol{visit}(r^+)$ for the root r^+ of the net. The order in which *ELIMINATE* 'signals' are given will be the order in which links can be removed from the net. Observe that *ELIMINATE* a^- will always be followed by *ELIMINATE* a^+. This is to be treated as a cue to remove the axiom link.

Example 4.15. For the net in Figure 4.2 with the ordered dominator tree in Figure 4.5 the algorithm suggests elimination of links in the following order:

$$⅋_1^- \ ⅋_2^- \ \otimes_1^- \ \otimes_3^+ \ f^- f^+ e^- e^+ \ \otimes_2^- \ d^- d^+ \ ⅋_3^- \ \otimes_1^+ \ \otimes_2^+ \ b^- b^+ \ ⅋^+ \ c^- c^+ a^- a^+.$$

For that in Figure 4.3 *ELIMINATE* will be called successively for:

$$⅋_1^- \ \otimes_1^- \ e^- e^+ \ ⅋_2^- \ \otimes_1^+ \ \otimes_2^+ \ a^- a^+ b^- b^+ \ ⅋^+ \ \otimes_2^- c^- c^+ d^- d^+.$$

We shall prove the algorithm correct and show that by following the *ELIMINATE* instructions one obtains a sequentialization of the initial essential net. Thus we need to show that whenever *ELIMINATE* is called for some node n, all links of which n is a hereditary premise have already been eliminated. In addition, each link must be eliminable at the time of removal i.e. with respect to suitable nets. Let us begin by showing that the Algorithm processes nodes in topological order.

Lemma 4.16. If w is reachable from v in $\mathcal{N}$, then $\boldsymbol{visit}(w)$ terminates before $\boldsymbol{visit}(v)$ does.

Proof. It suffices to show that if (v, w) is an edge in $\mathcal{N}$, the lemma holds. If v dominates w, then obviously $\boldsymbol{visit}(w)$ will be completed before $\boldsymbol{visit}(v)$, because the algorithm performs a depth-first search of $T_{\mathcal{N}}$. If v does not dominate w, then w must be the conclusion of a $\otimes^-$-link (call it l). Let v' be the other premise of l and suppose a is the least common ancestor of v and v' in $T_{\mathcal{N}}$. Then we have $a \neq v$ (because v does not dominate w) so v is a's successor in $T_{\mathcal{N}}$ and w is a's child (by previous remarks about the structure of dominator trees) Consider the child c of a which dominates v (in some cases we may have $c = v$). If c is positive, then $\boldsymbol{visit}(w)$ will be finished before $\boldsymbol{visit}(c)$ is called, because w is negative. If c is negative, then w is reachable from c, so $w < c$ (recall that $<$ denotes the order on a's children in $T_{\mathcal{N}}$). Consequently, $\boldsymbol{visit}(w)$ will terminate even before $\boldsymbol{visit}(c)$ is called. As $\boldsymbol{visit}(v)$ is called no earlier than $\boldsymbol{visit}(c)$, the lemma holds. □

The Lemma also demonstrates that:

Proposition 4.17. If w is reachable from v, then

- either v dominates w,
- or $\boldsymbol{visit}(w)$ is completed before $\boldsymbol{visit}(v)$ is called.

As the algorithm is running, *ELIMINATE* is signalled once for each node, which is to be regarded as a cue to eliminate that node. The next result will help us show that indeed the nodes are then eliminable.

Lemma 4.18. When *ELIMINATE* v is called and v is a premise of a link l in $\mathcal{N}$, then *ELIMINATE* has already been called for l's conclusion w.

Proof. The lemma holds for positive v, because $idom(v) = w$ holds then and *ELIMINATE* w will therefore be signalled before $\boldsymbol{visit}(v)$ is called.

If v is the sink of a $\otimes^+$-node p, we obviously have $p = w$. Because the net is correct, w dominates v. Since the dominator tree is searched depth-first, w will have been visited and immediately eliminated before $\boldsymbol{visit}(v)$ is called (remember that w is a $\otimes^+$-node and has no negative children).

Otherwise, there is an edge from v to its conclusion w (which must be a negative node then). If v dominates w, then $v = idom(w)$. Hence $\boldsymbol{visit}(w)$ will be complete before *ELIMINATE* v is called, because the Algorithm calls $\boldsymbol{visit}$ for negative children of a node before eliminating it. If v does not dominate w, by Proposition 4.17 $\boldsymbol{visit}(w)$ is complete before $\boldsymbol{visit}(v)$ is finished, so the desired condition is also met. □

Repeated applications of the previous lemma allow us to deduce:

Proposition 4.19. If *ELIMINATE* v is signalled and v is a hereditary premise of l, then *ELIMINATE* must have been called for l's conclusion before.

At any given moment during runtime we can consider all the nodes of $\mathcal{N}$ which have not been eliminated yet. The nodes will be part of possibly several disconnected graphs. Suppose v has not been eliminated yet. Let us call the graph containing v ***its interim net***.

Theorem 4.20 (Correctness). The interim nets are all correct essential nets and when *ELIMINATE* v is called, v's link is eliminable with respect to v's interim net.

Proof. We give an inductive argument. By Proposition 4.19, when *ELIMINATE* v is called, v will be a conclusion of its interim net. At the beginning, the interim net is simply the initial correct net. Later, as we show that v's link is eliminable in its interim net, Remark 4.13 is all one needs for the inductive step. Hence we turn to eliminability.

If v is a $⅋$-node, v is eliminable as a conclusion.

If v is a $\otimes^+$-node and *ELIMINATE* v is called, v is the root of its interim net. By Remark 4.13 and the fact that the Algorithm visits negative children before signalling *ELIMINATE* v, v has no negative children in the dominator tree of its interim net and so, by Lemma 4.10, v is eliminable with respect to it.

If v is a $\otimes^-$-node and *ELIMINATE* v is called, by Proposition 4.17 all nodes of v's interim net reachable from v must be dominated by v, because otherwise ***visit*** would have terminated for them and they could not have been present in the net. Therefore, the first condition of Lemma 4.12 is satisfied. To see that the second condition also holds, suppose there is a sink s_p reachable from v. We have just proved that v dominates s_p. Because v's interim net is correct, p dominates s_p too. Because of the tree structure of the domination relation, either v dominates p which is what is required, or p dominates v. However, in the latter case, p will have been eliminated before ***visit***(v) is called, so p would not be present in the interim net of v. Thus, by Proposition 4.12, v is eliminable. □

Theorem 4.14 is now an easy corollary of the above result.

Our algorithm can be used for automatic synthesis of the sequent-calculus proof implied by the order in which the links are eliminated. To do that, one should simply run it backwards.

The extraction of the proof is trivial as long as $⅋$-links are being taken out. However, the way in which $\otimes$-links are processed also proves convenient:

$$
(!_0)\quad \frac{\vdash A^+}{\vdash !^+ A^+} \qquad\qquad (!)\quad \frac{\vdash A^-, B^+}{\vdash ?^- A^-, !^+ B^+}
$$

$$
(\S_0)\quad \frac{\vdash A^+}{\vdash \S^+ A} \qquad\qquad (\mathbf{c}^-)\quad \frac{\vdash ?^- A^-, ?^- A^-, \Delta^-, C^+}{\vdash ?^- A^-, \Delta^-, C^+}
$$

$$
(\S)\quad \frac{\vdash A_1^- \cdots, A_k^-, B_1^-, \cdots, B_l^-, C^+ \quad k+l>0}{\vdash ?^- A_1^-, \cdots, ?^- A_k^-, \S^- B_1^-, \cdots, \S^- B_l, \S^+ C^+}
$$

Figure 4.8: Polarized IMLLL rules (in addition to those in Figure 4.1).

- after a $\otimes^+$-link whose premises are p_1 and p_2 has been eliminated, the algorithm visits all nodes dominated by p_1 before going to those dominated by p_2 (or the other way round),
- after a $\otimes^-$-link whose positive premise is p has been eliminated, the algorithm proceeds to the nodes dominated by p first.

Therefore, for the $\otimes^+$-rule, when the algorithm is run backwards, it will first produce a complete proof of $\vdash \Delta^-, A^+$, before it even starts to construct that of $\vdash \Gamma^-, B^+$. Then the $\otimes^+$-link connecting A and B will be introduced. For $\otimes^-$-links, $\vdash B^-, \Gamma^-, C^+$ will be 'proved' first and only then will a proof of $\vdash \Gamma, A^+$ be constructed. For an alternative account of this approach and an extension to MLL the reader is invited to consult [98].

4.2 IMLLL

Like for IMLL one can consider polarized IMLLL inference rules (see Figure 4.8). The standard sequents $C_1, \cdots, C_n \vdash A$ have been translated into

$$
\vdash \llcorner C_1 \lrcorner, \cdots, \llcorner C_n \lrcorner, \ulcorner A \urcorner\ ,
$$

where[1]

$$
\begin{array}{rclcrcl}
\ulcorner !A \urcorner & = & !^+ \ulcorner A \urcorner & \qquad & \llcorner !A \lrcorner & = & ?^- \llcorner A \lrcorner \\
\ulcorner \S A \urcorner & = & \S^+ \ulcorner A \urcorner & & \llcorner \S A \lrcorner & = & \S^- \llcorner A \lrcorner
\end{array}
$$

We also extend the definition of negation:

$$
\begin{array}{rclcrcl}
(\S^+ A)^\perp & = & \S^- A^\perp & \qquad & (\S^- A)^\perp & = & \S^+ A^\perp \\
(!^+ A)^\perp & = & ?^- A^\perp & & (?^- A)^\perp & = & !^+ A^\perp
\end{array}
$$

[1]We follow the standard notation from (light) linear logic: ? is dual to ! and § is self-dual

Definition 4.21. In addition to the rules defining IMLL nets, five new links can be used to construct IMLLL essential nets:

- the first four are links with single premises, called the $!^+$-,$?^-$-,$\S^+$- and $\S^-$-links respectively:

$$\begin{tikzcd} n^+ & n^- \arrow[d] & n^+ & n^- \arrow[d] \\ !^+ \arrow[u] & ?^- & \S^+ \arrow[u] & \S^- \end{tikzcd}$$

- the fifth is a ***contraction link***:

$$\begin{tikzcd} n_1^- \arrow[dr] & & n_2^- \arrow[dl] \\ & \mathrm{C}^- & \end{tikzcd}$$

As before, ϕ assigns a corresponding IMLLL formula to each vertex of the net: we define $\phi(\Box^x) = \Box^x\phi(n^x)$ for $\Box^x \in \{\, !^+, ?^-, \S^+, \S^- \,\}$. For each C^--node $\phi(n_1^-) = \phi(n_2^-) = ?^-A^-$ *must* hold for some formula A^-. Then we define $\phi(\mathrm{C}^-) = ?^-A^-$.

It will be convenient to think of the conclusions of the first four links as brackets and we shall often refer to them as bracketing nodes and links respectively. The positive nodes $!^+, \S^+$ will be regarded as opening brackets which can be closed by the negative ones $(\S^-, ?^-)$. Bracketing links will be used to interpret introduction rules for the IMLLL connectives ! and §. Given an essential net, we write $\mathcal{B}^+$ (respectively $\mathcal{B}^-$) to denote the set of its positive (respectively negative) bracketing nodes. As before, an essential net with conclusions $n_1^-, \cdots, n_k^-, n^+$ represents the sequent

$$\vdash \phi(n_1^-), \cdots, \phi(n_k^-), \phi(n^+).$$

We are going to take advantage of the similarity of C^--links to $\otimes^-$-links and assign to an IMLLL essential net $\mathcal{N}$ a corresponding IMLL net $\mathcal{N}_{\mathrm{IMLL}}$. This can be done by replacing all C^--nodes with $\otimes^-$-nodes, and then treating bracketing nodes as 'transparent' i.e. for brackets which are not conclusions of the net, we perform the erasures below:

$$\begin{tikzcd} {} \arrow[d] & & {} \arrow[dd] \\ \Box^- \arrow[d] & \rightsquigarrow & \\ {} & & {} \end{tikzcd} \qquad\qquad \begin{tikzcd} {} & & {} \\ \Box^+ \arrow[u] & \rightsquigarrow & \\ {} \arrow[u] & & {} \arrow[uu] \end{tikzcd}$$

and delete all bracketing nodes which are conclusions of the net afterwards:

$$\begin{tikzcd} {} & \rightsquigarrow \\ \Box^+ \arrow[u] & \end{tikzcd} \qquad\qquad \begin{tikzcd} {} \arrow[d] & \rightsquigarrow \\ \Box^- & \end{tikzcd}$$

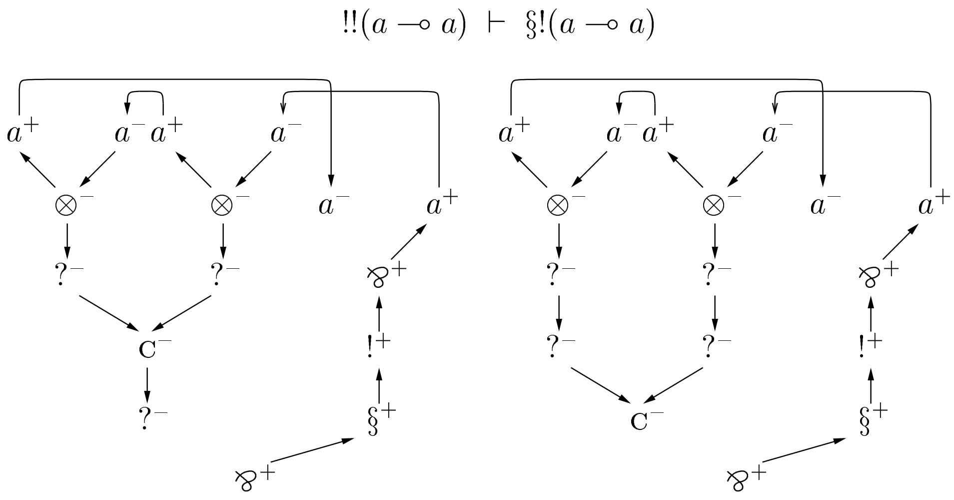


Figure 4.9: Two correct IMLLL essential nets of the same sequent.

The order in which the bracketing nodes are being removed is immaterial to the shape of the final net, so $\mathcal{N}_{\mathrm{IMLL}}$ is defined correctly.

A path p from the net (not necessarily starting from the root) is ***well-formed***, if all its initial subpaths contain at least as many as opening brackets as closing ones. Equivalently, this means that it is almost well-bracketed, but may still contain some unclosed opening brackets. Like in well-bracketed sequences, each closing bracket 'matches' the last opening bracket which has not yet been closed. Therefore each well-formed path defines a partial function $\delta_p : \mathcal{B}^- \rightharpoonup \mathcal{B}^+$, which assigns opening brackets to closing ones. Observe that each initial segment of a well-bracketed sequence is well-formed and in each final segment thereof the number of opening brackets cannot exceed that of closing ones. Well-bracketing plays a crucial role in our correctness criterion for IMLLL nets.

4.2.1 Correctness

Definition 4.22. An IMLLL essential net is correct if and only if:

(i-ii) hold (see Definition 4.3),

(iii) all paths from the root to a conclusion of the net and from $\parr^+$-nodes to their sinks are well-bracketed,

(iv) all paths from the root are well-formed and there exists a function $\delta : \mathcal{B}^- \longrightarrow \mathcal{B}^+$ such that for each such path p, we have $\delta_p = \delta \restriction dom\, \delta_p$,

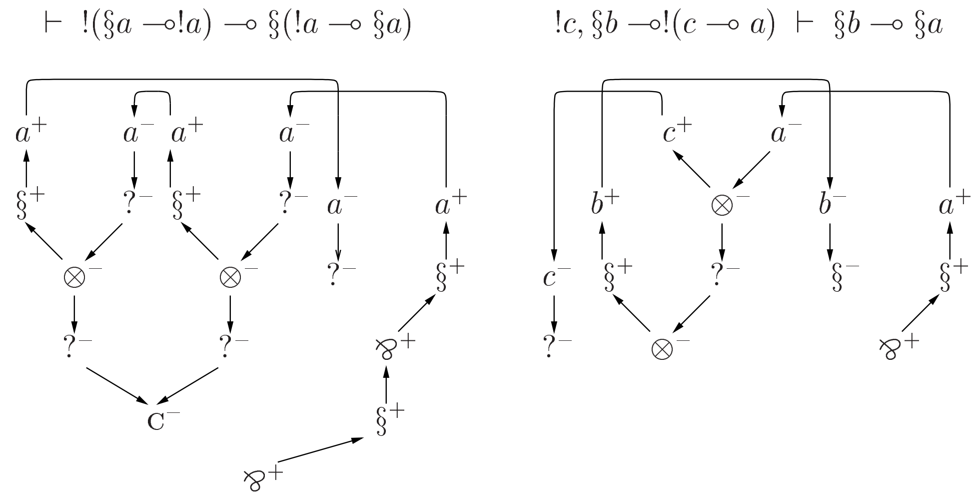


Figure 4.10: Correct IMLLL essential nets with associated sequents.

(v) if n^+ is a $!^+$-node and $\delta(n_1^-) = \delta(n_2^-) = n^+$, then $n_1^- = n_2^-$ and n_1^- is a $?^-$-node.

Figures 4.9 and 4.10 show several correct nets. Some incorrect essential nets breaking (iii),(iii)&(iv),(v) (first row) and (iii),(iv) (second row) are presented in Figure 4.11. Note that if an IMLLL essential net $\mathcal{N}$ is correct, $\mathcal{N}_{\text{IMLL}}$ is a correct IMLL net.

Remark 4.23. By (i) each node of a correct net is reachable from the root (by the same reasoning as in Remark 4.4). By (iv) each node is thus reachable via a well-formed path. Because this will also be true of each closing bracket, the function δ is unique for a given correct net.

It is possible to weaken (iv) by stipulating (iv') instead:

(iv') there exists a *partial* function $\delta : \mathcal{B}^- \rightharpoonup \mathcal{B}^+$ such that whenever (iii) requires that a path p be well-bracketed, we have $\delta_p = \delta \restriction \mathit{dom}\,\delta_p$.

To show the equivalence, we need the following fact.

Proposition 4.24. In an essential net satisfying (i-iii),(iv'),(v) each path p from the root is well-formed and $\delta_p = \delta \restriction \mathit{dom}\,\delta$ i.e. all of the conditions imply (iv).

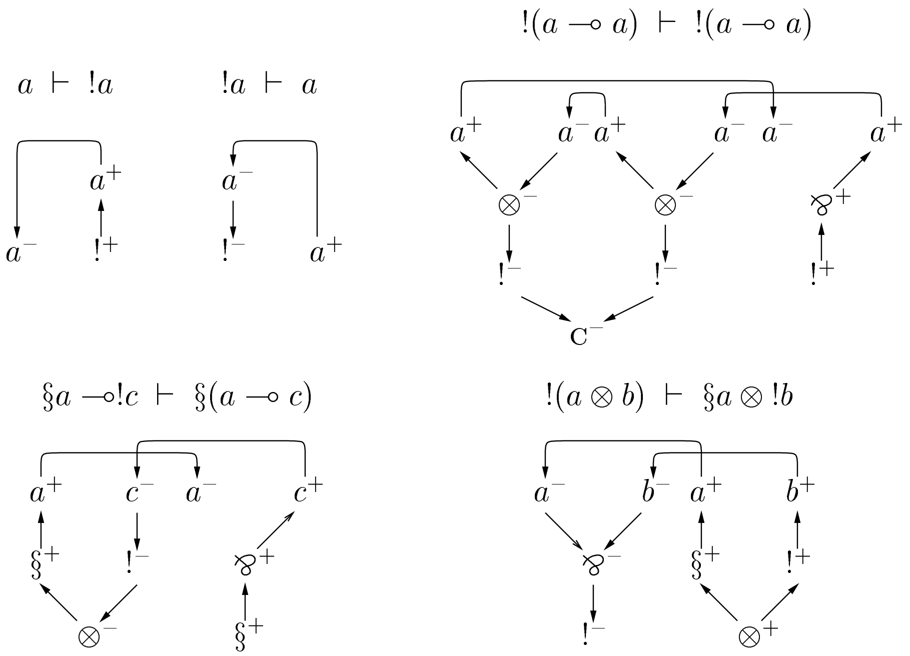


Figure 4.11: Incorrect IMLLL essential nets with associated sequents.

Proof. Suppose there exists a malformed path from the root. Consider the shortest such p i.e. $p = p'\Box^-$, where $\Box \in \{\, ?, \S \,\}$ and p' is well-formed. Then p' must be well-bracketed.

Observe that each path whose final node is negative can be extended (uniquely) with only negative nodes to a path ending in a conclusion of the net or the sink of a $⅋^+$-node (it suffices to follow the downward edges). Consider such an extension to p.

If a conclusion has been reached that way, by (iii) the extended path must be well-bracketed. This is impossible, as all initial subpaths of a well-bracketed path must be well-formed and p is not. If the extended path reaches the sink of a $⅋^+$-node, by (iii) its final fragment starting from the $⅋^+$-node is well-bracketed. However, the fragment consists of a final segment of p', the node $\Box^-$ and a number of negative nodes. p' is well-bracketed, so each of its final segments contains at least as many closing brackets as opening ones. Therefore, because of $\Box^-$, the supposedly well-bracketed sequence from the $⅋^+$-node to its sink will contain more closing brackets than opening ones at some point, which contradicts (iii).

In fact, we have just shown that each path ending with a closing bracket is part of a well-bracketed sequence of the kind mentioned in (iii). Therefore, we can indeed deduce (iv). $\Box$

4.2.2 Soundness and sequentialization

First we prove that the interpretation of logical rules consisting in adding links related to the connectives being introduced gives rise to correct nets.

Theorem 4.25 (Soundness). Each IMLLL cut-free derivation (based on rules in Figures 4.1 and 4.8) defines a correct essential net.

Proof. We use structural induction with respect to IMLLL proofs. For (i) and (ii) and the IMLL rules, we reason in the same way as in Theorem 4.5. It is clear that the additional IMLLL rules preserve (i) and (ii).

(iii) always follows by a straightforward appeal to (iii) for the nets corresponding to the premises of the logical rules.

To prove (iv) we note that for the $\otimes$-rules the function δ of the resultant net is a superposition of the δ functions characterizing the old nets. This is obvious for the $\otimes^+$-rule, but for the $\otimes^-$-rule we need to appeal to (iii) for justification. When the $!, \S$-rules are interpreted, the new negative bracketing nodes (if any) become associated with the positive one by (iii) for the old net, so (iv) will be satisfied by the new net and the new δ must be extended to take the new bracketing nodes into account.

(v) holds because it reflects the shape of the **(!)** rule and cannot be changed by any of the other rules (the same argument as for (iv)). $\square$

Finally, we prove the converse of the Soundness Theorem. In order to define sequentializability for IMLLL nets, we only need to clarify when bracketing and contraction nodes are eliminable.

Definition 4.26 (Eliminability). A $\square^+$-link is eliminable if and only if:

- it is terminal,
- all other conclusions of the net are of the shape $\square^-$,
- if $\square = !$, there is at most one other conclusion, which is then a $?^-$-node.

When eliminating a $\square^+$-node, one must also remove all the other conclusions. A C^--link is eliminable iff it is terminal.

Given that extension, we can adopt the standard definition of sequentializability (Definition 4.7). Finding a sequentialization of a net will then be equivalent to finding a proof of the associated sequent. It is easy to check that the removal of an eliminable link (for $\square^+$ in conjunction with the other links) from a correct net yields a correct net (for $\parr^+$- and $\otimes^-$-links (iii) will be essential).

Theorem 4.27 (Sequentialization). Each correct essential net for IMLLL is sequentializable.

Proof. We use induction on the number of nodes in the net. Obviously, an axiom link is sequentializable. For the rest of cases, we show how to find an eliminable link.

Consider the kind of node the root can be. It must be one of the following:

(1) $\parr^+$,

(2) $\otimes^+$,

(3) $\Box^+$ for $\Box \in \{\,!, \S\,\}$,

(4) a^+ for an atom a.

Case (1) is the easiest as the $\parr^+$-link is then eliminable. For (2) we perform transformations to ensure that when we appeal to the sequentialization theorem for IMLL, the existence of an eliminable link there will imply the same in our case. Finally, we will show how to reduce (3) and (4) to (2).

In Case (2) we transform the net by adding links between fresh atoms a^+ and a^- for each conclusion which is a $\Box^-$-node ($\Box \in \{\,\S, ?\,\}$): we replace the conclusion with

$$\begin{tikzcd} a^+ & & \Box^- \arrow[dl] \\ & \otimes^- \arrow[ul] & \end{tikzcd}$$

and in place of $\delta(\Box^-)$ we add

$$\begin{tikzcd} a^- & & \delta(\Box^-) \\ & \parr^+ \arrow[ur] & \end{tikzcd}$$

with a link from a^+ to a^-.

The modified net, which we shall call $\mathcal{M}$, is correct: the only conditions that do need verification are (ii) and (iii) for the new $\parr^+$-nodes. However, all paths from the root to a^- must pass through a^+ and hence through the associated $\otimes^-$- and $\Box^-$-nodes. If any of them violated (ii) or (iii), (iv) or (iii) respectively would be violated in the original net for a path ending in $\Box^-$. Thus, $\mathcal{M}$ is correct.

Let us now focus on the correct IMLL net $\mathcal{M}_{\mathrm{IMLL}}$. By Theorem 4.14 one of its terminal links is eliminable. This eliminable link cannot be any of the $\parr^+$- or $\otimes^-$-links we introduced: for $\parr^+$-links this follows from the fact that the root of the net is a $\otimes^+$-node, whereas if the $\otimes^-$-links were eliminable, a^- would have to be a conclusion, which it is not. Consequently, the eliminable link provided by the sequentialization theorem for IMLL will be eliminable in our case as well.

For (3) suppose $\phi^-(\{\Box^+\}) = \{\Box_1^-, \cdots, \Box_k^-\}$. If elements of this set are precisely the conclusions of the net, then $\Box^+$ is eliminable. If not, there exists $\Box_i^-$ which is not a conclusion. In order to take advantage of (2), we replace $\Box^+$ with

$$\begin{tikzcd} a^+ & & \Box^+ \\ & \otimes^+ \arrow[ul] \arrow[ur] & \end{tikzcd}$$

and $\Box_i^-$ with

$$\begin{tikzcd} a^- \arrow[dr] & & \Box_i^- \arrow[dl] \\ & \parr^- & \end{tikzcd}$$

adding a link from a^+ to a^-. Let us prove that the modified net is still correct. Certainly, it is acyclic as the new axiom links do not change the reachability relation between the vertices of the original net. By (iii),(iv) and (v) for the initial net, (iii),(iv) and (v) are still satisfied, because all paths from $\Box^+$ to each $\Box_i$ must be well-bracketed. Suppose (ii) does not hold. Since the original net was correct, the incorrect path t must involve the added link:

$$t = \otimes^+ a^+ a^- \parr^- \cdots s_p\,.$$

By Remark 4.23 there exists a path from $\Box^+$ to $\Box_i^-$. Let us replace a^+a^- in t with that path $\Box^+ \cdots \Box_i^-$ to get

$$t' = \otimes^+ \Box^+ \cdots \Box_i^- \parr^- \cdots s_p$$

t' satisfies (ii) as it is a path from the original net (to be precise, $\otimes^+$ and $\parr^-$ should be erased). Now, if p is not present in t, it must occur in t' between $\Box^+$ and $\Box_i^-$. This contradicts (iii) and (iv) for the original net as the path $p \cdots \Box_i^- \cdots s_p$ would not be well-bracketed (there is no opening bracket for $\Box_i^-$).

Note that the $\parr^-$-link introduced above cannot be eliminable, because $\Box_i^-$ was not a conclusion of the original net. Neither is the $\otimes^+$-link, because a^- is not a conclusion of the net. Therefore, when we appeal to (2), the eliminable link obtained from (2) will be present (and eliminable) in the original net.

In Case (4), if a^- is a conclusion, we have to do with an axiom link. Otherwise let us replace a^+ with

$$\begin{tikzcd} a^+ & & b^+ \\ & \otimes^+ \arrow[ul] \arrow[ur] & \end{tikzcd}$$

and a^- with

$$\begin{tikzcd}
a^- \arrow[dr] & & b^- \arrow[dl] \\
& \parr^- &
\end{tikzcd}$$

where b^+ and b^- are connected by a new axiom link. Note that the $\parr^-$-link will not be a conclusion of the net and hence it is not eliminable. Nor is the $\otimes^+$-link we introduced. Therefore, when we appeal to (2), the eliminable link provided by (2) will also be eliminable in our case and it will be a node of the original net. $\square$

Remark 4.28. The Soundness and Sequentialization theorems show that the notion of correct essential nets captures provability in IMLLL. However, a careful scrutiny of the preceding proof reveals that we have never used the assumption that the two premises of a contraction link must represent the same formulas. That is, we could consider essential nets without this condition, then their correct variants and still prove a sequentialization result! After all, C^--links are identical to $\parr^-$-links when the restriction is relaxed. The significant difference between such a result and Theorem 4.27 is that sequentializations of the 'liberal' nets no longer correspond to proofs in IMLLL. Nevertheless, the possibility of sequentializing the correct not necessarily essential nets will prove useful in our future considerations.

4.3 IMLLL2

In this section we extend our framework to handle second-order quantification. We begin by giving the expected polarizations of connectives

$$\ulcorner \forall X.A \urcorner \;=\; \forall^+ X.\ulcorner A \urcorner \qquad \llcorner \forall X.A \lrcorner \;=\; \exists^- X.\llcorner A \lrcorner$$

and rules for negating polarized sequences

$$\begin{array}{rcl} (X^+)^- &=& X^- \\ (\forall^+ X.A)^\perp &=& \exists^- X.A^\perp \end{array} \qquad\qquad \begin{array}{rcl} (X^-)^\perp &=& X^+ \\ (\exists^- X.A)^\perp &=& \forall^+ X.A^\perp \end{array}$$

For any formula A, $\mathrm{FV}(A)$ denotes the set of its free variables. The new polarized IMLLL2 rules are shown in Figure 4.12. and we hasten to explain how the substitution $A^-[T^+/X]$ is defined. Firstly, positive formulas will be used as substitutes (it does no harm to choose otherwise, but one has to be consistent). Then we set $X^+[T^+/X] = T^+$ and $X^-[T^+/X] = (T^+)^\perp$. In all other cases, substitution is simply propagated in the standard way. Moreover, to have an exact correspondence between terms and nets, we stipulate that in the $(\exists^-)$ rule, T^+ should be mentioned explicitly. If X occurs in A^-, we can recover it from $A^-[T^+/X]$ anyway, but for the sake of uniformity we will always specify it.

$$(\forall^+) \quad \frac{\vdash \Gamma^-, A^+}{\vdash \Gamma^-, \forall^+ X.A^+}\; X \notin \mathrm{FV}(\Gamma) \qquad\qquad (\exists^-) \quad \frac{\vdash \Gamma^-, A^-[T^+/X], B^+}{\vdash \Gamma^-, \exists^- X.A^-, B^+}$$

Figure 4.12: Polarized IMLLL2 rules (in addition to those in Figures 4.1, 4.8)

4.3.1 IMLLL2 nets

Definition 4.29. IMLLL2 essential nets are constructed from IMLLL links with the addition of

- axiom links for second-order variables:

$$\begin{tikzcd} X^- & X^+ \arrow[l, bend right] \end{tikzcd}$$

- $\forall^+$- and $\exists^-$-links:

$$\begin{tikzcd} n^+ & & n^- \arrow[d, "T^+"] \\ \forall^+_X \arrow[u] & & \exists^- X.A^- \end{tikzcd}$$

  X is called the ***eigenvariable*** of the $\forall^+$-link, and T^+ is the ***eigentype*** of the $\exists^-$-link. Note that the conclusion of a $\exists^-$-link is labelled with a formula.

The set of eigenvariables occurring in a net will be denoted by $\mathcal{E}$. Like for other kinds of essential nets, we define the formulas corresponding to nodes extending the previous definition with $\phi(\forall^+_X) = \forall^+ X.\phi(n^+)$ and $\phi(\exists^- X.A^-) = \exists^- X.A^-$. We require $\phi(n^-) = A^-[T^+/X]$ to hold (as well as the conditions on ϕ concerning contraction links, now up to α-equivalence). In addition:

(a) for any $X \in \mathcal{E}$ there must be exactly one $\forall^+_X$-node,

(b) if n is a conclusion, no eigenvariable may occur freely in $\phi(n)$, i.e. $\mathrm{FV}(\phi(n)) \cap \mathcal{E} = \varnothing$.

As usual, a net with conclusions $\{ n_1^-, \cdots, n_k^-, n^+ \}$ is to represent a proof of the sequent

$$\vdash \phi(n_1^-), \cdots, \phi(n_k^-), \phi(n^+)\,.$$

Condition (a) is introduced to remedy the usual problems with quantification: because identically named variables are used to represent binding, some mechanism is necessary to prevent variable capture. In the sequent calculus, one renames the conflicting variables as the need arises. In our nets we simply assume that different

$$\S(\forall V.!V), \forall V.\forall Z.!!(V \multimap Z) \;\vdash\; \forall X.\S(\forall Y.\S Y)$$

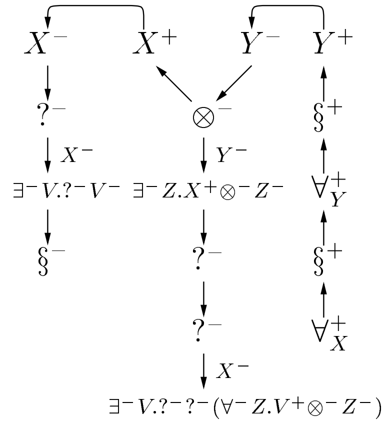


Figure 4.13: A correct IMLLL2 essential net.

variables are used in different instances of quantification. Condition (b) is also desirable, because its violation indicates an illegal use of the $(\forall^+)$ rule. Therefore, the very notion of an essential net already allows one to detect simple incorrect essential nets, e.g. for $X \vdash \forall X.X$. However, the definition still fails to catch many a forbidden application of the $(\forall^+)$ rule, after that has been obscured by subsequent uses of the $\exists^-$-rule: we give the obvious essential net for the (invalid) sequent $\vdash \forall Z.(a \otimes Z) \multimap (a \otimes \forall X.X)$ in Figure 4.15. Although it is an essential net, our correctness criterion will have to invalidate it.

Definition 4.30. An IMLLL2 essential net is correct if

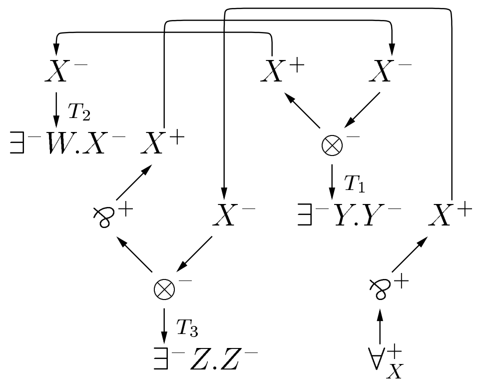
(i-v) are satisfied (see Definition 4.22),

(vi) any path from the root to the conclusion of a $\exists^-$-link whose eigentype contains a free eigenvariable X must pass through the $\forall^+_X$-node and the segment between the $\forall^+_X$-node and the conclusion must be well-formed.

Two correct nets are shown in Figures 4.13 and 4.14. The essential nets in Figure 4.15 do not satisfy the first and the second part of (vi) respectively. On the left, the $\exists^- Z.(a^- \invamp^- Z^-)$-node does not meet condition (vi): it is reachable from the root via a path does not go through the $\forall^+_X$-node.

Next we show the two expected results about the nets. The soundness result is shown relative to proofs that satisfy some conventions on the choice of names for second-order variables:

$$\forall Z.Z \vdash \forall X.(\forall Y.Y \multimap X)$$



where $T_1 = X^- ⅋^+ X^+$, $T_2 = X^+ \otimes^+ X^+$ and $T_3 = (\forall^+ W.X^+ \otimes^- X^-) ⅋^+ X^+$.

Figure 4.14: A correct IM(L)LL2 essential net.

- Whenever two proofs are being combined ($\otimes$-rules) and a variable X has a bound occurrence in one of them (i.e. it was bound by an application of the $(\forall^+)$ rule), it does not occur in the other.
- The $(\exists^-)$ rule must use a fresh variable for binding and T^+ may not contain free variables that have bound occurrences (this condition is important for vacuous uses of the rule, when T^+ has no connection with the previous proof).

It is not difficult to verify that each proof can be transformed into the required form by renaming (α-conversion) of variables if necessary.

Theorem 4.31 (Soundness). Each IMLLL2 cut-free derivation (subject to the above-mentioned constraints) defines a correct essential net.

Proof. It should be clear that the resultant net will always be an essential net, thanks to the additional requirements concerning proofs and the side condition of the $(\forall^+)$ rule.

For IMLLL rules and the preservation of the corresponding correctness criteria one can reason like in previous soundness proofs. To see that the IMLLL rules preserve (vi), it suffices to appeal to the convention on proofs and use the same arguments as previously for (ii).

The $(\forall^+)$ and $(\exists^-)$ rules are easily seen to preserve conditions (i)-(v). The $(\forall^+)$ rule also respects (vi), because the new $\forall^+$-node will be the root and (iv) held for the previous net. After the $(\exists^-)$-rule has been applied, there will be no additional variables for which to check (vi), so (vi) will be satisfied vacuously for the new eigentype. $\square$

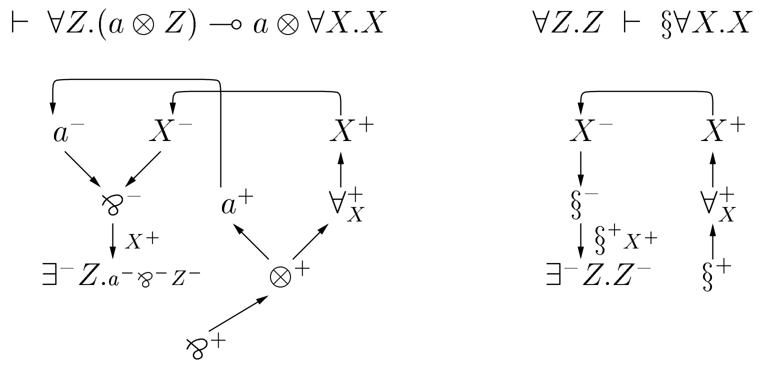


Figure 4.15: Incorrect essential nets violating (vi).

Sequentializability in the next theorem is determined by a new notion of eliminability of links in the standard way.

Definition 4.32 (Eliminability). A terminal $\forall^+$-link is eliminable. An $\exists^-$-link is eliminable if it is terminal and its eigentype does not contain any eigenvariables.

We say that an eigenvariable X is reachable from a link, if there exists a path from the conclusion of that link to an $\exists^-$-link whose eigentype contains X. A terminal $\otimes^-$-link l is eliminable, if and only if:

- its removal splits the net in two,
- whenever l's conclusion is reachable from some $\forall^+_X$-node, X is *not* reachable from l.

For other nodes, we adopt the previous definitions.

The definition of eliminable $\otimes^-$-links had to be extended to prevent second-order bindings from being broken (consider eliminating the terminal $\otimes^-$-link in Figure 4.16). To make sure the definition is adequate, i.e. enables inductive reasoning with respect to the size of correct nets, we need the following result.

Proposition 4.33. The removal of an eliminable link from an IMLLL2 essential net always yields correct essential nets.

Proof. This is largely self-evident except for $\otimes^-$- and $\exists^-$-links. For the former, we appeal to the additional eliminability condition. In the case of $\exists^-$-links all we need to show is that the new conclusion will have no eigenvariables. By definition, if a $\exists^-$-link is eliminable, no eigenvariables occur in its eigentype T^+ and, because it is also terminal, none occurs in its conclusion $\exists^- X.A^-$ either. Hence, $\mathcal{E} \cap \mathrm{FV}(A^-[T^+/X]) = \varnothing$ as required. $\square$

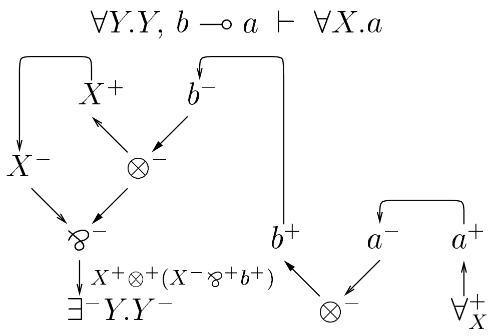


Figure 4.16: A correct net with an uneliminable terminal $\otimes^-$-link.

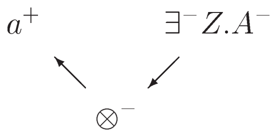
In the next proof, we take advantage of the Sequentialization Theorem for IMLLL. This will be possible, because if $\forall^+$- and $\exists^-$-links are ignored in a correct IMLLL2 essential net $\mathcal{N}$, and contraction links replaced with $⅋^-$-links[2], we obtain a correct IMLLL essential net $\mathcal{N}_{\mathrm{IMLLL}}$.

Theorem 4.34 (Sequentialization). Correct IMLLL2 essential nets are sequentializable.

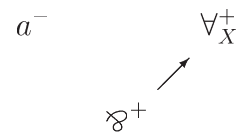
Proof. As before we show that any correct protonet $\mathcal{N}$ is either an axiom link or it contains an eliminable link.

Suppose $\mathcal{N}$ is not an axiom link, no terminal $\exists^-$-links are eliminable (so their eigentypes contain eigenvariables) and the root is not a $\forall^+_X$-node. We transform $\mathcal{N}$ into another correct IMLLL2 net in two stages:

- For each $\exists^- Z.A^-$-node which is a conclusion of the net, let us choose an eigenvariable X occurring in the associated eigentype. Then we replace the $\exists^- Z.A^-$-node with

  and plug

  in place of $\forall^+_X$ connecting a^+ and a^- with a new axiom link. The procedure is to be performed once for each such conclusion, though it may happen that

[2] When $\exists^-$-links are treated as transparent, the condition requiring that premises of contraction links be the same formulas may be broken.

the same $\forall^+_X$-node will be used several times. Of course, the result of the transformation is not unique and depends on the order in which the negative conclusions are processed and the choice of $\forall^+_X$-nodes. For our purpose, it suffices to consider any net that might result from the transformation. Clearly, none of its conclusions is a $\forall^+_X$- or an $\exists^- Z.A^-$-node.

- In the second stage, we process the terminal $\otimes^-$-links which are not eliminable for IMLLL2, but which would appear as such according to the definition of eliminability for IMLLL. Let us consider any conclusive $\otimes^-$-node n^-, which is reachable from some $\forall^+_X$-node and from which X can be reached. We modify it in the same way as the $\exists^-$-links above by inserting a $\otimes^-$- node below n^-, a $\wp^+$-node below the $\forall^+_X$-node in question and introducing the new axiom link.

Let us call the resultant net $\mathcal{N}'$. To prove $\mathcal{N}'$ correct, it suffices to check (ii) and (iii) for the new $\wp^+$-nodes and appeal to the correctness of $\mathcal{N}$ to confirm the other criteria.

If (ii) was violated in $\mathcal{N}'$, the first part of (vi) would be false in $\mathcal{N}$, so (ii) still holds. For (iii), let us fix one of the $\wp^+$-nodes, let c' be the conclusion of $\mathcal{N}'$ added in the same step, and let X be the associated variable. By the second half of (vi), any path from $\forall^+_X$ to c' is well-formed (either because it is of the kind mentioned in (vi) or because it can be extended to such a path). By (iii), the path is a final segment of a well-bracketed path, so it must well-bracketed as well, since it is already well-formed. Thus, $\mathcal{N}'$ is correct.

Observe that the fresh $\otimes^-$-nodes are not eliminable in $\mathcal{N}'$, as their removal would not split the net. The new $\wp^+$-nodes will not be conclusions of the net (since the positive conclusion of $\mathcal{N}$ was not a $\forall^+_X$-node), so they are not eliminable either. Consider the correct net $\mathcal{N}'_{\mathrm{IMLLL}}$. By Theorem 4.27, $\mathcal{N}'_{\mathrm{IMLLL}}$ contains an eliminable link (for IMLLL). The corresponding link in $\mathcal{N}$ will then be eliminable (for IMLLL2). $\square$

Condition (vi) stipulates domination of $\exists^- Z^-.A^-$-nodes by certain $\forall^+_X$-nodes. This dependency is generalized in the next theorem.

Theorem 4.35. In correct IMLLL2 essential nets each $\forall^+_X$-node dominates all nodes n such that $X \in \mathrm{FV}(\phi(n))$.

Proof. Suppose $X \in \mathrm{FV}(\phi(n))$. Because conclusions cannot contain eigenvariables, n must be a hereditary premise of the $\forall^+$-link or of some $\exists$–-link such that $X \notin \mathrm{FV}(\exists^- Z.A^-)$ and $X \in \mathrm{FV}(T^+)$. By (vi), such an $\exists^- Z.A^-$-node is dominated by the $\forall^+_X$-node. Therefore, by an analogue of Lemma 4.8, $\forall^+_X$ dominates n as well. $\square$

4.3.2 Protonets

Protonets are an abstraction of essential nets. They do not model provability in any of the logics we are interested in, but contain just enough information to make the correctness criteria expressible and meaningful. Our motivation for introducing protonets is of technical nature: affine essential nets can be analyzed smoothly, once the facts presented here are brought to light.

All sequentialization theorems for essential nets stated that some kind of directed graph can be deconstructed by successive eliminations of eliminable links. However, some features of the nets have not been used in the proofs, which merely showed that decomposition was possible with regard to some notion of eliminability. For instance, for IMLLL, we have not taken advantage of the requirement that the premises of a C^--link must represent identical formulas of the shape $?^-A^-$ (see Remark 4.28). A C^--link was deemed eliminable as soon as it became a conclusion of the net so, as far as eliminability is concerned, it was treated in the same way as a $⅋$-node. A similar comment applies to $\exists^-$-links. To achieve a precise correspondence with proofs, we had to label the conclusion with an existential formula $\exists^- X.A^-$, specify the eigentype T^+ such that the premise represented $A^-[T^+/X]$. But later that constraint did not play any role in the sequentialization proof, although it was crucial to the correspondence between sequentializable nets and proofs. Finally, the very formulas occurring as conclusions of $\exists^-$-links could be dispensed with as well, because the correctness criteria only refer to their free variables. Hence, we could well consider the following $\Diamond^-$-link instead:

$$\begin{array}{c} n^- \\ \downarrow{\scriptstyle W} \\ \Diamond^-_V \end{array}$$

where V and W are sets of variables. The $\Diamond^-$link is to describe a rather general form of manipulation on the set of second-order variables of formulas, in particular hiding through existential quantification. The set V represents the variables of the resultant formula (which we could write as $\Diamond^-_V(A^-)$). W contains the variables that may have been hidden by using $\Diamond^-_V$. Therefore, the free variables associated with n^- should be contained in $V \cup W$. $\exists^-$-links can be translated into the new framework in the following way:

$$\begin{array}{c} n^- \\ \downarrow T^+ \\ \exists^- X.A^- \end{array} \quad \rightsquigarrow \quad \begin{array}{c} n^- \\ \downarrow \mathrm{FV}(T^+) \\ \Diamond^-_{\mathrm{FV}(\exists^- X.A^-)} \end{array}$$

The additional condition relating the variable content of n^- and $V \cup W$ simply generalizes

$$\mathrm{FV}(A^-[T^+/X]) \subseteq \mathrm{FV}(T^+) \cup \mathrm{FV}(\forall^+ X.A^+)\,.$$

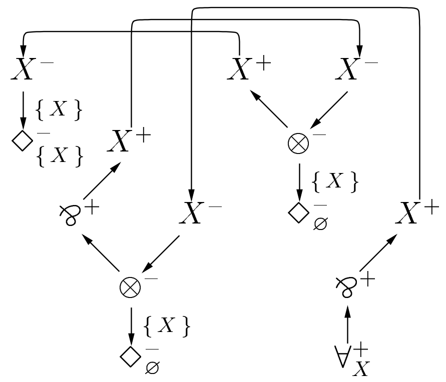


Figure 4.17: A correct protonet.

Definition 4.36. ***Protonets*** are constructed from IMLLL2 links with the exception of $\exists^-$-links, but with the addition of $\Diamond^-$-links:

$$\begin{array}{c} n^- \\ \Big\downarrow W \\ \Diamond^-_V \end{array}$$

where V, W are sets of second-order variables. W is called the ***eigentype*** of the $\Diamond^-$-link. The set of eigenvariables of a protonet will be denoted by $\mathcal{E}$ as before. For each node n of the net, we define the set of variables $\nu(n)$ associated with a given node n as follows:

$$\begin{array}{rclcrcl} \nu(a^+) & = & \nu(a^-) = \varnothing & \qquad & \nu(X^+) & = & \nu(X^-) = \{ X \} \\ \nu(\forall^+_X) & = & \nu(n^+) \setminus \{ X \} & & \nu(\Diamond^-_V) & = & V \end{array}$$

For other links with conclusion n, we set $\nu(n) = \nu(n_1) \cup \nu(n_2)$ or $\nu(n) = \nu(n_1)$ depending on the number of premises (n_1 and possibly n_2 are the premises). Protonets must satisfy the following ***healthiness*** conditions:

- the eigenvariables of $\forall^+$-links must be distinct,
- if n is a conclusion, then $\nu(n) \cap \mathcal{E} = \varnothing$,
- for each $\Diamond^-$-link $\nu(n^-) \subseteq V \cup W$.

Correct protonets are defined by analogy with the correctness of essential nets:

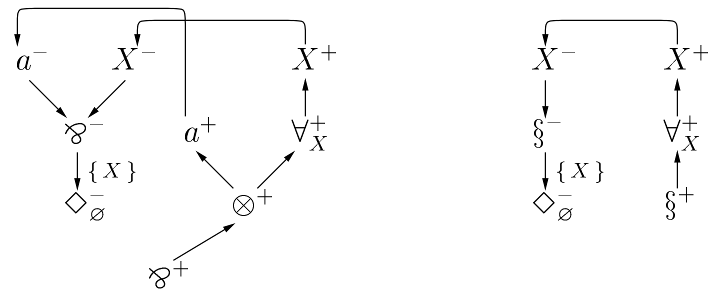


Figure 4.18: Incorrect protonets violating (vi).

Definition 4.37. A protonet is said to be correct just in case:

(i-v) from Definition 4.22 hold,

(vi) any path to from the root to the conclusion of a $\Diamond^-$-link whose eigentype contains a free eigenvariable X passes through the $\forall_X^+$-node and the path between the $\forall_X^+$-node and the conclusion must be well-formed.

A correct protonet is shown in Figure 4.17. Two incorrect ones appear in Figure 4.18. All of them are translations of essential nets. Eliminability of $\Diamond^-$-links is defined to make them compatible with $\exists^-$-links.

Definition 4.38 (Eliminability). A terminal $\Diamond^-$-link is eliminable just in case $W \cap \mathcal{E} = \varnothing$. We adopt previous definitions for other nodes.

The healthiness conditions ensure that protonets are a proper generalization of essential nets: an essential net is sequentializable if and only if the corresponding protonet, arising by the translation of $\exists^-$-links into $\Diamond^-$-links, is sequentializable. Similarly, removals of eliminable links do not break the correctness of protonets, which enables inductive arguments on the size of a net. We only check that this assertion is true for the new links, because the reasoning is the same as before for the rest of links, and we will see the healthiness conditions at work. When a $\Diamond^-$-link is eliminable, in addition to $W \cap \mathcal{E} = \varnothing$, we also have $V \cap \mathcal{E} = \varnothing$, because it is terminal. Thus, $(V \cup W) \cap \mathcal{E} = \varnothing$. Now, by the third healthiness condition, $\nu(n^-) \cap \mathcal{E} = \varnothing$ holds, so n^- will be a 'healthy' conclusion of the net, once the eliminable link is thrown away.

Remark 4.39. We could go on and define a 'logic' that underlies protonets. The polarized rule for $\Diamond_V^-$ would be:

$$\frac{\vdash A^-, \Gamma^-, B^+}{\vdash \Diamond_V^-(A^-), \Gamma^-, B^+} W$$

$$b, !a \vdash \S a$$

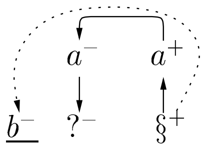


Figure 4.19: A correct affine net.

where $\mathrm{FV}(\Diamond^{-}_{V}(A)) = V$ by definition and $\mathrm{FV}(A) \subseteq V \cup W$. Instead of contraction, we would then consider a new connective C^{-} with a rule identical to the $⅋^{-}$-rule. Then both soundness and sequentializability could be proved and sequentializability would correspond to being a proof in the calculus. Of course, we make no claim that the logic is meaningful at all. What matters to us is that correct protonets are sequentializable, because the proof for IMLLL2 can be repeated for protonets only with some cosmetic changes.

4.4 IMLAL2

This is the last stage of our programme to develop essential nets for IMLAL2. We are going to tackle the polarized version of the weakening rule:

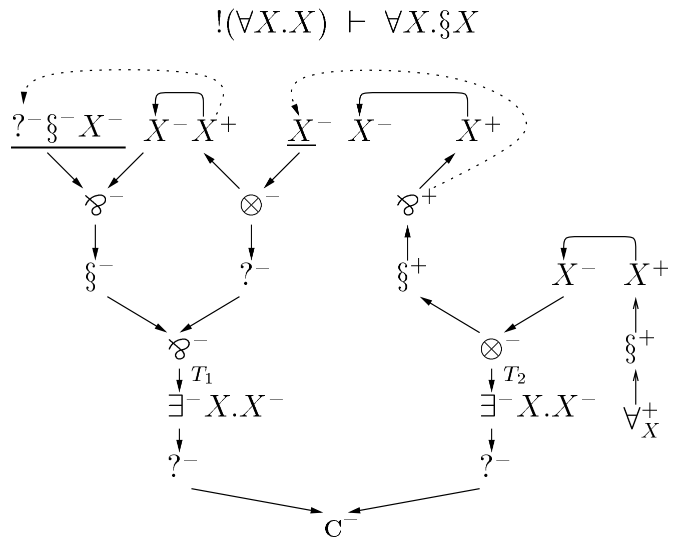
$$\textbf{(weak)} \quad \frac{\vdash \Gamma^{-}, A^{+}}{\vdash W^{-}, \Gamma^{-}, A^{+}}$$

Definition 4.40. Essential nets for IMLAL2 (***affine nets***) are built from IMLLL2 links with the addition of ***weakening links***:

$$\underline{W^{-}}$$

Weakening links have no premises—there is just one conclusion, called a weakening node. We set $\phi(\underline{W^{-}}) = W^{-}$. The nets are subject to all constraints concerning ϕ (for contraction and $\exists^{-}$-links).

Obviously, the affine extension applies not only to IMLLL2, but also to its sublogics IMLL, IMLL2 and IMLLL.



where $T_1 = (!^+\S^+X^+ \otimes^+ X^+)\otimes^+!^+(X^- \parr^+ X^+)$, $T_2 = \S^-(X^+ \otimes^- X^-) \parr^+ X^+$

Figure 4.20: A correct affine net.

4.4.1 Correctness

The correctness criteria we have considered so far use graph-theoretic properties that could also be verified if new edges were added to the net (not necessarily following the construction rules of essential nets). We will say that a directed edge ***strengthens*** a weakening node, if it leads from some positive node to this node. We decree that whenever a path uses the strengthening edge, the positive node which is the source of that edge is *not* counted as visited, as if there were two options: to visit the node, or to circumvent it using the strengthening. For instance, in Figure 4.19 the $\S^+$-node will be ignored.

Definition 4.41. An affine net is correct if it is possible to strengthen each weakening node once so that the resultant graph satisfies all the correctness criteria for IMLLL2 essential nets.

Two correct affine net is shown in Figures 4.19 and 4.20. To relate affine nets with protonets and put them on a more formal basis, we observe that:

Proposition 4.42. An affine essential net is correct if and only if it can be transformed to a correct protonet by the following modifications:

$f, e \vdash (a \multimap e) \otimes (a \multimap b) \multimap f$ $\qquad$ $(a \otimes b \multimap b) \multimap c \vdash a$

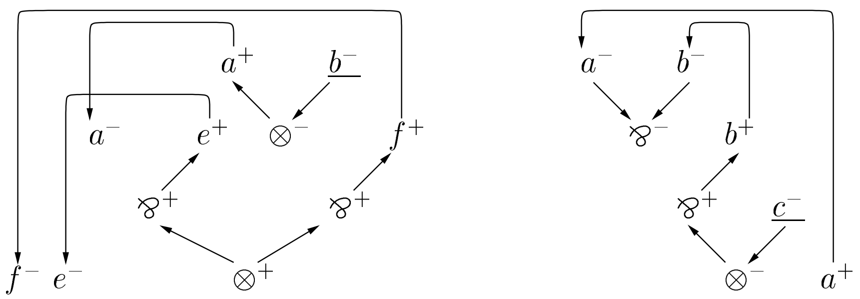


Figure 4.21: Incorrect IMAL nets.

- each weakening node $\underline{W^-}$ is replaced with

$$\begin{tikzcd} a^- \arrow[d, "\varnothing"] \\ \Diamond^-_{\mathrm{FV}(W^-)} \end{tikzcd}$$

where a^- is part of a new axiom link leading from a^+ attached to some positive node n^+ as shown below

$$\begin{tikzcd} a^+ & & n^+ \\ & \otimes^+ \arrow[ul] \arrow[ur] & \end{tikzcd}$$

- each $\exists^-$-link is replaced with the corresponding $\Diamond^-$-link.

The correct protonet that witnesses the correctness of an affine net will be called its ***extension***. Observe that for each node n of an affine net $\mathrm{FV}(\phi(n))$ is the same as $\nu(n')$, where n' is the corresponding node in its extension (e.g. $\Diamond^-_{\mathrm{FV}(W)}$ corresponds to $\underline{W^-}$). Therefore, an extension of an affine net has the same variable content as the original net.

Incorrect nets, like those shown in Figure 4.21, will not have any extensions. In Figures, instead of adding the dummy $\otimes^+$-nodes, we have simply drawn the additional links directly from n^+ with dotted lines. Note that it may be necessary to have several links from a single node n^+. As the proof of the next theorem shows, the strengthening edges indicate the moment when the weakening nodes could be introduced.

Theorem 4.43. Each cut-free derivation in IMLAL2 gives rise to a correct affine net.

Proof. To each sequent calculus derivation we assign a pair $(\mathcal{A}, \mathcal{P})$ where $\mathcal{A}$ is an affine net and $\mathcal{P}$ is its extension:

- all rules apart from **(weak)** and $(\exists^-)$ are interpreted as for IMLLL2 both for $\mathcal{A}$ and $\mathcal{P}$,
- $(\exists^-)$ is interpreted by a new $\exists^-$-link in $\mathcal{A}$ and the corresponding $\Diamond^-$-link in $\mathcal{P}$,
- **(weak)** is interpreted by an addition of the weakening node $\underline{W}^-$ to $\mathcal{A}$. The associated extension arises from $\mathcal{P}$ by modelling:

$$\dfrac{\dfrac{\vdash a^-, a^+ \quad \vdash \Gamma^-, B^+}{\vdash a^-, \Gamma^-, a^+ \otimes^+ B^+}}{\vdash \Diamond_{\mathrm{FV}(W)} a^-, \Gamma^-, a^+ \otimes^+ B^+}$$

  instead of the weakening rule. Here we need to assume that no variable from $\mathrm{FV}(W^-)$ has been used for universal quantification in $\vdash \Gamma^-, B^+$ (this is enforceable by α-conversion).

By Remark 4.39, $\mathcal{P}$ is always a correct protonet. $\square$

To optimize the verification of correctness in affine nets, it is worth understanding that some strengthening edges are more 'canonical' than others. The weakening rule commutes with any non-axiom IMAL2 rule, which makes it possible to 'push' its applications upwards in proofs. In IMAL2 this results in all weakenings being performed immediately after the identity rule i.e. a strengthening edge from the positive end of an axiom link can always be found. Unfortunately, this is no longer true for IMLAL2, because in general weakening cannot be permuted with the rules for § and !. However, this means that one can always find a strengthening edge from either an atomic node a^+, a $\S^+$-node or a $!^+$-node. Later we propose a notion of normal form for affine nets, which is based on a dual procedure that delays weakening i.e. pushes it downwards. The significant difference between the two is that the latter is a deterministic operation.

A necessary correctness condition

Correct protonets satisfy the correctness condition (ii). As an easy consequence, we show that correct affine nets must meet this property too.

Proposition 4.44. In correct affine nets each path from the root to the sink of a $\invamp^+$-node passes through the $\invamp^+$-node.

Proof. We reason by contradiction. Suppose there is a path from the root to the sink of a $\wp^+$-node that fails to visit the $\wp^+$-node. Consider any extension of the affine net. By definition, it must satisfy condition (ii). Yet, the path still exists in the extension, so the initial affine net could not be correct. □

By Proposition 4.44, the right affine net in Figure 4.21 is obviously incorrect.

4.4.2 Sequentialization

Before stating the theorem, we need to extend our definition of sequentializability by specifying when weakening links can be eliminated. This must be done to ensure that sequentializability corresponds to the existence of an IMLAL2 proof generating the net. The case of weakening links is rather simple.

Definition 4.45 (Eliminability). A terminal weakening link is eliminable. Definition 4.32 applies to the rest of links.

Theorem 4.46. Correct affine nets are sequentializable.

Proof. As before, when an eliminable link is taken out from a correct affine net, correct affine nets arise. Hence, we can prove the theorem by showing that any correct net is either an axiom link or one of its terminal links is eliminable.

Consider an extension of a correct affine net. By Remark 4.39, the extension is an axiom link (then the affine net must have been the same axiom link) or it has an eliminable link (with respect to the eliminability of links for protonets).

- If the eliminable link is a $\Diamond^-$-link corresponding to some $\exists^-$-link in the original affine net, then that $\exists^-$-link is eliminable in the affine net (recall that the ν-values for both are the same and the same variables occur in their eigentypes).
- If the eliminable link is a $\Diamond^-$-link corresponding to a weakening node $\underline{W^-}$ in the original net, the weakening node will then be eliminable in the original net.
- If a $\otimes^+$-node that has been added in connection with some weakening node is eliminable, we first consider the correct protonet whose root is a^+.
  - If it is not an axiom link, it will at least contain the $\Diamond^-_{\mathrm{FV}(W^-)}$-node corresponding to the weakening node in question. Therefore, when we subject the net to the same kind of case analysis, an eliminable link will be found in the next round.

  - If it is an axiom link, we ignore it and consider the other protonet resulting from the split (with root n^+). By identical reasoning an eliminable link will be identified in a finite number of steps.

- If the eliminable link is none of the above, it will have a corresponding copy in the original net, which will also be eliminable (if the removal of a node splits the extension, it will surely split the original affine net).

□

4.4.3 Collected and canonical nets

In essential nets for linear logics (without weakening) each node was reachable from the root. The **(weak)** rule no longer preserves this condition and introduces nodes that are unreachable. If a weakening link participates in other rules, new nodes—not necessarily weakening nodes—will become detached from the root. This is a manifestation of the fact that affine nets do not provide 'spontaneous' garbage collection. In what follows we aim to identify a notion of affine nets in which only weakening nodes are unreachable from the root. Nets like these come from proofs in which the use of weakening is delayed for 'as long as possible'.

Definition 4.47. An affine net is ***collected*** if its unreachable nodes are the weakening nodes.

Hence, in collected nets all axiom links are reachable from the root and each weakening node is

- either a conclusion,
- or a premise of a $⅋$-link or a contraction link, which is reachable from the root.

The last possibility is rather unwelcome, as we argue later at more length. Each correct affine net can be transformed to a collected form by contraction of its unreachable nodes to weakening nodes (we describe a subtler reduction in the next section). For such an eager procedure to generate another correct net the following lemma should be true.

Lemma 4.48. If the conclusion of a link in a correct affine net is not reachable from the root, neither are its premises.

Proof. The only unobvious case is that of a $⅋^+$-node and its sink. We need to show that if the sink of a $⅋^+$-node is not reachable, nor is its sink. Fortunately, this follows directly from Proposition 4.44. □

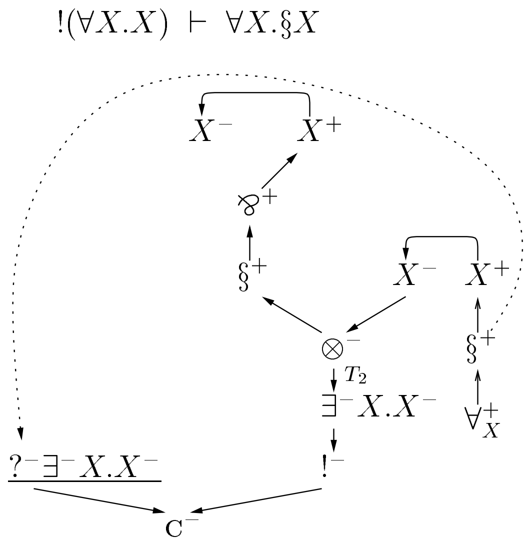


Figure 4.22: A correct collected affine net.

The collected counterpart of the net from Figure 4.20 is presented in Figure 4.22. For collected nets the correctness criterion collapses to the standard conditions for IMLLL2 without the need for a correct protonet as a witness. But, of course, to prove the result we need to show that an extension always exists in such cases.

Theorem 4.49. Collected affine essential nets satisfying (i) - (vi) are correct and, hence, sequentializable.

Proof. Given a collected affine net satisfying (i)-(vi), we show how to find the strengthening edges for each weakening node making sure that the conditions are preserved.

- We strengthen each weakening node which is a conclusion of the net using the root of the net. It is easy to see all the required conditions still hold.
- Weakening nodes that are sinks of $⅋^+$-nodes are strengthened from the positive premises of the corresponding $⅋^+$-nodes. Again, it is easy to verify that the correctness criteria are preserved.
- If a weakening node is a premise of a $⅋^-$-link, the conclusion of the link (call it c) must be reachable from the root. Consider an arbitrary path from the root to c and especially its final segment c' starting at the last negative atomic node that c visits:

  $$c' = a^- v_1 \cdots v_k c\,.$$

  Nodes v_i $(1 \leq i \leq k)$ can be of the following kind: a $\otimes^-$-,$⅋^-$-,$\S^-$-, $?^-$-node or a $\exists^- Z.A^-$-node.

– If no v_i is a bracketing node, we strengthen the weakening node using a^+. The reachability relation on nodes stays the same and the strengthening edge does not circumvent any bracketing nodes, any $\forall^+$-nodes or $\wp^+$-nodes (indeed, any positive nodes). Therefore, all correctness criteria will be preserved.

– If there exists i for which v_i is a (negative) bracketing node, we pick the largest such i and use $\delta(v_i)$ as the source of the strengthening edge.

  Because there exists at least one path from $\delta(v_i)$ to v_i (remember that the net is collected!), the graph remains acyclic as it was. As each such path is well-bracketed (by (iv)), all conditions referring to bracketing ((iii),(iv),(v) and the second half of (vi)) are preserved.

  Suppose (ii) does not hold and there exists a $\wp^+$-node p and a path s from the root to s_p that does not visit p. It will contain the strengthening edge then. Consider another path s', which is otherwise the same as s, but in which the strengthened link is replaced with a path from $\delta(v_i)$ to v_i. That path comes from a correct collected net, so p occurs in between $\delta(v_i)$ and v_i. But the path between p and s_p cannot be well-bracketed (v_i has no matching bracket) contradicting (iii) for the original net.

  Finally, suppose the strengthening edge bypasses some $\forall^+_X$-node, yet leads to a $\exists^-$-link whose eigentype contains X. Again, let us consider the path that results when the strengthening edge is replaced with a path from $\delta(v_i)$ to v_i, which must contain the $\forall^+_X$-node. But then the second half of (vi) would be violated in the original collected net, as we would have a malformed path from $\forall^+_X$ to some $\exists^-$-node with eigentype containing X.

- The last case is when a weakening node is a premise of a contraction link; then we proceed as for a $\wp^-$-node.

$\square$

We have already remarked that the weakening rule commutes with any IMLL2 rule. One source of unreachable nodes is the lack of such commutation between weakening and rules concerning ! and §. This would involve pushing a formula through the interface of a box, which can be done only in a few cases. For **(§)** and **(§$_0$)** the weakening formula must have the shape $?^-A^-$ or $\S^-A^-$. In the case of **(!$_0$)**, only one application of weakening (for $?^-A^-$) can be permuted upwards and for **(!)** commutation is out of question. However, by delegating weakening to earlier stages of a proof we would create unreachable nodes, contrary to our intentions.

$$!a \vdash (!a \otimes !a) \otimes !a$$

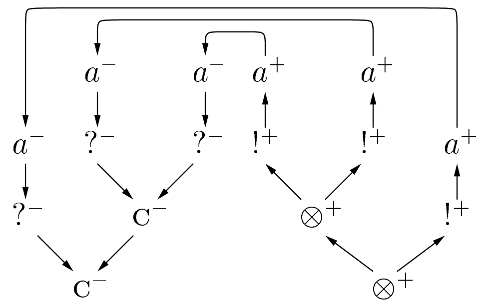


Figure 4.23: Conservation of the order of contractions.

The normalization should proceed in the opposite direction:

$$\begin{tikzcd} \underline{W^-} \arrow[d] \\ \Box^- \end{tikzcd} \quad \rightsquigarrow \quad \underline{\Box^- W^-} \qquad\qquad \Box \in \{\,\S, !\,\}$$

Note that now there is no limit on how many formulas can be pushed out of a box. An analogous problem arises in the term calculus, where it is handled by removing dummy variables from interfaces of boxes (Section 2.5.1). Yet another imperfection in the affine net setting is the interpretation of weakening followed by contraction (for the first time the term language has the edge!). The following reduction then applies:

$$\begin{tikzcd} \underline{W^-} \arrow[dr] & & n^- \arrow[dl] \\ & \mathrm{C}^- & \end{tikzcd} \quad \rightsquigarrow \quad n^-$$

A related deficiency is the conservation of the order in which contraction is performed on identical formulas (see Figure 4.23)—the term calculus gets around these difficulties and so will our game models. To fix the representation of normal forms we allow contraction links with a variable (greater than two) number of *unordered* premises. Additionally, in order to avoid the problems with weakening, we can stipulate that their premises be $?^-$-nodes. The derivation represented by the net in Figure 4.23 would generate the net in Figure 4.24.
It is possible to relate such nets with sequent-calculus in a precise way. A solution using discharged premises has been presented in [41], where weakening is a treated as a special case of contraction. This is no longer possible for affine logics, nevertheless the basic ideas are still applicable.

Finally, we examine the remaining case of weakening nodes being used as principal formulas in other (negative) rules. If two formulas are introduced by weakening

$!a \vdash (!a \otimes !a) \otimes !a$

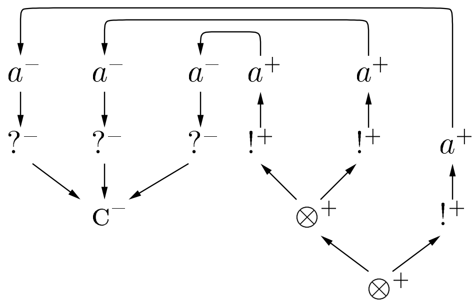
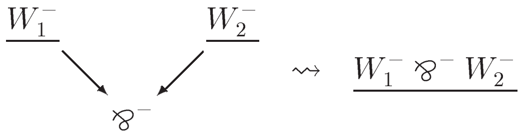
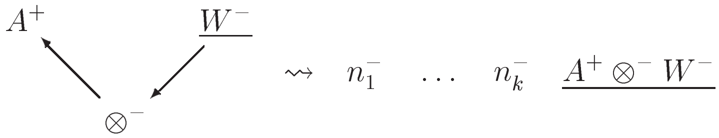


Figure 4.24: A net with a contraction link of arity 3.

and subsequently combined using the $(\wp^-)$ rule, we can replace them with one application of weakening:



The $(\otimes^-)$ rule is the most complicated one, as we will contract the whole empire of A^+ in one step (cf. the corresponding garbage collection rule for terms, where a whole term needs to be erased) in addition to the $\otimes^-$-node:



Definition 4.50. Collected affine nets with contraction links of changing arity are canonical if no premise of a contraction link is a weakening or a contraction node.

It is with respect to canonical nets that our full and faithful completeness results will hold.

4.5 Extensions

We briefly mention some features that were not included in our presentation, which was aimed at the introduction of normal forms of proofs.

First of all, our nets represent η-expanded proofs, but the sequentialization results easily apply to nets in which there exist axiom links between complex formulas. It is not so obvious though how the cut rule could be interpreted. This would raise two problems: the need for a new sequentialization result and a cut-elimination algorithm. The solution has already been proposed in [80]: a cut-link should resemble a $\otimes^-$-link in terms of orientation and polarity, but, of course, its conclusion cannot be regarded as a premise of other links. Therefore, once introduced it remains a 'conclusion', although on the other hand it cannot be treated as such in strict terms: paths from the root to conclusions are required to be well-bracketed, but this well-bracketing will often be broken for a cut-link, because it will not participate in subsequent (§) and (!) rules ('boxed' cuts). Nevertheless, due to (iv), for each cut-link there exists a sequence of negative bracketing nodes such that in any path from the root to the cut-node the unclosed positive brackets correspond to the negative ones. Hence, a cut-link could in principle be treated as a $\otimes^-$-link with several negative $!^-$-links placed below it. Besides, we should require that $\phi(n_1^+) = \phi(n_2^-)^\perp$ hold for its premises n_1^+, n_2^-.

A cut-elimination algorithm for the nets with cut-links can be implemented along the lines of the reduction rules for the sequent calculus as given in Chapter 2 (which in turn follow the original cut-elimination procedure for LLL proof nets with boxes [42]). When one is interested in η-expanded and canonical forms (as we are), extra normalization and η-expansion should follow the usual cut-elimination. Our motivation is compatibility with the categorical framework (to be presented in the next chapter) and representability in the lambda calculus (for example, the binary 'Church numerals' should be the only canonical proofs of BIN).

4.6 Terms and nets

Both proof-terms and essential nets are representations of sequent-calculus proofs. Essential nets have the advantage of implementing various commuting conversions for free: (after we add contraction links with variable arity) $\pi\sigma$-equivalent terms induce identical nets. The converse statement would amount to a *coherence theorem* for the class of *light affine categories* of the next chapter and we leave it for future research conjecturing that it is also true.

Chapter 5

Some category theory

In this chapter we outline the categorical structure that is needed to model various fragments of Light Affine Logic and derive categorical semantics of IMLAL and IMLAL2, which we call light affine categories and hyperdoctrines respectively. The presentation is aligned to the game models of the next chapter, which will be instances of the framework presented here. The results build upon research into categorical semantics of Intuitionistic Linear Logic [19, 86, 20, 76] and second-order polymorphism [112, 106, 65, 68, 31].

5.1 Autonomous categories

We begin with IMLL whose proofs can be interpreted as morphisms in symmetric monoidal closed categories (also called autonomous or closed) [87]. In fact, IMLL cut-free proofs (up to π-congruence) represent morphisms in free such categories [86, 76].

Definition 5.1. A ***symmetric monoidal category*** $\mathbb{C}$ is a category equipped with a bifunctor $\otimes : \mathbb{C} \times \mathbb{C} \longrightarrow \mathbb{C}$, an object I called the ***tensor unit*** and four isomorphisms

$$
\begin{array}{lcrcll}
\mathsf{a}_{A,B,C} & : & A \otimes (B \otimes C) & \longrightarrow & (A \otimes B) \otimes C & \text{associativity} \\
\mathsf{c}_{A,B} & : & A \otimes B & \longrightarrow & B \otimes A & \text{symmetry} \\
\mathsf{l}_A & : & I \otimes A & \longrightarrow & A & \text{left unit} \\
\mathsf{r}_A & : & A \otimes I & \longrightarrow & A & \text{right unit}
\end{array}
$$

defined for and natural in all objects A, B and C such that the diagrams in Figure 5.1 commute. Moreover, l_I and $\mathsf{r}_I : I \otimes I \longrightarrow I$ are required to coincide. We say that $\mathbb{C}$ is ***symmetric monoidal closed*** (SMCC) or ***autonomous*** just in case for each object $A \in \mathbb{C}$, the functor $(-) \otimes A$ has a specific right adjoint $A \multimap (-)$.

$$\begin{tikzcd}
& (A \otimes B) \otimes (C \otimes D) \arrow[dr, "\mathsf{a}_{A \otimes B, C, D}"] & \\
A \otimes (B \otimes (C \otimes D)) \arrow[ur, "\mathsf{a}_{A,B,C \otimes D}"] \arrow[d, "\mathsf{id}_A \otimes \mathsf{a}_{B,C,D}"'] & & ((A \otimes B) \otimes C) \otimes D \\
A \otimes ((B \otimes C) \otimes D) \arrow[rr, "\mathsf{a}_{A,B \otimes C,D}"'] & & (A \otimes (B \otimes C)) \otimes D \arrow[u, "\mathsf{a}_{A,B,C} \otimes \mathsf{id}_D"']
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes (B \otimes C) \arrow[r, "\mathsf{a}_{A,B,C}"] \arrow[d, "\mathsf{id}_A \otimes \mathsf{c}_{B,C}"'] & (A \otimes B) \otimes C \arrow[r, "\mathsf{c}_{A \otimes B, C}"] & C \otimes (A \otimes B) \arrow[d, "\mathsf{a}_{C,A,B}"] \\
A \otimes (C \otimes B) \arrow[r, "\mathsf{a}_{A,C,B}"'] & (A \otimes C) \otimes B \arrow[r, "\mathsf{c}_{A,C} \otimes \mathsf{id}_B"'] & (C \otimes A) \otimes B
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes (I \otimes B) \arrow[d, "\mathsf{a}_{A,I,B}"'] \arrow[dr, "\mathsf{id}_A \otimes \mathsf{l}_B"] & \\
(A \otimes I) \otimes B \arrow[r, "\mathsf{r}_A \otimes \mathsf{id}_B"'] & A \otimes B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \otimes B \arrow[rr, "\mathsf{id}_{A \otimes B}"] \arrow[dr, "\mathsf{c}_{A,B}"'] & & A \otimes B \\
& B \otimes A \arrow[ur, "\mathsf{c}_{B,A}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
& A \otimes I \arrow[dl, "\mathsf{r}_A"'] \arrow[d, "\mathsf{c}_{A,I}"] \\
A & I \otimes A \arrow[l, "\mathsf{l}_A"]
\end{tikzcd}$$

Figure 5.1: Commuting diagrams of autonomous categories.

Throughout the chapter the composition of two morphisms $f : A \longrightarrow B$ and $g : B \longrightarrow C$ will be denoted by $f; g : A \longrightarrow C$. $\mathsf{id}_A : A \longrightarrow A$ will be the identity morphism on A. Bifunctoriality of $\otimes$ means that $\mathsf{id}_A \otimes \mathsf{id}_B = \mathsf{id}_{A \otimes B}$ and $(f_1 \otimes f_2); (g_1 \otimes g_2) = (f_1; g_1) \otimes (f_2; g_2)$, where the morphisms involved have suitable types (such that the left-hand side makes sense). The existence of an adjunction between $(-) \otimes A$ and $A \multimap (-)$ is equivalent to the specification of a natural transformation

$$\mathsf{ev}_A : (A \multimap (-)) \otimes A \longrightarrow Id_{\mathbb{C}} \qquad \text{(where } Id_{\mathbb{C}} : \mathbb{C} \longrightarrow \mathbb{C} \text{ is the identity functor)}$$

for each $A \in \mathbb{C}$ such that given $f : C \otimes A \longrightarrow B$ there exists a unique morphism $\widehat{f} : C \longrightarrow A \multimap B$ satisfying $(\widehat{f} \otimes \mathsf{id}_A); \mathsf{ev}_{A,B} = f$.

Before we show how to interpret IMLL proofs in autonomous categories let us recall the term calculus in Figure 5.2. It includes new rules for the tensor unit, which

(id) $x : A \vdash x^A : A$

(exch) $$\frac{\Gamma, x : A, y : B, \Delta \vdash s : C}{\Gamma, y : B, x : A, \Delta \vdash s : C}$$

(cut) $$\frac{\Gamma, x : A \vdash s : B \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash s\{t\,/x^A\} : B}$$

(I-l) $$\frac{\Gamma \vdash s : A}{\Gamma, x : I \vdash \langle x\,/\ast\,\rangle s : A}$$

(I-r) $\vdash \ast : I$

($\otimes$-l) $$\frac{x : A, y : B, \Gamma \vdash s : C}{z : A \otimes B, \Gamma \vdash \langle z^{A\otimes B}\,/x^A \otimes y^B\,\rangle s : C}$$

($\otimes$-r) $$\frac{\Gamma \vdash s : A \qquad \Delta \vdash t : B}{\Gamma, \Delta \vdash s \otimes t : A \otimes B}$$

($\multimap$-l) $$\frac{\Gamma \vdash s : A \qquad y : B, \Delta \vdash t : C}{z : A \multimap B, \Gamma, \Delta \vdash \langle z^{A\multimap B}\,,\, s/y^B\,\rangle t : C}$$

($\multimap$-r) $$\frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \boldsymbol{\lambda} x^A.s : A \multimap B}$$

Figure 5.2: Rules defining the valid typing judgements of IMLL.

were irrelevant to the complexity considerations of Chapter 2. Yet they become important in categorical type correspondences: the tensor unit is used to represent empty variable contexts so that closed terms induce elements of categorical objects.

Remark 5.2. The IMLL syntax is the same as in [76] but we mention in passing that the explicit let-construct $\langle x\,/\ast\,\rangle-$ interpreting the left introduction rule of the unit could be omitted (at least for IMLL). A change like this would simplify the syntax without modifying the equational theory of proofs, because $\langle x\,/\ast\,\rangle-$ commutes with any IMLL rule anyway. Therefore, by leaving it out we could accommodate more IMLL commuting conversions in an implicit manner. However, the presence of $\langle x\,/\ast\,\rangle-$ allows one to make subtle distinctions in the theory of proofs after weakening is added, namely between **(I-l)** and weakening for I.

The commuting conversions in IMLL are either factored out by the syntax (all commutations with **(exch)**) or accounted for by notions of π- and σ-congruence [76]. The congruence is defined to be the contextual closure (in addition to reflexivity, symmetry and transitivity) of the axioms:

$$\begin{array}{ll} (\boldsymbol{\pi}\textbf{-cong}) & \pi C[t] \sim C[\pi t] \\ (\boldsymbol{\sigma}\textbf{-cong}) & C[t]\sigma \sim C[t\sigma] \end{array}$$

where π ranges over the let-constructs

$$\langle z^{A\otimes B} / x \otimes y \rangle - , \quad \langle z^{A\multimap B} , s/y \rangle - , \quad \langle x / * \rangle -$$

and σ over the explicit substitution constructs $-\{t / x^A\}$. The congruence axioms can be invoked on the condition that both sides of the respective judgments are typable and have the same type. Contexts are defined by:

$$\begin{array}{rl} C \;::=\; & [-] \mid s\{C / x^A\} \mid C\{s / x^A\} \mid s \otimes C \mid C \otimes s \mid \boldsymbol{\lambda} x^A.C \mid \\ & \langle z^{A\otimes B} / x \otimes y \rangle C \mid \langle z^{A\multimap B} , C/y^B \rangle s \mid \langle z^{A\multimap B} , s/y^B \rangle C \mid \langle x / * \rangle C \end{array}$$

Reduction in the calculus (Figure 5.3) implements a cut-elimination algorithm and concerns equivalence classes of terms (modulo $\pi\sigma$-congruence) rather than sole terms. This is slightly different from the reduction given in Chapter 2 as we explain in a moment (for example, η-expansion is allowed). Rules for the unit are also new. The associated equational theory of proofs is defined by $\pi\sigma$-congruence and (the contextual closure of) the rewrite system, which is strongly normalizing and confluent [76]. Observe that there are no critical pairs in the system thanks to the side-condition for (atom-β).

The (atom-β) rule is a restriction of (var-β) (Figure 2.4) to atomic types different from I—and so a different cut-elimination algorithm (from the locally poly-time procedure of Chapter 2) is in action. In proof-net terminology, only cuts involving atomic axiom links are now eliminated (in other cases they are only propagated); non-atomic axiom links will be expanded as long as atomic ones emerge. Before a cut is processed the substituted term must reach a canonical form dependent on its type. For linear arrow types this is a lambda abstraction, for tensor types $s \otimes t$, for I the term $*$. We can delay the elimination of the cuts as in (var-β), because η-expansion rules ensure that the cut can be eliminated using other β-rules. Analogously, the (rename-β) rule is no longer needed, because η-expansion and the other β rules subsume it. Consequently, cut-elimination is no longer lazy and rewriting (even in untyped form) need not be a polynomial-time operation. Nevertheless, only FP functions will be computed: after all, the only essential difference between this system and that from Figure 2.4 is the presence of η-expansion. The formulation considered in this chapter has a clear categorical description which can be modelled faithfully with games.

$$\begin{array}{rl}
(\text{atom-}\beta) & x^a\{s\,/x^a\} \to s : a \quad (a \text{ is a non-}I \text{ atomic type}) \\[6pt]
(I\text{-}\beta) & (\langle x\,/{*}\,\rangle s)\{\,{*}\,/x^I\} \to s \\[6pt]
(I\text{-}\eta) & x^I \to \langle x\,/{*}\,\rangle {*} \\[6pt]
(\otimes\text{-}\beta) & (\langle z^{A\otimes B}\,/x^A \otimes y^B\,\rangle s)\{u \otimes v\,/z^{A\otimes B}\} \to (s\{u\,/x^A\})\{v\,/y^B\} \\[6pt]
(\otimes\text{-}\eta) & z^{A\otimes B} \to \langle z^{A\otimes B}\,/x^A \otimes y^B\,\rangle (x \otimes y) \\[6pt]
(\multimap\text{-}\beta) & (\langle z^{A\multimap B}\,,\ s/y^B\,\rangle t)\{\boldsymbol{\lambda} x^A.u\,/z^{A\otimes B}\} \to t\{u\{s\,/x^A\}\,/y^B\} \\[6pt]
(\multimap\text{-}\eta) & z^{A\multimap B} \to \boldsymbol{\lambda} x^A.\langle z^{A\multimap B}\,,\ x/y^B\,\rangle y
\end{array}$$

Figure 5.3: Rules defining reduction of IMLL proofs.

Remark 5.3. If the $\langle x\,/{*}\,\rangle-$ construct was to be left out, the reduction rules for I would have to be replaced with

$$\begin{array}{rl}
(I\text{-}\beta)' & s\{\,{*}\,/x^I\} \to s \\
(I\text{-}\eta)' & x^I \to {*}
\end{array}$$

Next we show how IMLL typing judgments are interpreted in an arbitrary autonomous category $\mathbb{C}$. Suppose an object $[\![A]\!]$ of $\mathbb{C}$ is assigned to each base type A. The assignment is then extended to all types generated from base types with $\otimes$ and $\multimap$ by:

$$[\![I]\!] = I \qquad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!] \qquad [\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$$

Derivations of $x_1 : A_1, \cdots, x_k : A_k \vdash t : B$ will define morphisms in $\mathbb{C}$, denoted by

$$[\![x_1 : A_1, x_2 : A_2, \cdots, x_{k-1} : A_{k-1}, x_k : A_k \vdash t : B]\!]\,,$$

between

$$[\![x_1 : A_1, x_2 : A_2, \cdots, x_{k-1} : A_{k-1}, x_k : A_k]\!] = ((([\![A_1]\!] \otimes [\![A_2]\!]) \cdots) \otimes [\![A_{k-1}]\!]) \otimes [\![A_k]\!]$$

and $[\![B]\!]$. If the context is empty, the domain is simply $[\![\varnothing]\!] = I$. Observe that by composing $\mathsf{a}, \mathsf{a}^{-1}, \mathsf{c}, \mathsf{id}$ with the use of the functorial operation $\otimes$ on morphisms one can permute the objects in the representation of contexts as well as changing the bracketing. It turns out that all possible permuting isomorphisms with a given domain and codomain are the same thanks to the defining diagrams of autonomous categories. This property is an instance of the Coherence Theorem for symmetric monoidal (not necessarily closed) categories [72]. It also holds, if we add the families $\mathsf{l}, \mathsf{l}^{-1}, \mathsf{r}, \mathsf{r}^{-1}$ to the generating set (the inverses of l, r are especially useful when two contexts are merged and one of them is empty). The unique isomorphisms are called ***bookkeeping maps***.

The denotations of terms (in context) are defined by induction on the structure of derivations [86, 76]:

- $x : A \vdash x^A : A = \mathsf{id}_{\llbracket A \rrbracket} : \llbracket A \rrbracket \longrightarrow \llbracket A \rrbracket$,
- $\llbracket \Gamma, y : B, x : A, \Delta \vdash s : C \rrbracket = \kappa; \llbracket \Gamma, x : A, y : B, \Delta \vdash s : C \rrbracket$, where κ is the bookkeeping map that swaps the places of $\llbracket A \rrbracket$ and $\llbracket B \rrbracket$,
- $\llbracket \Gamma, \Delta \vdash s\{t / x^A\} : B \rrbracket = \kappa; (\mathsf{id}_{\llbracket \Gamma \rrbracket} \otimes \llbracket \Delta \vdash t : A \rrbracket); \llbracket \Gamma, x : A \vdash s : B \rrbracket$ where $\kappa : \llbracket \Gamma, \Delta \rrbracket \longrightarrow \llbracket \Gamma \rrbracket \otimes \llbracket \Delta \rrbracket$ is the appropriate bookkeeping map,
- $\llbracket \Gamma, x : I \vdash \langle x / * \rangle s : A \rrbracket = \mathsf{r}_{\llbracket \Gamma \rrbracket}; \llbracket \Gamma \vdash s : A \rrbracket$ ($\mathsf{r}_{\llbracket \Gamma \rrbracket}$ is redundant for $\Gamma = \varnothing$),
- $\llbracket \ \vdash * : I \rrbracket = \mathsf{id}_I : I \longrightarrow I$,
- $\llbracket z : A \otimes B, \Gamma \vdash \langle z^{A \otimes B} / x^A \otimes y^B \rangle s \rrbracket = \llbracket x : A, y : B, \Gamma \vdash s \rrbracket$,
- $\llbracket \Gamma, \Delta \vdash s \otimes t : A \otimes B \rrbracket = \kappa; (\llbracket \Gamma \vdash s : A \rrbracket \otimes \llbracket \Delta \vdash t : B \rrbracket)$, where $\kappa : \llbracket \Gamma, \Delta \rrbracket \longrightarrow \llbracket \Gamma \rrbracket \otimes \llbracket \Delta \rrbracket$ is the bookkeeping map,
- $\llbracket A \multimap B, \Gamma, \Delta \vdash \langle z^{A \multimap B} \, , \, s/y^B \rangle t : C \rrbracket$ closes the diagram below (κ stands for suitable bookkeeping maps)

$$\begin{tikzcd}
\llbracket A \multimap B, \Gamma, \Delta \rrbracket \arrow[rr, dashed, "\llbracket A \multimap B, \Gamma, \Delta \vdash \langle z^{A \multimap B} \, , \, s/y^B \rangle t : C \rrbracket"] \arrow[d, "\kappa"'] & & \llbracket C \rrbracket \\
(\llbracket A \multimap B \rrbracket \otimes \llbracket \Gamma \rrbracket) \otimes \llbracket \Delta \rrbracket \arrow[d, "(\mathsf{id}_{\llbracket A \multimap B \rrbracket} \otimes \llbracket \Gamma \vdash s : A \rrbracket) \otimes \mathsf{id}_{\llbracket \Delta \rrbracket}"] & & \llbracket B, \Delta \rrbracket \arrow[u, "\llbracket y : B, \Delta \vdash t : C \rrbracket"'] \\
(\llbracket A \multimap B \rrbracket \otimes \llbracket A \rrbracket) \otimes \llbracket \Delta \rrbracket \arrow[rr, "\mathsf{ev}_{\llbracket A \rrbracket, \llbracket B \rrbracket} \otimes \mathsf{id}_{\llbracket \Delta \rrbracket}"'] & & \llbracket B \rrbracket \otimes \llbracket \Delta \rrbracket \arrow[u, "\kappa"']
\end{tikzcd}$$

- $\llbracket \Gamma \vdash \boldsymbol{\lambda} x^A . s : A \multimap B \rrbracket$ is the mate of $\llbracket \Gamma, x : A \vdash s : B \rrbracket$ across the adjunction $(-) \otimes A \dashv A \multimap (-)$.

Because the interpreted objects are terms, one needs to show that the above definition does not depend on the choice of a derivation. Fortunately, two derivations of the same term can only differ by the order of applications of **(exch)** and **(I-l)**. Thus the problem reduces to the already mentioned Coherence Theorem.

The semantics is sound with respect to the congruence and reduction rules. For detailed proofs we refer the reader to [76]. Conversely, the classifying syntactic category build around the syntax is autonomous, i.e. autonomous categories are the categorical counterpart of IMLL. In the sections to follow we derive a categorical semantics for the rest of IMLAL2 connectives starting from §.

5.2 Symmetric monoidal functors

By the rule

$$(\S) \qquad \frac{x_1 : A_1, \ldots, x_k : A_k \;\vdash\; s : B \qquad \Box_i = \S, ! \quad 1 \leqslant i \leqslant k}{x_1' : \Box_1 A_1, \ldots, x_k' : \Box_k A_k \;\vdash\; \S\langle\, x_1'/x_1, \cdots, x_k'/x_k \,\rangle(s) : \S B}$$

we have $\S A \vdash \S B$ whenever $A \vdash B$. Besides, ($\S$-η) implies $\S(\mathsf{id}_A) = \mathsf{id}_{\S A}$ and the reduction rule ($\S$-$\S$) specializes to

$$\S\langle\, x/x' \,\rangle(s)\{\S\langle\, y/y' \,\rangle(t)\,/x\} \rightarrow \S\langle\, y/y' \,\rangle(s\{t\,/x'\})\,,$$

which means that § is a functor. In addition, $\S A \otimes \S B \vdash \S(A \otimes B)$ is provable, i.e. we will need a family of morphisms $\mathsf{m}_{A,B} : \S A \otimes \S B \longrightarrow \S(A \otimes B)$ that is natural in A and B (as expressed in the ($\S$-$\S$) rule). Also, $\vdash \S I$ is provable, which calls for a special morphism $\mathsf{m}_I : I \longrightarrow \S I$. Furthermore, because the order of variables in the interface of a §-box is irrelevant (i.e. **(§)** commutes with **(exch)**), § is symmetric monoidal.

Definition 5.4. A functor $\S : \mathbb{C} \longrightarrow \mathbb{C}$ is ***symmetric monoidal*** if there exists a natural transformation

$$\mathsf{m}_{A,B} : \S A \otimes \S B \longrightarrow \S(A \otimes B)$$

and a map $\mathsf{m}_I : I \longrightarrow \S I$ making the diagrams in Figure 5.4 commute.

For a start, this additional structure enables us to interpret the rule

$$(\S_0) \qquad \frac{\vdash\; s : B}{\vdash\; \S(s) : \S B}$$

$$
\begin{tikzcd}
\S I \otimes \S A \arrow[r, "\mathsf{m}_{I,A}"] & \S(I \otimes A) \arrow[d, "\S(\mathsf{l}_A)"] \\
I \otimes \S A \arrow[u, "\mathsf{m}_I \otimes \mathsf{id}_{\S A}"] \arrow[r, "\mathsf{l}_{\S A}"'] & \S A
\end{tikzcd}
\qquad
\begin{tikzcd}
\S A \otimes \S I \arrow[r, "\mathsf{m}_{A,I}"] & \S(A \otimes I) \arrow[d, "\S(\mathsf{r}_A)"] \\
\S A \otimes I \arrow[u, "\mathsf{id}_{\S A} \otimes \mathsf{m}_I"] \arrow[r, "\mathsf{r}_{\S A}"'] & \S A
\end{tikzcd}
$$

$$
\begin{tikzcd}
(\S A \otimes \S B) \otimes \S C \arrow[r, "\mathsf{m}_{A,B} \otimes \mathsf{id}_C"] & \S(A \otimes B) \otimes \S C \arrow[r, "\mathsf{m}_{A \otimes B, C}"] & \S((A \otimes B) \otimes C) \\
\S A \otimes (\S B \otimes \S C) \arrow[u, "\mathsf{a}_{\S A, \S B, \S C}"] \arrow[r, "\mathsf{id}_{\S A} \otimes \mathsf{m}_{B,C}"'] & \S A \otimes \S(B \otimes C) \arrow[r, "\mathsf{m}_{A, B \otimes C}"'] & \S(A \otimes (B \otimes C)) \arrow[u, "\S(\mathsf{a}_{A,B,C})"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
\S A \otimes \S B \arrow[r, "\mathsf{m}_{A,B}"] \arrow[d, "\mathsf{c}_{\S A, \S B}"'] & \S(A \otimes B) \arrow[d, "\S(\mathsf{c}_{A,B})"] \\
\S B \otimes \S A \arrow[r, "\mathsf{m}_{B,A}"'] & \S(B \otimes A)
\end{tikzcd}
$$

Figure 5.4: Commuting diagrams of a symmetric monoidal functor §.

by

$$[\![\ \vdash\ \S(s) : \S B]\!] = \mathsf{m}_I ; \S([\![\ \vdash\ s : B]\!])\,.$$

The diagrams associated with a symmetric monoidal functor allow for an extension of the Coherence Theorem: there is a unique natural transformation

$$\mathsf{m}_{A_1,\cdots,A_k} : ((\S A_1 \otimes \S A_2)\cdots) \otimes \S A_k \longrightarrow \S(((A_1 \otimes A_2)\cdots) \otimes A_k)$$

constructed from $\mathsf{id}, \mathsf{m}, \mathsf{a}, \mathsf{a}^-, \mathsf{c}, \mathsf{l}, \mathsf{l}^{-1}, \mathsf{r}, \mathsf{r}^{-1}$ using composition and the functorial operations $\otimes$ and §. The following restriction of **(§)**

$$(\S^-) \qquad \frac{x_1 : A_1, \ldots, x_k : A_k \ \vdash\ s : B \qquad 1 \leqslant i \leqslant k}{x_1' : \S A_1, \ldots, x_k' : \S A_k \ \vdash\ \S\langle\, x_1'/x_1, \cdots, x_k'/x_k \,\rangle(s) : \S B}$$

now admits an interpretation. Given a context $\Gamma = x_1 : A_1, \cdots, x_k : A_k$ let $\S\Gamma$ be the context $x_1' : \S A_1, \cdots, x_k' : \S A_k$. Then we can define

$$[\![\S\Gamma\ \vdash\ \S\langle\, x_1'/x_1, \cdots, x_k'/x_k \,\rangle(s) : \S B]\!] = \mathsf{m}_{A_1,\cdots,A_k} ; \S([\![\Gamma\ \vdash\ s : B]\!]).$$

Exponential redexes

$$
\begin{array}{ll}
(!\text{-}!) & !\langle\, x/x_0 \,\rangle(s)\{!\langle\, y/y_0 \,\rangle(t)\,/x\} \to\, !\langle\, y/y_0 \,\rangle(s\{t\,/x_0\}) \\
(\S\text{-}!) & \S\langle \cdots, x/x_0, \cdots\rangle(s)\{!\langle\, y/y_0 \,\rangle(t)\,/x\} \to \S\langle \cdots, y/y_0, \cdots\rangle(s\{t\,/x_0\}) \\
(\S\text{-}\S) & \S\langle \cdots, x/x_0, \cdots\rangle(s)\{\S\langle\, \overline{y/y_0} \,\rangle(t)\,/x\} \to \S\langle \cdots, \overline{y/y_0}, \cdots\rangle(s\{t\,/x_0\})
\end{array}
$$

η-redexes

$$
\begin{array}{ll}
(!\text{-}\eta) & z^{!A} \to\, !\langle\, z/z_0 \,\rangle(z_0) \\
(\S\text{-}\eta) & z^{\S A} \to \S\langle\, z/z_0 \,\rangle(z_0)
\end{array}
$$

Figure 5.5: Reduction rules for ! and §

The (§-§) rule (for a !-free syntax) is modelled soundly, because of functoriality of § and naturality of m:

$$
\begin{tikzcd}
\S A \otimes \S C \arrow[r, "\mathsf{id}_{\S A} \otimes \S(f)"] \arrow[d, "\mathsf{m}_{A,C}"'] & \S A \otimes \S B \arrow[d, "\mathsf{m}_{A,B}"] \\
\S(A \otimes C) \arrow[r, "\S(\mathsf{id}_A \otimes f)"'] & \S(A \otimes B)
\end{tikzcd}
$$

To cover its full version we have to able to interpret ! first.

Analogously to the case of §, it is easy to see that ! should be a functor and that there should exist a map $\mathsf{s}_I : I \longrightarrow !I$.

$$
(!_0) \qquad \frac{\vdash\ s : B}{\vdash\ !(s) : !B}
$$

is then denoted by

$$
[\![\ \vdash\ !(s) : !B]\!] = \mathsf{s}_I ; !([\![\ \vdash\ s : B]\!])\,,
$$

while the rule

$$
(!) \qquad \frac{x' : A\ \vdash\ s : B}{x : !A\ \vdash\ !\langle\, x/x' \,\rangle(s) : !B}
$$

gives rise to

$$[\![x : !A \vdash !\langle x/x' \rangle(s)]\!] = !([\![x : A \vdash s : B]\!])\,.$$

In order to assign denotations to the remaining instances of $(\S)$, let us observe that $!A \vdash \S A$ is provable, so we should have a family of maps $\mathsf{z}_A : !A \longrightarrow \S A$ at our disposal. Actually, $\mathsf{z} : ! \longrightarrow \S$ must be a natural transformation such that

$$\begin{tikzcd} & I \arrow[dl, "\mathsf{s}_I"'] \arrow[dr, "\mathsf{m}_I"] & \\ !I \arrow[rr, "\mathsf{z}_I"'] & & \S I \end{tikzcd}$$

commutes (as we shall see, this is due to the (!-§) rule). That is to say, z is as much (or, rather, as little) of a symmetric monoidal transformation as possible given that ! is not symmetric monoidal. The full $(\S)$ rule can now be interpreted through the weaker instance $(\S^-)$ by composition with appropriate maps from z. The semantics validates (!-§) since m and z are natural, and § is functorial:

$$\begin{tikzcd} \S A \otimes !C \arrow[r, "\mathsf{id}_{\S A} \otimes !(f)"] \arrow[d, "\mathsf{id}_{\S A} \otimes \mathsf{z}_C"'] & \S A \otimes !B \arrow[d, "\mathsf{id}_{\S A} \otimes \mathsf{z}_B"] \\ \S A \otimes \S C \arrow[r, "\mathsf{id}_{\S A} \otimes \S(f)"] \arrow[d, "\mathsf{m}_{A,C}"'] & \S A \otimes \S B \arrow[d, "\mathsf{m}_{A,B}"] \\ \S(A \otimes C) \arrow[r, "\S(\mathsf{id}_A \otimes f)"'] & \S(A \otimes B) \end{tikzcd}$$

For the special case of a !-box with an empty interface $\mathsf{s}_I ; \mathsf{z}_I = \mathsf{m}_I$ is a necessary condition:

$$\begin{tikzcd} I \arrow[r, "\mathsf{s}_I"] \arrow[dr, "\mathsf{m}_I"'] & !I \arrow[r, "!(f)"] \arrow[d, "\mathsf{z}_I"'] & !B \arrow[d, "\mathsf{z}_B"] \\ & \S I \arrow[r, "\S(f)"'] & \S B \end{tikzcd}$$

$$
\begin{array}{rl}
(\text{var-}\gamma) & t\{s\,/y^A\} \to t \\
(\otimes\text{-}\gamma) & \langle z^{A\otimes B}\,/x^A \otimes y^B\,\rangle s \to s \\
(\multimap\text{-}\gamma) & \langle z^{A\multimap B}\,,\, t/y^B\,\rangle s \to s \\
(!\text{-}\gamma) & !\langle\, y'/y\,\rangle(s) \to !(s) \\
(\S\text{-}\gamma) & \S\langle\, \cdots, y'/y, \cdots\rangle(s) \to \S\langle\, \cdots, \cdots\rangle(s) \\
(\forall\text{-}\gamma) & \langle z^{\forall\alpha.A}\,,\, B/y^{A[B/\alpha]}\,\rangle s \to s
\end{array}
$$

provided x, y do not occur in s

Figure 5.6: Garbage collection rules

5.3 Weakening and garbage collection

In our game categories the tensor unit is a terminal object, but in general such a requirement would be too strong. Therefore, we first derive a more economical set of properties which turn out to be precisely what one needs to model all reduction rules soundly.

Let $I : \mathbb{C} \longrightarrow \mathbb{C}$ be the constant functor assigning I to any object and id_I to any morphism. A natural transformation $\mathsf{w} : Id_{\mathbb{C}} \longrightarrow I$ is needed to model the logical rule **(weak)** and the reduction rule (var-γ), i.e. for each $f : A \longrightarrow B$ we have $f; \mathsf{w}_B = \mathsf{w}_A$.

$$
\textbf{(weak)} \quad \frac{\Gamma \vdash s : A}{\Gamma, y : B \vdash s : A}
$$

is then interpreted by

$$
[\![\Gamma, x : B \vdash s : A]\!] = (\mathsf{id}_{[\![\Gamma]\!]} \otimes \mathsf{w}_B); \mathsf{r}_{[\![\Gamma]\!]}; [\![\Gamma \vdash s : A]\!]
$$

This time two derivations of the same term may differ by the order of exchanges performed before and after the weakening, but because all bookkeeping maps are natural our definition remains correct.

Let us turn to the garbage collection rules. ($\multimap$-γ) and ($\otimes$-γ) are modelled soundly because of naturality of w. The latter requires the commutativity of

$$\begin{tikzcd}
A \otimes B \arrow[r, "\mathsf{w}_A \otimes \mathsf{w}_B"] \arrow[dr, "\mathsf{w}_{A \oplus B}"'] & I \otimes I \arrow[d, "\mathsf{l}_I = \mathsf{r}_I"] \\
& I
\end{tikzcd}$$

which easily follows from naturality of w and r. If both the reductum and the reduct are to be denoted by the same morphism in (!-γ) then

$$\begin{tikzcd}
!A \arrow[r, "\mathsf{w}_{!A}"] \arrow[dr, "!(\mathsf{w}_A)"'] & I \arrow[d, "\mathsf{s}_I"] \\
& !I
\end{tikzcd}$$

should commute for any A. By naturality, this is equivalent to commutativity for $A = I$ only:

$$\begin{tikzcd}
!A \arrow[d, "!(\mathsf{w}_A)"'] \arrow[dr, "\mathsf{w}_{!A}"] & \\
!I \arrow[r, "\mathsf{w}_{!I}"] \arrow[dr, "!(\mathsf{w}_I)"'] & I \arrow[d, "\mathsf{s}_I"] \\
& !I
\end{tikzcd}$$

Because $\mathsf{w}_A; \mathsf{w}_I = \mathsf{w}_A$, we also have $!(\mathsf{w}_A); !(\mathsf{w}_I) = !(\mathsf{w}_A)$. Therefore, all we need to model (!-γ) is $\mathsf{w}_{!I}; \mathsf{s}_I = !(\mathsf{w}_I)$.

Similarly, for a sound interpretation of the instance of (§-γ) in which variables of type $\S A$ are erased, we only need $\mathsf{w}_{\S I}; \mathsf{m}_I = \S(\mathsf{w}_I)$. The remaining case is that of $!A$-typed variables in interfaces of §-boxes, which requires

$$\begin{tikzcd}
!A \arrow[r, "\mathsf{z}_A"] \arrow[d, "\mathsf{w}_{!A}"'] & \S A \arrow[d, "\S(\mathsf{w}_A)"] \\
I \arrow[r, "\mathsf{m}_I"'] & \S I
\end{tikzcd}$$

$$\begin{tikzcd}
 & !A \arrow[dl, "\mathsf{r}^{-1}_{!A}"'] \arrow[d, "\mathsf{e}_A"] \arrow[dr, "\mathsf{l}^{-1}_{!A}"] & \\
!A\otimes I & !A\otimes !A \arrow[l, "\mathsf{id}_{!A}\otimes \mathsf{w}_{!A}"] \arrow[r, "\mathsf{w}_{!A}\otimes \mathsf{id}_{!A}"'] & I\otimes !A
\end{tikzcd}
\qquad
\begin{tikzcd}
!A \arrow[r, "\mathsf{e}_A"] \arrow[dr, "\mathsf{e}_A"'] & !A\otimes !A \arrow[d, "\mathsf{c}_{!A,!A}"] \\
 & !A\otimes !A
\end{tikzcd}$$

$$\begin{tikzcd}
!A \arrow[rr, "\mathsf{e}_A"] \arrow[d, "\mathsf{e}_A"'] & & !A\otimes !A \arrow[d, "\mathsf{e}_A\otimes \mathsf{id}_{!A}"] \\
!A\otimes !A \arrow[r, "\mathsf{id}_A\otimes \mathsf{e}_{!A}"'] & !A\otimes(!A\otimes !A) \arrow[r, "\mathsf{a}_{!A,!A,!A}"'] & (!A\otimes !A)\otimes !A
\end{tikzcd}$$

Figure 5.7: Diagrams of a commutative comonoid.

to commute. This condition turns out to follow from earlier requirements derived for (!-γ) and (!-§) (namely $\mathsf{s}_I;\mathsf{z}_I = \mathsf{m}_I$) and naturality of z:

$$\begin{tikzcd}
 & !A \arrow[r, "\mathsf{z}_A"] \arrow[dl, "\mathsf{w}_{!A}"'] \arrow[d, "!(\mathsf{w}_A)"] & \S A \arrow[d, "\S(\mathsf{w}_A)"] \\
I \arrow[r, "\mathsf{s}_I"'] & !I \arrow[r, "\mathsf{z}_I"'] & \S I
\end{tikzcd}$$

5.4 Commutative comonoid

To handle contraction we shall require the existence of a natural transformation $\mathsf{e}_A : !A \longrightarrow !A\otimes !A$ such that $(!A, \mathsf{e}_A, \mathsf{w}_A)$ is a ***commutative comonoid***, i.e. the equalities in Figure 5.7 are satisfied. The diagrams simply accommodate the commuting conversions generated by the contraction rule with **(weak)**,**(exch)** and itself. The rule

$$\textbf{(contr)}\quad \frac{x_1 : !A, x_2 : !A, \Gamma \ \vdash\ s : B}{x_1 : !A, \Gamma \ \vdash\ s[x_1/x_2] : B}$$

can then be modelled by

$$[\![x_1 : !A, \Gamma \ \vdash\ s[x_1/x_2] : B]\!] = \kappa_1; (\mathsf{e}_A \otimes \mathsf{id}_{[\![\Gamma]\!]}); \kappa_2; [\![x_1 : !A, x_2 : !A, \Gamma \ \vdash\ s : B]\!]$$

Suppose there are $n \geqslant 2$ free occurrences of z in s.

$$s\{!\langle\, x/x_0\,\rangle(t)\,/z\} \to s'\{!\langle\, x/x_0\,\rangle(t)\,/z_1\}\cdots\{!\langle\, x/x_0\,\rangle(t)\,/z_n\}$$

where s' is obtained from s by distinguishing the n occurrences of z as $z_1, \cdots, z_n$ respectively.

Figure 5.8: The replication rule.

where κ_1, κ_2 are bookkeeping maps. Naturality of e amounts to a sound interpretation of replication (Figure 5.8).

5.5 Terminal object

We have already mentioned that the game models we consider in the next chapter have a terminal object, which is at the same time the tensor unit. The terminality of the unit can be characterized using $\mathsf{w} : Id_{\mathbb{C}} \longrightarrow I$ in a very simple way:

Lemma 5.5. *I is terminal if and only if $\mathsf{w}_I = \mathsf{id}_I$.*

In other words, I becomes terminal if the (I-**l**) rule is regarded as an instance of weakening. This corresponds to replacing the rules (I-β) and (I-η) with (I-β)$'$ and (I-η)$'$ respectively. (I-β)$'$ is in fact an instance of (var-γ). Note that for terminal I there is a unique $\mathsf{w} : Id_{\mathbb{C}} \longrightarrow I$. Besides, it turns out that in a model of IMLAL in which I is terminal, $\S I$ and $!I$ are also terminal, i.e. isomorphic to I.

Proposition 5.6. *If I is terminal in a category satisfying the conditions that make our interpretation of (!-γ) and (§-γ) sound, then $\S I$ and $!I$ are isomorphic to I.*

Proof. Take $\S I$ for example. We have $\mathsf{w}_{\S I}; \mathsf{m}_I = \S(\mathsf{w}_I)$. Because $\mathsf{w}_I = \mathsf{id}_I$, $\mathsf{w}_{\S I}; \mathsf{m}_I = \mathsf{id}_{\S I}$ holds. As I is terminal, $\mathsf{m}_I; \mathsf{w}_{\S I} = \mathsf{id}_I$ holds too, so $\S I$ is isomorphic to I. $\square$

Indeed, in our game models $\S I, !I, I$ are all interpreted by the same object, the empty game, which is terminal. Observe that if $I, \S I, !I$ are terminal, we have:

$$\mathsf{w}_{\S I}; \mathsf{m}_I = \S(\mathsf{w}_I) \quad \mathsf{w}_{!I}; \mathsf{s}_I = !(\mathsf{w}_I) \quad \mathsf{s}_I; \mathsf{z}_I = \mathsf{m}_I$$

for free.

5.6 Light affine categories

We summarize our findings in a single definition for ease of reference.

Definition 5.7. A ***light affine category*** $\mathbb{C}$ is an autonomous category equipped with

1. a symmetric monoidal functor $\S : \mathbb{C} \longrightarrow \mathbb{C}$,
2. a functor $! : \mathbb{C} \longrightarrow \mathbb{C}$ with a map $\mathsf{s}_I : I \longrightarrow !I$,
3. a natural transformation $\mathsf{w} : Id_{\mathbb{C}} \longrightarrow I$ satisfying
$$\mathsf{w}_{!I}; \mathsf{s}_I = !(\mathsf{w}_I) \qquad \mathsf{w}_{\S I}; \mathsf{m}_I = \S(\mathsf{w}_I)$$
where $\mathsf{m}_I : I \longrightarrow \S I$ is the canonical map of $\S$,
4. a natural transformation $\mathsf{z} : ! \longrightarrow \S$ such that
$$\mathsf{s}_I; \mathsf{z}_I = \mathsf{m}_I\,,$$
5. a natural transformation $\mathsf{e} : ! \longrightarrow ! \otimes !$ defining a commutative commonoid $(!A, \mathsf{e}_A, \mathsf{w}_A)$ for any $A \in \mathbb{C}$.

Definition 5.8. Write $LACat$ for the category of all (small) light affine categories and functors preserving (all elements of) their structure in a strict way (not only up to isomorphism).

5.7 Type variables and universal quantification

The interpretation of second-order variables necessitates the availability of still more complicated categorical objects as outlined, e.g. in [31] for System F. We sketch the desired structure adapting it to our case as appropriate.

First we show how to interpret types possibly containing (free) type variables. The basic idea is that each second-order type with n distinct free variables is a description of an n-ary operation on some universe U. Thus each type will induce a morphism
$$\underbrace{U \times \cdots \times U}_{n} \longrightarrow U$$
in a category $\mathbb{C}$ whose objects are $U^0, U^1, U^2, \ldots$, i.e. $\mathbb{C}$ has finite products and is generated from some object U. In particular, it contains a terminal object

to represent the empty (type variable) context. Types with n variables define morphisms in $\mathbb{C}(U^n, U)$ using operations

$$\otimes_n, \multimap_n \colon \mathbb{C}(U^n, U) \times \mathbb{C}(U^n, U) \longrightarrow \mathbb{C}(U^n, U)$$

and

$$\S_n, !_n : \mathbb{C}(U^n, U) \longrightarrow \mathbb{C}(U^n, U).$$

A type variable simply corresponds to a suitable projection. Additionally, for quantification, we will need

$$\forall_n : \mathbb{C}(U^n \times U, U) \longrightarrow \mathbb{C}(U^n, U),$$

which is no longer an endooperation. Before we turn to the conditions on $\forall_n$, let us complete the definition of the structure that will enable us to interpret the non-polymorphic features.

Since types with at most n free variables are interpreted as morphisms in $\mathbb{C}(U^n, U)$, $\mathbb{C}(U^n, U)$ itself should be a light affine category (whose objects are $\mathbb{C}$ morphisms with codomain U). Other $\mathbb{C}$ morphisms, say from $\mathbb{C}(U^m, U^n)$, represent n-tuples of types with at most m-variables and substitution for type variables is modelled by composition in $\mathbb{C}$. Since it is a functorial operation, the hom-functor $\mathbb{C}(\_, U) : \mathbb{C}^{op} \longrightarrow Set$ lifts to a functor $\mathbb{C}^{op} \longrightarrow LACat^{-}$, i.e. an indexed category ($LACat^{-}$ is the category of all small light affine categories and all functors between them). Moreover, as substitution distributes over all syntactic constructs, for any $f \in \mathbb{C}(U^m, U^n)$ the functor $\mathbb{C}(f, U) : \mathbb{C}(U^n, U) \longrightarrow \mathbb{C}(U^m, U)$, called a ***reindexing*** functor, must preserve the structure of a light affine category. Hence, $\mathbb{C}(\_, U) : \mathbb{C}^{op} \longrightarrow LACat$ should be a strict $\mathbb{C}$-indexed light affine category.

We turn to quantification now. As already indicated an operation

$$\forall_n : \mathbb{C}(U^n \times U, U) \longrightarrow \mathbb{C}(U^n, U)$$

is needed for any $n \in \mathbb{N}$. Quantification and (capture-avoiding) substitution commute, so $\forall_n$ ought to be natural in U^n:

$$g; \forall_n(h) = \forall_m((g \times \mathsf{id}_U); h)$$

for $g \in \mathbb{C}(U^m, U), h \in \mathbb{C}(U^n \times U, U)$. Just as for System F [31], the $\forall_n$ operation must lift to a functor which is right adjoint to $\mathbb{C}(\pi_1, U)$, where $\pi_1 : U^n \times U \longrightarrow U^n$ is the first projection. By definition the specification of the adjunction is equivalent to providing a bijection

$$\widetilde{(-)} : \mathbb{C}(U^n \times U, U)(\pi_1^*(f), g) \rightleftarrows \mathbb{C}(U^n, U)(f, \forall_n(g)) \widehat{(-)},$$

natural in f and g, where $\pi_1^* = \mathbb{C}(\pi_1, U) : \mathbb{C}(U^n, U) \longrightarrow \mathbb{C}(U^n \times U, U)$. The **($\forall$-l)** and **($\forall$-r)** rules can then be interpreted as follows. Given

$$[\![\Gamma \vdash s : A]\!] \in \mathbb{C}(U^n \times U, U)(\pi_1^*([\![\Gamma]\!]), [\![A]\!]),$$

i.e. α is free in Γ as stipulated in the side-condition for $(\forall\text{-}\mathbf{r})$, we set

$$[\![\Gamma \vdash \mathbf{\Lambda}\alpha.s : \forall\alpha.A]\!] = \widetilde{[\![\Gamma \vdash s : A]\!]}$$

and

$$[\![\Gamma, z : \forall\alpha.A \vdash \langle z^{\forall\alpha.A}\, , B/y^{A[B/\alpha]}\rangle t : C]\!] = \\ (\mathsf{id}_{[\![\Gamma]\!]} \otimes \mathbb{C}(\langle\, \mathsf{id}_{U^n}, [\![B]\!]\,\rangle, U)(\widehat{\mathsf{id}_{\forall_n([\![A]\!])}})); [\![\Gamma, y : A[B/\alpha] \vdash t : C]\!] \;.$$

Note that $(\forall\text{-}\gamma)$ is validated given $\mathsf{w} : Id_{\mathbb{C}} \longrightarrow I$ because of w's naturality.

Let us now consider $\forall_n; \mathbb{C}(f, U)$ and $\mathbb{C}(f \times \mathsf{id}_U, U); \forall_m$, both functors from $\mathbb{C}(U^n \times U, U)$ to $\mathbb{C}(U^m, U)$. We have already argued that they should be identical on objects. In fact much more should be required. Firstly, they should also act the same on morphisms. Secondly, consider the canonical natural transformation between them given at $h \in \mathbb{C}(U^n \times U, U)$ by

$$\widetilde{\mathbb{C}(f \times \mathsf{id}_U, U)(\widehat{\mathsf{id}_{\forall_n(h)}})}.$$

In order for our semantics to be sound one should require the ***Beck-Chevalley condition*** [31] to hold: the canonical transformation must be the identity. The role of this condition is to accommodate the commutativity of substitution and universal quantification not only for types but also terms, and validate η-expansion.

5.8 Light affine hyperdoctrines

To wrap up this chapter, we list all the required properties below. We call the categorical semantics of IMLAL2 light affine hyperdoctrines.

Definition 5.9. A ***light affine hyperdoctrine*** is specified by the following data:

- A category $\mathbb{C}$ with finite products, which consists of a distinguished object U that generates all other objects using the operation of forming finite products. $\mathbb{C}$ is called the ***base*** category of the hyperdoctrine.
- A strict $\mathbb{C}$-indexed light affine category

$$\mathbb{C}(\_, U) : \mathbb{C}^{op} \longrightarrow LACat.$$

- For each object U^n $(n \in \mathbb{N})$ of $\mathbb{C}$ we are given a functor $\forall_n : \mathbb{C}(U^n \times U, U) \longrightarrow \mathbb{C}(U^n, U)$, which is right adjoint to the functor $\mathbb{C}(\pi_1, U) : \mathbb{C}(U^n, U) \longrightarrow \mathbb{C}(U^n \times U, U)$, where $\pi_1 : U^n \times U \longrightarrow U^n$ is the first projection. Moreover, the Beck-Chevalley condition holds: for any $f \in \mathbb{C}(U^m, U^n)$

– the diagram of functors

$$\begin{tikzcd}
\mathbb{C}(U^n \times U, U) \arrow[r, "\forall_n"] \arrow[d, "\mathbb{C}(f \times \mathsf{id}_U, U)"'] & \mathbb{C}(U^n, U) \arrow[d, "\mathbb{C}(f, U)"] & U^m \arrow[d, "f"] \\
\mathbb{C}(U^m \times U, U) \arrow[r, "\forall_m"'] & \mathbb{C}(U^m, U) & U^n
\end{tikzcd}$$

commutes,

– the canonical natural transformation between

$$\forall_n; \mathbb{C}(f, U) \qquad \text{and} \qquad \mathbb{C}(f \times \mathsf{id}_U, U); \forall_m$$

is an identity.

Since all our definitions have been motivated by the reductions in the corresponding rewriting theory, it is rather simple to prove a soundness result for the categorical semantics. As usual the completeness part would require the construction of a classifying syntactic category, but due to the auxiliary nature of this chapter we omit this issue here and proceed to the main subject of the thesis, game models.

Chapter 6

Game models

Traditional model theory is concerned with finding models of provability. Each formula of the logic in question can then be related with some property of the model. A model is *sound* if all conditions corresponding to provable formulas are fulfilled. Most sound models will also satisfy a number of properties which cannot be proved in the logic yet can be expressed using its language. Such models are called *incomplete*. *Complete* models do exist though and their construction is typically based on the syntax of the logic. This does not seem to contribute more information about the logic than the specification of the logic itself.

Denotational semantics strives to interpret proofs rather than study their mere existence i.e. provability. Denotational models are categories of an appropriate type whose objects are used to interpret formulas and whose morphisms denote proofs. A sound denotational model provides a morphism for each proof in such a way that proofs deemed equal are assigned the same ones. Most sound models also contain undefinable elements—morphisms that are not denotations of any proofs. Denotational models containing only definable elements are called ***fully complete*** and are also complete in the traditional sense if we identify truth inside the model with the existence of a morphism. In general the notion of full completeness is much stronger: not only should there be no morphisms associated with unprovable formulas, but also those corresponding to provable formulas must all be interpretations of proofs. Given a fully complete model, we are guaranteed that its elements represent proofs, but there may be fully complete models, where a single element denotes two different ones. This is not desirable for our purposes as we are going to represent various datatypes using proofs. We will be interested in finding models whose morphisms are in one-to-one correspondence with the space of all proofs. Such models are called ***fully and faithfully complete***. They exist, but only until recently could be constructed only by recourse to syntax. This has changed with the onset of game semantics. The terms *full completeness* and *full and faithful completeness* originate from categorical type theory, which views models as structure-preserving functors from the corresponding syntactic category.

A model is fully (respectively faithfully) complete just in case the induced functor is full (respectively faithful).

Game semantics has provided the first fully and faithfully complete models of various fragments of linear logic, notably multiplicative linear logic (MLL) with the MIX rule [3](the first such result), without the MIX rule [63] and multiplicative additive linear logic (MALL) [10]. The full completeness problem for full linear logic is still open, but there exist game models modelling MELL in a less satisfactory fashion [14, 15]. The methodology has also proved successful with regard to intuitionistic logic [102] and second-order polymorphism [60, 102]. Its most celebrated achievements are the advances in research on full abstraction, where the results include syntax-independent characterizations of observational equivalence for programming languages with a variety of features (purely functional call-by-name PCF [6, 64], PCF with control [78], call-by-value languages [8, 59], Idealized Algol with active [7] and passive expressions [9], finite non-determinism [51], probabilistic languages [32], languages with local exceptions [79]). The first of these results solved a long-standing open problem of finding a fully abstract semantics for PCF.

In this chapter we apply game-semantic techniques to light affine logic. The models possess the categorical structure described in Chapter 5 and will be proved fully and faithfully complete with respect to the notion of canonical nets from Chapter 4. Because IMLAL2 corresponds to polynomial-time computability, fully and faithfully complete models for IMLAL2 give a semantic characterization of P and FP.

6.1 Games and strategies

We consider two-player games between P (Proponent) and O (Opponent) in the AJM style [3, 6, 1]. Every play is started by O and thereafter it alternates between P and O.

Definition 6.1. A ***game*** G is a triple $\langle\, M_G, \lambda_G, P_G \,\rangle$ where

- M_G is a set of moves,
- $\lambda_G : M_G \longrightarrow \{\, \mathrm{O}, \mathrm{P} \,\}$ partitions moves into those that O can make or **O-*moves***, and those that P can make or **P-*moves*** (we will write M_G^O, M_G^P for the set of O-moves and P-moves of G respectively),
- P_G is a prefix-closed subset of M_G^{alt} (the set of finite alternating sequences of moves from M_G, each beginning with an O-move); we call elements of P_G ***positions*** or ***plays***.

For example, $\varnothing = \langle\, \varnothing, \varnothing, \{\, \epsilon \,\} \,\rangle$ (where ϵ is the empty sequence) is a game called the ***empty game***. Given a fixed set $\mathcal{T}$ of tokens, there is a family of single-move games G_d in which it is only possible to play the token $d \in \mathcal{T}$:

$$G_d = \langle\, \{\, d \,\}, \{\, (d, \mathrm{O}) \,\}, \{\, \epsilon, d \,\} \,\rangle\,.$$

We call the games ***atomic***. A slightly more interesting game is:

$$G' = \langle\, \{\, a, b, c \,\}, \{\, (a, \mathrm{O}), (b, \mathrm{P}), (c, \mathrm{P}) \,\}, \{\, \epsilon, a, ab, aba, abac \,\} \,\rangle.$$

In general, M_G and P_G need not be finite.

Games can be combined to produce more complex ones. We take advantage of two standard constructions called the ***tensor game*** and the ***linear function space game*** corresponding to the two IMAL connectives. The tensor game is defined by

$$\begin{array}{rcl} M_{A\otimes B} & = & M_A + M_B \\ \lambda_{A\otimes B} & = & [\lambda_A, \lambda_B] \\ P_{A\otimes B} & = & \{\, s \in M^{alt}_{A\otimes B} \mid s \restriction A \in P_A,\ s \restriction B \in P_B \,\} \end{array}$$

where $M_A + M_B$ is a disjoint sum of the two sets, $\lambda_{A\otimes B}$ is the canonical map $[\lambda_A, \lambda_B] : M_A + M_B \longrightarrow \{\, O, P \,\}$ and $s \restriction A$ (respectively $s \restriction B$) means the subsequence of s consisting of moves which come from A (respectively B). It is a consequence of this definition that every $s \in P_{A\otimes B}$ satisfies the ***O-Switching Condition***: for each pair of consecutive moves mm' in s, if m and m' are from different components (i.e. one is from A, the other from B), then m' is an O-move. Plays of the tensor game may start either in A or in B. For example:

$$\begin{array}{rcl} M_{G_d\otimes G'} & = & \{\, a, b, c, d \,\} \\ \lambda_{G_d\otimes G'} & = & \{\, (a, \mathrm{O}), (b, \mathrm{P}), (c, \mathrm{P}), (d, \mathrm{O}) \,\} \\ P_{G_d\otimes G'} & = & \{\, \epsilon, a, d, ab, aba, abd, abac, abacd \,\}. \end{array}$$

The linear function space game (or ***linear arrow game***) is constructed as follows:

$$\begin{array}{rcl} M_{A\multimap B} & = & M_A + M_B \\ \lambda_{A\multimap B} & = & [\overline{\lambda}_A, \lambda_B] \\ P_{A\multimap B} & = & \{\, s \in M^{alt}_{A\multimap B} \mid s \restriction A \in P_A,\ s \restriction B \in P_B \,\}. \end{array}$$

$\overline{\lambda}_G : M_G \longrightarrow \{\, \mathrm{O}, \mathrm{P} \,\}$ reverses the ownership of moves in G ($\overline{\lambda}(m) = \mathrm{O}$ if and only if $\lambda(m) = \mathrm{P}$). The game satisfies an analogous ***P-Switching Condition***. The initial moves of $A \multimap B$ are the (copies of) initial moves in B. Here are two examples:

$$\begin{array}{rcl} M_{G_d\multimap G'} & = & \{\, a, b, c, d \,\} \\ \lambda_{G_d\multimap G'} & = & \{\, (a, \mathrm{O}), (b, \mathrm{P}), (c, \mathrm{P}), (d, \mathrm{P}) \,\} \\ P_{G_d\multimap G'} & = & \{\, \epsilon, a, ad, ab, aba, abad, abac \,\} \end{array}$$

$$\begin{aligned} M_{G' \multimap G_d} &= \{\, a, b, c, d \,\} \\ \lambda_{G' \multimap G_d} &= \{\, (a, \mathrm{P}), (b, \mathrm{O}), (c, \mathrm{O}), (d, \mathrm{O}) \,\} \\ P_{G' \multimap G_d} &= \{\, \epsilon, d, da, dab, daba, dabac \,\}. \end{aligned}$$

Observe the small difference between $((G_c \multimap G_a) \multimap G_b) \multimap G_a$ and G'. The former has three moves, the latter has four, but otherwise they are the same.

Games of some kind will always be objects in our models. Their morphisms will be strategies. Informally, strategies advise the Proponent what move to make next, but they are not required to provide the information all the time. If P adopts a strategy and it does not offer any moves, P cannot proceed further and the play is over. Informally, we say that P loses then.

Definition 6.2. A ***deterministic P-strategy***, or simply ***strategy***, for a game G is a non-empty, prefix-closed subset σ of P_G satisfying:

(i) for any even-length s, if $s \in \sigma$ and $sm \in P_G$ then $sm \in \sigma$,

(ii) (*determinacy*) if even-length sm and sm' are both in σ, then $m = m'$.

We write $\sigma : G$ then.

Example 6.3. There are three strategies for G'. The strategy $\{\, \epsilon, a \,\}$ provides no instructions at all; the next one $\{\, \epsilon, a, ac, aca \,\}$ tells P how to respond to the first O-move, but does not offer any advice after the next O-move; finally, P can always rely on $\{\, \epsilon, a, ac, aca, acab \,\}$ for help.

Now we are ready to construct a category of games and strategies. As its objects we take the collection of all games. The morphisms between two games G_1 and G_2 will be strategies for the game $G_1 \multimap G_2$, hence we have to show how to compose $\sigma : G_1 \multimap G_2$ with $\tau : G_2 \multimap G_3$. The idea is to allow the two strategies to interact in order to produce a strategy for $G_1 \multimap G_3$.

Moves of G_2 can provide common ground for interaction as each of them has a double identity: if a move from G_2 is an O-move in $G_1 \multimap G_2$, then it is a P-move in $G_2 \multimap G_3$ and vice versa. The new positions of $G_1 \multimap G_3$ can be developed by playing the two strategies against each other and hiding moves from G_2. If a move is played in G_2, we will try to engage one of the strategies to prolong the sequence by letting it see just the moves from two components including G_2. If the strategy, say σ, responds in G_2, the move will be seen as an O-move in $G_2 \multimap G_3$ and so τ may be asked to react to it when shown the state of the game in G_2 and G_3. That interaction may remain in G_2 forever, in which case the response of the composite strategy will be undefined. However, once the interaction produces a move in G_1 or G_3, it is considered to be the reaction of the composite strategy.

Formally, for any games G_1, G_2 and G_3 we define $\mathcal{L}(G_1, G_2, G_3)$ to be the set of finite sequences s of moves from $M_{G_1} + M_{G_2} + M_{G_3}$ such that for any pair of consecutive moves mm' in s, if $m \in M_{G_i}$ and $m' \in M_{G_j}$ then $|i - j| \leqslant 1$. We call $\mathcal{L}(G_1, G_2, G_3)$ the set of ***interaction sequences*** over (G_1, G_2, G_3). Note that there will always be intervening moves from G_2 between a move from G_1 and another from G_3.

Let σ and τ be strategies for the games $G_1 \multimap G_2$ and $G_2 \multimap G_3$ respectively. We define the ***composite*** $\sigma\,;\tau$ of σ and τ for $G_1 \multimap G_3$ as

$$\sigma\,;\tau = \{\, s \restriction (G_1, G_3) \mid s \in \mathcal{L}(G_1, G_2, G_3),\ s \restriction (G_1, G_2) \in \sigma,\ s \restriction (G_2, G_3) \in \tau \,\}.$$

Example 6.4. Consider the following games:

$$\begin{array}{rcl}
G_1 & = & (\{\, a, b \,\}, \{\, (a, \mathrm{P}), (b, \mathrm{O}) \,\}, \{\, \epsilon, b, ba \,\}),\\
G_2 & = & (\{\, c, d, e, f \,\}, \{\, (c, \mathrm{P}), (d, \mathrm{O}), (e, \mathrm{P}), (f, \mathrm{O}) \,\},\\
 & & \quad \{\, \epsilon, d, dc, f, fe, dcf, dcfe, fed, fedc \,\}),\\
G_3 & = & (\{\, g, h \,\}, \{\, (g, \mathrm{P}), (h, \mathrm{O}) \,\}, \{\, \epsilon, h, hg \,\});
\end{array}$$

and strategies $\sigma : G_1 \multimap G_2$, $\tau : G_2 \multimap G_3$:

$$\begin{array}{rcl}
\sigma & = & \{\, \epsilon, f, fg, d, db, dba, dbac, fed, fedb, fedba, fedbac \,\}\\
\tau & = & \{\, \epsilon, h, hf, hfe, hfed, hfedc, hfedcg \,\}
\end{array}$$

The only maximal interaction sequence over (G_1, G_2, G_3) is $hfedbacg$:

$$\begin{array}{ccc}
G_1 & G_2 & G_3\\
 & & h\\
 & f & \\
 & e & \\
 & d & \\
b & & \\
a & & \\
 & c & \\
 & & g
\end{array}$$

Hence, we have $\sigma;\tau = \{\epsilon, h, hb, hba, hbag\}$.

Let $\mathsf{id}_G : G \multimap G$ be the strategy that just copies moves between the right and the left instance of G starting from the right. id_G is called the ***identity strategy***. Indeed, we have $\mathsf{id}_{G_1};\sigma = \sigma$ and $\sigma;\mathsf{id}_{G_2} = \sigma$ for any $\sigma : G_1 \multimap G_2$. Composition of strategies is associative [1], so games and strategies form a category.

6.2 IMAL

A fully and faithfully complete model of IMAL is the first in the series of game models to be presented and serves as a basis for future extensions. The canonical nets for IMAL—which will be proved to correspond exactly to the elements of the model—were characterized uniquely by matchings between atomic nodes, each of which was reachable from the root. However, not all matchings define correct nets, so it is interesting to characterize those that do. In the setting of games, the good matchings will provide P with a strategy that always enables him to make a legal move after O has made one. This can be regarded as a crude notion of 'winning', which we will refine in the future and require that the 'victory' be also achieved in style. Typically, this will consist in introducing some extra constraints on the way P can play.

6.2.1 IMAL games

IMAL formulas are generated from atoms and the connectives $\otimes$ and $\multimap$. Similarly, assuming the set of tokens coincides with the set of atoms, ***IMAL games*** will be constructed from atomic games using the tensor and linear arrow game constructions. We also include the empty game to interpret the unit. Using switching conditions one can define the games independently of the syntax through regular expressions that capture plays.

First we make the two game constructions introduced earlier less abstract, by using tags to implement the disjoint sum: $X + Y = \{\, l \,\} \cdot X \cup \{\, r \,\} \cdot Y$ for tensor and $X + Y = \{\, L \,\} \cdot X \cup \{\, R \,\} \cdot Y$ for linear function space ('$\cdot$' stands for concatenation of sets). This concretization is not so important for IMAL as it will be for its extensions: the letters give a rough idea of the size of the game, which becomes crucial once the games are allowed to evolve. For example, the game $(G_a \multimap G_b) \otimes G_c \multimap G_d$ corresponding to the formula $(a \multimap b) \otimes c \multimap d$ is

$$\begin{array}{rcl} M_G & = & \{\, LlLa, LlRb, Lrc, Rd \,\}, \\ \lambda_G & = & \{\, (LlLa, \mathrm{O}), (LlRb, \mathrm{P}), (Lrc, \mathrm{P}), (Rd, \mathrm{O}) \,\}, \\ P_G & = & \{\epsilon, (Rd), (Rd)(LlRb), (Rd)(Lrc), (Rd)(LlRb)(LlLa), \\ & & (Rd)(LlRb)(LlLa)(Lrc)\}. \end{array}$$

We shall often just write a when G_a is meant. In general, any move of an IMAL game has the form $m = sa$, where $s \in \{\, l, r, L, R \,\}^*$ and $a \in \mathcal{T}$. That shape enables us to define λ_G directly from moves:

> m is a P-move if and only if s contains an odd number of L's.

From now on, we are going to separate the token from the rest of the move and refer to s as a move and to a as its ***token***.

Definition 6.5. ***Games over token set*** $\mathcal{T}$ and alphabet X are triples

$$G = \langle\, M_G, \mu_G, P_G \,\rangle$$

where $M_G \subseteq X^*$, $\mu_G : M_G \longrightarrow \mathcal{T}$, $P_G \subseteq (M_G)^{alt}$ and $X = \{\, l, r, L, R \,\}$.

The atomic games are represented in this convention as

$$G_a = \langle\, \{\, \epsilon \,\}, \{\, (\epsilon, a) \,\}, \{\, \epsilon, \text{‘}\epsilon\text{’} \,\} \,\rangle\,.$$

The first element of P_{G_a} is the empty string, the second is a non-empty string whose only character is the empty string. The μ function in the game $(G_a \multimap G_b) \otimes G_c \multimap G_d$ is

$$\{\, (LlL, a), (LlR, b), (Lr, c), (R, d) \,\}.$$

6.2.2 IMAL strategies

We also adopt a more concrete definition of strategies in which $M_G \subseteq X^*$ as in the new definition of games. Recall that strategies tell P how to prolong the game in a given position, though in general a strategy will not cover all cases. IMAL proofs will turn out to be denoted by strategies which are always reliable. Let us introduce several properties that will help to characterize them.

If, for every odd-length $s \in \sigma$, there is some m such that $sm \in \sigma$, we say that σ is ***total***. If for any even-length $sm_Om_P \in \sigma$, the tokens of m_O and m_P are identical (i.e. $\mu_G(m_O) = \mu_G(m_P)$), then σ is said to be ***token-reflecting***. We say that σ is ***history-free*** if there is a partial function $f : M_G^O \rightharpoonup M_G^P$ such that for any odd-length $sm \in \sigma$, we have $smm' \in \sigma$ if and only if $f(m)$ is defined, $f(m) = m'$ and $smm' \in P_G$. We write $\sigma = \sigma_f$ just in case f is the least such function[1]. Further, σ is said to be ***injective history-free*** if the least such f is injective.

Proposition 6.6. For IMAL games the following families of strategies compose:

- total strategies,
- token-reflecting strategies,
- history-free strategies,
- injective history-free strategies.

[1]This definition follows [3] rather than [5], where a stronger property was required to hold: if P followed a history-free strategy σ_f, whenever f suggested a move, it would be correct. In the current definition f may sometimes err, although it must give a playable move for at least one position.

All these properties except token-reflection are also meaningful in the broader setting of games and give rise to lluf subcategories (same objects) of the general category of games and strategies. Let $\mathcal{G}$ be the category whose objects are IMAL games and morphisms are strategies satisfying all of the above properties. We call the latter ***winning (IMAL) strategies***.

Example 6.7. The strategy σ_f for

$$(b \otimes e) \otimes (e \multimap (c \multimap d) \otimes a) \multimap (a \otimes b) \otimes (c \multimap d)$$

generated by

$$f : \begin{cases} Rll & \mapsto & LrRr & Rlr & \mapsto & Lll \\ LrRlL & \mapsto & RrL & RrR & \mapsto & LrRlR \\ LrL & \mapsto & Llr & & & \end{cases}$$

is an IMAL winning strategy. Its maximal positions are

$(Rlr)(Lll)$	bb
$(Rll)(LrRr)(LrL)(Llr)$	$aaee$
$(RrR)(LrRlR)(LrL)(Llr)$	$ddee$
$(RrR)(LrRlR)(LrRlL)(RrL)(Rlr)(Lll)$	$ddccbb$
$(RrR)(LrRlR)(LrRlL)(RrL)(Rll)(LrRr)(LrL)(Llr)$	$ddccaaee$

In the right column we have listed the corresponding sequences of tokens.

Theorem 6.8. $\mathcal{G}$ is a model of IMAL.

Proof. As set out in Chapter 5, all we need to show is that $\mathcal{G}$ is an autonomous category equipped with a natural transformation $\mathsf{w} : Id_{\mathcal{G}} \longrightarrow I$.

The action of $\otimes$ on objects is simply the tensor game construction. For strategies $\sigma_1 : A_1 \longrightarrow B_1$ and $\sigma_2 : A_2 \longrightarrow B_2$, $\sigma_1 \otimes \sigma_2 : A_1 \otimes A_2 \longrightarrow B_1 \otimes B_2$ is the strategy that plays according to σ_1 or σ_2 in a 'non-communicating way':

$$\sigma_1 \otimes \sigma_2 = \{ s \in P_{A_1 \otimes A_2 \multimap B_1 \otimes B_2} \quad | \quad s \upharpoonright (A_1, B_1) \in \sigma_1, \quad s \upharpoonright (A_2, B_2) \in \sigma_2 \}.$$

It is worth noticing that the strategy which is being followed can only be changed by O when he switches between B_1 and B_2 on the right. It easily follows that $(\sigma_1; \tau_1) \otimes (\sigma_2; \tau_2) = (\sigma_1 \otimes \sigma_2); (\tau_1 \otimes \tau_2)$ and $\mathsf{id}_{A \otimes B} = \mathsf{id}_A \otimes \mathsf{id}_B$.

Observe that the games $C \otimes A \multimap B$ and $C \multimap (A \multimap B)$ are 'identical' modulo a renaming of moves across the bijection $(M_C + M_A) + M_B \cong M_C + (M_A + M_B)$. Therefore, the desired adjunction takes on a trivial form. The tensor unit I in our context is the empty game. Hence, $G \multimap I = I$ for any G and in consequence I is a terminal object. The unique morphisms into I are simply empty strategies, which are all part of the (unique) natural transformation w. The canonical maps $\mathsf{id}, \mathsf{a}, \mathsf{c}, \mathsf{l}, \mathsf{r}$ copy moves from one side to another and are obviously isomorphic winning strategies. Because of their copy-cat nature, the transformations they induce are natural and the defining diagrams commute. $\square$

$\mathcal{G}$ will be proved fully and faithfully complete. The proof consists in exhibiting a correspondence between positions of a special kind and paths in canonical affine essential nets. In this way we will also show that to check whether a token-reflecting injective history-free strategy is total, it suffices to test its responses assuming that O plays in a very restricted way.

Let us begin with the definition of the *enabling* (or *justification*) relation [64, 101] between moves of an IMAL game. The set $in(G) \subseteq M_G$ of initial moves of an IMAL game G is defined by

$$in(G) = \{\, m \mid ms \in P_G \,\}.$$

Definition 6.9. For $x, y \in M_G$, where G is an IMAL game, we say that y ***enables*** (or ***justifies***) x if G has the form $C[X \multimap Y]$ for some one-holed context C[2], IMAL games X and Y such that $x \in in(X)$ and $y \in in(Y)$.

Note that if x is enabled by y, the occurrence of $\multimap$ mentioned in the definition is unique. We call it the *enabler*. In $((a \otimes b) \multimap_1 c) \multimap_2 d$ the move R enables LR, and LLl, LLr are enabled by LR. The enablers are $\multimap_2$ and $\multimap_1$ respectively. In IMAL games each move is either initial or it is enabled some other move(s). By definition of the $\multimap$ game when a non-initial move is made in a play, some move enabling it must have already been made.

Definition 6.10. In a given play an O-move is ***short-sighted*** if it is enabled by the P-move that precedes it. By convention, the first O-move in any position is considered short-sighted. A position (play) is ***short-sighted*** if every O-move therein is short-sighted.

For $a \otimes (b \multimap c) \multimap (d \multimap e) \otimes f$ the plays $ed, ecbd, fa$ are short-sighted, but $edf, ecbdfa$ are not. The notion of a short-sighted move has been considered in [102] for HO/N-games [64, 101], where it plays an important role: short-sighted positions are simply P-views. It turns out that this notion is also useful in our case, although different methods must be used to analyze it. For example, a result analogous to the next lemma holds trivially in the innocent framework, though it is not so obvious in ours.

Our first claim that for any short-sighted position sp (of an IMAL game) ending in a P-move p: if p enables an O-move o then spo is a position. From the definition of the linear arrow game we know that o can be played at sp unless it has already been used in s. The lemma below rules out such a possibility.

[2]

$$C ::= \quad C \multimap H \quad | \quad H \multimap C \quad | \quad C \otimes H \quad | \quad H \otimes C \quad | \quad [\,]$$

Lemma 6.11. Let G be a IMAL game. Suppose p_1, p_2 are P-moves such that an O-move o is enabled by both p_1 and p_2. G has no short-sighted position of the following shape: $\cdots p_1 o \cdots p_2 \cdots$.

Proof. W.l.o.g. assume $G = C_1^-[O \multimap C_2^-[P_1 \otimes P_2]]$[3], where o, p_1, p_2 are initial moves of O, P_1, P_2 respectively. Note that between p_1 and p_2 an O-move from P_1 must be made. Consider the earliest such move o' in s i.e. there are not any moves from P_1 between p_1 and o'. However, in a short-sighted position o' has to be preceded by an enabling P-move. A move enabling o' must come from P_1, which is a contradiction. □

Hence, if a short-sighted move is available to O and O has always played short-sightedly, O will be able to use his short-sighted move to extend the interim position. This determines a short-sighted 'strategy' for O. As it happens, such strategies will be shown representative of O's potential: we are going to show that if O cannot beat P by playing short-sighted moves, O will not be able to do so in any other way. Our next step is to define strategies for P that take only short-sighted O-moves into account.

Definition 6.12. A ***short-sighted*** strategy σ for G is a non-empty, prefix-closed subset P_G such that for any even-length $s \in \sigma$, if $sm \in P_G$ then

$$sm \in \sigma \text{ if and only if } m \text{ is short-sighted.}$$

A ***weakly winning (IMAL) strategy*** is a short-sighted strategy which is injective history-free, token-reflecting and total (clearly the properties make sense for short-sighted strategies too).

We write σ^f for the weak IMAL strategy generated by $f : M_O \rightharpoonup M_P$. Note that if f generates a winning strategy σ_f, its restriction to positions in which O plays short-sightedly will generate a weakly winning strategy. In fact, as we next show, it will be equal to σ^f.

Lemma 6.13. If $sop \in \sigma_f$ and p is a P-move, there exists $s' \in \sigma^f$ such that $s'op \in \sigma^f$.

Proof. Let $s'o$ be the 'P-view' of so. $s'o$ is then a (short-sighted) position: O-moves therein are legal because of their short-sightedness and so are P-moves by totality of σ_f. □

[3]

$$\begin{array}{lllllllll} C^- ::= & C^+ \multimap H & \mid & H \multimap C^- & \mid & C^- \otimes H & \mid & H \otimes C^- & \\ C^+ ::= & C^- \multimap H & \mid & H \multimap C^+ & \mid & C^+ \otimes H & \mid & H \otimes C^+ & \mid \ [\,] \end{array}$$

Example 6.14. σ^f for the function f from Example 6.7 contains the following maximal positions (we have only used tokens this time)

$$aaee \qquad bb \qquad ddee \qquad ddcc.$$

6.2.3 Affine nets defined by strategies

Short-sighted O-moves have a nice interpretation in essential nets. Suppose G is an IMAL game and consider the syntactic tree of the corresponding polarized formula $\ulcorner G \urcorner$ in which branches have been oriented according to the directional rules for IMAL nets (branches from $\invamp^+$-nodes to their sinks must be deleted as well). We call this 'oriented' tree $\mathcal{T}_{\ulcorner G \urcorner}$. Analogously, we can consider $\mathcal{T}_{\llcorner G \lrcorner}$. When referring to nodes of such trees we use the same terminology as for the nets, i.e. their leaves are polarized atomic nodes and the internal nodes are polarized connectives that have premises rather than children. By analogy to games we distinguish initial atomic nodes for each node of $\mathcal{T}_{\ulcorner G \urcorner}$:

$$\begin{array}{rclrcl} in(a^+) & = & \{ a^+ \} & in(a^-) & = & \{ a^- \} \\ in(\otimes^+) & = & in(n_1^+) \cup in(n_2^+) & in(\otimes^-) & = & in(n_2^-) \\ in(\invamp^+) & = & in(n_2^+) & in(\invamp^-) & = & in(n_1^-) \cup in(n_2^-) \end{array}$$

where n_1^x, n_2^y $(x, y = +, -)$ correspond to the premises of the respective links. Suppose o, p are an O-move and a P-move in G. Let o^+ and p^- be the corresponding atomic nodes in $\mathcal{T}_{\ulcorner G \urcorner}$. Now we have $m \in in(G)$ if and only if $m^+ \in in(\text{root of } \mathcal{T}_{\ulcorner G \urcorner})$ (or equivalently $m^- \in in(\text{root of } \mathcal{T}_{\llcorner G \lrcorner})$). Initial nodes are neatly characterized by paths in the tree:

Lemma 6.15. Call a path *monotone* if the nodes it visits have the same polarity.

(i) $o^+ \in in(n)$ if and only if there exists a monotone path from n to o^+.

(ii) $p^- \in in(n)$ if and only if there exists a monotone path from p^- to n.

Moreover, the respective paths are unique once they exist.

Proof. The proof is a simple case-by-case analysis and we omit it. We encourage the reader to check the claims for the trees arising from essential nets in Figures 4.2, 4.3 (after removing their axiom links). $\square$

The Lemma implies an appealing characterization of short-sighted O-moves:

Theorem 6.16. A P-move p enables an O-move o in G if and only if there exists a $\mathcal{T}_{\ulcorner G \urcorner}$-path from p^- to o^+.

Proof. Suppose p enables o i.e. $G = C^-[X \multimap Y]$, $o \in in(X)$ and $p \in in(Y)$. Consider the $\otimes^-$-node n corresponding to the enabler $\multimap$. Let n_1^+ and n_2^- be its premises. By (i) from the previous Lemma there exists a path from n_1^+ to o^+, by (ii) there exists a path from p^- to n_2^-. Hence p^- and o^+ are connected by a $\mathcal{T}_{\ulcorner G \urcorner}$-path.

Conversely, suppose there is a path from p^- to o^+. Note that if an edge leads from a to b, then both nodes have the same polarity unless a is a $\otimes^-$-node (b is then positive). Hence the path from p^- to o^+ must pass through a $\otimes^-$-node n which is entered from some negative node n_2^-. Let n_1 be the positive premise of n. By (i) and (ii), $o^+ \in in(n_1)$ and $p^- \in in(n_2)$, hence p enables o. Moreover, the named paths are unique. □

Given a weakly winning strategy σ^f for G we define the corresponding net $\mathcal{N}_f$ from $\mathcal{T}_{\ulcorner G \urcorner}$ in two steps.

(1) For each pair of moves o, p such that $f(o) = p$ we connect the corresponding atomic nodes p^+ and o^- with an axiom link. The resultant graph will be called $\mathcal{T}_f$.

$\mathcal{T}_f$ is not always an essential net, because there might be atomic nodes not connected by axiom links. Thanks to the links one can interpret each position from σ^f as a $\mathcal{T}_f$-path from the root to the atomic node corresponding to the final move of the position:

- initial O-moves are interpreted by paths that exist by (i) of Lemma 6.15,
- each P-move corresponds to crossing an axiom link,
- non-initial (short-sighted) O-moves are represented by paths as described in Theorem 6.16.

It follows that each link added in (1) is reachable (otherwise σ^f would not be generated by f). Conversely, any path from the root to an atomic node which visits each atomic node at most once corresponds to some position from σ^f. We prove that the reachability relation in $\mathcal{T}_f$ already satisfies one of the correctness properties for essential nets:

Lemma 6.17. Each $\mathcal{T}_f$-path from the root to the sink of a $\invamp^+$-node visits the $\invamp^+$-node first.

Proof. Consider a path from the root to the sink of a $\invamp^+$-node with any cycles removed. Clearly, it must have crossed an axiom link. Let p^- be the last atomic node (it is necessarily negative so we write p^-) visited by the path before it reached the sink. Consider the position in σ^f that ends in p. By Lemma 6.15 we have $G = C^+[X \multimap Y]$ and $p \in in(X)$ for some X and Y. Because the position includes a move from X, by the definition of the $\multimap$ game, it must contain an initial move from Y. Therefore, the original path must have passed through an initial atomic node of $\ulcorner Y \urcorner$. However, each path that visits an initial node of Y would have to visit the $\parr^+$-node in question too. $\square$

Next we show what to do with the atomic nodes of $\mathcal{T}_f$ that are not connected by axiom links. In the following 'unreachable' means 'unreachable from the root of $\mathcal{T}_f$'.

(2) For each unreachable node n, which is a premise of a reachable link, prune the tree rooted at n and insert a weakening node $\underline{\phi(n)}$ in its place. Let $\mathcal{N}_f$ be the resultant net.

The procedure leads to an affine net, provided n is a negative node and no axiom links are broken when the trees are being erased. The first condition holds, because a positive premise is always linked with the conclusion of the link of which it is a premise. Therefore, if the conclusion is reachable, so is the premise. The second condition follows by the same reasoning as in Lemma 4.48 (this time using Lemma 6.17 instead of Proposition 4.44), because all axiom links in $\mathcal{T}_f$ are reachable from the root. This completes the definition of an affine net $\mathcal{N}_f$ from a weakly winning strategy. Next we verify that $\mathcal{N}_f$ is correct.

Proposition 6.18. $\mathcal{N}_f$ is a correct canonical affine net.

Proof. We check conditions (i) and (ii) from Definition 4.3 (see Theorem 4.49). $\mathcal{N}_f$ is canonical by construction: the only nodes that are not reachable from the root are the weakening nodes.

(ii) has already been proved as Lemma 6.17 for $\mathcal{N}_f$ and step (2) does not affect it. It remains to prove that $\mathcal{N}_f$ is acyclic.

Suppose there is a cycle in $\mathcal{N}_f$. It must involve atomic nodes, as cycles cannot be formed solely by connective-nodes. Because all atomic nodes are reachable from the root, there exists an infinite path s from the root containing repeated occurrences of atomic nodes. Consider its longest initial subpath s' that ends in a negative atomic node p^- and does not visit any nodes twice. Let o^+ be the next atomic node s would visit. Note that this means that p enables o in G. Consider the position that corresponds to s': o already occurs in it, it ends in p and p enables o—precisely the configuration ruled out by Lemma 6.11. $\square$

Example 6.19. The essential net $\mathcal{N}_f$ corresponding to σ_f from Example 6.7 (via σ^f from Example 6.14) is almost exactly the one in Figure 4.3. It suffices to add a new $\invamp^-$-link (call it $\invamp_0^-$) below $\invamp_1^-$ and $\otimes_1^-$ and a $\invamp^+$-link (let it be $\invamp_0^+$) below the new $\invamp^-$-link and $\otimes_1^+$. The maximal short-sighted positions listed in Example 6.14 are then interpreted by the following paths:

aaee	$\invamp_0^+ \otimes_1^+ \otimes_2^+ a^+ a^- \invamp_2^- \otimes_1^- e^+ e^-$
bb	$\invamp_0^+ \otimes_1^+ \otimes_2^+ b^+ b^-$
ddee	$\invamp_0^+ \otimes_1^+ \invamp^+ d^+ d^- \otimes_2^- \invamp_2^- \otimes_1^- e^+ e^-$
ddcc	$\invamp_0^+ \otimes_1^+ \invamp^+ d^+ d^- \otimes_2^- c^+ c^-$

We have thus shown that σ_f defines a correct canonical net $\mathcal{N}_f$ via σ^f. As the function f corresponds exactly to the axiom links of $\mathcal{N}_f$ we have:

Theorem 6.20. $\mathcal{G}$ is a fully and faithfully complete model of IMAL with respect to affine IMAL nets.

As already mentioned, we have demonstrated that in order to check whether σ_f is total, it suffices to verify that the moves suggested by f lead P to a win over short-sighted Opponent. We will take advantage of that fact when illustrating the schematic nature of winning strategies next.

Suppose $\sigma_f : G$ is winning. We are going to show that it defines 'natural' *winning* strategies for games of the shape $G[G'_1/\tau_1, \cdots, G'_n/\tau_n]$, where $\tau_i \in \mathcal{T}$ and G'_i is an IMAL game ($1 \leqslant i \leqslant n$). The way these strategies will be defined and the shape of IMAL games allow us to reduce the problem to the simple case of $n = 1$ and $G'_1 = \tau_1 \otimes \tau_1$ or $G'_1 = \tau_1 \multimap \tau_1$; in these two cases we refer to $G[G'_1/\tau_1]$ as G' and G'' respectively. The strategies $\sigma_{f'} : G'$ and $\sigma_{f''} : G''$ are defined as follows:

1.

$$f'(x) = \begin{cases} f(x) & \mu_{G'}(x) \neq \tau_1 \\ f(o)r & \mu_{G'}(x) = \tau_1 \text{ and } x = or \\ f(o)l & \mu_{G'}(x) = \tau_1 \text{ and } x = ol \end{cases}$$

2.

$$f''(x) = \begin{cases} f(x) & \mu_{G''}(x) \neq \tau_1 \\ f(o)R & \mu_{G''}(x) = \tau_1 \text{ and } x = oR \\ oL & \mu_{G''}(x) = \tau_1 \text{ and } x = f(o)L \end{cases}$$

It is easy to see that $f' : M^O_{G'} \rightharpoonup M^P_{G'}$ and $f'' : M^O_{G''} \rightharpoonup M^P_{G''}$, because σ_f is injective history-free and token-reflecting.

Proposition 6.21. $\sigma_{f'} : G'$ and $\sigma_{f''} : G''$ are winning strategies.

Proof. The case of $\sigma_{f'}$ is easy, so we only tackle $\sigma_{f''}$. The only short-sighted positions to check (which are not already in σ_f) are those containing moves made according to the last two cases of the definition of f''. Observe that P-moves in the third case always end a short-sighted position, because they do not enable any O-moves. Therefore, it suffices to check that the strategy 'wins' in the following two cases.

- Consider $s(oR)$ for some $s \in \sigma_{f''}$. Then all moves in s must have been made as in the first or the second case of the definition of f'', so s can be seen as a position of σ_f (after all the trailing R's are erased). Thus $s(oR)(f(o)R)$ is a position in G'', because $sof(o)$ is a position in G.
- Consider $s(oR)(f(o)R)(f(o)L)$ for $s \in \sigma_{f''}$. Then $s(oR)(f(o)R)$ is a position of the first kind, i.e. it is a position in G in which some moves are adorned with R's. Hence, $s(oR)(f(o)R)(f(o)L)(oL)$ is a play in G'' by switching conditions.

$\square$

Proposition 6.21 shows that winning strategies provide a pattern which can be extended to generate winning strategies when the tokens unfold. The construction of f' and f'' could also be repeated for strategies that are injective history-free but not necessarily token-reflecting. Then the resultant strategies would not be total, because we could substitute a game for the P-token involved (leaving the corresponding O-token as it is) making it impossible for P to respond to the expanded O-moves, because of a shortage of moves. Therefore, token-reflection and injective history-freeness are equivalent to the ability of strategies to expand. In other words, our game model could be presented as consisting of expandable total and history-free strategies as was the case for the first fully-complete game model [3]. The extension relies on simply copying of moves back and forth and will be used to construct models of second-order linear logic and also to analyze equivalence of positions in IMLAL.

6.3 IMLAL

In IMAL games both the set of moves and the set of positions are finite. This is because each move represents an atomic formula and the game constructions do not allow repetitions of moves during play. Consequently, there is a bound on the length of positions—the number of available moves. This observation reflects the fact that the process of cut-elimination in MLL has linear time complexity. To express more complex computations, a restricted form of copying is admitted in IMLAL through the use of shriek games.

6.3.1 ! and § games

The $(!_0)$ and $(!)$ rules require that in the course of the game $!G$ we should be able to replay whatever was possible in G. Because of the contraction rule, we must also be prepared to contain two copies of $!G$ in a single game $!G$. Nevertheless, no communication is allowed between G and $!G$. This suggests modelling $!G$ as an interleaving of plays from G, where moves of $!G$ are 'promoted' copies of moves of G. The higher rank will enable us to distinguish moves of G from those of $!G$ and impose appropriate restrictions. Besides, the structure of the interleaving ought to be unambiguous: we must be able to assess when new (copies of) plays of G start in $!G$, when control is switched to an old strand of play and which old play it is. All of the postulates are vital, because even a slight departure from the regime jeopardizes our aim to achieve full completeness. Also, the ability to represent numeric functions crucially depends on the number of plays inside $!G$ games. A solution based on pointers in the style of HO/N-games [64] is possible, if G has only one initial move, but then $!G$ will already have more, so the construction will not work properly for $!!G$. Nor could we handle the $\otimes$ connective then. For a detailed discussion of related problems see [93]. The exponential of the AJM framework [5] satisfies all of our needs and we adopt the definition here. The equivalence of positions will have to be modified to suit our setting, though.

The construction of a ! game relies on tagging moves of G with natural numbers. Moves with the same tag should be viewed as belonging to the same copy of G inside $!G$.

$$\begin{array}{lcl} M_{!G} & = & \mathbb{N} \cdot M_G \\ \mu_{!G}(im) & = & \mu_G(m) \\ P_{!G} & = & \{s \in M_{!G}^{alt} \mid \forall i \in \mathbb{N}.\ s \upharpoonright i \in P_G\}\ . \end{array}$$

The numeric labels reflect the promotion of moves from G (e.g. moves of $!G$ are literally longer than the corresponding moves in G). Positions of $!G$ consist of a potentially infinite interleaving of plays of G in which the switching from one play to another can be performed only by O. The tags serve to distinguish different strands of play, so if O wishes to start a new round of G rather than continue some old play of G, he will choose a fresh label. This highlights the built-in O-switching condition: whenever two consecutive moves in a play of $!G$ come from different copies of G, the second must be an O-move. From this perspective $!G$ is like an infinite tensor product of copies of G.

Example 6.22. The players play out just one copy of the original game,

$$\begin{array}{llllllllll}
 & !(& (a & \multimap & b) & \otimes & (c & \multimap & d) &) \\
O & & & & 1lR & & & & & \\
P & & 1lL & & & & & & & \\
O & & & & & & & & 1rR & \\
P & & & & & & & 1rL & &
\end{array}$$

whereas the third move, shown below, opens a new instance of G.

$$\begin{array}{llllllllll}
 & !(& (a & \multimap & b) & \otimes & (c & \multimap & d) &) \\
O & & & & 1lR & & & & & \\
P & & 1lL & & & & & & & \\
O & & & & & & & & 2rR & \\
P & & & & & & & 2rL & &
\end{array}$$

Note that without the labels it would be impossible to tell whether the third move is an opening move in another copy of G.

The (§) and ($\S_0$) rules suggest that playing the game $\S G$ is similar to engaging in G. However, contrary to the previous case, there is no need for duplication, since contraction is forbidden for formulas of the shape $\S A$. Nevertheless, moves of $\S G$ should be distinguished from those of G as, e.g., $\S G \multimap G$ is not provable. We use $\star$ to that end and define $\S G$ by:

$$\begin{array}{lll}
M_{\S G} & = & \{\star\} \cdot M_G \\
\mu_{\S G}(\star m) & = & \mu_G(m) \\
P_{\S G} & = & \{s \in M_{\S G}^{alt} \mid s \restriction \star \in P_G\} .
\end{array}$$

6.3.2 IMLAL games

IMLAL games are generated by combining atomic and empty games using the four game constructions. They are games over $X = \{l, r, L, R, \star\} \cup \mathbb{N}$ in the sense of Definition 6.5. As before, $\lambda_G(m) = \mathrm{O}$ if and only if the number of L's in m is even.

A move of an IMLAL game has the form $i_1 i_2 \cdots i_n$ where each i_j ranges over $\{ l, r, L, R, \star \} \cup \mathbb{N}$. Let $i_{r_1} \cdots i_{r_d}$ be the subsequence of $i_1 \cdots i_n$ consisting only of numbers and $\star$'s. We define the ***depth*** of the move to be d, the *index at depth* j or j***-index*** to be the *sequence* $i_1 i_2 \cdots i_{r_j - 1} i_{r_j}$. The sequence $i_1 i_2 \cdots i_{r_j - 1}$ will be called a j***-base***. For example, the 1-, 2- and 3-indices of the move $2r \star Ll33R$ are

2, $2r\star$ and $2r \star Ll3$ respectively. The corresponding bases are ϵ, $2r$ and $2r \star Ll$. A game is said to have ***depth*** k just in case k is the maximum depth of its moves. As before, each move m has its token $\mu_G(m)$.

Example 6.23. Consider the provable formula $!(!a \otimes !b) \multimap !\S a$.

$$(R3\star)(L5l2) \text{ and } (R3\star)(L4r5)$$

are positions of the associated IMLAL game.

6.3.3 Network protocol

Fix an IMLAL game G of depth k. By an *i-index* of G, we mean the i-index of some move of G (so $i \leqslant k$). ***Threads of depth*** i, or simply i***-threads***, of G are named by i-indices of G. A thread named by θ is the set of moves of G whose i-index is θ (in particular they are prefixed by θ). An i-thread named by θ is said to be a ***P-thread*** if there are an odd number of occurrences of L in θ, otherwise it is said to be an ***O-thread***. θ is then called a P-index and an O-index respectively. If the rightmost symbol of the index θ is a number we say the i-thread is of ***!-type***, otherwise the thread is of **§*-type***.

For any game G, we write $T^P_{G,i}$ (respectively $T^O_{G,i}$) for the set of P-threads (respectively O-threads) of G at depth i. We shall omit the subscript i whenever we can get away with it. For any IMLAL games G_1 and G_2 it is easy to see that

$$\begin{array}{rclrclrcl} T^O_{G_1 \multimap G_2} & \cong & T^P_{G_1} + T^O_{G_2} & T^O_{G_1 \otimes G_2} & \cong & T^O_{G_1} + T^O_{G_2} & T_{!G_1} & \cong & \mathbb{N} \cdot T_{G_1} \\ T^P_{G_1 \multimap G_2} & \cong & T^O_{G_1} + T^P_{G_2} & T^P_{G_1 \otimes G_2} & \cong & T^P_{G_1} + T^P_{G_2} & T_{\S G_1} & \cong & T_{G_1} \end{array}$$

We shall often analyze a play s by considering its subsequence consisting of moves belonging to a given i-thread. We call that subsequence the i***-thread of the play*** s. (Thus the *i-thread of a game* is a set of moves, and the *i-thread of a play* is a sequence of moves.) It is straightforward to prove that any O-thread (respectively P-thread) of a play must begin with an O-move (respectively P-move).

Example 6.24. Consider the game $!!(a \multimap a) \multimap !!(a \multimap a)$ and the play

$$(R14R)\ (L23R)\ (L23L)\ (L29R)\ (L29L)\ (R14R)$$

which we refer to as $m_1 \cdots m_6$ in the picture (where only the respective numeric subsequences of the six occurrences are shown)

$$\begin{array}{lccccccc} & !!(a & \multimap & a) & \multimap & !!(a & \multimap & a) \\ m_1 & & & & & & & 14 \\ m_2 & & & 23 & & & & \\ m_3 & 23 & & & & & & \\ m_4 & & & 29 & & & & \\ m_5 & 29 & & & & & & \\ m_6 & & & & & 14 & & \end{array}$$

The two 1-threads of the play are m_1m_6 (named by $R1$) and $m_2m_3m_4m_5$ (named by $L2$); and the three 2-threads are m_1m_6 (named by $R14$), m_2m_3 (named by $L23$) and m_4m_5 (named by $L29$).

If a move has depth d, it will belong to threads at depths from 1 up to d. Note that they may be both O-threads and P-threads and that the fact that a move is in an O-thread at depth l does not mean that it will belong to O-threads at other depths. In particular a P-move may not be part of any P-threads (take m_6 for example).

The key idea leading to a good model of IMLAL is to organize threads at each depth into networks in accord with a global protocol. We will be interested in plays that obey the protocol at every depth (up to the depth of G of course). The protocol will be introduced step by step with each condition accompanied by a motivating example—a strategy satisfying all the properties introduced so far yet not representing any proofs.

Definition 6.25. A ***network*** at depth d consists of threads at depth d, only one of which is an O-thread. Each thread at depth d must be assigned to some network at depth d. Networks are disjoint, i.e. different networks cannot have common threads. At each depth each P-move must be in the same network as the preceding O-move.

- Consider the obvious strategy for $a \multimap !a$ whose maximal positions are $(Rn)L$ for $n \in \mathbb{N}$. The formula is unprovable, so we cannot afford to admit any strategies for the corresponding game to our framework. Indeed, the protocol as defined so far rules them out: the P-move (L) does not belong to any threads at depth 1, so it cannot be part of a network at that depth, but on the other hand, it should be in the same network as the first move.

  Similar reasoning applies to the unprovable formulas $!^k a \multimap !^l a$ for $l > k$. The second move will never be part of O's network at depth $k+1$. If $l < k$, the P-move will belong to a thread at depth $l+1$ and so it should be part of a network at that depth. But the initial O-move cannot belong to a thread at depth $l+1$: its depth is only l!

  The argument shows that each P-move must have the same depth as the preceding O-move. Observe that there is nothing wrong with $!a \multimap !a$, where we have networks consisting of Rn and Lm for $n, m \in \mathbb{N}$.

- Consider the respective token-reflecting strategies for the games denoting the unprovable formulas $(\Box_3 a \multimap \Box_2 b) \multimap \Box_1(a \multimap b)$ where each occurrence of

$\Box_i$ can be replaced with ! or §. The sequence of the first four moves will always have the shape:

$$(R?_1R)\ (LR?_2)\ (LL?_3)\ (R?_1L)$$

with $?_i = \star$ (if $\Box_i = \S$) or $?_i \in \mathbb{N}$. The fourth move is supposed to be in the same network as the third one, so the threads $LL?_3$ and $R?_1$ should be in the same network. The problem is they are both O-threads and thus come from different networks. Note that the network structure of the obvious strategy for

$$\S(a \multimap b) \multimap (\S a \multimap \S b)$$

is perfectly in order.

- Take the game $\S(a \otimes b) \multimap \S a \otimes \S b$ and the two positions $(Rl\star)(L \star l)$ and $(Rr\star)(L\star r)$. Again we show that the candidate token-reflecting strategy does not observe the protocol. The moves come from the three threads: O-threads $Rl\star$ and $Rr\star$ and the P-thread $L\star$. The plays violate the protocol, because the first would force $L\star$ to be in a network with $Rl\star$ and the second would place it together with $Rr\star$. Therefore, all the threads would have to be in the same network, but since two of them are O-threads, this is illegal. The obvious strategy for $\S a \otimes \S b \multimap \S(a \otimes b)$ is in order and gives rise to a correctly formed network consisting of the O-thread $R\star$ and two P-threads $Ll\star$ and $Lr\star$. There is a protocol-compliant strategy for $!(a \otimes b) \multimap !a \otimes !b$, but the threads Ln for $n \in \mathbb{N}$ must form networks with only one of Rln_1 and Rrn_2. In other words, for any $n \in \mathbb{N}$ strategies observing the protocol may use only one move with 1-index Ln, like the history-free strategy defined by:

$$Rln \mapsto L(2n)l \qquad Rrn \mapsto L(2n+1)r$$

Each network comprises only a finite number of threads.

- This is a necessary restriction indeed. Consider the strategy for

$$!(a \multimap a) \multimap \S(a \multimap a)$$

generated by:

$$R \star R \ \mapsto \ L1R \qquad LnL \ \mapsto \ L(n+1)R$$

for $n > 0$, which corresponds to the "infinite" Church numeral and is therefore undefinable. The strategy would require a network consisting of $R\star$ and Ln for all n.

A network whose O-thread is of !-type (call it a !-network) can have at most one P-thread which must also be of !-type.

- This requirement must be imposed in order to interpret the **(!)** rule properly. Consider the game $!!(a \multimap a) \multimap !!(a \multimap a)$ and the play from Example 6.24. Obviously the two 1-threads therein form a correct network at depth 1, but at depth 2, there would have to be a network consisting of $R14$, $L23$ and $L29$, which violates the proposed rule. At first this may seem too strong, because the formula is provable. Its proof will not be denoted by the strategy in question but by one that does comply with the protocol and produces plays of the shape:

$$(R14R)\ (L23R)\ (L23L)\ (R14L).$$

We also require some consistency in the structure of !-networks. Call two threads ***related*** if their indices i_1, i_2 are in_1 and in_2 respectively for some $n_1, n_2 \in \mathbb{N}$, i.e. they have the same d-base at some depth d.

IF TWO O-THREADS OF A !-NETWORK AT d ARE RELATED, THE CORRESPONDING P-THREADS (IF THEY EXIST) SHOULD ALSO BE RELATED.

This is because of the way !-networks are encapsulated in other networks by the **(§)** and **(!)** rules: each of the threads engaged in a !-network at depth $d+1$ is embedded into a unique thread at d. Besides, the optional P-threads must exist in both cases or do not exist in any.

- The condition rules out the strategy $\sigma_f :!!a \multimap \S!a$, where $f(R \star 2) = Lmn$ and $f(R \star 1) = Lm'n'$ for $m' \neq m$, which would not represent any proofs.

The protocol and the shape of IMLAL games imply switching conditions for networks and for threads. They also reveal that networks at $d+1$ are embedded in networks at depth d. We sum up all the properties of the protocol in a more descriptive form in Figure 6.1. Because most of the examples we have considered were negative, we show several 'healthy strategies' for which the network structure is outlined pictorially. Threads belonging to the same network are marked with the same kind of pattern.

Example 6.26. In the first case, we have one network at depth 1 and three networks at depth 2.

(p1) A ***network*** (at depth d) comprises an O-thread and finitely many P-threads (all at depth d).

(p2) ***Network and Thread Creation***: Only O can open a new network, and he does so whenever he starts a new O-thread; whenever P opens a new thread in response to an O-move from a network, a new P-thread is added to that network.

(p3) ***!-Network***: A network whose O-thread is of !-type (call the network a *!-network*) has at most one P-thread which must also be of !-type. If O-threads of two !-networks are related, then, if the networks have P-threads, the P-threads are also related.

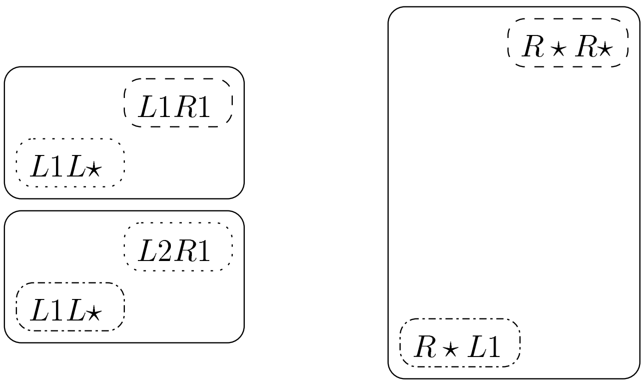
(p4) No thread can belong to more than one network.

(p5) ***Switching Condition***: Only O can switch network, i.e. revisit one opened earlier or enter the *threadless universe*[a]. Only P can switch from one thread to another existing thread within the same network.

[a]At depth d, the threadless universe of G consists of moves of depth $< d$.

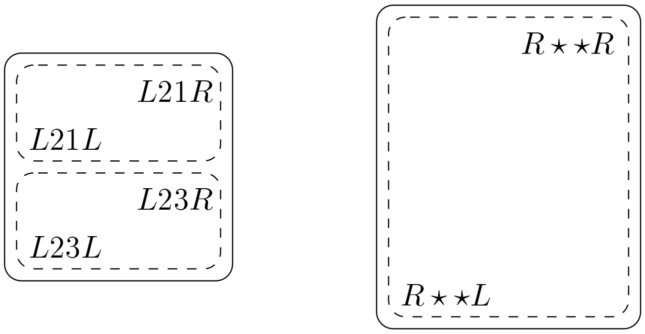
Figure 6.1: The Network Protocol

$$!\,(\ \S a \multimap\ !a\ ) \quad \multimap \quad \S\,(\ !a \multimap \S a\,)$$



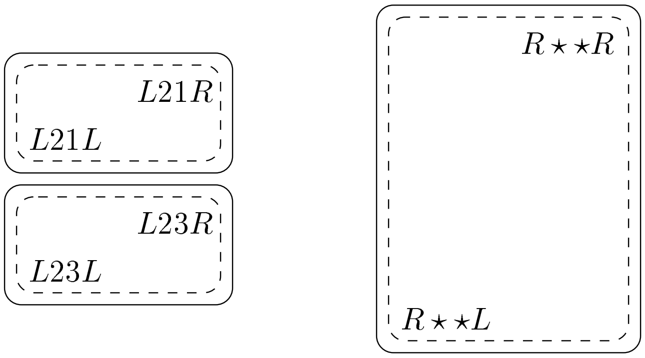
The next two strategies are for the same game, but correspond to different proofs.

$$!\,!\,(\; a \;\multimap\; a \;) \quad\multimap\quad \S\,\S\,(\; a \;\multimap\; a \;)$$



The difference is reflected by the network at depth 1: it consists of two threads for the first strategy and three threads for the other.

$$!\,!\,(\; a \;\multimap\; a \;) \quad\multimap\quad \S\,\S\,(\; a \;\multimap\; a \;)$$



Next we give a formal definition of strategies that comply with the network protocol and finally prove that indeed they are a model of IMLAL.

Let us denote the set of threads of a game G by T_G, which partitions into T_G^O (the set of O-threads) and T_G^P (the set of P-threads), we write $T_{G,i}$ for the set of i-threads. The formalization of the network protocol uses two (kinds of) functions.

Definition 6.27. A ***thread function*** (at depth i) is a partial function $t_{G,i} : M_G \rightharpoonup T_{G,i}$ that maps a move to the i-thread in which it occurs (i.e. to its i-index). Whenever the game G and depth i are clear from the context, we will abbreviate $t_{G,i}(m)$ to t_m. We say that a partial function

$$\eta_{G,i} : T_{G,i}^P \rightharpoonup T_{G,i}^O$$

networks a position $s \in P_G$ at depth i (with respect to $t_{G,i}$) just in case for each odd $j \leqslant |s|$, $t_{G,i}(s_j)$ is defined if and only if $t_{G,i}(s_{j+1})$ is defined, and if both are defined, one of the following holds (we drop the subscripts from $\eta_{G,i}$ and $t_{G,i}$ whenever we safely can):

$$\begin{array}{rrcl@{\qquad}rrcl} (i) & \eta(t(s_j)) & = & t(s_{j+1}) & (ii) & t(s_j) & = & \eta(t(s_{j+1})) \\ (iii) & \eta(t(s_j)) & = & \eta(t(s_{j+1})) & (iv) & t(s_j) & = & t(s_{j+1}) \end{array}$$

(This is just to say that at depth i the P-move s_{j+1} is in the same network as the O-move s_j: there are four cases depending on whether s_j and s_{j+1} are in P-threads or O-threads.)

Lemma 6.28. Let s be a position networked at depth i by $\mu_{G,i}$. Suppose there is an O-thread t^O and a P-thread t_P that form a network in s. If s contains an odd number of moves from t_P, then also an odd number of moves from t^O have been made in s.

Proof. By switching conventions. □

6.3.4 Networked strategies

Definition 6.29 (Networked strategies). We say that a strategy σ on G is *networked at depth* i if there exists a function $\eta_{G,i}$ that networks every $s \in \sigma$ at depth i. A strategy is ***networked*** if and only if it is networked at any depth.

Networked strategies compose

Consider the games A, B and C and suppose u is an interaction sequence of two positions $p_1 \in P_{A \multimap B}$ and $p_2 \in P_{B \multimap C}$ which are networked by $\eta_{A \multimap B,i}$ and $\eta_{B \multimap C,i}$ respectively at a given depth i. We would like to show that $u \upharpoonright (A, C)$ can be networked and that all positions of the composite strategy are networked by the same network function $\eta_{A \multimap C,i}$.

Let us fix a network function η_G. We shall write $t' \prec_G t$ to mean $\eta_G(t) = t'$, so for each P-thread t there can be at most one O-thread t' such that $t' \prec t$. Our argument below is with respect to a fixed depth i. For ease of writing, we shall drop all references to the depth and omit the subscript i from $\eta_{A \multimap C,i}$, $t_{A \multimap C,i}$ etc.

We will say that threads $t_1, t_2, \cdots, t_k$ from an ***interaction sequence of threads*** if

$$t_1 \prec_1 t_2 \prec_2 t_3 \prec_3 \cdots \prec_{k-1} t_{k-1} \prec_k t_k$$

where $\prec_i \in \{\prec_{A \multimap B}, \prec_{B \multimap C}\}$ $(1 \leqslant i \leqslant k)$ and $\prec_i \neq \prec_{i+1}$ $(1 \leqslant i < k)$.

Let us define another network function $\eta_{A \multimap C}$: for any $t^O \in T^O_{A \multimap C}$ and $t^P \in T^P_{A \multimap C}$, we decree that $\eta_{A \multimap C}(t^P) = t^O$ (i.e. $t^O \prec_{A \multimap C} t^P$) just in case there exists an interaction sequence of threads between t^O and t^P i.e. there exist $b_1, \cdots, b_k \in T_B$ such that

$$t^O \prec_1 b_1 \prec_2 b_2 \prec_3 \cdots \prec_{k-1} b_{k-1} \prec_k b_k \prec_{k+1} t^P \tag{6.1}$$

subject to the above-mentioned conditions. (Each b_i in (6.1) is actually the embedding image of the B-thread in $T_{A\multimap B}$ or $T_{B\multimap C}$ as appropriate, but we stick to b_i by abuse of notation.) Note that if t^O and t^P are both A-threads (respectively C-threads) and $t^O \prec_{A\multimap B} t^P$ (respectively $t^O \prec_{B\multimap C} t^P$), then $t^O \prec_{A\multimap C} t^P$ which is the case of $k = 0$. Because $\eta_{A\multimap B}$ and $\eta_{B\multimap C}$ are partial functions, we first note that $\eta_{A\multimap C}$ is a partial function from $T^P_{A\multimap C}$ to $T^O_{A\multimap C}$.

Proposition 6.30. Consider any contiguous segment $m_O m_1 \cdots m_k m_P$ of u such that $m_O \in M^O_{A\multimap C}$, $m_P \in M^P_{A\multimap C}$, and $m_i \in M_B$ for $i = 1, \cdots, k$. If m_O is part of a thread then m_O and m_P are in the same network with respect to $\eta_{A\multimap C}$, i.e. writing t^O as the O-thread of m_O's network, we have either $t^O = t_{m_P}$ or $t^O \prec_{A\multimap C} t_{m_P}$.

Proof. We induct on the length of $u' = u \upharpoonright (A, C)$. Let t^O be the O-thread of the network to which m_O belongs. If t_{m_O} is an O-thread, then $t^O = t_{m_O}$. Otherwise t_{m_O} is a P-thread and m_O cannot be the first move made in it. From the induction hypothesis for the initial P-move (in u') in that thread we get $t \prec_{A\multimap C} t_{m_O}$ for some $t \in T^O_{A\multimap C}$ i.e. there exist $b_1, \cdots, b_l$ such that $t \prec' b_1 \prec' b_2 \prec' \cdots \prec' b_l \prec' t_{m_O}$. Thus $t^O = t$. (We write $\prec'$ for either $\prec_{A\multimap B}$ or $\prec_{B\multimap C}$. It should be clear from the context which is meant.)

Let us introduce an auxiliary relation $\prec_B \subseteq T^O_{A\multimap C} \times T_B$. We decree $d \prec_B b$ just in case there exists an interaction sequence of threads from starting with t and ending in b. $\prec_B$ is also a partial function from T_B to $T^O_{A\multimap C}$. Then we have:

Lemma 6.31. Writing t_{m_i} simply as t_i, for each i, $t^O \prec_B t_i$.

Proof. We prove the Lemma by induction on i. Consider m_1. It must belong to the same network as m_O in one of the two components ($A \multimap B$ or $B \multimap C$) so:

- If t_{m_O} is an O-thread, then $t_{m_O} \prec' t_1$.
- If t_{m_O} is a P-thread then:
  - If t_1 is an O-thread then $t_1 = b_l$,
  - If t_1 is a P-thread then $b_l \prec' t_1$.

In all cases $t^O \prec_B t_1$ holds. If $t_{i+1} = t_i$, the induction hypothesis says what is required, so suppose $t_{i+1} \neq t_i$. Recall that the induction hypothesis says there exist $b_1, \cdots, b_k$ such that

$$t^O \prec_1 b_1 \prec_2 b_2 \prec_3 \cdots \prec_k b_k \prec_{k+1} t_i.$$

m_{i+1} is a P-move in one of the components ($A \multimap B$ or $B \multimap C$), so it is in the same network as m_i inside that component. We argue by case analysis.

- If t_i is an O-thread (in the above-mentioned component), t_{i+1} must be a P-thread of the same network, so $t_i \prec' t_{i+1}$. By the induction hypothesis $t^O \prec_B t_i$, so $t^O \prec_B t_{i+1}$ holds as well.
- If t_i is a P-thread then:
  - If t_{i+1} is an O-thread, we have $t_{i+1} \prec' t_i$. But then $b_k = t_{i+1}$ (because $\prec_{k+1}$ is a partial function), so $t^O \prec_B t_{i+1}$ as required.
  - If t_{i+1} is a P-thread, we have $b_k \prec' t_{i+1}$, so $t^O \prec t_{i+1}$.

□

Now consider the final move m_P. It is in the same network as m_k (in one of the two components). Thus:

- If t_k is an O-thread then t_{m_P} must be a P-thread and $t_k \prec' t_{m_P}$ must hold. By Lemma 6.31, we have $t^O \prec_B t_k$, so $t^O \prec_{A \multimap C} t_{m_P}$ as required.
- If t_k is a P-thread then:
  - Suppose t_{m_P} is an O-thread. We have $t_{m_P} \prec' t_k$; also by Lemma 6.31, we have $t^O \prec_B t_k$. Therefore $t_{m_P} = t^O$ (because t_{m_P} is an A- or C-thread and the t_i's are B-threads).
  - Suppose t_{m_P} is a P-thread. By Lemma 6.31, there exist $b_1, \cdots, b_k$ such that
  $$t^O \prec_1 b_1 \prec_2 \cdots \prec_k b_k \prec_{k+1} t_k.$$
  Then either $b_k \prec_{k+1} t_{m_P}$ or (if $k = 0$) $t^O \prec_1 t_{m_P}$.

Hence $t^O \prec_{A \multimap C} t_{m_P}$ in all cases. □

Thus if strategies σ and τ are networked at depth i by $\eta_{A \multimap B}$ and $\eta_{B \multimap C}$ respectively, $\sigma\,;\tau$ is networked by $\eta_{A \multimap C}$ at depth i. As the argument is independent of i we have just proved that networked strategies compose.

Our next step is a technical result which identifies a bounding factor in the interaction between threads.

Lemma 6.32. Suppose s is an interaction sequence of two strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$. The length of any interaction sequence of threads of s at depth i is smaller than the number of i-bases (of moves) in s.

Proof. For a contradiction, suppose there exists an interaction sequence longer than the number of i-bases in s. Hence, there exists an interaction sequence between two threads with the same base, say t_1 and t_2. Consider the moment when t_2 was opened in s. By Lemma 6.28, the number of moves in all threads from the interaction sequence of threads between t_1 and t_2 must have been odd. Therefore the number of moves in t_1 was also odd. But t_1 and t_2 have the same base, so they come from the same !-subgame and this state would violate the switching condition for !. □

It is easy to see that for any O-thread t, the set consisting of threads t' such that there exists an interaction sequence of threads between t and t' has a tree structure. We will be interested only in the branches of the tree that actually take part in the interaction between the strategies (i.e. moves from the threads occur in some interaction sequences). We call this tree ***the interactive tree of*** t. The last lemma provides a way of estimating the depth of the tree.

Nesting of networks

It follows from our definition of networks that networks at depth $d+1$ are embedded in networks at depth d: if threads t_1 and t_2 belong to the same network at $d+1$ and t_1', t_2' are the d-threads to which they belong respectively, then t_1' and t_2' are from the same network at depth d. This dependency manifests itself during composition. Observe that for any j-thread t occurring in an interaction sequence, there exists a unique O-thread $\widehat{t}$ (also at depth i) in A or C such that t belongs to the interactive tree of $\widehat{t}$. Now suppose that t_{d+1} is a thread at depth $d+1$. Let i_1 and i_2 be the d-indices of t_{d+1} and $\widehat{t}_{d+1}$ respectively. Then at depth d we have $\widehat{i_1} = \widehat{i_2}$ i.e. interaction sequences of threads at $d+1$ are embedded into interaction sequences of threads at d.

Compactly networked strategies

Definition 6.33. We say that a networked strategy (by η) is ***compactly networked*** if $\eta^{-1}(\{\, t^O \,\})$ is finite for every O-thread t^O.

Lemma 6.34. Suppose σ, τ are compactly networked strategies and t^O is an O-thread, which is in A or C. The interactive tree of t^O is finite.

Proof. We use induction on the depth of t^O. Suppose t^O is a thread at depth 1. By Lemma 6.32 the depth of t^O's interactive tree is bounded (since the shape of the game is). As σ and τ are compactly networked, by König's lemma, t^O's interactive tree must be finite.

Suppose t^O is at depth $i+1$. Let t_i be the i-index of t^O. By induction hypothesis, $\widehat{t_i}$'s interactive tree is finite. From the remarks about nesting we see that each thread from t^O's interactive tree has an i-index that belongs to $\widehat{t_i}$'s interactive tree. Hence, because we play an IMLAL game, the $i+1$-bases that could be used by threads in t_O's interactive tree form a finite set. Now we repeat the same reasoning as in the base case: by Lemma 6.32 the depth of the interactive tree of t^O must have bounded depth and, because it is finitely branching, it must be finite as required. $\square$

An immediate consequence is:

Proposition 6.35. Compactly networked strategies compose.

and

Proposition 6.36. Total compactly networked strategies compose.

Proof. Suppose *infinite chattering* occurs i.e. there exists an infinite interaction sequence with a finite set of moves in A and C. Let d be the depth of the moves involved. By Proposition 6.34 the number of threads (at depths $1, 2, \cdots, d$) inside the infinite sequence can only be finite. So is the length of moves, because it is bounded by the shape of the IMLAL game we play. Therefore, the sequence cannot be infinite, because no move may be repeated. $\square$

Suitably networked strategies

Definition 6.37. A networked strategy is ***suitably networked*** when for any O-thread t^O, if t^O is of !-type and $\eta(t_1^P) = \eta(t_2^P) = t^O$, then $t_1^P = t_2^P$ and t_1^P is of !-type.

The condition formalizes the first part of (p3) from Figure 6.1.

Proposition 6.38. Suitably networked strategies compose.

Proof. If σ and τ satisfy (p3), so does $\sigma\,;\tau$, because if

$$t^O \prec_1 b_1 \prec_2 b_2 \prec_3 \cdots \prec_{k-1} b_{k-1} \prec_k b_k \prec_{k+1} t^P$$

and t^P is a !-thread, then by (p3) applied alternately to σ and τ we deduce that $b_1, \cdots, b_k$ and t^O are unique and are all !-threads. $\square$

6.3.5 Equivalence of positions

When introducing shriek games we emphasized that their definition is intended to provide a mechanism for distinguishing between different plays of G inside $!G$. Numeric labelling has been used for that purpose, but on reflection we see that this introduces too much distinction. All we wanted was an unambiguous decomposition of plays in $!G$, whereas by adding labels we have differentiated between plays with the same structure which should be deemed equivalent. To make up for this overly concrete representation, we define a notion of equivalence between positions $\approx_G \subseteq P_G \times P_G$. For $m \in M_G$ we write $core(m)$ for m in which each numeric tag has been replaced with $\Box$, so e.g. $core(R(56)l3L) = R\Box l\Box L$.

Definition 6.39. Given an IMAL game G, $s, t \in P_G$, $s \approx_G t$ is defined to hold if and only if:

- $core^*(s) = core^*(t)$ (in particular $|s| = |t|$)
- if $s_i = ws'_i$ and $s_j = ws'_j$, then there exists v such that $|v| = |w|$, $t_i = vt'_i$ and $t_j = vt'_j$,
- if $t_i = vt'_i$ and $t_j = vt'_j$, then there exists w such that $|w| = |v|$, $s_i = ws'_i$ and $s_j = ws'_j$.

For instance, the two positions from the game $G = !((a \multimap b) \otimes (c \multimap d))$ mentioned in Example 6.22 are not equivalent, but

$$(3lR)\,(3lL)\,(17rR)\,(17rL) \approx_G (71lR)\,(71lL)\,(16rR)\,(16rL).$$

The equivalence coincides with the following inductively defined relation which extends the one from [5]:

- for single-move games G_a, we have $\epsilon \approx_{G_a} \epsilon$ and 'ϵ' $\approx_{G_a}$ 'ϵ',
- $s \approx_{A\otimes B} t$ just in case $s \upharpoonright l \approx_A t \upharpoonright l$, $s \upharpoonright r \approx_B t \upharpoonright r$ and $hd^*(s) = hd^*(t)$ (i.e. s_i and t_i begin with the same letter for any i),
- $s \approx_{A\multimap B} t$ just in case $s \upharpoonright L \approx_A t \upharpoonright L$, $s \upharpoonright R \approx_B t \upharpoonright R$ and $hd^*(s) = hd^*(t)$.
- $s \approx_{\S A} t$ if $s \upharpoonright \star \approx_A t \upharpoonright \star$,
- and, most importantly, $s \approx_{!A} t$ if

  $$\exists \alpha \in S(\mathbb{N}).\forall i \in \mathbb{N}.s \upharpoonright i \approx_A t \upharpoonright \alpha(i) \ \wedge\ \alpha^*(hd^*(s)) = hd^*(t),$$

  where $S(\mathbb{N})$ is the collection of permutations of $\mathbb{N}$.

Given $f : X \rightharpoonup Y$, $f^* : X^* \rightharpoonup Y^*$ is the pointwise application of f to words over alphabet X producing words over Y. $hd : X^* \rightharpoonup X$ returns the first letter of a non-empty string over alphabet X. $s \upharpoonright c$ where $c \in \{ l, r, L, R \} \cup \mathbb{N}$ is the subsequence of s consisting of moves that begin with the 'letter' c.

The equivalence relation $\approx_G$ on positions extends to a *partial* equivalence relation over strategies for G [5]. Two strategies are deemed ***equivalent*** if they can simulate each other up to the equivalence of positions: $\sigma \approx_G \tau$ holds just in case for all $s \in \sigma$, $t \in \tau$, $sa \in P_G$, $tc \in P_G$, if $sa \approx_G tc$ then the following bisimulation-style properties hold:

- $sab \in \sigma \;\Rightarrow\; \exists d \in M_G. tcd \in \tau \wedge sab \approx_G tcd,$
- $tcd \in \tau \;\Rightarrow\; \exists b \in M_G. sab \in \sigma \wedge sab \approx_G tcd.$

Example 6.40. The **(!)** rule indicates that in a fully and faithfully complete game model there should be a bijective correspondence between strategies for G and $!G$. It is easy to convert a strategy $\sigma : G$ into one for $!G$ (call it $!\sigma$). After all, plays of $!G$ are just interleavings of plays of G. Therefore, it suffices to allow $!\sigma$ to 'imitate' σ in all opened instances of G. Then we have $!\sigma \approx !\sigma$ because $!\sigma$ mimics σ in each instance of G opened in $!G$. However, the retrieval of a unique strategy for G from one for $!G$ is not possible yet. Consider the game $!(a \otimes a \multimap a)$ and the strategy defined by the function

$$f \;:\; \begin{cases} ((2n)R) & \mapsto & ((2n)Ll) \\ ((2n+1)R) & \mapsto & ((2n+1)Lr) \end{cases}$$

This strategy mimics two different strategies for $a \otimes a \multimap a$ depending on the number of the instance which is opened by O, and each of them can legitimately claim to be the associated strategy for $a \otimes a \multimap a$. To eliminate this undesirable ambiguity, one requires strategies to be $\approx$-***reflexive*** (i.e. $\sigma \approx \sigma$ should hold) [5]. The intuition is that $\approx$-reflexive strategies 'behave equivalently at equivalent positions'. σ_f is not $\approx$-reflexive: its response at even-numbered instances is different from that at equivalent but odd-numbered ones): we have $((2n)R) \approx ((2n+1)R)$, but $((2n)R)((2n)Ll) \not\approx ((2n+1)R)((2n+1)Lr)$.

Although $\approx$-reflexivity helps us to force an exact correspondence between strategies in G and $!G$, it is still too weak to be used in our context. Because networks are to model applications of the **(§)** and **(!)** rules, equivalence should be based on the preservation of the network structure rather than mere equivalence of positions. We illustrate the subtle difference with an example.

Example 6.41. Consider the two strategies $\sigma_{f_1}, \sigma_{f_2} :!(a \otimes a) \multimap \S(a \otimes a)$ where

$$\begin{array}{ll} f_1(R \star l) = L1l & f_2(R \star l) = L2l \\ f_1(R \star r) = L1r & f_2(R \star r) = L3r \end{array}$$

We have $\sigma_{f_1} \approx \sigma_{f_2}$, but that is not what we would like to see. The network in σ_{f_1} has two threads, whereas σ_{f_2} has three. In the latter case the P-threads come from the same occurrence of ! which indicates the use of contraction. In contrast, the former strategy imitates the simple proof in which after $a \otimes a \vdash a \otimes a$ is derived, one applies the $(\S)$ rule.

This problem has yet another guise.

Example 6.42. $\approx$-reflexive strategies need not satisfy the condition concerning related !-threads, while one might expect to impose it through an appropriate notion of equivalence between positions. Consider the strategy $\sigma_{f_3} :!!a \multimap \S!a$, such that $f_3(R \star 2) = Lmn$ and $f_3(R \star 1) = Lm'n'$ for $m' \neq m$. Plainly, $\sigma_{f_3} \approx \sigma_{f_3}$.

The shortcomings of the $\approx$ relation are due to the fact that the network structure is determined globally and sometimes the players will not be able to play out all the threads of a network in a single position. For that reason, it will be generally impossible to test copies of !-networks in the same play. If the $\approx$ relation is to be of use to capture the second part of (p3), we need to allow O to reveal any interaction between threads. We allow that by introducing new moves into the game. They will enable players to backtrack in the "game tree" so that they can explore new positions, even though in the current framework the play would have to end because of lack of available moves.

Technically, each token a will be modelled by a two-move game $a \multimap a$. Given an IMLAL game G, we write $\widehat{G}$ for the IMLAL game that arises by replacing each token a of G with $a \multimap a$ (cf. Proposition 6.21).

Definition 6.43. Let $\sigma_f : G$ be an injective history-free total strategy. $\widehat{\sigma}_f : \widehat{G}$ is the strategy $\sigma_{\widehat{f}}$ generated by $\widehat{f}$ such that

$$\widehat{f}(xR) = f(x)R \quad \text{and} \quad \widehat{f}(f(x)L) = xL.$$

By Proposition 6.21 $\widehat{\sigma_f}$ is total. Obviously, it is also injective history-free. Moreover, if σ_f is networked, so is $\widehat{\sigma}_f$ and they are networked by the same network function.

The Proposition below shows that $\widehat{\sigma_f} : \widehat{G}$ is already able to explore any arbitrary finite amount of information provided by the underlying history-free function. In particular, when the strategy is compactly networked, players can open all threads of a network in a single position!

Proposition 6.44. Suppose $\sigma_f : G$ is total injective history-free. Then for each finite subset f' of f, there exists a position $s \in \widehat{\sigma}_f$ such that for all $x \in dom\, f'$, s contains xR and $f(x)R$.

Proof. Let us construct a directed graph whose vertices are pairs of moves $\tilde{x} = xf(x)$ for $x \in dom f$. There is an edge from $\tilde{x}$ to $\tilde{y}$ if $f(x)$ enables y (Definition 6.9). Additionally, there is a vertex $\odot$ which is connected to all $\tilde{x}$ such that x is an initial move of $\widehat{G}$ (which are essentially the initial moves of G).

By Lemma 6.11 the graph is acyclic and by Lemma 6.13 each vertex is reachable from $\odot$. Besides, any finite path from $\odot$ to another vertex corresponds to some short-sighted position by totality of the strategy and the fact that short-sighted moves by O always extend a short-sighted position.

Clearly, all vertices $\tilde{x}$ for $x \in dom\, f'$ can be visited by a depth-first-like sweep of the graph in which one is not required visit all children of the interim vertex. Besides, this can be done in a finite number of steps. The procedure (with backtracking) can be emulated with the additional moves of $\widehat{G}$ (used to 'retreat' from a node that has been visited but whose children do not have to be) so that the traversal corresponds to a position in $\widehat{\sigma_f}$. □

Remark 6.45. The technical idea of considering two-move games instead of singleton games to allow players to make more moves underpins the fully complete fair games model of MLL without MIX [63]. In contrast to the first fully complete model of MLL+MIX [3], players in fair games are required to reach any move of the current game in a single play. It was shown that strategies which fail to take up this chance arise from proofs using MIX.

Because $\approx$ turned out unsatisfactory, we propose a new relation $\approx_n$ based on the expansion just considered.

Definition 6.46. For two injective history-free and total strategies σ, τ:

$$\sigma \approx_n \tau \iff \widehat{\sigma} \approx \widehat{\tau}.$$

σ_f will be called ***consistent*** if and only if σ_f is $\approx_n$-reflexive.

$\approx_n$ is a partial equivalence relation, because $\approx$ was. Besides, $\sigma \approx_n \tau$ implies $\sigma \approx \tau$.

Let us revisit the previous examples. The following position is now to be found in $\widehat{\sigma}_{f_1}$ (Example 6.41):

$$s_1 = (R \star lR)\,(L1lR)\,(L1lL)\,(R \star lL)\,(R \star rR)\,(L1rR).$$

$\widehat{\sigma}_{f_2}$ contains

$$s_2 = (R \star lR)\,(L2lR)\,(L2lL)\,(R \star lL)\,(R \star rR)\,(L3rR).$$

We have $s_1 \not\approx s_2$, but all their prefixes are $\approx$-equivalent. Hence $\widehat{\sigma}_{f_1} \not\approx \widehat{\sigma}_{f_2}$, i.e. $\sigma_{f_1} \not\approx_n \sigma_{f_2}$.

What about the strategy σ_{f_3} (Example 6.42)? Is it consistent? To our satisfaction, the answer is no, but we need to assume additionally that σ_{f_3} is compactly networked. This ensures that the network of $R\star$ consists of a finite number of threads with indices $i_1, \cdots, i_k$ for $k > 1$. Therefore, if $\widehat{\sigma_{f_3}}$ is $\approx$-reflexive, threads with indices of the form $R \star n$ must form networks with threads indexed by Li_jn $(1 \leqslant j \leqslant k)$. Thus there must exist threads $R \star m_1$ and $R \star m_2$ $(m_1 \neq m_2)$ whose companion threads are i_ln_1 and i_ln_2 for some l $(1 \leqslant l \leqslant k)$. Then we have the two following positions in $\widehat{\sigma}_{f_3}$:

$$s_1 = (R \star 1R)\,(LmnR)\,(LmnL)\,(R \star 1L)\,(R \star 2R)\,(Lm'n'R)$$

$$s_2 = (R \star m_1R)\,(Li_ln_1R)\,(Li_ln_1L)\,(R \star m_1L)\,(R \star m_2R)\,(Li_ln_2R)$$

Clearly, $s_1 \not\approx s_2$, but all prefixes of the two positions of the same length are $\approx$-equivalent. Consequently, $\sigma_{f_3} \not\approx_n \sigma_{f_3}$, i.e. σ_{f_3} is inconsistent. Analogously, one can prove the following result:

Proposition 6.47. If σ_f is a total injective history-free compactly networked strategy which is *consistent*, all related !-threads in the associated networks have related counterparts (if any).

Consistency makes it possible to represent total compactly-networked strategies by finite means. Because their network structure is identical up to renumbering of !-threads, we can extract full information about networks by opening O-threads with indices ending in 0 or $\star$ only.

Definition 6.48. Given a strategy $\sigma : G$, a 'substrategy' $\sigma_0 \subseteq P_G$ by induction as follows:

- if $s \in \sigma_0$, $|s|$ is even, $sa \in \sigma$ and all O-indices in a are 0 or $\star$, then $sa \in \sigma_0$;
- if $s \in \sigma_0$, $|s|$ is odd, $sa \in \sigma$,then $sa \in \sigma_0$.

Note that $\sigma_0 \subseteq \sigma$, but σ_0 is not a strategy in the standard sense.

Proposition 6.49. If σ is compactly networked, σ_0 is finite. $\square$

Remark 6.50. If σ is an injective history-free compactly networked strategy which is total and consistent, σ_0 can be regarded as its finite representation in that σ_0 uniquely defines a $\approx_n$-equivalence class of strategies with the same properties.

Finally, we would like to construct a compositional framework incorporating $\approx_n$-equivalence. For that we need a class of strategies that compose. $\widehat{\sigma}$ was defined for total strategies which do not compose in general. Total compactly networked do. For them once can easily prove:

Lemma 6.51. 1. $\widehat{\sigma};\widehat{\tau} = \widehat{\sigma;\tau}$.

2. If $\sigma_1 \approx_n \sigma_2$ and $\tau_1 \approx_n \tau_2$, then $\sigma_1;\tau_1 \approx_n \sigma_2;\tau_2$.

□

The Lemma guarantees that one can indeed build a quotient category using consistent strategies and $\approx_n$.

6.3.6 The model

Definition 6.52. $\mathcal{G}_n$ is the category of IMLAL games whose morphisms between A and B are $\approx_n$-equivalence classes of token-reflecting injective history-free total strategies for $A \multimap B$ which are

- suitably and compactly networked,
- and consistent.

The definition is correct: $\mathcal{G}_n$ is a category by Proposition 6.35, Proposition 6.38 and Lemma 6.51. For brevity, we shall call strategies with all these properties ***winning (IMLAL) strategies***. If σ is a winning strategy, σ_0 is referred to as ***0-winning***. Next we show that $\mathcal{G}_n$ satisfies all the category-theoretic requirements for a model of IMLAL i.e. it is a light affine category (Definition 5.7).

Theorem 6.53. $\mathcal{G}_n$ is a model of IMLAL.

Proof. By the same reasoning as in the proof of Theorem 6.8 we can prove that $\mathcal{G}_n$ is an autonomous category. Recall that the empty game $\varnothing$ is the tensor unit.

In order to define the two endofunctors $!, \S : \mathcal{G}_n \longrightarrow \mathcal{G}_n$ we begin with some operations on strategies. Given a strategy $\sigma : G_1 \multimap G_2$, let $\S\sigma : \S G_1 \multimap \S G_2$ be the strategy

$$\S\sigma = \{\, (\underline{m}_1)\cdots(\underline{m}_k) \quad | \quad m_1 \cdots m_k \in \sigma \,\}$$

where $\underline{Rm} = R\star m$ and $\underline{Lm} = L\star m$. It is easy to see that if $\sigma = \sigma_f$, $\S\sigma$ is generated by the following g:

$$g(d_1 \star m) = d_2 \star m' \quad \text{iff} \quad f(d_1 m) = d_2 m'$$

for $d_1, d_2 \in \{ L, R \}$. Our definition of a similar operation for ! must be necessarily more complicated as σ may be called to act several times:

$$!\sigma = \{ s \in P_{!G_1 \multimap !G_2} \quad | \quad \forall_{n\in\mathbb{N}}\, s \upharpoonright n \in \sigma \}$$

where $s \upharpoonright n$ is the subsequence of s consisting of moves having the shape dnm for $d \in \{ L, R \}$. I.e. $!\sigma$ imitates σ in identically numbered threads of G_1 and G_2 and it is only O (in the subgame $!G_2$) who can change the number and the network. If $\sigma = \sigma_f$, we have $!\sigma = \sigma_g$, where:

$$g(d_1 nm) = d_2 nm' \quad \text{iff} \quad f(d_1 m) = d_2 m'$$

for all $d_1, d_2 \in \{ L, R \}$. It is straightforward to check that if $\sigma \approx_n \tau$, both $!\sigma \approx_n !\tau$ and $\S\sigma \approx_n \S\tau$ hold. Therefore, the following definition is correct.

$$\begin{array}{rcl@{\qquad}rcl} !(G) & = & !G & \S(G) & = & \S G \\ !([\sigma]_{\approx_n}) & = & [!\sigma]_{\approx_n} & \S([\sigma]_{\approx_n}) & = & [\S\sigma]_{\approx_n} \end{array}$$

Because $\S(\sigma;\tau) = \S\sigma;\S\tau$, $!(\sigma;\tau) = !\sigma;!\tau$, $\S(\mathsf{id}_G) = \mathsf{id}_{\S G}$ and $!(\mathsf{id}_G) = \mathsf{id}_{!G}$, it is obvious that ! and § are functors. Besides, both § and ! leave the empty game unchanged, so $\S I$ and $!I$ will be interpreted as terminal objects. Hence, there are unique maps (empty strategies)

$$\mathsf{m}_I : I \longrightarrow \S I, \qquad \mathsf{s}_I : I \longrightarrow !I, \qquad \mathsf{z}_I : !I \longrightarrow \S I.$$

The definition of w is also trivial and all conditions regarding the special maps for units (namely $\mathsf{w}_{!I}; \mathsf{s}_I = !(\mathsf{w}_I)$, $\mathsf{w}_{\S I}; \mathsf{m}_I = \S(\mathsf{w}_I)$, $\mathsf{s}_I; \mathsf{z}_I = \mathsf{m}_I$) are satisfied.

The canonical maps

$$\begin{array}{rl} \mathsf{m}_{G_1,G_2} : & \S G_1 \otimes \S G_2 \longrightarrow \S(G_1 \otimes G_2) \\ \mathsf{z}_G : & !G \longrightarrow \S G \\ \mathsf{e}_G : & !G \longrightarrow !G \otimes !G \end{array}$$

are given by $\approx_n$-equivalence classes of the following strategies, which we denote with the same symbols.

- $\mathsf{m}_{G_1,G_2} : \S G_1 \otimes \S G_2 \multimap \S(G_1 \otimes G_2)$

$$\begin{array}{rcl@{\qquad}l} R\star lm & \mapsto & Ll\star m & m \in M^O_{G_1} \\ Ll\star m & \mapsto & R\star lm & m \in M^P_{G_1} \\ R\star rm & \mapsto & Lr\star m & m \in M^O_{G_2} \\ Lr\star m & \mapsto & R\star rm & m \in M^P_{G_2} \end{array}$$

- $\mathsf{z}_G :!G \multimap \S G$

$$\begin{array}{rcll} R\star m & \mapsto & L0m & m \in M_G^O \\ L0m & \mapsto & R\star m & m \in M_G^P \end{array}$$

- $\mathsf{e}_G :!G \multimap !G \otimes !G$

$$\begin{array}{rcll} Rlnm & \mapsto & L(2n)m & m \in M_G^O \\ Rrnm & \mapsto & L(2n+1)m & m \in M_G^O \\ Lnm & \mapsto & R\left(\begin{array}{rcl} \text{even } n & \mapsto & l \\ \text{odd } n & \mapsto & r \end{array}\right)\lfloor\frac{n}{2}\rfloor m & m \in M_G^P \end{array}$$

m consists of 'copycat' strategies, so it is natural and makes all diagrams of a symmetric monoidal functor commute. Hence, § is symmetric monoidal. Similarly, because of its schematic nature, z and e are natural transformations, and e defines a commutative commonoid as specified in Chapter 5. □

It is straightforward to see that the interpretation of proofs induced by the categorical structure identifies axiom links with history-free functions and the function δ with network functions.

6.3.7 Full Completeness

Let $\sigma = \sigma_f$ be a strategy satisfying all properties of Definition 6.52. Consider its 0-winning subset σ_0 (which by abuse of notation we also write as σ_{f_0} where f_0 is a finite subset of f). σ_{f_0} is finite. We are going to zero in on σ^{f_0}, the subset of σ_{f_0} in which O must play short-sightedly (using the terminology from Definition 6.12 σ^{f_0} could be called weakly 0-winning). By Lemma 6.13 it is right to use the annotation f_0, because P will have to use each element of f_0 at least once. Obviously, σ^{f_0} is then also finite.

Using σ^{f_0} we construct a tree $\mathcal{T}_{f_0}$ in d stages where d is the depth of G. We start with $\mathcal{T}_{f_0}^0$ defined to be the syntactic tree of $\ulcorner G \urcorner$ in which branches are oriented according to the directional rules defining the essential nets for IMLAL. It is convenient to think of the branches as labelled with elements of $\{\, l, r, L, R, \star \,\}$ and natural numbers to make the connection between branches and moves clear (cf. Figure 6.3). We will also feel free to use expressions like 'the i-index of a node'.

For $i = 0, \cdots, d-1$ we generate $\mathcal{T}_{f_0}^{i+1}$ from $\mathcal{T}_{f_0}^i$ as follows (Example 6.58 demonstrates the whole procedure on a concrete example).

- Consider any occurrence of $?^-$ at depth $i+1$ i.e. one that is a hereditary premise of exactly i other ! or §-nodes. Let i be its i-base (determined by the labels assigned in the previous steps; during construction of $\mathcal{T}_{f_0}^{i+1}$ they will be assigned at level $i+1$).

- Suppose there are n threads with base i that occur in σ^{f_0} (i.e. there is some play in σ^{f_0} in which such a thread is opened). If $n < 2$ we take no action. Otherwise, we replace the occurrence of $?^-$ in $\mathcal{T}^i_{f_0}$ with a C^--node and attach n copies of the $\mathcal{T}^i_{f_0}$-subtree rooted at that occurrence of $!^-$ under the contraction node. We label the respective edges between the C^--node and the $!^-$-nodes with the numbers that extend i to the n indices of threads at depth $i+1$.

At the end we set $\mathcal{T}_{f_0} = \mathcal{T}^d_{f_0}$. Subsequently, as for IMAL, we convert $\mathcal{T}_{f_0}$ to a canonical affine net $\mathcal{N}_{f_0}$ in two steps:

(1) we connect all atomic nodes o^+ and p^- for which $f_0(o) = p$ with an axiom link (if $f_0(o) = p$, we are guaranteed to find the corresponding nodes in $\mathcal{T}_{f_0}$ as we have grown branches for all threads that are used by the strategy),

(2) subtrees rooted at nodes which are not reachable from the root, but which are premises of links that are reachable are all contracted to weakening nodes (we can argue that the node must be negative as for IMAL, so it makes sense to introduce a weakening node; the nodes will not be premises of contraction links, because we have not introduced them for unused threads during the construction of $\mathcal{T}_{f_0}$).

(1) enables us to interpret all positions from σ^{f_0} as unique paths from the root to the atomic node representing the final move (in the same way as for IMAL). When the weakening nodes are introduced, no axiom links will be broken either. Hence, $\mathcal{N}_{f_0}$ is a canonical affine net. Moreover, each $?^-$- or $\S^-$-node in it corresponds to a thread that is opened in σ^{f_0}.

The correspondence between paths and positions has a nice feature: the dangling (unclosed) brackets (i.e. $!^+, \S^+, ?^-, \S^-$-nodes) in paths from the root turn out to indicate the current network of play.

Lemma 6.54. Suppose $sm \in \sigma^{f_0}$ and m has depth d. Then there are exactly d unclosed (hence positive) brackets in the corresponding path in $\mathcal{N}_{f_0}$ and the ith open bracket ($1 \leqslant i \leqslant d$) has the i-index of the O-thread of the network to which m belongs at depth i. Besides, if a negative bracket closes a positive one, it represents a P-thread from the network whose O-thread is represented by the positive bracket.

Proof. We give a proof by induction on the length of $|sm| \in \sigma^{f_0}$.

If $s = \epsilon$, then m is an initial move. Thus it can be part of O-threads only and the opening brackets will indeed correspond to the O-threads that m initializes.

If m is a P-move, we apply the inductive hypothesis to s and because of (p5) (P must play in the networks of the preceding O-move), the Lemma is true for sm as well. In this case, the path corresponding to sm is that for s extended by crossing an axiom link.

If m is a non-initial O-move, then m is short-sighted and enabled by the preceding P-move p. Let

$$p^- q_1^- \cdots q_k^- \otimes^- r_1^+ \cdots r_l^+ m^+$$

be the unique path from p^- to m^+ in $\mathcal{N}_{f_0}$ (Theorem 6.16). Suppose the depth of p is d. By the induction hypothesis the path corresponding to s (ending in p) has d open brackets $b_1^+, \cdots, b_d^+$ representing the O-threads of networks to which p belongs at levels $1, 2, \cdots, d$ respectively. Let $q_{i_1}^-, \cdots, q_{i_v}^-$ and $r_{j_1}^+, r_{j_2}^+, \cdots, r_{j_w}^+$ be the subsequences of $q_1^- \cdots q_k^-$ and $r_1^+ \cdots r_l^+$ respectively, consisting of all negative and respectively positive brackets. Thus the bracket $q_{i_j}^-$ represents the P-thread to which p belongs at depth $d-j+1$. Note that it will be q_{i_j} that closes b_{d-j+1} and at the $\otimes^-$-node only $b_1, \cdots, b_{d-v}$ will be open. As m^+ does not belong to the threads represented by q_{i_j} this is desirable: by playing m and O leaves the P threads at depths $d-v+1, \cdots, d$ and also changes the networks at these depths. Therefore, the old membership at these depths should be revoked.

However, m will belong to the same threads as p at depths $1, \cdots, d-v$, so $b_1^-, \cdots, b_{d-v}^-$ should be left open as is the case. By playing m O may additionally open new O-threads at depths greater than $d-v$ and thereby start new networks at these depths. The potentially new O-threads are then (represented by) $r_{j_1}^+, \cdots, r_{j_w}^+$ (r_{j_k} at depth $d-v+k$) and they will be open in the path corresponding so sm as required. $\square$

Example 6.55. We consider a maximal position of a winning strategy for

$$!(\S a \multimap !a) \multimap \S(!a \multimap \S a)$$

that defines the essential net on the left in Figure 4.10. Below we show the position, the corresponding path and the unclosed brackets after each move. Note that $\S_1^+$ is open all the time, because all moves are part of the same network at depth 1. In contrast, $\S_2^+$ is closed when the third move is played, because the third move does not belong to the same network at depth 2 as the first two moves.

$(R\star R\star)$	$(L3R4)$	$(L3L\star)$	$(L8R9)$	$(L8L\star)$	$(R\star L7)$
$\parr^+ \S_1^+ \parr^+ \S_2^+ a^+$	a^-	$?^- \otimes^- \S_3^+ a^+$	a^-	$?^- \otimes^- \S_4^+ a^+$	a^-
$\S_1^+ \S_2^+$	$\S_1^+ \S_2^+$	$\S_1^+ \S_3^+$	$\S_1^+ \S_3^+$	$\S_1^+ \S_4^+$	$\S_1^+ \S_4^+$

Proposition 6.56. $\mathcal{N}_f$ is a correct IMLAL net.

Proof. Because $\mathcal{N}_f$ is canonical, it suffices to verify (i)-(v) from Definition 4.22 (see Theorem 4.49). For (i) and (ii) we reason like for IMAL. Thanks to the preceding

(iv) holds too, because δ was shown to correspond to the function that networks σ^{f_0}. Therefore, because of (p3), we have (v), so it remains to prove (iii).

Consider a path from the root that passes through a $\wp^+$-node p and ends in its sink s_p. Let a^+ be the first atomic node visited after the $\wp^+$-node and let b^- be the last node visited before reaching the sink (the polarities are implied by the structure of the net). Then the path is of the following form:

$$spq_1^+ \cdots q_v^+ a^+ \cdots b^- r_1^- \cdots r_w^- s_p$$

Suppose there are k opening (positive) brackets in s. Hence, a and b are in the same threads at depths $1, 2, \cdots, k$ because of the shape of the game (the underlying formula). By the preceding Lemma the k open brackets in s cannot be closed in

$$spq_1^+ \cdots q_v^+ a^+ \cdots b^-.$$

Let $b_{k+1}, \cdots, b_d$ be the remaining (if any) unclosed brackets in the above sequence (hence b's depth will be d and it will belong to some threads at depths $k+1, \cdots, d$). All these nodes must have corresponding closing nodes in $r_1^- \cdots r_w^-$ then, because the $\wp^+$-node is the premise of exactly k bracketing nodes. Therefore, the brackets open in

$$spq_1^+ \cdots q_v^+ a^+ \cdots b^- r_1^- \cdots r_w^- s_p$$

are exactly those that were open in s. Hence:

$$pq_1^+ \cdots q_v^+ a^+ \cdots b^- r_1^- \cdots r_w^- s_p$$

must be well-bracketed. □

We have just proved that each strategy σ_f determines a correct canonical affine net $\mathcal{N}_{f_0}$. In conjunction with Theorem 6.53 (and the remark following it) this implies

Theorem 6.57. $\mathcal{G}_n$ is a fully and faithfully complete model of IMLAL with respect to canonical affine nets.

By remarks following Proposition 6.21 and the definition of consistent strategies $\mathcal{G}_n$ could be presented equivalently as consisting of $\approx$-reflexive compactly networked history-free total strategies that are expandable (in the sense of Proposition 6.21) to strategies with the same properties when arbitrary IMLAL games are substituted for tokens.

We end this section by showing an example of a net derived from a winning IMLAL strategy.

Example 6.58. Consider the winning IMLAL strategy for

$$\S\S a\otimes!(\S a \multimap a\otimes!a) \multimap \S a\otimes!(!a \multimap \S a\otimes!a)$$

defined (among other mappings) by:

$$\begin{array}{rcl@{\qquad\qquad}rcl} Rr0Rr0 & \mapsto & Rr0L1 & Rr0Rl\star & \mapsto & Lr1Rr8 \\ Lr1L\star & \mapsto & Rr0L2 & Rl\star & \mapsto & Lr2Rl \\ Lr2L\star & \mapsto & Lr3Rr0 & Lr3L\star & \mapsto & Ll\star\star \end{array}$$

There are 3 stages in the extraction of the associate affine net. We show the intermediate trees in Figure 6.2 and the final net in Figure 6.3.

6.4 IMLAL2

Recall that IMAL and IMLAL games have been defined over a countable set of tokens $\mathcal{T}$. For the purpose of modelling second-order quantification we shall allow new tokens in addition to the elements of the fixed set $\mathcal{T}$. Let $\mathcal{T}'$ be the smallest set satisfying the following properties:

1. $\mathcal{T} \subseteq \mathcal{T}'$,
2. if $e \in \mathcal{T}$ and G is an IMLAL game constructed using tokens from $\mathcal{T}'$, then $[\forall e.G] \in \mathcal{T}'$.

Tokens of the form $[\forall e.G]$ are called ***second-order tokens***. The rest are ***ground tokens***. As we shall see shortly, a second-order token $[\forall a.G]$ is a description of the game operation: $A \mapsto G[A/a]$. 'α-equivalent' second-order tokens are considered identical.

6.4.1 Static games

Static games will be the objects that model second-order formulas.

Definition 6.59. ***Static games*** are IMLAL games built over the token set $\mathcal{T}'$.

We write $\mathcal{S}$ for the set of all static games. For instance,

$$\S a \otimes (b \multimap [\forall c.c \otimes \S(!c \multimap c)]) \in \mathcal{S};$$

the three tokens from left to right are referenced by $l\star$, rL and rR respectively.

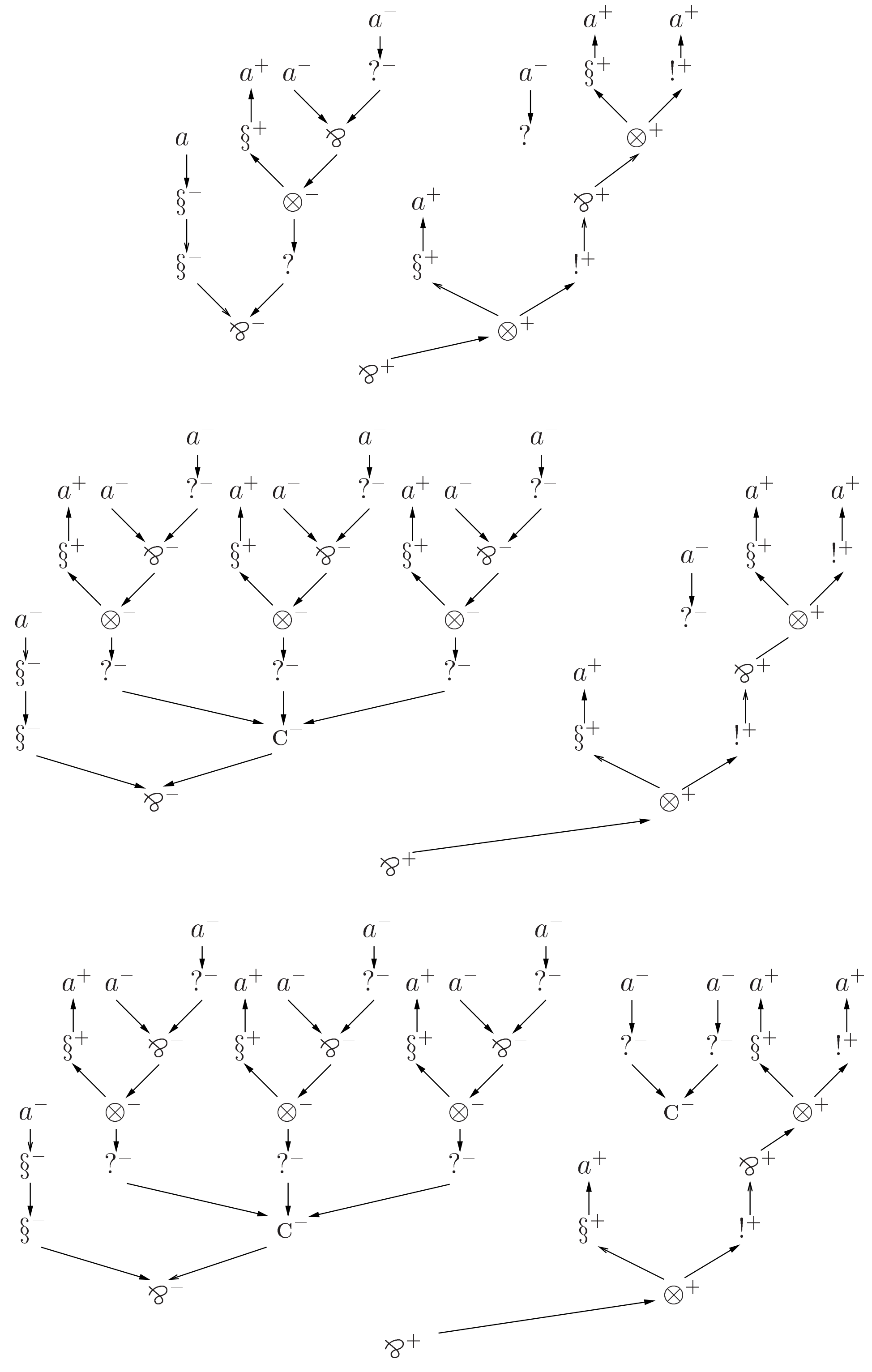


Figure 6.2: $\mathcal{T}^0_{f_0}, \mathcal{T}^1_{f_0}, \mathcal{T}^2_{f_0} = \mathcal{T}_{f_0}$ developed from the strategy in Example 6.58.

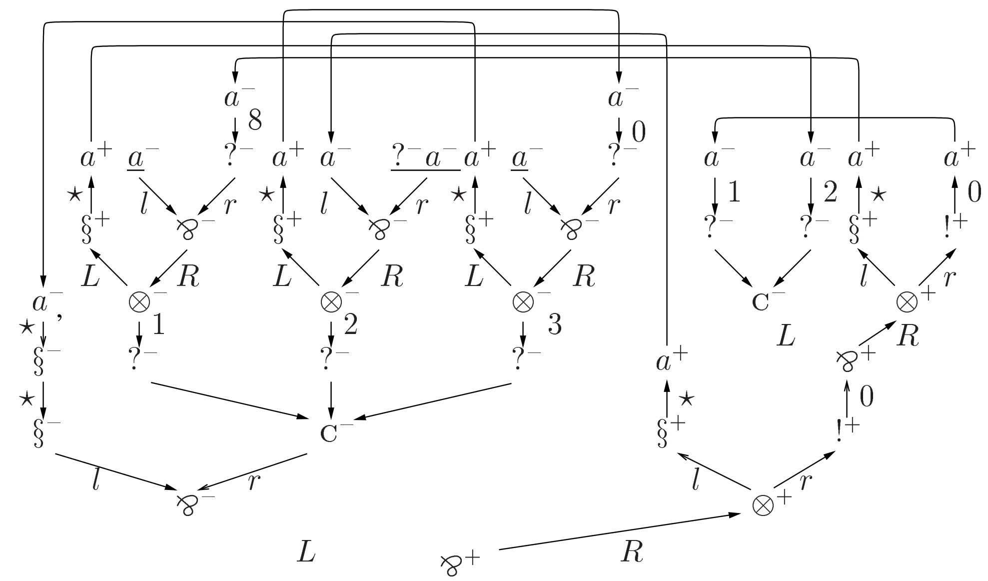


Figure 6.3: The essential net defined by the strategy from Example 6.58.

Static games already come equipped with a notion of move and position, but they do not make provision for interpreting the additional second-order rules. Nevertheless, we are going to take advantage of the existing structure to develop a suitable 'dynamic' framework. The key modification consists in allowing the game to change (evolve) during play. To track the progress of the evolution, we introduce a notion of *generalized static games*, which are generated from $\mathcal{T}'$ in a similar way to static games except that we use a more general version of the ! constructor. $!_{\{G_i\}_{i\in I}}G$ is a game in which plays indexed by elements of I must come from the associated game G_i, whereas those with indices from $\mathbb{N}\setminus I$ originate from G:

$$
\begin{array}{lcl}
M_{!_{\{G_i\}_{i\in I}}G} & = & \bigcup_{i\in I}\{\,i\,\}\cdot M_{G_i}\cup(\mathbb{N}\setminus I)\cdot M_G\\
\mu_{!_{\{G_i\}_{i\in I}}G}(im) & = & \text{if } i\notin I \text{ then } \mu_G(m) \text{ else } \mu_{G_i}(m)\\
P_{!_{\{G_i\}_{i\in I}}G} & = & \{s\in M^{alt}_{!_{\{G_i\}_{i\in I}}G} \;\mid\; \forall i\in I.\, s\restriction i\in P_{G_i},\,\forall i\notin I.\, s\restriction i\in P_G\}.
\end{array}
$$

Definition 6.60. The set $\mathcal{S}_g$ of ***generalized static games*** is the smallest set satisfying:

1. $G_\tau\in\mathcal{S}_g$ for $\tau\in\mathcal{T}'$ (i.e. singleton games with tokens from $\mathcal{T}'$);
2. if $G_1,G_2\in\mathcal{S}_g$, then $G_1\otimes G_2, G_1\multimap G_2\in\mathcal{S}_g$,
3. if $G\in\mathcal{S}_g$, then $\S G\in\mathcal{S}_g$;

4. if, given $I \subseteq \mathbb{N}$, $G_i \in \mathcal{S}_g$ for all $i \in I$ and $G \in \mathcal{S}_g$, then $!_{\{G_i\}_{i\in I}} G \in \mathcal{S}_g$.

Recall that we usually write just τ instead of G_τ and note that in general $[\forall e.H] \notin \mathcal{S}_g$ for $H \in \mathcal{S}_g$, e.g. $[\forall e.!_{1\mapsto e} e \otimes e] \notin \mathcal{S}_g$. However, if the generalized ! constructor does not occur in H, i.e. if $[\forall e.H] \in \mathcal{T}'$, we do have $[\forall e.H] \in \mathcal{S}_g$. The ! construction used for IMLAL corresponds to taking $I = \varnothing$. If for some $j \in I$, $G_j = G$, then

$$!_{\{G_i\}_{i\in I}} G = !_{\{G_i\}_{i\in I\setminus\{j\}}} G\,.$$

Each generalized static game has two kinds of tokens: ground ($\mathcal{T}$) and second-order ($\mathcal{T}' \setminus \mathcal{T}$). In our new setting moves with ground tokens will be playable in the same way as for IMLAL. Moves with second-order tokens will be treated differently—as descriptions of game evolutions. They cannot be played right away on their own. If a move with a second-order token θ is to be played, the player making it will be required to import a *static* game as an argument for the evolution operator θ, thus causing the current game to grow locally, with the game $\theta(A)$ grafted in the place previously occupied by θ. As the evolution may sometimes take place in a single thread of a !-subgame, the result of evolution from a static game need not be a static game, but it will always be a generalized static game. Furthermore, the evolution is closed with respect to generalized static games. After inducing an evolutionary step, the same player must proceed: he may play a move with a ground token from the new game and so complete the second-order move or he may continue the evolution process by playing a move with a second-order token and importing another (static) game, and so on. However, after finitely many such steps the player is required to play a move with a ground token from the evolved game, thus finally completing the second-order move.

Remark 6.61. An evolving game model first appeared in a LICS'97 paper [60] by Hughes, where he shows how a fully complete model for System F can be constructed. The pace of the evolution was slower therein: each move caused a single change to the game.

A more abstract version (for the $\forall, \Rightarrow$-fragment) that merged some of the evolutionary steps has been considered in [102]. It extended the innocent framework [64] and was based on prenex forms, which allowed for a separation of evolution arguments from the rest of a move. Players were required to provide evolution arguments before specifying the actual moves and games had to be continually transformed to prenex form after each move to maintain simplicity.

In our framework evolution arguments become integrated parts of moves and are intertwined with other parts of moves. The concept of prenex forms cannot be applied because of $\otimes$ and thus our definitions of moves and positions seem necessarily more involved.

It is worth comparing the framework of evolving games with the first game model of affine polymorphism (IMAL2) [1] in which evolution occurs implicitly. The board evolves in the minds of the players, but they never know what evolution arguments cause the changes. The only source of information about the growth of the game is the set of locations used in a position. Similarly to our framework, the game rules treat the two players in an asymmetric way: O can imagine he can effect any possible evolutionary step and thus he is allowed to play any move that evolution could make available, whereas P must play conservatively in that his move must be correct regardless of the argument O may have meant. For instance, after O begins in the game corresponding to $\forall X.X$, P is not able to respond, because O may have just imported a singleton game which gives P no room to play. Analogously, no P-moves can be made in $\forall X.X \otimes X$, but there is a simple strategy for (the unprovable sequent) $\forall X.X \otimes X \multimap \forall X.X \otimes \forall X.X$ based on copying moves from left to right. There will also be one for $\forall X.(A[X] \otimes B) \multimap \forall X.A[X] \otimes B$, which is not provable either. The complete absence of evolution arguments means that proofs that are different up to formulas hidden by existential quantification may have the same denotations.

Informally, IMLAL2 games are going to be static games that evolve into generalized ones during play. Starting from $G \in \mathcal{S}$, the interim (generalized static) game (which we can think of as the 'current game board') grows as the play unfolds, so that over the lifetime of a play, many more moves than are specified in the initial game G become available. We make the first step towards defining these generalized moves by introducing the *playable strings*.

Definition 6.62. A string $s \in (\{l, r, L, R, \star\} \cup \mathbb{N} \cup \mathcal{S})^*$ is a ***playable string*** of a *generalized static* game G if:

- either $s \in \{\, l, r, L, R, \star\,\}^*$, $s \in M_G$ and $\mu_G(s) \in \mathcal{T}$—then we say that G does not evolve (as a consequence of playing s) and call $\mu_G(s)$ the token of s;
- or there exist $A_1 \in \mathcal{S}$ (called ***evolution argument***) and $s_1 \in \{l, r, L, R, \star\}^*$ such that $s = s_1 A_1 s_2$, $s_1 \in M_G$, $\mu_G(s_1) = [\forall e.H]$ and s_2 is a playable string of the game $H[A_1/e]$. The token of s is defined to be the same as that of s_2 (although playability is defined with respect to a different game). If s_2 causes $H[A_1/e]$ to evolve into H', then s turns G into $G[H'/[\forall e.H]]$;
- or $s = s_1 n s_2$, where $s_1 \in \{\, l, r, L, R, \star\,\}^*$ points at a subgame of the shape $!_{\{H_i\}_{i \in I}} H$ and s_2 is playable for the game

$$K = \begin{cases} H_n & n \in I \\ H & n \notin I \end{cases}$$

Then if s_2 causes K to evolve into K', $s_1 n s_2$ makes G evolve into

$$G[!_{\{\, K_j \,\}_{j\in J}} H / !_{\{\, H_i \,\}_{i\in I}} H]\,,$$

where

$$J = \begin{cases} I & n \in I \\ I \cup \{\, n \,\} & n \notin I \end{cases}$$

and $K_j = H_j$ for all $j \in J \setminus \{\, n \,\}$, but $K_n = K'$.

If $s \in (\{l, r, L, R, \star\} \cup \mathbb{N} \cup \mathcal{S})^*$, we call $s \restriction \{l, r, L, R, \star\} \cup \mathbb{N}$ the ***location*** of s and write it as $\lfloor s \rfloor$. A string from $(\{l, r, L, R, \star\} \cup \mathbb{N})^*$ is an ***O-location*** if s contains an even number of L's. Otherwise it is a ***P-location***. The sets containing them are called L_O and L_P respectively. Intuitively, an O-location is the occurrence of an O-move in a game.
Note that in general it may take several evolution arguments to form a complete playable string and that they may need to be played consecutively. Suppose s is a playable move such that $s = m_1 E_1 m_2$, where m_1 does not end with an element of $\mathcal{S}$ (although it might contain such elements). We shall write evolution arguments as superscripts (e.g. $\cdots^{E_1}\cdots$ above). From the moment such an E_1 is played we say that the location $\lfloor m_1 \rfloor$ is being defined. It may turn out that m_2 begins with a contiguous segment of evolution arguments in which case all of them form part of the definition. If $m_2 = E_2 \cdots E_n m_3$ for $E_i \in \mathcal{S}$ $(2 \leqslant i \leqslant n)$ and m_3 begins with a character from $\{\, l, r, L, R, \star \,\} \cup \mathbb{N}$ we write $\lceil (\lfloor m_1 \rfloor, i) \rceil = E_i$ $(1 \leqslant i \leqslant n)$. Observe that in general $\lceil - \rceil : (L_O \cup L_P) \times \mathbb{N} \rightharpoonup \mathcal{S}$. A pair $(l, n) \in (L_O \cup L_P) \times \mathbb{N}$ will be called a ***complete location***. Complete O- and P-locations are defined in the obvious way. Thus the definition of a location is a specification of the content of several associated complete locations. If (l, n) has been defined by some move, so has (l, m) for any $1 \leqslant m < n$. A playable string s can be seen as defining a set $\{\, l_1, \cdots, l_{n_s} \,\} \subset L_O \cup L_P$ of locations. If l_i $(1 \leqslant i \leqslant n_s)$ contains d occurrences of natural numbers, we say that the associated evolution arguments were provided, used or inserted at depth d.

Definition 6.63. A playable string s for a *generalized static* game G is an ***O-playable string*** if the strings $l_1, \cdots, l_{n_s}, \lfloor s \rfloor$ are all O-locations. ***P-strings*** are defined by analogy.

In other words, playable strings acquire owners if the locations they define belong to the player who owns the location of the string.

Example 6.64. (i) The O-playable string

$$\underbrace{[\forall e.e] \cdots [\forall e.e]}_{k}\, a,$$

which consists of just $k+1$ evolution arguments, causes the singleton game $[\forall b.b]$ to evolve into a. There are k intermediate steps—all equal to $[\forall b.b]$—and a is the move's token. The move defines each $\lceil(\epsilon, i)\rceil$ $(1 \leqslant i \leqslant k)$ to be $[\forall e.e]$, but $\lceil(\epsilon, k+1)\rceil = a$.

(ii) $R1^{[\forall e.[\forall f.e \multimap f]]}4R^{!a \multimap \S[\forall b.b]}{}^{\S c \multimap d}RR$ is an O-playable string in $a \multimap ![\forall b.!(b \multimap b)]$ whose token is d. It makes the game evolve into

$$a \multimap !_{\{\, 1 \mapsto !_{\{\, 4 \mapsto (!a \multimap \S[\forall b.b]) \multimap (\S c \multimap d)\,\}}{}^{[\forall e.[\forall f.e \multimap f]] \multimap [\forall e.[\forall f.e \multimap f]]}\,\}}[\forall b.!(b \multimap b)]$$

The playable string $R14LR\star^{b}$ develops it further into

$$a \multimap !_{\{\, 1 \mapsto !_{\{\, 4 \mapsto (!a \multimap \S b) \multimap (\S c \multimap d)\,\}}{}^{[\forall e.[\forall f.e \multimap f]] \multimap [\forall e.[\forall f.e \multimap f]]}\,\}}[\forall b.!(b \multimap b)]$$

(iii) ${}^{[\forall b.b] \multimap a}L^{a \multimap b}L$ is a playable string for $[\forall e.e]$, which transforms the game to $(a \multimap b) \multimap a$, but it is neither O- nor P-playable: it defines L (a P-location), while the move's location is an O-location.

Definition 6.65 (Moves). For a *static* game G the set M_G of moves is defined by

- if m is an O- or P-playable string for G (construed as a generalized static game), then $m \in M_G$;
- if there exist $m_0, \cdots, m_k \in M_G$ and $G_1, \cdots, G_{k+1} \in \mathcal{S}_g$ such that m_i turns G_i into G_{i+1} $(0 \leqslant i \leqslant k,\ G_0 = G)$ and m is a playable string for G_{k+1}, then $m \in M_G$.

Definition 6.66 (Positions). The empty sequence is a position. If a finite sequence s of moves is a position and G' is the evolved game at that point, then sm is a position provided

(i) m is a playable string of G' (let G'' be the resultant game);

(ii) sm is alternating, i.e. if $|s|$ is even, m is O-playable, otherwise it must be P-playable;

(iii) the sequence consisting of the respective locations of the elements in sm is a position of G'' (in the sense of IMLAL games over $\mathcal{T}'$).

Definition 6.67. ***IMLAL2 games*** are (static) games with moves and positions defined as above.

Example 6.68. We examine the maximal position

$$\underset{G_1}{(R\star R^{!a}\star^{a}{}^{\S a})}\ \underset{G_2}{(L1R^{!(b\otimes c)}2l)}\ \underset{G_3}{(L1L^{!d\otimes\S e}l2)}\ \underset{G_4}{(L2R^{!f\otimes !g}l5)}\ \underset{G_5}{(L2L^{!(h\otimes i)}6l)}\ \underset{G_6}{(R\star L^{\S j!k}r\star)}$$

for the game

$$!([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap [\forall x.\S[\forall y.[\forall z.y]]])$$

where the sequence $G_1 - G_6$ represents the evolution of the initial game:

$$G_1\ \ !([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap \S a)$$

$$G_2\ \ !_{\{1\mapsto[\forall x.x]\multimap !(b\otimes c)\}}([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap \S a)$$

$$G_3\ \ !_{\{1\mapsto !d\otimes\S e\multimap !(b\otimes c)\}}([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap \S a)$$

$$G_4\ \ !_{\left\{\begin{array}{lll}1 & \mapsto & !d\otimes\S e\multimap !(b\otimes c)\\ 2 & \mapsto & [\forall x.x]\multimap !f\otimes !g\end{array}\right\}}([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap \S a)$$

$$G_5\ \ !_{\left\{\begin{array}{lll}1 & \mapsto & !d\otimes\S e\multimap !(b\otimes c)\\ 2 & \mapsto & !(h\otimes i)\multimap !f\otimes !g\end{array}\right\}}([\forall x.x] \multimap [\forall x.x]) \multimap \S([\forall y.[\forall x.x \otimes y]] \multimap \S a)$$

$$G_6\ \ !_{\left\{\begin{array}{lll}1 & \mapsto & !d\otimes\S e\multimap !(b\otimes c)\\ 2 & \mapsto & !(h\otimes i)\multimap !f\otimes !g\end{array}\right\}}([\forall x.x] \multimap [\forall x.x]) \multimap \S(!k \otimes \S j \multimap \S a)$$

After the first move we have $\lceil(R\star R,1)\rceil = !a$, $\lceil(R\star R\star,1)\rceil = a$ and $\lceil(R\star R\star,2)\rceil = \S a$. After the second one $\lceil(L1R,1)\rceil = !(b\otimes c)$. The third move defines $\lceil(L1L,1)\rceil$ to be $!d\otimes\S e$, the next one makes $\lceil(L2R,1)\rceil$ equal to $!f\otimes !g$. The fifth one sets $\lceil(L2L,1)\rceil$ to $!(h\otimes i)$ and finally $\lceil(R\star L,1)\rceil$ and $\lceil(R\star L,2)\rceil$ are defined to be $\S j$ and $!k$ respectively.

In our next example the original game is $[\forall c.c]$. Let $V = [\forall c.c] \multimap a$. For each i, the following sequence is a position

$$^{V}R \qquad L^{V}R \qquad LL^{V}R \qquad \cdots \qquad \underbrace{L\cdots L}_{i}{}^{V}R$$

and the game after i steps is G_i, where $G_0 = [\forall c.c]$ and $G_{i+1} = G_i \multimap a$ for $i \geqslant 0$. Besides, $\lceil\underbrace{(L\cdots L,1)}_{i}\rceil = V$ for any $0 \leqslant i \leqslant i$.

Of course, it makes sense to consider strategies in the new evolving framework as well as token-reflection and totality. Strategies can also be composed using the standard mechanism. For instance, networked and suitably networked strategies form a class closed under composition as do token-reflecting strategies. Next we focus on several new properties with the aim of characterizing the proof space of IMLAL2.

6.4.2 Symbolic and expanded strategies

First we consider the simple scenario in which the evolution arguments provided by O are guaranteed to be ground tokens (regarded as singleton games). Formally we say that O plays ***symbolically*** (or makes ***symbolic moves*** in a position, if for each complete O-location (l, n) defined therein we have $\lceil (l, n) \rceil \in \mathcal{T}$. For the rest of the section we assume that O plays symbolically. We say that a P-strategy σ is ***symbolic***, just in case for every even-length $s \in \sigma$ if sm is a position then $sm \in \sigma$, if and only if the O-move m is symbolic.

The purpose of the next definition is to introduce strategies that are parametric in the sense that the evolution arguments given by P are determined by those given by O in a schematic fashion. First we set $\mathcal{S}(L_O \times \mathbb{N})$ to be the collection of formal objects defined by the following rules:

- every ground token and every complete O-location are in $\mathcal{S}(L_O \times \mathbb{N})$ i.e. $\mathcal{T}, L_O \times \mathbb{N} \subseteq \mathcal{S}(L_O \times \mathbb{N})$,
- if G_1 and G_2 are in $\mathcal{S}(L_O \times \mathbb{N})$ then $G_1 \otimes G_2$, $G_1 \multimap G_2$, $\S G_1$, $!G_1$ are in $\mathcal{S}(L_O \times \mathbb{N})$,
- if $G \in \mathcal{S}(L_O \times \mathbb{N})$ then for each $e \in \mathcal{T}$, $[\forall e.G]$ is in $\mathcal{S}(L_O \times \mathbb{N})$.

For example, we have $c \multimap (R3r\star, 1998) \otimes [\forall a.((L2Lr, 2001) \multimap a \otimes b)] \in \mathcal{S}(L_O \times \mathbb{N})$. Informally, $\mathcal{S}(L_O \times \mathbb{N})$ is the collection of static games in which complete O-locations are treated as extra ground tokens.

Definition 6.69. The pair $f : L_O \rightharpoonup L_P$ and $F : L_P \times \mathbb{N} \rightharpoonup \mathcal{S}(L_O \times \mathbb{N})$, where f is injective, defines a *symbolic* strategy σ if for all even-length $sm_O m_P \in \sigma$ (m_O is necessarily symbolic):

- $\lfloor m_P \rfloor = f(\lfloor m_O \rfloor)$,
- for all $u, v \in (\{l, r, L, R, \star\} \cup \mathbb{N} \cup \mathcal{S})^*$ such that u does not end and v does not begin with an evolution argument, and for $A_1, \cdots, A_n \in \mathcal{S}$ such that $m_P = u^{A_1 \cdots A_n} v$ we have $A_i = F'(\lfloor u \rfloor, i)$ $(1 \leqslant i \leqslant n)$, where $F'(\lfloor u \rfloor, i)$ is obtained from $F(\lfloor u \rfloor, i)$ by replacing each complete O-location (l, m) therein with $\lceil (l, m) \rceil$ (as defined in sm_O). Each such $\lceil (l, m) \rceil$ must already be defined in sm_O.

We write $\sigma = \sigma_{f,F}$ if f, F are the least such functions.

By definition, it is easy to see that

1. f and F are related: each P-location in $dom(F)$ is a prefix of some element in $cod(f)$.
2. By leastness of f and F and injectivity of the former, elements of $dom(f) \cup cod(f)$ are incomparable with respect to the prefix order $\leq$. Hence, given $s \in (\{\, l, r, L, R, \star \,\} \cup \mathbb{N})^*$, there exists at most one $s' \in dom(f) \cup cod(f)$ such that $s = s'x$.

Note that a symbolic strategy is *location-wise* history-free, in the sense that the location of P's response at any position depends only on the location of the preceding O-move.

Example 6.70.

- The identity (symbolic) strategy for

  $$[\forall x.[\forall y.x \otimes y]] \multimap [\forall x.[\forall y.x \otimes y]]$$

  is generated by the following pair of maps:

  $$\begin{array}{rclrcl} f(Rl) & = & Ll & F(L,1) & = & (R,1) \\ f(Rr) & = & Lr & F(L,2) & = & (R,2). \end{array}$$

  In general, $\sigma_{f,F}$ is an identity strategy if and only if it is total and

  – for all $Rx, Ly \in dom\, f$

  $$f(Rx) = Lx \qquad f(Ly) = Ry$$

  – for all $(Rx, n), (Ly, m) \in dom\, F$

  $$f(Rx, n) = (Lx, n) \qquad f(Ly, m) = (Ry, m).$$

- The symbolic strategy τ for

  $$[\forall z.z] \multimap [\forall x.[\forall y.\S(x \multimap y)]]$$

  defined by

  $$\begin{array}{rclrcl} f(R \star R) & = & L3Rr & F(L,1) & = & [\forall y.!((R,1) \multimap [\forall v.v \otimes y])] \\ f(L3L) & = & L4Rl & F(L,2) & = & (R,2) \\ f(L4L) & = & R \star L & F(L3R,1) & = & (R,2) \\ & & & F(L4R,1) & = & (R,1) \end{array}$$

  contains the following maximal position:

  $$(R^{a\,b} \star R)\,(L^{[\forall y.!(a \multimap [\forall v.v \otimes y])]\,b}3R^{b}r)\,(L3L)\,(L4R^{a}l)\,(L4L)\,(R \star L).$$

Symbolic strategies cannot be composed because of the lack of symmetry in the way evolution arguments are used by the two players. During interaction, in the middle component, the roles of O and P are united and the asymmetry prevents any meaningful interaction from taking place. To solve the problem, we define how P equipped with a symbolic strategy could respond to not necessarily symbolic moves of O. The idea is that he should use F to import an appropriate game even though O may have imported a non-singleton game. P will also have to point at some move of that new game. Fortunately, O must have done the same when playing his argument, so P will be able to 'borrow' the pointer from the preceding P-move in addition to using the location suggested by f. Such 'copy-cat' extensions are a semantic form of η-expansion [60, 74].

Definition 6.71. Consider a symbolic strategy $\sigma = \sigma_{f,F}$. The ***expanded strategy*** $\overline{\sigma}$ is defined by the following algorithm.

Suppose the odd-length position $sm \in \overline{\sigma}$ is such that $\lfloor m \rfloor = t$. Find $u \leq t$ such that $u \in \mathit{cod}(f) \cup \mathit{dom}(f)$ (by previous remarks if such a u exists it will be unique). This decomposes the O-move m into $m_u m_v$ such that $t = uv$, m_u ends in an element of $\{\, l, r, L, R, \star \,\} \cup \mathbb{N}$ and $\lfloor m_u \rfloor = u$. There are two cases.

(i) If $u \in \mathit{dom}(f)$, play $\overline{f(u)}m_v$, i.e. $sm(\overline{f(u)}m_v) \in \overline{\sigma}$ (provided it is a position).

(ii) If $u \in \mathit{cod}(f)$, play $f^{-1}(u)m_v$, i.e. $sm(f^{-1}(u)m_v) \in \overline{\sigma}$ (if it is a position).

In case (i) if u is encountered in the play for the first time, $\overline{f(u)}$ is obtained from $f(u)$ by inserting appropriate evolution arguments using F (as specified in Definition 6.69); otherwise $\overline{f(u)} = f(u)$.

Example 6.72. We come back to τ from Example 6.70 and show a position from $\overline{\tau}$ in a top-down fashion:

$$\begin{array}{l}
(R^{[\forall a.[\forall b.b\otimes a]]\,[\forall c.c]} \star R^{d\otimes e} r) \\
(L^{[\forall y.!([\forall a.[\forall b.b\otimes a]]\multimap[\forall v.v\otimes y])]\,[\forall c.c]} 3 R^{[\forall c.c]} l^{d\otimes e} r) \\
(L3L^{!f\,\S h} l\star) \\
(L4R^{[\forall a.[\forall b.b\otimes a]]} l^{!f\,\S h} l\star)\,.
\end{array}$$

Remark 6.73. In any play the evolution argument for each complete location is given explicitly only once. Thus in case (i) $\overline{f(u)} = f(u)$ if the u in question has already been met in the history of play. In case (ii) all locations that are prefixes of $f^{-1}(u)$ have already been defined, since the corresponding instance of case (i) must have been encountered first. This explains the apparent asymmetry in the two cases.

Remark 6.74. If $\sigma_{f,F}$ is networked, then the same networks exist in $\overline{\sigma}$, but in addition we may have networks created by the m_v moves—they will always have exactly two threads of the same kind.

Lemma 6.75 (Zigzag). In any position of an expanded strategy generated by f and F, applications of (i) and (ii) for any $u \in dom\, f$ alternate and f will be applied in case (i) first, if at all.

Proof. Otherwise, in the generalized static subgame rooted at u, we would have two consecutive moves by the same player or a position beginning with a P-move. □

It is straightforward to see that, like symbolic strategies, expanded strategies are (location-wise) history-free. We also have:

Proposition 6.76. Expanded strategies are closed under composition.

Proof. The underlying symbolic strategy of the composite strategy is defined by interaction sequences in which O-moves in A and C are symbolic. The extraction of the two conditions poses no problems—the reasoning is actually the same as in the proof that history-free strategies compose [5]. □

Many properties of expanded strategies are inherited from the underlying symbolic strategies:

Proposition 6.77. $\overline{\sigma}$ is (compactly, suitably) networked if and only if σ satisfies the respective property. If σ is token-reflecting, then $\overline{\sigma}$ is total if and only if σ is.

Proof. For (compactly, suitably) networked strategies, see Remark 6.74. For totality, we appeal to Proposition 6.21. Were $\overline{\sigma}$ not total, we would be able to find an IMAL game for which the expansion of a total strategy is not total, which contradicts the Proposition. □

6.4.3 Local strategies

For networked expanded strategies we consider yet another constraint concerning the manner in which evolution arguments supplied by O can be used by P. We shall require that after O has used an argument at depth i, it can only be used by P inside the associated i-network. In particular, evolution arguments played by P and referring (via F) to arguments provided by O cannot be inserted by P at any depth lower than that at which they were introduced by O.

Definition 6.78. A *networked symbolic* strategy is ***local*** if for all $(u, m) \in dom\, F$ whenever $(l, n) \in L_O \times \mathbb{N}$ occurs in $F(u, m)$ and l has depth d, then u has depth greater than or equal to d and the threads represented by the d-indices of l and u belong to the same network.

In expanded strategies evolution arguments proposed by O are used in two ways: with the help of F and by copying. We already know that if the underlying symbolic strategy is networked copying takes place between two threads of the same network. Therefore, if the symbolic strategy is in addition local, the induced expanded strategy also follows the spirit of locality. Accordingly, we call it a ***local expanded strategy***.

Example 6.79. Consider the symbolic strategy for the game $\forall a.a \multimap \S[\forall b.b]$ defined by (f, F) such that

$$f(R\star) = LR1, \qquad f(LLR3) = LLL4,$$

and

$$F(L, 1) = (![\forall d.d] \multimap !(R\star, 1)) \multimap !(R\star, 1), \qquad F(LLL4, 1) = (R\star, 1).$$

It contains the following position

$$(R\star^c)\,(L^{(![\forall d.d] \multimap !c) \multimap !c}R1)\,(LLR3)\,(LLL4^c).$$

The strategy is *not* local for two reasons:

- $R\star$ occurs in $F(L, 1)$ and the depth of L is smaller than that of $R\star$,
- $LLL4$ and $R\star$ are from different networks.

However, if we change the game to $\forall a.a \multimap [\forall b.\S b]$, $R\star$ to R in the definition of F (keeping the same f), we will get a local strategy with the play

$$(R^c\star)\,(L^{(![\forall d.d] \multimap !c) \multimap !c}R1)\,(LLR3)\,(LLL4^c)\,.$$

Proposition 6.80. Local expanded strategies compose.

Proof. Follows easily from the way the network function of the composite strategy is defined. $\square$

6.4.4 Bounded strategies

Now we would like to introduce totality into our framework. The first problem we need to face is the fact that total expanded strategies do not compose in general. For IMLAL we resorted to the notion of compactly networked strategies to ensure that—in conjunction with this additional property—totality is compositional. In the present framework this is no longer sufficient, as there is a new source of infinite behaviour—unbounded evolution.

Example 6.81. Consider the strategy for $[\forall x.x] \multimap [\forall y.y]$ in which P replies replies to the first O-move R^a with $L^{([\forall x.x]\multimap[\forall y.y])\multimap a}R$. This puts O in the same position as at the beginning of the game and the play could continue indefinitely.

As strategies corresponding to IMLAL2 proofs are in general infinite, we cannot simply require that f and F be finite (this would work for IMAL2 though). Instead, we are going to impose three conditions that prevent the game from growing too large during play, i.e there must exist a bound on the size of the developing game [4]. However, this condition is not preserved by composition! This is due to the existence of two other factors that can induce unbounded growth during interaction. We identify them in the following examples.

Example 6.82. Consider a symbolic strategy σ for the game

$$!([\forall a.a] \multimap [\forall a.a]) \multimap !([\forall a.a] \multimap [\forall a.a])$$

such that when O opens the thread indexed by Rn, P will open n threads with indices prefixed by L and copy the argument used by O. Roughly, O's choice of an index will determine which 'numeral' is played out by P. Here is the maximal play of the strategy in which 2 is played

$$(R2R^a)\ (L1R^a)\ (L1L^b)\ (L2R^b)\ (L2L^c)\ (R2L^c).$$

In general when O plays (RiR^a), P can use threads numbered

$$\frac{i(i-1)}{2}, \frac{i(i-1)}{2}+1, \cdots, \frac{i(i-1)}{2}+(i-1) = \frac{(i+1)i}{2}-1\,.$$

Let τ be another symbolic strategy, this time for

$$\varnothing \multimap !([\forall a.a] \multimap [\forall a.a])$$

[4] By the ***size of a game*** we mean the depth of the underlying syntactic tree in which second-order tokens (but not quantifiers) are unfolded, so, e.g. the size of $[\forall a.a \multimap a] \multimap b$ is 3.

in which P duplicates the argument provided by O:

$$(RiR^a)\,(RiL^{a\otimes a}r).$$

In $\overline{\tau};\overline{\sigma}$ the size of the game can get arbitrarily large. This effect is caused by the unbounded number of threads in networks from σ: in τ P plays $a\otimes a$, which $\sigma_{f,F}$ copies, and $\tau_{g,G}$ then duplicates so that $(a\otimes a)\otimes(a\otimes a)$ is eventually played. The longer the exchange takes, the larger the size of the game.

The strategy in question is not suitably networked, which might seem to be the source of problems. This is not true: the same problem recurs, when the above strategy is modified to one for

$$!!([\forall a.a] \multimap [\forall a.a]) \multimap !\S([\forall a.a] \multimap [\forall a.a]).$$

In order to exclude this kind of behaviour we will be interested in strategies for which there exists a bound on the number of threads in networks.

Example 6.83. Let σ $:![\forall a.[\forall b.a]] \multimap ![\forall a.a]$ be the symbolic strategy in which P imitates O, but in addition, if O opens Rn, P will substitute n copies of the argument $\lceil Rn, 1\rceil$ for b

$$(Rn^a)\;(Ln^{a\,\underbrace{((a\otimes a)\otimes\cdots)\otimes a}_{n}})\,.$$

The other strategy $\tau : [\forall a.[\forall b.(b \multimap b) \multimap a]] \multimap [\forall a.[\forall b.a]]$ is basically an expansion of the obvious IMAL strategy for $((b \multimap b) \multimap a) \multimap a$. P simply copies the arguments given by O:

$$(R^{ab})\;(L^{ab}R)\;(LLR)\;(LLL).$$

As before, composition leads to unlimited development of the game, though in the underlying symbolic strategies the size of the game is bounded. This time the size of arguments used in $\sigma_{f,F}$ is to blame. Arbitrarily large arguments are played, but they do not contribute to the size of the game in $\sigma_{f,F}$. Nevertheless, their presence can be detected and exploited during composition to produce games of arbitrary size.

Motivated by the three examples, we define a class of strategies that satisfy all of the proposed conditions and show that their conjunction is preserved by composition.

Definition 6.84. A *local expanded* strategy $\overline{\sigma}_{f,F}$ is ***bounded*** if and only if

1. $\sigma_{f,F}$ is networked and there is a (uniform) bound NT on the number of threads in its networks (in particular $\sigma_{f,F}$ is compactly networked),

2. there is a bound HA on the size of evolution arguments provided by F,
3. the size of the evolving game in any play from $\sigma_{f,F}$ never exceeds HG.

Example 6.85. Boundedness does not rule out infinite behaviour. Rather, it restricts the pattern which can be used to generate infinite plays. Consider the identity strategy for the game $[\forall x.x] \multimap [\forall y.y]$ which is defined by $f(R) = L$ and $F(L, 1) = (R, 1)$. Suppose O plays $R^{[\forall x.x]\multimap a}R$. P plays copy-cat and so responds with $L^{[\forall x.x]\multimap a}R$. From this point onwards, both players can behave analogously, engaging in an infinite exchange that can make the game grow in every step.

Lemma 6.86. Bounded strategies compose.

Proof. Suppose $\overline{\sigma}_{f_1,F_1}$ and $\overline{\sigma}_{f_2,F_2}$ are bounded strategies and NT, HA, HG are the larger of the respective bounds associated with them. Let $\overline{\sigma}_{g,G} = \sigma_{f_1,F_1}; \sigma_{f_2,F_2}$ (by Proposition 6.76 expanded strategies compose). We shall focus on $\sigma_{g,G}$ and show that the requisite bounds exist. Hence, we consider interaction sequences in which O moves in A and C are symbolic—these are the sequences that define $\sigma_{g,G}$. Note that in such sequences O-moves in A or C do not increase the size of the evolving game. We shall show that for any $i \in \mathbb{N}$ there exist bounds NT_i and HA_i such that

(a) for any i the number of threads interacting in an interactive tree of threads at depth i is bounded by NT_i (hence, the number of threads in any network of the composite strategy is also bounded),

(b) the size of evolution arguments used at depth i is bounded by HA_i.

Finally, given any interaction sequence, we shall show that after all evolution arguments that can possibly be used at depths $0, 1, \cdots, i$ are taken into account:

(c) the size of the resultant game remains smaller than some HG_i.

We reason by induction on depth i. At depth 0 (a) holds vacuously and $NT_0 = 1$ by convention. The length of locations at which evolution arguments can be supplied by F_1 or F_2 is restricted by HG. Thus there can be only a finite number of them, say, N_0. Once provided (case (i) of Definition 6.71), they can be copied in subsequent moves (m_v in case (i) or (ii)) and sometimes used in other arguments following F_1 or F_2 (case (i) again). Since F_1 and F_2 will be used only N_0 times, the size of the largest argument that can occur in an interaction sequence of the kind under consideration can reach $HA_0 = N_0 \cdot HA$ (this is because arguments that are used earlier can be used to amplify subsequent ones). Thus after taking into account all the arguments played at depth 0 according to case (i) the size of the game can grow to $HG + HA_0$. The copying of arguments by m_v can affect the size too. Suppose an evolution argument is used as part of some move m as in case (i) and this occurs at location l. Let $m_1, \cdots, m_k$ be the moves that follow m in which the argument has been copied at locations $l_1, \cdots, l_k$ respectively. Note that this copying can only take place if each move m_i $(1 \leqslant i \leqslant k)$ has been made using f_1 or f_2 for arguments of depth 0. Because of the bound HG, there can be only a finite number of such arguments, say, they form a finite subset f'. Suppose each m_i has been made using $x_i \in dom\, f'$ i.e. $\lfloor m_i \rfloor$ is prefixed by either x_i or $f(x_i)$. Let $\delta_i = |f'(x_i)| - |x_i|$ for $i = 1, \cdots, k$. Then $|l_{i+1}| - |l_i| = |m_{i+1}| - |m_i| = (-1)^{\epsilon_i}\delta_i$ for $\epsilon_i \in \{ 0, 1 \}$ and $i = 1, \cdots, k-1$. Thus:

$$|l_k| = |l| + \sum_{i=1}^{k}(-1)^{\epsilon_i}\delta_i.$$

Let $S = \sum_{x \in dom f'} ||f'(x)| - |x||$ (the sum is finite!). By Lemma 6.75 $\sum_{i=1}^{k}(-1)^{\epsilon_i}\delta_i \leqslant S$, so

$$|l_k| \leqslant S + |l|.$$

Because $|l| \leqslant HG$ and the size of arguments is bounded by HA_0, the size of the game after all possible modifications caused by arguments used at depth 0 will not exceed $HG_0 = HG + HA_0 + S$.

Next we make the inductive step which generalizes the previous reasoning.

(a) Consider a thread at depth $i+1$. Its i-index determines a thread at depth i, which belongs to some interactive tree of threads with at most NT_i threads by induction hypothesis (a). By (c) the number of $i+1$-bases that extend the i-indices from the above-mentioned interactive tree is finite and bounded, because only evolution arguments played at depths $0, 1, \cdots, i$ can contribute them. Therefore, the depth of any interactive tree of threads at $i+1$ is also bounded by Proposition 6.32. Because the number of threads in a network in $\overline{\sigma}_{f_1,F_1}$ and $\overline{\sigma}_{f_2,F_2}$ is bounded (by NT), the number of threads in an interactive tree of threads at $i+1$ is bounded by some NT_{i+1}.

(b) Consider evolution arguments used at depth $i+1$. Their primary sources are F_1 and F_2 (case (i) in Definition 6.71). The alternative source is copying (via m_v), which does not change the size of arguments. For this reason, it suffices to consider the former case only.

Because the strategies we consider are local, an evolution argument provided by O can be used by P (in the sense of case (i)) only within the same network. However, during interaction, arguments introduced by P are seen as provided by O by the other strategy. Therefore, in any interaction sequence, once an argument is introduced by O, the argument itself or its copy might be used in any thread of the associated interactive tree of threads. Because

- the length of the location at which arguments are introduced by case (i) is bounded by HG, and
- the number of threads in an interactive tree of threads is smaller than NT_{i+1} by (a),

the number of arguments introduced at depth $i+1$ using F_1 and F_2 inside an interactive tree of threads at depth $i+1$ is uniformly bounded by some N_{i+1}. After being introduced in this way, they can be copied and possibly used later as parts of other arguments as specified F_1 or F_2. Besides, arguments introduced at depths $0, 1, \cdots, i$ can be used in arguments provided at $i+1$. Let $HA' = max(HA_0, HA_1, \cdots, HA_i)$. Then the size of arguments that occur at depth $i+1$ will be bounded by $HA_{i+1} = N_{i+1} \cdot HA + HA'$.

(c) Recall that for (c) we consider the size of the game taking into account only the arguments used at depths $1, 2, \cdots, i+1$. By induction hypothesis the size is bounded by HG_i at depth i. When evolution arguments are used at depth $i+1$ as in case (i) from Definition 6.71, this takes place at some location l whose length is less than HG. By (b) the size of the argument does not exceed HA_{i+1} then, so such arguments alone will not make the game larger than $max(HG_i, HA_{i+1} + HG)$. Further increases can be effected through copying from the above locations (m_v in Definition 6.71), but note that everything must occur in threads from some interactive tree of threads. The copying moves are then made using f_1 or f_2 for strings which are either prefixes of the indices of the $i+1$-threads involved or their extensions with depth $i+1$. Their number is bounded, since the size of moves in f_1 and f_2 is bounded (by HG) and NT_{i+1} bounds the number of relevant threads. Therefore, only a bounded number of elements from f_1 and f_2 can be involved, say, from some $f' \subseteq f_1 \cup f_2$. The copying may continue for a number of moves. Suppose the original argument is introduced at location l (hence $|l| \leqslant HG$) where l is part of some move m, Let $m_1, \cdots, m_k$ be the subsequent moves that copy the argument to locations $l_1, \cdots, l_k$ respectively such that each m_i ($1 \leqslant i \leqslant k$) is made using $x_i \in dom\, f' \cup cod\, f'$. Let $\delta_i = |f(x_i)| - |x_i|$ for $i = 1, \cdots, k$. Then $|l_{i+1}| - |l_i| = (-1)^{\epsilon_i}\delta_i$, where $\epsilon_i \in \{ 0, 1 \}$. Thus

$$|l_k| = |l| + \sum_{i=1}^{k} (-1)^{\epsilon_i}\delta_i.$$

Let $S \geqslant \sum_{x \in dom f'} ||f(x)| - |x||$ (as the number of elements from f_1 and f_2 involved is bounded and $||f(x)| - |x|| \leqslant HG$, we can choose a uniform bound S). By Lemma 6.75 $\sum_{i=1}^{k}(-1)^{\epsilon_i}\delta_i \leqslant S$, so $|l_k| \leqslant |l| + S$. Hence, copying via m_v might increase the size of the game by S, in relation to the effect of arguments introduced by case (i). Therefore, the overall height of the game (after arguments at depth $0, 1, \cdots, i+1$ are considered in each interaction sequence with O playing symbolically in A and C) is bounded by $HG_{i+1} = max(HG_i, HA_{i+1} + HG + S)$.

Because the length of strings in f_1 and f_2 is bounded (by HG), so is their depth. Let d_1 and d_2 be the respective bounds. Hence, F_1 or F_2 provide evolution arguments only at depths that do not exceed $d = max(d_1, d_2)$. Thus for $i > d$ case (i) never occurs and all arguments at these depths are copies of those provided by O in A or C, i.e. they are ground tokens. Clearly, this will not change the size of the game. Also recall that any network at these depths will have two threads, so the same will be true of each network from the composite strategy.

To sum up, all the anticipated bounds exist:

- the composite strategy is compactly networked and the number of threads in any network is smaller than $NT' = max(2, NT_1, \cdots, NT_d)$,
- the size of evolution arguments in the composite strategy is bounded by $HA' = HA_d$
- the size of the game during play from $\sigma_{g,G}$ does not exceed HG_d.

□

As expected, the notion of boundedness guarantees that totality of strategies is preserved during composition.

Theorem 6.87. Total bounded strategies compose.

Proof. By Proposition 6.76 it suffices to prove that no infinite chattering can occur if O plays symbolically in A and C. From the proof of the preceding Lemma we know that the size of the evolving game remains bounded throughout such an infinite play. Suppose all the moves inside the infinite sequence in the middle component have depth d. All of them originate then from threads belonging to the same interactive tree of threads at depths $1, 2, \cdots, d$, which—by the proof of the last Lemma—have bounded numbers of threads. Hence, the size of the game during the infinite play is bounded and so is the number of indices of threads that the moves represent. Clearly, only a finite number of moves can be played under these conditions. □

The last two of the three examples motivating the introduction of boundedness were strategies that were not behaving 'consistently' with respect to !-threads. This suggests that once a suitable notion of consistency is incorporated into our framework, it should be possible to weaken the definition of bounded strategies (Definition 6.84). Indeed, Proposition 6.96 will show that consistent strategies which are compactly networked and satisfy

3. the size of the evolving game in any play from $\sigma_{f,F}$ never exceeds HG

can be finitely represented and, therefore, 1. and 2. of Definition 6.84 hold then. Hence, boundedness reduces to compact networking and 3. for consistent strategies.

6.4.5 Consistent strategies

We have used numeric indices to indicate that fresh or old threads are used in plays, so, as for IMLAL, we need to impose a notion of equivalence that will make up for this overly concrete representation. Fortunately, the previous formulation is general enough to be used in the current setting, but we need to take evolution arguments into account as well: in equivalent positions players must employ the same evolution arguments. Given a position $s = s_1 \cdots s_{|s|}$, we define $\lfloor s \rfloor$ as $\lfloor s_1 \rfloor \cdots \lfloor s_{|s|} \rfloor$.

Definition 6.88. Given $s^1, s^2 \in P_G$, where G is a *static* game, $s^1 \approx_e s^2$ holds if and only if

1. $\lfloor s^1 \rfloor \approx \lfloor s^2 \rfloor$ (Definition 6.39),
2. for $i = 1, 2$
$$\text{if } s^i_j = c_1 \cdots c^A_k t^i_j, \text{then } s^{3-i}_j = d_1 \cdots d^A_k \cdots t^{3-i}_j,$$
where $c_l, d_l \in X$ $(1 \leqslant l \leqslant k)$, $t^1_j, t^2_j \in X^*$ and $X = \{\, l, r, L, R, \star \,\} \cup \mathbb{N} \cup \mathcal{S}$.

$\approx_e$ extends to a partial equivalence relation $\approx_e$ on strategies in the same way as $\approx$ did for IMLAL. Similarly, we have:

Lemma 6.89. if $\sigma_1 \approx_e \sigma_2$ and $\tau_1 \approx_e \tau_2$, then $\sigma_1; \tau_1 \approx_e \sigma_2; \tau_2$.

For IMLAL games the basic relation $\approx$ on strategies was unsatisfactory because it could not account for the global nature of networks. This was a serious defect, as networks are used to model the structure of proofs. For this reason we considered auxiliary extensions $\widehat{\sigma}$ of strategies, which extracted more information about σ during play so that $\approx$-equivalence could become sufficient to characterize strategies with equivalent network structures. Our definition of $\widehat{\sigma}$ was an instance of copy-cat expansion and it will be superseded by the notion of a *uniform family* of strategies.

Definition 6.90. A *total expanded* strategy is called ***consistent*** just in case it is $\approx_e$-reflexive.

6.4.6 Winning strategies

Definition 6.91. ***Winning IMLAL2 strategies*** are expanded symbolic strategies that are:

1. total,
2. token-reflecting,
3. local,
4. consistent,
5. bounded.

By Theorem 6.87 and Lemma 6.89

Theorem 6.92. Winning IMLAL2 strategies compose.

6.4.7 Uniformity

Static games are representations of IMLAL2 formulas (it suffices to erase the additional square brackets delimiting second-order tokens to reveal the underlying formula). On this understanding we can distinguish between free and bound occurrences of a ground token in a static game. We can also substitute static games for (any) tokens (it is straightforward to see that this operation is well-defined and produces static games). Therefore, each static game with at most n free ground tokens defines an n-ary operation on $\mathcal{S}$ by substitution. Note that the names of tokens are irrelevant for this purpose, so to avoid ambiguities we order the tokens using indices, e.g. let $\mathcal{T} = \{ t_1, t_2, t_3, \cdots \}$. To describe the n-ary operations we shall only use games in which the first n ground tokens may occur. We will write $G(t_1, \cdots, t_n)$ to indicate that.

Definition 6.93. A family of winning IMLAL2 strategies

$$\sigma_{G_1,\cdots,G_n} : G(G_1, \cdots, G_n)$$

for $(G_1, \cdots, G_n) \in \mathcal{S}^n$, which we denote by $\{ \sigma \} : G$, is ***uniform*** if it satisfies the following ***uniformity condition***: suppose $\sigma_{t_1,\cdots,t_n} = \overline{\sigma}_{f,F}$ and $\sigma_{G_1,\cdots,G_n} = \overline{\sigma}_{g,G}$,

- for any $x \in dom\, g \setminus dom\, f$:

$$g(x) = \begin{cases} f(x')y & \text{if } x = x'y \text{ for } x' \in dom\, f \\ f^{-1}(x')y & \text{if } x = x'y \text{ for } x' \in cod\, f \end{cases}$$

- for any $(z, n) \in dom\, G \setminus dom\, F$:

$$G(z) = \begin{cases} (f(z')y, n) & \text{if } z = z'y \text{ for } z' \in dom\, f \\ (f^{-1}(z')y, n) & \text{if } z = z'y \text{ for } z' \in cod\, f. \end{cases}$$

As already discussed after Proposition 6.21 the condition of injective history-freeness and token reflection are implied by uniformity, so they could be omitted from the definition of IMLAL2 winning strategies.

$\sigma_{G_1,\cdots,G_n}$ is thus uniquely developed from $\sigma_{t_1,\cdots,t_n}$ in the same sense as expanded strategies are extensions of symbolic strategies [5]. Therefore, there exists a one-to-one correspondence between uniform families of strategies for G and winning IMLAL2 strategies for the game

$$[\forall t_1.[\forall t_2.[\cdots[\forall t_n.G(t_1,\cdots,t_n)]\cdots]]],$$

which will give rise to the adjunction that is necessary to interpret quantification.

Proposition 6.94. Suppose $\overline{\sigma}_{f,F}$ is a total expanded strategy for $G(t_1,\cdots,t_n)$, and so the pair (f, F) also defines a total symbolic strategy, say, $\sigma'_{f,F}$ for

$$G' = [\forall t_1.[\forall t_2.[\cdots[\forall t_n.G]\cdots]]]$$

such that $\overline{\sigma}'$ is an expanded strategy for G'. Then $\sigma_{f,F}$ generates a uniform family of winning IMLAL2 strategies for G if and only if $\overline{\sigma}'$ is a winning IMLAL2 strategy for G'.

Uniform families of winning IMLAL2 strategies can be composed by pointwise composition of the component strategies. Analogously, we can define their $\approx_e$-equivalence classes and the associated quotient category.

6.4.8 A light affine hyperdoctrine

Let us see now how a light affine hyperdoctrine (see Definition 5.9 for reference) emerges from the framework of evolving games.

[5] It is wrong to think that any winning IMLAL2 strategy $\sigma_{t_1,\cdots,t_n}$ defines a uniform family of winning IMLAL2 strategies. As we have already observed when analyzing IMLAL, the expansions need not be consistent!

Base category

$\mathbb{C}$ is defined as follows.

- Let U be the set $\mathcal{S}$ of static games. The objects of $\mathbb{C}$ are of the shape U^n for $n \in \mathbb{N}$ (the products are calculated in the category of sets and we write 1 for the terminal object U^0).
- Morphisms between U^m and U^n are n-tuples
$$(G_1(t_1, \cdots, t_m), \cdots, G_n(t_1, \cdots, t_m))$$
of static games with free tokens from $\{ t_1, \cdots, t_m \}$.
- Composition works by component-wise substitution for the free tokens.
- The identity morphism on U^n is $(t_1, \cdots, t_n)$.

Indexed light affine category

The hom-functor $\mathbb{C}(\_, U)$ lifts to a functor from $\mathbb{C}^{op}$ to $LACat$:

- $\mathbb{C}(U^n, U)$ is a category whose objects are static games with free ground tokens from $\{ t_1, \cdots, t_n \}$, so $\mathbb{C}(1, U)$ consists of games without any free ground tokens. Given an object $H(t_1, \cdots, t_n)$ we will also write H for the induced n-ary endooperator on $\mathcal{S}$.
- A morphism between two objects H_1 and H_2 is a $\approx_e$-equivalence class of uniform families of strategies for $H_1 \multimap H_2$.

By Theorem 6.53 $\mathbb{C}(U^n, U)$ is a light affine category (the IMLAL game constructions do not introduce new tokens).

For $f = (K_1, \cdots, K_m) : U^n \longrightarrow U^m$ the functor $\mathbb{C}(f, U) : \mathbb{C}(U^m, U) \longrightarrow \mathbb{C}(U^n, U)$ is defined by

- $\mathbb{C}(f, U)(H(t_1, \cdots, t_m)) = H(K_1(t_1, \cdots, t_n), \cdots, K_m(t_1, \cdots, t_n))$ (further we write $H; (K_1, \cdots, K_m)$ for the rhs),
- Given $\{ \sigma \} : H_1 \multimap H_2$ we set
$$\mathbb{C}(f, U)([\{ \sigma \}]_{\approx_e}) = [\{ \tau \}]_{\approx_e} : H_1; (K_1, \cdots, K_m) \multimap H_2; (K_1, \cdots, K_m),$$
where $\tau_{W_1, \cdots, W_n} = \sigma_{K_1;(W_1, \cdots, W_n), \cdots, K_m;(W_1, \cdots, W_n)}$.

$\{ \tau \}$ is a uniform family, because of $\{ \sigma \}$'s uniformity. Besides, $\mathbb{C}(f, U)$ is based on composition, so it (strictly) preserves the structure of light affine categories, and indeed we have $\mathbb{C}(\_, U) : \mathbb{C}^{op} \longrightarrow LACat$.

Quantification

First we define the functor $\forall_n : \mathbb{C}(U^n \times U, U) \longrightarrow \mathbb{C}(U^n, U)$:

- $\forall_n(H(t_1, \cdots, t_n, t_{n+1})) = [\forall t_{n+1}.H(t_1, \cdots, t_n, t_{n+1})]$,
- suppose $\{\,\sigma\,\} : H_1 \multimap H_2$ and $\sigma_{t_1,\cdots,t_n,t_{n+1}} = \overline{\sigma}_{f,F}$,

  $$\forall_n([\{\,\sigma\,\}]_{\approx_e}) = [\{\,\tau\,\}]_{\approx_e} : [\forall t_{n+1}.H_1] \multimap [\forall t_{n+1}.H_2],$$

  where $\tau_{t_1,\cdots,t_n} = \overline{\sigma}_{f,F'}$ for $F' = F \cup \{\,(L, 1) \mapsto (R, 1)\,\}$ [6].

The anticipated correspondence

$$\widetilde{(-)} : \mathbb{C}(U^n \times U, U)(H_1(t_1, \cdots, t_n, t_{n+1}), H_2(t_1, \cdots, t_n, t_{n+1})) \\ \rightleftarrows \\ \mathbb{C}(U^n, U)(H_1(t_1, \cdots, t_n), [\forall t_{n+1}.H_2(t_1, \cdots, t_n, t_{n+1})]) : \widehat{(-)}\ ,$$

is described below.

- Given $[\{\,\sigma\,\}]_{\approx_e} : H_1 \multimap H_2$ such that $\sigma_{t_1,\cdots,t_n,t_{n+1}} = \overline{\sigma}_{f,F}$, let

  $$\widetilde{[\{\,\sigma\,\}]}_{\approx_e} = [\{\,\widetilde{\sigma}\,\}]_{\approx_e}$$

  for $\{\,\widetilde{\sigma}\,\} : H_1 \multimap [\forall t_{n+1}.H_2]$ such that $\widetilde{\sigma}_{t_1,\cdots,t_n} = \overline{\sigma}_{f,F[(R,1)/t_{n+1}]}$. $F[(R, 1)/t_{n+1}]$ is essentially the same as F except that if t_{n+1} occurs in $F(u)$ for some u, we replace it with $(R, 1)$.

- Given $[\{\,\tau\,\}]_{\approx_e} : H_1 \multimap [\forall t_{n+1}.H_2]$ such that $\tau_{t_1,\cdots,t_n} = \overline{\sigma}_{g,G}$, let

  $$\widehat{[\{\,\tau\,\}]}_{\approx_e} = [\{\,\widehat{\tau}\,\}]_{\approx_e}$$

  for $\{\,\widehat{\tau}\,\} : H_1 \multimap H_2$ such that $\widehat{\tau}_{t_1,\cdots,t_n} = \overline{\sigma}_{g,G[t_{n+1}/(R,1)]}$. $G[t_{n+1}/(R, 1)]$ is defined dually to $F[(R, 1)/t_{n+1}]$.

The definition is correct as the correspondence almost amounts to an identity (see previous footnote). Hence, it is bijective and suitably natural. It also satisfies the Beck-Chevalley condition, because:

- the functorial diagram commutes for objects: we have

  $$[\forall t_{n+1}.G]; (G_1, \cdots, G_n) = [\forall t_{n+1}.G; (G_1, \cdots, G_n)]$$

  in $\mathbb{C}$, for composition in $\mathbb{C}$ is substitution,

[6] Here we barely modify the original strategy, so 'definition by $\tau_{t_1,\cdots,t_n}$' is correct: we are guaranteed that a family $\{\,\tau\,\}$ of IMLAL2 winning strategies will arise.

- the diagram also commutes for morphisms, since for $f = (G_1, \cdots, G_n)$ both $(\mathbb{C}(f, U); \forall_n)([\{\sigma\}]_{\approx_e})$ and $(\forall_m; \mathbb{C}(f \times t_{n+1}, U))([\{\sigma\}]_{\approx_e})$ are defined by copycat expansions of $\{\sigma\}$ for the same games,
- the strategy

$$\widetilde{\mathbb{C}(f \times t_{n+1}, U)}(\widehat{[\mathsf{id}_{\forall_n(H)}]_{\approx_e}})$$

is an identity, since $\mathsf{id}_{\forall_n(H)}$ is a family of identity strategies which $\widehat{(-)}$ modifies only symbolically (recall $[t_{n+1}/(R, 1)]$) and $\mathbb{C}(f \times t_{n+1}, U)$ then converts to another family of 'almost' identity strategies (up to the slight change caused by $\widehat{(-)}$) in which $\widetilde{(-)}$ reverses the cosmetic changes effected by the initial $\widehat{(-)}$.

Theorem 6.95. The framework specified above is a model IMLAL2. $\mathbb{C}(1, U)$ consists of static games without free tokens and winning IMLAL2 strategies.

6.4.9 Full completeness

For any strategy IMLAL2 winning strategy ρ, let ρ_0 be the 'substrategy' in which O can only open threads with indices ending in 0 or $\star$. The definition of uniform families of strategies (which must be $\approx_e$-reflexive) ensures that σ_0 contains complete information about the networks in ρ i.e. !-networks whose O-threads have an index that does not end with 0 are simply 'clones' of the networks where 0 was used. Besides, families in the same equivalence class have networks with identical structure. Next we show that ρ_0 can be specified using a finite amount of data under a certain condition.

Proposition 6.96. If $\sigma = \overline{\sigma}_{f,F}$ satisfies 3. of Definition 6.84 and is compactly networked, then σ_0 is finitely generated. In particular, if σ is bounded, then σ_0 is finitely generated.

Proof. It suffices to consider positions of σ_0 in which O plays symbolically and show that only a finite number of positions from $\sigma_{f,F}$ will be explored.

By 3. there is a bound on the size of the game during play. Hence, the depth of moves is also bounded. Let d be the bound.

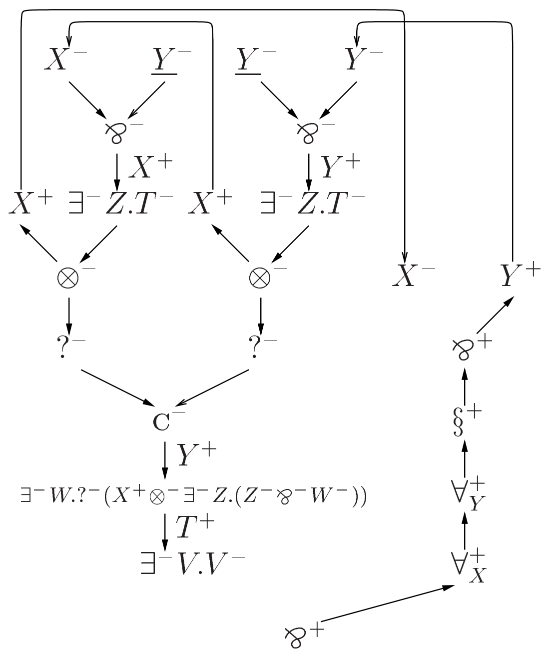
Since the size of the game is bounded and each network has a finite number of threads, there will be a finite number of O-threads that can be opened at each depth (recall that their indices must end in 0 or $\star$!). Because the deepest network is at d, the network function of the symbolic substrategy of σ_0 must be finite. As the size of the game is bounded, only a finite number of moves can be made, so (abusing notation) we can write $\sigma_0 = \overline{\sigma}_{f_0,F_0}$, where f_0, F_0 are finite. $\square$

From now on we assume that σ is a winning IMLAL2 strategy for a static game G and consider a subset of σ_0 in which O must also play short-sightedly. Following the notation for IMLAL we denote it by σ^{f_0,F_0}. We are going to use σ^{f_0,F_0} to define a canonical essential net. Let us assume that ξ is an injective assignment of second-order variables to elements of $L_0 \times \mathbb{N}$. First we construct a tree $\mathcal{T}_{F_0}$ which will to converted to the net $\mathcal{N}_{f_0,F_0}$ in the next step.

- We start from the syntactic tree of $\ulcorner G \urcorner$ (second-order tokens are treated like ordinary tokens, so they will be leaves e.g. $[\forall a.a]^+$, $[\forall b.b \otimes a]^-$). Let us orient its branches according to the directional rules for essential nets: premises of nodes in essential nets become children of nodes in the tree. Then we apply the same procedure as for IMLAL to introduce contraction nodes at each level, trying to make the two steps below for tokens $[\forall a.G]^+$ at the respective depths as soon as possible.
  - Tokens of the shape $[\forall a.G]^+$—as soon as they appear as leaves—should be replaced with the oriented syntactic tree of $\ulcorner \forall \xi(l,n).G[\xi(l,n)/a] \urcorner$. (l,n) is the complete O-location at which the token occurs, which we can read off the tree, once labels l, r, L, R are associated with $\otimes^+$-,$\otimes^-$-,$\parr^+$-,$\parr^-$-nodes, numeric labels with incoming edges of $?^-$-nodes, 0's with $!^+$-nodes and $\star$'s with §-nodes. The node corresponding to the quantifier is to be labelled with $\forall^+_{\xi(l,n)}$ and we give an auxiliary label n (which will not be part of the final net) to the outgoing edge to keep track of the numerical components of complete locations: if the token occurs at the end of a branch then
    - if it is the premise of a $\forall^+$-node whose outgoing edge is labelled with n, its auxiliary label will be $n+1$,
    - otherwise it is 0.
  - Tokens of the shape $[\forall a.G]^-$, provided their complete location (calculated as in the previous case) is $(l,n) \in \mathit{dom}\, F_0$, should be tackled in the following way (otherwise they remain as they are). Suppose $F_0(l,n) = H$. Let H' be H in which every complete O-location (l',n') is replaced with $\xi(l',n')$. The token should be replaced with an $\exists^-$-link whose conclusion is labelled with $\exists^- \xi(l,n).\llcorner G \lrcorner$ and whose eigentype is $\ulcorner H' \urcorner$.

Because F_0 is finite, the construction terminates. $\mathcal{N}_{f_0,F_0}$ is obtained by adding axiom links as specified by f_0. Finally, the unreachable nodes which are premises of reachable ones are contracted to weakening nodes as before. Note that this will eliminate the $[\forall a.G]^-$ tokens that were not unfolded. The net corresponding to τ from Example 6.70 is presented in Figure 6.4.

Theorem 6.97. $\mathcal{N}_{f_0,F_0}$ is a correct canonical affine net.



$$
\begin{aligned}
T^+ &= \forall^+ W.!^+(X^- \parr^+ \forall^+ Z.Z^+ \otimes^+ W^+)\\
T^- &= Z^- \parr^- Y^-
\end{aligned}
$$

Figure 6.4: $\mathcal{N}_{f_0,F_0}$ developed from the strategy in Example 6.70.

Proof. Like for the other logics our proof relies on a correspondence between short-sighted positions and paths from the root ending in atomic nodes. The relationship is established in the same way as before, though there are new complications due to second-order tokens. For the quantifier-free fragments a move with a ground token consisted of letters indicating a branch of the syntactic tree ending with the suitable atom. This main idea remains the same, but when the branch pointing at an atom involves a $\forall^+_X$-node, some ground token will have to played for the first time the node is involved. Later the token is unnecessary, as the corresponding game evolution has already taken place. The same applies to the $\forall^- Z.A^-$-nodes, but the evolution argument that is played will correspond to the associated eigentype.

Conditions (i)-(v) can then be verified by the same reasoning as for IMLAL. To check (vi), suppose there exists a $\forall^- Z.A^-$-node with eigentype T such that some $X \in \mathrm{FV}(T)$ is an eigenvariable and there is a path from the root to the $\forall^- Z.A^-$-node that does not visit the $\forall^+_X$-node. Consider the last atomic/variable node which is visited by the path and the corresponding short-sighted position. As the path does not pass through the $\forall^+_X$-node, P will be required to use an argument that should have been provided by O! Hence, the strategy is not total, which contradicts our assumption.

Next suppose there exists a malformed path from some $\forall^+_X$-node to a $\forall^- Z.A^-$-

node whose eigentype contains X. By (iv) all paths in the net are well-formed, so the malformed path consists of a well-bracketed sequence followed by a path in which the next bracketing node is a closing one. This implies that in the associated position an argument introduced by O in some network will not be used inside it, so the strategy is not local, contrarily to our assumption. $\square$

Theorem 6.98. The light affine hyperdoctrine of static games and $\approx_e$-equivalence classes of uniform families of winning IMLAL2 strategies is a fully and faithfully complete model of IMLAL2 with respect to canonical affine nets.

By Theorem 2.7 we obtain the main result of the thesis—a game model that captures polynomial-time computability.

Theorem 6.99. Winning IMLAL2 strategies for the games

$$[\forall b.!(b \multimap b) \multimap (!(b \multimap b) \multimap \S(b \multimap b))] \multimap \\ \S^n\,[\forall b.!(b \multimap b) \multimap (!(b \multimap b) \multimap \S(b \multimap b))]$$

where n ranges over natural numbers define precisely the FP functions.

Chapter 7

Future work

7.1 Semantics

Fully and faithfully complete models of Light Affine Logic inherit its polynomial-time character and enable polynomial-time computability to be investigated semantically. Hence, they constitute an ideal setting in which original abstract properties related to computational complexity could be uncovered. In the long run such an approach should bring about novel criteria for expressibility, permitting one to certify that certain algorithms are not programmable in a given programming language interpretable in the model (in the spirit of [28]) or, possibly even more strongly, that certain functions are not representable and thus do not belong to the associated complexity class. Our model should also be employed to give another proof of the fact that (untyped) reduction in IMLAL2 can be implemented in polynomial time. This will become possible, if various bounds in our proofs are estimated more accurately. Another obstacle towards this goal is the validation of η-conversion in the current framework coupled with the way strategies are represented: the history-free function corresponds to the η-expanded form and therefore one may need exponential time to convert the representations of two strategies into that of the composite strategy. Hence, a more economical representation of strategies is necessary to overcome the problems. The findings reported in [75, 73] are of relevance to this direction, which would essentially entail a construction of a game model for the underlying untyped calculus of light affine logic. If various complexities could be read off strategies in such a model it would be a first step towards semantic-based complexity inference.

There are also some syntactic issues that could be resolved by game semantics. For example, it is not known whether the poly-time complexity is preserved once the $\S$ functor becomes strictly monoidal. This strictness creates additional opportunities for duplication [4] and facilitates more flexible coding, but it is not clear how the corresponding syntax should look like. It seems that the relaxation of the global matching requirement in essential nets could produce an adequate proof-net syntax.

7.2 Syntax

Our work has unveiled the largest sublanguage BC^{-} of BC that admits a translation into LAL. This poses the question whether BC^{-} itself can be used to define all FP functions (probably not) and if not what complexity class it actually represents. There are two dual directions related to that point that should be pursued. Firstly, it is important that programming languages capturing complexity classes allow as many natural definitions as possible. Typically, many natural constructions will be instances of techniques that otherwise lead to superpolynomial runtime. A degree of flexibility could be attained by introducing higher-order functions as first-class values [55]. For instance, $BC^{\pm}$ enriched with higher-order recursion schemes would make a relatively convenient FP programming language. The appropriate methodology preventing us from admitting too generous a construction is vital for that purpose and representability in LAL (or its FP-sound extensions) could be adopted as such. The major advantage of such an approach is the promise of automatic complexity inference, as the two fundamental theorems on LAL relate the type of a program with the degree of the polynomial that bounds the complexity. Since we can translate various programming constructions into LAL, we should find ourselves in a good position to estimate their effect on complexity. Secondly, it is worth investigating whether LAL can be simplified in a way that preserves its two essential properties. For instance, it may be the case that full polymorphism is redundant and it may be replaced with a weaker form (e.g. ML type schemes).

A long-term objective is to obtain a hierarchy of logics and game models for other complexity classes, especially LOGSPACE, NP and PSPACE, and to conduct a comparative study. This could serve as a starting point to develop experimental programming languages for resource-bounded computation tailored to the expected time or space complexity.

Bibliography

[1] S. Abramsky. Semantics of interaction. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 1–32. Cambridge University Press, Cambridge, 1997.

[2] S. Abramsky, K. Honda, and G. McCusker. Fully abstract game semantics for general reference. In *Proceedings of IEEE Symposium on Logic in Computer Science, 1998*. Computer Society Press, 1998.

[3] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[4] S. Abramsky, R. Jagadeesan, and M. Hofmann. Safe variables in light logics. private communication, 2000.

[5] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In *Theoretical Aspects of Computer Software: TACS'94, Sendai, Japan*, pages 1–15. Springer-Verlag, 1994. LNCS Vol. 789.

[6] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[7] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*. Birkhaüser, 1997.

[8] S. Abramsky and G. McCusker. Call-by-value games. In M. Nielsen and W. Thomas, editors, *Proceedings of Computer Science Logic '97*. Springer-Verlag, 1998. Lecture Notes in Computer Science.

[9] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Computer Science*, 227:3–42, 1999.

[10] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *Proceedings, Fourteenth IEEE Symposium on Logic in Computer Science*, pages 431–442. IEEE Computer Society Press, 1999.

[11] A. Asperti. Light affine logic. In *Proceedings of 13th IEEE Annual Symposium on Logic in Computer Science 1998*. IEEE Computer Society, 1998.

[12] A. Asperti and L. Roversi. Intuitionistic light affine logic (proof-nets, normalization complexity, expressive power). To appear in ACM Transactions on Computational Logic, 2001.

[13] P. Baillot. Stratified coherent spaces: a denotational semantics for light linear logic (extended abstract). In *(LICS affiliated) Second International Workshop on Implicit Computational Complexity*, 2000.

[14] P. Baillot, V. Danos, T. Erhard, and L. Regnier. AJM's games model is a model of classical linear logic. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1997.

[15] P. Baillot, V. Danos, T. Erhard, and L. Regnier. Timeless games. In *Proceedings of CSL'97, 11th Annual Conference of the European Association of Computer Science Logic*, volume 1414 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[16] S. Bellantoni and S. A. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[17] G. Bellin. Proof nets for multiplicative and additive linear logic. Technical Report LFCS-91-161, LFCS, Division of Informatics, University of Edinburgh, 1991.

[18] G. Bellin. Proof nets for classical MLL and linear lambda terms. A March 1992 posting to the LINEAR mailing list, 1992.

[19] P. N. Benton, G. M. Bierman, V. C. V. de Paiva, and J. M. E. Hyland. Term assignment for intuitionistic linear logic. Technical Report Technical Report 262, Computer Laboratory, University of Cambridge, 1992.

[20] G. M. Bierman. What is a categorical model of intuitionistic linear logic. In *Proc. International Conference on Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, 1995.

[21] A. Blumensath. Bounded arithmetic and descriptive complexity. In *Proceedings of CSL'00*, volume 1862 of *Lecture Notes in Computer Science*, pages 232–246. Springer-Verlag, 2000.

[22] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A*. Elsevier, 1990.

[23] G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Complexity classes and rewrite systems with polynomial interpretation. In *Proceedings of the Twelfth Annual Conference of the European Association for Computer Science Logic, CSL'98*, volume 1584 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[24] S. Buss. *Bounded Arithmetic*. Bibliopolis, 1986.

[25] V.-H. Caseiro. Equations for defining poly-time functions. PhD thesis, University of Oslo, Department of Informatics, 1997.

[26] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[27] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Sciences*, pages 24–30. North-Holland, 1964.

[28] Loic Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83:57–69, 1991.

[29] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the Association for Computing Machinery*, 18(1):4–18, 1971.

[30] S. A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97, 1975.

[31] R. L. Crole. *Categories for Types*. Cambridge University Press, 1993.

[32] V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. IEEE Symposium on Logic in Computer Science*. Computer Science Society, 2000.

[33] V. Danos and J.-B. Joinet. Linear logic and elementary time. *Information and Computation*, 2001. to appear, presented at ICC'99.

[34] V. Danos and L. Regnier. The structures of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[35] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1987.

[36] H. Devarajan, D. J. D. Hughes, G. D. Plotkin, and V. R. Pratt. Full completeness of the multiplicative linear logic of Chu spaces. In *Proceedings of LICS'99*, pages 234–242. IEEE Computer Society Press, 1999.

[37] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–73. SIAM-ACM Press, 1974.

[38] Dan R. Ghica and Guy McCusker. Reasoning about idealized algol using regular expressions. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 103–115. Springer-Verlag, 2000.

[39] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[40] J.-Y. Girard. Quantifiers in linear logic II. Preprint, Équipe de Logique de Paris VII, 1991.

[41] J.-Y. Girard. Proof-nets : the parallel syntax for proof-theory. In *Logic and Algebra*. Marcel Dekker, New York, 1996.

[42] J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.

[43] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989. Cambridge Tracts in Theoretical Computer Science 7.

[44] J.-Y. Girard, A. Scedrov, and P. J. Scott. Bounded linear logic. *Theoretical Computer Science*, 97:1–66, 1992.

[45] A. Goerdt. Characterizing complexity classes by higher type primitive recursive definitions. *Theoretical Computer Science*, 100:45–66, 1992.

[46] E. Grädel. The expressive power of second order Horn logic. In *Proceedings of the 8th Symposium on Theoretical Aspects of Computer Science STACS 91*, volume 480 of *LNCS*, pages 466–477. Springer-Verlag, 1991.

[47] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[48] Y. Gurevich. Algebras of feasible functions. In *Proceedings of thr 24th Annual Symposium on Foundations of Computer Science FOCS'93*, pages 210–214, 1983.

[49] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[50] W. G. Handley. Bellantoni and Cook's characterization of polynomial time with some related results. In S. Wainer's Marktoberdorf'97 Lecture Notes, 1997.

[51] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings of Fourteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1999.

[52] M. A. Harrison and O. H. Ibarra. Multi-tape and multi-head pushdown automata. *Information and Control*, 13:433–470, 1968.

[53] G. G. Hillebrand, P. C. Kanellakis, and H. G. Mairson. Database query languages embedded in the typed lambda calculus. *Information and Computation*, 127(2):117–144, 1996.

[54] M. Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. A talk presented at CSL'97, Aarhus, Denmark, 1997.

[55] M. Hofmann. Type systems for polynomial-time computation. Habilitationsschrift, Technische Universität Darmstadt, 1998.

[56] M. Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proceedings of 14th IEEE Annual Symposium on Logic in Computer Science*. IEEE Computer Society, 1999.

[57] M. Hofmann. Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 2000. To appear. Ftp-able from Hofmann's home page.

[58] M. Hofmann. The strength of non-size increasing computation. Available from the author's webpage, 2001.

[59] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation (extended abstract). In *Proc. of ICALP'97, Borogna, Italy, July, 1997*. Springer-Verlag, 1997. LNCS.

[60] D. H. D. Hughes. Games and definability for System F. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science*. IEEE Computer Science Society, 1997.

[61] D. H. D. Hughes. *Games and full completeness for System F.* PhD thesis, University of Oxford, 2000.

[62] J. M. E. Hyland. Game semantics. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 131–182. Cambridge Univ. Press, 1997.

[63] J. M. E. Hyland and C.-H. L. Ong. Fair games and full completeness for multiplicative linear logic without the MIX-rule. preprint, 1993.

[64] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163(2):285–408, 2000.

[65] J. M. E. Hyland and A. M. Pitts. The theory of constructions: Categorical semantics and topos-theoretic models. *Contemporary Mathematics*, 92:137–199, 1989.

[66] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[67] N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer-Verlag, 1998.

[68] B.P.F. Jacobs. *Categorical Type Theory.* PhD thesis, Nijmegen University, 1991.

[69] N. D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[70] N. D. Jones. The expressive power of higher-order types, or life without CONS. *Journal of Functional Programming*, 2000. submitted for publication.

[71] M. I. Kanovich, M. Okada, and A. Scedrov. Phase semantics for light linear logic. *Electronic Notes in Theoretical Computer Science*, 6, 1997. Proceedings of the 13th Annual Conference on Mathematical Foundations of Programming Semantics, Pittsburgh, Pennsylvania, March 1997.

[72] G. M. Kelly and S. MacLane. Coherence in closed categories. *Journal of Pure and Applied Algebra*, 1:97–140, 1971.

[73] A. D. Ker. *Innocent game models of untyped lambda calculus.* PhD thesis, University of Oxford, 2001.

[74] A. D. Ker, H. Nickau, and C.-H. L. Ong. A universal innocent game model for the Böhm tree lambda theory. In *Proceedings of CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, pages 405–419. Springer-Verlag, 1999.

[75] A. D. Ker, H. Nickau, and C.-H. L. Ong. Innocent game models of untyped λ-calculus. *Theoretical Computer Science*, 2000. 44 pages. To appear.

[76] T. W. Koh and C.-H. L. Ong. Explicit substitution internal languages for autonomous and $*$-autonomous categories. *Electronic Notes in Theoretical Computer Science*, 29, 1999. Proceedings of the 8th Conf. on Category Theory and Computer Science 1999, 30 pp.

[77] Y. Lafont. Soft linear logic and polynomial time. Available from the author's homepage at `http://iml.univ-mrs.fr/`, 2001.

[78] J. Laird. *A semantic analysis of control.* PhD thesis, University of Edinburgh, 1998.

[79] J. Laird. A fully abstract games semantics of local exceptions. In *Proceedings of 16th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2001.

[80] F. Lamarche. Proof nets for intuitionistic linear logic 1: Essential nets. Preprint ftp-able from *Hypatia*, 1994.

[81] D. Leivant. Stratified functional programs and computational complexity. In *Proc. 20th ACM POPL*, pages 325–333. ACM Press, 1993.

[82] D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 110:391–420, 1994.

[83] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, 1993.

[84] R. Loader. Linear logic, totality and full completeness. In *Proceedings of 9th* IEEE *Symoposium on Logic in Computer Science, Paris, July 1994*, pages 292–298. IEEE Computer Science Society Press, 1994.

[85] R. Loader. *Models of Lambda Calculi and Linear Logic: structural, equational and proof-theoretic characterisations*. PhD thesis, University of Oxford, 1994.

[86] I. Mackie, L. Román, and S. Abramsky. An internal language for autonomous categories. *Journal of Applied Categorical Structures*, 1(3):311–343, 1993.

[87] S. MacLane. *Categories for the Working Mathematician*. Spinger-Verlag, 1971. Graduate Text in Mathematics 5.

[88] P. Malacaria and C. Hankin. Generalized flowcharts and games (extended abstract). In *Proceedings of ICALP 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 363–374. Springer-Verlag, 1998.

[89] P. Malacaria and C. Hankin. A new approach to control flow analysis. In *Proceedings of CC 1998*, volume 1383 of *Lecture Notes in Computer Science*, pages 95–108. Springer-Verlag, 1998.

[90] P. Malacaria and C. Hankin. Non-deterministic games and program analysis: an application to security. In *Proceedings, Fourteenth IEEE Symposium on Logic in Computer Science*, pages 443–452. IEEE Computer Society Press, 1999.

[91] P. Malacaria and L. Régnier. Some results on the interpretation of the λ-calculus in operator algebras. In *Proceedings of LICS'91*. IEEE Computer Society Press, 1991.

[92] J.-Y. Marion. Analysing the implicit complexity of programs. *Information and Computation*, 2001. to appear, presented at ICC'99.

[93] G. McCusker. *Games for recursive types*. BCS Distinguished Dissertation. Cambridge University Press, 1998.

[94] A. A. Muchnik. On two approaches to the classification of recursive functions. In V. A. Kozmidiadi and A. A. Muchnik, editors, *Problems in Mathematical Logic, Complexity of Algorithms and Classes of Computable Functions*, pages 123–128. Mir, Moscow, 1970.

[95] A. S. Murawski and C.-H. L. Ong. Exhausting Strategies, Joker Games and Full Completeness for IMLL with Unit. *Electronic Notes in Theoretical Computer Science*, 29, 1999. Proceedings of the 8th Conf. on Category Theory and Computer Science 1999, 31 pp.

[96] A. S. Murawski and C.-H. L. Ong. Can safe recursion be interpreted in light logic? In *(LICS affiliated) Second International Workshop on Implicit Computational Complexity*, 2000. 20 pages.

[97] A. S. Murawski and C.-H. L. Ong. Discreet Games, Light Affine Logic and PTIME Computation. In *Proceedings of CSL'00*, volume 1862 of *LNCS*, pages 427–441. Springer-Verlag, 2000.

[98] A. S. Murawski and C.-H. L. Ong. Dominator trees and fast verification of proof nets. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science*, pages 181–191. IEEE Computer Society Press, 2000.

[99] A. S. Murawski and C.-H. L. Ong. SLR^{-} is PTIME complete. Unpublished note. Ftp-able from Ong's home page, 2000.

[100] A. S. Murawski and C.-H. L. Ong. Evolving games and affine polymorphism. In *Proceedings of TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 360–375. Springer-Verlag, 2001.

[101] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium of Logical Foundations of Computer Science*. Springer-Verlag, 1994. LNCS.

[102] H. Nickau and C.-H. L. Ong. Games for System F. Working paper, 1999.

[103] C. H. Papadimitriou. A note on the expressive power of PROLOG. *EATCS Bulletin*, 26:21–23, 1985.

[104] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[105] M. Pedicini. Remarks on elementary linear logic. *Electronic Notes in Theoretical Computer Science*, 3, 1996.

[106] A. M. Pitts. Polymorphism is set-theoretical, constructively. In D. H. Pitt *et al.*, editor, *Proc. Conf. Category Theory and Computer Science, Edinburgh*, Berlin, 1987. Springer-Verlag. LNCS. Vol. 287.

[107] H. E. Rose. *Sub-recursion: functions and hierarchy.* Clarendon Press, Oxford, 1984.

[108] L. Roversi. A polymorphic language which is typable and poly-step. In *Proceedings of the Asian Computing Science Conference (ASIAN'98)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[109] L. Roversi. A PTIME completeness proof for light logics. In *Proceedings of CSL'99*, volume 1683 of *Lecture Notes in Computer Science.* Springer-Verlag, 1999.

[110] L. Roversi. Light affine logic as a programming language: a first contribution. *International Journal of Foundations of Computer Science*, 11(1):113 – 152, March 2000.

[111] V. Y. Sazonov. Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung and Kybernetik*, 16:319–323, 1980.

[112] R. A. G. Seely. Categorical semantics for higher order polymorphic lambda calculus. *Journal of Symbolic Logic*, 52:969–989, 1987.

[113] K. Terui. Light affine lambda calculus and poly-time strong normalization (extended abstract). In *Proceedings of 16th IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 2001.

[114] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing STOC82*, pages 137–146, 1982.

Index

$LACat$, 112
L_0, 160
L_P, 160
S_2^1, 14
$\mathcal{B}^-$, 70
$\mathcal{B}^+$, 70
$\mathcal{E}$, 78
$\mathcal{N}_{\mathrm{IMLL}}$, 70
δ, 71
δ_p, 71
$\mathcal{L}^*$, 7
$\nu(n)$, 85
$\phi(n)$, 59
$\sigma_{f,F}$, 164
$\approx_G$, 144
$\approx_n$, 147
$\approx_e$, 174
$\llcorner A \lrcorner$, 55
$\ulcorner A \urcorner$, 55
3-COLOURABILITY, 11

affine nets, 87
  collected, 92
  correctness, 88
ALOGSPACE, 9
alternating computation, 9
alternating Turing machine, 9
atomic nodes, 57

base, 114
base of a move, 132
BC, 16, 33
BC^-, 33–37
BC^- + **case** + **e-shift**, 48–52
$BC^\pm$, 40–47
Beck-Chevalley condition, 114
bookkeeping maps, 103
bounded
  arithmetic, 14
  logics, 12
  recursion, 14

canonical link of a node, 58
$\mathbf{case}_K(: u)$, 38
category
  autonomous, 98
  light affine, 112
  symmetric monoidal, 98
  symmetric monoidal closed, 98
coercions, 31
commutative comonoid, 110
commuting conversions, 23
completeness
  full, 116
  full and faithful, 116
composite, 120
comprehension scheme, 14
concatenation
  $concat(x, y)$, 34
  $\mathbf{concat}(x : y)$, 34
  $\mathbf{concat}(: x, y \mid 0)$, 36
  $\mathbf{concat}(: x, y)$, 40
conclusion
  of a link, 58
  of a net, 58
π-congruence, 24
σ-congruence, 24
cons-free programs, 13
contraction, 18
  link, 70
Cook's Thesis, 7

critical arguments, 16

database queries, 12
depth
  of a formula, 24
  of a game, 133
  of a move, 132
dereliction rule, 18
descriptive complexity, 11
domination ordering, 62
dominator tree of a graph, 62
duplicable objects, 18

e-shift(: n), 47
eigentype
  of a $\Diamond^-$-link, 85
  of a $\exists^-$-link, 78
eigenvariable
  of a $\forall^+$-link, 78
  reachable from a link, 81
eliminable link
  IMLL, 61
empire of a node, 63
enabling relation, 124
equivalence
  of positions, 144
  of strategies, 145, 147
essential nets, 54–97
  correctness
    IMLL, 59
    IMLLL, 71
    IMLLL2, 79
  for IMLL, 57
  for IMLLL, 70
  for IMLLL2, 78
evolution argument, 159
existential second-order logic (SO$\exists$), 11
existential states, 9
expanded strategy, 165
extension of an affine net, 89

first-order logic (FO), 12
FO+IFP, 12
FO+LFP, 12
FPTIME (FP), 7–29
function interpretable in IMLAL2, 36
functor
  symmetric monoidal, 104
FV(Γ), 19

game, 117
  atomic, 118
  empty, 118
  generalized static, 157
  IMAL, 121
  IMLAL, 132
  IMLAL2, 162
  linear arrow, 118
  linear function space, 118
  over token set, 122
  static, 155
  tensor, 118

hereditary premise of a link, 58

IMAL2, 22
IMLAL, 22
IMLAL2, 20, 87–97
IMLL, 55–69
IMLL2, 22
IMLLL, 22, 69–77
IMLLL2, 22, 77–87
index of a move, 132
infinite chattering, 143
inflationary fixed points (IFP), 12
injective history-free, 122
interaction sequence, 120
  of threads, 139
interactive tree of a thread, 142
interface, 22
interim net of a node, 68
iteration principle, 30

justification relation, 124

least fixed points (LFP), 11

lifting principle, 32
light affine hyperdoctrine, 114
light affine logic (LAL), 18–27
linear logic, 18
- soft (SLL), 28
- bounded (BLL), 18
- elementary (ELL), 28
- light (LLL), 18

linear safe recursion, 33
links
- $?^-$, 70
- $\Diamond^-$, 84
- $\exists^-$, 78
- $\forall^+$, 78
- $⅋^-$, 58
- $⅋^+$, 58
- $!^+$, 70
- $\otimes^-$, 58
- $\otimes^+$, 58
- $\S^+$, 70
- $\S^-$, 70
- contraction, 70
- weakening, 87

location, 160
- complete, 160
- O-, 160
- P-, 160

LOGSPACE, 10, 13

model
- fully and faithfully complete, 116
- fully complete, 116

move
- in static games, 161
- O-, 117
- P-, 117
- short-sighted, 124
- symbolic, 163

multi-head two-way automata, 10

negative formula, 56
negative polarized form, 55
network, 134, 137
- !-, 137
- creation, 137

network protocol, 137
networking of a position, 138
NLOGSPACE, 9, 10
nodes
- $?^-$, 70
- $\Diamond_V^-$, 84
- $\exists^- X.A^-$, 78
- $\forall_X^+$, 78
- $⅋^-$, 58
- $⅋^+$, 58
- $!^+$, 70
- $\otimes^-$, 58
- $\otimes^+$, 58
- $\S^+$, 70
- $\S^-$, 70
- contraction, 70
- weakening, 87

non-deterministic Turing machines, 9
non-size-increasing computation, 39
normal
- arguments, 15
- contractibility, 32
- variables, 32

NPTIME (NP), 9, 11

order of a type, 13
ordered dominator tree, 65

P-completeness, 7
P=NP problem, 11
perm$_L$(: u), 38
perm-rec, 38
play (position), 117
playable string, 159
- O-, 160
- P-, 160

polarized
- atoms, 55
- connectives, 55
- nodes, 58

position

in static games, 161
short-sighted, 124
position (play), 117
positive formula, 56
positive polarized form, 55
premise of a link, 58
PROLOG, 14
promotion rule, 18
protonets, 85
correctness, 86
healthiness conditions, 85
PTIME (P), 7–29
pushdown tape, 10
PV, 14

QPTIME, 12

recursion on safe arguments, 38
redex, 25
at depth i, 26
β, 25
exponential, 25
garbage collection, 25
replication, 25
reindexing functor, 113
related !-threads, 136
root of a net, 58

safe
arguments, 15
composition, 16
recursion, 16
variables, 32
second-order logic (SO), 11
sequentializability, 61
sequentializable essential net, 61
sequentialization of an essential net, 62
sink of a $\otimes^+$-node, 58
size of a game, 168
SO∃-Horn, 11
SO-Horn, 11
space complexity, 9
strategy, 119
$\approx$-reflexive, 145
$\approx_n$-reflexive, 147
0-winning, 149
bounded, 169
compactly networked, 142
consistent, 147, 174
deterministic, 119
history-free, 122
identity, 120
local, 167
local expanded, 167
networked, 139
short-sighted, 125
suitably networked, 143
symbolic, 163
token-reflecting, 122
total, 122
weakly winning (IMAL), 125
winning (IMAL), 123
winning (IMLAL), 149
winning (IMLAL2), 174
stratification, 15
strengthening of a weakening link, 88
strip, 31
switching condition
for threads, 137
O, 118
P, 118
symbolic play, 163

tensor unit, 98
terminal link, 58
thread
!-, 133
§-, 133
at depth i, 133
O-, 133
P-, 133
thread function, 138
tiering, 15
token
ground, 155

of a move, 121
Turing machine, 8

uniform family of strategies, 175
uniformity condition, 175
universal states, 9

weakening rule, 18
well-formed path, 71