Labelled Sequent Calculi and Automated Reasoning for Assertions in Separation Logic

HÓU, Zhé

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

August 2020

Except where otherwise indicated, this thesis is my own original work.

HÓU, Zhé
7 August 2020

To my parents and grandma.

Acknowledgments

I owe special thanks to my supervisors Rajeev Goré, Alwen Tiu, and Ranald Clouston, without whom this dissertation would not exist. Alwen and Ranald, I miss the days when I could just walk into your offices and discuss all sorts of problems with you; your ideas, advice, and technical feedbacks are invaluable to me. Raj, I am forever grateful for your supervision, your attention to details and care for students have changed me, I could not have found a better supervisor. I would also like to thank my advisers Peter Baumgartner, John Slaney and Dirk Pattinson for their help on various problems. And of course, thank John Lloyd for granting me the scholarship and the travel grant that made this Ph.D. possible at the first place!

I am very lucky as a student. In every stage of my student life, the best student around me always happens to be my best friend. In chronological order, JIA Chao, JIANG Zhe, DAI Huichen, and SHAO Wen, I was molded by you, motivated by you, I could not have reached this far if you had not come into my life. Thank YANG Di and Jimmy Thomson for sharing the office with me, helping me, and chatting with me in the past three years; thank RONG Sheng for remotely electronically entertaining with me in the past years; thank Asim Azam for being the funniest guy ever; and thank Bryce Kositz for helping me bind this thesis!

To James Brotherston, thank you for the email exchanges, discussions, and the arrangement for my visit in London. To Sungwoo Park, thank you for all the technical discussions, too.

Thank all the reviewers for the insightful, constructive, and detailed comments.

Last but absolutely not least, I would like to sincerely thank my parents HOU Keyi and DENG Ji who never cease supporting me and loving me. Thank my grandma LIU Cuiying for bringing me up, and thank all other family members for missing me all the time. I would also like to extend my gratitude to ZHANG Yaqing for giving me much love and having fun with me in the irrational part of my life.

Abstract

Separation logic (SL) is an extension of Hoare logic to reason about programs with mutable data structures. This dissertation studies automated reasoning for the assertional language (i.e., formulae that represent pre- and post-conditions) of separation logic. We start from the core of separation logic called Boolean BI (BBI), then consider various propositional abstract separation logics (PASLs) as extensions of BBI. These logics are "abstract" because they are independent of any particular concrete memory model. Finally, we look at separation logic with Reynolds's heap model semantics.

We first try to capture BI logics using nested sequents. Our design has a special structure of interleaving additive and multiplicative nested sequents. We discover that this nested structure causes much complication when considering the rules for transferring information between nested sequents. Then we change the angle to solve this problem by using labelled sequents.

We present a labelled sequent calculus LS_{BBI} for BBI. The calculus is simple, sound, complete, and enjoys cut-elimination. We show that all the structural rules in LS_{BBI} can be localised around applications of certain logical rules, based on which we demonstrate a free variable calculus that deals with the structural rules lazily in a constraint system. We propose a heuristic method to quickly solve certain constraints, and show some experimental results to confirm that our approach is feasible for proof search.

Based on LS_{BBI}, we develop a modular proof theory for various PASLs using cut-free labelled sequent calculi. We first extend LS_{BBI} to handle Calcagno et al.'s original logic of separation algebras by adding rules for partial-determinism and cancellativity. We prove the completeness of our calculus via an intermediate calculus that enables us to construct counter-models from the failure to find a proof. We then capture other PASLs by adding rules for other properties in separation theories. We present a theorem prover based on our labelled calculi for these logics.

For concrete heap semantics, we consider Reynolds's SL with all logical connectives but without arbitrary predicates. This logic is not recursively enumerable but is very useful in practice. We give a sound labelled sequent calculus LS_{SL} for this logic. We illustrate the subtle deficiencies of some existing proof calculi for SL, and show that our rules repair these problems. We extend LS_{SL} with rules for linked lists and binary trees, giving a sound, complete and terminating proof system for a popular fragment called symbolic heap. Our prover shows comparable results with Smallfoot on valid formulae and on formulae extracted from program verification examples, but it is not competitive on large invalid formulae. However, our prover handles the largest fragment of logical connectives in SL.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Programming is the daily job for many people working in information technology, computer science and engineering. Program code is compiled to build software or applications and is now embedded in almost every aspect of our lives. It is certainly important to write, compile and build programs, but it is also important to know that the programs are running in the way they are expected to. Testing is a common method to reduce errors in programs and software, but one can never *prove* that a program is correct by testing, as Edsger Dijkstra said [30]:

> Program testing can be used to show the presence of bugs, but never to show their absence!

Another way to attack the problem is to formally *verify* that a program is correct, this is a job about logical reasoning and proofs, and this is where Hoare logic [48] comes into play by analysing a piece of program code and the states before and after the code. Hoare logic employs a simple programming language with sequencing, assignment command, conditional command of the form "if...then...else", and "while" loop. Although this language is Turing complete, which means it can express any computable computation, modern programming languages often have more complicated features such as pointers, references etc., the use of which is error-prone in program code. Separation logic deals with this issue by extending Hoare logic with memory manipulation commands in the programming language. However, a new way to solve the problem also opens up more problems. The assertions in Hoare logic to express the states of programs are formulated in classical (first-order) logic with arithmetic, which is a well-developed logic but is no longer sufficient in separation logic. To express and reason about memory resources in separation logic, the assertional language in Hoare logic is also extended with new logical connectives and special predicates. Consequently, we need a new logic to reason about the assertions in separation logic.

Separation logic is a huge topic involving programming languages, assertional reasoning, and specification verification. In this dissertation we will not look at all of them, but will zoom in on the part of assertions, whose logic is historically ambiguously also called "separation logic". Our aim is to find proof methods to verify the

validity of assertions. This is particularly useful since some Hoare logic rules rely on this type of reasoning. The assertional language of separation logic, on the one hand is built upon classical logic, is also an application of the less known bunched logics [74], which involve two non-classical logics: intuitionistic logic [58] and linear logic [40]. This dissertation first studies bunched logics, then considers separation logics.

This chapter sets up the scene for our work and gives necessary background knowledge in classical logic, intuitionistic logic and linear logic. We discuss these logics at a high level in terms of syntax, semantics, proof theory, and proof search. Bunched logics will be reviewed in Chapter 2 and separation logics in Chapter 5. Chapter 3 and 4 give our main results on bunched logics, and Chapter 6 and 7 present our contributions on separation logics respectively with abstract and concrete semantics.

1.1 Logic and Reasoning

Logic in the broad sense is the study of valid reasoning, it is a cross-discipline subject that features philosophy, mathematics, and computer science. Logical reasoning takes many forms, historically we have been reasoning via informal natural languages, formal inferences, mathematics etc.. But unlike the fantasy advertised by many fictions and movies, logical reasoning never tells us anything fundamentally new, it only helps to dig out the facts and relations hidden in our knowledge. The logics we consider in this dissertation are *symbolic* logics, which focus on symbolic abstractions that capture formal reasoning using inferences. The notion of symbolism emerged centuries ago from the wild proposal of Leibniz, who intended to find a universal language that can express everything and a calculus of reasoning with some mechanisation to decide the truth of assertions in that language. Hilbert's programme started in around 1900 aimed at a related task to capture mathematics in consistent and complete theories. But after Gödel, Turing et al.'s work from 1931 onward, we now know that Leibniz and Hilbert were far too ambitious. However, the idea of formalising reasoning using abstract symbols has the advantage that the person, or even a machine, who carries out the proof does not need to understand the meaning of the proof and the object to be proved at all, one only has to apply the inference rules as instructed by the calculus and its mechanisation. This is exactly what machines are good at and why symbolism is the key to make automated reasoning possible. Here we call the symbolic representation of a language as the *syntax*, the meaning of symbols as *semantics*, the calculus of reasoning as *proof theory*, and the mechanisation as *proof search*. These will be the aspects we look at logic and reasoning in the sequel.

1.1.1 Syntax and Semantics of Classical Logic

We start from classical logic, which is widely used and arguably is the oldest of all logics, it is also a part of our long term target separation logic. We shall use the

notations of Fitting [34], Troelstra and Schwichtenberg [93], and Harrison [47] for the introduction to classical logic.

For the syntax, we consider the following logical connectives as first-class citizens: $\neg, \vee, \wedge, \rightarrow, \forall, \exists$. Logical constants are $\bot, \top$. Note that $=$ is not first-class in this definition. We use the following notation for objects in the language:

- variables are denoted by $x, y, z, ...$
- function symbols are denoted by $f, g, h, ...$
- constants are denoted by $a, b, c, ...$
- terms are denoted by $t, s, r, ...$
- atomic formulae are $P, Q, ...$
- relations/predicates symbols are denoted by R with subscripts or superscripts
- arbitrary formulae are written as $A, B, C, ...$

The arity of a function or a relation is the number of arguments it takes. Constants can be seen as functions of arity 0, i.e., functions with no arguments. *Terms* are built up from variables and functions as follows:

- any variable is a term;
- any expression $f(t_1, \cdots, t_n)$, where f is a n-ary function symbol and $t_1, \cdots, t_n$ are terms, is a term.

Formulae in classical (first-order) logic are defined inductively as:

- $\top, \bot$ are atomic formulae;
- any expression $R(t_1, \cdots, t_n)$, where R is a n-ary relation/predicate symbol and $t_1, \cdots, t_n$ are terms, is an atomic formula;
- if A is a formula, so is $\neg A$;
- if A, B are formulae, so are $A \vee B$, $A \wedge B$, $A \rightarrow B$;
- if A is a formula and x is a variable, then $\forall x.A$ and $\exists x.A$ are formulae.

The semantics of classical logic are determined by a non-empty *domain* D of objects that we can quantify over, an *interpretation* I that specifies the meaning of functions and relations, and a *valuation* v that specifies the meaning of variables. The pair (D, I) is called a *model*. We write D^n to mean a tuple of n elements in D. Formally, the interpretation I

- maps each n-ary function symbol f to a function $f_I : D^n \rightarrow D$;
- maps each n-ary relation symbol R to a boolean function $R_I : D^n \rightarrow \{\text{true,false}\}$.

The valuation v maps each variable to an item in the domain D. We write $t^{I,v}$ for the value of term t valuated by interpretation I and valuation v. The value of terms under the interpretation I and valuation v are defined as follows:

- $x^{I,v} = v(x)$

$$[R(t_1,\cdots,t_n)]^{I,v} = \begin{cases}\text{true} & \text{if } R_I(t_1^{I,v},\cdots,t_n^{I,v}) = true\\ \text{false} & \text{otherwise}\end{cases} \qquad [\neg A]^{I,v} = \begin{cases}\text{true} & \text{if } A^{I,v} \text{ is false}\\ \text{false} & \text{otherwise}\end{cases}$$

$$[A \wedge B]^{I,v} = \begin{cases}\text{true} & \text{if } A^{I,v} \text{ and } B^{I,v} \text{ are true}\\ \text{false} & \text{otherwise}\end{cases} \qquad [A \vee B]^{I,v} = \begin{cases}\text{true} & \text{if } A^{I,v} \text{ is true or}\\ & B^{I,v} \text{ is true}\\ \text{false} & \text{otherwise}\end{cases}$$

$$[A \to B]^{I,v} = \begin{cases}\text{true} & \text{if } A^{I,v} \text{ is false or}\\ & B^{I,v} \text{ is true}\\ \text{false} & \text{otherwise}\end{cases} \qquad [\exists x.A]^{I,v} = \begin{cases}\text{true} & \text{if for some } d \in D,\\ & A^{I,v[d/x]} \text{ is true}\\ \text{false} & \text{otherwise}\end{cases}$$

$$[\forall x.A]^{I,v} = \begin{cases}\text{true} & \text{if for all } d \in D,\\ & A^{I,v[d/x]} \text{ is true}\\ \text{false} & \text{otherwise}\end{cases} \qquad \bot^{I,v} = \text{false} \qquad \top^{I,v} = \text{true}$$

Table 1.1: The semantics of classical logic.

- $[f(t_1,\cdots,t_n)]^{I,v} = f_I(t_1^{I,v},\cdots,t_n^{I,v})$

A special case of a zero arity function is that a constant a is valuated as a^I, since no variables are involved. The semantics for each type of formula is given in Table 1.1, where $v[d/x]$ is a valuation that agrees with v but with x mapped to d.

A classical logic formula A is true in the model (D, I) if $A^{I,v}$ is true for every valuation v. A formula is *valid* if it is true in all models; a formula is *satisfiable* if it is true in some model.

1.1.2 Sequent Calculus for Classical Logic

There are often many proof calculi for a logic. In the case for classical logic, Hilbert gave a set of axioms and deduction rules, now called the Hilbert system (discussed in Section 2.2), Gentzen [39] later gave two types of proof calculi, namely the natural deduction calculus NK and the sequent calculus LK. Natural deduction calculus simulates the way humans do reasoning, while sequent calculus is more mechanical and amenable to mathematical analysis and automated reasoning. For this reason, we will only cover the latter calculus here.

A *sequent* in LK is an expression of the form

$$A_1,\cdots,A_n \vdash B_1,\cdots,B_m$$

where $A_1,\cdots,A_n$ and $B_1,\cdots,B_m$ are both sequences of formulae. The symbol $\vdash$ is called *turnstile*, which should be thought of as an implication. The left hand side of $\vdash$ is called the *antecedent*, the right hand side is the *succedent*. Commas in the antecedent mimic conjunction $\wedge$, while commas in the succedent are proxies for disjunction $\vee$.

Therefore the above sequent can be seen as

$$A_1 \wedge \cdots \wedge A_n \to B_1 \vee \cdots \vee B_m.$$

We shall use uppercase greek letters such as Γ, Δ for arbitrary structures, i.e., comma separated formulae. An inference rule r is in the form

$$\frac{\rho_1 \quad \cdots \quad \rho_n}{\alpha}\, r$$

where the sequents $\rho_1, \cdots, \rho_n$ are *premises* and the sequent α is the *conclusion*. A rule with no premises is called a *zero-premise rule*, a rule is *unary* (resp. *binary*) if it has one premise (resp. two premises). We may apply substitutions in a rule application. A substitution $[y/x]$ on a formula A replaces every occurrence of x in A by y. Inference rules in *LK* [39] are given in Figure 1.1[1].

The original *LK* calculus does not include rules for logical constants $\top$ and $\bot$, which can be encoded as $P \to P$ and $P \wedge \neg P$ respectively. We give the derived rules for $\top$ and $\bot$ as below.

$$\frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta}\, \top L \qquad \frac{}{\Gamma \vdash \top, \Delta}\, \top R \qquad \frac{}{\Gamma, \bot \vdash \Delta}\, \bot L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\, \bot R$$

A formula A is *provable/derivable* if we can build a derivation tree using the inference rules such that $\vdash A$ is the bottom sequent of the derivation, and each top sequent in the derivation is the empty sequent, i.e., the top of a zero-premise rule. See Figure 1.2 for an example derivation of $A \to (B \to (A \wedge B))$ in *LK*.

Usually we are interested in two basic properties of a proof calculus: soundness and completeness. A proof calculus is *sound* if every provable formula is valid, it is *complete* if every valid formula is provable. It is proven that *LK* is as powerful as *NK* and the Hilbert system: they are all sound and complete for classical logic.

1.1.3 Backward Proof Search and Cut-elimination

When we prove a formula A using *LK*, instead of starting from zero-premise rules and working downward, we often start from the sequent $\vdash A$ and apply the inference rules backwards (upwards). The applications of rules form a *derivation tree* where $\vdash A$ is the bottom sequent, and each rule application with n premises forks n branches in the tree. When we apply a zero-premise rule on a sequent upwards, we say the corresponding branch is *closed*. The formula is proved when every branch in the derivation tree is closed, this process is called *backward proof search*. A backward proof search procedure is effectively a proof by contradiction, explained as follows: a sequent $\Gamma \vdash \Delta$ can be falsified when "everything in Γ is true and everything in Δ is false". We start from $\vdash A$, assuming that A is false, and try to find counter-examples to falsify this sequent

[1]The rules $\vee L$ and $\wedge R$ copy all the context upwards, while the rule $\to L$ splits the context in the conclusion. These are Gentzen's original rules in *LK*.

Identity and Cut:

$$\frac{}{A \vdash A}\ id \qquad \frac{\Gamma \vdash \Delta, A \qquad A, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ cut$$

Logical Rules:

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L_1 \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L_2 \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R_1 \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R_2$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee L \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge R$$

$$\frac{\Gamma \vdash A, \Delta \qquad B, \Gamma' \vdash \Delta'}{\Gamma, \Gamma', A \to B \vdash \Delta, \Delta'}\ \to L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ \to R$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ \neg L \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ \forall L \qquad \frac{\Gamma \vdash A[y/x]\Delta}{\Gamma \vdash \forall x.A, \Delta}\ \forall R \qquad \frac{\Gamma, A[y/x] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\ \exists L \qquad \frac{\Gamma \vdash A[t/x]\Delta}{\Gamma \vdash \exists x.A, \Delta}\ \exists R$$

Structural Rules:

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ WL \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ WR \qquad \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ CL \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\ CR$$

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\ EL \qquad \frac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\ ER$$

Side Condition: in $\forall R$ and $\exists L$, y does not occur in the conclusion.

Figure 1.1: The sequent calculus LK for classical logic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\ id}{A, B \vdash A}\ WL \qquad \dfrac{\dfrac{}{B \vdash B}\ id}{B, A \vdash B}\ WL}{A, B \vdash A \wedge B}\ EL}{A \vdash B \to (A \wedge B)}\ \wedge R}{\vdash A \to (B \to (A \wedge B))}\ \to R\ \to R$$

Figure 1.2: An example derivation of $A \to (B \to (A \wedge B))$ in LK.

using backward proof search. The fact that the proof search ends with every branch closed means that every possibility for A to be false leads to a contradiction, so A must be valid. If a branch in the derivation cannot be closed whatsoever, then that branch is indeed a counter-example that falsifies A.

One important aspect of backward proof search is to control the structural rules, especially cut and contraction, because they can be applied as often as one wishes hence they may yield infinite proof search. Logical rules, on the other hand, break the formulae down to smaller subformulae, and eliminate logical connectives upwards, thus purely logical rule applications from the original LK would eventually stop. As a result, many variants of LK have arisen to ensure that proof search is driven by logical connectives.

It is easy to make the rules EL and ER admissible. We only need to assume that the antecedent and succedent of a sequent are made of multisets rather than lists. The rules WL, WR for weakening[2] can be absorbed into zero-premise rules, in fact, the derived rules for $\top R$ and $\bot L$ already have weakening built in. That is, we allow contexts Γ, Δ to appear in the conclusion. Similarly, we can change the rule *id* to

$$\frac{}{\Gamma, A \vdash A, \Delta}\ id$$

Contraction, handled by the rules CL and CR, needs more care. We have to copy some formulae or structures from the conclusion to the premise(s) when necessary. The following rules for $\wedge L$ and $\vee R$ merge the two cases in the original LK:

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R$$

The new $\wedge L$ rule, for example, builds in a contraction of $A \wedge B$, and applications of $\wedge L_1$ and $\wedge L_2$ to obtain A and B respectively. In $\rightarrow L$, we now need to copy the entire context to both premises. That is,

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

Finally, the $\forall L$ rule assumes that $\forall x.A$ is true in the conclusion, and replaces x by some t in the premise. Since every item d in the domain makes $A[d/x]$ true, the t we choose in the rule application may not be the (only) one we need, thus we should keep the formula $\forall x.A$ so that we can instantiate x to other items in the domain at higher places in the derivation, similarly for the rule $\exists R$. In light of this, the rules $\forall L$ and $\exists R$ are modified as follows:

$$\frac{\Gamma, \forall x.A, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ \forall L \qquad \frac{\Gamma \vdash \exists x.A, A[t/x], \Delta}{\Gamma \vdash \exists x.A, \Delta}\ \exists R$$

Note that the principal formula in the above rules is copied to the premise. Thus the principal formula can be used as many times as possible in the proof search. The

Identity and Cut:

$$\frac{}{\Gamma, A \vdash A, \Delta}\ id$$

Logical Rules:

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge R$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee L \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\ \to L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ \to R$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ \neg L \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R$$

$$\frac{\Gamma, \forall x.A, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ \forall L \qquad \frac{\Gamma \vdash A[y/x]\Delta}{\Gamma \vdash \forall x.A, \Delta}\ \forall R \qquad \frac{\Gamma, A[y/x] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\ \exists L \qquad \frac{\Gamma \vdash \exists x.A, A[t/x], \Delta}{\Gamma \vdash \exists x.A, \Delta}\ \exists R$$

Side Condition: in $\forall R$ and $\exists L$, y does not occur in the conclusion.

Figure 1.3: The alternative sequent calculus LK' for classical logic.

above modifications yield an alternative version of LK, here called LK', summarised in Figure 1.3. The proof systems LK and LK' are equivalent.

We say an inference rule

$$\frac{\rho_1 \quad \cdots \quad \rho_n}{\alpha}\ r$$

is *admissible* in a calculus when we can prove that, if the premises $\rho_1, \cdots, \rho_n$ are all derivable in the calculus, then the conclusion α is also derivable in the calculus without using the rule r. With the above modifications, all the structural rules in LK are admissible, so they are not needed for the completeness of LK anymore.

There is another rule that may cause uncertainty in proof search: the *cut* rule. In a backward *cut* application, we have to guess the formula A (called the *cut* formula) in the premise, that formula can be any (sub)formula in the conclusion, or even a new formula. The presence of the *cut* rule certainly is a headache in proof search, this leads to Gentzen's Hauptsatz (main theorem), the *cut-elimination* theorem: every formula

[2] Called "thinning" in the original *LK*.

that is derivable in *LK* can be derived without using the *cut* rule. The profound impact of cut-elimination, however, is not in our scope of discussion.

The alternative proof system LK' is more amenable to backward proof search and automation. The only trouble in proof search would be the new $\forall L$ and $\exists R$ rule, which do not eliminate the corresponding connective in the premise, and may still cause non-termination, which unfortunately is inevitable for classical (first-order) logic reasoning. At the worst, we can still tell a machine to apply the rules *fairly*, e.g., iteratively apply each rule in every possible way so that each possible rule application could eventually be tried. In this way, if a formula is valid, we will eventually find a derivation; but if a formula is invalid, our algorithm may never stop.

1.2 Intuitionistic Logic and Linear Logic

While classical logic has been used for a long time in the history, there are situations where we demand different ways of reasoning. The assertional language of separation logic (cf. Section 5.1), for example, finds classical logic inadequate, thus extends classical logic with some features from bunched logics (cf. Section 2.1), which, besides the use of classical logic, also mix the flavours of two non-classical logics: intuitionistic logic and linear logic. We briefly discuss these two non-classical logics in this section.

1.2.1 Intuitionistic Logic

The need for intuitionism, dating back to early 20th century, comes from the notion of "constructive proof" in mathematics, which shows each step of reasoning by giving an evidence of a mathematical object [92]. Classical logic validates two theorems that have the same effect on provability: $\neg\neg A \to A$, i.e., double negation elimination; and $A \vee \neg A \leftrightarrow \top$, i.e., the law of excluded middle. It turns out that these two classical theorems are the source of non-constructivism in classical logic, and denying either of them from classical logic gives intuitionistic logic.

Let us see some examples of non-constructive proofs [47]. A typical example proof using double negation elimination is for the following problem:

> Prove that at least one of $e + \pi$ and $e - \pi$ is irrational.

Proof. Proof by contradiction. Assume that both $e + \pi$ and $e - \pi$ are rational, then their sum $2e$ should be rational, which is false, therefore at least one of them is irrational. □

The above proof is perfectly fine, but we still do not know which one of $e + \pi$ and $e - \pi$ is irrational, that is, no evidence is given in the proof. For another example using the law of excluded middle, consider the following:

> Prove that there are two irrational numbers x, y such that x^y is rational.

Proof. Consider $\sqrt{2}^{\sqrt{2}}$, if it is rational, then $x = y = \sqrt{2}$ is a solution; otherwise let $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$, $x^y = \sqrt{2}^2 = 2$ is rational. □

Again, the proof is legitimate, but there is no concrete evidence of x and y.

A logic can be defined by its syntax and semantics, or it can be defined by syntax and proof theory. In this and the next subsection, we shall define logics by giving its syntax and proof theory.

The syntax of intuitionistic logic is the same as classical logic, we only need to modify the sequent calculus *LK* so that theorems such as double negation elimination and the law of excluded middle cannot be proven. Gentzen originally proposed a sequent calculus *LJ* for intuitionistic logic as *LK* with the restriction that in the succedent of each sequent, no more than one formula is permissible.

In this dissertation we will mainly use intuitionistic logic at the propositional level, that is, we do not consider terms, functions, relations, and quantifiers; the atomic formulae will then be just atomic propositions. The sequent calculus for propositional intuitionistic logic thus does not have rules for quantifiers, and there is an easy adaptation from *LK* without quantifier rules, one only need to change the $\to R$ rule to the following:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B, \Delta} \to R$$

This variant of *LJ* is often called LJ', which allows multiple formulae in the succedent, but the context Δ in the rule $\to R$ does not appear in the premise. The sequent calculus LJ' is sound and complete for intuitionistic logic, and it enjoys cut-elimination.

1.2.2 Linear Logic

Another situation where classical logic is inappropriate is reasoning about resources. For example, we can phrase "if I have a coin, I can buy a chocolate" as $coin \to choc$, but the idempotence of $\wedge$ in classical logic (i.e., $A \leftrightarrow A \wedge A$) makes the above formula equivalent to $coin \to choc \wedge choc$, which, depending on the interpretation of $\wedge$, might be understood as "if I have a coin, I can buy two chocolates", which is obviously not the same as the original sentence. So logical connectives such as conjunction need to be interpreted carefully when resource reasoning is involved. If we interpret $A \wedge A$ as two copies of A, then we should not make $\wedge$ idempotent.

Girard's linear logic [40] deals with this problem by allowing two different flavours of logical connectives: normal logical connectives (additives) as in classical logic, and linear connectives (multiplicatives) that are not idempotent.

Even at the propositional level, the syntax of linear logic is more involved than propositional (classical) logic, here we shall adapt the notation from Troelstra and Schwichtenberg [93] instead, which gives more resemblance to classical logic and is less confusing. Additive logical constants are $\top$ and $\bot$, multiplicatively we have "unit"

1 and "zero"[3] 0. Additive logical connectives are $\wedge, \vee, \rightarrow$ as usual; while multiplicatively the conjunction $*$ is called "tensor" (or "times", written as $\circ$ in some literature), the disjunction $+$ is called "par", the implication $\multimap$ is called "lollipop". Note that there is only one negation $\neg$. Moreover, to give controlled access to weakening and contraction, we add *exponentials* ? (called "why not") and ! (called "of course").

Proof theoretically, linear logic is a typical *substructural logic*, as it lacks some of the structural rules in *LK*, i.e., the rules for weakening and contraction. While the rules for additive connectives and the negation $\neg$ are the same as the ones in *LK* (cf. Figure 1.1, we include $\top R$ and $\bot L$, but not $\top L$ nor $\bot R$; we also do not need rules for $\rightarrow$, which can be encoded as $A \rightarrow B \equiv (!A) \multimap B$, the rules for multiplicative connectives treat the context in a sequent differently. The rules for unit and zero are given below, where $1R$ and $0L$ require that there is no context at all:

$$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}\,{}^{1L} \qquad \frac{}{\vdash 1}\,{}^{1R} \qquad \frac{}{0 \vdash}\,{}^{0L} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash 0, \Delta}\,{}^{0R}$$

The rules for tensor, par and lollipop are similar to the *LK* rules in Section 1.1.3, although here we do not build in structural rules, but instead emphasise the resource reading by forcing that the context has to be split in binary rules (i.e., the context in the conclusion cannot be copied to both premises). The rules are shown as follows:

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A * B \vdash \Delta}\,{}^{*L} \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A * B, \Delta, \Delta'}\,{}^{*R}$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A + B \vdash \Delta, \Delta'}\,{}^{+L} \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A + B, \Delta}\,{}^{+R}$$

$$\frac{\Gamma \vdash A, \Delta \qquad B, \Gamma' \vdash \Delta'}{\Gamma, \Gamma', A \multimap B \vdash \Delta, \Delta'}\,{}^{\multimap L} \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}\,{}^{\multimap R}$$

The exponential ! allows a formula to be used zero or more times, while the dual exponential ? means that a formula can be obtained zero or more times. We allow weakening and contraction for ! formulae in the antecedent and ? formulae in the succedent. The rules for ! and ? are given below.

$$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}\,{}^{W!} \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,{}^{L!} \qquad \frac{!\Gamma, \vdash A, ?\Delta}{!\Gamma \vdash !A, ?\Delta}\,{}^{R!} \qquad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,{}^{C!}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash ?A, \Delta}\,{}^{W?} \qquad \frac{!\Gamma, A \vdash ?\Delta}{!\Gamma, ?A \vdash ?\Delta}\,{}^{L?} \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash ?A, \Delta}\,{}^{R?} \qquad \frac{\Gamma \vdash ?A, ?A, \Delta}{\Gamma \vdash ?A, \Delta}\,{}^{C?}$$

Exponentials are not used in the sequel, but we will use multiplicative connectives from linear logic, although in slightly different notations. The *cut* rule

[3] In Girard's original presentation, 0 is additive falsity, $\top$ is additive truth; $\bot$ is multiplicative falsity ("zero"), and 1 is multiplicative truth ("unit"). Also, the symbols for logical connectives are different.

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ cut$$

can be eliminated in this system and completeness is still maintained.

In Section 2.1, we will introduce bunched logics using $*$, $-\!\!*$, $\vee^*$, $\sim$, $\top^*$, $\bot^*$ as the multiplicative conjunction, implication, disjunction, negation, truth and falsity respectively.

1.3 Kripke Semantics and Labelled Sequent Calculus

We have introduced two non-classical logics from the syntax and proof theory point of view, here we discuss how to present and manipulate the semantics for non-classical logics, using intuitionistic logic as the running example.

In this dissertation we will be interested in Kripke style semantics for non-classical logics, here we give a brief review based on Goré's presentation [42]. The core in the semantics is usually called a *Kripke frame*, which is a pair (W, R) of a non-empty set W of possible worlds and a relation R on W. The relation R is often a binary relation for many logics, although later we will be interested in ternary relations. Given two worlds $w, w' \in W$, we say w' is a *successor* of w, or equivalently, w is a *predecessor* of w', if $R(w, w')$ holds.

For propositional non-classical logics, we use a valuation v from atomic propositions to sets of worlds, i.e., $v : PVar \to \mathcal{P}(W)$, where $PVar$ is the set of propositions and $\mathcal{P}(W)$ denotes the powerset of W. Given a proposition p, $v(p)$ is the set of worlds that make p true. The triple (W, R, v) is called a *Kripke model*. The semantics for logical connectives are defined using a *forcing relation* $\Vdash$ between a world and a formula. We write $w \Vdash A$ if A is true at w; we write $w \nVdash A$ if A is not true at w.

Kripke semantics are not particularly useful for classical logic, in which every formula is true in the same world, so we only need a singleton set W, which is not very interesting. This is not the case for intuitionistic logic. Truth is *persistent* in intuitionistic logic: if a world makes p true, then its successor also makes p true. Formally, for any valuation v and any two worlds w and w',

$$\text{if } w \in v(p) \text{ and } R(w, w') \text{ then } w' \in v(p).$$

The semantics for intuitionistic conjunction and disjunction are the same as in classical logic, that is,

$$w \Vdash A \wedge B \text{ iff } w \Vdash A \text{ and } w \Vdash B$$
$$w \Vdash A \vee B \text{ iff } w \Vdash A \text{ or } w \Vdash B$$

But intuitionistic implication is interpreted with the binary relation R:

$$w \Vdash A \to B \text{ iff } \forall w'.(R(w, w') \text{ and } w' \Vdash A) \text{ implies } w' \Vdash B.$$

And of course, every world makes $\top$ true and no worlds make $\bot$ true.

For easier access to semantics in proof search, we can integrate the Kripke semantics into sequent calculus, giving a *labelled sequent calculus* [72], where the structures Γ and Δ in a sequent $\Gamma \vdash \Delta$ consist of *labelled formulae* of the form $w : A$ and *relational atoms* of the form wRw'. The intended meaning is that $w : A$ means A is true at w, and wRw' means that $R(w, w')$ holds. The rules for $\wedge$, for example, are straightforward:

$$\frac{\Gamma, w : A, w : B \vdash \Delta}{\Gamma, w : A \wedge B \vdash \Delta} \wedge L \qquad \frac{\Gamma \vdash w : A, \Delta \qquad \Gamma \vdash w : B, \Delta}{\Gamma \vdash w : A \wedge B, \Delta} \wedge R$$

The rules for $\to$ are more tricky, since we need to consider the quantifier in the semantics. From the backward proof search reading, if $w : A \to B$ occurs in the antecedent and the sequent is falsifiable, then $w : A \to B$ is true, so for every wRw', either $w' : A$ is false or $w' : B$ is true, giving the following rule:

$$\frac{\Gamma, wRw' \vdash w' : A, \Delta \qquad \Gamma, wRw', w' : B \vdash \Delta}{\Gamma, wRw', w : A \to B \vdash \Delta} \to L$$

On the other hand, if $w : A \to B$ is in the succedent and the sequent is falsifiable, then $w : A \to B$ is false. By negating the semantics, we obtain that there exists some w' such that wRw' holds and $w' : A$ is true and $w' : B$ is false. Since we do not know what w' exactly is, we follow the method from the $\exists L$ rule in *LK*: create a fresh world for w'. Thus the $\to R$ rule runs as follows:

$$\frac{\Gamma, wRw', w' : A \vdash w' : B, \Delta}{\Gamma \vdash w : A \to B, \Delta} \to R$$

w' does not occur in the conclusion.

Binary relations are sufficient for capturing the semantics of intuitionistic logic, but in this dissertation we will be concerned with more complex logics for resource reasoning, these logics involve ternary relations instead. For example, we may want to capture that the combination of two resources x and y form another resource z, this will sometimes be denoted as $x \circ y = z$. To be consistent with the Kripke semantics notation, we shall write $R(x, y, z)$ for such a ternary relation. We may also use $\triangleright$ for the relation symbol R, written infix, as in $(x, y \triangleright z)$. We prefer to use the former form in the semantics, and use the latter form in our labelled calculi.

As a hindsight, although labelled sequent calculus does not tell us much about proof theory itself, it certainly is a convenient way to analyse how the relation R relates the worlds and how to assign formulae to the worlds they are true at. This is vital to our main contributions in this dissertation, but we shall stop the spoiler for now.

Chapter 2

The Logics of Bunched Implications

We have seen the proof theory and semantics of classical logic, intuitionistic logic, and linear logic in Chapter 1. In this chapter we give a literature review for more specific logics that we shall consider in this dissertation. These include the logic of bunched implications (BI) and its neighbours, i.e., Boolean BI (BBI), Classical BI (CBI), etc.. We will see that these logics combine the flavours of classical logic, intuitionistic logic, and linear logic in certain ways, but meanwhile are different from the previous logics both proof theoretically and semantically.

BI was developed in 1999 by O'Hearn and Pym with a semantic motivation on resource reasoning, although the logic and its neighbours were mainly developed in terms of syntax and proof theory [75]. Algebraic semantics were discussed as an alternative perspective for defining the logic, but the original Kripke semantics proposed for BI were only sound and complete for BI without $\bot$, and incomplete for BI [84]. Complete Kripke semantics for BI and BBI were developed years later by Galmiche et al. [38, 36]. The semantics, proof theory, and applications of CBI were then studied by Brotherston and Calcagno [17].

Interestingly, the resource reasoning aspect of BI and BBI had found successful applications even before their semantics were properly developed. Early in 2000, Reynolds proposed extensions of Hoare logic to reason about shared mutable data structures using BI as the assertion logic [87]. This perspective was also pursued by Ishtiaq, O'Hearn, Yang etc. who gave different extensions of Hoare logic using BBI instead [54, 76]. In 2002, Reynolds used BBI as the assertion logic and formalised the above as separation logic [88], which has been a hot topic and has received much attention in the program verification community. We will come back with a literature review for separation logic in Chapter 5.

In this chapter we will first give the syntax and semantics for BI, BBI, and CBI in Section 2.1, as these logics are the ones we will use in later chapters. Then we review the proof theoretical perspective of these logics. Respectively in Section 2.2 we present the Hilbert system, which is useful when proving completeness of other calculi and generating random theorems; and in Section 2.3 we give Brotherston's Display Calculi, which is the basis for developing shallow and deep inference calculi

in Chapter 3. More specific BI variants and applications of BI logics are discussed in Section 2.4.

2.1 BI and Its Neighbours

This section introduces BI and its neighbours. These logics commonly have two counterparts that can be freely combined when building formulae: the additive connectives allow weakening and contraction; and the multiplicative connectives behave like linear connectives and forbid weakening and contraction. Both types of connectives could be either classical, in the sense that double negation elimination is allowed, or intuitionistic. The above choices yield four possible combinations as summarised below.

Logic	Additives	Multiplicatives
BI	intuitionistic	intuitionistic
dMBI	intuitionistic	classical
BBI	classical	intuitionistic
CBI	classical	classical

These variants of BI are sometimes viewed from an algebraic perspective. Combining Heyting algebra (as additives) and Lambek algebra (as multiplicatives) gives BI; changing the former to Boolean algebra gives BBI; Boolean additives and de Morgan multiplicatives result in CBI; finally, Heyting additives and de Morgan multiplicatives give dMBI. In this view the names of these logics are self-explanatory.

Since most applications of BI logics are based on BBI, which is also the focus in this dissertation, we shall pay more attention to BBI than other variants. On the contrary, dMBI is a rarely used variant, thus will not be discussed here.

2.1.1 The Logic of Bunched Implications

We start with a rather informal introduction to BI, as we will not use this logic directly. More technical details will come in the next subsection when we introduce BBI.

With resource reasoning in mind, O'Hearn and Pym developed the syntax of BI as a combination of additive connectives $\top, \bot, \to, \wedge, \vee$ and multiplicative connectives $\top^*, *, -\!\!*$, defined formally as below, where p denotes an atomic proposition:

$$A ::= \ p \ \mid \ \top \ \mid \ \bot \ \mid \ A \wedge A \ \mid \ A \vee A \mid \ A \to A \ \mid \ \top^* \ \mid \ A * A \ \mid \ A -\!\!* A$$

By tradition $*$ is called "star" and $-\!\!*$ is called "magic wand". We use $\top^*$ as the multiplicative unit as opposed to I in the literature just to reduce ambiguity when we introduce alphabetic symbols later.

We will not give the Kripke-style semantics of BI as it is not very important to our following discussion, instead, we will give a Hilbert-style system for BI in Section 2.2 to complete the picture. Interested readers are referred to Pym et al. and Galmiche et

al.'s work for a detailed account on BI semantics [85, 38]. Informally, we shall borrow the following reading of BI connectives from Pym [84]:

$A \to B$**:** if I were to obtain enough money to make A true, then I should also have enough to make B true.

$A \wedge B$**:** the money I have got is both enough to make A true and enough to make B true (but may not be enough to make A and B true at the same time).

$A \vee B$**:** the money I have got is enough to make A true or to make B true.

$A * B$**:** I can use part of my money to make A true, and the left over is enough to make B true (and vice versa).

$A \mathrel{-\!\!*} B$**:** if you were to give me enough money to make A true, combined with what I already have, I should then have enough to make B true.

In this reading the multiplicative unit $\top^*$ simply means "I don't have any money", $\top$ is "true no matter how much I have", but $\bot$ is much more complicated to capture. Saying $\bot$ is "false no matter how much I have" might result in incompleteness of the semantics. See [38] for more detailed and formal discussion.

BI differs from linear logic in several ways [84]. Firstly, Girard's reading of linear logic has a "proofs-as-actions" flavour [40], where a proposition is a resource and a proof is a way to manipulate resources. In BI, the reading is "completely declarative", a proposition is just a statement whose truth value may involve consideration of resources. Secondly, also more technically, in BI we expect the following [84]:

$$coin \mathrel{-\!\!*} choc \nvdash coin \to choc$$

That is, "if I could obtain a coin then I would have enough to buy a chocolate" does not imply that "if I had a coin I would be able to buy a chocolate". The former combines my existing money with the extra coin I get, while the latter only uses the coin. In linear logic, however, when $A \to B$ is defined by $!A \multimap B$, we have the following:

$$coin \multimap choc \vdash coin \to choc$$

The intuitionistic reading in the above explanation of connectives may not be obvious, to clarify this, we use Galmiche et al.'s notation below [38]. One may think of a relation $\sqsubseteq$ to "compare" resources, so $m \sqsubseteq n$ means that m is less than or equal to n. The intuitionistic implication can be interpreted using a Kripke style forcing relation as follows:

$$m \Vdash A \to B \text{ iff } \forall n \text{ such that } m \sqsubseteq n,\ n \Vdash A \text{ implies } n \Vdash B.$$

Whereas the classical implication does not concern other resources:

$$m \Vdash A \to B \text{ iff } m \Vdash A \text{ implies } m \Vdash B.$$

The linear reading of the multiplicative connectives involves an operation $\circ$ to "combine" resources, thus $m \circ n$ is the combination of m and n. The $*$ connective effectively considers two parts that can be combined to form the current resource:

$$w \Vdash A * B \text{ iff } \exists m, n \text{ s.t. } m \circ n = w \text{ and } m \Vdash A \text{ and } n \Vdash B.$$

Therefore $A * A \iff A$ is not true in general. For example, saying "the money I have is enough to buy a chocolate" does not imply that "a part of my money is enough to buy a chocolate, the rest is enough to buy another chocolate". The $-\!\!*$ connective features combining the current resource with a foreign resource:

$$n \Vdash A -\!\!* B \text{ iff } \forall m, w \text{ if } m \circ n = w \text{ and } m \Vdash A \text{ then } w \Vdash B.$$

The notion of BI validity is defined based on its proof theory rather than semantics. Traditionally, the sequent calculus LBI for BI has nested structures (but not nested sequents) in the sequent [84]. This is a natural consequence of having two types of connectives. That is, we need two types of structural connectives ";" and "," in the sequent to respectively denote additive conjunction and multiplicative conjunction in the antecedent of the sequent. The succedent only contains a single formula since both types of connectives are intuitionistic. As a result, we may have the following nested structure in a sequent:

$$A_1; (A_2, A_3, (A_5; A_6; \cdots)) \vdash B$$

Another perspective is that the antecedent is now a tree with ";" and "," as internal nodes and formulae as leaves. Such trees are called *bunches*, thus we have the name of this logic as "bunched implications". Formally, we use Γ and Δ for bunches, defined below, where A is a BI formula:

$$\Gamma ::= \ A \mid \varnothing_m \mid \Gamma, \Gamma \mid \varnothing_a \mid \Gamma; \Gamma$$

Just as we have two types of structural connectives, we also have two structural units: $\varnothing_a$ for additive connectives, and $\varnothing_m$ for multiplicative connectives. The "turnstile" in the sequent can also be interpreted in different ways, $\varnothing_a \vdash A$ corresponds to an additive implication, and $\varnothing_m \vdash A$ corresponds to a multiplicative implication. We write $\Gamma(\Delta)$ for a bunch Γ in which Δ appears as a sub-tree.

The sequent calculus LBI for BI [84] is given in Figure 2.1. The exchange rule E allows the order of comma or semicolon separated formulae to be shuffled, i.e.,

$$A; B \equiv B; A \text{ and } A, B \equiv B, A.$$

The additive rules and multiplicative rules look very similar in LBI, but weakening and contraction can only operate on semicolons, not commas. The rule $\vee L$ requires two structures Γ and Δ in the conclusion, each contains the formula $A \vee B$. A simpler formulation

Identities:

$$\frac{}{A \vdash A}\ axiom \qquad \frac{\Gamma \vdash A \qquad \Delta(A) \vdash B}{\Delta(\Gamma) \vdash B}\ cut$$

Structural Rules:

$$\frac{\Gamma(\Delta) \vdash A}{\Gamma(\Delta;\Delta') \vdash A}\ W \qquad \frac{\Gamma(\Delta;\Delta) \vdash A}{\Gamma(\Delta) \vdash A}\ C \qquad \frac{\Gamma \vdash A}{\Delta \vdash A}\ E, (\Delta \equiv \Gamma)$$

Units:

$$\frac{\Gamma(\varnothing_m) \vdash A}{\Gamma(\top^*) \vdash A}\ \top^* L \qquad \frac{}{\varnothing_m \vdash \top^*}\ \top^* R \qquad \frac{}{\Gamma(\bot) \vdash A}\ \bot L$$

$$\frac{\Gamma(\varnothing_a) \vdash A}{\Gamma(\top) \vdash A}\ \top L \qquad \frac{}{\varnothing_a \vdash \top}\ \top R$$

Multiplicative Rules:

$$\frac{\Gamma \vdash A \qquad \Delta(\Delta', B) \vdash C}{\Delta(\Delta', \Gamma, A \mathrel{-\!\!*} B) \vdash C}\ {-\!\!*}\, L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \mathrel{-\!\!*} B}\ {-\!\!*}\, R \qquad \frac{\Gamma(A, B) \vdash C}{\Gamma(A * B) \vdash C}\ *L \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A * B}\ *R$$

Additive Rules:

$$\frac{\Gamma \vdash A \qquad \Delta(\Delta'; B) \vdash C}{\Delta(\Delta'; \Gamma; A \to B) \vdash C}\ \to L \qquad \frac{\Gamma; A \vdash B}{\Gamma \vdash A \to B}\ \to R \qquad \frac{\Gamma(A; B) \vdash C}{\Gamma(A \wedge B) \vdash C}\ \wedge L$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma; \Delta \vdash A \wedge B}\ \wedge R \qquad \frac{\Gamma(A) \vdash C \qquad \Delta(B) \vdash C}{\Gamma(A \vee B); \Delta(A \vee B) \vdash C}\ \vee L \qquad \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \vee A_2}\ \vee R, (i = 1, 2)$$

Figure 2.1: The sequent calculus LBI for BI.

$$\frac{\Gamma(A) \vdash C \qquad \Gamma(B) \vdash C}{\Gamma(A \vee B) \vdash C}$$

can be derived via a contraction on $\Gamma(A \vee B)$ in the conclusion.

A formula A is *valid* if either $\varnothing_a \vdash A$ is provable or $\varnothing_m \vdash A$ is provable in LBI. But since the former implies the latter, the above can be redefined as

> A formula A is *valid* if $\varnothing_m \vdash A$ is provable in LBI.

This notion of validity is different from the validity of BBI and CBI as we will see later. The multiplicative unit $\varnothing_m$ stands for the empty resource, so a formula A is valid in BI can be understood as "A is true when I don't have anything", but BBI and CBI validity of a formula A is characterised as "A is true no matter what I have".

2.1.2 Boolean BI

This subsection gives a formal introduction to BBI, the definitions here will be vital in Chapter 4 where we discuss a labelled sequent calculus for BBI.

BBI formulae are defined inductively as follows, where p is an atomic proposition, $\top^*, *, -\!\!*$ are the multiplicative unit, multiplicative conjunction, and multiplicative implication respectively:

$$A ::= \ p \mid \top \mid \bot \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \top^* \mid A * A \mid A -\!\!* A$$

Since the additive part in BBI is classical, we can simplify the language by just using $\to$ and $\bot$ as additive connectives and consider $\top, \neg, \wedge, \vee$ as defined ones.

As for BI, BBI was first defined as the above syntax plus a Hilbert system [84]. The corresponding semantics of BBI came later in terms of non-deterministic monoids and ternary relations [36].

A non-deterministic monoid is a triple $(\mathcal{M}, \circ, \epsilon)$ where $\mathcal{M}$ is a non-empty set, $\epsilon \in \mathcal{M}$ and $\circ : \mathcal{M} \times \mathcal{M} \to \mathcal{P}(\mathcal{M})$ is a mapping from pairs of members of $\mathcal{M}$ to subsets of $\mathcal{M}$. The extension of $\circ$ to $\mathcal{P}(\mathcal{M})$ is defined as $X \circ Y = \bigcup\{x \circ y : x \in X, y \in Y\}$. The following conditions hold in this monoid:

Identity: $\forall a \in \mathcal{M}.\ \epsilon \circ a = \{a\}$
Commutativity: $\forall a, b \in \mathcal{M}.\ a \circ b = b \circ a$
Associativity: $\forall a, b, c \in \mathcal{M}.\ a \circ (b \circ c) = (a \circ b) \circ c$.

Galmiche and Larchey-Wendling also gave another view of the semantics in terms of a ternary relation. As it is also a tradition in modal logic to define the semantics based on accessibility relation of worlds, we adopt the ternary relation view of the semantics in the following.

The ternary relation over worlds is defined by $R \subseteq \mathcal{M} \times \mathcal{M} \times \mathcal{M}$ such that $R(a, b, c)$ if and only if $c \in a \circ b$. We therefore have the following conditions for all $a, b, c, d \in \mathcal{M}$:

Identity: $R(\epsilon, a, b)$ iff $a = b$
Commutativity: $R(a, b, c)$ iff $R(b, a, c)$
Associativity: If $\exists k$ s.t. $R(a, k, d)$ and $R(b, c, k)$ then $\exists l$ s.t. $R(a, b, l)$ and $R(l, c, d)$.

Thus we obtain a BBI *relational frame* $(\mathcal{M}, R, \epsilon)$ from a non-deterministic monoid $(\mathcal{M}, \circ, \epsilon)$ in the obvious way. Intuitively, the relation $R(x, y, z)$ means that z can be partitioned into two parts: x and y. The identity condition can be read as every world can be partitioned into an empty world and itself. Commutativity captures that partitioning z into x and y is the same as partitioning z into y and x. Finally, associativity means that if z can be partitioned into x and y, and x can be further partitioned into u and v, then all together z consists of u, v and y. Therefore there must exist an element w which is the combination of v and y, such that w and u form

z. Note that since we do not restrict this monoid to be cancellative (i.e., $x \circ y = x \circ z$ implies $y = z$), $R(x, y, x)$ does not imply $y = \epsilon$.

A BBI model $(\mathcal{M}, R, \epsilon, v)$ consists of a relational frame $(\mathcal{M}, R, \epsilon)$ and a *valuation* $v : PVar \to \mathcal{P}(\mathcal{M})$ where $PVar$ is the set of propositional variables (i.e., atomic propositions) and $\mathcal{P}(\mathcal{M})$ is the powerset of $\mathcal{M}$. A *forcing relation* "$\Vdash$" between elements of $\mathcal{M}$ and BBI-formulae is defined as follows [36]:

$$
\begin{array}{ll}
m \Vdash \top^* \text{ iff } m = \epsilon & m \Vdash p \text{ iff } p \in PVar \text{ and } m \in v(p) \\
m \Vdash \bot \text{ iff never} & m \Vdash A \vee B \text{ iff } m \Vdash A \text{ or } m \Vdash B \\
m \Vdash \top \text{ iff always} & m \Vdash A \wedge B \text{ iff } m \Vdash A \text{ and } m \Vdash B \\
m \Vdash \neg A \text{ iff } m \nVdash A & m \Vdash A \to B \text{ iff } m \nVdash A \text{ or } m \Vdash B \\
m \Vdash A * B \text{ iff } \exists a, b.(R(a, b, m) \text{ and } a \Vdash A \text{ and } b \Vdash B) & \\
m \Vdash A \mathrel{-\!\!*} B \text{ iff } \forall a, b.(R(m, a, b) \text{ and } a \Vdash A) \text{ implies } b \Vdash B) &
\end{array}
$$

Given a BBI model $(\mathcal{M}, R, \epsilon, v)$, a BBI formula A is true at $m \in \mathcal{M}$ if $m \Vdash A$. A BBI formula A is *valid* if it is true at every world in every BBI model.

2.1.3 Classical BI

Classical logic differs from intuitionistic logic by having a stronger negation, this is also the case for BI logics. The multiplicative part in BI and BBI are both intuitionistic and only have $\top^*, *, \mathrel{-\!\!*}$ as logical connectives. In CBI, where both the additive part and the multiplicative part are classical, multiplicative negation is present in the language, from which we can also define multiplicative falsity and disjunction. The syntax of CBI is defined as below:

$$
\begin{array}{c}
A ::= p \mid \top \mid \bot \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \\
\top^* \mid \bot^* \mid \sim A \mid A * A \mid A \vee^* A \mid A \mathrel{-\!\!*} A
\end{array}
$$

where $\bot^*, \sim, \vee^*$ are respectively the multiplicative falsity, negation, and disjunction.

The semantics of CBI extend that of BBI by adding more items to specialise the model. We shall use the definition by Brotherston and Calcagno [17] as below.

Definition 2.1.1. *A CBI model is a tuple $(\mathcal{M}, R, \epsilon, -, \infty, v)$ where $(\mathcal{M}, R, \epsilon, v)$ is a BBI model and $- : \mathcal{M} \to \mathcal{M}$ and $\infty \in \mathcal{M}$ such that for each $m \in \mathcal{M}$, $-m$ is the unique element of $\mathcal{M}$ satisfying $R(m, -m, \infty)$. We extend $-$ pointwise to $\mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ by $-N =_{def} \{-n \mid n \in N\}$.*

The forcing relation between elements of $\mathcal{M}$ and CBI-formulae extends that for BBI (end of Section 2.1.2) with the following clauses for the new logical connectives [17]:

$$
\begin{array}{ll}
m \Vdash \bot^* \text{ iff } m \neq \infty & m \Vdash \sim A \text{ iff } -m \nVdash A \\
m \Vdash A \vee^* B \text{ iff } \forall a, b.(R(a, b, -m) \text{ implies } -a \Vdash A \text{ or } -b \Vdash B) &
\end{array}
$$

The truth and validity of CBI formulae are defined similar to those for BBI. That is, given a CBI model $(\mathcal{M}, R, \epsilon, -, \infty, v)$, a CBI formula A is true at $m \in \mathcal{M}$ if $m \Vdash A$; it is valid if it is true at every world in every CBI model.

CBI has a more complete set of logical connectives than other BI logics. This could be convenient when designing a proof system with an emphasis on duality of logical connectives, such as the ones that will be shown later in Chapter 3. However, the semantics of CBI is more involved than that of BBI, and some elements in the semantics, such as $-$ and ∞, cannot be interpreted in some resource models. A typical computational model based on CBI is the finance model [17], where we have positive resource as "the money I own" and negative resource as "the money I owe". In this setting, $\mathcal{M}$ is the set of integers Z, $R(i,j,k)$ iff $i + j = k$, ϵ is 0, $-$ is just the negative sign on integers, and ∞ is also 0.

2.2 Hilbert Systems for BI Logics

The additive part of BI logics is either intuitionistic or classical propositional logic, whose Hilbert system can be extended to obtain axiomatisation for the corresponding BI logic. For this reason, we first give the Hilbert systems for intuitionistic propositional logic à la Troelstra and Schwichtenberg [93], as shown in Figure 2.2.

Axioms:

$A \to (B \to A)$ $\quad A \to A \vee B$ $\quad B \to A \vee B$

$A \wedge B \to A$ $\quad A \wedge B \to B$ $\quad A \to (B \to (A \wedge B))$

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

$(A \to C) \to ((B \to C) \to (A \vee B \to C))$ $\quad \bot \to A$

Deduction Rules:

$$\frac{\vdash A \qquad \vdash A \to B}{\vdash B}\ MP$$

Figure 2.2: The Hilbert system for intuitionistic propositional logic.

The axioms and deduction rules in a Hilbert system should be seen as schemata. For example, we can uniformly replace A by a formula when using the above axioms and deduction rule. When $\neg A$ is understood as $A \to \bot$, the Hilbert system for classical propositional logic can be obtained by adding to Figure 2.2 the axiom of *double negation elimination*:

$$\neg\neg A \to A.$$

Alternatively, one can also add *the law of excluded middle* $A \vee \neg A$ or add *Peirce's law* $((A \to B) \to A) \to A$ to the axioms in Figure 2.2 to get the Hilbert system for classical propositional logic.

The Hilbert system for BI and BBI respectively consist of the axioms and rules for intuitionistic and classical propositional logic for the additive fragment, and additional

axioms and rules for the multiplicative fragment. For the latter, we use the axiomatisation given in [36], and listed in Figure 2.3. The axioms and rules in Figure 2.3 are exactly those of multiplicative intuitionistic linear logic (MILL).

Axioms

$A \to (\top^* * A)$

$(\top^* * A) \to A$

$(A * B) \to (B * A)$

$(A * (B * C)) \to ((A * B) * C)$

Deduction Rules

$$\frac{\vdash A \to C \qquad \vdash B \to D}{\vdash (A * B) \to (C * D)} *$$

$$\frac{\vdash A \to (B \mathrel{-\!\!*} C)}{\vdash (A * B) \to C} \mathrel{-\!\!*} 1 \qquad \frac{\vdash (A * B) \to C}{\vdash A \to (B \mathrel{-\!\!*} C)} \mathrel{-\!\!*} 2$$

Figure 2.3: Some axioms and rules for the Hilbert system for BI and BBI.

Although we may use CBI in the sequel, its Hilbert axiomatisation is not particularly important in this dissertation. In short, Pym suggested adding the axiom

$$(A \mathrel{-\!\!*} \bot^*) \mathrel{-\!\!*} \bot^* \to A$$

but the Hilbert system of CBI is not discussed in much detail [84]. The Hilbert system for CBI is obtained by adding the axioms for multiplicative classical linear logic (MLL) to the Hilbert system for BBI. Interested readers may also take the translation between CBI and a modal logic given by Brotherston and Calcagno [17] and reverse engineer their Hilbert axiomatisation for the corresponding modal logic.

2.3 Display Calculi for BI Logics

The display calculi we discuss in this section are given by Brotherston [16], who first proposed a display calculus for CBI with Calcagno [17], and then extended their result to a unified display calculus framework for the four BI logics mentioned in Section 2.1. Their methodology will be used when we try to give nested sequent calculi for BI logics in Chapter 3. That is, we first aim at a system for the logic with the largest set of logical connectives, then work backwards to trim the system down for logics with fewer connectives.

Generally speaking, a display calculus also has a sequent-like structure, which is called a *consecution* in some literature, but here we shall just stay with the word *sequent*. *Structures* range over capital Greek letters $\Gamma, \Delta, ...$, and a sequent is in the form $\Gamma \vdash \Delta$. As in a sequent calculus, we will have structural connectives, but not limited to the ones for the unit, conjunction and disjunction, we also need the ones for negation and implication, as shown below [16]:

Structural connectives:

Additive	Multiplicative	Arity	Antecedent meaning	Succedent meaning
$\varnothing_a$	$\varnothing_m$	0	truth	falsity
$\sharp$	$\flat$	1	negation	negation
;	,	2	conjunction	disjunction
$\Rightarrow$	$-\!\circ$	2	undefined	implication

We assume that unary structural connectives bind tighter than binary structural connectives. A structure Γ in a sequent may consist of either a formula or (sub)structure(s) connected by a structural connective. As usual, we call the top-level structural connective in a structure the *main structural connective*. Neither $\Rightarrow$ nor $-\!\circ$ is allowed to appear as the main structural connective of an antecedent in a sequent since their meaning in that case is undefined.

In a display calculus, there are logical rules to manipulate logical connectives, structural rules to generate or remove structures, and also display postulates to move structures across the turnstile and display or undisplay structures. Display postulates are written as $S <>_D S'$ where S, S' are sequents.

The first group of rules, called DL_{IL}, are for the additive part of BI (i.e., intuitionistic logic), as shown in Figure 2.4.

Antecedent structural connectives: $\varnothing_a \quad ;$
Succedent structural connectives: $\Rightarrow$
Display postulates: $\Gamma_1;\Gamma_2 \vdash \Delta <>_D \Gamma_1 \vdash \Gamma_2 \Rightarrow \Delta <>_D \Gamma_2;\Gamma_1 \vdash \Delta$
Logical rules:

$$\frac{}{\bot \vdash \Delta}\,{\scriptstyle \bot L} \qquad \frac{\varnothing_a \vdash \Delta}{\top \vdash \Delta}\,{\scriptstyle \top L} \qquad \frac{}{\Gamma \vdash \top}\,{\scriptstyle \top R} \qquad \frac{A;B \vdash \Delta}{A \wedge B \vdash \Delta}\,{\scriptstyle \wedge L}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,{\scriptstyle \wedge R} \qquad \frac{A \vdash \Delta \qquad B \vdash \Delta}{A \vee B \vdash \Delta}\,{\scriptstyle \vee L} \qquad \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \vee A_2}\,{\scriptstyle \vee R,\, i \in \{1,2\}}$$

$$\frac{\Gamma \vdash A \qquad B \vdash \Delta}{A \to B \vdash \Gamma \Rightarrow \Delta}\,{\scriptstyle \to L} \qquad \frac{\Gamma;A \vdash B}{\Gamma \vdash A \to B}\,{\scriptstyle \to R}$$

Structural rules:

$$\frac{\varnothing_a;\Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,{\scriptstyle \varnothing_a L} \qquad \frac{\Gamma_1;(\Gamma_2;\Gamma_2) \vdash \Delta}{(\Gamma_1;\Gamma_2);\Gamma_3 \vdash \Delta}\,{\scriptstyle AAL} \qquad \frac{\Gamma_1 \vdash \Delta}{\Gamma_1;\Gamma_2 \vdash \Delta}\,{\scriptstyle WL} \qquad \frac{\Gamma;\Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,{\scriptstyle CL}$$

Figure 2.4: Inference rules in DL_{IL}.

For the additive part of BBI, we extend the rules in Figure 2.4 with the rules in Figure 2.5, except that the $\vee R$ and $\to L$ rule in Figure 2.4 are replaced with the corresponding ones in Figure 2.5. We call the resultant system DL_{CL}.

The inference rules for the multiplicative part of BI and BBI are called DL_{LM} and given in Figure 2.6.

Finally, the multiplicative part of CBI is captured by DL_{dMM}, which extends the

Antecedent structural connectives: $\emptyset_a \quad \sharp \quad ;$
Succedent structural connectives: $\emptyset_a \quad \sharp \quad ;$
Display postulates:

$$\Gamma_1;\Gamma_2 \vdash \Delta <>_D \Gamma_1 \vdash \sharp\Gamma_2;\Delta <>_D \Gamma_2;\Gamma_1 \vdash \Delta$$

$$\Gamma \vdash \Delta_1;\Delta_2 <>_D \Gamma;\sharp\Delta_1 \vdash \Delta_2 <>_D \Gamma \vdash \Delta_2;\Delta_1$$

$$\Gamma \vdash \Delta <>_D \sharp\Delta \vdash \sharp\Gamma <>_D \sharp\sharp\Gamma \vdash \Delta$$

Logical rules:

$$\frac{\Gamma \vdash \emptyset_a}{\Gamma \vdash \bot}\ {\scriptstyle \bot R} \qquad \frac{\sharp A \vdash \Delta}{\neg A \vdash \Delta}\ {\scriptstyle \neg L} \qquad \frac{\Gamma \vdash \sharp A}{\Gamma \vdash \neg A}\ {\scriptstyle \neg R}$$

$$\frac{\Gamma \vdash A \qquad B \vdash \Delta}{A \to B \vdash \sharp\Gamma;\Delta}\ {\scriptstyle \to L} \qquad \frac{\Gamma \vdash A;B}{\Gamma \vdash A \vee B}\ {\scriptstyle \vee R}$$

Structural rules:

$$\frac{\Gamma \vdash \Delta;\emptyset_a}{\Gamma \vdash \Delta}\ {\scriptstyle \emptyset_a R}$$

Figure 2.5: Some Inference rules in DL_{CL}.

Antecedent structural connectives: $\emptyset_m \quad ,$
Succedent structural connectives: $-\!\circ$
Display postulates: $\Gamma_1,\Gamma_2 \vdash \Delta <>_D \Gamma_1 \vdash \Gamma_2 -\!\circ \Delta <>_D \Gamma_2,\Gamma_1 \vdash \Delta$

Logical rules:

$$\frac{\emptyset_m \vdash \Delta}{\top^* \vdash \Delta}\ {\scriptstyle \top^* L} \qquad \frac{A,B \vdash \Delta}{A * B \vdash \Delta}\ {\scriptstyle * L} \qquad \frac{\Gamma \vdash A \qquad B \vdash \Delta}{A -\!\!* B \vdash \Gamma -\!\circ \Delta}\ {\scriptstyle -\!\!* L}$$

$$\frac{}{\emptyset_m \vdash \top^*}\ {\scriptstyle \top^* R} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1,\Gamma_2 \vdash A * B}\ {\scriptstyle * R} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A -\!\!* B}\ {\scriptstyle -\!\!* R}$$

Structural rules:

$$\frac{\emptyset_m,\Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ {\scriptstyle \emptyset_m L} \qquad \frac{\Gamma_1,(\Gamma_2,\Gamma_2) \vdash \Delta}{(\Gamma_1,\Gamma_2),\Gamma_3 \vdash \Delta}\ {\scriptstyle MAL}$$

Figure 2.6: Inference rules in DL_{LM}

rules in Figure 2.6 with the rules in Figure 2.7, except that the rule $-\!\!* L$ is replaced by corresponding rule in Figure 2.7.

Thus we obtain the display calculi for BI, BBI, dMBI, and CBI respectively from $DL_{IL} + DL_{LM}$, $DL_{CL} + DL_{LM}$, $DL_{IL} + DL_{dMM}$ and $DL_{CL} + DL_{dMM}$. These display calculi are sound and complete w.r.t. the semantics or Hilbert system given in previous sections for corresponding logics. Moreover, Brotherston's display calculi for BI logics inherit the advantage from Belnap's Display Logic [5] in that cut-admissibility can be easily checked as long as the eight conditions for display calculi are met [16].

The inference rules in a display calculus can only be applied to top level structures, if one wants to apply a rule to a substructure, one has to "display" that substructure on the top level by using display postulates. Thus Belnap [5] originally showed a *display theorem* which ensures that any substructure in a sequent can be displayed. The same property also holds for Brotherston's display calculi.

Although display calculi are powerful in theory, proof search in such systems is often not practical because the display postulates can be used at any time to shuffle

Antecedent structural connectives: $\varnothing_m \quad \flat \quad ,$
Succedent structural connectives: $\varnothing_m \quad \flat \quad ,$
Display postulates:

$$\Gamma_1, \Gamma_2 \vdash \Delta <>_D \Gamma_1 \vdash \flat\Gamma_2, \Delta <>_D \Gamma_2, \Gamma_1 \vdash \Delta$$

$$\Gamma \vdash \Delta_1, \Delta_2 <>_D \Gamma, \flat\Delta_1 \vdash \Delta_2 <>_D \Gamma \vdash \Delta_2, \Delta_1$$

$$\Gamma \vdash \Delta <>_D \flat\Delta \vdash \flat\Gamma <>_D \flat\flat\Gamma \vdash \Delta$$

Logical rules:

$$\frac{}{\bot^* \vdash \varnothing_m}\ {\bot^* L} \qquad \frac{\Gamma \vdash \varnothing_m}{\Gamma \vdash \bot^*}\ {\bot^* R} \qquad \frac{\flat A \vdash \Delta}{\sim A \vdash \Delta}\ {\neg L} \qquad \frac{\Gamma \vdash \flat A}{\Gamma \vdash \sim A}\ {\neg R}$$

$$\frac{\Gamma \vdash A \qquad B \vdash \Delta}{A -\!\!* \ B \vdash \flat\Gamma, \Delta}\ {-\!\!* L} \qquad \frac{A \vdash \Delta_1 \qquad B \vdash \Delta_2}{A \vee^* B \vdash \Delta_1, \Delta_2}\ {\vee^* L} \qquad \frac{\Gamma \vdash A, B}{\Gamma \vdash A \vee^* B}\ {\vee^* R}$$

Structural rules:

$$\frac{\Gamma \vdash \Delta, \varnothing_m}{\Gamma \vdash \Delta}\ {\varnothing_m R}$$

Figure 2.7: Some Inference rules in DL_{dMM}.

structures in a sequent, generating redundant proofs. In Chapter 3 we continue from Brotherston's display calculi and discover more effective ways for proof search.

2.4 Other BI variants and their applications

So far we have briefly discussed BI, BBI, CBI, their semantics and proof theory. There are many other variants, mainly based on BBI and come from practical computational models. The BBI semantics are based on non-deterministic commutative monoids (cf. Section 2.1.2), the monoids can also be partial-deterministic or total-deterministic to give BBI_{PD} and BBI_{TD} respectively. We shall sometimes call the original BBI as BBI_{ND} to distinguish it from other BBI variants. As expected, partial-determinism forces that

$$\forall a, b, c, d \in \mathcal{M}.\ c \in a \circ b \text{ and } d \in a \circ b \text{ implies } c = d.$$

Totality can be defined on top of partial-determinism as

$$\forall a, b \in \mathcal{M}.\ \exists c \in \mathcal{M}\ s.t.\ a \circ b = c.$$

A number of more application-oriented properties can be added to BBI to form even more specific logics, which will be discussed later in Section 5.2.

Unfortunately, among the logics we discussed in this chapter, only BI is decidable [38], BBI_{ND}, BBI_{PD}, BBI_{TD}, CBI are all undecidable [20, 62].

Nevertheless, a majority of applications of bunched logics are based on the undecidable BBI, more specifically, on BBI_{PD}. The most famous example might be separation logic [88, 54, 76], which inspired us to start a journey on proof methods and automated reasoning for these logics. Of course, there are computational models

based on BI or other variants, Pym discussed the following models based on BI in his book [84]: logic programming, interference and non-interference in imperative programming, Petri nets, CCS-like models, and pointer models. The latter is actually the prototype of separation logic, proposed by Reynolds, Ishtiaq, O'Hearn, et al. before the word "separation logic" was settled [76]. The intuitionistic nature of BI implies that if a part of memory makes an assertion true, any extension of that part of memory will also make the assertion true. This monotonicity property makes the early version of separation logic less expressive than the one based on BBI. In fact, the former logic can be translated into the latter version [88].

Besides BI and BBI, Brotherston et al. worked actively on CBI and proposed various possible applications of it [17], including Abelian groups, effect algebras, languages, generalised heaps, bit arithmetic, integer modulo arithmetic, syntactic models, deny-guarantee model etc., and the finance model we covered at the end of Section 2.1.3.

Chapter 3

Nested Sequents and Deep Inference for BI Logics

BI logics have many applications in computer science. Our target is one of the most important ones: separation logic. The assertion logic of separation logic is based on either BI or BBI, thus proof search for BI logics is the key to support precondition strengthening and postcondition weakening in separation logic. We have given an overview of BI logics, their semantics and proof theory in the previous chapter. The latter includes Hilbert systems and display calculi. However, neither of these two methods are efficient for proof search. The case for Hilbert systems is well-known, and for display calculi, the display postulates and explicit structural rules are both responsible for generating large and redundant structures in proof search. Therefore despite the proof theoretical importance of Hilbert systems and display calculi, there is a need for other proof methods for BI logics that are more proof-search-friendly.

This chapter takes the same avenue as Postniece's work on nested sequent calculi [83]. That is, we start from Brotherston's display calculi [16] and convert them into shallow inference systems with nested sequents, then we absorb the display rules and structural rules by allowing the logical rules to be applied on any nested sequent at any level and using propagation rules to transfer structures between different nested sequents, giving a deep inference system. Also inspired by Goré et al.'s nested sequent calculus, we decide to take a detour by working on CBI instead of BBI, because the nested sequent structure and inference rules in their calculi reply on the residuation of logical and structural connectives. In this sense CBI, with a more complete set of logical connectives, is easier to work with. The hope is that an inference system for CBI, or for an extended logic with even more connectives, can be trimmed down to handle logics with fewer connectives such as BBI. For this we need to show a "separation property" [44] to come back to our original goal towards proof search for BBI. The reader may be aware of another work on nested sequents for BBI by Park et al. [77], which was developed roughly at the same time as our work in this chapter, but we were not aware of that work until we had proceeded into the next chapter. Therefore we could not have started our work by adapting Park et al.'s nested sequents. In fact,

their nested sequents are in a very different style compared to ours.

Section 3.1 gives an overview of nested sequent calculus, shallow and deep inference, followed by Section 3.2 where we give a shallow nested sequent calculus for CBI. We discuss a deep inference system for CBI in Section 3.3, which unfortunately is not complete w.r.t. the display calculi for CBI. Section 3.4 concludes this Chapter with a discussion of Park et al.'s nested sequent system that is sound and complete w.r.t. the Hilbert system for BBI.

3.1 Nested Sequents with Shallow and Deep Inference

We discuss here the general idea of nested sequent calculus, shallow and deep inference systems without going to full details, taking Postniece's work on bi-intuitionistic logic as an example [83]. Bi-intuitionistic logic (BiInt) is intuitionistic logic plus dual-intuitionistic logic, thus every logical connective in BiInt has a dual connective. For example, $\wedge$ is dual to $\vee$, $\top$ is dual to $\bot$, and there are two less commonly seen connectives $-\!\!<$ and $\sim$, respectively called "exclusion" and "paraconsistent negation", dual to $\rightarrow$ and $\neg$. As we would expect, $-\!\!<$ has shallow nested sequent rules dual to (intuitionistic) $\rightarrow$:

$$\frac{A \vdash B, Y}{X, A -\!\!< B \vdash Y} -\!\!<_L \qquad \frac{X \vdash A, Y \qquad X, B \vdash Y}{X \vdash A -\!\!< B, Y} -\!\!<_R$$

where we write A, B for formulae, X, Y and later W for structures defined below[1]:

$$X ::= \emptyset \mid A \mid (X, X) \mid X \triangleright X$$

The $-\!\!< L$ rule discards the context X in the antecedent of the conclusion when viewed "backwards". The paraconsistent negation can be defined by $\sim A = \top -\!\!< A$. The shallow nested sequent calculus $LBiInt_1$ for BiInt [83] has an additional structural connective $\triangleright$ compared to the normal sequent calculus LJ for Int (cf. Section 1.2.1). This structural connective mimics the effect of the turnstile $\vdash$, which on the top level is a proxy for implication $\rightarrow$, therefore making the sequent into *nested sequents*. That is, any substructure $(X' \triangleright Y')$ in a sequent can be deemed a nested sequent, a sequent $X \vdash Y$ is then a tree where $\vdash$ is the top level turnstile, and $\triangleright$ connectives are lower level turnstiles. However, since $\triangleright$ may appear on both sides of a sequent, it would not completely make sense if we did not have the dual connective $-\!\!<$ to $\rightarrow$. That is, $\triangleright$ in the antecedent is a structural proxy for $-\!\!<$, while in the succedent is a proxy for $\rightarrow$. Although having nested sequents, as a *shallow inference system*, $LBiInt_1$ only allows us to apply inference rules on the top-level sequent, which is why we distinguish the top-level turnstile from lower level ones. If we want to manipulate lower level structures,

[1] Courtesy of Postniece's notation, which is slightly different from sequent calculi we present in other chapters of this dissertation where we use $\Gamma, \Delta, \cdots$ for structures.

we have to "display" them to the top-level as in display calculi. This operation is called "zoom-in" in the nested sequent calculus. The shallow system $LBiInt_1$ has the following "display rules":

$$\frac{(X_1 \rhd Y_1), X_2 \vdash Y_2}{X_1, X_2 \vdash Y_1, Y_2}\, s_L \qquad \frac{X_1 \vdash Y_1, (X_2 \rhd Y_2)}{X_1, X_2 \vdash Y_1, Y_2}\, s_R$$

$$\frac{X_2 \vdash Y_2, Y_1}{X_2 \rhd Y_2 \vdash Y_1}\, \rhd_L \qquad \frac{X_1, X_2 \vdash Y_2}{X_1 \vdash X_2 \rhd Y_2}\, \rhd_R$$

The above rules make the shallow system essentially the same as Goré's display calculus for BiInt [43], but with a few minor differences such as built-in structural rules.

In nested sequent calculi it is often important to distinguish the *polarity* of contexts. Roughly speaking, a *simple* context is a structure with a hole $[]$, denoted by $\Sigma[]$, which is not under the scope of any $\rhd$. If $\Sigma[]$ is on the left hand side of the nearest $\rhd$ (i.e., the hole is to the left of the current level turnstile), it is a *negative* context; occurring on the right hand side of the nearest $\rhd$ makes $\Sigma[]$ *positive*. Associating $\Sigma[]$ and other structures with "," does not change polarity, neither does further nesting in $\rhd$. A positive context is written as $\Sigma^+[]$ and a negative context is $\Sigma^-[]$. We will see formal definition of these notions in later sections.

Postniece then presented another shallow system called $LBiInt_2$ in which all the structural rules except for $\rhd_L$ and $\rhd_R$ are absorbed into logical rules. Based on these shallow inference systems, a more interesting result is Postniece's *deep inference system* $DBiInt$, which allows us to apply rules at any level of the sequent. Moreover, $DBiInt$ has four *propagation rules* that copy formulae from a nested sequent to another, as shown below:

$$\frac{\Sigma^-[A, (A, X \rhd Y)]}{\Sigma^-[A, X \rhd Y]}\, \rhd L_1 \qquad \frac{\Sigma^+[(X \rhd Y, A), A]}{\Sigma^+[X \rhd Y, A]}\, \rhd R_1$$

$$\frac{\Sigma[A, X \rhd (W, (A, Y \rhd Z))]}{\Sigma[A, X \rhd (W, (Y \rhd Z))]}\, \rhd L_2 \qquad \frac{\Sigma[((X \rhd Y, A), W) \rhd Z, A]}{\Sigma[((X \rhd Y), W) \rhd Z, A]}\, \rhd R_2$$

The rule $\rhd L_1$ copies a formula A on the left hand side of $\rhd$ outside to a higher level sequent, the nested sequent to be applied on must be in a negative hole. The rule $\rhd R_1$ does exactly the opposite. The rule $\rhd L_2$ copies a formula A on the left hand side of $\rhd$ to the left hand side of a inner (lower level) nested sequent, mirrored by the rule $\rhd R_2$.

The deep inference system $DBiInt$ is sound and complete w.r.t. the shallow inference systems, which are shown sound and complete for BiInt. However, naive proof search in $DBiInt$ is not terminating, thus Postniece [83] went on the give a variant called $DBiInt_1$ which gives terminating and more efficient proof search.

3.2 SI_{CBI}: A Shallow Inference System for CBI

This section and the next one discuss an attempt following Postniece's route. We use Brotherston's display calculus for CBI as the base system and first convert it to a shallow nested sequent calculus, then a deep one. As mentioned before, we choose CBI because it has more multiplicative connectives, our plan is to show that a sound and complete deep system for CBI can be trimmed down to a sound and complete system for BBI etc.. However, the logical connectives in CBI are still not enough for the duality we require, so we have to add "exclusion" connectives for both the additive part and the multiplicative part as in BiInt but without adding extra expressive power.

Unlike the nested sequent calculi for BiInt, we now need to consider different flavours of turnstiles, because we have an additive implication and a multiplicative implication. As a result, we use $\vdash_a$ or $\triangleright_a$ as the turnstiles in an additive sequent where only ";" and $\triangleright_m$ are allowed to connect structures, and use $\vdash_m$ or $\triangleright_m$ as the turnstile in a multiplicative sequent where only "," and $\triangleright_a$ are allowed to connect structures. Our nested sequents form a tree in which the levels have interleaving flavour of turnstiles. A foreseeable advantage of this type of nested sequents is that we can isolate additive reasoning from multiplicative reasoning, thus it is possible to use the state-of-the-art first-order provers to reason about the additive sequents, and use some specific method to reason about the multiplicative sequents.

3.2.1 Inference Rules in SI_{CBI}

Recall the syntax of CBI from Section 2.1.3:

$$A ::= p \mid \top \mid \bot \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \\ \top^* \mid \bot^* \mid \sim A \mid A * A \mid A \vee^* A \mid A -\!\!* A$$

where p is an atomic proposition, we may use A, B, F as arbitrary formula. We now add two more logical connectives to the above definition, respectively the additive exclusion $-\!\!<$ and multiplicative exclusion $-\!\!\times$. The additive exclusion $A -\!\!< B$ is defined as $A \wedge \neg B$, and $A -\!\!\times B$ is defined as $A * \sim B$. Note that unlike in BiInt, we do not have "paraconsistent negation" since both parts of CBI are classical and the paraconsistent negations collapse with the normal negations.

The structures in our nested sequent $X_1 \vdash_a X_2$ (or $Y_1 \vdash_m Y_2$) are defined as below, where X is an additive structure and Y is a multiplicative structure, A is a CBI formula:

$$X ::= \varnothing_a \mid A \mid X; X \mid Y \triangleright_m Y$$
$$Y ::= \varnothing_m \mid A \mid Y, Y \mid X \triangleright_a X$$

Note that , and ; work as multiset union and bind tighter than $\triangleright_a$ and $\triangleright_m$, which are effectively the same as $\vdash_a$ and $\vdash_m$ respectively, but we write them in different styles to

emphasise that $\vdash_a$ and $\vdash_m$ are the top level turnstiles. The structural connectives are proxies for some logical connectives, thus we can translate structures into formulae as in Figure 3.1.

$$\begin{array}{cc}
\tau(X_1 \vdash_a X_2) = \tau^-(X_1) \to \tau^+(X_2) & \tau(Y_1 \vdash_m Y_2) = \tau^-(Y_1) -\!\!* \, \tau^+(Y_2) \\
\tau^-(\varnothing_a) = \top & \tau^+(\varnothing_a) = \bot \\
\tau^-(\varnothing_m) = \top^* & \tau^+(\varnothing_m) = \bot^* \\
\tau^-(A) = A & \tau^+(A) = A \\
\tau^-(X_1; X_2) = \tau^-(X_1) \wedge \tau^-(X_2) & \tau^+(X_1; X_2) = \tau^+(X_1) \vee \tau^+(X_2) \\
\tau^-(Y_1, Y_2) = \tau^-(Y_1) * \tau^-(Y_2) & \tau^+(Y_1, Y_2) = \tau^+(Y_1) \vee^* \tau^+(Y_2) \\
\tau^-(X_1 \triangleright_a X_2) = \tau^-(X_1) -\!\!< \tau^+(X_2) & \tau^+(X_1 \triangleright_a X_2) = \tau^-(X_1) \to \tau^+(X_2) \\
\tau^-(Y_1 \triangleright_m Y_2) = \tau^-(Y_1) >\!\!- \tau^+(Y_2) & \tau^+(Y_1 \triangleright_m Y_2) = \tau^-(Y_1) -\!\!* \, \tau^+(Y_2)
\end{array}$$

Figure 3.1: Structure translation of nested sequents

The rotation of the sequent tree to move a sequent to the root node is exactly the "zooming-in" operation on a substructure within a nested sequent or the "display" operation in display calculi. A *context* is a structure with a hole $[]$. We write $\Sigma[X]$ for the structure obtained by filling the hole $[]$ in the context $\Sigma[]$ with a structure X. A *simple context* is then defined as follows, where Z is a structure either additive or multiplicative:

$$\Sigma[] ::= [] \mid \Sigma[], (Z) \mid (Z), \Sigma[] \mid \Sigma[]; (Z) \mid (Z); \Sigma[]$$

The polarity is defined à la Postniece [83]. The hole in a simple context is never in the scope of $\triangleright$, and is neutral. Positive and negative contexts are defined non-traditionally as any further nesting of $\triangleright$ **does not** change polarity. We write $\Sigma^-[]$ to indicate that $\Sigma[]$ is negative and $\Sigma^+[]$ is positive. The details are as follows:

- If $\Sigma[]$ is a simple context then $\Sigma[] \triangleright Z$ is a negative context and $Z \triangleright \Sigma[]$ is a positive context.
- If $\Sigma[]$ is a positive/negative context then so are $(\Sigma[], Z)$, $(Z, \Sigma[])$, $(\Sigma[]; Z)$, $(Z; \Sigma[])$, $\Sigma[] \triangleright Z$, and $Z \triangleright \Sigma[]$.

For example, the context $([] \triangleright Y) \triangleright Y'$ is negative and $(X \triangleright []) \triangleright Y$ is positive.

The shallow inference nested sequent calculus for CBI is shown in Figure 3.2 and Figure 3.3. We refer to this calculus as SI_{CBI}. We write A, B for formulae and X and Y possibly with subscripts are respectively additive and multiplicative structures. Notice that the additive (resp. multiplicative) structural unit $\varnothing_a$ (resp. $\varnothing_m$) is assumed in our system, so the rules for them are neglected. That is, we assume that $X; \varnothing_a \equiv X$ and $Y, \varnothing_m \equiv Y$. Similarly, we do not explicitly state associativity and commutativity rules for both structures as they are built in. Our presentation of the rules might be verbose in the sense that some logical connectives can be defined from others, but we show the

Identity and cut:

$$\frac{}{X_1; A \vdash_a X_2; A}\, id_a \qquad \frac{X_1 \vdash_a X_2; A \qquad A; X_3 \vdash_a X_4}{X_1; X_3 \vdash_a X_2; X_4}\, cut_a$$

$$\frac{}{A \vdash_m A}\, id_m \qquad \frac{Y_1 \vdash_m Y_2, A \qquad A, Y_3 \vdash_m Y_4}{Y_1, Y_3 \vdash_m Y_2, Y_4}\, cut_m$$

Logical rules:

$$\frac{X_1 \vdash_a X_2}{X_1; \top \vdash_a X_2}\, \top L \qquad \frac{}{\varnothing_a \vdash_a \top}\, \top R$$

$$\frac{Y_1 \vdash_m Y_2}{Y_1, \top^* \vdash_m Y_2}\, \top^* L \qquad \frac{}{\varnothing_m \vdash_m \top^*}\, \top^* R$$

$$\frac{}{\bot \vdash_a \varnothing_a}\, \bot L \qquad \frac{X_1 \vdash_a X_2}{X_1 \vdash_a \bot; X_2}\, \bot R$$

$$\frac{}{\bot^* \vdash_m \varnothing_m}\, \bot^* L \qquad \frac{Y_1 \vdash_m Y_2}{Y_1 \vdash_m \bot^*, Y_2}\, \bot^* R$$

$$\frac{X_1; A_i \vdash_a X_2}{X_1; A_1 \wedge A_2 \vdash_a X_2}\, \wedge L, i \in \{1,2\} \qquad \frac{X_1 \vdash_a A; X_2 \qquad X_3 \vdash_a B; X_4}{X_1; X_3 \vdash_a A \wedge B; X_2; X_4}\, \wedge R$$

$$\frac{X_1; A \vdash_a X_2 \qquad X_3; B \vdash_a X_4}{X_1; X_3; A \vee B \vdash_a X_2; X_4}\, \vee L \qquad \frac{X_1 \vdash_a A_i; X_2}{X_1 \vdash_a A_1 \vee A_2; X_2}\, \vee R, i \in \{1,2\}$$

$$\frac{X_1 \vdash_a A; X_2 \qquad X_3; B \vdash_a X_4}{X_1; X_3; A \to B \vdash_a X_2; X_4}\, \to L \qquad \frac{X_1; A \vdash_a B; X_2}{X_1 \vdash_a A \to B; X_2}\, \to R$$

$$\frac{X_1; A \vdash_a B; X_2}{X_1; A \prec B \vdash_a X_2}\, \prec L \qquad \frac{X_1 \vdash_a A; X_2 \qquad X_3; B \vdash_a X_4}{X_1; X_3 \vdash_a A \prec B; X_2; X_4}\, \prec R$$

$$\frac{Y_1, A, B \vdash_m Y_2}{Y_1, A * B \vdash_m Y_2}\, *L \qquad \frac{Y_1 \vdash_m A, Y_2 \qquad Y_3 \vdash_m B, Y_4}{Y_1, Y_3 \vdash_m A * B, Y_2, Y_4}\, *R$$

$$\frac{Y_1, A \vdash_m Y_2 \qquad Y_3, B \vdash_m Y_4}{Y_1, Y_3, A \vee^* B \vdash_m Y_2, Y_4}\, \vee^* L \qquad \frac{Y_1 \vdash_m A, B, Y_2}{Y_1 \vdash_m A \vee^* B, Y_2}\, \vee^* R$$

$$\frac{Y_1 \vdash_m A, Y_2 \qquad Y_3, B \vdash_m Y_4}{Y_1, Y_3, A \mathrel{-\!\!*} B \vdash_m Y_2, Y_4}\, \mathrel{-\!\!*} L \qquad \frac{Y_1, A \vdash_m B, Y_2}{Y_1 \vdash_m A \mathrel{-\!\!*} B, Y_2}\, \mathrel{-\!\!*} R$$

$$\frac{Y_1, A \vdash_m B, Y_2}{Y_1, A \mathrel{-\!\!\times} B \vdash_m Y_2}\, \mathrel{-\!\!\times} L \qquad \frac{Y_1 \vdash_m A, Y_2 \qquad Y_3, B \vdash_m Y_4}{Y_1, Y_3 \vdash_m A \mathrel{-\!\!\times} B, Y_2, Y_4}\, \mathrel{-\!\!\times} R$$

Figure 3.2: Shallow inference nested sequent calculus SI_{CBI} for CBI part 1.

Structural rules:

$$\frac{X_1 \vdash_a X_2}{X_1; X_3 \vdash_a X_2}\, WL \qquad \frac{X_1 \vdash_a X_2}{X_1 \vdash_a X_3; X_2}\, WR$$

$$\frac{X_1; X_3; X_3 \vdash_a X_2}{X_1; X_3 \vdash_a X_2}\, CL \qquad \frac{X_1 \vdash_a X_2; X_3; X_3}{X_1 \vdash_a X_2; X_3}\, CR$$

$$\frac{\;X_1; A \rhd_m \varnothing_m \vdash_a X_2\;}{X_1; A \vdash_a X_2}\, T_aL \qquad \frac{\;X_1 \vdash_a \varnothing_m \rhd_m A; X_2\;}{X_1 \vdash_a A; X_2}\, T_aR$$

$$\frac{\;Y_1, A \rhd_a \varnothing_a \vdash_m Y_2\;}{Y_1, A \vdash_m Y_2}\, T_mL \qquad \frac{\;Y_1 \vdash_m \varnothing_a \rhd_a A, Y_2\;}{Y_1 \vdash_m A, Y_2}\, T_mR$$

Display rules:

$$\frac{\;X_1 \vdash_a X_2; (Y_1 \rhd_m Y_2)\;}{(X_1 \rhd_a X_2), Y_1 \vdash_m Y_2}\, D_1 \qquad \frac{\;Y_1 \vdash_m Y_2, (X_1 \rhd_a X_2)\;}{(Y_1 \rhd_m Y_2); X_1 \vdash_a X_2}\, D_2$$

$$\frac{\;X_1 \rhd_a X_3 \vdash_m X_2 \rhd_a X_4\;}{X_1; X_2 \vdash_a X_3; X_4}\, \rhd_1 \qquad \frac{\;Y_1 \rhd_m Y_3 \vdash_a Y_2 \rhd_m Y_4\;}{Y_1, Y_2 \vdash_m Y_3, Y_4}\, \rhd_2$$

Figure 3.3: Shallow inference nested sequent calculus SI_{CBI} for CBI part 2.

rules for them nonetheless for the sake of duality. We say a sequent is additive if the top turnstile is $\vdash_a$; it is multiplicative if the top turnstile is $\vdash_m$. We say a rule is additive if it only manipulates additive sequents. A rule is multiplicative if it only manipulates multiplicative sequents. Display rules are neither additive nor multiplicative because the conclusion and the premise in these rules may be either types of sequents.

We use double lines in some structural rules and display rules to denote that these rules are reversible. The display postulates are captured by the four display rules. We explicitly allow multiple structures on both side of turnstile, but by the definition of structures, these rules are equivalent to those in Brotherston's DL_{CBI}. Since we force that additive and multiplicative sequents must be interleaving, the structural rules and display rules are quite complicated to ensure the correct structure. The rules T_aL, T_aR, T_mL, T_mR deal with situations where the formula is multiplicative but the turnstile is additive (and vice versa), in which case we add units to make the formula the child of the current node. For example, $A * B$ is a multiplicative formula, but if it appears in $A * B \vdash_a X$, we cannot directly apply the $*L$ rule. We have to use the rule T_aL to encapsulate the formula into a multiplicative (inner) sequent $(A * B \rhd_m \varnothing_m) \vdash_a X$, display the inner sequent to get $A * B \vdash_m (\varnothing_m \rhd_a X)$, then apply the $*L$ rule.

In Figure 3.4 we use a simple example to illustrate how to prove the formula $(A * B) \to (B * A)$ using SI_{CBI}. In general, to prove a formula A, we start with the end sequent $\varnothing_a \vdash_a A$, and then apply the rules in SI_{CBI} backwards.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash_m A}\,id_m \quad \dfrac{}{B \vdash_m B}\,id_m}{A, B \vdash_m B * A}\,{*R}}{A, B \vdash_m (\varnothing_a \triangleright_a B * A)}\,T_m R}{A * B \vdash_m (\varnothing_a \triangleright_a B * A)}\,{*L}}{(A * B \triangleright_m \varnothing_m) \vdash_a B * A}\,D_2}{A * B \vdash_a B * A}\,T_a L}{\varnothing_a \vdash_a (A * B) \to (B * A)}\,{\to R}$$

Figure 3.4: An example derivation of $(A * B) \to (B * A)$ in SI_{CBI}.

3.2.2 Soundness of SI_{CBI}

We now provide a translation from SI_{CBI} structures to DL_{CBI} structures, and then we prove that the former is sound in regard to the latter. The detailed translation is shown in Figure 3.5 where $\tau()_D$ is the translation to display sequents, and $\tau()^-$, $\tau()^+$ are translations to formulae or structures. The following lemma states that if a sequent in SI_{CBI} is derivable, then the translated sequent in DL_{CBI} is also derivable.

$$\begin{array}{ll}
\tau(X_1 \vdash_a X_2)_D = \tau(X_1)^- \vdash \tau(X_2)^+ & \tau(Y_1 \vdash_m Y_2)_D = \tau(Y_1)^- \vdash \tau(Y_2)^+ \\
\tau(\varnothing_a)^* = \varnothing_a & \tau(\varnothing_m)^* = \varnothing_m \\
\tau(A)^* = A & \\
\tau(X_1; X_2)^* = \tau(X_1)^*; \tau(X_2)^* & \tau(Y_1, Y_2)^* = \tau(Y_1)^*, \tau(Y_2)^* \\
\tau(Y_1 \triangleright_m Y_2)^- = \tau(Y_1)^-, \flat\tau(Y_2)^+ & \tau(X_1 \triangleright_a X_2)^- = \tau(X_1)^-; \sharp\tau(X_2)^+ \\
\tau(Y_1 \triangleright_m Y_2)^+ = \flat\tau(Y_1)^-, \tau(Y_2)^+ & \tau(X_1 \triangleright_a X_2)^+ = \sharp\tau(X_1)^-; \tau(X_2)^+
\end{array}$$

The $*$ sign stands for either $+$ or $-$, and it is fixed in a certain translation.

Figure 3.5: Structure translation from SI_{CBI} to DL_{CBI}

Lemma 3.2.1. *If a sequent $X_1 \vdash_a X_2$ ($Y_1 \vdash_m Y_2$) is provable in SI_{CBI}, then there is a proof of $\tau(X_1 \vdash_a X_2)_D$ ($\tau(Y_1 \vdash_m Y_2)_D$) in DL_{CBI}.*

Proof. By induction on the depth of derivation.

(i) Base case is that for any sequents in SI_{CBI} that can be proved in one step, we can find a proof in DL_{CBI} for the translated sequent. Obviously the applicable rules in SI_{CBI} are id_a, id_m, $\top R$, $\top^* R$, $\bot L$, and $\bot^* L$, all of which are easy to prove by deriving our rules in DL_{CBI}. Now we give the detailed proofs.

The proof for id_a rule is as follows.

$$\begin{aligned}
\tau(conclusion)_D &= \tau(X_1; A \vdash_a X_2; A)_D = \tau(X_1; A)^- \vdash \tau(X_2; A)^+ \\
&= \tau(X_1)^-; \tau(A)^- \vdash \tau(X_2)^+; \tau(A)^+ = \tau(X_1)^-; A \vdash \tau(X_2)^+; A
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\,{\scriptstyle id}}{\tau(X_1)^-; A \vdash A}\,{\scriptstyle WL}}{\tau(X_1)^-; A \vdash \tau(X_2)^+; A}\,{\scriptstyle WR}$$

Note that the identity $A \vdash A$ is derived from Lemma 4.3 in [16]. The original *id* rule in DL_{CBI} uses atomic propositions only.

The case for id_m is easier.

$$\tau(conclusion)_D = \tau(A \vdash_m A)_D = \tau(A)^- \vdash \tau(A)^+ = A \vdash A$$

$$\dfrac{}{A \vdash A}\,{\scriptstyle id}$$

The cases for $\top R$, $\top^* R$, $\bot L$, and $\bot^* L$ are done by just translating, as shown sequentially below.

For $\top R$:

$$\tau(conclusion)_D = \tau(\varnothing_a \vdash_a \top)_D = \varnothing_a \vdash \top$$

$$\dfrac{}{\varnothing_a \vdash \top}\,{\scriptstyle \top R}$$

For $\top^* R$:

$$\tau(conclusion)_D = \tau(\varnothing_m \vdash_m \top^*)_D = \varnothing_m \vdash \top^*$$

$$\dfrac{}{\varnothing_m \vdash \top^*}\,{\scriptstyle \top^* R}$$

For $\bot L$:

$$\tau(conclusion)_D = \tau(\varnothing_a \vdash_a \bot)_D = \varnothing_a \vdash \bot$$

$$\dfrac{}{\varnothing_a \vdash \bot}\,{\scriptstyle \bot L}$$

For $\bot^* L$:

$$\tau(conclusion)_D = \tau(\varnothing_m \vdash_m \bot^*)_D = \varnothing_m \vdash \bot^*$$

$$\dfrac{}{\varnothing_m \vdash \bot^*}\,{\scriptstyle \bot^* L}$$

Therefore the base case holds.

(ii) The induction hypothesis is that if we can simulate all proofs in SI_{CBI} of depth n using DL_{CBI} for the translated sequents, then we can also simulate proofs of depth $n+1$ in SI_{CBI} for the translated sequents. The last steps of those proofs in SI_{CBI} can only be done by applying one of the rules (with non-empty premise) in Figure 3.2 and Figure 3.3. **Note that** the display postulates in DL_{CBI} imply commutativity of ; and ,. For simplicity, we will use this in the following text without explicitly stating it.

Now we show that if a rule in SI_{CBI} (below left) is applied in the last step, then we can find a proof of the form (below right) in DL_{CBI}.

$$\frac{premise}{conclusion}\,\rho \qquad\qquad \frac{\begin{array}{c}\tau(premise)_D\\ \vdots\end{array}}{\tau(conclusion)_D}$$

Since the induction hypothesis says we already have a proof of $\tau(premise)_D$, it is sufficient to show that we can derive $\tau(conclusion)_D$ from $\tau(premise)_D$. Therefore we have a proof in DL_{CBI} for the translated sequent when the $(n+1)$th applied rule is ρ.

First we show that the two cut rules in SI_{CBI} can be simulated in DL_{CBI}.

For cut_a:

$$\begin{aligned}\tau(premise_{left})_D &= \tau(X_1 \vdash_a X_2; A)_D = \tau(X_1)^- \vdash \tau(X_2)^+; A\\ \tau(premise_{right})_D &= \tau(A; X_3 \vdash_a X_4)_D = A; \tau(X_3)^- \vdash \tau(X_4)^+\\ \tau(conclusion)_D &= \tau(X_1; X_3 \vdash_a X_2; X_4)_D = \tau(X_1; X_3)^- \vdash \tau(X_2; X_4)^+\\ &= \tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+; \tau(X_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash \tau(X_2)^+; A}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A}\,AD2a \qquad \dfrac{A; \tau(X_3)^- \vdash \tau(X_4)^+}{A \vdash \sharp\tau(X_3)^-; \tau(X_4)^+}\,AD1a}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \sharp\tau(X_3)^-; \tau(X_4)^+}\,cut}{\tau(X_1)^-; \sharp\tau(X_2)^+; \tau(X_3)^- \vdash \tau(X_4)^+}\,AD1b}{\tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+; \tau(X_4)^+}\,AD2b$$

Similarly for cut_m:

$$\begin{aligned}\tau(premise_{left})_D &= \tau(Y_1 \vdash_m Y_2, A)_D = \tau(Y_1)^- \vdash \tau(Y_2)^+, A\\ \tau(premise_{right})_D &= \tau(A, Y_3 \vdash_m Y_4)_D = A, \tau(Y_3)^- \vdash \tau(Y_4)^+\\ \tau(conclusion)_D &= \tau(Y_1, Y_3 \vdash_m Y_2, Y_4)_D = \tau(Y_1, Y_3)^- \vdash \tau(Y_2, Y_4)^+\\ &= \tau(Y_1)^-, \tau(Y_3)^- \vdash \tau(Y_2)^+, \tau(Y_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash \tau(Y_2)^+, A}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A}\,MD2a \qquad \dfrac{A, \tau(Y_3)^- \vdash \tau(Y_4)^+}{A \vdash \flat\tau(Y_3)^-, \tau(Y_4)^+}\,MD1a}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash \flat\tau(Y_3)^-, \tau(Y_4)^+}\,cut}{\tau(Y_1)^-, \flat\tau(Y_2)^+, \tau(Y_3)^- \vdash \tau(Y_4)^+}\,MD1b}{\tau(Y_1)^-, \tau(Y_3)^- \vdash \tau(Y_2)^+, \tau(Y_4)^+}\,MD2b$$

Now we consider structural rules.

For WL:

$$\begin{aligned}\tau(premise)_D &= \tau(X_1 \vdash_a X_2)_D = \tau(X_1)^- \vdash \tau(X_2)^+\\ \tau(conclusion)_D &= \tau(X_1; X_3 \vdash_a X_2)_D = \tau(X_1; X_3)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+\end{aligned}$$

$$\frac{\tau(X_1)^- \vdash \tau(X_2)^+}{\tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+}\ WL$$

For WR:

$$\begin{aligned}\tau(premise)_D &= \tau(X_1 \vdash_a X_2)_D = \tau(X_1)^- \vdash \tau(X_2)^+ \\ \tau(conclusion)_D &= \tau(X_1 \vdash_a X_2; X_3)_D = \tau(X_1)^- \vdash \tau(X_2; X_3)^+ = \tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+\end{aligned}$$

$$\frac{\tau(X_1)^- \vdash \tau(X_2)^+}{\tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+}\ WR$$

For CL:

$$\begin{aligned}\tau(premise)_D &= \tau(X_1; X_3; X_3 \vdash_a X_2)_D = \tau(X_1; X_3; X_3)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; \tau(X_3)^-; \tau(X_3)^- \vdash \tau(X_2)^+ \\ \tau(conclusion)_D &= \tau(X_1; X_3 \vdash_a X_2)_D = \tau(X_1; X_3)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^-; \tau(X_3)^-; \tau(X_3)^- \vdash \tau(X_2)^+}{\tau(X_3)^-; \tau(X_3)^- \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\ AD1a}{\tau(X_3)^- \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\ CL}{\tau(X_1)^-; \tau(X_3)^- \vdash \tau(X_2)^+}\ AD1b$$

For CR:

$$\begin{aligned}\tau(premise)_D &= \tau(X_1 \vdash_a X_2; X_3; X_3)_D = \tau(X_1)^- \vdash \tau(X_2; X_3; X_3)^+ \\ &= \tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+; \tau(X_3)^+ \\ \tau(conclusion)_D &= \tau(X_1 \vdash_a X_2; X_3)_D = \tau(X_1)^- \vdash \tau(X_2; X_3)^+ = \tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+; \tau(X_3)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \tau(X_3)^+; \tau(X_3)^+}\ AD2a}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \tau(X_3)^+}\ CR}{\tau(X_1)^- \vdash \tau(X_2)^+; \tau(X_3)^+}\ AD2b$$

For the T_aL rule,

$$\begin{aligned}\tau(premise)_D &= \tau(X_1; A \triangleright_m \varnothing_m \vdash_a X_2)_D = \tau(X_1; A \triangleright_m \varnothing_m)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; \tau(A \triangleright_m \varnothing_m)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; (\tau(A)^-, \flat\tau(\varnothing_m)^+) \vdash \tau(X_2)^+ = \tau(X_1)^-; (A, \flat\varnothing_m) \vdash \tau(X_2)^+ \\ \tau(conclusion)_D &= \tau(X_1; A \vdash_a X_2)_D = \tau(X_1; A)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; \tau(A)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; A \vdash \tau(X_2)^+\end{aligned}$$

And we have the following inference:

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^-;(A,\flat\varnothing_m)\vdash\tau(X_2)^+}{A,\flat\varnothing_m\vdash\sharp\tau(X_1)^-;\tau(X_2)^+}\,AD1a}{A\vdash(\sharp\tau(X_1)^-;\tau(X_2)^+),\varnothing_m}\,MD2b}{A\vdash\sharp\tau(X_1)^-;\tau(X_2)^+}\,\varnothing_m R}{\tau(X_1)^-;A\vdash\tau(X_2)^+}\,AD1b$$

For T_aR,

$$\begin{aligned}\tau(premise)_D &= \tau(X_1\vdash_a\varnothing_m\triangleright_m A;X_2)_D=\tau(X_1)^-\vdash\tau(\varnothing_m\triangleright_m A;X_2)^+\\ &=\tau(X_1)^-\vdash\tau(\varnothing_m\triangleright_m A)^+;\tau(X_2)^+\\ &=\tau(X_1)^-\vdash(\flat\tau(\varnothing_m)^-,\tau(A)^+);\tau(X_2)^+=\tau(X_1)^-\vdash(\flat\varnothing_m,A);\tau(X_2)^+\\ \tau(conclusion)_D &=\tau(X_1\vdash_a A;X_2)_D=\tau(X_1)^-\vdash\tau(A;X_2)^+=\tau(X_1)^-\vdash\tau(A)^+;\tau(X_2)^+\\ &=\tau(X_1)^-\vdash A;\tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^-\vdash(\flat\varnothing_m,A);\tau(X_2)^+}{\tau(X_1)^-;\sharp\tau(X_2)^+\vdash\flat\varnothing_m,A}\,AD2a}{\varnothing_m,(\tau(X_1)^-;\sharp\tau(X_2)^+)\vdash A}\,MD1b}{\tau(X_1)^-;\sharp\tau(X_2)^+\vdash A}\,\varnothing_m L}{\tau(X_1)^-\vdash A;\tau(X_2)^+}\,AD2b$$

For T_mL,

$$\begin{aligned}\tau(premise)_D &=\tau(Y_1,A\triangleright_a\varnothing_a\vdash_m Y_2)_D=\tau(Y_1,A\triangleright_a\varnothing_a)^-\vdash\tau(Y_2)^+\\ &=\tau(Y_1)^-,\tau(A\triangleright_a\varnothing_a)^-\vdash\tau(Y_2)^+\\ &=\tau(Y_1)^-,(\tau(A)^-;\sharp\tau(\varnothing_a)^+)\vdash\tau(Y_2)^+=\tau(Y_1)^-,(A;\sharp\varnothing_a)\vdash\tau(Y_2)^+\\ \tau(conclusion)_D &=\tau(Y_1,A\vdash_m Y_2)_D=\tau(Y_1,A)^-\vdash\tau(Y_2)^+=\tau(Y_1)^-,\tau(A)^-\vdash\tau(Y_2)^+\\ &=\tau(Y_1)^-,A\vdash\tau(Y_2)^+\end{aligned}$$

And we have the following inference:

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^-,(A;\sharp\varnothing_a)\vdash\tau(Y_2)^+}{A,\sharp\varnothing_a\vdash\flat\tau(Y_1)^-,\tau(Y_2)^+}\,MD1a}{A\vdash(\flat\tau(Y_1)^-,\tau(Y_2)^+);\varnothing_a}\,AD2b}{A\vdash\flat\tau(Y_1)^-,\tau(Y_2)^+}\,\varnothing_a R}{\tau(Y_1)^-,A\vdash\tau(Y_2)^+}\,MD1b$$

The case for T_mR is symmetric. The cases for display rules in SI_{CBI} are as follows.

For D_1,

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(X_1 \vdash_a X_2; (Y_1 \triangleright_m Y_2))_D = \tau(X_1)^- \vdash \tau(X_2; (Y_1 \triangleright_m Y_2))^+ \\ &= \tau(X_1)^- \vdash \tau(X_2)^+; \tau(Y_1 \triangleright_m Y_2)^+ = \tau(X_1)^- \vdash \tau(X_2)^+; (\flat\tau(Y_1)^-, \tau(Y_2)^+) \\ \tau(\mathit{conclusion})_D &= \tau((X_1 \triangleright_a X_2), Y_1 \vdash_m Y_2)_D = \tau((X_1 \triangleright_a X_2))^-, \tau(Y_1)^- \vdash \tau(Y_2)^+ \\ &= (\tau(X_1)^-; \sharp\tau(X_2)^+), \tau(Y_1)^- \vdash \tau(Y_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\tau(X_1)^- \vdash \tau(X_2)^+; (\flat\tau(Y_1)^-, \tau(Y_2)^+)}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,AD2a}{(\tau(X_1)^-; \sharp\tau(X_2)^+), \tau(Y_1)^- \vdash \tau(Y_2)^+}\,MD1b$$

The case for D_2 is dual:

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(Y_1 \vdash_m Y_2, (X_1 \triangleright_a X_2))_D = \tau(Y_1)^- \vdash \tau(Y_2, (X_1 \triangleright_a X_2))^+ \\ &= \tau(Y_1)^- \vdash \tau(Y_2)^+, \tau(X_1 \triangleright_a X_2)^+ = \tau(Y_1)^- \vdash \tau(Y_2)^+, (\sharp\tau(X_1)^-; \tau(X_2)^+) \\ \tau(\mathit{conclusion})_D &= \tau((Y_1 \triangleright_m Y_2); X_1 \vdash_a X_2)_D = \tau((Y_1 \triangleright_m Y_2))^-; \tau(X_1)^- \vdash \tau(X_2)^+ \\ &= (\tau(Y_1)^-, \flat\tau(Y_2)^+); \tau(X_1)^- \vdash \tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\tau(Y_1)^- \vdash \tau(Y_2)^+, (\sharp\tau(X_1)^-; \tau(X_2)^+)}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,MD2a}{(\tau(Y_1)^-, \flat\tau(Y_2)^+); \tau(X_1)^- \vdash \tau(X_2)^+}\,AD1b$$

For $\triangleright_1$,

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(X_1 \triangleright_a X_3 \vdash_m X_2 \triangleright_a X_4)_D = \tau(X_1 \triangleright_a X_3)^- \vdash \tau(X_2 \triangleright_a X_4)^+ \\ &= \tau(X_1)^-; \sharp\tau(X_3)^+ \vdash \sharp\tau(X_2)^-; \tau(X_4)^+ \\ \tau(\mathit{conclusion})_D &= \tau(X_1; X_2 \vdash_a X_3; X_4)_D = \tau(X_1; X_2)^- \vdash \tau(X_3; X_4)^+ \\ &= \tau(X_1)^-; \tau(X_2)^- \vdash \tau(X_3)^+; \tau(X_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\tau(X_1)^-; \sharp\tau(X_3)^+ \vdash \sharp\tau(X_2)^-; \tau(X_4)^+}{\tau(X_1)^-; \sharp\tau(X_3)^+; \tau(X_2)^- \vdash \tau(X_4)^+}\,AD1b}{\tau(X_1)^-; \tau(X_2)^- \vdash \tau(X_3)^+; \tau(X_4)^+}\,AD2b$$

The case for $\triangleright_2$ is similar since no weakening nor contraction is used here.

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(Y_1 \triangleright_m Y_3 \vdash_a Y_2 \triangleright_m Y_4)_D = \tau(Y_1 \triangleright_m Y_3)^- \vdash \tau(Y_2 \triangleright_m Y_4)^+ \\ &= \tau(Y_1)^-, \flat\tau(Y_3)^+ \vdash \flat\tau(Y_2)^-, \tau(Y_4)^+ \\ \tau(\mathit{conclusion})_D &= \tau(Y_1, Y_2 \vdash_m Y_3, Y_4)_D = \tau(Y_1, Y_2)^- \vdash \tau(Y_3, Y_4)^+ \\ &= \tau(Y_1)^-, \tau(Y_2)^- \vdash \tau(Y_3)^+, \tau(Y_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\tau(Y_1)^-, \flat\tau(Y_3)^+ \vdash \flat\tau(Y_2)^-, \tau(Y_4)^+}{\tau(Y_1)^-, \flat\tau(Y_3)^+, \tau(Y_2)^- \vdash \tau(Y_4)^+}\,MD1b}{\tau(Y_1)^-, \tau(Y_2)^- \vdash \tau(Y_3)^+, \tau(Y_4)^+}\,MD2b$$

Note that the derivations for the rules T_aL, T_aR, T_mL, T_MR, D_1, D_2, $\triangleright_1$, and $\triangleright_2$ only use invertible rules in DL_{CBI}, thus we can also prove that there are derivations from conclusion to premise, so these rules in SI_{CBI} are reversible.

The following deals with logical rules with non-empty premises. Again, the cases for $\top L$, $\top^* L$, $\bot R$ and $\bot^* R$ are proved by merely translation and a direct application of corresponding rules.

For $\top L$,

$$\tau(\mathit{premise})_D = \tau(X_1 \vdash_a X_2)_D = \tau(X_1)^- \vdash \tau(X_2)^+$$
$$\tau(\mathit{conclusion})_D = \tau(X_1; \top \vdash_a X_2)_D = \tau(X_1; \top)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; \top \vdash \tau(X_2)^+$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash \tau(X_2)^+}{\tau(X_1)^-; \varnothing_a \vdash \tau(X_2)^+}\,unit}{\varnothing_a \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,AD1a}{\top \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,\top L}{\tau(X_1)^-; \top \vdash \tau(X_2)^+}\,AD1b$$

For $\top^* L$,

$$\tau(\mathit{premise})_D = \tau(Y_1 \vdash_m Y_2)_D = \tau(Y_1)^- \vdash \tau(Y_2)^+$$
$$\tau(\mathit{conclusion})_D = \tau(Y_1, \top^* \vdash_m Y_2)_D = \tau(Y_1, \top^*)^- \vdash \tau(Y_2)^+ = \tau(Y_1)^-, \top^* \vdash \tau(Y_2)^+$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash \tau(Y_2)^+}{\tau(Y_1)^-, \varnothing_m \vdash \tau(Y_2)^+}\,unit}{\varnothing_m \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,MD1a}{\top^* \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,\top^* L}{\tau(Y_1)^-, \top^* \vdash \tau(Y_2)^+}\,MD1b$$

For $\bot R$,

$$\tau(\mathit{premise})_D = \tau(X_1 \vdash_a X_2)_D = \tau(X)^- \vdash \tau(X_2)^+$$
$$\tau(\mathit{conclusion})_D = \tau(X_1 \vdash_a \bot; X_2)_D = \tau(X_1)^- \vdash \tau(\bot; X_2)^+ = \tau(X)^- \vdash \bot; \tau(X_2)^+$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash \tau(X_2)^+}{\tau(X_1)^- \vdash \varnothing_a; \tau(X_2)^+}\,unit}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \varnothing_a}\,AD2a}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash \bot}\,\bot R}{\tau(X_1)^- \vdash \bot; \tau(X_2)^+}\,AD2b$$

For $\bot^* R$

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(Y_1 \vdash_m Y_2)_D = \tau(Y_1)^- \vdash \tau(Y_2)^+ \\ \tau(\mathit{conclusion})_D &= \tau(Y_1 \vdash_m \bot^*, Y_2)_D = \tau(Y_1)^- \vdash \tau(\bot^*, Y_2)^+ = \tau(Y)^- \vdash \bot^*, \tau(Y_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash \tau(Y_2)^+}{\tau(Y_1)^- \vdash \varnothing_m, \tau(Y_2)^+}\,\mathit{unit}}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash \varnothing_m}\,\mathit{MD2a}}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash \bot^*}\,\bot^* R}{\tau(Y_1)^- \vdash \bot^*, \tau(Y_2)^+}\,\mathit{MD2b}$$

For $\wedge L$,

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(X_1; A_1 \vdash_a X_2)_D = \tau(X_1; A_1)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; \tau(A_1)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; A_1 \vdash \tau(X_2)^+ \\ \tau(\mathit{conclusion})_D &= \tau(X_1; A_1 \wedge A_2 \vdash_a X_2)_D = \tau(X_1; A_1 \wedge A_2)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; \tau(A_1 \wedge A_2)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; A_1 \wedge A_2 \vdash \tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^-; A_1 \vdash \tau(X_2)^+}{A_1 \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,\mathit{AD1a}}{A_1; A_2 \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,\mathit{WL}}{A_1 \wedge A_2 \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,\wedge L}{\tau(X_1)^-; A_1 \wedge A_2 \vdash \tau(X_2)^+}\,\mathit{AD1b}$$

The case for keeping A_2 in the premise has a similar proof but weakens A_1 instead.
For $\wedge R$,

$$\begin{aligned}\tau(\mathit{premise}_{left})_D &= \tau(X_1 \vdash_a A; X_2)_D = \tau(X_1)^- \vdash \tau(A; X_2)^+ \\ &= \tau(X_1)^- \vdash \tau(A)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A; \tau(X_2)^+ \\ \tau(\mathit{premise}_{right})_D &= \tau(X_3 \vdash_a B; X_4)_D = \tau(X_3)^- \vdash \tau(B; X_4)^+ \\ &= \tau(X_3)^- \vdash \tau(B)^+; \tau(X_4)^+ = \tau(X_3)^- \vdash B; \tau(X_4)^+ \\ \tau(\mathit{conclusion})_D &= \tau(X_1; X_3 \vdash_a A \wedge B; X_2; X_4)_D = \tau(X_1; X_3)^- \vdash \tau(A \wedge B; X_2; X_4)^+ \\ &= \tau(X_1)^-; \tau(X_3)^- \vdash A \wedge B; \tau(X_2)^+; \tau(X_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash A; \tau(X_2)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A}\,\mathit{AD2a} \quad \dfrac{\tau(X_3)^- \vdash B; \tau(X_4)^+}{\tau(X_3)^-; \sharp\tau(X_4)^+ \vdash B}\,\mathit{AD2a}}{\tau(X_1)^-; \tau(X_3)^-; \sharp\tau(X_2)^+; \sharp\tau(X_4)^+ \vdash A \wedge B}\,\wedge R}{\tau(X_1)^-; \tau(X_3)^-; \sharp\tau(X_2)^+ \vdash A \wedge B; \tau(X_4)^+}\,\mathit{AD2b}}{\tau(X_1)^-; \tau(X_3)^- \vdash A \wedge B; \tau(X_2)^+; \tau(X_4)^+}\,\mathit{AD2b}$$

The cases for $\vee L$ and $\vee R$ are dual to $\wedge R$ and $\wedge L$.

For $\vee L$

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(X_1; A \vdash_a X_2)_D = \tau(X_1; A)^- \vdash \tau(X_2)^+ \\
&= \tau(X_1)^-; \tau(A)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; A \vdash \tau(X_2)^+ \\
\tau(premise_{right})_D &= \tau(X_3; B \vdash_a X_4)_D = \tau(X_3; B)^- \vdash \tau(X_4)^+ \\
&= \tau(X_3)^-; \tau(B)^- \vdash \tau(X_4)^+ = \tau(X_3)^-; B \vdash \tau(X_4)^+ \\
\tau(conclusion)_D &= \tau(X_1; X_3; A \vee B \vdash_a X_2; X_4)_D = \tau(X_1; X_3; A \vee B)^- \vdash \tau(X_2; X_4)^+ \\
&= \tau(X_1)^-; \tau(X_3)^-; A \vee B \vdash \tau(X_2)^+; \tau(X_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^-; A \vdash \tau(X_2)^+}{A \vdash \tau(X_2)^+; \sharp\tau(X_1)^-}\,AD1a \qquad \dfrac{\tau(X_3)^-; B \vdash \tau(X_4)^+}{B \vdash \tau(X_4)^+; \sharp\tau(X_3)^-}\,AD1a}{A \vee B \vdash \sharp\tau(X_1)^-; \sharp\tau(X_3)^-; \tau(X_2)^+; \tau(X_4)^+}\,\vee L}{\dfrac{\tau(X_1)^-; A \vee B \vdash \sharp\tau(X_3)^-; \tau(X_2)^+; \tau(X_4)^+}{\tau(X_1)^-; \tau(X_3)^-; A \vee B \vdash \tau(X_2)^+; \tau(X_4)^+}\,AD1b}\,AD1b$$

For $\vee R$,

$$\begin{aligned}
\tau(premise)_D &= \tau(X_1 \vdash_a A_1; X_2)_D = \tau(X_1)^- \vdash \tau(A_1; X_2)^+ \\
&= \tau(X_1)^- \vdash \tau(A_1)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A_1; \tau(X_2)^+ \\
\tau(conclusion)_D &= \tau(X_1 \vdash_a A_1 \vee A_2; X_2)_D = \tau(X_1)^- \vdash \tau(A_1 \vee A_2; X_2)^+ \\
&= \tau(X_1)^- \vdash \tau(A_1 \vee A_2)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A_1 \vee A_2; \tau(X_2)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash A_1; \tau(X_2)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A_1}\,AD2a}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A_1; A_2}\,WR}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A_1 \vee A_2}\,\vee R}{\tau(X_1)^- \vdash A_1 \vee A_2; \tau(X_2)^+}\,AD2b$$

Weakening A_1 instead yields the proof for the case of A_2 in the premise of the rule.
For $\to L$,

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(X_1 \vdash_a A; X_2)_D = \tau(X_1)^- \vdash \tau(A; X_2)^+ \\
&= \tau(X_1)^- \vdash \tau(A)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A; \tau(X_2)^+ \\
\tau(premise_{right})_D &= \tau(X_3; B \vdash_a X_4)_D = \tau(X_3; B)^- \vdash \tau(X_4)^+ \\
&= \tau(X_3)^-; \tau(B)^- \vdash \tau(X_4)^+ = \tau(X_3)^-; B \vdash \tau(X_4)^+ \\
\tau(conclusion)_D &= \tau(X_1; X_3; A \to B \vdash_a X_2; X_4)_D = \tau(X_1; X_3; A \to B)^- \vdash \tau(X_2; X_4)^+ \\
&= \tau(X_1)^-; \tau(X_3)^-; A \to B \vdash \tau(X_2)^+; \tau(X_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash A; \tau(X_2)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A}\,AD2a \quad \dfrac{\tau(X_3)^-; B \vdash \tau(X_4)^+}{B \vdash \tau(X_4)^+; \sharp\tau(X_3)^-}\,AD1a}{A \to B \vdash \sharp\tau(X_1)^-; \sharp\tau(X_3)^-; \sharp\sharp\tau(X_2)^+; \tau(X_4)^+}\,{\to}L}{\sharp\tau(X_2)^+; A \to B \vdash \sharp\tau(X_1)^-; \sharp\tau(X_3)^-; \tau(X_4)^+}\,AD1b}{A \to B \vdash \sharp\tau(X_1)^-; \sharp\tau(X_3)^-; \tau(X_2)^+; \tau(X_4)^+}\,AD2b}{\tau(X_1)^-; A \to B \vdash \sharp\tau(X_3)^-; \tau(X_2)^+; \tau(X_4)^+}\,AD1b}{\tau(X_1)^-; \tau(X_3)^-; A \to B \vdash \tau(X_2)^+; \tau(X_4)^+}\,AD1b$$

For $\to R$

$$\begin{aligned}\tau(premise)_D &= \tau(X_1; A \vdash_a B; X_2)_D = \tau(X_1; A)^- \vdash \tau(B; X_2)^+ \\ &= \tau(X_1)^-; \tau(A)^- \vdash \tau(B)^+; \tau(X_2)^+ = \tau(X_1)^-; A \vdash B; \tau(X_2)^+ \\ \tau(conclusion)_D &= \tau(X_1 \vdash_a A \to B; X_2)_D = \tau(X_1)^- \vdash \tau(A \to B; X_2)^+ \\ &= \tau(X_1)^- \vdash \tau(A \to B)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A \to B; \tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^-; A \vdash B; \tau(X_2)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+; A \vdash B}\,AD2a}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A \to B}\,{\to}R}{\tau(X_1)^- \vdash A \to B; \tau(X_2)^+}\,AD2b$$

The cases for $\prec$ rules are similar to $\wedge$ since $A \prec B =_{def} A \wedge \neg B$.
For $\prec L$,

$$\begin{aligned}\tau(premise)_D &= \tau(X_1; A \vdash_a B; X_2)_D = \tau(X_1; A)^- \vdash \tau(B; X_2)^+ \\ &= \tau(X_1)^-; \tau(A)^- \vdash \tau(B)^+; \tau(X_2)^+ = \tau(X_1)^-; A \vdash B; \tau(X_2)^+ \\ \tau(conclusion)_D &= \tau(X_1; A \prec B \vdash_a X_2)_D = \tau(X_1; A \prec B)^- \vdash \tau(X_2)^+ \\ &= \tau(X_1)^-; \tau(A \prec B)^- \vdash \tau(X_2)^+ = \tau(X_1)^-; A \wedge \neg B \vdash \tau(X_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X_1)^-; A \vdash B; \tau(X_2)^+}{\tau(X_1)^-; A; \sharp B \vdash \tau(X_2)^+}\,AD2a}{A; \sharp B \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,AD1a}{\sharp B \vdash \sharp A; \sharp\tau(X_1)^-; \tau(X_2)^+}\,AD1a}{\neg B \vdash \sharp A; \sharp\tau(X_1)^-; \tau(X_2)^+}\,\neg L}{A; \neg B \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,AD1b}{A \wedge \neg B \vdash \sharp\tau(X_1)^-; \tau(X_2)^+}\,\wedge L}{\tau(X_1)^-; A \wedge \neg B \vdash \tau(X_2)^+}\,AD1b$$

For $-\!\!\prec R$,

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(X_1 \vdash_a A; X_2)_D = \tau(X_1)^- \vdash \tau(A; X_2)^+ \\
&= \tau(X_1)^- \vdash \tau(A)^+; \tau(X_2)^+ = \tau(X_1)^- \vdash A; \tau(X_2)^+ \\
\tau(premise_{right})_D &= \tau(X_3; B \vdash_a X_4)_D = \tau(X_3; B)^- \vdash \tau(X_4)^+ \\
&= \tau(X_3)^-; \tau(B)^- \vdash \tau(X_4)^+ = \tau(X_3)^-; B \vdash \tau(X_4)^+ \\
\tau(conclusion)_D &= \tau(X_1; X_3 \vdash_a A -\!\!\prec B; X_2; X_4)_D = \tau(X_1; X_3)^- \vdash \tau(A -\!\!\prec B; X_2; X_4)^+ \\
&= \tau(X_1)^-; \tau(X_3)^- \vdash A \wedge \neg B; \tau(X_2)^+; \tau(X_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X_1)^- \vdash A; \tau(X_2)^+}{\tau(X_1)^-; \sharp\tau(X_2)^+ \vdash A}\,AD2a \qquad \dfrac{\dfrac{\dfrac{\tau(X_3)^-; B \vdash \tau(X_4)^+}{\tau(X_3)^- \vdash \sharp B; \tau(X_4)^+}\,AD1a}{\tau(X_3)^-; \sharp\tau(X_4)^+ \vdash \sharp B}\,AD2a}{\tau(X_3)^-; \sharp\tau(X_4)^+ \vdash \neg B}\,\neg R}{\tau(X_1)^-; \tau(X_3)^-; \sharp\tau(X_2)^+; \sharp\tau(X_4)^+ \vdash A \wedge \neg B}\,\wedge R}{\tau(X_1)^-; \tau(X_3)^-; \sharp\tau(X_2)^+ \vdash A \wedge \neg B; \tau(X_4)^+}\,AD2b}{\tau(X_1)^-; \tau(X_3)^- \vdash A \wedge \neg B; \tau(X_2)^+; \tau(X_4)^+}\,AD2b$$

The cases for multiplicative rules are similar to those above but without using weakening and contraction.

For $*L$,

$$\begin{aligned}
\tau(premise)_D &= \tau(Y_1, A_1, A_2 \vdash_m Y_2)_D = \tau(Y_1, A_1, A_2)^- \vdash \tau(Y_2)^+ \\
&= \tau(Y_1)^-, \tau(A_1)^-, \tau(A_2)^- \vdash \tau(Y_2)^+ = \tau(Y_1)^-, A_1, A_2 \vdash \tau(Y_2)^+ \\
\tau(conclusion)_D &= \tau(Y_1, A_1 * A_2 \vdash_m Y_2)_D = \tau(Y_1, A_1 * A_2)^- \vdash \tau(Y_2)^+ \\
&= \tau(Y_1)^-, \tau(A_1 * A_2)^- \vdash \tau(Y_2)^+ = \tau(Y_1)^-, A_1 * A_2 \vdash \tau(Y_2)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^-, A_1, A_2 \vdash \tau(Y_2)^+}{A_1, A_2 \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,MD1a}{A_1 * A_2 \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,*L}{\tau(Y_1)^-, A_1 * A_2 \vdash \tau(Y_2)^+}\,MD1b$$

For $*R$,

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(Y_1 \vdash_m A, Y_2)_D = \tau(Y_1)^- \vdash \tau(A, Y_2)^+ \\
&= \tau(Y_1)^- \vdash \tau(A)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A, \tau(Y_2)^+ \\
\tau(premise_{right})_D &= \tau(Y_3 \vdash_m B, Y_4)_D = \tau(Y_3)^- \vdash \tau(B, Y_4)^+ \\
&= \tau(Y_3)^- \vdash \tau(B)^+, \tau(Y_4)^+ = \tau(Y_3)^- \vdash B, \tau(Y_4)^+ \\
\tau(conclusion)_D &= \tau(Y_1, Y_3 \vdash_m A * B, Y_2, Y_4)_D = \tau(Y_1, Y_3)^- \vdash \tau(A * B, Y_2, Y_4)^+ \\
&= \tau(Y_1)^-, \tau(Y_3)^- \vdash A * B, \tau(Y_2)^+, \tau(Y_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash A, \tau(Y_2)^+}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A}\,MD2a \quad \dfrac{\tau(Y_3)^- \vdash B, \tau(Y_4)^+}{\tau(Y_3)^-, \flat\tau(Y_4)^+ \vdash B}\,MD2a}{\tau(Y_1)^-, \tau(Y_3)^-, \flat\tau(Y_2)^+, \flat\tau(Y_4)^+ \vdash A * B}\,{*R}}{\dfrac{\tau(Y_1)^-, \tau(Y_3)^-, \flat\tau(Y_2)^+ \vdash A * B, \tau(Y_4)^+}{\tau(Y_1)^-, \tau(Y_3)^- \vdash A * B, \tau(Y_2)^+, \tau(Y_4)^+}\,MD2b}\,MD2b$$

For $\vee^*\ L$

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(Y_1, A \vdash_m Y_2)_D = \tau(Y_1, A)^- \vdash \tau(Y_2)^+ \\
&= \tau(Y_1)^-, \tau(A)^- \vdash \tau(Y_2)^+ = \tau(Y_1)^-, A \vdash \tau(Y_2)^+ \\
\tau(premise_{right})_D &= \tau(Y_3, B \vdash_m Y_4)_D = \tau(Y_3, B)^- \vdash \tau(Y_4)^+ \\
&= \tau(Y_3)^-, \tau(B)^- \vdash \tau(Y_4)^+ = \tau(Y_3)^-, B \vdash \tau(Y_4)^+ \\
\tau(conclusion)_D &= \tau(Y_1, Y_3, A \vee^* B \vdash_m Y_2, Y_4)_D = \tau(Y_1, Y_3, A \vee^* B)^- \vdash \tau(Y_2, Y_4)^+ \\
&= \tau(Y_1)^-, \tau(Y_3)^-, A \vee^* B \vdash \tau(Y_2)^+, \tau(Y_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^-, A \vdash \tau(Y_2)^+}{A \vdash \tau(Y_2)^+, \flat\tau(Y_1)^-}\,MD1a \quad \dfrac{\tau(Y_3)^-, B \vdash \tau(Y_4)^+}{B \vdash \tau(Y_4)^+, \flat\tau(Y_3)^-}\,MD1a}{A \vee^* B \vdash \flat\tau(Y_1)^-, \flat\tau(Y_3)^-, \tau(Y_2)^+, \tau(Y_4)^+}\,{\vee^* L}}{\dfrac{\tau(Y_1)^-, A \vee^* B \vdash \flat\tau(Y_3)^-, \tau(Y_2)^+, \tau(Y_4)^+}{\tau(Y_1)^-, \tau(Y_3)^-, A \vee^* B \vdash \tau(Y_2)^+, \tau(Y_4)^+}\,MD1b}\,MD1b$$

For $\vee^*\ R$,

$$\begin{aligned}
\tau(premise)_D &= \tau(Y_1 \vdash_m A_1, A_2, Y_2)_D = \tau(Y_1)^- \vdash \tau(A_1, A_2, Y_2)^+ \\
&= \tau(Y_1)^- \vdash \tau(A_1)^+, \tau(A_2)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A_1, A_2, \tau(Y_2)^+ \\
\tau(conclusion)_D &= \tau(Y_1 \vdash_m A_1 \vee^* A_2, Y_2)_D = \tau(Y_1)^- \vdash \tau(A_1 \vee^* A_2, Y_2)^+ \\
&= \tau(Y_1)^- \vdash \tau(A_1 \vee^* A_2)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A_1 \vee^* A_2, \tau(Y_2)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash A_1, A_2, \tau(Y_2)^+}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A_1, A_2}\,MD2a}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A_1 \vee^* A_2}\,{\vee^* R}}{\tau(Y_1)^- \vdash A_1 \vee^* A_2, \tau(Y_2)^+}\,MD2b$$

For $-\!\!*\ L$,

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(Y_1 \vdash_m A, Y_2)_D = \tau(Y_1)^- \vdash \tau(A, Y_2)^+ \\
&= \tau(Y_1)^- \vdash \tau(A)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A, \tau(Y_2)^+
\end{aligned}$$

$$\begin{aligned}\tau(\mathit{premise}_{\mathit{right}})_D &= \tau(Y_3, B \vdash_m Y_4)_D = \tau(Y_3, B)^- \vdash \tau(Y_4)^+ \\ &= \tau(Y_3)^-, \tau(B)^- \vdash \tau(Y_4)^+ = \tau(Y_3)^-, B \vdash \tau(Y_4)^+ \\ \tau(\mathit{conclusion})_D &= \tau(Y_1, Y_3, A \mathrel{-\!\!*} B \vdash_m Y_2, Y_4)_D = \tau(Y_1, Y_3, A \mathrel{-\!\!*} B)^- \vdash \tau(Y_2, Y_4)^+ \\ &= \tau(Y_1)^-, \tau(Y_3)^-, A \mathrel{-\!\!*} B \vdash \tau(Y_2)^+, \tau(Y_4)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash A, \tau(Y_2)^+}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A}\,MD2a \quad \dfrac{\tau(Y_3)^-, B \vdash \tau(Y_4)^+}{B \vdash \tau(Y_4)^+, \flat\tau(Y_3)^-}\,MD1a}{A \mathrel{-\!\!*} B \vdash \flat\tau(Y_1)^-, \flat\tau(Y_3)^-, \flat\flat\tau(Y_2)^+, \tau(Y_4)^+}\,{\mathrel{-\!\!*}}L}{\flat\tau(Y_2)^+, A \mathrel{-\!\!*} B \vdash \flat\tau(Y_1)^-, \flat\tau(Y_3)^-, \tau(Y_4)^+}\,MD1b}{A \mathrel{-\!\!*} B \vdash \flat\tau(Y_1)^-, \flat\tau(Y_3)^-, \tau(Y_2)^+, \tau(Y_4)^+}\,MD2b}{\tau(Y_1)^-, A \mathrel{-\!\!*} B \vdash \flat\tau(Y_3)^-, \tau(Y_2)^+, \tau(Y_4)^+}\,MD1b}{\tau(Y_1)^-, \tau(Y_3)^-, A \mathrel{-\!\!*} B \vdash \tau(Y_2)^+, \tau(Y_4)^+}\,MD1b$$

For $-\!\!* R$

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(Y_1, A \vdash_m B, Y_2)_D = \tau(Y_1, A)^- \vdash \tau(B, Y_2)^+ \\ &= \tau(Y_1)^-, \tau(A)^- \vdash \tau(B)^+, \tau(Y_2)^+ = \tau(Y_1)^-, A \vdash B, \tau(Y_2)^+ \\ \tau(\mathit{conclusion})_D &= \tau(Y_1 \vdash_m A \mathrel{-\!\!*} B, Y_2)_D = \tau(Y_1)^- \vdash \tau(A \mathrel{-\!\!*} B, Y_2)^+ \\ &= \tau(Y_1)^- \vdash \tau(A \mathrel{-\!\!*} B)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A \mathrel{-\!\!*} B, \tau(Y_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^-, A \vdash B, \tau(Y_2)^+}{\tau(Y_1)^-, \flat\tau(Y_2)^+, A \vdash B}\,MD2a}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A \mathrel{-\!\!*} B}\,{\mathrel{-\!\!*}}R}{\tau(Y_1)^- \vdash A \mathrel{-\!\!*} B, \tau(Y_2)^+}\,MD2b$$

The cases for $-\!\!\times$ rules are similar to $*$ since $A \mathrel{-\!\!\times} B =_{def} A* \sim B$.

For $-\!\!\times L$,

$$\begin{aligned}\tau(\mathit{premise})_D &= \tau(Y_1, A \vdash_m B, Y_2)_D = \tau(Y_1, A)^- \vdash \tau(B, Y_2)^+ \\ &= \tau(Y_1)^-, \tau(A)^- \vdash \tau(B)^+, \tau(Y_2)^+ = \tau(Y_1)^-, A \vdash B, \tau(Y_2)^+ \\ \tau(\mathit{conclusion})_D &= \tau(Y_1, A \mathrel{-\!\!\times} B \vdash_m Y_2)_D = \tau(Y_1, A \mathrel{-\!\!\times} B)^- \vdash \tau(Y_2)^+ \\ &= \tau(Y_1)^-, \tau(A \mathrel{-\!\!\times} B)^- \vdash \tau(Y_2)^+ = \tau(Y_1)^-, A* \sim B \vdash \tau(Y_2)^+\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y_1)^-, A \vdash B, \tau(Y_2)^+}{\tau(Y_1)^-, A, \flat B \vdash \tau(Y_2)^+}\,MD2a}{A, \flat B \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,MD1a}{\flat B \vdash \flat A, \flat\tau(Y_1)^-, \tau(Y_2)^+}\,MD1a}{\sim B \vdash \flat A, \flat\tau(Y_1)^-, \tau(Y_2)^+}\sim L}{A, \sim B \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}\,MD1b}{A* \sim B \vdash \flat\tau(Y_1)^-, \tau(Y_2)^+}*L}{\tau(Y_1)^-, A* \sim B \vdash \tau(Y_2)^+}\,MD1b$$

For $\multimap\!\!\times\ R$,

$$\begin{aligned}
\tau(premise_{left})_D &= \tau(Y_1 \vdash_m A, Y_2)_D = \tau(Y_1)^- \vdash \tau(A, Y_2)^+ \\
&= \tau(Y_1)^- \vdash \tau(A)^+, \tau(Y_2)^+ = \tau(Y_1)^- \vdash A, \tau(Y_2)^+ \\
\tau(premise_{right})_D &= \tau(Y_3, B \vdash_m Y_4)_D = \tau(Y_3, B)^- \vdash \tau(Y_4)^+ \\
&= \tau(Y_3)^-, \tau(B)^- \vdash \tau(Y_4)^+ = \tau(Y_3)^-, B \vdash \tau(Y_4)^+ \\
\tau(conclusion)_D &= \tau(Y_1, Y_3 \vdash_m A \multimap\!\!\times\ B, Y_2, Y_4)_D = \tau(Y_1, Y_3)^- \vdash \tau(A \multimap\!\!\times\ B, Y_2, Y_4)^+ \\
&= \tau(Y_1)^-, \tau(Y_3)^- \vdash A* \sim B, \tau(Y_2)^+, \tau(Y_4)^+
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(Y_1)^- \vdash A, \tau(Y_2)^+}{\tau(Y_1)^-, \flat\tau(Y_2)^+ \vdash A}\,MD2a \qquad \dfrac{\dfrac{\dfrac{\tau(Y_3)^-, B \vdash \tau(Y_4)^+}{\tau(Y_3)^- \vdash \flat B, \tau(Y_4)^+}\,MD1a}{\tau(Y_3)^-, \flat\tau(Y_4)^+ \vdash \flat B}\,MD2a}{\tau(Y_3)^-, \flat\tau(Y_4)^+ \vdash \sim B}\sim R}{\tau(Y_1)^-, \tau(Y_3)^-, \flat\tau(Y_2)^+, \flat\tau(Y_4)^+ \vdash A* \sim B}*R}{\tau(Y_1)^-, \tau(Y_3)^-, \flat\tau(Y_2)^+ \vdash A* \sim B, \tau(Y_4)^+}\,MD2b}{\tau(Y_1)^-, \tau(Y_3)^- \vdash A* \sim B, \tau(Y_2)^+, \tau(Y_4)^+}\,MD2b$$

The above derivations have considered every rule in SI_{CBI}, thus the inductive case holds. Therefore the said lemma can be concluded. □

Theorem 3.2.2 (Soundness)**.** *If a CBI formula F is provable in SI_{CBI} (i.e., $\varnothing_a \vdash_a F$ is provable in SI_{CBI}), then F is provable in DL_{CBI} (i.e., $\varnothing_a \vdash F$ is provable in DL_{CBI}).*

Proof. From the above lemma, if $\varnothing_a \vdash_a F$ is provable in SI_{CBI}, then there is a proof of $\tau(\varnothing_a \vdash_a F)_D$ in DL_{CBI}.

According to our translation (cf. Figure 3.5),

$$\tau(\varnothing_a \vdash_a F)_D = \tau(\varnothing_a)^- \vdash \tau(F)^+ = \varnothing_a \vdash F$$

therefore we have a proof of $\varnothing_a \vdash F$ in DL_{CBI}. □

3.2.3 Completeness of SI_{CBI}

The logical rules and most of structural rules in SI_{CBI} are very similar to those in DL_{CBI}, if there were not any difference in structures, it would be easy to prove that these two sets of rules are equivalent. To prove the correspondence between SI_{CBI} and DL_{CBI}, we introduce another structure translation to convert the display calculus structures into shallow nested sequent structures. The design principle of this translation is to simulate display calculus structures in nested sequents with interleaving additive and multiplicative turnstiles, and ensure that structures in DL_{CBI} can be translated into equivalent structures in SI_{CBI}. The detailed translation is shown in Figure 3.6. It is straightforward to show that this translation is faithful.

$\tau(X_1 \vdash X_2)_s = \tau(X_1)_{la} \vdash_a \tau(X_2)_{ra}$

$\tau(\varnothing_a)_{la} = \varnothing_a$	$\tau(\varnothing_a)_{ra} = \varnothing_a$
$\tau(\varnothing_a)_{lm} = \varnothing_a \triangleright_a \varnothing_a$	$\tau(\varnothing_a)_{rm} = \varnothing_a \triangleright_a \varnothing_a$
$\tau(\sharp X)_{la} = (\varnothing_a \triangleright_a \tau(X)_{ra}) \triangleright_m \varnothing_m$	$\tau(\sharp X)_{ra} = \varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \varnothing_a)$
$\tau(\sharp X)_{lm} = \varnothing_a \triangleright_a \tau(X)_{ra}$	$\tau(\sharp X)_{rm} = \tau(X)_{la} \triangleright_a \varnothing_a$
$\tau(X_1; X_2)_{la} = \tau(X_1)_{la}; \tau(X_2)_{la}$	$\tau(X_1; X_2)_{ra} = \tau(X_1)_{ra}; \tau(X_2)_{ra}$
$\tau(X_1; X_2)_{lm} = (\tau(X_1)_{la}; \tau(X_2)_{la}) \triangleright_a \varnothing_a$	$\tau(X_1; X_2)_{rm} = \varnothing_a \triangleright_a (\tau(X_1)_{ra}; \tau(X_2)_{ra})$
$\tau(F)_{la} = F$	$\tau(F)_{ra} = F$
$\tau(F)_{lm} = F$	$\tau(F)_{rm} = F$
$\tau(\varnothing_m)_{la} = \varnothing_m \triangleright_m \varnothing_m$	$\tau(\varnothing_m)_{ra} = \varnothing_m \triangleright_m \varnothing_m$
$\tau(\varnothing_m)_{lm} = \varnothing_m$	$\tau(\varnothing_m)_{rm} = \varnothing_m$
$\tau(\flat X)_{la} = \varnothing_m \triangleright_m \tau(X)_{rm}$	$\tau(\flat X)_{ra} = \tau(X)_{lm} \triangleright_m \varnothing_m$
$\tau(\flat X)_{lm} = (\varnothing_m \triangleright_m \tau(X)_{rm}) \triangleright_a \varnothing_a$	$\tau(\flat X)_{rm} = \varnothing_a \triangleright_a (\tau(X)_{lm} \triangleright_m \varnothing_m)$
$\tau(X_1, X_2)_{la} = (\tau(X_1)_{lm}, \tau(X_2)_{lm}) \triangleright_m \varnothing_m$	$\tau(X_1, X_2)_{ra} = \varnothing_m \triangleright_m (\tau(X_1)_{rm}, \tau(X_2)_{rm})$
$\tau(X_1, X_2)_{lm} = \tau(X_1)_{lm}, \tau(X_2)_{lm}$	$\tau(X_1, X_2)_{rm} = \tau(X_1)_{rm}, \tau(X_2)_{rm}$

In the translation, F denotes a formula, and X, X_1, X_2 are structures.

Figure 3.6: Structure translation from DL_{CBI} to SI_{CBI}

We do not deal with negations in this translation, but instead we use $A \to \bot$ to encode $\neg A$, and $A \multimap \bot^*$ to encode $\sim A$. By treating negations as abbreviated implications, we deal with fewer rules in the proof system, although this is not of great significance. We assume that the initial sequent to be translated is additive. If both sides of the initial sequent are multiplicative, the translation will convert it to $Y_1 \triangleright_m \varnothing_m \vdash_a \varnothing_m \triangleright_m Y_2$, and then we can use $\triangleright_2$ rule to get $Y_1 \vdash_m Y_2$. If only the right hand side is multiplicative, then the translation will convert the sequent into $X_1 \vdash_a \varnothing_m \triangleright_m Y_1$, we give the following inference:

$$\dfrac{\dfrac{\dfrac{X_1 \triangleright_a \varnothing_a \vdash_m Y_1}{(X_1 \triangleright_a \varnothing_a), \varnothing_m \vdash_m Y_1}\,{\scriptstyle unit}}{X_1 \vdash_a \varnothing_a; (\varnothing_m \triangleright_m Y_1)}\,{\scriptstyle D_1}}{X_1 \vdash_a \varnothing_m \triangleright_m Y_1}\,{\scriptstyle unit}$$

The case for multiplicative structure only on the left hand side is similar. So it does not matter which turnstile we use for the initial sequent, the subsequent translation will make sure it is converted into a corresponding structure in SI_{CBI}. Therefore SI_{CBI} together with the translation in Figure 3.6 can be seen as an intermediate tool to show that we can encode the display calculus for CBI into a structure with nested interleaving additive and multiplicative turnstiles. In the following we refer to the interderivability of sequents as equivalence over sequents. We have the following equivalence relations in our translation:

Lemma 3.2.3. *The following equivalence relations hold in our translation from DL_{CBI} sequents to SI_{CBI} sequents.*

$$\tau(X)_{la}; X_1 \vdash_a X_2 = (\tau(X)_{lm} \triangleright_m \varnothing_m); X_1 \vdash_a X_2 \qquad \tau(X)_{lm}, Y_1 \vdash_m Y_2 = (\tau(X)_{la} \triangleright_a \varnothing_a), Y_1 \vdash_m Y_2$$
$$X_1 \vdash_a X_2; \tau(X)_{ra} = X_1 \vdash_a X_2; (\varnothing_m \triangleright_m \tau(X)_{rm}) \qquad Y_1 \vdash_m Y_2, \tau(X)_{rm} = Y_1 \vdash_m Y_2, (\varnothing_a \triangleright_a \tau(X)_{ra})$$

Proof. We only give the proof of the right column here, the left column is symmetric. That is, we only prove the following holds:

$$\frac{\tau(X)_{lm}, Y_1 \vdash_m Y_2}{\tau(X)_{la} \triangleright_a \varnothing_a), Y_1 \vdash_m Y_2} \qquad \frac{Y_1 \vdash_m Y_2, \tau(X)_{rm}}{Y_1 \vdash_m Y_2, (\varnothing_a \triangleright_a \tau(X)_{ra})}$$

For the left equivalence, by the definition of structures in DL_{CBI},

$$X ::= F \mid \varnothing_a \mid \sharp X_1 \mid X_1; X_2 \mid \varnothing_m \mid \flat X_1 \mid X_1, X_2.$$

If $X = F$, then the equivalence is proved by the $T_m L$ rule.

For other cases, our translation either translates the premise and the conclusion to be the same, or to the following form:

$$\frac{X', Y_1 \vdash_m Y_2}{((X' \triangleright_m \varnothing_m) \triangleright_a \varnothing_a), Y_1 \vdash_m Y_2}$$

Then we have the following inference:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{X', Y_1 \vdash_m Y_2}{X', Y_1 \vdash_m \varnothing_m, Y_2}\,unit}{X' \triangleright_m \varnothing_m \vdash_a Y_1 \triangleright_m Y_2}\,\triangleright_2}{X' \triangleright_m \varnothing_m \vdash_a \varnothing_a; (Y_1 \triangleright_m Y_2)}\,unit}{((X' \triangleright_m \varnothing_m) \triangleright_a \varnothing_a), Y_1 \vdash_m Y_2}\,D_2$$

Since every rules we used above is reversible, the equivalence is proved. The case for the last equivalence relation is similar. If $X = F$, then an application of $T_m R$ rule is sufficient to prove it. For other possibilities, the translation either converts the premise and the conclusion to the same sequent, or to the following form:

$$\frac{Y_1 \vdash_m Y_2, X'}{Y_1 \vdash_m Y_2, (\varnothing_a \triangleright_a (\varnothing_m \triangleright_m X'))}$$

Then a similar inference can be used to prove the equivalence:

$$\dfrac{\dfrac{\dfrac{\dfrac{Y_1 \vdash_m Y_2, X'}{Y_1, \varnothing_m \vdash_m Y_2, X'}\,unit}{Y_1 \rhd_m Y_2 \vdash_a \varnothing_m \rhd_m X'}\,\rhd_2}{(Y_1 \rhd_m Y_2); \varnothing_a \vdash_a \varnothing_m \rhd_m X'}\,unit}{Y_1 \vdash_m Y_2, (\varnothing_a \rhd_a (\varnothing_m \rhd_m X'))}\,D_2$$

□

To help prove the subsequent lemmas and theorems, we need another set of equivalence relations.

Lemma 3.2.4. *The following equivalence relations hold in our translation from DL_{CBI} sequents to SI_{CBI} sequents.*

$$\tau(X)_{la} \vdash_a F = \tau(X)_{lm} \vdash_m F \quad F \vdash_a \tau(X)_{ra} = F \vdash_m \tau(X)_{rm}$$

Proof. Since we will only use the backward direction of the equivalence, we will only prove those here. The other direction is analogous. That is, we prove the following holds:

$$\dfrac{\tau(X)_{la} \vdash_a F}{\tau(X)_{lm} \vdash_m F} \qquad \dfrac{F \vdash_a \tau(X)_{ra}}{F \vdash_m \tau(X)_{rm}}$$

For the left case, we have the following inference:

$$\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a F}{\tau(X)_{la}; \varnothing_a \vdash_a \varnothing_a; F}\,unit}{\tau(X)_{la} \rhd_a \varnothing_a \vdash_m \varnothing_a \rhd_a F}\,\rhd_1}{\tau(X)_{la} \rhd_a \varnothing_a \vdash_m F}\,T_m R}{\tau(X)_{lm} \vdash_m F}\,\text{Lemma 3.2.3}$$

The case on the right is similar:

$$\dfrac{\dfrac{\dfrac{\dfrac{F \vdash_a \tau(X)_{ra}}{F; \varnothing_a \vdash_a \varnothing_a; \tau(X)_{ra}}\,unit}{F \rhd_a \varnothing_a \vdash_m \varnothing_a \rhd_a \tau(X)_{ra}}\,\rhd_1}{F \vdash_m \varnothing_a \rhd_a \tau(X)_{ra}}\,T_m L}{F \vdash_m \tau(X)_{rm}}\,\text{Lemma 3.2.3}$$

Since the rules we used here are all reversible, it should not be surprising that these inferences can go both directions. □

The following lemma shows that our translation is faithful. That is, a provable sequent in DL_{CBI} is translated to a corresponding provable sequent in SI_{CBI}.

Lemma 3.2.5. *If a sequent $X \vdash Y$ is provable in DL_{CBI}, then $\tau(X \vdash Y)_S$ is provable in SI_{CBI}.*

Proof. By induction on the depth of derivation.

(i) The base case is that if a sequent can be proved in DL_{CBI} with one step, then we can find a proof of the translated sequent in SI_{CBI}. Applicable rules in DL_{CBI} for this case are id, $\top R$, $\top^* R$, $\bot L$, and $\bot^* L$.

For id,

$$\tau(P \vdash P)_S = \tau(P)_{la} \vdash_a \tau(P)_{ra} = P \vdash_a P$$

which can be proved by the id_a rule in SI_{CBI}.

For $\top R$,

$$\tau(\varnothing_a \vdash \top)_S = \tau(\varnothing_a)_{la} \vdash_a \tau(\top)_{ra} = \varnothing_a \vdash_a \top$$

Which can be proved by the $\top R$ rule in SI_{CBI}.

For $\top^* R$,

$$\tau(\varnothing_m \vdash \top^*)_S = \tau(\varnothing_m)_{la} \vdash_a \tau(\top^*)_{ra} = \varnothing_m \rhd_m \varnothing_m \vdash_a \top^*$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing_m \vdash_m \top^*}\,{\scriptstyle \top^* R}}{\varnothing_m, \varnothing_m \vdash_m \varnothing_m, \top^*}\,{\scriptstyle unit}}{\varnothing_m \rhd_m \varnothing_m \vdash_a \varnothing_m \rhd_m \top^*}\,{\scriptstyle \rhd_2}}{\varnothing_m \rhd_m \varnothing_m \vdash_a \top^*}\,{\scriptstyle T_a R}$$

For $\bot L$,

$$\tau(\bot \vdash \varnothing_a)_S = \tau(\bot)_{la} \vdash_a \tau(\varnothing_a)_{ra} = \bot \vdash_a \varnothing_a$$

Which can be proved directly by $\bot L$ in SI_{CBI}.

For $\bot^* L$,

$$\tau(\bot^* \vdash \varnothing_m)_S = \tau(\bot^*)_{la} \vdash_a \tau(\varnothing_m)_{ra} = \bot^* \vdash_a \varnothing_m \rhd_m \varnothing_m$$

Then we have the following inference in SI_{CBI} on the resultant sequent.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\bot^* \vdash_m \varnothing_m}\,{\scriptstyle \bot^* L}}{\bot^*, \varnothing_m \vdash_m \varnothing_m, \varnothing_m}\,{\scriptstyle unit}}{\bot^* \rhd_m \varnothing_m \vdash_a \varnothing_m \rhd_m \varnothing_m}\,{\scriptstyle \rhd_2}}{\bot^* \vdash_a \varnothing_m \rhd_m \varnothing_m}\,{\scriptstyle T_a L}$$

Therefore the base case holds.

(ii) The induction hypothesis is that if by using SI_{CBI} we can simulate all proofs in DL_{CBI} of depth n, then we can also simulate all proofs of depth $n+1$. Similar to the soundness proof, the $(n+1)$th step in a proof in DL_{CBI} can only be the application of one of the rules with nonempty premise, exemplified below left. The induction hypothesis ensures that we have a proof of $\tau(premise)_S$, thus we only need to show that below right holds in SI_{CBI}.

$$\frac{premise}{conclusion}\ \rho' \qquad\qquad \frac{\dfrac{\tau(premise)_S}{\vdots}}{\tau(conclusion)_S}$$

We prove all the cases for each rules in DL_{CBI}, then we can conclude that the inductive case holds.

For the *cut* rule,

$$\begin{aligned}\tau(premise_{left})_S &= \tau(X \vdash F)_S = \tau(X)_{la} \vdash_a \tau(F)_{ra} = \tau(X)_{la} \vdash_a F\\ \tau(premise_{right})_S &= \tau(F \vdash Y)_S = \tau(F)_{la} \vdash_a \tau(Y)_{ra} = F \vdash_a \tau(Y)_{ra}\\ \tau(conclusion)_S &= \tau(X \vdash Y)_S = \tau(X)_{la} \vdash_a \tau(Y)_{ra}\end{aligned}$$

$$\frac{\tau(X)_{la} \vdash_a F \qquad F \vdash_a \tau(Y)_{ra}}{\tau(X)_{la} \vdash_a \tau(Y)_{ra}}\ cut_a$$

We do not consider structural rules that deal with $\emptyset_a$ and $\emptyset_m$ (i.e., $\emptyset_a L$, $\emptyset_a R$, $\emptyset_m L$ and $\emptyset_m R$) since we assume they are the units for additive sequent and multiplicative sequent respectively. Similarly, we assume associativity in our definition of structures, so AAL, AAR, MAL and MAR are ignored. Now we prove the rest of structural rules.

For WL,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash Z)_S = \tau(X)_{la} \vdash_a \tau(Z)_{ra}\\ \tau(conclusion)_S &= \tau(X; Y \vdash Z)_S = \tau(X; Y)_{la} \vdash_a \tau(Z)_{ra} = \tau(X)_{la}; \tau(Y)_{la} \vdash_a \tau(Z)_{ra}\end{aligned}$$

$$\frac{\tau(X)_{la} \vdash_a \tau(Z)_{ra}}{\tau(X)_{la}; \tau(Y)_{la} \vdash_a \tau(Z)_{ra}}\ WL$$

For WR,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash Z)_S = \tau(X)_{la} \vdash_a \tau(Z)_{ra}\\ \tau(conclusion)_S &= \tau(X \vdash Y; Z)_S = \tau(X)_{la} \vdash_a \tau(Y; Z)_{ra} = \tau(X)_{la} \vdash_a \tau(Y)_{ra}; \tau(Z)_{ra}\end{aligned}$$

$$\frac{\tau(X)_{la} \vdash_a \tau(Z)_{ra}}{\tau(X)_{la} \vdash_a \tau(Y)_{ra}; \tau(Z)_{ra}}\ WR$$

For CL,

$$\begin{aligned}\tau(premise)_S &= \tau(X; X \vdash Z)_S = \tau(X; X)_{la} \vdash_a \tau(Z)_{ra} = \tau(X)_{la}; \tau(X)_{la} \vdash_a \tau(Z)_{ra}\\ \tau(conclusion)_S &= \tau(X \vdash Z)_S = \tau(X)_{la} \vdash_a \tau(Z)_{ra}\end{aligned}$$

$$\frac{\tau(X)_{la}; \tau(X)_{la} \vdash_a \tau(Z)_{ra}}{\tau(X)_{la} \vdash_a \tau(Z)_{ra}}\ CL$$

For CR,

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(X \vdash Z; Z)_S = \tau(X)_{la} \vdash_a \tau(Z; Z)_{ra} = \tau(X)_{la} \vdash_a \tau(Z)_{ra}; \tau(Z)_{ra} \\ \tau(\mathit{conclusion})_S &= \tau(X \vdash Z)_S = \tau(X)_{la} \vdash_a \tau(Z)_{ra}\end{aligned}$$

$$\frac{\tau(X)_{la} \vdash_a \tau(Z)_{ra}; \tau(Z)_{ra}}{\tau(X)_{la} \vdash_a \tau(Z)_{ra}}\, CR$$

We now move on to prove logical rules.
For $\top L$,

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(\varnothing_a \vdash X)_S = \tau(\varnothing_a)_{la} \vdash_a \tau(X)_{ra} = \varnothing_a \vdash_a \tau(X)_{ra} \\ \tau(\mathit{conclusion})_S &= \tau(\top \vdash X)_S = \tau(\top)_{la} \vdash_a \tau(X)_{ra} = \top \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\frac{\varnothing_a \vdash_a \tau(X)_{ra}}{\top \vdash_a \tau(X)_{ra}}\, \top L$$

For $\bot R$,

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(X \vdash \varnothing_a)_S = \tau(X)_{la} \vdash_a \tau(\varnothing_a)_{ra} = \tau(X)_{la} \vdash_a \varnothing_a \\ \tau(\mathit{conclusion})_S &= \tau(X \vdash \bot)_S = \tau(X)_{la} \vdash_a \tau(\bot)_{ra} = \tau(X)_{la} \vdash_a \bot\end{aligned}$$

$$\frac{\tau(X)_{la} \vdash_a \varnothing_a}{\tau(X)_{la} \vdash_a \bot}\, \bot L$$

We use $F \to \bot$ to encode $\neg F$ in SI_{CBI}, thus for $\neg L$

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(\sharp F \vdash X)_S = \tau(\sharp F)_{la} \vdash_a \tau(X)_{ra} = (\varnothing_a \triangleright_a \tau(F)_{ra}) \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra} \\ &= (\varnothing_a \triangleright_a F) \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra} \\ \tau(\mathit{conclusion})_S &= \tau(\neg F \vdash X)_S = \tau(\neg F)_{la} \vdash_a \tau(X)_{ra} = F \to \bot \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\varnothing_a \triangleright_a F) \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra}}{((\varnothing_a \triangleright_a F) \triangleright_m \varnothing_m); \varnothing_a \vdash_a \tau(X)_{ra}}\, unit}{\varnothing_a \triangleright_a F \vdash_m \varnothing_m, (\varnothing_a \triangleright_a \tau(X)_{ra})}\, D_2}{\varnothing_a \triangleright_a F \vdash_m \varnothing_a \triangleright_a \tau(X)_{ra}}\, unit}{\varnothing_a; \varnothing_a \vdash_a F; \tau(X)_{ra}}\, \triangleright_1}{\varnothing_a \vdash_a F; \tau(X)_{ra}}\, unit \qquad \dfrac{}{\bot \vdash_a \varnothing_a}\, \bot L}{\varnothing_a; F \to \bot \vdash_a \tau(X)_{ra}; \varnothing_a}\, \to L}{F \to \bot \vdash_a \tau(X)_{ra}}\, unit$$

For $\neg R$,

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(X \vdash \sharp F)_S = \tau(X)_{la} \vdash_a \tau(\sharp F)_{ra} = \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (\tau(F)_{la} \triangleright_a \varnothing_a)\\ &= \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (F \triangleright_a \varnothing_a)\\ \tau(\mathit{conclusion})_S &= \tau(X \vdash \neg F)_S = \tau(X)_{la} \vdash_a \tau(\neg F)_{ra} = \tau(X)_{la} \vdash_a F \to \bot\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (F \triangleright_a \varnothing_a)}{\tau(X)_{la} \vdash_a \varnothing_a; (\varnothing_m \triangleright_m (F \triangleright_a \varnothing_a))}\,\mathit{unit}}{(\tau(X)_{la} \triangleright_a \varnothing_a), \varnothing_m \vdash_m F \triangleright_a \varnothing_a}\,D_1}{\tau(X)_{la} \triangleright_a \varnothing_a \vdash_m F \triangleright_a \varnothing_a}\,\mathit{unit}}{\tau(X)_{la}; F \vdash_a \varnothing_a; \varnothing_a}\,\triangleright_1}{\tau(X)_{la}; F \vdash_a \varnothing_a}\,\mathit{unit}}{\tau(X)_{la}; F \vdash_a \bot}\,\bot R}{\tau(X)_{la} \vdash_a F \to \bot}\,\to R$$

For $\wedge L$,

$$\begin{aligned}\tau(\mathit{premise})_S &= \tau(F; G \vdash X)_S = \tau(F; G)_{la} \vdash_a \tau(X)_{ra} = \tau(F)_{la}; \tau(G)_{la} \vdash_a \tau(X)_{ra}\\ &= F; G \vdash_a \tau(X)_{ra}\\ \tau(\mathit{conclusion})_S &= \tau(F \wedge G \vdash X)_S = \tau(F \wedge G)_{la} \vdash_a \tau(X)_{ra} = F \wedge G \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{F; G \vdash_a \tau(X)_{ra}}{F; F \wedge G \vdash_a \tau(X)_{ra}}\,\wedge L}{F \wedge G; F \wedge G \vdash_a \tau(X)_{ra}}\,\wedge L}{F \wedge G \vdash_a \tau(X)_{ra}}\,CL$$

For $\wedge R$,

$$\begin{aligned}\tau(\mathit{premise}_{left})_S &= \tau(X \vdash F)_S = \tau(X)_{la} \vdash_a \tau(F)_{ra} = \tau(X)_{la} \vdash_a F\\ \tau(\mathit{premise}_{right})_S &= \tau(Y \vdash G)_S = \tau(Y)_{la} \vdash_a \tau(G)_{ra} = \tau(Y)_{la} \vdash_a G\\ \tau(\mathit{conclusion})_S &= \tau(X; Y \vdash F \wedge G)_S = \tau(X; Y)_{la} \vdash_a \tau(F \wedge G)_{ra}\\ &= \tau(X)_{la}; \tau(Y)_{la} \vdash_a F \wedge G\end{aligned}$$

$$\dfrac{\tau(X)_{la} \vdash_a F \qquad \tau(Y)_{la} \vdash_a G}{\tau(X)_{la}; \tau(Y)_{ra} \vdash_a F \wedge G}\,\wedge R$$

For $\vee L$

$$\tau(\mathit{premise}_{left})_S = \tau(F \vdash X)_S = \tau(F)_{la} \vdash_a \tau(X)_{ra} = F \vdash_a \tau(X)_{ra}$$

$$\begin{aligned}\tau(premise_{right})_S &= \tau(G \vdash Y)_S = \tau(G)_{la} \vdash_a \tau(Y)_{ra} = G \vdash_a \tau(Y)_{ra}\\ \tau(conclusion)_S &= \tau(F \vee G \vdash X; Y)_S = \tau(F \vee G)_{la} \vdash_a \tau(X; Y)_{ra}\\ &= F \vee G \vdash_a \tau(X)_{ra}; \tau(Y)_{ra}\end{aligned}$$

$$\frac{F \vdash_a \tau(X)_{ra} \qquad G \vdash_a \tau(Y)_{ra}}{F \vee G \vdash_a \tau(X)_{ra}; \tau(Y)_{ra}} \vee L$$

For $\vee R$,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash F; G)_S = \tau(X)_{la} \vdash_a \tau(F; G)_{ra} = \tau(X)_{la} \vdash_a \tau(F)_{ra}; \tau(G)_{ra}\\ &= \tau(X)_{la} \vdash_a F; G\\ \tau(conclusion)_S &= \tau(X \vdash F \vee G)_S = \tau(X)_{la} \vdash_a \tau(F \vee G)_{ra} = \tau(X)_{la} \vdash_a F \vee G\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a F; G}{\tau(X)_{la} \vdash_a F; F \vee G} \vee R}{\tau(X)_{la} \vdash_a F \vee G; F \vee G} \vee R}{\tau(X)_{la} \vdash_a F \vee G} CR$$

For $\to L$,

$$\begin{aligned}\tau(premise_{left})_S &= \tau(X \vdash F)_S = \tau(X)_{la} \vdash_a \tau(F)_{ra} = \tau(X)_{la} \vdash_a F\\ \tau(premise_{right})_S &= \tau(G \vdash Y)_S = \tau(G)_{la} \vdash_a \tau(Y)_{ra} = G \vdash_a \tau(Y)_{ra}\\ \tau(conclusion)_S &= \tau(F \to G \vdash \sharp X; Y)_S = \tau(F \to G)_{la} \vdash_a \tau(\sharp X; Y)_{ra}\\ &= \tau(F \to G)_{la} \vdash_a \tau(\sharp X)_{ra}; \tau(Y)_{ra}\\ &= F \to G \vdash_a (\varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \varnothing_a)); \tau(Y)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{G \vdash_a \tau(Y)_{ra} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a F}{\varnothing_a; \tau(X)_{la} \vdash_m F; \varnothing_a} unit}{\varnothing_a \triangleright_a F \vdash_m \tau(X)_{la} \triangleright_a \varnothing_a} \triangleright_1}{(\varnothing_a \triangleright_a F), \varnothing_m \vdash_m \tau(X)_{la} \triangleright_a \varnothing_a} unit}{\varnothing_a \vdash_a F; (\varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \varnothing_a))} D_1}{\varnothing_a; F \to G \vdash_a (\varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \varnothing_a)); \tau(Y)_{ra}} \to L}{F \to G \vdash_a (\varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \varnothing_a)); \tau(Y)_{ra}} unit$$

For $\to R$,

$$\begin{aligned}\tau(premise)_S &= \tau(X; F \vdash G)_S = \tau(X; F)_{la} \vdash_a \tau(G)_{ra} = \tau(X)_{la}; \tau(F)_{la} \vdash_a \tau(G)_{ra}\\ &= \tau(X)_{la}; F \vdash_a G\\ \tau(conclusion)_S &= \tau(X \vdash_a F \to G)_S = \tau(X)_{la} \vdash_a \tau(F \to G)_{ra}\\ &= \tau(X)_{la} \vdash_a F \to G\end{aligned}$$

$$\frac{\tau(X)_{la}; F \vdash_a G}{\tau(X)_{la} \vdash_a F \to G} \to R$$

Multiplicative rules are significantly more complicated to prove since we translate all structures to be additive initially. The following deals with the cases for multiplicative rules.

For $\top^* L$,

$$\begin{aligned}\tau(premise)_S &= \tau(\varnothing_m \vdash X)_S = \tau(\varnothing_m)_{la} \vdash_a \tau(X)_{ra} = \varnothing_m \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra}\\ \tau(conclusion)_S &= \tau(\top^* \vdash X)_S = \tau(\top^*)_{la} \vdash_a \tau(X)_{ra} = \top^* \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varnothing_m \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra}}{(\varnothing_m \triangleright_m \varnothing_m); \varnothing_a \vdash_a \tau(X)_{ra}}\,unit}{\varnothing_m \vdash_m \varnothing_m, (\varnothing_a \triangleright_a \tau(X)_{ra})}\,D_2}{\top^* \vdash_m \varnothing_m, (\varnothing_a \triangleright_a \tau(X)_{ra})}\,\top^* L}{(\top^* \triangleright_m \varnothing_m); \varnothing_a \vdash_a \tau(X)_{ra}}\,D_2}{\top^* \triangleright_m \varnothing_m \vdash_a \tau(X)_{ra}}\,unit}{\top^* \vdash_a \tau(X)_{ra}}\,T_a L$$

For $\bot^* R$,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash \varnothing_m)_S = \tau(X)_{la} \vdash_a \tau(\varnothing_m)_{ra} = \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m \varnothing_m\\ \tau(conclusion)_S &= \tau(X \vdash \bot^*)_S = \tau(X)_{la} \vdash_a \tau(\bot^*)_{ra} = \tau(X)_{la} \vdash_a \bot^*\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a \varnothing_m \triangleright_m \varnothing_m}{\tau(X)_{la} \vdash_a \varnothing_a; (\varnothing_m \triangleright_m \varnothing_m)}\,unit}{(\tau(X)_{la} \triangleright_a \varnothing_a), \varnothing_m \vdash_m \varnothing_m}\,D_1}{(\tau(X)_{la} \triangleright_a \varnothing_a), \varnothing_m \vdash_m \bot^*}\,\bot^* R}{\tau(X)_{la} \vdash_a \varnothing_a; (\varnothing_m \triangleright_m \bot^*)}\,D_1}{\tau(X)_{la} \vdash_a \varnothing_m \triangleright_m \bot^*}\,unit}{\tau(X)_{la} \vdash_a \bot^*}\,T_a R$$

We use $F \multimap \bot^*$ to encode $\sim F$, thus for the $\sim L$ rule,

$$\begin{aligned}\tau(premise)_S &= \tau(\flat F \vdash X)_S = \tau(\flat F)_{la} \vdash_a \tau(X)_{ra} = \varnothing_m \triangleright_m \tau(F)_{rm} \vdash_a \tau(X)_{ra}\\ &= \varnothing_m \triangleright_m F \vdash_a \tau(X)_{ra}\\ \tau(conclusion)_S &= \tau(\sim F \vdash X)_S = \tau(\sim F)_{la} \vdash_a \tau(X)_{ra} = F \multimap \bot^* \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\emptyset_m \rhd_m F \vdash_a \tau(X)_{ra}}{(\emptyset_m \rhd_m F);\emptyset_a \vdash_a \tau(X)_{ra}}\,unit}{\emptyset_m \vdash_m F,(\emptyset_a \rhd_a \tau(X)_{ra})}\,D_1 \quad \dfrac{}{\bot^* \vdash_m \emptyset_m}\,\bot^*L}{\emptyset_m, F{-\!\!*}\,\bot^* \vdash_m \emptyset_m,(\emptyset_a \rhd_a \tau(X)_{ra})}\,{-\!\!*}\,L}{F{-\!\!*}\,\bot^* \vdash_m \emptyset_m,(\emptyset_a \rhd_a \tau(X)_{ra})}\,unit}{((F{-\!\!*}\,\bot^*) \rhd_m \emptyset_m);\emptyset_a \vdash_a \tau(X)_{ra}}\,D_2}{(F{-\!\!*}\,\bot^*) \rhd_m \emptyset_m \vdash_a \tau(X)_{ra}}\,unit}{F{-\!\!*}\,\bot^* \vdash_a \tau(X)_{ra}}\,T_aL$$

For $\sim R$,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash \flat F)_S = \tau(X)_{la} \vdash_a \tau(\flat F)_{ra} = \tau(X)_{la} \vdash_a \tau(F)_{lm} \rhd_m \emptyset_m \\ &= \tau(X)_{la} \vdash_a F \rhd_m \emptyset_m \\ \tau(conclusion)_S &= \tau(X \vdash {\sim} F)_S = \tau(X)_{la} \vdash_a \tau({\sim} F)_{ra} = \tau(X)_{la} \vdash_a F{-\!\!*}\,\bot^*\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a F \rhd_m \emptyset_m}{\tau(X)_{la} \vdash_a \emptyset_a;(F \rhd_m \emptyset_m)}\,unit}{(\tau(X)_{la} \rhd_a \emptyset_a), F \vdash_m \emptyset_m}\,D_1}{(\tau(X)_{la} \rhd_a \emptyset_a), F \vdash_m \bot^*}\,\bot^*R}{(\tau(X)_{la} \rhd_a \emptyset_a), \emptyset_m, F \vdash_m \bot^*}\,unit}{(\tau(X)_{la} \rhd_a \emptyset_a), \emptyset_m \vdash_m F{-\!\!*}\,\bot^*}\,{-\!\!*}\,R}{\tau(X)_{la} \vdash_a \emptyset_a;(\emptyset_m \rhd_m (F{-\!\!*}\,\bot^*))}\,D_1}{\tau(X)_{la} \vdash_a \emptyset_m \rhd_m (F{-\!\!*}\,\bot^*)}\,unit}{\tau(X)_{la} \vdash_a F{-\!\!*}\,\bot^*}\,T_aR$$

For $*L$,

$$\begin{aligned}\tau(premise)_S &= \tau(F,G \vdash X)_S = \tau(F,G)_{la} \vdash_a \tau(X)_{ra} \\ &= (\tau(F)_{lm},\tau(G)_{lm}) \rhd_m \emptyset_m \vdash_a \tau(X)_{ra} = (F,G) \rhd_m \emptyset_m \vdash_a \tau(X)_{ra} \\ \tau(conclusion)_S &= \tau(F * G \vdash X)_S = \tau(F * G)_{la} \vdash_a \tau(X)_{ra} = F * G \vdash_a \tau(X)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(F,G) \rhd_m \emptyset_m \vdash_a \tau(X)_{ra}}{((F,G) \rhd_m \emptyset_m);\emptyset_a \vdash_a \tau(X)_{ra}}\,unit}{F,G \vdash_m \emptyset_m,(\emptyset_a \rhd_a \tau(X)_{ra})}\,D_2}{F * G \vdash_m \emptyset_m,(\emptyset_a \rhd_a \tau(X)_{ra})}\,*L}{((F * G) \rhd_m \emptyset_m);\emptyset_a \vdash_a \tau(X)_{ra}}\,D_2}{(F * G) \rhd_m \emptyset_m \vdash_a \tau(X)_{ra}}\,unit}{F * G \vdash_a \tau(X)_{ra}}\,T_aL$$

For $*R$,

$$\begin{aligned}\tau(premise_{left})_S &= \tau(X \vdash F)_S = \tau(X)_{la} \vdash_a \tau(F)_{ra} = \tau(X)_{la} \vdash_a F\\ \tau(premise_{right})_S &= \tau(Y \vdash G)_S = \tau(Y)_{la} \vdash_a \tau(G)_{ra} = \tau(Y)_{la} \vdash_a G\\ \tau(conclusion)_S &= \tau(X, Y \vdash F * G)_S = \tau(X, Y)_{la} \vdash_a \tau(F * G)_{ra}\\ &= (\tau(X)_{lm}, \tau(Y)_{lm}) \triangleright_m \varnothing_m \vdash_a F * G\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(Y)_{la} \vdash_a G}{\tau(Y)_{lm} \vdash_m G}\,\text{Lemma 3.2.4} \quad \dfrac{\tau(X)_{la} \vdash_a F}{\tau(X)_{lm} \vdash_m F}\,\text{Lemma 3.2.4}}{\tau(X)_{lm}, \tau(Y)_{lm} \vdash_m F * G}\,{*R}}{\tau(X)_{lm}, \tau(Y)_{lm}, \varnothing_m \vdash_m \varnothing_m, F * G}\,unit}{(\tau(X)_{lm}, \tau(Y)_{lm}) \triangleright_m \varnothing_m \vdash_a \varnothing_m \triangleright_m (F * G)}\,{\triangleright_2}}{(\tau(X)_{lm}, \tau(Y)_{lm}) \triangleright_m \varnothing_m \vdash_a F * G}\,T_aR$$

For $\vee^* L$,

$$\begin{aligned}\tau(premise_{left})_S &= \tau(F \vdash X)_S = \tau(F)_{la} \vdash_a \tau(X)_{ra} = F \vdash_a \tau(X)_{ra}\\ \tau(premise_{right})_S &= \tau(G \vdash Y)_S = \tau(G)_{la} \vdash_a \tau(Y)_{ra} = G \vdash_a \tau(Y)_{ra}\\ \tau(conclusion)_S &= \tau(F \vee^* G \vdash X, Y)_S = \tau(F \vee^* G)_{la} \vdash_a \tau(X, Y)_{ra}\\ &= F \vee^* G \vdash_a \varnothing_m \triangleright_m (\tau(X)_{rm}, \tau(Y)_{rm})\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{G \vdash_a \tau(Y)_{ra}}{G \vdash_m \tau(Y)_{rm}}\,\text{Lemma 3.2.4} \quad \dfrac{F \vdash_a \tau(X)_{ra}}{F \vdash_m \tau(X)_{rm}}\,\text{Lemma 3.2.4}}{F \vee^* G \vdash_m \tau(X)_{rm}, \tau(Y)_{rm}}\,{\vee^* L}}{F \vee^* G, \varnothing_m \vdash_m \varnothing_m, \tau(X)_{rm}, \tau(Y)_{rm}}\,unit}{(F \vee^* G) \triangleright_m \varnothing_m \vdash_a \varnothing_m \triangleright_m (\tau(X)_{rm}, \tau(Y)_{rm})}\,{\triangleright_2}}{F \vee^* G \vdash_a \varnothing_m \triangleright_m (\tau(X)_{rm}, \tau(Y)_{rm})}\,T_aL$$

For $\vee^* R$,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash F, G)_S = \tau(X)_{la} \vdash_a \tau(F, G)_{ra}\\ &= \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (\tau(F)_{rm}, \tau(G)_{rm}) = \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (F, G)\\ \tau(conclusion)_S &= \tau(X \vdash F \vee^* G)_S = \tau(X)_{la} \vdash_a \tau(F \vee^* G)_{ra} = \tau(X)_{la} \vdash_a F \vee^* G\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a \varnothing_m \rhd_m (F, G)}{\tau(X)_{la} \vdash_a \varnothing_a ; (\varnothing_m \rhd_m (F, G))}\,unit}{(\tau(X)_{la} \rhd_a \varnothing_a), \varnothing_m \vdash_m F, G}\,D_1}{(\tau(X)_{la} \rhd_a \varnothing_a), \varnothing_m \vdash_m F \vee^* G}\,\vee^* R}{\tau(X)_{la} \vdash_a \varnothing_a ; (\varnothing_m \rhd_m (F \vee^* G))}\,D_1}{\tau(X)_{la} \vdash_a \varnothing_m \rhd_m (F \vee^* G)}\,unit}{\tau(X)_{la} \vdash_a F \vee^* G}\,T_a R$$

For $-\!\!*\ L$,

$$\begin{aligned}
\tau(premise_{left})_S &= \tau(X \vdash F)_S = \tau(X)_{la} \vdash_a \tau(F)_{ra} = \tau(X)_{la} \vdash_a F \\
\tau(premise_{right})_S &= \tau(G \vdash Y)_S = \tau(G)_{la} \vdash_a \tau(Y)_{ra} = G \vdash_a \tau(Y)_{ra} \\
\tau(conclusion)_S &= \tau(F -\!\!* \ G \vdash \flat X, Y)_S = \tau(F -\!\!* \ G)_{la} \vdash_a \tau(\flat X, Y)_{ra} \\
&= F -\!\!* \ G \vdash_a \varnothing_m \rhd_m (\tau(\flat X)_{rm}, \tau(Y)_{rm}) \\
&= F -\!\!* \ G \vdash_a \varnothing_m \rhd_m (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m), \tau(Y)_{rm})
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a F}{\tau(X)_{lm} \vdash_m F}\,\text{Lemma 3.2.4}}{\varnothing_m, \tau(X)_{lm} \vdash_m F, \varnothing_m}\,unit}{\varnothing_m \rhd_m F \vdash_a \tau(X)_{lm} \rhd_m \varnothing_m}\,\rhd_2}{(\varnothing_m \rhd_m F); \varnothing_a \vdash_a \tau(X)_{lm} \rhd_m \varnothing_m}\,unit}{\varnothing_m \vdash_m F, (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m))}\,D_2 \qquad \dfrac{G \vdash_a \tau(Y)_{ra}}{G \vdash_m \tau(Y)_{rm}}\,\text{Lemma 3.2.4}}{F -\!\!* \ G, \varnothing_m \vdash_m (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m)), \tau(Y)_{rm}}\,-\!\!* \ L}{F -\!\!* \ G, \varnothing_m \vdash_m \varnothing_m, (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m)), \tau(Y)_{rm}}\,unit}{(F -\!\!* \ G) \rhd_m \varnothing_m \vdash_a \varnothing_m \rhd_m (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m), \tau(Y)_{rm})}\,\rhd_2}{F -\!\!* \ G \vdash_a \varnothing_m \rhd_m (\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \varnothing_m), \tau(Y)_{rm})}\,T_a L$$

For $-\!\!*\ R$,

$$\begin{aligned}
\tau(premise)_S &= \tau(X, F \vdash G)_S = \tau(X, F)_{la} \vdash_a \tau(G)_{ra} = (\tau(X)_{lm}, \tau(F)_{lm}) \rhd_m \varnothing_m \vdash_a G \\
&= (\tau(X)_{lm}, F) \rhd_m \varnothing_m \vdash_a G \\
\tau(conclusion)_S &= \tau(X \vdash F -\!\!* \ G)_S = \tau(X)_{la} \vdash_a \tau(F -\!\!* \ G)_{ra} \\
&= \tau(X)_{la} \vdash_a F -\!\!* \ G
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\tau(X)_{lm}, F) \rhd_m \varnothing_m \vdash_a G}{(\tau(X)_{lm}, F) \rhd_m \varnothing_m \vdash_a \varnothing_m \rhd_m G}\,{\scriptstyle T_aR}}{\tau(X)_{lm}, \varnothing_m, F \vdash_m G, \varnothing_m}\,{\scriptstyle \rhd_2}}{\tau(X)_{lm}, \varnothing_m, F \vdash_m G}\,{\scriptstyle unit}}{(\tau(X)_{la} \rhd_a \varnothing_a), \varnothing_m, F \vdash_m G}\,\text{Lemma 3.2.3}}{(\tau(X)_{la} \rhd_a \varnothing_a), \varnothing_m \vdash_m F \multimap G}\,{\scriptstyle \multimap R}}{\tau(X)_{la} \vdash_a \varnothing_a; (\varnothing_m \rhd_m (F \multimap G))}\,{\scriptstyle D_1}}{\tau(X)_{la} \vdash_a \varnothing_m \rhd_m (F \multimap G)}\,{\scriptstyle unit}}{\tau(X)_{la} \vdash_a F \multimap G}\,{\scriptstyle T_aR}$$

Since we assume commutativity for structures in SI_{CBI}, (almost) half of the display rules in DL_{CBI} can be neglected. Further, $AD3b$, $AD3a$, $MD3b$, and $MD3a$ can be seen as special cases with multiple applications of others. Therefore We only have to prove the cases for $AD1b$, $AD2b$, $MD1b$, and $MD2b$, and show that they are invertible.

For $AD1b$,

$$\begin{aligned}\tau(premise)_S &= \tau(X \vdash \sharp Y; Z)_S = \tau(X)_{la} \vdash_a \tau(\sharp Y; Z)_{ra} = \tau(X)_{la} \vdash_a \tau(\sharp Y)_{ra}; \tau(Z)_{ra} \\ &= \tau(X)_{la} \vdash_a (\varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a)); \tau(Z)_{ra} \\ \tau(conclusion)_S &= \tau(Y; X \vdash Z)_S = \tau(Y; X)_{la} \vdash_a \tau(Z)_{ra} = \tau(Y)_{la}; \tau(X)_{la} \vdash_a \tau(Z)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a \tau(Z)_{ra}; (\varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a))}{\tau(X)_{la}; \varnothing_a \vdash_a \tau(Z)_{ra}; (\varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a))}\,{\scriptstyle unit}}{\tau(X)_{la} \rhd_a \tau(Z)_{ra} \vdash_m \varnothing_a \rhd_a (\varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a))}\,{\scriptstyle \rhd_1}}{\tau(X)_{la} \rhd_a \tau(Z)_{ra} \vdash_m \varnothing_m, (\varnothing_a \rhd_a (\varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a)))}\,{\scriptstyle unit}}{((\tau(X)_{la} \rhd_a \tau(Z)_{ra}) \rhd_m \varnothing_m); \varnothing_a \vdash_a \varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a)}\,{\scriptstyle D_2}}{(\tau(X)_{la} \rhd_a \tau(Z)_{ra}) \rhd_m \varnothing_m \vdash_a \varnothing_m \rhd_m (\tau(Y)_{la} \rhd_a \varnothing_a)}\,{\scriptstyle unit}}{(\tau(X)_{la} \rhd_a \tau(Z)_{ra}), \varnothing_m \vdash_m \varnothing_m, (\tau(Y)_{la} \rhd_a \varnothing_a)}\,{\scriptstyle \rhd_2}}{\tau(X)_{la} \rhd_a \tau(Z)_{ra} \vdash_m \tau(Y)_{la} \rhd_a \varnothing_a}\,{\scriptstyle unit}}{\tau(Y)_{la}; \tau(X)_{la} \vdash_a \varnothing_a; \tau(Z)_{ra}}\,{\scriptstyle \rhd_1}}{\tau(Y)_{la}; \tau(X)_{la} \vdash_a \tau(Z)_{ra}}\,{\scriptstyle unit}$$

The case for $AD2b$ is completely dual:

$$\begin{aligned}\tau(premise)_S &= \tau(X; \sharp Y \vdash Z)_S = \tau(X; \sharp Y)_{la} \vdash_a \tau(Z)_{ra} = \tau(X)_{la}; \tau(\sharp Y)_{la} \vdash_a \tau(Z)_{ra} \\ &= \tau(X)_{la}; ((\varnothing_a \rhd_a \tau(Y)_{ra}) \rhd_m \varnothing_m) \vdash_a \tau(Z)_{ra} \\ \tau(conclusion)_s &= \tau(X \vdash Z; Y)_S = \tau(X)_{la} \vdash_a \tau(Z; Y)_{ra} = \tau(X)_{la} \vdash_a \tau(Z)_{ra}; \tau(Y)_{ra}\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{((\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m); \tau(X)_{la} \vdash_a \tau(Z)_{ra}}{((\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m); \tau(X)_{la} \vdash_a \varnothing_a; \tau(Z)_{ra}}\, unit}{((\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m) \triangleright_a \varnothing_a \vdash_m \tau(X)_{la} \triangleright_a \tau(Z)_{ra}}\, \triangleright_1}{(((\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m) \triangleright_a \varnothing_a), \varnothing_m \vdash_m \tau(X)_{la} \triangleright_a \tau(Z)_{ra}}\, unit}{(\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m \vdash_a \varnothing_a; (\varnothing_m \triangleright_m (\tau(X)_{la} \triangleright_a \tau(Z)_{ra}))}\, D_1}{(\varnothing_a \triangleright_a \tau(Y)_{ra}) \triangleright_m \varnothing_m \vdash_a \varnothing_a \triangleright_m (\tau(X)_{la} \triangleright_a \tau(Z)_{ra})}\, unit}{(\varnothing_a \triangleright_a \tau(Y)_{ra}), \varnothing_m \vdash_m \varnothing_m, (\tau(X)_{la} \triangleright_a \tau(Z)_{ra})}\, \triangleright_2}{\varnothing_a \triangleright_a \tau(Y)_{ra} \vdash_m \tau(X)_{la} \triangleright_a \tau(Z)_{ra}}\, unit}{\tau(X)_{la}; \varnothing_a \vdash_a \tau(Z)_{ra}; \tau(Y)_{ra}}\, \triangleright_1}{\tau(X)_{la} \vdash_a \tau(Z)_{ra}; \tau(Y)_{ra}}\, unit$$

For $MD1b$,

$$\begin{aligned}
\tau(premise)_S &= \tau(X \vdash \flat Y, Z)_S = \tau(X)_{la} \vdash_a \tau(\flat Y, Z)_{ra} \\
&= \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (\tau(\flat Y)_{rm}, \tau(Z)_{rm}) \\
&= \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m ((\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)), \tau(Z)_{rm}) \\
\tau(conclusion)_S &= \tau(Y, X \vdash Z)_S = \tau(Y, X)_{la} \vdash_a \tau(Z)_{ra} = (\tau(Y)_{lm}, \tau(X)_{lm}) \triangleright_m \varnothing_m \vdash_a \tau(Z)_{ra}
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\tau(X)_{la} \vdash_a \varnothing_m \triangleright_m ((\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)), \tau(Z)_{rm})}{\tau(X)_{la} \vdash_a \varnothing_a; (\varnothing_m \triangleright_m ((\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)), \tau(Z)_{rm}))}\, unit}{(\tau(X)_{la} \triangleright_a \varnothing_a), \varnothing_m \vdash_m (\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)), \tau(Z)_{rm}}\, D_1}{\tau(X)_{lm}, \varnothing_m \vdash_m (\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)), \tau(Z)_{rm}}\, \text{Lemma 3.2.3}}{\tau(X)_{lm} \triangleright_m \tau(Z)_{rm} \vdash_a \varnothing_m \triangleright_m (\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m))}\, \triangleright_2}{\tau(X)_{lm} \triangleright_m \tau(Z)_{rm} \vdash_a \varnothing_a; (\varnothing_m \triangleright_m (\varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)))}\, unit}{((\tau(X)_{lm} \triangleright_m \tau(Z)_{rm}) \triangleright_a \varnothing_a), \varnothing_m \vdash_m \varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)}\, D_1}{(\tau(X)_{lm} \triangleright_m \tau(Z)_{rm}) \triangleright_a \varnothing_a \vdash_m \varnothing_a \triangleright_a (\tau(Y)_{lm} \triangleright_m \varnothing_m)}\, unit}{(\tau(X)_{lm} \triangleright_m \tau(Z)_{rm}); \varnothing_a \vdash_a \varnothing_a; (\tau(Y)_{lm} \triangleright_m \varnothing_m)}\, \triangleright_1}{\tau(X)_{lm} \triangleright_m \tau(Z)_{rm} \vdash_a \tau(Y)_{lm} \triangleright_m \varnothing_m}\, unit}{\tau(Y)_{lm}, \tau(X)_{lm} \vdash_m \varnothing_m, \tau(Z)_{rm}}\, \triangleright_2}{\tau(Y)_{lm}, \tau(X)_{lm} \vdash_m \varnothing_m, (\varnothing_a \triangleright_a \tau(Z)_{ra})}\, \text{Lemma 3.2.3}}{((\tau(Y)_{lm}, \tau(X)_{lm}) \triangleright_m \varnothing_m); \varnothing_a \vdash_a \tau(Z)_{ra}}\, D_2}{(\tau(Y)_{lm}, \tau(X)_{lm}) \triangleright_m \varnothing_m \vdash_a \tau(Z)_{ra}}\, unit$$

It is dual for $MD2b$,

$$\begin{aligned}
\tau(premise)_S &= \tau(X, \flat Y \vdash Z)_S = \tau(X, \flat Y)_{la} \vdash_a \tau(Z)_{ra} = (\tau(X)_{lm}, \tau(\flat Y)_{lm}) \triangleright_m \varnothing_m \vdash_a \tau(Z)_{ra} \\
&= (\tau(X)_{lm}, ((\varnothing_m \triangleright_m \tau(Y)_{rm}) \triangleright_a \varnothing_a)) \triangleright_m \varnothing_m \vdash_a \tau(Z)_{ra} \\
\tau(conclusion)_s &= \tau(X \vdash Z, Y)_S = \tau(X)_{la} \vdash_a \tau(Z, Y)_{ra} = \tau(X)_{la} \vdash_a \varnothing_m \triangleright_m (\tau(Z)_{rm}, \tau(Y)_{rm})
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\tau(X)_{lm},((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a)) \rhd_m \varnothing_m \vdash_a \tau(Z)_{ra}}{((\tau(X)_{lm},((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a)) \rhd_m \varnothing_m);\varnothing_a \vdash_a \tau(Z)_{ra}}\,unit}{\tau(X)_{lm},((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a) \vdash_m \varnothing_m,(\varnothing_a \rhd_a \tau(Z)_{ra})}\,D_2}{\tau(X)_{lm},((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a) \vdash_m \varnothing_m,\tau(Z)_{rm}}\,\text{Lemma 3.2.3}}{((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a) \rhd_m \varnothing_m \vdash_a \tau(X)_{lm} \rhd_m \tau(Z)_{rm}}\,\rhd_2}{(((\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a) \rhd_m \varnothing_m);\varnothing_a \vdash_a \tau(X)_{lm} \rhd_m \tau(Z)_{rm}}\,unit}{(\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a \vdash_m \varnothing_m,(\varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \tau(Z)_{rm}))}\,D_2}{(\varnothing_m \rhd_m \tau(Y)_{rm}) \rhd_a \varnothing_a \vdash_m \varnothing_a \rhd_a (\tau(X)_{lm} \rhd_m \tau(Z)_{rm})}\,unit}{(\varnothing_m \rhd_m \tau(Y)_{rm});\varnothing_a \vdash_a \varnothing_a;(\tau(X)_{lm} \rhd_m \tau(Z)_{rm})}\,\rhd_1}{\varnothing_m \rhd_m \tau(Y)_{rm} \vdash_a \tau(X)_{lm} \rhd_m \tau(Z)_{rm}}\,unit}{\tau(X)_{lm},\varnothing_m \vdash_m \tau(Z)_{rm},\tau(Y)_{rm}}\,\rhd_2}{(\tau(X)_{la} \rhd_a \varnothing_a),\varnothing_m \vdash_m \tau(Z)_{rm},\tau(Y)_{rm}}\,\text{Lemma 3.2.3}}{\tau(X)_{la} \vdash_a \varnothing_a;(\varnothing_m \rhd_m (\tau(Z)_{rm},\tau(Y)_{rm}))}\,D_1}{\tau(X)_{la} \vdash_a \varnothing_m \rhd_m (\tau(Z)_{rm},\tau(Y)_{rm})}\,unit$$

All the rules we use to prove the display rules are reversible (including the lemmas used), thus these display rules are also reversible.

We have proved that no matter which rule is used in the $(n+1)$th step of the derivation in DL_{CBI}, we can always simulate the derivation in SI_{CBI}, thus the inductive case is proved. □

We may translate two equivalent structures in DL_{CBI} to structures in different forms in SI_{CBI}, but the translated structures should agree on their provability.

Lemma 3.2.6. *If two structures in DL_{CBI} are equivalent by the structural rules in DL_{CBI}, then they will either be translated to the same nested sequent in SI_{CBI} or the results of translation are inter-derivable in SI_{CBI}.*

Proof. The cases where two structures in DL_{CBI} are equivalent by weakening, contraction, and display postulates are proved in Lemma 3.2.5. Now the remaining cases are the equivalences by associativity rules, commutativity rules and unit rules.

The cases of associativity rules and commutativity rules follow directly by applying the translation.

Equivalence by additive unit rules in DL_{CBI} follow directly by translating as well, for multiplicative units, Lemma 3.2.3 is sufficient for the proof. □

Finally, we are ready to state the completeness theorem for SI_{CBI}. The completeness is shown w.r.t. the display calculus DL_{CBI}, as the latter is proven complete for CBI.

Theorem 3.2.7 (Completeness)**.** *If a formula F is provable in DL_{CBI} (i.e., $\varnothing_a \vdash F$ is provable in DL_{CBI}), then F is provable in SI_{CBI} (i.e., $\varnothing_a \vdash_a F$ is provable in SI_{CBI}).*

Proof. Since $\emptyset_a \vdash F$ is provable in DL_{CBI}, then by using Lemma 3.2.5, $\tau(\emptyset_a \vdash F)_S$ is provable in SI_{CBI}. By our translation, we have:

$$\tau(\emptyset_a \vdash F)_S = \tau(\emptyset_a)_{la} \vdash_a \tau(F)_{ra} = \emptyset_a \vdash_a F$$

therefore we have a proof of $\emptyset_a \vdash_a F$ in SI_{CBI}. □

3.2.4 Display Property of SI_{CBI}

At the end of Section 2.3 we briefly discussed the display property for Brotherston's calculi, including DL_{CBI}. This property describes the ability to "display" structures in the sequent by using display postulates, it is fundamental for display calculi, and it is also enjoyed by our shallow nested sequent calculus. In the following we focus on displaying nested sequents rather than arbitrary structures, since our rules can be applied on the structures as long as the sequent they belong to is displayed to the top level. To show the display property of the shallow system SI_{CBI}, we need three lemmas, as shown below.

The first lemma deals with the cases where a structure in a simple context needs to be displayed.

Lemma 3.2.8. *For any simple context $\Sigma[]$, any nested sequent structures Z_1 and Z_2, any nested sequent $X_1 \triangleright_a X_2$ ($Y_1 \triangleright_m Y_2$ resp.), there exist structures W_1 and W_2 such that the sequent $W_1; X_1 \vdash_a X_2; W_2$ ($W_1, Y_1 \vdash_m Y_2, W_2$ resp.) is derivable from either of $\Sigma[X_1 \triangleright_a X_2], Z_1 \vdash_m Z_2$ and $Z_1 \vdash_m \Sigma[X_1 \triangleright_a X_2], Z_2$ ($\Sigma[Y_1 \triangleright_m Y_2]; Z_1 \vdash_a Z_2$ and $Z_1 \vdash_a \Sigma[Y_1 \triangleright_m Y_2]; Z_2$ resp.) by using only D_1/D_2 rules. The reverse direction also holds.*

Proof. By induction on the size of context $\Sigma[]$.

- Base case: if $\Sigma[] = []$.
  - for $X_1 \triangleright_a X_2$, we have the following inference if $\Sigma[]$ is on the left hand side.

    $$\frac{\emptyset_a; X_1 \vdash_a X_2; (Z_1 \triangleright_m Z_2)}{(X_1 \triangleright_a X_2), Z_1 \vdash_m Z_2}\ D_1$$

    thus $W_1 = \emptyset_a$ and $W_2 = Z_1 \triangleright_m Z_2$. If $\Sigma[]$ is on the right hand side, then

    $$\frac{(Z_1 \triangleright_m Z_2); X_1 \vdash_a X_2; \emptyset_a}{Z_1 \vdash_m (X_1 \triangleright_a X_2), Z_2}\ D_2$$

    thus $W_1 = Z_1 \triangleright_m Z_2$ and $W_2 = \emptyset_a$.
  - for $Y_1 \triangleright_m Y_2$, if $\Sigma[]$ is on the left hand side, then

    $$\frac{\emptyset_m, Y_1 \vdash_m Y_2, (Z_1 \triangleright_a Z_2)}{(Y_1 \triangleright_m Y_2); Z_1 \vdash_a Z_2}\ D_2$$

so $W_1 = \emptyset_m$ and $W_2 = Z_1 \triangleright_a Z_2$. If $\Sigma[]$ is on the right hand side, then

$$\frac{(Z_1 \triangleright_a Z_2), Y_1 \vdash_m Y_2, \emptyset_m}{Z_1 \vdash_a (Y_1 \triangleright_m Y_2); Z_2}\ D_1$$

so $W_1 = Z_1 \triangleright_a Z_2$ and $W_2 = \emptyset_m$.

- Inductive cases:
  - if $\Sigma[] = \Sigma'[]; W$, as a legitimate nested sequent structure, the sequent nested inside can only be multiplicative. Then we can apply the induction hypothesis on $\Sigma'[]$ as shown below.

$$\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{IH} \\ \Sigma'[Y_1 \triangleright_m Y_2]; W; Z_1 \vdash_a Z_2 \end{array}$$

    The case where $\Sigma[]$ is on the right hand side is symmetric.

$$\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{IH} \\ Z_1 \vdash_a \Sigma'[Y_1 \triangleright_m Y_2]; W; Z_2 \end{array}$$

  - if $\Sigma[] = \Sigma'[], W$, then the sequent nested inside must be additive. Therefore we apply the induction hypothesis to $\Sigma'[]$ similarly.

$$\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{IH} \\ \Sigma'[X_1 \triangleright_a X_2], W, Z_1 \vdash_m Z_2 \end{array}$$

    The cases where $\Sigma[]$ is on the right hand side is symmetric.

$$\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{IH} \\ Z_1 \vdash_m \Sigma'[X_1 \triangleright_a X_2], W, Z_2 \end{array}$$

The reserve direction holds because we only use D_1 and D_2 rules in this proof, both of these rules are reversible. □

The second lemma considers displaying a structure in a positive context.

Lemma 3.2.9. *For any positive context $\Sigma^+[]$, any nested sequent structures Z_1 and Z_2, any nested sequent $X_1 \triangleright_a X_2$, there exist structures W_1 and W_2 such that the sequent $W_1; X_1 \vdash_a X_2; W_2$ is derivable from one of the following sequents using only D_1 / D_2 rules:*

1. $\Sigma[X_1 \triangleright_a X_2], Z_1 \vdash_m Z_2$

2. $\Sigma[X_1 \rhd_a X_2]; Z_1 \vdash_a Z_2$
3. $Z_1 \vdash_m \Sigma[X_1 \rhd_a X_2], Z_2$
4. $Z_1 \vdash_a \Sigma[X_1 \rhd_a X_2]; Z_2$

and for any nested sequent $Y_1 \rhd_m Y_2$, the sequent $W_1, Y_1 \vdash_m Y_2, W_2$ is derivable from one of the following sequents using only D_1/D_2 rules:

1. $\Sigma[Y_1 \rhd_m Y_2], Z_1 \vdash_m Z_2$
2. $\Sigma[Y_1 \rhd_m Y_2]; Z_1 \vdash_a Z_2$
3. $Z_1 \vdash_m \Sigma[Y_1 \rhd_m Y_2], Z_2$
4. $Z_1 \vdash_a \Sigma[Y_1 \rhd_m Y_2]; Z_2$.

Proof. By induction on the size of context $\Sigma[]$.

- Base case:
  - For $X_1 \rhd_a X_2$, the context $\Sigma[]$ can only be of the form $U \rhd_m V, []$. Let us go through the four cases in the lemma.
    1. This case cannot occur (the resultant sequent is not well-defined).
    2. We have the following inference:
    $$\dfrac{\dfrac{(U \rhd_m V, (Z_1 \rhd_a Z_2)); X_1 \vdash_a X_2; \emptyset_a}{U \vdash_m V, (X_1 \rhd_a X_2), (Z_1 \rhd_a Z_2)}\, D_2}{(U \rhd_m V, (X_1 \rhd_a X_2)); Z_1 \vdash_a Z_2}\, D_2$$
    then $W_1 = U \rhd_m V, (Z_1 \rhd_a Z_2)$, and $W_2 = \emptyset_a$.
    3. This case cannot occur.
    4. We have the following inference:
    $$\dfrac{\dfrac{((Z_1 \rhd_a Z_2), U \rhd_m V); X_1 \vdash_a X_2; \emptyset_a}{(Z_1 \rhd_a Z_2), U \vdash_m V, (X_1 \rhd_a X_2)}\, D_2}{Z_1 \vdash_a (U \rhd_m V, (X_1 \rhd_a X_2)); Z_2}\, D_1$$
    then $W_1 = (Z_1 \rhd_a Z_2), U \rhd_m V$ and $W_2 = \emptyset_a$.
  - For $Y_1 \rhd_m Y_2$, the context $\Sigma[]$ can only be of the form $U \rhd_a V; []$. We have the following for the four cases in the lemma:
    1. We have the following inference:
    $$\dfrac{\dfrac{(U \rhd_a V; (Z_1 \rhd_m Z_2)), Y_1 \vdash_m Y_2, \emptyset_m}{U \vdash_a V; (Y_1 \rhd_m Y_2); (Z_1 \rhd_m Z_2)}\, D_1}{(U \rhd_a V; (Y_1 \rhd_m Y_2)), Z_1 \vdash_m Z_2}\, D_1$$
    then $W_1 = U \rhd_a V; (Z_1 \rhd_m Z_2)$ and $W_2 = \emptyset_m$.
    2. In this case structure cannot occur.
    3. We have the following inference:

$$\dfrac{\dfrac{((Z_1 \rhd_m Z_2); U \rhd_a V), Y_1 \vdash_m Y_2, \varnothing_m}{(Z_1 \rhd_m Z_2); U \vdash_a V; (Y_1 \rhd_m Y_2)}\, D_1}{Z_1 \vdash_m (U \rhd_a V; (Y_1 \rhd_m Y_2)), Z_2}\, D_2$$

then $W_1 = (Z_1 \rhd_m Z_2); U \rhd_a V$ and $W_2 = \varnothing_m$.

4. This case cannot occur.

- Inductive cases:
  - For $X_1 \rhd_a X_2$:
    - If $\Sigma[] = \Sigma'[], W$ or $\Sigma[] = \Sigma'[]; W$, then $\Sigma'[]$ must be positive. So we can apply the induction hypothesis directly as in Lemma 3.2.8.
    - If $\Sigma[] = \Sigma'[] \rhd_a U$, then $\Sigma'[]$ must be positive, thus we can apply the induction hypothesis on $\Sigma'[]$, the four cases in the lemma are shown as follows.
      1. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ \varnothing_a; \Sigma'[X_1 \rhd_a X_2] \vdash_a U; (Z_1 \rhd_m Z_2) \end{array}}{(\Sigma'[X_1 \rhd_a X_2] \rhd_a U), Z_1 \vdash_m Z_2}\, D_1$$

      2. This case cannot occur.
      3. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_m Z_2); \Sigma'[X_1 \rhd_a X_2] \vdash_a U \end{array}}{Z_1 \vdash_m Z_2, (\Sigma'[X_1 \rhd_a X_2] \rhd_a U)}\, D_2$$

      4. This case cannot occur.
    - If $\Sigma[] = \Sigma'[] \rhd_m U$, then $\Sigma'[]$ must be positive, thus we can apply the induction hypothesis on $\Sigma'[]$.
      1. This case cannot occur.
      2. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ \varnothing_m, \Sigma'[X_1 \rhd_a X_2] \vdash_m U, (Z_1 \rhd_a Z_2) \end{array}}{(\Sigma'[X_1 \rhd_a X_2] \rhd_m U); Z_1 \vdash_a Z_2}\, D_2$$

      3. This case cannot occur.
      4. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_a Z_2), \Sigma'[X_1 \rhd_a X_2] \vdash_m U \end{array}}{Z_1 \vdash_a Z_2; (\Sigma'[X_1 \rhd_a X_2] \rhd_m U)} \, D_1$$

* If $\Sigma[] = U \rhd_a \Sigma'[]$, then $\Sigma'[]$ must be positive. It cannot be simple because then the structure will not be well-defined. So we can apply the induction hypothesis on $\Sigma'[]$. The four cases run as below.
  1. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ U \vdash_a \Sigma'[X_1 \rhd_a X_2]; (Z_1 \rhd_m Z_2) \end{array}}{(U \rhd_a \Sigma'[X_1 \rhd_a X_2]), Z_1 \vdash_m Z_2} \, D_1$$
  2. This case cannot occur.
  3. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_m Z_2); U \vdash_a \Sigma'[X_1 \rhd_a X_2]; \emptyset_a \end{array}}{Z_1 \vdash_m Z_2, (U \rhd_a \Sigma'[X_1 \rhd_a X_2])} \, D_2$$
  4. This case cannot occur.
* If $\Sigma[] = U \rhd_m \Sigma'[]$, then $\Sigma'[]$ can only be positive or a simple context. The case where $\Sigma'[]$ is a simple context is proved in Lemma 3.2.8. If $\Sigma'[]$ is positive, then we can apply the induction hypothesis as follows.
  1. This case cannot occur.
  2. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ U \vdash_m \Sigma'[X_1 \rhd_a X_2], (Z_1 \rhd_a Z_2) \end{array}}{(U \rhd_m \Sigma'[X_1 \rhd_a X_2]); Z_1 \vdash_a Z_2} \, D_2$$
  3. This case cannot occur.
  4. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_a Z_2), U \vdash_m \Sigma'[X_1 \rhd_a X_2], \emptyset_m \end{array}}{Z_1 \vdash_a Z_2; (U \rhd_m \Sigma'[X_1 \rhd_a X_2])} \, D_1$$

– For $Y_1 \rhd_m Y_2$, the proof is analogous to the above cases, since we do not operate on the inside sequent.

* If $\Sigma[] = \Sigma'[], W$ or $\Sigma[] = \Sigma'[]; W$, then $\Sigma'[]$ must be positive. So we can apply the induction hypothesis directly as in Lemma 3.2.8.
* If $\Sigma[] = \Sigma'[] \triangleright_a U$, then $\Sigma'[]$ must be positive, thus we can apply the induction hypothesis on $\Sigma'[]$.
  1. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ \emptyset_a; \Sigma'[Y_1 \triangleright_m Y_2] \vdash_a U; (Z_1 \triangleright_m Z_2) \end{array}}{(\Sigma'[Y_1 \triangleright_m Y_2] \triangleright_a U), Z_1 \vdash_m Z_2} D_1$$
  2. In this case the structure is not well-defined.
  3. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_m Z_2); \Sigma'[Y_1 \triangleright_m Y_2] \vdash_a U \end{array}}{Z_1 \vdash_m Z_2, (\Sigma'[Y_1 \triangleright_m Y_2] \triangleright_a U)} D_2$$
  4. This case cannot occur.
* If $\Sigma[] = \Sigma'[] \triangleright_m U$, then $\Sigma'[]$ must be positive, thus we can apply the induction hypothesis on $\Sigma'[]$.
  1. This case cannot occur.
  2. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ \emptyset_m, \Sigma'[Y_1 \triangleright_m Y_2] \vdash_m U, (Z_1 \triangleright_a Z_2) \end{array}}{(\Sigma'[Y_1 \triangleright_m Y_2] \triangleright_m U); Z_1 \vdash_a Z_2} D_2$$
  3. This case cannot occur.
  4. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_a Z_2), \Sigma'[Y_1 \triangleright_m Y_2] \vdash_m U \end{array}}{Z_1 \vdash_a Z_2; (\Sigma'[Y_1 \triangleright_m Y_2] \triangleright_m U)} D_1$$
* If $\Sigma[] = U \triangleright_a \Sigma'[]$, then $\Sigma'[]$ must be either positive or a simple context. The case where it is a simple context is handled in Lemma 3.2.8. For the case where it is positive, we can apply the induction hypothesis as follows.
  1. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ U \vdash_a \Sigma'[Y_1 \rhd_m Y_2]; (Z_1 \rhd_m Z_2) \end{array}}{(U \rhd_a \Sigma'[Y_1 \rhd_m Y_2]), Z_1 \vdash_m Z_2} D_1$$

2. This case cannot occur
3. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_m Z_2); U \vdash_a \Sigma'[Y_1 \rhd_m Y_2]; \emptyset_a \end{array}}{Z_1 \vdash_m Z_2, (U \rhd_a \Sigma'[Y_1 \rhd_m Y_2])} D_2$$

4. This case cannot occur.

∗ If $\Sigma[] = U \rhd_m \Sigma'[]$, then $\Sigma'[]$ can only be positive. It cannot be simple because then the structure will not be well-defined. So we can apply the induction hypothesis on $\Sigma'[]$.

1. This case cannot occur.
2. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ U \vdash_m \Sigma'[Y_1 \rhd_m Y_2], (Z_1 \rhd_a Z_2) \end{array}}{(U \rhd_m \Sigma'[Y_1 \rhd_m Y_2]); Z_1 \vdash_a Z_2} D_2$$

3. This case cannot occur.
4. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_a Z_2), U \vdash_m \Sigma'[Y_1 \rhd_m Y_2], \emptyset_m \end{array}}{Z_1 \vdash_a Z_2; (U \rhd_m \Sigma'[Y_1 \rhd_m Y_2])} D_1$$

The reverse direction works because we only use D_1/D_2 rules in the above proof, and they are both reversible. □

The last lemma shows the ability to display a structure in a negative context.

Lemma 3.2.10. *For any negative context $\Sigma^-[]$, any nested sequent structures Z_1 and Z_2, any nested sequent $X_1 \rhd_a X_2$, there exist structures W_1 and W_2 such that the sequent $W_1; X_1 \vdash_a X_2; W_2$ is derivable from one of the following sequents using only D_1/D_2 rules:*

1. $\Sigma[X_1 \rhd_a X_2], Z_1 \vdash_m Z_2$
2. $\Sigma[X_1 \rhd_a X_2]; Z_1 \vdash_a Z_2$
3. $Z_1 \vdash_m \Sigma[X_1 \rhd_a X_2], Z_2$

4. $Z_1 \vdash_a \Sigma[X_1 \rhd_a X_2]; Z_2$

and for any nested sequent $Y_1 \rhd_m Y_2$, the sequent $W_1, Y_1 \vdash_m Y_2, W_2$ is derivable from one of the following sequents using only D_1/D_2 rules:

1. $\Sigma[Y_1 \rhd_m Y_2], Z_1 \vdash_m Z_2$
2. $\Sigma[Y_1 \rhd_m Y_2]; Z_1 \vdash_a Z_2$
3. $Z_1 \vdash_m \Sigma[Y_1 \rhd_m Y_2], Z_2$
4. $Z_1 \vdash_a \Sigma[Y_1 \rhd_m Y_2]; Z_2$.

Proof. By induction on the size of context $\Sigma[]$.

- Base case:
  - For $X_1 \rhd_a X_2$, the context $\Sigma[]$ can only be of the form $V, [] \rhd_m U$. We analyse the four cases as follows.
    1. The structure in is case is not a nested sequent structure.
    2. We have the following inference:
    $$\dfrac{\dfrac{\varnothing_a; X_1 \vdash_a X_2; (V \rhd_m U, (Z_1 \rhd_a Z_2))}{V, (X_1 \rhd_a X_2) \vdash_m U, (Z_1 \rhd_a Z_2)}\, D_1}{(V, (X_1 \rhd_a X_2) \rhd_m U); Z_1 \vdash_a Z_2}\, D_2$$
    then $W_1 = \varnothing_a$, and $W_2 = (V \rhd_m U, (Z_1 \rhd_a Z_2))$.
    3. The structure cannot occur.
    4. We have the following inference:
    $$\dfrac{\dfrac{\varnothing_a; X_1 \vdash_a X_2; ((Z_1 \rhd_a Z_2), V \rhd_m U)}{(Z_1 \rhd_a Z_2), V, (X_1 \rhd_a X_2) \vdash_m U}\, D_1}{Z_1 \vdash_a (V, (X_1 \rhd_a X_2) \rhd_m U); Z_2}\, D_1$$
    then $W_1 = \varnothing_a$ and $W_2 = ((Z_1 \rhd_a Z_2), V \rhd_m U)$.
  - For $Y_1 \rhd_m Y_2$, the context $\Sigma[]$ can only be of the form $V; [] \rhd_a U$, thus the four cases are as follows.
    1. We have the following inference:
    $$\dfrac{\dfrac{\varnothing_m, Y_1 \vdash_m Y_2, (V \rhd_a U; (Z_1 \rhd_m Z_2))}{V; (Y_1 \rhd_m Y_2) \vdash_a U; (Z_1 \rhd_m Z_2)}\, D_2}{(V; (Y_1 \rhd_m Y_2) \rhd_a U), Z_1 \vdash_m Z_2}\, D_1$$
    then $W_1 = \varnothing_m$ and $W_2 = V \rhd_a U; (Z_1 \rhd_m Z_2)$.
    2. This case cannot occur.
    3. We have the following inference:
    $$\dfrac{\dfrac{\varnothing_m, Y_1 \vdash_m Y_2, ((Z_1 \rhd_m Z_2); V \rhd_a U)}{(Z_1 \rhd_m Z_2); V; (Y_1 \rhd_m Y_2) \vdash_a U}\, D_2}{Z_1 \vdash_m (V; (Y_1 \rhd_m Y_2) \rhd_a U), Z_2}\, D_2$$

then $W_1 = \emptyset_m$ and $W_2 = (Z_1 \triangleright_m Z_2); V \triangleright_a U$.

4. The structure in this case cannot occur.

- Inductive cases:
  - For $X_1 \triangleright_a X_2$:
    - If $\Sigma[] = \Sigma'[], W$ or $\Sigma[] = \Sigma'[]; W$, then $\Sigma'[]$ must be negative. So we can apply the induction hypothesis directly exactly the same as in Lemma 3.2.8.
    - If $\Sigma[] = \Sigma'[] \triangleright_a U$, then $\Sigma'[]$ must be negative. It cannot be simple because then the structure would not be well-defined. Thus we can apply the induction hypothesis on $\Sigma'[]$, the four cases are shown below.
      1. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ \emptyset_a; \Sigma'[X_1 \triangleright_a X_2] \vdash_a U; (Z_1 \triangleright_m Z_2) \end{array}}{(\Sigma'[X_1 \triangleright_a X_2] \triangleright_a U), Z_1 \vdash_m Z_2} D_1$$

      2. This case cannot occur.
      3. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_m Z_2); \Sigma'[X_1 \triangleright_a X_2] \vdash_a U \end{array}}{Z_1 \vdash_m Z_2, (\Sigma'[X_1 \triangleright_a X_2] \triangleright_a U)} D_2$$

      4. This case cannot occur.
    - If $\Sigma[] = \Sigma'[] \triangleright_m U$, then $\Sigma'[]$ can only be negative or a simple context. The latter is covered in Lemma 3.2.8. For the former case, we can apply the induction hypothesis on $\Sigma'[]$. The following are for the four cases:
      1. This case cannot occur.
      2. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ \emptyset_m, \Sigma'[X_1 \triangleright_a X_2] \vdash_m U, (Z_1 \triangleright_a Z_2) \end{array}}{(\Sigma'[X_1 \triangleright_a X_2] \triangleright_m U); Z_1 \vdash_a Z_2} D_2$$

      3. This case cannot occur.
      4. We have the following inference:

$$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_a Z_2), \Sigma'[X_1 \triangleright_a X_2] \vdash_m U \end{array}}{Z_1 \vdash_a Z_2; (\Sigma'[X_1 \triangleright_a X_2] \triangleright_m U)} \, D_1$$

* If $\Sigma[] = U \triangleright_a \Sigma'[]$, then $\Sigma'[]$ must be negative. So we apply the induction hypothesis on $\Sigma'[]$, giving the following four cases:
  1. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ U \vdash_a \Sigma'[X_1 \triangleright_a X_2]; (Z_1 \triangleright_m Z_2) \end{array}}{(U \triangleright_a \Sigma'[X_1 \triangleright_a X_2]), Z_1 \vdash_m Z_2} \, D_1$$
  2. This case cannot occur.
  3. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_m Z_2); U \vdash_a \Sigma'[X_1 \triangleright_a X_2]; \varnothing_a \end{array}}{Z_1 \vdash_m Z_2, (U \triangleright_a \Sigma'[X_1 \triangleright_a X_2])} \, D_2$$
  4. This case cannot occur.
* If $\Sigma[] = U \triangleright_m \Sigma'[]$, then $\Sigma'[]$ can only be negative, then we can apply the induction hypothesis as follows for each case:
  1. This case cannot occur.
  2. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ U \vdash_m \Sigma'[X_1 \triangleright_a X_2], (Z_1 \triangleright_a Z_2) \end{array}}{(U \triangleright_m \Sigma'[X_1 \triangleright_a X_2]); Z_1 \vdash_a Z_2} \, D_2$$
  3. This case cannot occur.
  4. We have the following inference:
  $$\dfrac{\begin{array}{c} W_1'; X_1 \vdash_a X_2; W_2' \\ \vdots \text{ IH} \\ (Z_1 \triangleright_a Z_2), U \vdash_m \Sigma'[X_1 \triangleright_a X_2], \varnothing_m \end{array}}{Z_1 \vdash_a Z_2; (U \triangleright_m \Sigma'[X_1 \triangleright_a X_2])} \, D_1$$

– For $Y_1 \triangleright_m Y_2$, the proof is analogous to the above cases, detailed below.
  * If $\Sigma[] = \Sigma'[], W$ or $\Sigma[] = \Sigma'[]; W$, then $\Sigma'[]$ must be positive. So we can apply the induction hypothesis directly exactly the same as in Lemma 3.2.8.

* If $\Sigma[] = \Sigma'[] \rhd_a U$, then $\Sigma'[]$ must be either negative or a simple context. The latter is handled in Lemma 3.2.8. For the former case we apply the induction hypothesis on $\Sigma'[]$ as follows for each subcase:
  1. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ \emptyset_a; \Sigma'[Y_1 \rhd_m Y_2] \vdash_a U; (Z_1 \rhd_m Z_2) \end{array}}{(\Sigma'[Y_1 \rhd_m Y_2] \rhd_a U), Z_1 \vdash_m Z_2} D_1$$
  2. This case cannot occur.
  3. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_m Z_2); \Sigma'[Y_1 \rhd_m Y_2] \vdash_a U \end{array}}{Z_1 \vdash_m Z_2, (\Sigma'[Y_1 \rhd_m Y_2] \rhd_a U)} D_2$$
  4. This case cannot occur.
* If $\Sigma[] = \Sigma'[] \rhd_m U$, then $\Sigma'[]$ must be negative, otherwise the structure would not be well-defined. Thus we can apply the induction hypothesis on $\Sigma'[]$.
  1. This case cannot occur.
  2. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ \emptyset_m, \Sigma'[Y_1 \rhd_m Y_2] \vdash_m U, (Z_1 \rhd_a Z_2) \end{array}}{(\Sigma'[Y_1 \rhd_m Y_2] \rhd_m U); Z_1 \vdash_a Z_2} D_2$$
  3. This case cannot occur.
  4. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_a Z_2), \Sigma'[Y_1 \rhd_m Y_2] \vdash_m U \end{array}}{Z_1 \vdash_a Z_2; (\Sigma'[Y_1 \rhd_m Y_2] \rhd_m U)} D_1$$
* If $\Sigma[] = U \rhd_a \Sigma'[]$, then $\Sigma'[]$ must be negative, we can apply the induction hypothesis as follows.
  1. We have the following inference:
$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ U \vdash_a \Sigma'[Y_1 \rhd_m Y_2]; (Z_1 \rhd_m Z_2) \end{array}}{(U \rhd_a \Sigma'[Y_1 \rhd_m Y_2]), Z_1 \vdash_m Z_2} D_1$$

2. This case cannot occur.
3. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_m Z_2); U \vdash_a \Sigma'[Y_1 \rhd_m Y_2]; \emptyset_a \end{array}}{Z_1 \vdash_m Z_2, (U \rhd_a \Sigma'[Y_1 \rhd_m Y_2])}\ D_2$$

4. This case cannot occur.

* If $\Sigma[] = U \rhd_m \Sigma'[]$, then $\Sigma'[]$ can only be negative. So we can apply the induction hypothesis on $\Sigma'[]$ when the structures are well-defined.
1. This case cannot occur.
2. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ U \vdash_m \Sigma'[Y_1 \rhd_m Y_2], (Z_1 \rhd_a Z_2) \end{array}}{(U \rhd_m \Sigma'[Y_1 \rhd_m Y_2]); Z_1 \vdash_a Z_2}\ D_2$$

3. This case cannot occur.
4. We have the following inference:

$$\dfrac{\begin{array}{c} W_1', Y_1 \vdash_m Y_2, W_2' \\ \vdots \text{ IH} \\ (Z_1 \rhd_a Z_2), U \vdash_m \Sigma'[Y_1 \rhd_m Y_2], \emptyset_m \end{array}}{Z_1 \vdash_a Z_2; (U \rhd_m \Sigma'[Y_1 \rhd_m Y_2])}\ D_1$$

The reverse direction works because we only use D_1/D_2 rules in the above proof, and they are both reversible. □

Finally, we can state the display property of SI_{CBI} as in immediate result of the above lemmas.

Theorem 3.2.11 (Display Property of SI_{CBI})**.** *Given any context $\Sigma[]$, any nested sequent $X_1 \rhd_a X_2$ ($Y_1 \rhd_m Y_2$ resp.), there exist structures W_1 and W_2 such that $W_1; X_1 \vdash_a X_2; W_2$ ($W_1, Y_1 \vdash_m Y_2, W_2$ resp.) is derivable from $\Sigma[X_1 \rhd_a X_2]$ ($\Sigma[Y_1 \rhd_m Y_2]$ resp.) by using only D_1/D_2 rules, and vice versa.*

Proof. Lemma 3.2.8 to 3.2.10 suffice to prove this. □

The next step, as in Postniece's work [83], is to develop a calculus with local reasoning and is able to transfer information between additive and multiplicative sequents.

3.3 DI_{CBI}: Towards Deep Inference for CBI

In this section we present a deep inference nested sequent calculus based on the shallow inference system SI_{CBI} in the previous section. Unfortunately, we have found numerous problems with our deep system that are hard to conquer. We discuss these problems and the incompleteness of our deep inference system.

3.3.1 Inference Rules in DI_{CBI}

Recall that in the shallow system, $X_1 \vdash_a X_2$ and $Y_1 \vdash_m Y_2$ denote nested shallow sequents, where X_1, X_2 are additive structures, and Y_1, Y_2 are multiplicative structures. All the inference rules can only be applied on the top level turnstile. In the deep system, nested deep sequents are defined as $X_1 \triangleright_a X_2$ and $Y_1 \triangleright_m Y_2$. The definitions of polarities, structures, sequents, formulae and contexts remain the same. A rule in the deep system is additive if its conclusion is additive, otherwise the rule is multiplicative[2]. Inference rules can be applied to any nested sequents in the deep system, thus we can zoom in on the structure to look at a sequent without explicitly displaying it. The rules for identity, constants, and additive logical connectives are shown in Figure 3.7; multiplicative logical rules are in Figure 3.8; propagation rules are shown in Figure 3.9. We refer to this inference system as DI_{CBI}.

The rules for additive connectives now have weakening and contraction built in. Thus the $\wedge R$ rule, for example, copies the context to both premises. The cases for multiplicative connectives are much more complicated. The $*R_1$ rule, for example, splits the structure in the context and only sends a part of the structure to the left premise, and sends the rest of the structure and the whole context to the right premise.

The propagation rules are even more complex compared to the deep nested sequent calculus for BiInt, because our nested sequents have interleaving flavours of additives and multiplicatives. Generally speaking, the additive structure and the multiplicative structure cannot communicate with each other, but there are a few special cases when one side of a nested sequent is the structural unit. For example, the rules T_aL and T_aR says that $A \triangleright_m \varnothing_m$ (resp. $\varnothing_m \triangleright_m A$) on the left (resp. right) hand side of a nested sequent is equivalent to A, thus we are safe to copy A to the left (resp. right) hand side of the upper level $\triangleright_a$ sequent, building in contraction. On the other hand, the rules T_mL and T_mR work similarly, but forbid contraction of the multiplicative structure and have weakening of additive structures built in. Moving formulae from an additive (resp. multiplicative) sequent to another additive (resp. multiplicative) sequent, usually with a different flavour sequent level in between, is less restricted. The rules P_a1 and P_a2 copy additive structures from one sequent to the other sequent. Correspondingly, we can cut-and-paste multiplicative structures from one sequent to

[2]This reading may be counter-intuitive when we later introduce interaction rules, but this concept is not important there.

Identity and logical constants:

$$\frac{}{\Sigma[X_1; A \rhd_a A; X_2]}\, id_a \qquad \frac{}{A \rhd_m A}\, id_m \qquad \frac{}{\Sigma[X_1; \bot \rhd_a X_2]}\, \bot L \qquad \frac{\Sigma[X_1 \rhd_a X_2]}{\Sigma[X_1 \rhd_a \bot; X_2]}\, \bot R$$

$$\frac{\Sigma[X_1 \rhd_a X_2]}{\Sigma[X_1; \top \rhd_a X_2]}\, \top L \qquad \frac{}{\Sigma[X_1 \rhd_a \top; X_2]}\, \top R \qquad \frac{}{\bot^* \rhd_m \varnothing_m}\, \bot^* L \qquad \frac{\Sigma[Y_1 \rhd_m Y_2]}{\Sigma[Y_1 \rhd_m \bot^*, Y_2]}\, \bot^* R$$

$$\frac{\Sigma[Y_1 \rhd_m Y_2]}{\Sigma[Y_1, \top^* \rhd_m Y_2]}\, \top^* L \qquad \frac{}{\varnothing_m \rhd_m \top^*}\, \top^* R$$

Additive Logical rules:

$$\frac{\Sigma[X_1; A \wedge B; A; B \rhd_a X_2]}{\Sigma[X_1; A \wedge B \rhd_a X_2]}\, \wedge L \qquad \frac{\Sigma[X_1 \rhd_a A \wedge B; A; X_2] \qquad \Sigma[X_1 \rhd_a A \wedge B; B; X_2]}{\Sigma[X_1 \rhd_a A \wedge B; X_2]}\, \wedge R$$

$$\frac{\Sigma[X_1 \rhd_a A \vee B; A; B; X_2]}{\Sigma[X_1 \rhd_a A \vee B; X_2]}\, \vee R \qquad \frac{\Sigma[X_1; A \vee B; A \rhd_a X_2] \qquad \Sigma[X_1; A \vee B; B \rhd_a X_2]}{\Sigma[X_1; A \vee B \rhd_a X_2]}\, \vee L$$

$$\frac{\Sigma[X_1; A \rhd_a B; A \to B; X_2]}{\Sigma[X_1 \rhd_a A \to B; X_2]}\, \to R \qquad \frac{\Sigma^-[X_1; A -\!\!< B; A \rhd_a B; X_2]}{\Sigma[X_1; A -\!\!< B \rhd_a X_2]}\, -\!\!< L$$

$$\frac{\Sigma[X_1; A \to B \rhd_a A; X_2] \qquad \Sigma[X_1; A \to B; B \rhd_a X_2]}{\Sigma[X_1; A \to B \rhd_a X_2]}\, \to L$$

$$\frac{\Sigma[X_1 \rhd_a A; A -\!\!< B; X_2] \qquad \Sigma[X_1; B \rhd_a A -\!\!< B; X_2]}{\Sigma[X_1 \rhd_a A -\!\!< B; X_2]}\, -\!\!< R$$

Figure 3.7: Axioms and additive logical rules in DI_{CBI}.

the other sequent, as shown in rules P_m1 and P_m2. The rules P_a3 and P_a4 copy formulae from an additive sequent to an upper level additive sequent. Note that we only move a formula from a sequent to the same side of another sequent. There are also many other propagation rules we can think of, but we shall stop here for the moment and show the soundness of the existing rules first. Notice that the rules P_a1, P_a2, P_m1 do not have context, which means these rules are not genuine deep inference rules. We will discuss this issue in the soundness proof.

3.3.2 Soundness of DI_{CBI}

We prove the soundness of DI_{CBI} by showing that its inference rules are valid with respect to the shallow system SI_{CBI}. In the theorem below we denote the DI_{CBI} sequent by Π and the corresponding SI_{CBI} sequent by Π'.

Theorem 3.3.1 (Soundness)**.** *For any nested sequent structures X_1, X_2, Y_1, and Y_2, if $\vdash_{DI_{CBI}} \Pi : X_1 \rhd_a X_2$ then $\vdash_{SI_{CBI}} \Pi' : X_1 \vdash_a X_2$; if $\vdash_{DI_{CBI}} \Pi : Y_1 \rhd_m Y_2$ then $\vdash_{SI_{CBI}} \Pi' : Y_1 \vdash_m Y_2$.*

Proof. We show that each deep inference rule in DI_{CBI} is derivable in SI_{CBI}. The context in a deep rule can only be simple, positive, or negative. If the context is either positive

Multiplicative Logical rules:

$$\frac{\Sigma[Y_1, A, B \rhd_m Y_2]}{\Sigma[Y_1, A * B \rhd_m Y_2]}\ {*L} \qquad \frac{Y_1 \rhd_m A, Y_3 \qquad \Sigma[Y_2 \rhd_m B, Y_4]}{\Sigma[Y_1, Y_2 \rhd_m A * B, Y_3, Y_4]}\ {*R_1}$$

$$\frac{Y_1, A \rhd_m Y_3 \qquad \Sigma[Y_2, B \rhd_m Y_4]}{\Sigma[Y_1, Y_2, A \vee^* B \rhd_m Y_3, Y_4]}\ {\vee^* L_1} \qquad \frac{\Sigma[Y_1 \rhd_m A, Y_3] \qquad Y_2 \rhd_m B, Y_4}{\Sigma[Y_1, Y_2 \rhd_m A * B, Y_3, Y_4]}\ {*R_2}$$

$$\frac{\Sigma[Y_1, A \rhd_m Y_3] \qquad Y_2, B \rhd_m Y_4}{\Sigma[Y_1, Y_2, A \vee^* B \rhd_m Y_3, Y_4]}\ {\vee^* L_2} \qquad \frac{\Sigma[Y_1 \rhd_m A, B, Y_2]}{\Sigma[Y_1 \rhd_m A \vee^* B, Y_2]}\ {\vee^* R}$$

$$\frac{Y_1 \rhd_m A, Y_3 \qquad \Sigma[Y_2, B \rhd_m Y_4]}{\Sigma[Y_1, Y_2, A -\!\!* B \rhd_m Y_3, Y_4]}\ {-\!\!* L_1} \qquad \frac{\Sigma[Y_1, A \rhd_m B, Y_2]}{\Sigma[Y_1 \rhd_m A -\!\!* B, Y_2]}\ {-\!\!* R}$$

$$\frac{\Sigma[Y_1 \rhd_m A, Y_3] \qquad Y_2, B \rhd_m Y_4}{\Sigma[Y_1, Y_2, A -\!\!* B \rhd_m Y_3, Y_4]}\ {-\!\!* L_2} \qquad \frac{Y_1 \rhd_m A, Y_3 \qquad \Sigma[Y_2, B \rhd_m Y_4]}{\Sigma[Y_1, Y_2 \rhd_m A -\!\!\times B, Y_3, Y_4]}\ {-\!\!\times R_1}$$

$$\frac{\Sigma[Y_1, A \rhd_m B, Y_2]}{\Sigma[Y_1, A -\!\!\times B \rhd_m Y_2]}\ {-\!\!\times L} \qquad \frac{\Sigma[Y_1 \rhd_m A, Y_3] \qquad Y_2, B \rhd_m Y_4}{\Sigma[Y_1, Y_2 \rhd_m A -\!\!\times B, Y_3, Y_4]}\ {-\!\!\times R_2}$$

Figure 3.8: Multiplicative logical rules in DI_{CBI}.

Propagation rules:

$$\frac{\Sigma[X_1; A; (A \rhd_m \varnothing_m) \rhd_a X_2]}{\Sigma[X_1; (A \rhd_m \varnothing_m) \rhd_a X_2]}\ {T_a L} \qquad \frac{\Sigma[X_1 \rhd_a A; (\varnothing_m \rhd_m A); X_2]}{\Sigma[X_1 \rhd_a (\varnothing_m \rhd_m A); X_2]}\ {T_a R}$$

$$\frac{\Sigma[Y_1, A \rhd_m Y_2]}{\Sigma[Y_1, (X_1; A \rhd_a X_2) \rhd_m Y_2]}\ {T_m L} \qquad \frac{\Sigma[Y_1 \rhd_m A, Y_2]}{\Sigma[Y_1 \rhd_m (X_1 \rhd_a A; X_2), Y_2]}\ {T_m R}$$

$$\frac{(X_1; A \rhd_a B; X_2) \rhd_m (X_3; A \rhd_a B; X_4)}{(X_1 \rhd_a X_2) \rhd_m (X_3; A \rhd_a B; X_4)}\ {P_{a1}} \qquad \frac{(X_1; A \rhd_a B; X_2) \rhd_m (X_3; A \rhd_a B; X_4)}{(X_1; A \rhd_a B; X_2) \rhd_m (X_3 \rhd_a X_4)}\ {P_{a2}}$$

$$\frac{\Sigma[X_1; A \rhd_a B; X_2; (\varnothing_m \rhd_m (X_3; A \rhd_a B; X_4))]}{\Sigma[X_1 \rhd_a X_2; (\varnothing_m \rhd_m (X_3; A \rhd_a B; X_4))]}\ {P_{a3}} \qquad \frac{\Sigma[X_1; A \rhd_a B; X_2; (\varnothing_m \rhd_m (X_3; A \rhd_a B; X_4))]}{\Sigma[X_1; A \rhd_a B; X_2; (\varnothing_m \rhd_m (X_3 \rhd_a X_4))]}\ {P_{a4}}$$

$$\frac{(Y_1, A \rhd_m B, Y_2) \rhd_a (Y_3 \rhd_m Y_4)}{(Y_1 \rhd_m Y_2) \rhd_a (Y_3, A \rhd_m B, Y_4)}\ {P_{m1}} \qquad \frac{\Sigma[Y_1, A \rhd_m B, Y_2, (\varnothing_a \rhd_a (Y_3 \rhd_m Y_4))]}{\Sigma[Y_1 \rhd_m Y_2, (\varnothing_a \rhd_a (Y_3, A \rhd_m B, Y_4))]}\ {P_{m2}}$$

Figure 3.9: Propagation rules in DI_{CBI}.

or negative, then we use the display property of the shallow system to display the sequent and prove the rule. The case where the context is just simple follows similarly.

Suppose an additive unary rule ρ and a multiplicative unary rule ρ' are respectively of the form

$$\frac{\Sigma[U \rhd_a Z]}{\Sigma[V \rhd_a W]}\ {\rho} \quad \text{and} \quad \frac{\Sigma[U \rhd_m Z]}{\Sigma[V \rhd_m W]}\ {\rho'}$$

By the display property, ρ and ρ' can be displayed to the following forms respec-

tively:

$$\frac{P; U \vdash_a Z; Q}{P; V \vdash_a W; Q} \quad \text{and} \quad \frac{P, U \vdash_m Z, Q}{P, V \vdash_m W, Q}$$

Since the context $\Sigma[]$ in the premise and the conclusion are exactly the same, by using the same sequence of D_1/D_2 rule applications, we will get the same structures P and Q in the premise and in the conclusion.

If the unary rule is not a zero-premise rule, then it is reduced to show that the following hold respectively:

$$\frac{U \vdash_a Z}{V \vdash_a W} \quad \text{and} \quad \frac{U \vdash_m Z}{V \vdash_m W}$$

on the condition that the reduced derivations do not involve display rules in SI_{CBI}. This is because the other unary rules in SI_{CBI} do not change or use the context at all, so the derivations without the context P, Q can be used as the derivations with the context, one simply needs to add the context in every sequent.

Now we show the cases for each rule.

For id_a, the displayed form can be proved by just applying the corresponding rule in SI_{CBI} and then using the display property of SI_{CBI}.

$$\dfrac{\dfrac{}{P; X_1; A \vdash_a A; X_2; Q}\, id_a}{\Sigma[X_1; A \vdash_a A; X_2]}\, \text{Thm. 3.2.11}$$

For $\bot L$,

$$\dfrac{\dfrac{\dfrac{}{\bot \vdash_a \varnothing_a}\, \bot L}{P; X_1; \bot \vdash_a X_2; Q}\, WL\&WR \text{ multiple times}}{\Sigma[X_1; \bot \vdash_a X_2]}\, \text{Thm. 3.2.11}$$

For $\top R$,

$$\dfrac{\dfrac{\dfrac{}{\varnothing_a \vdash_a \top}\, \top R}{P; X_1 \vdash_a \top; X_2; Q}\, WL\&WR \text{ multiple times}}{\Sigma[X_1 \vdash_a \top; X_2]}\, \text{Thm. 3.2.11}$$

The rules $\bot R$, $\top L$, id_m, $\bot^* L$, $\bot^* R$, $\top^* L$, $\top^* R$ can all be proved by direct applications of corresponding rules as they are either of exactly the same form or can be reduced to the same form as in the shallow system.

For $\wedge L$, the reduced form is proved as follows.

$$\dfrac{\dfrac{\dfrac{X_1; A \wedge B; A; B \vdash_a X_2}{X_1; A \wedge B; A; A \wedge B \vdash_a X_2}\, \wedge L}{X_1; A \wedge B; A \wedge B; A \wedge B \vdash_a X_2}\, \wedge L}{X_1; A \wedge B \vdash_a X_2}\, CL \text{ 2 times}$$

For $\wedge R$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1 \vdash_a A \wedge B; A; X_2]}{P; X_1 \vdash_a A \wedge B; A; X_2; Q}\,\text{Thm. 3.2.11} \quad \dfrac{\Sigma[X_1 \vdash_a A \wedge B; B; X_2]}{P; X_1 \vdash_a A \wedge B; B; X_2; Q}\,\text{Thm. 3.2.11}}{P; P; X_1; X_1 \vdash_a A \wedge B; A \wedge B; A \wedge B; X_2; X_2; Q; Q}\,\wedge R}{P; X_1 \vdash_a A \wedge B; X_2; Q}\,\textit{CL\&CR} \text{ multiple times}}{\Sigma[X_1 \vdash_a A \wedge B; X_2]}\,\text{Thm. 3.2.11}$$

For $\vee L$, the reduced form is proved as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1; A \vee B; A \vdash_a X_2]}{P; X_1; A \vee B; A \vdash_a X_2; Q}\,\text{Thm. 3.2.11} \quad \dfrac{\Sigma[X_1; A \vee B; B \vdash_a X_2]}{P; X_1; A \vee B; B \vdash_a X_2; Q}\,\text{Thm. 3.2.11}}{P; P; X_1; X_1; A \vee B; A \vee B; A \vee B \vdash_a X_2; X_2; Q; Q}\,\vee L}{P; X_1; A \vee B \vdash_a X_2; Q}\,\textit{CL\&CR} \text{ multiple times}}{\Sigma[X_1; A \vee B \vdash_a X_2]}\,\text{Thm. 3.2.11}$$

For $\vee R$, the reduced form is proved as follows.

$$\dfrac{\dfrac{\dfrac{X_1 \vdash_a A \vee B; A; B; X_2}{X_1 \vdash_a A \vee B; A \vee B; B; X_2}\,\vee R}{X_1 \vdash_a A \vee B; A \vee B; A \vee B; X_2}\,\vee R}{X_1 \vdash_a A \vee B; X_2}\,\textit{CR} \text{ 2 times}$$

For $\rightarrow L$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1; A \rightarrow B \vdash_a A; X_2]}{P; X_1; A \rightarrow B \vdash_a A; X_2; Q}\,\text{Thm. 3.2.11} \quad \dfrac{\Sigma[X_1; A \rightarrow B; B \vdash_a X_2]}{P; X_1; A \rightarrow B; B \vdash_a X_2; Q}\,\text{Thm. 3.2.11}}{P; P; X_1; X_1; A \rightarrow B; A \rightarrow B; A \rightarrow B \vdash_a X_2; X_2; Q; Q}\,\rightarrow L}{P; X_1; A \rightarrow B \vdash_a X_2; Q}\,\textit{CL\&CR} \text{ multiple times}}{\Sigma[X_1; A \rightarrow B \vdash_a X_2]}\,\text{Thm. 3.2.11}$$

For $\rightarrow R$, the reduced form is proved as follows.

$$\dfrac{\dfrac{X_1; A \vdash_a B; A \rightarrow B; X_2}{X_1 \vdash_a A \rightarrow B; A \rightarrow B; X_2}\,\rightarrow R}{X_1 \vdash_a A \rightarrow B; X_2}\,\textit{CR}$$

For $-\!\!\prec L$, the reduced form is proved as follows.

$$\dfrac{\dfrac{X_1; A -\!\!\prec B; A \vdash_a B; X_2}{X_1; A -\!\!\prec B; A -\!\!\prec B \vdash_a X_2}\,-\!\!\prec L}{X_1; A -\!\!\prec B \vdash_a X_2}\,\textit{CL}$$

For $-\!\!\prec R$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1 \vdash_a A; A -\!\!\prec B; X_2]}{P; X_1 \vdash_a A; A -\!\!\prec B; X_2; Q}\,\text{Thm. 3.2.11} \quad \dfrac{\Sigma[X_1; B \vdash_a A -\!\!\prec B; X_2]}{P; X_1; B \vdash_a A -\!\!\prec B; X_2; Q}\,\text{Thm. 3.2.11}}{P; P; X_1; X_1 \vdash_a A -\!\!\prec B; A -\!\!\prec B; A -\!\!\prec B; X_2; X_2; Q; Q}\,-\!\!\prec R}{P; X_1 \vdash_a A -\!\!\prec B; X_2; Q}\,\textit{CL\&CR} \text{ multiple times}}{\Sigma[X_1 \vdash_a A -\!\!\prec B; X_2]}\,\text{Thm. 3.2.11}$$

For $*L$, the displayed form can be reduced to the same form as in the shallow system.

$$\frac{Y_1, A, B \vdash_m Y_2}{Y_1, A * B \vdash_m Y_2} *L$$

For $*R_1$,

$$\dfrac{\dfrac{Y_1 \vdash_m A, Y_3 \qquad \dfrac{\Sigma[Y_2 \vdash_m B, Y_4]}{P, Y_2 \vdash_m B, Y_4, Q}\,\text{Thm. 3.2.11}}{P, Y_1, Y_2 \vdash_m A * B, Y_3, Y_4, Q} *R_1}{\Sigma[Y_1, Y_2 \vdash_m A * B, Y_3, Y_4]}\,\text{Thm. 3.2.11}$$

For $\vee^*\ L_1$,

$$\dfrac{\dfrac{Y_1, A \vdash_m Y_3 \qquad \dfrac{\Sigma[Y_2, B \vdash_m Y_4]}{P, Y_2, B \vdash_m Y_4, Q}\,\text{Thm. 3.2.11}}{P, Y_1, Y_2, A \vee^* B \vdash_m Y_3, Y_4, Q} \vee^* L_1}{\Sigma[Y_1, Y_2, A \vee^* B \vdash_m Y_3, Y_4]}\,\text{Thm. 3.2.11}$$

The cases for $*R_2$ and $\vee^*\ L_2$ are similar to the above.

For $\vee^*\ R$, the reduced form can be proved by directly applying the corresponding rule in SI_{CBI}.

$$\frac{Y_1 \vdash_m A, B, Y_2}{Y_1 \vdash_m A \vee^* B, Y_2} \vee^* R$$

For $-\!\!*\ L_1$,

$$\dfrac{\dfrac{Y_1 \vdash_m A, Y_3 \qquad \dfrac{\Sigma[Y_2, B \vdash_m Y_4]}{P, Y_2, B \vdash_m Y_4, Q}\,\text{Thm. 3.2.11}}{P, Y_1, Y_2, A -\!\!* B \vdash_m Y_3, Y_4, Q} -\!\!* L}{\Sigma[Y_1, Y_2, A -\!\!* B \vdash_m Y_3, Y_4]}\,\text{Thm. 3.2.11}$$

For $-\!\!*\ L_2$, the proof is analogous.

$$\dfrac{\dfrac{\dfrac{\Sigma[Y_1 \vdash_m A, Y_3]}{P, Y_1 \vdash_m A, Y_3, Q}\,\text{Thm. 3.2.11} \qquad Y_2, B \vdash_m Y_4}{P, Y_1, Y_2, A -\!\!* B \vdash_m Y_3, Y_4, Q} -\!\!* L}{\Sigma[Y_1, Y_2, A -\!\!* B \vdash_m Y_3, Y_4]}\,\text{Thm. 3.2.11}$$

For $-\!\!*\ R$, the reduced form can be proved by directly applying the corresponding rule in SI_{CBI}.

$$\frac{Y_1, A \vdash_m B, Y_2}{Y_1 \vdash_m A -\!\!* B, Y_2} \vee^* R$$

For $-\!\!\times\ L$, the reduced form can be proved by directly applying the corresponding rule in SI_{CBI}.

$$\frac{Y_1, A \vdash_m B, Y_2}{Y_1, A \multimap B \vdash_m Y_2} \multimap L$$

For $\multimap R_1$,

$$\dfrac{\dfrac{Y_1 \vdash_m A, Y_3 \qquad \overset{\Sigma[Y_2, B \vdash_m Y_4]}{\vdots\ \text{Thm. 3.2.11}} \quad P, Y_2, B \vdash_m Y_4, Q}{P, Y_1, Y_2 \vdash_m A \multimap B, Y_3, Y_4, Q} \multimap R}{\underset{\Sigma[Y_1, Y_2 \vdash_m A \multimap B, Y_3, Y_4]}{\vdots\ \text{Thm. 3.2.11}}}$$

For $\multimap R_2$,

$$\dfrac{\dfrac{\overset{\Sigma[Y_1 \vdash_m A, Y_3]}{\vdots\ \text{Thm. 3.2.11}} \quad P, Y_1 \vdash_m A, Y_3, Q \qquad Y_2, B \vdash_m Y_4}{P, Y_1, Y_2 \vdash_m A \multimap B, Y_3, Y_4, Q} \multimap R}{\underset{\Sigma[Y_1, Y_2 \vdash_m A \multimap B, Y_3, Y_4]}{\vdots\ \text{Thm. 3.2.11}}}$$

For T_aL, the reduced form can be proved as follows.

$$\dfrac{\dfrac{X_1; A; (A \triangleright_m \varnothing_m) \vdash_a X_2}{X_1; (A \triangleright_m \varnothing_m); (A \triangleright_m \varnothing_m) \vdash_a X_2} T_aL}{X_1; (A \triangleright_m \varnothing_m) \vdash_a X_2} CL$$

For T_aR, the proof is analogous.

$$\dfrac{\dfrac{X_1 \vdash_a A; (\varnothing_m \triangleright_m A); X_2}{X_1 \vdash_a (\varnothing_m \triangleright_m A); (\varnothing_m \triangleright_m A); X_2} T_aR}{X_1 \vdash_a (\varnothing_m \triangleright_m A); X_2} CR$$

For T_mL and T_mR the reduced form is exactly the same as in the shallow system, thus a direct application of corresponding rules is enough.

The other propagation rules cannot be proved by simply showing a derivation for the reduced form, because their derivations in SI_{CBI} may involve display rules that manipulate the surrounding context. For P_a1, the following derivation is fine, but a derivation with contexts in the conclusion will be stuck at the topmost $\triangleright_1$ application.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(X_1; A \triangleright_a B; X_2) \vdash_m (X_3; A \triangleright_a B; X_4)}{X_1; A \vdash_a X_2; B; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))} D_1}{(X_1 \triangleright_a X_2) \vdash_m (A \triangleright_a B; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4)))} \triangleright_1}{((X_1 \triangleright_a X_2) \triangleright_m \varnothing_m); A \vdash_a B; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))} D_2}{(((X_1 \triangleright_a X_2) \triangleright_m \varnothing_m); A \triangleright_a B) \vdash_m (X_3; A \triangleright_a B; X_4)} D_1}{((X_1 \triangleright_a X_2) \triangleright_m \varnothing_m); X_3; A; A \vdash_a B; B; X_4} \triangleright_1}{((X_1 \triangleright_a X_2) \triangleright_m \varnothing_m); X_3; A \vdash_a B; X_4} CL\&CR}{(X_1 \triangleright_a X_2) \vdash_m (X_3; A \triangleright_a B; X_4)} D_2$$

For P_a2 the proof is analogous to P_a1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(X_1; A \triangleright_a B; X_2) \vdash_m (X_3; A \triangleright_a B; X_4)}{((X_1; A \triangleright_a B; X_2) \triangleright_m \varnothing_m); A; X_3 \vdash_a B; X_4}\, D_2}{(((X_1; A \triangleright_a B; X_2) \triangleright_m \varnothing_m); A \triangleright_a B) \vdash_m (X_3 \triangleright_a X_4)}\, \triangleright_1}{((X_1; A \triangleright_a B; X_2) \triangleright_m \varnothing_m); A \vdash_a B; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4))}\, D_1}{(X_1; A \triangleright_a B; X_2) \vdash_m (A \triangleright_a B; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4)))}\, D_2}{X_1; A; A \vdash_a B; B; X_2; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4))}\, \triangleright_1}{X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4))}\, CL\&CR}{(X_1; A \triangleright_a B; X_2) \vdash_m (X_3 \triangleright_a X_4)}\, D_1$$

For P_a3, the context P, Q in the conclusion actually does not matter.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))]}{P; X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4)); Q}\, \text{Thm. 3.2.11}}{(P; X_1 \triangleright_a X_2; Q) \vdash_m (A \triangleright_a B; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4)))}\, \triangleright_1}{((P; X_1 \triangleright_a X_2; Q) \triangleright_m \varnothing_m); A \vdash_a B; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))}\, D_2}{(((P; X_1 \triangleright_a X_2; Q) \triangleright_m \varnothing_m); A \triangleright_a B) \vdash_m (X_3; A \triangleright_a B; X_4)}\, D_1}{((P; X_1 \triangleright_a X_2; Q) \triangleright_m \varnothing_m); X_3; A; A \vdash_a B; B; X_4}\, \triangleright_1}{((P; X_1 \triangleright_a X_2; Q) \triangleright_m \varnothing_m); X_3; A \vdash_a B; X_4}\, CL\&CR}{(P; X_1 \triangleright_a X_2; Q) \vdash_m (X_3; A \triangleright_a B; X_4)}\, D_2}{P; X_1 \vdash_a X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4)); Q}\, D_1}{\Sigma[X_1 \vdash_a X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))]}\, \text{Thm. 3.2.11}$$

For P_a4, the proof is analogous.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4))]}{P; X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3; A \triangleright_a B; X_4)); Q}\, \text{Thm. 3.2.11}}{(P; X_1; A \triangleright_a B; X_2; Q) \vdash_m (X_3; A \triangleright_a B; X_4)}\, D_1}{((P; X_1; A \triangleright_a B; X_2; Q) \triangleright_m \varnothing_m); X_3; A \vdash_a B; X_4}\, D_2}{(((P; X_1; A \triangleright_a B; X_2; Q) \triangleright_m \varnothing_m); A \triangleright_a B) \vdash_m (X_3 \triangleright_a X_4)}\, \triangleright_1}{((P; X_1; A \triangleright_a B; X_2; Q) \triangleright_m \varnothing_m); A \vdash_a B; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4))}\, D_1}{(P; X_1; A \triangleright_a B; X_2; Q) \vdash_m (A \triangleright_a B; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4)))}\, D_2}{P; X_1; A; A \vdash_a B; B; X_2; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4)); Q}\, \triangleright_1}{P; X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4)); Q}\, CL\&CR}{\Sigma[X_1; A \vdash_a B; X_2; (\varnothing_m \triangleright_m (X_3 \triangleright_a X_4))]}\, \text{Thm. 3.2.11}$$

The rules P_m1 and P_m2 can be proved similarly without using contraction. If we had allowed context in the conclusion of P_m1, we would have to use weakening to discard the context, in that case we will need to present two separation rules because weakening is not reversible. For P_m2, having context is not an issue. For P_m1, the proof runs as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(Y_1, A \triangleright_m B, Y_2) \vdash_a (Y_3 \triangleright_m Y_4)}{Y_1, A \vdash_m B, Y_2, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4))}\, D_2}{(Y_1 \triangleright_m Y_2) \vdash_a (A \triangleright_m B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4)))}\, \triangleright_2}{((Y_1 \triangleright_m Y_2) \triangleright_a \varnothing_a), A \vdash_m B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4))}\, D_1}{(((Y_1 \triangleright_m Y_2) \triangleright_a \varnothing_a), A \triangleright_m B) \vdash_a (Y_3 \triangleright_m Y_4)}\, D_2}{((Y_1 \triangleright_m Y_2) \triangleright_a \varnothing_a), Y_3, A \vdash_m B, Y_4}\, \triangleright_2}{(Y_1 \triangleright_m Y_2) \vdash_a (Y_3, A \vdash_m B, Y_4)}\, D_1$$

The reverse direction is proved by the revertibility of the rules used in the above proof.

For P_m2, the proof runs as below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma[Y_1, A \vdash_m Y_2, B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4))]}{P, Y_1, A \vdash_m Y_2, B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4)), Q}\, \text{Thm. 3.2.11}}{(P, Y_1 \triangleright_m Y_2, Q) \vdash_a (A \triangleright_m B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4)))}\, \triangleright_2}{((P, Y_1 \triangleright_m Y_2, Q) \triangleright_a \varnothing_a), A \vdash_m B, (\varnothing_a \triangleright_a (Y_3 \triangleright_m Y_4))}\, D_1}{(((P, Y_1 \triangleright_m Y_2, Q) \triangleright_a \varnothing_a), A \triangleright_m B) \vdash_a (Y_3 \triangleright_m Y_4)}\, D_2}{((P, Y_1 \triangleright_m Y_2, Q) \triangleright_a \varnothing_a), Y_3, A \vdash_m B, Y_4}\, \triangleright_2}{(P, Y_1 \triangleright_m Y_2, Q) \vdash_a (Y_3, A \vdash_m B, Y_4)}\, D_1}{P, Y_1 \vdash_m Y_2, (\varnothing_a \triangleright_a (Y_3, A \vdash_m B, Y_4)), Q}\, D_2}{\Sigma[Y_1 \vdash_m Y_2, (\varnothing_a \triangleright_a (Y_3, A \vdash_m B, Y_4))]}\, \text{Thm. 3.2.11}$$

The reverse direction is proved by the revertibility of the rules in the above proof.

We conclude that DI_{CBI} is sound since every rule in DI_{CBI} is derivable from SI_{CBI}, which is already proven sound. □

3.3.3 Incompleteness of DI_{CBI}

A key to prove the completeness of the deep nested sequent system is to show that the display rules in the shallow nested sequent system are admissible in the deep system, therefore we only need the propagation rules to transfer formulae and do not have to display any structure. However, to show this, we need to prove that for any rule in the deep system, if the premise can be rearranged to another form using a display rule in the shallow system, then the rearranged conclusion by that display rule can be derived from the rearranged premise using existing rules in the deep system.

Another important issue is the "distribution lemma" as in *DBiInt*: provability of $\Sigma[(X \triangleright Y), (Z \triangleright W)]$ implies provability of $\Sigma[X, Z \triangleright Y, W]$. This lemma is essential when proving the admissibility of contraction in *DBiInt*, but in our setting, it is not obvious that such a property holds. By the construction of our nested sequents, the distribution lemma in our setting can be described as: if $\Sigma[(Y_1 \triangleright_m Y_2); (Y_3 \triangleright_m Y_4)]$ is provable then $\Sigma[Y_1, Y_3 \triangleright_m Y_2, Y_4]$ is provable; another case is: if $\Sigma[(X_1 \triangleright_a X_2), (X_3 \triangleright_a X_4)]$ is provable then $\Sigma[X_1; X_3 \triangleright_a X_2; X_4]$ is provable. Neither of these two cases are straightforward.

When proving the above properties we find that our propagation rules in DI_{CBI} are not sufficient. For example, propagation rules in Figure 3.10 are just some of the rules we could think of. From these rules, we can go further and consider what could happen if a deeper nested structure needs to be displayed in the shallow inference system, which may require a formula to be transferred from/to a deeper nested sequent in the deep system. Some of these cases are shown in Figure 3.11. But the problem does not stop there, we have to add new rules one after another to show the cases when an even deeper nested structure gets displayed, and there seems to be no end in doing so. That is, there are always new rules that cannot be derived by existing rules. Some of the rules we came up with do not support deep inference. They are only sound if no context is around, this is another reason why we do not consider our deep inference system as being successful.

Propagation rules:

$$\frac{X_1 \rhd_a A; X_2}{(X_1 \rhd_a X_2) \rhd_m A}\ T_aLt \qquad \frac{X_1; A \rhd_a X_2}{A \rhd_m (X_1 \rhd_a X_2)}\ T_aRt$$

$$\frac{Y_1 \rhd_m A, Y_2}{\Sigma[X_1; (Y_1 \rhd_m Y_2) \rhd_a A; X_2]}\ T_mLt \qquad \frac{Y_1, A \rhd_m Y_2}{\Sigma[X_1; A \rhd_a (Y_1 \rhd_m Y_2); X_2]}\ T_mRt$$

$$\frac{\Sigma[((X_1; A \rhd_a X_2) \rhd_m \varnothing_m); X_3; A \rhd_a X_4]}{\Sigma[((X_1 \rhd_a X_2) \rhd_m \varnothing_m); X_3; A \rhd_a X_4]}\ P_{a9} \qquad \frac{\Sigma[((X_1 \rhd_a A; X_2) \rhd_m \varnothing_m); X_3 \rhd_a A; X_4]}{\Sigma[((X_1 \rhd_a X_2) \rhd_m \varnothing_m); X_3 \rhd_a A; X_4]}\ P_{a10}$$

$$\frac{\Sigma[((X_1; A \rhd_a X_2) \rhd_m \varnothing_m); X_3; A \rhd_a X_4]}{\Sigma[((X_1; A \rhd_a X_2) \rhd_m \varnothing_m); X_3 \rhd_a X_4]}\ P_{a11} \qquad \frac{\Sigma[((X_1 \rhd_a A; X_2) \rhd_m \varnothing_m); X_3 \rhd_a A; X_4]}{\Sigma[((X_1 \rhd_a A; X_2) \rhd_m \varnothing_m); X_3 \rhd_a X_4]}\ P_{a12}$$

$$\frac{\Sigma[((Y_1, A \rhd_m Y_2) \rhd_a \varnothing_a), Y_3 \rhd_m Y_4]}{\Sigma[(X_1; (Y_1 \rhd_m Y_2) \rhd_a X_2), Y_3, A \rhd_m Y_4]}\ P_{m9} \qquad \frac{\Sigma[((Y_1 \rhd_m A, Y_2) \rhd_a \varnothing_a), Y_3 \rhd_m Y_4]}{\Sigma[(X_1; (Y_1 \rhd_m Y_2) \rhd_a X_2), Y_3 \rhd_m A, Y_4]}\ P_{m10}$$

$$\frac{\Sigma[((Y_1 \rhd_m Y_2) \rhd_a \varnothing_a), Y_3, A \rhd_m Y_4]}{\Sigma[(X_1; (Y_1, A \rhd_m Y_2) \rhd_a X_2), Y_3 \rhd_m Y_4]}\ P_{m11} \qquad \frac{\Sigma[((Y_1 \rhd_m Y_2) \rhd_a \varnothing_a), Y_3 \rhd_m A, Y_4]}{\Sigma[(X_1; (Y_1 \rhd_m A, Y_2) \rhd_a X_2), Y_3 \rhd_m Y_4]}\ P_{m12}$$

Figure 3.10: Deep inference nested sequents calculus DI_{CBI} for CBI part 3

Even worse, we have to develop other rules that handle the cases where an additive formula falls in a multiplicative sequent and vice versa. We call this type of rule *interaction rules*. Some examples of interaction rules are given in Figure 3.12. Typically, we need to convert such formulae into one or more structures to maintain the interleaving nature of the nested sequents. This type of rule read like logical rules, but often result in structures instead of formulae in the premise(s), although the structures are just proxies to the corresponding formulae.

Similar to the case of propagation rules, it seems that we can always think of new interaction rules that apply on a formula in the conclusion, but split the context in different ways, e.g., distribute a subformula to a deeper level nested sequent. This situation usually happens on binary multiplicative rules that require splitting context,

Additional Propagation rules:

$$\frac{\Sigma[X_1; A \rhd_a ((\varnothing_a \rhd_a (A \rhd_m \varnothing_m)) \rhd_m \varnothing_m); X_2]}{\Sigma[X_1 \rhd_a ((\varnothing_a \rhd_a (A \rhd_m \varnothing_m)) \rhd_m \varnothing_m); X_2]}\ T_aLn \qquad \frac{\Sigma[X_1; (\varnothing_m \rhd_m ((\varnothing_m \rhd_m A) \rhd_a \varnothing_a)) \rhd_a A; X_2]}{\Sigma[X_1; (\varnothing_m \rhd_m ((\varnothing_m \rhd_m A) \rhd_a \varnothing_a)) \rhd_a X_2]}\ T_aRn$$

$$\frac{\Sigma[Y_1, A \rhd_m Y_2]}{\Sigma[Y_1 \rhd_m ((\varnothing_m \rhd_m (A \rhd_a \varnothing_a)) \rhd_a \varnothing_a), Y_2]}\ T_mLn \qquad \frac{\Sigma[Y_1 \rhd_m A, Y_2]}{\Sigma[Y_1, (\varnothing_a \rhd_a ((\varnothing_a \rhd_a A) \rhd_m \varnothing_m)) \rhd_m Y_2]}\ T_mRn$$

$$\frac{(X_1; A \rhd_a X_2), (\varnothing_a \rhd_a (A \rhd_m \varnothing_m)) \rhd_m \varnothing_m}{(X_1 \rhd_a X_2), (\varnothing_a \rhd_a (A \rhd_m \varnothing_m)) \rhd_m \varnothing_m}\ T_aLnd \qquad \frac{\varnothing_m \rhd_m ((\varnothing_m \rhd_m A) \rhd_a \varnothing_a), (X_1 \rhd_a A; X_2)}{\varnothing_m \rhd_m ((\varnothing_m \rhd_m A) \rhd_a \varnothing_a), (X_1 \rhd_a X_2)}\ T_aRnd$$

$$\frac{Y_1, A \rhd_m Y_2}{\Sigma[X_1; (Y_1 \rhd_m Y_2); (\varnothing_m \rhd_m (A \rhd_a \varnothing_a)) \rhd_a X_2]}\ T_mLnd \qquad \frac{Y_1 \rhd_m A, Y_2}{\Sigma[X_1 \rhd_a ((\varnothing_a \rhd_a A) \rhd_m \varnothing_m); (Y_1 \rhd_m Y_2); X_2]}\ T_mRnd$$

Figure 3.11: Deep inference nested sequents calculus DI_{CBI} for CBI part 4

as shown in Figure 3.13.

As a consequence, our attempt to obtain a nested sequent calculus to solve theorem proving for BI logics is not successful, the technical difficulty comes in when we analyse how additive sequents interact with multiplicative sequents, which is related to the over complicated structure of sequent we are considering. But our effort in this chapter is not in vain. Learning about what we cannot do is just as important as discovering what we can do. Instead of giving up, we wonder if there is a simple and natural way to express the interaction between the additive part and the multiplicative part of BI logics and to reason about the formulae in these logics. This question leads us to a new direction to attack our problem, as will be detailed in the next chapter.

Interaction rules:

$$\frac{}{\Sigma[Y_1, \bot \rhd_m Y_2]}\ \bot Li \qquad \frac{\Sigma[Y_1 \rhd_m (\varnothing_a \rhd_a \bot), Y_2]}{\Sigma[Y_1 \rhd_m \bot, Y_2]}\ \bot Ri$$

$$\frac{\Sigma[X_1; (\bot^* \rhd_m \varnothing_m) \rhd_a X_2]}{\Sigma[X_1; \bot^* \rhd_a X_2]}\ \bot^* Li \qquad \frac{\Sigma[X_1 \rhd_a (\varnothing_m \rhd_m \bot^*); X_2]}{\Sigma[X_1 \rhd_a \bot^*; X_2]}\ \bot^* Ri$$

$$\frac{\Sigma[Y_1, (\top \rhd_a \varnothing_a) \rhd_m Y_2]}{\Sigma[Y_1, \top \rhd_m Y_2]}\ \top Li \qquad \frac{}{\Sigma[Y_1 \rhd_m \top, Y_2]}\ \top Ri$$

$$\frac{\Sigma[X_1; (\top^* \rhd_m \varnothing_m) \rhd_a X_2]}{\Sigma[X_1; \top^* \rhd_a X_2]}\ \top^* Li \qquad \frac{\Sigma[X_1 \rhd_a (\varnothing_m \rhd_m \top^*); X_2]}{\Sigma[X_1 \rhd_a \top^*; X_2]}\ \top^* Ri$$

$$\frac{\Sigma[Y_1, (A; B \rhd_a \varnothing_a) \rhd_m Y_2]}{\Sigma[Y_1, A \wedge B \rhd_m Y_2]}\ \wedge Li \qquad \frac{\Sigma[Y_1 \rhd_m A, Y_2] \qquad \Sigma[Y_1 \rhd_m B, Y_2]}{\Sigma[Y_1 \rhd_m A \wedge B, Y_2]}\ \wedge Ri$$

$$\frac{\Sigma[Y_1, A \rhd_m Y_2] \qquad \Sigma[Y_1, B \rhd_m Y_2]}{\Sigma[Y_1, A \vee B \rhd_m Y_2]}\ \vee Li \qquad \frac{\Sigma[Y_1 \rhd_m (\varnothing_a \rhd_a A; B), Y_2]}{\Sigma[Y_1 \rhd_m A \vee B, Y_2]}\ \vee Ri$$

$$\frac{\Sigma[Y_1, (\varnothing_a \rhd_a A) \rhd_m Y_2] \qquad \Sigma[Y_1, B \rhd_m Y_2]}{\Sigma[Y_1, A \to B \rhd_m Y_2]}\ \to Li \qquad \frac{\Sigma[Y_1 \rhd_m (A \rhd_a B), Y_2]}{\Sigma[Y_1 \rhd_m A \to B, Y_2]}\ \to Ri$$

$$\frac{\Sigma[Y_1, (A \rhd_a B) \rhd_m Y_2]}{\Sigma[Y_1, A -\!\!< B \rhd_m Y_2]}\ -\!\!< Li \qquad \frac{\Sigma[Y_1, (\varnothing_a \rhd_a A) \rhd_m Y_2] \qquad \Sigma[Y_1, B \rhd_m Y_2]}{\Sigma[Y_1 \rhd_m A -\!\!< B, Y_2]}\ -\!\!< Ri$$

$$\frac{\Sigma[X_1; (A, B \rhd_m \varnothing_m) \rhd_a X_2]}{\Sigma[X_1; A * B \rhd_a X_2]}\ *Li \qquad \frac{\varnothing_m \rhd_m A \qquad \Sigma[X_1 \rhd_a B; X_2]}{\Sigma[X_1 \rhd_a A * B; X_2]}\ *Ri1$$

$$\frac{A \rhd_m \varnothing_m \qquad \Sigma[X_1; B \rhd_a X_2]}{\Sigma[X_1; A \vee^* B \rhd_a X_2]}\ \vee^* Li1 \qquad \frac{\Sigma[X_1 \rhd_a (\varnothing_m \rhd_m A, B); X_2]}{\Sigma[X_1 \rhd_a A \vee^* B; X_2]}\ \vee^* Ri$$

$$\frac{\Sigma[X_1; (\varnothing_m \rhd_m A) \rhd_a X_2] \qquad B \rhd_m \varnothing_m}{\Sigma[X_1; A -\!\!* B \rhd_a X_2]}\ -\!\!* Li1 \qquad \frac{\Sigma[X_1 \rhd_a (A \rhd_m B); X_2]}{\Sigma[X_1 \rhd_a A -\!\!* B; X_2]}\ -\!\!* Ri$$

$$\frac{\varnothing_m \rhd_m A \qquad \Sigma[X_1; B \rhd_a X_2]}{\Sigma[X_1; A -\!\!* B \rhd_a X_2]}\ -\!\!* Li2 \qquad \frac{\Sigma[X_1; (\varnothing_m \rhd_m A) \rhd_a X_2] \qquad B \rhd_m \varnothing_m}{\Sigma[X_1 \rhd_a A -\!\!\times B; X_2]}\ -\!\!\times Ri1$$

$$\frac{\Sigma[X_1; (A \rhd_m B) \rhd_a X_2]}{\Sigma[X_1; A -\!\!\times B \rhd_a X_2]}\ -\!\!\times Li \qquad \frac{\varnothing_m \rhd_m A \qquad \Sigma[X_1; B \rhd_a X_2]}{\Sigma[X_1 \rhd_a A -\!\!\times B; X_2]}\ -\!\!\times Ri2$$

Figure 3.12: Deep inference nested sequents calculus DI_{CBI} for CBI part 5

Additional Interaction rules:

$$\frac{\Sigma[X_1; (Y_1, Y_2 \triangleright_m \varnothing_m); (((Y_1 \triangleright_m \varnothing_m) \triangleright_a A), Y_2 \triangleright_m \varnothing_m) \triangleright_a A * B; X_2] \qquad Y_2 \triangleright_m B}{\Sigma[X_1; (Y_1, Y_2 \triangleright_m \varnothing_m) \triangleright_a A * B; X_2]} *Ri2$$

$$\frac{\Sigma[X_1; A \vee^* B \triangleright_a (\varnothing_m \triangleright_m Y_2, (A \triangleright_a (\varnothing_m \triangleright_m Y_1))); (\varnothing_m \triangleright_m Y_1, Y_2); X_2] \qquad B \triangleright_m Y_2}{\Sigma[X_1; A \vee^* B \triangleright_a (\varnothing_m \triangleright_m Y_1, Y_2); X_2]} \vee^* Li2$$

$$\frac{\Sigma[X_1; A \mathrel{-\!\!*} B \triangleright_a (((Y_1 \triangleright_m \varnothing_m) \triangleright_a A) \triangleright_m Y_2); (Y_1 \triangleright_m Y_2); X_2] \qquad B \triangleright_m Y_2}{\Sigma[X_1; A \mathrel{-\!\!*} B \triangleright_a (Y_1 \triangleright_m Y_2); X_2]} \mathrel{-\!\!*} Li3$$

$$\frac{Y_1 \triangleright_m A \qquad \Sigma[X - 1; A \mathrel{-\!\!*} B \triangleright_a (Y_1 \triangleright_m (B \triangleright_a (\varnothing_m \triangleright_m Y_2))); (Y_1 \triangleright_m Y_2); X_2]}{\Sigma[X_1; A \mathrel{-\!\!*} B \triangleright_a (Y_1 \triangleright_m Y_2); X_2]} \mathrel{-\!\!*} Li4$$

$$\frac{\Sigma[X_1; (Y_1 \triangleright_m Y_2); (((Y_1 \triangleright_m \varnothing_m) \triangleright_a A) \triangleright_m Y_2) \triangleright_a A \mathrel{-\!\!\times} B; X_2] \qquad B \triangleright_m Y_2}{\Sigma[X_1; (Y_1 \triangleright_m Y_2) \triangleright_a A \mathrel{-\!\!\times} B; X_2]} \mathrel{-\!\!\times} Ri3$$

$$\frac{Y_1 \triangleright_m A \qquad \Sigma[X_1; (Y_1 \triangleright_m Y_2); (Y_1 \triangleright_m (B \triangleright_a (\varnothing_m \triangleright_m Y_2))) \triangleright_a A \mathrel{-\!\!\times} B; X_2]}{\Sigma[X_1; (Y_1 \triangleright_m Y_2) \triangleright_a A \mathrel{-\!\!\times} B; X_2]} \mathrel{-\!\!\times} Ri4$$

$$\frac{((Y_1 \triangleright_m \varnothing_m) \triangleright_a A), Y_2 \triangleright_m (X_1; (Y_1, Y_2 \triangleright_m \varnothing_m) \triangleright_a A * B; X_2) \qquad Y_2 \triangleright_m B}{Y_1, Y_2 \triangleright_m (X_1 \triangleright_a A * B; X_2)} *Ri3$$

$$\frac{(X_1; A \vee^* B \triangleright_a (\varnothing_m \triangleright_m Y_1, Y_2); X_2) \triangleright_m (A \triangleright_a (\varnothing_m \triangleright_m Y_1)), Y_2 \qquad B \triangleright_m Y_2}{(X_1; A \vee^* B \triangleright_a X_2) \triangleright_m Y_1, Y_2} \vee^* Li3$$

$$\frac{(X_1; A \mathrel{-\!\!*} B \triangleright_a (Y_1 \triangleright_m Y_2); X_2), ((Y_1 \triangleright_m \varnothing_m) \triangleright_a A) \triangleright_m Y_2 \qquad B \triangleright_m Y_2}{(X_1; A \mathrel{-\!\!*} B \triangleright_a X_2), Y_1 \triangleright_m Y_2} \mathrel{-\!\!*} Li5$$

$$\frac{Y_1 \triangleright_m A \qquad (X_1; A \mathrel{-\!\!*} B \triangleright_a (Y_1 \triangleright_m Y_2); X_2), Y_1 \triangleright_m (B \triangleright_a (\varnothing_m \triangleright_m Y_2))}{(X_1; A \mathrel{-\!\!*} B \triangleright_a X_2), Y_1 \triangleright_m Y_2} \mathrel{-\!\!*} Li6$$

$$\frac{((Y_1 \triangleright_m \varnothing_m) \triangleright_a A) \triangleright_m Y_2, (X_1; (Y_1 \triangleright_m Y_2) \triangleright_a A \mathrel{-\!\!\times} B; X_2) \qquad B \triangleright_m Y_2}{Y_1 \triangleright_m Y_2, (X_1 \triangleright_a A \mathrel{-\!\!\times} B; X_2)} \mathrel{-\!\!\times} Ri5$$

$$\frac{Y_1 \triangleright_m A \qquad Y_1 \triangleright_m (B \triangleright_a (\varnothing_m \triangleright_m Y_2)), (X_1; (Y_1 \triangleright_m Y_2) \triangleright_a A \mathrel{-\!\!\times} B; X_2)}{Y_1 \triangleright_m Y_2, (X \triangleright_a A \mathrel{-\!\!\times} B; X_2)} \mathrel{-\!\!\times} Ri6$$

Figure 3.13: Deep inference nested sequents calculus DI_{CBI} for CBI part 6

3.4 Park et al.'s Nested Sequent Calculus for BBI

After we moved on to solve theorem proving for BI logics in a new direction, Jonghyun Park and Sungwoo Park published their nested sequent calculus S_{BBI} for BBI and proved it's soundness and completeness [77]. The structure of their nested sequents, unlike ours, does not have different flavours of turnstiles, thus naturally does not impose an interleaving structure of additive sequents and multiplicative sequents. Instead, their nested sequents only have an additive flavour sequent, called "world sequent", and express the multiplicative structures using $\varnothing_m$, (W, W) and a special construction $W\langle W\rangle$, where W is a world sequent. Multiplicative structures are only allowed in the antecedent of a sequent. These are formally defined as follows:

$$\begin{array}{rrcl} \text{World Sequent} & W & ::= & \Gamma \Rightarrow_{\mathcal{B}} \Delta \\ \text{Truth Context} & \Gamma & ::= & \cdot \mid \Gamma; S \\ \text{Falsehood Context} & \Delta & ::= & \cdot \mid \Delta; A \\ \text{World State} & S & ::= & A \mid \varnothing_m \mid W, W \mid W\langle W\rangle \end{array}$$

The $\cdot$ can be seen as the additive unit $\varnothing_a$. Their clever construction of sequents reflects the ternary relation in BBI semantics as follows: a world sequent gives the true and false formulae in a world; a sequent $\Gamma; (W_1, W_2) \Rightarrow_{\mathcal{B}} \Delta$ means that the current world w_0 has two children w_1 and w_2, denoted by the world sequents W_1 and W_2 respectively, thus $R(w_1, w_2, w_0)$ holds; a sequent $\Gamma; W_1\langle W_2\rangle \Rightarrow_{\mathcal{B}} \Delta$ says that the current world w_0 has a sibling w_1 represented by the world sequent W_1, and w_0, w_1 have a common parent w_2 represented by the world sequent W_2, so $R(w_0, w_1, w_2)$ holds. The above are formally described as below, where $\Vdash_s$ is a forcing relation between worlds and structures in S_{BBI}:

$$\begin{array}{l} w \Vdash_S W_1, W_2 \text{ iff } \exists w_1, w_2 \text{ s.t. } R(w_1, w_2, w) \text{ and } w_1 \Vdash_S W_1 \text{ and } w_2 \Vdash_S W_2 \\ w \Vdash_S W_1\langle W_2\rangle \text{ iff } \exists w_1, w_2 \text{ s.t. } R(w, w_1, w_2) \text{ and } w_1 \Vdash W_1 \text{ and } w_2 \Vdash_S W_2 \end{array}$$

One can readily see the similarity between their structural connective "," and the logical connective $*$; although $W\langle W\rangle$ does not exactly resemble $-\!\!*$, it does provide a way to reason about $-\!\!*$ since we can now talk about sibling and parent. However, we point out that Park et al.'s following observations on BBI_{ND} [77, page 2] are incorrect:

1. A node can have multiple parent nodes, but each parent node determines a unique sibling node. Hence no node can have two parent nodes with the same sibling node.
2. A node can have multiple child nodes, but each child node determines another unique child node. Hence we can divide all child nodes into groups of two sibling nodes.

Observation 1 is the claim that partial-determinism holds and Observation 2 is the claim that cancellativity holds, but neither of these properties hold for BBI_{ND}, which

is what their nested sequent calculus is designed for. Interestingly, as will be shown in Section 6.5, Park et al.'s prover BBeye could not prove the formula (18) which is valid if partial-determinism (Observation 1) holds.

Their nested sequent calculus S_{BBI} has structural rules such as weakening, contraction, exchange. Their logical rules for additive connectives ($\wedge, \vee, \rightarrow, \neg$) are ordinary. The tricky part is their multiplicative rules and structural rules that "display" structures. The rules for $*$ are as expected:

$$\frac{\Gamma; (A \Rightarrow_{\mathcal{B}} \cdot), (B \Rightarrow_{\mathcal{B}} \cdot) \Rightarrow_{\mathcal{B}} \Delta}{\Gamma; A * B \Rightarrow_{\mathcal{B}} \Delta} *L \qquad \frac{\Gamma_1 \Rightarrow_{\mathcal{B}} \Delta_1; A \qquad \Gamma_2 \Rightarrow_{\mathcal{B}} \Delta_2; B}{(\Gamma_1 \Rightarrow_{\mathcal{B}} \Delta_1), (\Gamma_2 \Rightarrow_{\mathcal{B}} \Delta_2) \Rightarrow_{\mathcal{B}} A * B} *R$$

The rules for $-\!\!*$ make use of the $W\langle W\rangle$ structure in the following way:

$$\frac{\Gamma_1 \Rightarrow_{\mathcal{B}} \Delta_1; A \qquad \Gamma_2; B \Rightarrow_{\mathcal{B}} \Delta_2}{(\Gamma_1 \Rightarrow_{\mathcal{B}} \Delta_1)\langle \Gamma_2 \Rightarrow_{\mathcal{B}} \Delta_2\rangle; A -\!\!* B \Rightarrow_{\mathcal{B}} \cdot} -\!\!* L \qquad \frac{\Gamma; (A \Rightarrow_{\mathcal{B}} \cdot)\langle \cdot \Rightarrow_{\mathcal{B}} B\rangle \Rightarrow_{\mathcal{B}} \Delta}{\Gamma \Rightarrow_{\mathcal{B}} \Delta; A -\!\!* B} -\!\!* R$$

They have the following rule to shuffle structures and "display" their sequents:

$$\frac{\Gamma; W_1, (W_2, W_3 \Rightarrow_{\mathcal{B}} \cdot) \Rightarrow_{\mathcal{B}} \Delta}{\Gamma; (W_1, W_2 \Rightarrow_{\mathcal{B}} \cdot), W_3 \Rightarrow_{\mathcal{B}} \Delta} EA \qquad \frac{\Gamma_1; (\Gamma_2 \Rightarrow_{\mathcal{B}} \Delta_2), (\varnothing_m \Rightarrow_{\mathcal{B}} \cdot) \Rightarrow_{\mathcal{B}} \Delta_1}{\Gamma_1; \Gamma_2 \Rightarrow_{\mathcal{B}} \Delta_1; \Delta_2} \varnothing_m U \& \varnothing_M D$$

$$\frac{\Gamma_{c1}; (\Gamma_{c2} \Rightarrow_{\mathcal{B}} \Delta_{c2})\langle \Gamma \Rightarrow_{\mathcal{B}} \Delta\rangle \Rightarrow_{\mathcal{B}} \Delta_{c1}}{\Gamma; (\Gamma_{c1} \Rightarrow_{\mathcal{B}} \Delta_{c1}), (\Gamma_{c2} \Rightarrow_{\mathcal{B}} \Delta_{c2}) \Rightarrow_{\mathcal{B}} \Delta} TC \qquad \frac{\Gamma_p; (\Gamma \Rightarrow_{\mathcal{B}} \Delta), (\Gamma_s \Rightarrow_{\mathcal{B}} \Delta_s) \Rightarrow_{\mathcal{B}} \Delta_p}{\Gamma; (\Gamma_s \Rightarrow_{\mathcal{B}} \Delta_s)\langle \Gamma_p \Rightarrow_{\mathcal{B}} \Delta_p\rangle \Rightarrow_{\mathcal{B}} \Delta} TP$$

Their nested sequent calculus is a shallow inference system, so inference rules can only be applied on the top level sequent. Park et al. showed that their system S_{BBI} is sound and complete with respect to the Hilbert system for BBI (cf. Section 2.2) and display calculus for BBI (cf. Section 2.3). They also proved a cut-elimination theorem for S_{BBI}. We do not know if their incorrect observations are reflected in their proof system. They went on to give a more proof search friendly system based on S_{BBI}, called CS_{BBI}, which is also a shallow inference system, but build in weakening and contraction. As a result, weakening and contraction are admissible in CS_{BBI}, although it does have explicit rules to copy structures and even sequents, such as the rule EA_C, which is intricate and can hardly be presented in one line. They discussed several tactics for proof search. Some of their tactics are found useful later in our systems.

More interestingly, they showed that their rules for "displaying" structures in CS_{BBI} can be eliminated by labelling formulae with the corresponding worlds, explicitly expressing the ternary relation in the sequent, and allowing inference rules to be applied on any nested sequents, therefore supporting deep inference. Coincidentally, our independent work goes in the direction of labelled sequent calculus, not only expressing ternary relational atoms and labelled formulae in the sequent, but also supporting direct reasoning about ternary relational atoms via rules that capture the semantics of the ternary relation. In our system the inference rules are much simpler, and a derivation for the same formula is usually shorter than that in CS_{BBI}. We will give the details about our labelled calculus in the coming chapter.

Chapter 4

Labelled Sequent Calculus and Proof Search for BBI

Nested sequent calculi for BI logics are not trivial to design. Even Park et al.'s successful proof system requires very complicated rules to manipulate structures [77]. To avoid the complication caused by the mixture of additive and multiplicative structures, we take a cue from labelled sequent calculus for modal logics and develop a labelled sequent calculus for BBI directly instead of detouring through CBI. There are two reasons we do not consider CBI here: first, in labelled sequents we do not care about duality of connectives; and second, BBI is the basis of our objective separation logic after all. The idea is simple: put the semantics in the sequent and reason about them. The independent research from Park et al., as discussed in Section 3.4, also suggests that labelling formulae and expressing the ternary relation explicitly is a promising angle. Their final labelled proof system, however, does not manipulate the ternary relation directly, but still uses the structural rules for nested sequents to transfer information between worlds. Our sequent rules further build in the properties of the ternary relation in the semantics, resulting in a much simpler and more natural labelled sequent calculus.

Reynolds said in 2002 that classical separation logic was based on BBI when the exact semantics of BBI were not given. We now know that the original BBI corresponds to the non-deterministic monoidal semantics, but actually the BBI used in the heap model of separation logic is more specific and it restricts that the monoids are partial, and satisfy a number of other properties such as *indivisible unit*: the empty resource cannot be split into non-empty resources. For various reasons, some people, e.g., Park et al., seeking proof methods for the assertion language of separation logic start by looking at BBI_{ND}, partly because its Hilbert system and display calculus have already been established. This chapter also focuses on BBI with non-deterministic semantics because theorem proving for BBI_{ND} is of proof theoretical interest. Compared to the nested sequent calculi discussed in the previous chapter, our labelled calculus has great extensibility and flexibility of which we will take advantage in later chapters to handle other variants of BBI that satisfy some necessary properties in separation logic.

We give our labelled sequent calculus in Section 4.1 where we also show its soundness and completeness for BBI, followed by a cut-elimination proof in Section 4.2. We then consider proof search using labelled sequents in Section 4.3, 4.4 and 4.5. Experiments and implementation are shown in Section 4.6. We discuss the contribution of this chapter and related work in Section 4.7.

The material contained in this chapter is an extension of a conference paper by Hóu, Tiu, and Goré [53].

4.1 LS_{BBI}: A Labelled Sequent Calculus for BBI

Recall the syntax and semantics of BBI from Section 2.1.2. Our labelled sequent calculus is designed to simulate the reasoning of semantics using sequents. We explicitly use relational atoms in the sequent. Our structural rules are not like the ones in a traditional sequent calculus, but rather like Negri's modal rules [72]. Compared to Negri's labelled calculi for modal logics, our approach has two main differences. Firstly, the semantics of modal logics employ binary relations, whereas the semantics of BI logics use ternary relations. As a consequence, instead of using binary relational atoms in the calculus, we naturally build in a ternary relation structure in our calculus, and our structural rules manipulate ternary relational atoms. Secondly, some conditions in BBI semantics require unifying equivalent labels. To capture this, we use explicit global label substitutions in some structural rules. This technique is not used in the labelled calculi for modal logics.

4.1.1 Inference Rules in LS_{BBI}

We now give the details of our labelled sequent calculus with some discussion on how it is related to the semantics.

The labelled sequent calculus for BBI employs a ternary relation of worlds that is based on a non-deterministic monoid structure, à la Galmiche et al. [36].

We assume an infinite set LVar of *label variables*, and a *label constant* ϵ such that $\epsilon \notin \mathit{LVar}$. The latter is a syntactic counterpart of the ϵ world in the semantics. We shall mainly use lower case letters to range over the set $\mathcal{L}$ of all *labels*, i.e, $\mathcal{L} = \mathit{LVar} \cup \{\epsilon\}$. A *labelled formula* is an expression of the form $x : A$, where x is a label and A is a formula. A *relational atom* is an expression of the form $(x, y \rhd z)$, where x, y and z are labels. The relational atom $(x, y \rhd z)$ encodes the ternary relation $R(x, y, z)$ in the semantics. Given a relational frame $(\mathcal{M}, R, \epsilon)$, the intended interpretations of labels are worlds in $\mathcal{M}$, and the intended interpretation of the symbol $\rhd$ is the ternary relation R in the model. The interpretations of labelled formulae and relational atoms in the semantics are dependent on the interpretation of labels. The latter is given by a mapping $\rho : \{\epsilon\} \cup \mathit{LVar} \to \mathcal{M}$, with $\rho(\epsilon) = \epsilon$. That is, we fix the interpretation of the label constant ϵ to be the world ϵ in the semantics. This, however, does not

forbid mapping a label variable to the world ϵ. So in the following, when we say "an arbitrary label variable $w \in LVar$", we mean that w can be mapped to any world in the semantics. Given such a mapping ρ, a labelled formula $w : A$ is true iff $\rho(w) \Vdash A$, and a relational atom $(x, y \triangleright z)$ is true iff $R(\rho(x), \rho(y), \rho(z))$ holds.

A sequent is of the form $\Gamma \vdash \Delta$, where Γ and Δ are *structures*, defined formally via:

$$\Gamma ::= w : A \mid (x, y \triangleright z) \mid \Gamma; \Gamma \qquad\qquad \Delta ::= w : A \mid \Delta; \Delta$$

Note that relational atoms do not occur in the succedent Δ. In our definition of sequents, the structural connective ";" in the antecedent means (additive) "and" whereas in the succedent it means (additive) "or". We assume implicitly that ';' is associative and commutative. However, there is a subtlety in this interpretation of ';'. Unlike traditional Gentzen sequents, a labelled sequent $\Gamma \vdash \Delta$ does not in general correspond to a formula $\bigwedge \Gamma \rightarrow \bigvee \Delta$. This is because the interpretation of a labelled formula is dependent on the interpretation of the label it is attached to. If, however, all formulas in Γ are attached to the same label, then the Γ corresponds to the formula $\bigwedge \Gamma$.

Our use of ';' as the structural connective in labelled sequents is slightly different from the traditional (labelled) sequent notation where "," is used as the structural connective, but our notation is consistent with sequent systems for the family of Bunched Implication (BI) logics, where ";" is the additive structural connective, and "," is used to denote the multiplicative structural connective. The multiplicative structural connective is not explicitly presented in our sequent notation, but as we shall see later, it is encoded implicitly in the relational atoms.

The inference rules of our labelled system LS_{BBI} are shown in Figure 4.1, where we use p as a proposition, A, B are formulae, w, x, y, z are in the set $LVar \cup \{\epsilon\}$ of labels, ϵ is the label constant.

Definition 4.1.1 (Principal formula and relational atom)**.** *The formula shown explicitly in the conclusion of each rule is the principal formula. The relational atom shown explicitly in the conclusion of each rule is the principal relational atom.*

Our inference rules are designed to capture the semantics of BBI (cf. end of Section 2.1.2). For example, reading upwards, the left rule for $*$ considers the situation where the formula $A * B$ is true in z, this involves an existential condition in the semantics:

$$\exists a, b \text{ s.t. } (a, b \triangleright z) \text{ holds and } a \Vdash A \text{ and } b \Vdash B.$$

As in the sequent calculus for classical logic, we create fresh labels for existentially quantified variables. Therefore $*L$ creates a premise containing a new relational atom $(x, y \triangleright z)$ where x, y are fresh, and makes A true in x and B true in y. The $*R$ rule considers the case where the formula $A * B$ is false in z, by negating the semantics of $*$, we obtain that

$$\forall a, b, \text{ if } (a, b \triangleright z) \text{ holds, then } a \nVdash A \text{ or } b \nVdash B.$$

Identity and Cut:

$$\frac{}{\Gamma; w : p \vdash w : p; \Delta}\ id \qquad \frac{\Gamma \vdash x : A; \Delta \qquad \Gamma'; x : A \vdash \Delta'}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\ cut$$

Logical Rules:

$$\frac{}{\Gamma; w : \bot \vdash \Delta}\ {\bot}L \qquad \frac{\Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]}{\Gamma; w : \top^* \vdash \Delta}\ {\top^*}L \qquad \frac{}{\Gamma \vdash w : \top; \Delta}\ {\top}R \qquad \frac{}{\Gamma \vdash \epsilon : \top^*; \Delta}\ {\top^*}R$$

$$\frac{\Gamma; w : A; w : B \vdash \Delta}{\Gamma; w : A \wedge B \vdash \Delta}\ {\wedge}L \qquad \frac{\Gamma \vdash w : A; \Delta \qquad \Gamma \vdash w : B; \Delta}{\Gamma \vdash w : A \wedge B; \Delta}\ {\wedge}R$$

$$\frac{\Gamma \vdash w : A; \Delta \qquad \Gamma; w : B \vdash \Delta}{\Gamma; w : A \to B \vdash \Delta}\ {\to}L \qquad \frac{\Gamma; w : A \vdash w : B; \Delta}{\Gamma \vdash w : A \to B; \Delta}\ {\to}R$$

$$\frac{(x, y \rhd z); \Gamma; x : A; y : B \vdash \Delta}{\Gamma; z : A * B \vdash \Delta}\ {*}L \qquad \frac{(x, y \rhd z); \Gamma; x : A \vdash z : B; \Delta}{\Gamma \vdash y : A \mathrel{-\!\!*} B; \Delta}\ {\mathrel{-\!\!*}}R$$

$$\frac{(x, y \rhd z); \Gamma \vdash x : A; z : A * B; \Delta \qquad (x, y \rhd z); \Gamma \vdash y : B; z : A * B; \Delta}{(x, y \rhd z); \Gamma \vdash z : A * B; \Delta}\ {*}R$$

$$\frac{(x, y \rhd z); \Gamma; y : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad (x, y \rhd z); \Gamma; y : A \mathrel{-\!\!*} B; z : B \vdash \Delta}{(x, y \rhd z); \Gamma; y : A \mathrel{-\!\!*} B \vdash \Delta}\ {\mathrel{-\!\!*}}L$$

Structural Rules:

$$\frac{(y, x \rhd z); (x, y \rhd z); \Gamma \vdash \Delta}{(x, y \rhd z); \Gamma \vdash \Delta}\ E \qquad \frac{(u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x); \Gamma \vdash \Delta}{(x, y \rhd z); (u, v \rhd x); \Gamma \vdash \Delta}\ A$$

$$\frac{(x, \epsilon \rhd x); \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ U \qquad \frac{(x, w \rhd x); (y, y \rhd w); (x, y \rhd x); \Gamma \vdash \Delta}{(x, y \rhd x); \Gamma \vdash \Delta}\ A_C$$

$$\frac{(\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash \Delta[w'/w]}{(\epsilon, w \rhd w'); \Gamma \vdash \Delta}\ Eq_1 \qquad \frac{(\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash \Delta[w'/w]}{(\epsilon, w' \rhd w); \Gamma \vdash \Delta}\ Eq_2$$

Side conditions:
In ${\top^*}L$, Eq_1 and Eq_2, $w \neq \epsilon$.
In ${*}L$ and ${\mathrel{-\!\!*}}\ R$, x and y are label variables that do not occur in the conclusion.
In A and A_C, the label w does not occur in the conclusion.

Figure 4.1: The labelled sequent calculus LS_{BBI} for Boolean BI.

Therefore ${*}R$ checks existing relational atoms in the conclusion, and when $(x, y \rhd z)$ is found for any labels x and y, the rule creates two premises for A being false in x and B being false in y respectively. The rules for $\mathrel{-\!\!*}$ are analogous: the ${\mathrel{-\!\!*}}\ L$ rule uses an existing relational atom in the conclusion as ${\mathrel{-\!\!*}}\ L$ involves an universal condition; and the ${\mathrel{-\!\!*}}\ R$ rule creates a new relational atom with fresh labels since it involves an existential condition. Similarly, in rules A, A_C, the label w must be fresh in the premise,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots;w_2:r\vdash w_2:r;\cdots}\,id \quad \dfrac{}{\cdots;w_1:q\vdash w_1:q;\cdots}\,id}{(w_2,w_1\rhd w_0);(\epsilon,w_1\rhd w_1);(w_1,w_2\rhd w_0);\epsilon:p;\epsilon:\top^*;w_1:q;w_2:r\vdash w_0:r*q}\,{*R}}{(\epsilon,w_1\rhd w_1);(w_1,w_2\rhd w_0);\epsilon:p;\epsilon:\top^*;w_1:q;w_2:r\vdash w_0:r*q}\,E}{(\epsilon,w_4\rhd w_1);(w_1,w_2\rhd w_0);\epsilon:p;\epsilon:\top^*;w_4:q;w_2:r\vdash w_0:r*q}\,{Eq_1}}{(w_3,w_4\rhd w_1);(w_1,w_2\rhd w_0);w_3:p;w_3:\top^*;w_4:q;w_2:r\vdash w_0:r*q}\,{\top^*L}}{(w_3,w_4\rhd w_1);(w_1,w_2\rhd w_0);w_3:p\wedge\top^*;w_4:q;w_2:r\vdash w_0:r*q}\,{\wedge L}}{w_0:((p\wedge\top^*)*q)*r\vdash w_0:r*q}\,{*L\times 2}}{\vdash w_0:(((p\wedge\top^*)*q)*r)\to(r*q)}\,{\to R}$$

Figure 4.2: An example derivation for $(((p\wedge\top^*)*q)*r)\to(r*q)$ in LS_{BBI}.

as it also encodes an existential condition. Note that the rule A_C is a special case of the rule A where the two relational atoms are the same. A_C is required to guarantee the admissibility of contraction, as will be shown in the proof of Lemma 4.2.8.

In the rule $\top^*L$, there is an operation of global substitution $[\epsilon/x]$ in the premise. A substitution $\Gamma[y/x]$ is defined in the usual way: simultaneously replace every occurrence of x in Γ by y, where $x\in LVar$ and $y\in LVar\cup\{\epsilon\}$. The structural rules Eq_1 and Eq_2 also involve substitutions. Both of them are needed since substituting for the label constant ϵ is forbidden. That is, if the principal relational atom is $(\epsilon,\epsilon\rhd w)$, then we can only use the rule Eq_2 to replace every w by ϵ, but we cannot use the Eq_1 rule to replace every ϵ by w. The case where Eq_2 cannot be used is symmetric. We sometimes may write $(\Gamma\vdash\Delta)[y/x]$ for globally replacing every x by y in the entire sequent. Explicit substitution in the rules is a major novelty in our labelled sequent calculus, it makes derivations shorter in general, although it may also cause difficulties when analysing proof search methods.

The *additive rules* ($\bot L$, $\top R$, $\wedge L$, $\wedge R$, $\to L$, $\to R$) and the *multiplicative rules* ($\top^*L$, $\top^*R$, $*L$, $*R$, $-\!\!*\,L$, $-\!\!*\,R$) respectively deal with the additive/ multiplicative connectives. The *zero-premise rules* are those with no premises (id, $\bot L$, $\top R$, $\top^*R$).

We adopt a traditional (LK) style *cut* rule with context splitting. This helps give a clearer view of the use of weakening and contraction admissibility in our cut-elimination proof.

Figure 4.2 shows an example derivation in LS_{BBI}, where we omit the unimportant parts, and use $r\times n$ if a rule r is applied n times. To prove a formula, we start (at the bottom) by labelling the formula with an arbitrary label variable w, then try to apply the rules in LS_{BBI} backwards. A sequent is *provable/derivable* if every branch in the backward proof search can be closed by using a zero-premise rule.

The fact that weakening and contraction are forbidden in the multiplicative fragment of BBI is reflected in our calculus as follows. The rules $*L$ and $-\!\!*\,R$ create new relational atoms when moving from conclusions to premises, so $\vdash x:(p*p)\to p$ is not derivable. In any cut-free derivation for any formula, a relational atom of the form

$(w, w \triangleright w)$ where $w \neq \epsilon$ can never be created. We will show later that LS_{BBI} is sound, so the labelled sequent $\vdash x : p \to (p * p)$ is also not derivable in LS_{BBI}.

Definition 4.1.2 (Sequent Validity)**.** *A sequent $\Gamma \vdash \Delta$ in LS_{BBI} is valid if for all $(\mathcal{M}, R, \epsilon)$, v and ρ, if every member of Γ is true then so is some member of Δ.*

Note that BBI-validity of a formula A corresponds to the validity of the sequent $\vdash x : A$, where x is an arbitrary label variable. This correspondence is also adopted in other work for BBI [62, 77] and CBI [17], but is stronger than that used in the sequent calculus LBI for BI[1] [84], in which a formula A is valid iff $\varnothing_m \vdash A$ is provable, where $\varnothing_m$ is the multiplicative structural unit. For example, $\varnothing_m \vdash \top^*$ is provable in LBI, but the sequent $\vdash x : \top^*$ is not provable (although the sequent $\vdash \epsilon : \top^*$ is provable) in LS_{BBI}. Translated to our setting, validity of a formula A in BI would correspond to provability of $\vdash \epsilon : A$.

4.1.2 Soundness of LS_{BBI}

The soundness proof reasons about the falsifiability of sequents, as defined below.

Definition 4.1.3 (Sequent Falsifiability)**.** *A sequent $\Gamma \vdash \Delta$ in LS_{BBI} is falsifiable if there exist some $(\mathcal{M}, R, \epsilon)$, v and ρ, such that every relational atom and labelled formula in Γ is true and every labelled formula in Δ is false, where: (1) $w : A$ is true iff $\rho(w) \Vdash A$; (2) $w : A$ is false iff $\rho(w) \nVdash A$; and (3) $(x, y \triangleright z)$ is true iff $R(\rho(x), \rho(y), \rho(z))$ holds.*

Theorem 4.1.1 (Soundness)**.** *For any label $w \in LVar$ and any BBI formula F, if $\vdash w : F$ is derivable in LS_{BBI}, then F is valid in BBI_{ND}.*

Proof. Note first that $w \in LVar$ implies that w cannot be the label constant ϵ, although it can be mapped to the world ϵ. To prove the soundness of LS_{BBI}, we show that each rule preserves falsifiability upwards, as this is a more natural direction in terms of backward proof search. Therefore to prove that a rule is sound, we need to show that if the conclusion is falsifiable, then at least one of the premises is falsifiable (usually in the same choice of v, ρ, and $\mathcal{M}$). We show the case for each rule as below.

id Since there is no premise in this rule, we simply need to show that the conclusion is not falsifiable.

Suppose the sequent $\Gamma; w : P \vdash w : P; \Delta$ is falsifiable, then Γ must be true and $\rho(w) \Vdash A$ and $\rho(w) \nVdash A$ and Δ must be false. However, $\rho(w) \Vdash A$ and $\rho(w) \nVdash A$ cannot hold at the same time for any $(\mathcal{M}, \triangleright, \epsilon)$, v and ρ, so we have a contradiction, thus this sequent is not falsifiable.

[1] Since BI is defined from a proof theoretic perspective, the validity of BI formulae is defined by sequent validity in LBI.

cut Suppose the conclusion $\Gamma; \Gamma' \vdash \Delta; \Delta'$ is falsifiable in some model, given any world x from this model and any formula A, either A is true in x or A is false in x. If the former case is true, then the right premise is falsifiable in the same model; if the latter case is true, then the left premise is falsifiable in the same model.

$\bot L$ We only need to show that the conclusion is not falsifiable, which is because $\bot$ cannot be true in any world.

$\top^* L$ Assume $\Gamma; w : \top^* \vdash \Delta$ is falsifiable, then Γ is true and $\rho(w) \Vdash \top^*$ and Δ is false.

From the semantics of $\top^*$ we know that $\rho(w) \Vdash \top^*$ iff $\rho(w) = \epsilon$. Therefore by choosing the same ρ, v, and $\mathcal{M}$ for the premise, replacing every w by ϵ in Γ and Δ preserves their valuations, as we know that $\rho(\epsilon) = \epsilon$. That is, $\Gamma[\epsilon/w]$ must be true and $\Delta[\epsilon/w]$ must be false. So the premise is falsifiable.

$\top R$ The conclusion cannot be falsifiable because $\top$ cannot be false in any world.

$\top^* R$ The conclusion cannot be falsifiable because $\top^*$ cannot be false in the world ϵ.

$\wedge L$ Suppose the conclusion is falsifiable, which means $A \wedge B$ is true in w for some model. By the semantics of $\wedge$, both A and B are true in w, therefore the premise is falsifiable in the same model.

$\wedge R$ Suppose the conclusion is falsifiable, so $A \wedge B$ is false in w in some model. Then either A is false in w or B is false in w (or both). For the former case, the left premise is falsifiable in the same model; for the latter case, the right premise is falsifiable in the same model.

$\to L$ Suppose $A \to B$ is true in w in some model that falsifies the conclusion, then either A is false in w or B is true in w (or both). For the former case, the left premise is falsifiable in the same model; for the latter case, the right premise is falsifiable in the same model.

$\to R$ Suppose there is a model that falsifies the conclusion and $A \to B$ is false in w in that model. Then it must be the case that A is true in w and B is false in w, therefore the premise is falsifiable in the same model.

$* L$ Assume the conclusion is falsifiable, so under some v, ρ, $\mathcal{M}$, we have that Γ is true and $\rho(z) \Vdash A * B$ and Δ is false.

From the semantics of $A * B$, we know that $\exists a, b$ s.t. $R(a, b, \rho(z))$ and $a \Vdash A$ and $b \Vdash B$. So we can choose a mapping ρ' with $\rho' = (x \mapsto a) \cup (y \mapsto b) \cup \rho$. Since x and y are fresh, they should not affect mappings in ρ for labels that occur in the conclusion. If ρ already has mappings for x, y, we can safely override these mappings. Then, under ρ', the following hold: $(x, y \triangleright z)$ is true and Γ is true and $\rho'(x) \Vdash A$ and $\rho'(y) \Vdash B$ and Δ is false. Thus the premise is falsifiable in the same model with the label mapping ρ'.

$*R$ Assume under some v, ρ, and $\mathcal{M}$, $(x, y \triangleright z)$ is true and Γ is true and $\rho(z) \nVdash A * B$ and Δ is false.

The semantics of $A * B$ yields the following:

$$\begin{aligned}\rho(z) \nVdash A * B &\Leftrightarrow \neg(\exists a, b.\ (R(a, b, \rho(z))\ \textit{and}\ a \Vdash A\ \textit{and}\ b \Vdash B)) \\ &\Leftrightarrow \forall a, b.\ (R(a, b, \rho(z))\ \textit{doesn't hold or}\ a \nVdash A\ \textit{or}\ b \nVdash B)\end{aligned}$$

If we pick the same set of v, ρ, $\mathcal{M}$ for the premises, however, in both premises of the $*R$ rule the relational atom $(x, y \triangleright z)$ already exists, which means $\rho(x), \rho(y) \triangleright \rho(z)$ holds. So the possibility is only that either $\rho(x) \nVdash A$ or $\rho(y) \nVdash B$. Assume the former one holds, then the left premise is falsifiable, otherwise the right premise is falsifiable.

$-\!\!* L$ Suppose $(x, y \triangleright z)$ is true and Γ is true and $\rho(y) \Vdash A -\!\!* B$ and Δ is false for some v, ρ, $\mathcal{M}$. By the semantics of $-\!\!*$, for any a, b such that $R(a, \rho(y), b)$, $a \nVdash A$ or $b \Vdash B$. Since $R(\rho(x), \rho(y), \rho(z))$ is true, we know that $\rho(x) \nVdash A$ or $\rho(z) \Vdash B$. For the former case, the left premise is falsifiable in the same model; for the latter case, the right premise is falsifiable in the same model.

$-\!\!* R$ Suppose the conclusion is falsifiable for some v, ρ, $\mathcal{M}$, where $\rho(y) \nVdash A -\!\!* B$. By negating the semantics, we have the following:

$$\begin{aligned}\rho(y) \nVdash A -\!\!* B &\Leftrightarrow \neg(\forall a, b.((R(a, \rho(y), b)\ \textit{and}\ a \Vdash A)\ \textit{implies}\ b \Vdash B)) \\ &\Leftrightarrow \neg(\forall a, b.(\neg(R(a, \rho(y), b)\ \textit{and}\ a \Vdash A)\ \textit{or}\ b \Vdash B)) \\ &\Leftrightarrow \exists a, b.((R(a, \rho(y), b)\ \textit{and}\ a \Vdash A)\ \textit{and}\ b \nVdash B)\end{aligned}$$

Suppose the worlds a, b respectively makes A true and makes B false and the relation $R(a, \rho(y), b)$ holds. Let us choose a mapping ρ' that extends ρ with $\{(x \mapsto a), (z \mapsto b)\}$. Since x, y do not occur in the conclusion, the mappings of them do not conflict with mappings for labels in the conclusion. Thus in this model with the label mapping ρ', we have $\rho'(x) \Vdash A$ and $\rho'(z) \nVdash B$, which means that the premise is falsifiable.

E Suppose the conclusion is falsifiable, so $R(\rho(x), \rho(y), \rho(z))$ holds in some model. By commutativity, $R(\rho(y), \rho(x), \rho(z))$ also holds, thus the premise is also falsifiable in the same model.

A Suppose the conclusion is falsifiable, thus there is a model that makes $R(\rho(x), \rho(y), \rho(z))$ and $R(\rho(u), \rho(v), \rho(x))$ true. By associativity, there is some world a such that $R(\rho(u), a, \rho(z))$ and $R(\rho(y), \rho(v), a)$ are true. Let us extend ρ with $(w \mapsto a)$ and let the result be ρ', this should not affect existing mappings for labels occur in the conclusion. Then the extended model with $\mathcal{M}$, v, ρ' falsifies the premise.

A_C This is just a special case of the rule A where the two principal relational atoms are the same. The soundness is proved by the above argument.

U Suppose the conclusion is falsifiable in some $\mathcal{M}$, v, ρ. If the label x occurs in the conclusion, then by identity, $R(\rho(x), \epsilon, \rho(x))$ holds, thus the premise is falsifiable in the same model. If x does not occur in the conclusion, we just need to extend the model with it as follows: $\mathcal{M}' = \mathcal{M} \cup \{a\}$ and $\rho' = \rho \cup \{(a \mapsto x)\}$, where a does not occur in $\mathcal{M}$. Again by identity, $R(\rho(x), \epsilon, \rho(x))$ holds. Now the premise is falsifiable in the model with $\mathcal{M}'$, ρ', and the same choice of valuation v.

Eq_1/Eq_2 Suppose the conclusion is falsifiable in some model where $R(\epsilon, \rho(w), \rho(w'))$ holds. By commutativity and identity, $\rho(w) = \rho(w')$ in this model, thus the relation $R(\epsilon, \rho(w'), \rho(w'))$ holds. Moreover, globally replacing every $\rho(w)$ with $\rho(w')$ does not change the falsifiability of the sequent. So the premise is falsifiable in the same model. The case for Eq_2 can be argued the same way. □

4.1.3 Completeness of LS_{BBI}

Although not explicitly shown, our labelled sequent calculus LS_{BBI} is able to deal with other classical logical connectives such as $\vee$ and $\neg$. The rules for these connectives can be derived from existing rules in LS_{BBI} via $\neg A \equiv A \to \bot$ and $A \vee B \equiv \neg(\neg A \wedge \neg B)$. We give the derived rules for these connectives as below, these rules will be used in the derivations in this section.

$$\frac{\Gamma \vdash w : A; \Delta}{\Gamma; w : \neg A \vdash \Delta}\ {\scriptstyle \neg L} \qquad \frac{\Gamma; w : A \vdash \Delta}{\Gamma \vdash w : \neg A; \Delta}\ {\scriptstyle \neg R}$$

$$\frac{\Gamma; w : A \vdash \Delta \qquad \Gamma; w : B \vdash \Delta}{\Gamma; w : A \vee B \vdash \Delta}\ {\scriptstyle \vee L} \qquad \frac{\Gamma \vdash w : A; w : B; \Delta}{\Gamma \vdash w : A \vee B; \Delta}\ {\scriptstyle \vee R}$$

We prove the completeness of LS_{BBI} by showing that every derivation of a formula in the Hilbert system for BBI (cf. Section 2.2) can be translated to a derivation in LS_{BBI}, possibly using the *cut* rule. That is, a formula A is translated to the labelled sequent $\vdash w : A$.

Theorem 4.1.2 (Completeness)**.** *For any label $w \in LVar$ and any BBI formula F, if F is valid in BBI_{ND}, then $\vdash w : F$ is derivable in LS_{BBI}.*

Proof. Again, it is not possible for the label variable w to be the label constant ϵ. As the Hilbert system for BBI is complete, there is a derivation Π of F in the Hilbert system. We show that one can construct an LS_{BBI} derivation Π' of the sequent $\vdash w : F$, for any label $w \neq \epsilon$. It is enough to show that each axiom and each rule of the Hilbert system can be derived. Please refer to Section 2.2 for the Hilbert system for BBI. The axiom $A \to (B \to A)$ can be proved as follows:

$$\dfrac{\dfrac{\dfrac{}{w : A; w : B \vdash w : A}\,id}{w : A \vdash w : B \to A}\,{\to}R}{\vdash w : A \to (B \to A)}\,{\to}R$$

The proof for axiom $A \to (A \vee B)$ runs as below.

$$\dfrac{\dfrac{\dfrac{}{w : A \vdash w : A; w : B}\,id}{w : A \vdash w : A \vee B}\,{\vee}R}{\vdash w : A \to (A \vee B)}\,{\to}R$$

The axiom $B \to (A \vee B)$ can be proved similar as above.
The axiom $(A \wedge B) \to A$ is shown as follows:

$$\dfrac{\dfrac{\dfrac{}{w : A; w : B \vdash w : A}\,id}{w : A \wedge B \vdash w : A}\,{\wedge}L}{\vdash w : (A \wedge B) \to A}\,{\to}R$$

The axiom $(A \wedge B) \to B$ can be proven by the same routine. Next we give a derivation for the axiom $A \to (B \to (A \wedge B))$ as below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{w : A; w : B \vdash w : B}\,id \quad \dfrac{}{w : A; w : B \vdash w : A}\,id}{w : A; w : B \vdash w : A \wedge B}\,{\wedge}R}{w : A \vdash w : B \to (A \wedge B)}\,{\to}R}{\vdash w : A \to (B \to (A \wedge B))}\,{\to}R$$

The axiom $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$ is proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{w : A \to (B \to C); w : A \vdash w : A; w : C}\,id \quad w : A \to (B \to C); w : B; w : A \vdash w : C}{w : A \to (B \to C); w : A \to B; w : A \vdash w : C}\,{\to}L}{w : A \to (B \to C); w : A \to B \vdash w : A \to C}\,{\to}R}{w : A \to (B \to C) \vdash w : (A \to B) \to (A \to C)}\,{\to}R}{\vdash w : (A \to (B \to C)) \to ((A \to B) \to (A \to C))}\,{\to}R$$

The top right premise can be derived as below:

$$\dfrac{\dfrac{}{\cdots; w : A \vdash w : A; \cdots}\,id \quad \dfrac{\dfrac{}{w : B; \cdots \vdash w : B; \cdots}\,id \quad \dfrac{}{w : C; \cdots \vdash w : C}\,id}{w : B \to C; w : B; w : A \vdash w : C}\,{\to}L}{w : A \to (B \to C); w : B; w : A \vdash w : C}\,{\to}L$$

The proof for the axiom $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$ runs as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{w : B \to C; w : A \vee B \vdash w : A; w : C \quad \dfrac{}{w : C; w : B \to C; w : A \vee B \vdash w : C}\,id}{w : A \to C; w : B \to C; w : A \vee B \vdash w : C}\,{\to}L}{w : A \to C; w : B \to C \vdash w : (A \vee B) \to C}\,{\to}R}{w : A \to C \vdash w : (B \to C) \to ((A \vee B) \to C)}\,{\to}R}{\vdash w : (A \to C) \to ((B \to C) \to ((A \vee B) \to C))}\,{\to}R$$

where the derivation for the left premise is shown below:

$$\dfrac{\dfrac{}{\cdots ; w : A \vdash w : A; \cdots}\,id \qquad \dfrac{\dfrac{}{w : B \vdash w : B; \cdots}\,id \qquad \dfrac{}{w : C; \cdots \vdash \cdots ; w : C}\,id}{w : B \to C; w : B \vdash w : A; w : C}\,{\to L}}{w : B \to C; w : A \vee B \vdash w : A; w : C}\,{\vee L}$$

The proof for the axiom $\bot \to A$ is trivial:

$$\dfrac{\dfrac{}{w : \bot \vdash w : A}\,{\bot L}}{\vdash w : \bot \to A}\,{\to R}$$

The proof for the double negation elimination axiom is also easy:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{w : A \vdash w : A}\,id}{\vdash w : \neg A; w : A}\,{\neg R}}{w : \neg\neg A \vdash w : A}\,{\neg L}}{\vdash w : (\neg\neg A) \to A}\,{\to R}$$

The deduction rule of modus ponens can be proved by the *cut* rule, assuming that each premise of modus ponens can be derived.

$$\dfrac{\vdash w : A \to B \qquad \dfrac{\vdash w : A \qquad \dfrac{}{w : A \vdash w : A; w : B}\,id}{\vdash w : A; w : B}\,{cut} \qquad \dfrac{}{w : B \vdash w : B}\,id}{w : A \to B \vdash w : B}\,{\to L}}{\vdash w : B}\,{cut}$$

The two open branches are derivable since their end sequents are exactly the premises of the modus ponens rule.

Now let us continue with the axioms and deduction rules in the multiplicative fragment. The axiom $A \to (\top^* * A)$ is proven via the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots \vdash \epsilon : \top^*; \cdots}\,{\top^* R} \qquad \dfrac{}{\cdots ; w : A \vdash w : A; \cdots}\,id}{(\epsilon, w \rhd w); \cdots ; w : A \vdash w : \top^* * A}\,{*R}}{(w, \epsilon \rhd w); w : A \vdash w : \top^* * A}\,E}{w : A \vdash w : \top^* * A}\,U}{\vdash w : A \to (\top^* * A)}\,{\to R}$$

The axiom $(\top^* * A) \to A$ can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon, w \rhd w); w : A \vdash w : A}\,id}{(\epsilon, w_2 \rhd w); w_2 : A \vdash w : A}\,{Eq_1}}{(w_1, w_2 \rhd w); w_1 : \top^*; w_2 : A \vdash w : A}\,{\top^* L}}{w : \top^* * A \vdash w : A}\,{*L}}{\vdash w : (\top^* * A) \to A}\,{\to R}$$

We give a derivation for the axiom $(A * B) \to (B * A)$ as below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots ; w_2 : B \vdash w_2 : B; \cdots}\,id \quad \dfrac{}{\cdots ; w_1 : A \vdash w_1 : A; \cdots}\,id}{(w_2, w_1 \rhd w); \cdots ; w_1 : A; w_2 : B \vdash w : B * A}\,{*R}}{(w_1, w_2 \rhd w); w_1 : A; w_2 : B \vdash w : B * A}\,E}{w : A * B \vdash w : B * A}\,{*L}}{\vdash w : (A * B) \to (B * A)}\,{\to R}$$

The proof for the axiom $(A * (B * C)) \to ((A * B) * C)$ runs as below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots ; w_1 : A \vdash w_1 : A; \cdots}\,id \quad \dfrac{}{\cdots ; w_3 : B \vdash w_3 : B; \cdots}\,id}{\cdots ; (w_1, w_3 \rhd w_5); w_1 : A; w_3 : B \vdash w_5 : A * B; \cdots}\,{*R} \quad \dfrac{}{\cdots ; w_4 : C \vdash w_4 : C; \cdots}\,id}{\cdots ; (w_5, w_4 \rhd w); (w_1, w_3 \rhd w_5); w_1 : A; w_3 : B; w_4 : C \vdash w : (A * B) * C}\,{*R}}{\cdots ; (w_4, w_5 \rhd w); (w_1, w_3 \rhd w_5); w_1 : A; w_3 : B; w_4 : C \vdash w : (A * B) * C}\,E}{(w_2, w_1 \rhd w); \cdots ; (w_4, w_3 \rhd w_2); w_1 : A; w_3 : B; w_4 : C \vdash w : (A * B) * C}\,A}{(w_1, w_2 \rhd w); (w_3, w_4 \rhd w_2); w_1 : A; w_3 : B; w_4 : C \vdash w : (A * B) * C}\,{E \times 2}}{(w_1, w_2 \rhd w); w_1 : A; w_2 : (B * C) \vdash w : (A * B) * C}\,{*L}}{w : A * (B * C) \vdash w : (A * B) * C}\,{*L}}{\vdash w : (A * (B * C)) \to ((A * B) * C)}\,{\to R}$$

Suppose there is a derivation in the Hilbert system for BBI using the deduction rule $*$ as below:

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \vdash A \to C & \vdash B \to D\end{array}}{\vdash (A * B) \to (C * D)}\,*$$

We give a derivation in LS_{BBI} as below, assuming as the induction hypotheses that the two premises can be derived.

$$\dfrac{\dfrac{\vdash w_1 : A \to C \qquad \dfrac{\dfrac{}{\cdots ; w_1 : A \vdash w_1 : A; \cdots}\,id \qquad (w_1, w_2 \rhd w); \cdots ; w_2 : B; w_1 : C \vdash w : C * D}{(w_1, w_2 \rhd w); w_1 : A; w_2 : B; w_1 : A \to C \vdash w : C * D}\,{\to L}}{(w_1, w_2 \rhd w); w_1 : A; w_2 : B \vdash w : C * D}\,cut}{w : A * B \vdash w : C * D}\,{*L}}{\vdash w : (A * B) \to (C * D)}\,{\to R}$$

where the rightmost branch is derived as follows:

$$\dfrac{\vdash w_2 : B \to D \qquad \dfrac{\dfrac{}{\cdots ; w_2 : B \vdash w_2 : B; \cdots}\,id \qquad (w_1, w_2 \rhd w); \cdots ; w_2 : D; w_1 : C \vdash w : C * D}{(w_1, w_2 \rhd w); \cdots ; w_2 : B; w_2 : B \to D; w_1 : C \vdash w : C * D}\,{\to L}}{(w_1, w_2 \rhd w); \cdots ; w_2 : B; w_1 : C \vdash w : C * D}\,cut$$

and the rightmost branch of the above derivation can be proved as below:

$$\dfrac{\dfrac{}{\cdots ; w_1 : C \vdash w_1 : C; \cdots}\,id \quad \dfrac{}{\cdots ; w_2 : D \vdash w_2 : D; \cdots}\,id}{(w_1, w_2 \rhd w); \cdots ; w_2 : D; w_1 : C \vdash w : C * D}\,{*R}$$

Now the only two open branches can be proved by the induction hypotheses.

Next consider the rule $-\!\!* 1$, suppose Π is the derivation:

$$\dfrac{\begin{array}{c}\Pi_1\\ A \to (B \mathrel{-\!\!*} C)\end{array}}{(A * B) \to C} \mathrel{-\!\!*} 1$$

We start a backwards proof search from the end sequent as follows:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_1'\\ \vdash w_1 : A \to (B \mathrel{-\!\!*} C)\end{array} \qquad \begin{array}{c}\Pi_2\\ (w_1, w_2 \rhd w); w_1 : A \to (B \mathrel{-\!\!*} C); w_1 : A; w_2 : B \vdash w : C\end{array}}{(w_1, w_2 \rhd w); w_1 : A; w_2 : B \vdash w : C}\, cut}{w : A * B \vdash w : C} *L}{\vdash w : (A * B) \to C} \to R$$

where Π_1' comes from Π_1 via the induction hypothesis, Π_2 is shown as below, where $\Gamma = \{(w_1, w_2 \rhd w); w_1 : A; w_2 : B\}$).

$$\dfrac{\dfrac{}{\Gamma \vdash w_1 : A}\, id \qquad \dfrac{\dfrac{}{\Gamma; w_1 : B \mathrel{-\!\!*} C \vdash w_2 : B}\, id \qquad \dfrac{}{\Gamma; w_1 : B \mathrel{-\!\!*} C; w : C \vdash w : C}\, id}{\Gamma; w_1 : B \mathrel{-\!\!*} C \vdash w : C} \mathrel{-\!\!*} L}{\Gamma; w_1 : A \to (B \mathrel{-\!\!*} C) \vdash w : C} \to L$$

Finally, suppose there is a derivation in the Hilbert system using the rule $\mathrel{-\!\!*} 2$ as below:

$$\dfrac{\begin{array}{c}\Pi\\ \vdash (A * B) \to C\end{array}}{\vdash A \to (B \mathrel{-\!\!*} C)} \mathrel{-\!\!*} 2$$

Assuming as the induction hypothesis that the premise $(A * B) \to C$ is derivable in LS_{BBI}, we give the following derivation for the end sequent:

$$\dfrac{\dfrac{\dfrac{\vdash w_2 : (A * B) \to C \qquad (w_1, w \rhd w_2); w : A; w_1 : B; w_2 : (A * B) \to C \vdash w_2 : C}{(w_1, w \rhd w_2); w : A; w_1 : B \vdash w_2 : C}\, cut}{w : A \vdash w : B \mathrel{-\!\!*} C} \mathrel{-\!\!*} R}{\vdash w : A \to (B \mathrel{-\!\!*} C)} \to R$$

where the right branch runs as follows:

$$\dfrac{\dfrac{\dfrac{}{\cdots; w_1 : B \vdash w_1 : B; \cdots}\, id \qquad \dfrac{}{\cdots; w : A \vdash w : A; \cdots}\, id}{(w_1, w \rhd w_2); w : A; w_1 : B \vdash w_2 : A * B; \cdots} \qquad \dfrac{}{\cdots;; w_2 : C \vdash w_2 : C}\, id}{(w_1, w \rhd w_2); w : A; w_1 : B; w_2 : (A * B) \to C \vdash w_2 : C} \to L$$

The only open branch can be derived by the induction hypothesis. □

Corollary 4.1.3 (Formula validity). *For any label variable $w \in LVar$ and any BBI formula F, the formula F is valid in BBI_{ND} iff $\vdash w : F$ is derivable in LS_{BBI}.*

Proof. The "if" direction comes from the soundness proof (of Theorem 4.1.1); the "only-if" direction follows from the completeness proof (of Theorem 4.1.2) because when we prove a formula F, we derive $\vdash w : F$ for an arbitrary label variable w. Thus F is true at any world for any valuation v, mapping ρ, and relational frame $(\mathcal{M}, R, \epsilon)$. □

4.2 Cut-elimination for LS_{BBI}

This section proves the cut-elimination theorem for our labelled sequent calculus. The general proof outlined here is similar to the cut-elimination proof for labelled systems for modal logic [72], i.e., we start by proving a substitution lemma for labels, followed by proving the invertibility of inference rules, weakening admissibility, and contraction admissibility, before proceeding to the main cut-elimination proof. As there are many case analyses in these proofs, we only outline the important parts here.

Given a derivation Π, its *height* $ht(\Pi)$ is defined as the length of the longest branch in Π. We first show that weakening individual labelled formula or relational atom is admissible.

Lemma 4.2.1. *For all structures* Γ, Δ, *labelled formula* $w : A$, *and relational atom* $(x, y \triangleright z)$, *if* $\Gamma \vdash \Delta$ *is derivable, then the following sequents can be derived within the same height:*

$$\Gamma; w : A \vdash \Delta \qquad \Gamma \vdash w : A; \Delta \qquad (x, y \triangleright z); \Gamma \vdash \Delta.$$

Proof. By induction on $ht(\Pi)$. Since id, $\bot L$, $\top R$, and $\top^* R$ all have weakening built in, the base case trivially holds. For the inductive cases, the only nontrivial case is for $*L$ and $-\!\!* R$, where new labels have to be introduced. These labels can be systematically renamed to make sure that they do not clash with the labels in the weakened formula/relational atom. This technique is also used in the proof of Lemma 4.2.2. □

The substitution lemma shows that provability is preserved under arbitrary substitutions of labels. When proving this lemma, we will frequently use the fact that, when applied on a sequent, the sequence (i.e., the order matters) of substitutions $[y/x][z/y]$ have the same effect as $[z/x][z/y]$, for any label z and any label variables $x, y \in LVar$.

Lemma 4.2.2 (Substitution)**.** *For any label variable* $x \in LVar$ *and any label* y, *if* Π *is a* LS_{BBI} *derivation for the sequent* $\Gamma \vdash \Delta$ *then there is a* LS_{BBI} *derivation* Π' *of the sequent* $\Gamma[y/x] \vdash \Delta[y/x]$ *where every occurrence of label* x *is replaced by label* y, *such that* $ht(\Pi') \leq ht(\Pi)$.

Proof. By induction on $ht(\Pi)$.
(Base case) If $ht(\Pi) = 0$, then the only applicable rules are id, $\bot L$, $\top R$ and $\top^* R$. Since ϵ cannot be in the domain of a substitution, $x \neq \epsilon$. If x is not on the principal formula, then the substitution does not affect the original derivation. Note that since we are not allowed to substitute for the label ϵ, the proof for $\top^* R$ can only be this case. If x is

the label of the principal formula, we can still derive the required sequent because the substitution is global, so *id*, if used in the original derivation, can still be used in the new derivation; and $\bot L$, $\top R$ are independent of labels.
(Inductive case) If $ht(\Pi) > 0$, then consider the last rule applied in the derivation. We consider three main cases.

1. Neither x nor y is the label of the principal formula.

   (a) Suppose the last rule applied is $\top^* L$, and $x \neq w$ and $y \neq w$, and Π is the following derivation:

   $$\dfrac{\begin{array}{c}\Pi_1 \\ \Gamma'[\epsilon/w] \vdash \Delta[\epsilon/w]\end{array}}{\Gamma'; w : \top^* \vdash \Delta}\ \top^* L$$

   By the induction hypothesis, there is a derivation Π_1' of $\Gamma'[\epsilon/w][y/x] \vdash \Delta[\epsilon/w][y/x]$ with $ht(\Pi_1') \leq ht(\Pi_1)$. Since x and y are different from w, this sequent is equal to $\Gamma'[y/x][\epsilon/w] \vdash \Delta[y/x][\epsilon/w]$. Therefore Π' is constructed as follows.

   $$\dfrac{\begin{array}{c}\Pi_1' \\ \Gamma'[y/x][\epsilon/w] \vdash \Delta[y/x][\epsilon/w]\end{array}}{\Gamma'[y/x]; w : \top^* \vdash \Delta[y/x]}\ \top^* L$$

   Obviously $ht(\Pi') \leq ht(\Pi)$.

   (b) If the last rule applied is Eq_1, we distinguish the following cases: x is not w or w'; $x = w$; $x = w'$.

      i. $x \neq w$ and $x \neq w'$. The original derivation is as follows.

      $$\dfrac{\begin{array}{c}\Pi_1 \\ (\epsilon, w \triangleright w); \Gamma'[w/w'] \vdash \Delta[w/w']\end{array}}{(\epsilon, w' \triangleright w); \Gamma' \vdash \Delta}\ Eq_1$$

      A. If $y \neq w$ and $y \neq w'$, by the induction hypothesis, there is a derivation Π_1' of $(\epsilon, w \triangleright w); \Gamma'[w/w'][y/x] \vdash \Delta[w/w'][y/x]$ with $ht(\Pi_1') \leq ht(\Pi_1)$. Since x, y, w, w' are different labels, this sequent is equal to $(\epsilon, w \triangleright w); \Gamma'[y/x][w/w'] \vdash \Delta[y/x][w/w']$. Thus the derivation Π' is constructed as follows.

      $$\dfrac{\begin{array}{c}\Pi_1' \\ (\epsilon, w \triangleright w); \Gamma'[y/x][w/w'] \vdash \Delta[y/x][w/w']\end{array}}{(\epsilon, w' \triangleright w); \Gamma'[y/x] \vdash \Delta[y/x]}\ Eq_1$$

      B. If $y = w$, this case is similar to Case 1.(b).i.A.

      C. Suppose $y = w'$. We need to derive $(\epsilon, y \triangleright w); \Gamma'[y/x] \vdash \Delta[y/x]$. If $y \neq \epsilon$, we construct Π' by first applying Eq_1 bottom-up:

      $$\dfrac{(\epsilon, w \triangleright w); \Gamma'[y/x][w/y] \vdash \Delta[y/x][w/y]}{(\epsilon, y \triangleright w); \Gamma'[y/x] \vdash \Delta[y/x]}\ Eq_1$$

Now the premise is equal to $(\epsilon, w \triangleright w); \Gamma'[w/y][w/x] \vdash \Delta[w/y][w/x]$, and by the induction hypothesis, there is a derivation Π_1' of this sequent, with $ht(\Pi_1') \leq ht(\Pi_1)$.

If $y = \epsilon$, then we need to apply Eq_2 instead of Eq_1:

$$\frac{(\epsilon, \epsilon \triangleright \epsilon); \Gamma'[\epsilon/x][\epsilon/w] \vdash \Delta[\epsilon/x][\epsilon/w]}{(\epsilon, \epsilon \triangleright w); \Gamma'[\epsilon/x] \vdash \Delta[\epsilon/x]}\ Eq_2$$

Note that the sequent $(\epsilon, \epsilon \triangleright \epsilon); \Gamma'[\epsilon/x][\epsilon/w] \vdash \Delta[\epsilon/x][\epsilon/w]$ is the same as

$$(\epsilon, \epsilon \triangleright \epsilon); \Gamma'[w/w'][\epsilon/w][\epsilon/x] \vdash \Delta[w/w'][\epsilon/w][\epsilon/x].$$

So the premise can be proved by two successive applications of the induction hypothesis to Π_1, one using substitution $[\epsilon/w]$ and the other using substitution $[\epsilon/x]$. Here we can apply the induction hypothesis twice to Π_1 because substitution does not increase the height of derivations.

ii. $x = w$ (so w cannot be ϵ).

A. If $y \neq w'$, then Π has the form:

$$\frac{\begin{array}{c}\Pi_1 \\ (\epsilon, x \triangleright x); \Gamma'[x/w'] \vdash \Delta[x/w']\end{array}}{(\epsilon, w' \triangleright x); \Gamma' \vdash \Delta}\ Eq_1$$

By the induction hypothesis we have the following derivation:

$$\begin{array}{c}\Pi_1' \\ (\epsilon, y \triangleright y); \Gamma'[x/w'][y/x] \vdash \Delta[x/w'][y/x]\end{array}$$

The end sequent is equal to the following:

$$(\epsilon, y \triangleright y); \Gamma'[y/x][y/w'] \vdash \Delta[y/x][y/w'].$$

Then by using Eq_1, we construct Π' as follows:

$$\frac{\begin{array}{c}\Pi_1' \\ (\epsilon, y \triangleright y); \Gamma'[y/x][y/w'] \vdash \Delta[y/x][y/w']\end{array}}{(\epsilon, w' \triangleright y); \Gamma'[y/x] \vdash \Delta[y/x]}\ Eq_1$$

B. If $y = w'$, then Π has the form:

$$\frac{\begin{array}{c}\Pi_1 \\ (\epsilon, x \triangleright x); \Gamma'[x/y] \vdash \Delta[x/y]\end{array}}{(\epsilon, y \triangleright x); \Gamma' \vdash \Delta}\ Eq_1$$

By the induction hypothesis, we have the following derivation:

$$\begin{array}{c}\Pi_1' \\ (\epsilon, y \triangleright y); \Gamma'[x/y][y/x] \vdash \Delta[x/y][y/x]\end{array}$$

Since in the end sequent we first replace every y by x, then change every x back to y, the effect is the same as just keeping every y unchanged and only replace every x by y. Thus the end sequent is equal to:

$$(\epsilon, y \triangleright y); \Gamma'[y/x] \vdash \Delta[y/x]$$

which is exactly what we need to derive. Therefore we let $\Pi' = \Pi'_1$. Notice that in this case $ht(\Pi') < ht(\Pi)$.

iii. $x = w'$.

A. If $y \neq w$ and $y \neq \epsilon$, the original derivation is as follows.

$$\dfrac{\begin{array}{c}\Pi_1 \\ (\epsilon, w \triangleright w); \Gamma'[w/x] \vdash \Delta[w/x]\end{array}}{(\epsilon, x \triangleright w); \Gamma' \vdash \Delta}\ Eq_1$$

By the induction hypothesis (instead of replacing every x by y, we now replace every y by w), we have the following derivation:

$$\begin{array}{c}\Pi'_1 \\ (\epsilon, w \triangleright w); \Gamma'[w/x][w/y] \vdash \Delta[w/x][w/y]\end{array}$$

The end sequent is equal to:

$$(\epsilon, w \triangleright w); \Gamma'[y/x][w/y] \vdash \Delta[y/x][w/y]$$

Thus Π' is constructed as follows.

$$\dfrac{\begin{array}{c}\Pi'_1 \\ (\epsilon, w \triangleright w); \Gamma'[y/x][w/y] \vdash \Delta[y/x][w/y]\end{array}}{(\epsilon, y \triangleright w); \Gamma'[y/x] \vdash \Delta[y/x]}\ Eq_1$$

B. If $y = \epsilon$ and $w \neq \epsilon$, we need to derive the following sequent:

$$(\epsilon, \epsilon \triangleright w); \Gamma'[\epsilon/x] \vdash \Delta[\epsilon/x]$$

By the induction hypothesis, replacing every w by ϵ in Π_1, then using the rule Eq_2, we get the new derivation:

$$\dfrac{\begin{array}{c}\Pi'_1 \\ (\epsilon, \epsilon \triangleright \epsilon); \Gamma'[\epsilon/x][\epsilon/w] \vdash \Delta[\epsilon/x][\epsilon/w]\end{array}}{(\epsilon, \epsilon \triangleright w); \Gamma'[\epsilon/x] \vdash \Delta[\epsilon/x]}\ Eq_2$$

C. If $y = w$, then the premise of the last rule is exactly what we need to derive.

(c) If the last rule applied is Eq_2, we consider three cases: $x \neq w$ and $y \neq w$; $x = w$; and $y = w$. These are symmetric to the case where the last rule is Eq_1, already discussed above.

(d) If the last rule in the derivation is any other rule, we can simply apply the induction hypothesis on the premise(s) using the substitution $[y/x]$ and then use the same rule to derive the conclusion. The only two cases that need more care are for $*L$ and $-\!\!* R$, in the situation where y coincides with a label to be created in the premise. In these cases, we need to first apply the induction hypothesis using $[y'/y]$ on the premise where y' is fresh and different from y to ensure the freshness of labels created by $*L$ and $-\!\!* R$, then apply the induction hypothesis again on the premise using $[y/x]$, finally apply the corresponding rule to derive the conclusion. Note

that all structural rules only have this case, as they do not have a principal formula in the conclusion.

2. y is the label of the principal formula. Most of the cases follow similarly as above, except for $\top^* L$. In this case the original derivation is as follows.

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma'[\epsilon/y] \vdash \Delta[\epsilon/y]\end{array}}{\Gamma'; y : \top^* \vdash \Delta}\ \top^* L$$

Our goal is to derive $\Gamma'[y/x]; y : \top^* \vdash \Delta[y/x]$. Applying $\top^* L$ to it as in backward proof search, we get

$$\Gamma'[y/x][\epsilon/y] \vdash \Delta[y/x][\epsilon/y]$$

Note that this sequent is equal to $\Gamma'[\epsilon/y][\epsilon/x] \vdash \Delta[\epsilon/y][\epsilon/x]$, and from the induction hypothesis we know that there is a derivation of this sequent of height less than or equal to $ht(\Pi)$.

3. x is the label of the principal formula.

   (a) For the additive rules, since the labels stay the same in the premises and conclusions of the rules, even if the label of the principal formula is replaced by some other label, we can still apply the induction hypothesis on the premise, then use the rule to derive the conclusion.

   For $\wedge L$,

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma'; x : A; x : B \vdash \Delta\end{array}}{\Gamma'; x : A \wedge B \vdash \Delta}\ \wedge L \qquad \rightsquigarrow \qquad \dfrac{\begin{array}{c}\Pi_1'\\ \Gamma'[y/x]; y : A; y : B \vdash \Delta[y/x]\end{array}}{\Gamma'[y/x]; y : A \wedge B \vdash \Delta[y/x]}\ \wedge L$$

   For $\wedge R$,

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma' \vdash x : A; \Delta\end{array} \qquad \begin{array}{c}\Pi_2\\ \Gamma' \vdash x : B; \Delta\end{array}}{\Gamma' \vdash x : A \wedge B; \Delta}\ \wedge R \rightsquigarrow$$

$$\dfrac{\begin{array}{c}\Pi_1'\\ \Gamma'[y/x] \vdash y : A; \Delta[y/x]\end{array} \qquad \begin{array}{c}\Pi_2'\\ \Gamma'[y/x] \vdash y : B; \Delta[y/x]\end{array}}{\Gamma'[y/x] \vdash y : A \wedge B; \Delta[y/x]}\ \wedge R$$

   For $\rightarrow L$,

$$\dfrac{\begin{array}{c}\Pi_1\\ \Gamma' \vdash x : A; \Delta\end{array} \qquad \begin{array}{c}\Pi_2\\ \Gamma'; x : B \vdash \Delta\end{array}}{\Gamma'; x : A \rightarrow B \vdash \Delta}\ \rightarrow L \rightsquigarrow$$

$$\dfrac{\begin{array}{cc}\Pi_1' & \Pi_2' \\ \Gamma'[y/x] \vdash y : A; \Delta[y/x] & \Gamma'[y/x]; y : B \vdash \Delta[y/x]\end{array}}{\Gamma'[y/x]; y : A \to B \vdash \Delta[y/x]} \to L$$

For $\to R$,

$$\dfrac{\Pi_1}{\dfrac{\Gamma'; x : A \vdash x : B; \Delta}{\Gamma' \vdash x : A \to B; \Delta} \to R} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Pi_1'}{\Gamma'[y/x]; y : A \vdash y : B; \Delta[y/x]} \to R}{\Gamma'[y/x] \vdash y : A \to B; \Delta[y/x]}$$

(b) For multiplicative rules that do not produce fresh labels ($*R$, $-\!\!* L$, $\top^* L$), we can proceed similarly as in the additive cases, except for the $\top^* L$ rule. For the $\top^* L$ rule, suppose the original derivation Π is

$$\dfrac{\Pi_1}{\dfrac{\Gamma'[\epsilon/x] \vdash \Delta[\epsilon/x]}{\Gamma'; x : \top^* \vdash \Delta} \top^* L}$$

If the label x of the principal formula is replaced by some (other) label y, then we need a derivation of the sequent $\Gamma'[y/x]; y : \top^* \vdash \Delta[y/x]$. Using the $\top^* L$ rule, we have:

$$\dfrac{\Gamma'[y/x][\epsilon/y] \vdash \Delta[y/x][\epsilon/y]}{\Gamma'[y/x]; y : \top^* \vdash \Delta[y/x]} \top^* L$$

Note that the premise now is equal to $\Gamma'[\epsilon/x][\epsilon/y] \vdash \Delta[\epsilon/x][\epsilon/y]$, and can be proved using the induction hypothesis on Π_1.

If $y = \epsilon$, then Π' is obtained by applying Lemma 4.2.1 to Π_1.

(c) For the multiplicative rules that generate fresh labels in the premise ($*L$ and $-\!\!* R$), if the label of the principal formula is replaced by a label other than the newly created labels in the rules, then we proceed similarly as in the additive cases. If the label of the principal formula is replaced by one of the newly created labels, then we just need to create a different new label in the new relation.

For $*L$, we have the derivation:

$$\dfrac{\Pi_1}{\dfrac{(y, z \triangleright x); \Gamma'; y : A; z : B \vdash \Delta}{\Gamma'; x : A * B \vdash \Delta} *L}$$

If x is substituted by y (the case for substituting x to z is symmetric), then we need a derivation of $\Gamma'[y/x]; y : A * B \vdash \Delta[y/x]$. Note that the $*L$ rule requires the relation $(y, z \triangleright x)$ to be fresh, so in the original derivation y and z cannot be in Γ' or Δ. Therefore by the induction hypothesis we must have a derivation Π_1' for

$$(y', z' \triangleright x); \Gamma'; y' : A; z' : B \vdash \Delta,$$

where y' and z' are new labels, such that $ht(\Pi_1') \leq ht(\Pi_1)$. Applying the induction hypothesis again to Π_1', we have a derivation Π_1'' for $(y', z' \rhd y); \Gamma'[y/x]; y' : A; z' : B \vdash \Delta[y/x]$, with $ht(\Pi_1'') \leq ht(\Pi_1)$. Thus the derivation Π' is constructed as follows.

$$\dfrac{\begin{array}{c}\Pi_1'' \\ (y', z' \rhd y); \Gamma'[y/x]; y' : A; z' : B \vdash \Delta[y/x]\end{array}}{\Gamma'[y/x]; y : A * B \vdash \Delta[y/x]} *L$$

The case for $-\!\!* R$ is similar. suppose Π is:

$$\dfrac{\begin{array}{c}\Pi_1 \\ (y, x \rhd z); \Gamma; y : A \vdash z : B; \Delta'\end{array}}{\Gamma \vdash x : A -\!\!* B; \Delta'} -\!\!* R$$

If x is replaced by y, then we have the following derivation.

$$\dfrac{\begin{array}{c}\Pi_1' \\ (y', y \rhd z'); \Gamma[y/x]; y' : A \vdash z' : B; \Delta'[y/x]\end{array}}{\Gamma[y/x] \vdash y : A -\!\!* B; \Delta'[y/x]} -\!\!* R$$

If x is replaced by z, then we have the following derivation.

$$\dfrac{\begin{array}{c}\Pi_1' \\ (y', z \rhd z'); \Gamma[z/x]; y' : A \vdash z' : B; \Delta'[z/x]\end{array}}{\Gamma[z/x] \vdash z : A -\!\!* B; \Delta'[z/x]} -\!\!* R$$

□

Before we prove the admissibility of weakening, we show some intermediate lemmas about weakening of formulae and relational atoms. The next lemma shows that the labelled formula $\epsilon : \top^*$ in the antecedent of a sequent is not used in any derivation since there is no rule that can be applied to it, therefore this labelled formula can be removed without affecting provability.

Lemma 4.2.3. *If $\Gamma; \epsilon : \top^* \vdash \Delta$ is derivable, then $\Gamma \vdash \Delta$ is derivable with the same series of rule applications.*

Proof. By a straightforward induction on the height of derivation n.
(Base case) If $n = 0$, then $\Gamma; \epsilon : \top^* \vdash \Delta$ must be the conclusion of one of id, $\bot L$, $\top R$, $\top^* R$. Note that $\epsilon : \top^*$ in the antecedent cannot be the principal formula of any of those rules, therefore those rules are applicable to $\Gamma \vdash \Delta$ as well.
(Inductive case) If $n > 0$, consider the last rule in the derivation. It is obvious that $\epsilon : \top^*$ in the antecedent of a sequent cannot be the principal formula of any rules, therefore it has to appear in the premise(s) of the last rule application. Thus we can apply the induction hypothesis on those premise(s) and then use the corresponding rule to derive $\Gamma \vdash \Delta$. □

In general, if a formula is never principal in a derivation, it can obviously be omitted, thus we have the following lemma by the same argument:

Lemma 4.2.4. *If $w : A$ is not the principal formula of any rule application in the derivation of $\Gamma; w : A \vdash \Delta$ ($\Gamma \vdash w : A; \Delta$ resp.), then there is a derivation of $\Gamma \vdash \Delta$ with the same series of rule applications.*

For the same reason, we can replace a formula that is never used in a derivation by any structure, as stated below.

Lemma 4.2.5. *If $w : A$ is not the principal formula of any rule application (w may be changed in some rule applications with global substitutions) in the derivation of $\Gamma; w : A \vdash \Delta$ ($\Gamma \vdash w : A; \Delta$ resp.), then there is a derivation of $\Gamma; \Gamma' \vdash \Delta$ ($\Gamma \vdash \Delta'; \Delta$ resp.), and in the new derivation, the structure Γ (Δ resp.) is not altered except that certain labels in Γ (Δ resp.) are changed by rule applications with substitutions.*

Proof. By induction on the height of derivation n.
(Base case) If $n = 0$, since $w : A$ is not the principal formula, the substituted sequent is also the conlcusion of rules id, $\bot L$, $\top R$, $\top^* R$. This is the same as the base case of the proof for Lemma 4.2.3.
(Inductive case) If $n > 0$, consider the last rule in the derivation. Since $w : A$ is not the principal formula, for all rules except $\top^* L$, the original derivation has $w : A$ in the premise(s) of the last rule, therefore we can apply the induction hypothesis on the premise(s) and then use the rule to get the desired derivation.

For an example of $\wedge L$, suppose $w : A$ is in the antecedent, the original derivation is converted as follows.

$$\dfrac{\begin{array}{c}\Pi\\ \Gamma; w : A; x : B; x : C \vdash \Delta\end{array}}{\Gamma; w : A; x : B \wedge C \vdash \Delta}\,{\scriptstyle \wedge L} \quad \rightsquigarrow \quad \dfrac{\begin{array}{c}\Pi'\\ \Gamma; \Gamma'; x : B; x : C \vdash \Delta\end{array}}{\Gamma; \Gamma'; x : B \wedge C \vdash \Delta}\,{\scriptstyle \wedge L}$$

Other cases except $\top^* L$ are similar.
If the last rule is $\top^* L$, then we convert the derivation as follows.

$$\dfrac{\begin{array}{c}\Pi\\ \Gamma[\epsilon/w]; \epsilon : A \vdash \Delta[\epsilon/w]\end{array}}{\Gamma; w : A; w : \top^* \vdash \Delta}\,{\scriptstyle \top^* L} \quad \rightsquigarrow \quad \dfrac{\begin{array}{c}\Pi'\\ \Gamma[\epsilon/w]; \Gamma'[\epsilon/w] \vdash \Delta[\epsilon/w]\end{array}}{\Gamma; \Gamma'; w : \top^* \vdash \Delta}\,{\scriptstyle \top^* L}$$

We incorporate two steps here. First, by the induction hypothesis, we have a derivation of $\Gamma[\epsilon/w]; \Gamma' \vdash \Delta[\epsilon/w]$. Then by the Substitution Lemma 4.2.2, there is a derivation Π' of $\Gamma[\epsilon/w]; \Gamma'[\epsilon/w] \vdash \Delta[\epsilon/w]$, from which we can derive the final sequent.

Therefore the only change to Γ' in the new derivation is that some of its labels might be changed by the rules $\top^* L$, Eq_1, or Eq_2. □

With the above results, admissibility of weakening is proved by a simple induction on the height of derivations so we state the lemma without proof.

Lemma 4.2.6 (Weakening admissibility)**.** *If $\Gamma \vdash \Delta$ is derivable in LS_{BBI}, then for all structures Γ' and Δ', the sequent $\Gamma;\Gamma' \vdash \Delta;\Delta'$ is derivable with the same height in LS_{BBI}.*

Note 4.2.1. *The admissibility of general weakening shows that if $\Gamma \vdash \Delta$ is derivable, then $\Gamma;\Gamma' \vdash \Delta;\Delta'$ is derivable. A stronger argument here is that in the derivation of the latter sequent, Γ' and Δ' are never changed except that some labels might be changed by rule applications with substitutions. This is similar as in Lemma 4.2.5.*

Next we show that the inference rules in LS_{BBI} are *invertible*, in the sense that if the conclusion of a rule is derivable then the premises are also derivable[2].

Lemma 4.2.7 (Invertibility of rules)**.** *If Π is a cut-free LS_{BBI} derivation of the conclusion of a rule then there is a cut-free LS_{BBI} derivation for each premise, with height $\leq ht(\Pi)$.*

Proof. As the additive rules in LS_{BBI} are exactly the same as those in Negri's labelled system for modal logic or *G*3*c* (cf. [73]), the proof for them is similar. The main difference is that the rest of our rules are of different forms. However, as most of our rules do not modify the side structures, simply by applying the induction hypothesis and then using the corresponding rule, we get the new derivation. The cases where the last rule applied is $\top^* L$, Eq_1, or Eq_2 follow essentially the same, except a global substitution needs to be considered, but that is of no harm.

Rules E, A, U, A_C, $*R$ and $-\!\!* L$ are trivially invertible as the conclusion is a subset of the premise, and weakening is height-preserving admissible.

To prove the cases for $*L$ and $-\!\!* R$, we do inductions on the height n of the derivation. In each case below, it is obvious that each premise is always cut-free derivable with less or same height as the conclusion.

The case for $*L$ is as follows. Suppose the rule instance is as below:

$$\frac{(x, y \rhd z);\Gamma; x : A; y : B \vdash \Delta}{\Gamma; z : A * B \vdash \Delta}\ {}_{*L}$$

(Base case) If $n = 0$, then the conclusion of $*L$ is one of the conlucsions of id, $\bot L$, $\top R$, $\top^* R$, notice that the identity rule is restricted to atomic propositions, therefore the premise of $*L$ is also the conclusions of the corresponding axiom rule.
(Inductive case) If $n > 0$, and the last rule applied is not $*L$ or $-\!\!* R$, then no fresh labels are involved, so we can safely apply the induction hypothesis on the premise of the last rule and then use the rule to get the derivation. If the last rule is $*L$ or $-\!\!* R$, but the principal formula is in Γ or Δ, we proceed similarly, and use the Substitution Lemma 4.2.2 to ensure that the created labels are fresh. If the principal formula is $z : A * B$, then the premise of the last rule yields the desired conclusion.

The case for $-\!\!* R$ follows similarly.

For $\top^* L$, again, we do an induction on the height n of the derivation.

[2]This lemma is sometimes called the *inversion lemma* in the literature.

(Base case) If $n = 0$, then $\Gamma; x : \top^* \vdash \Delta$ is the conclusion of one of id, $\bot L$, $\top R$, $\top^* R$; and $x : \top^*$ cannot be the principal formula. Note that in the first three cases the principal formulae can be labelled with anything. Since, in the sequent $\Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]$, the label x is uniformly replaced by ϵ, this sequent can be the conclusion of the corresponding rule as well. For $\top^* R$, since $\top^*$ on the right hand side can only be labelled with ϵ, replacing x to ϵ does not change its label, thus this case is proved.
(Inductive case) If $n > 0$, consider the last rule applied in the derivation.

1. If the principal formula or relational atom does not involve the label x, then we can apply the induction hypothesis directly on the premise of the last rule, then use the last rule to get the derivation.

2. Otherwise, if the principal formula or relational atom has label x, and the last rule is not $\top^* L$, we proceed similarly, except replacing the label in the principal relational atom or formula. The detail is exemplified using $*L$.

   For $*L$, we have the following derivation:

$$\frac{\begin{array}{c}\Pi\\ (y, z \rhd x); \Gamma; x : \top^*; y : A; z : B \vdash \Delta\end{array}}{\Gamma; x : \top^*; x : A * B \vdash \Delta} \, {}^{*L}$$

   The condition of the rule $*L$ guarantees that y and z cannot be in Γ and Δ, so we do not have to worry if they are identical to x. By applying the induction hypothesis and then the $*L$ rule, we get the following derivation:

$$\frac{\begin{array}{c}\Pi'\\ (y, z \rhd \epsilon); \Gamma[\epsilon/x]; y : A; z : B \vdash \Delta[\epsilon/x]\end{array}}{\Gamma[\epsilon/x]; \epsilon : A * B \vdash \Delta[\epsilon/x]} \, {}^{*L}$$

   Another way to do this is by using the Substitution Lemma 4.2.2, replacing x by ϵ, we get a derivation to the premise that has a redundant $\epsilon : \top^*$, since we know that this labelled formula on the left hand side does not contribute to the derivation, we can safely derive the sequent without it using the same inference, cf. Lemma 4.2.4.

   The case where the last rule is $-\!\!* R$ is similar.

   If the last rule is Eq_1, we consider the following cases:

   (a) The label of $\top^*$ is not in the principal relational atom (i.e., $x \neq w$ and $x \neq w'$). The original derivation is as follows.

$$\frac{\begin{array}{c}\Pi\\ (\epsilon, w \rhd w); \Gamma[w/w']; x : \top^* \vdash \Delta[w/w']\end{array}}{(\epsilon, w' \rhd w); \Gamma; x : \top^* \vdash \Delta} \, {}^{Eq_1}$$

By the induction hypothesis, we have the following derivation:

$$\begin{array}{c}\Pi' \\ (\epsilon, w \triangleright w); \Gamma[w/w'][\epsilon/x] \vdash \Delta[w/w'][\epsilon/x]\end{array}$$

Note that since x, w, w' are all different, the end sequent is equal to the following:

$$(\epsilon, w \triangleright w); \Gamma[\epsilon/x][w/w'] \vdash \Delta[\epsilon/x][w/w']$$

from which we can use the rule Eq_1 and derive $(\epsilon, w' \triangleright w); \Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]$.

(b) $x = w$. The original derivation is as follows.

$$\dfrac{\begin{array}{c}\Pi \\ (\epsilon, x \triangleright x); \Gamma[x/w']; x : \top^* \vdash \Delta[x/w']\end{array}}{(\epsilon, w' \triangleright x); \Gamma; x : \top^* \vdash \Delta}\, Eq_1$$

By the Substitution Lemma, replacing every x by ϵ in the premise of the last rule, we get the following derivation:

$$\begin{array}{c}\Pi' \\ (\epsilon, \epsilon \triangleright \epsilon); \Gamma[x/w'][\epsilon/x]; \epsilon : \top^* \vdash \Delta[x/w'][\epsilon/x]\end{array}$$

The end sequent is equal to:

$$(\epsilon, \epsilon \triangleright \epsilon); \Gamma[\epsilon/x][\epsilon/w']; \epsilon : \top^* \vdash \Delta[\epsilon/x][\epsilon/w']$$

By Lemma 4.2.3, $\epsilon : \top^*$ in the antecedent can be omitted. Apply the Eq_1 rule on this sequent without $\epsilon : \top^*$, we finally get $(\epsilon, w' \triangleright \epsilon); \Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]$.

(c) $x = w'$. The original derivation is as follows.

$$\dfrac{\begin{array}{c}\Pi \\ (\epsilon, w \triangleright w); \Gamma[w/x]; w : \top^* \vdash \Delta[w/x]\end{array}}{(\epsilon, x \triangleright w); \Gamma; x : \top^* \vdash \Delta}\, Eq_1$$

By the induction hypothesis, we have the following derivation:

$$\begin{array}{c}\Pi' \\ (\epsilon, \epsilon \triangleright \epsilon); \Gamma[w/x][\epsilon/w] \vdash \Delta[w/x][\epsilon/w]\end{array}$$

Now the end sequent is equal to:

$$(\epsilon, \epsilon \triangleright \epsilon); \Gamma[\epsilon/x][\epsilon/w] \vdash \Delta[\epsilon/x][\epsilon/w]$$

By using the rule Eq_2 on this sequent, we derive $(\epsilon, \epsilon \triangleright w); \Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]$.

The case where the last rule is Eq_2 is similar to the case for Eq_1.

If the last rule is $\top^* L$, then the derivation to the premise of the last rule yields the new derivation.

The invertibility of Eq_1 and Eq_2 follows from the Substitution Lemma, as the reverse versions of these two rules are only about globally replacing labels. □

The proof of the admissibility of contraction on additive formulae is similar to that for classical sequent calculus since the LS_{BBI} rules for these connectives are the same. In the multiplicative rules, the principal formula is retained in the premise, so admissibility of contraction on multiplicative formulae follows trivially. We need to prove that contraction on relational atoms is admissible, as stated in the next lemma.

Lemma 4.2.8. *For any structures* Γ, Δ*, and relational atom* $(x, y \triangleright z)$*: if* Π *is a cut-free* LS_{BBI} *derivation of* $(x, y \triangleright z); (x, y \triangleright z); \Gamma \vdash \Delta$*, then there is a cut-free* LS_{BBI} *derivation* Π' *of* $(x, y \triangleright z); \Gamma \vdash \Delta$ *with* $ht(\Pi') \leq ht(\Pi)$.

Proof. Let $n = ht(\Pi)$. The proof is by induction on n. Most structural rules only have one principal relational atom, so it is easy to show that contraction can permute through them upwards.
(Base case) If $n = 0$, then the rule must be one of id, $\bot L$, $\top R$, and $\top^* R$. All of those rules are independent of relational atoms, so the case with only one presence of contracted relational atom holds as well.

(Inductive case) If $n > 0$, consider the bottom-most (last) rule.

If the last rule does not involve relations ($\top^* L$, $\wedge L$, $\wedge R$, $\to L$, $\to R$), we use the induction hypothesis on the premise(s) of this rule and then use that same rule to derive the sequent with only one relational atom. For each multiplicative connective rule, the situation is similar, as the relevant relational atom appears in the premises of these rules.

The structural rules with only one principal relational atom are easily proved to be closed under contraction.

The case for the rule A needs more care, as it involves two principal relational atoms. If the two principal relational atoms are different, then the admissibility of contraction follows similarly as above. But if the principal relational atoms are identical, the situation is a bit tricky:

$$\frac{\begin{array}{c}\Pi\\ (x, w \triangleright x); (y, y \triangleright w); (x, y \triangleright x); (x, y \triangleright x); \Gamma \vdash \Delta\end{array}}{(x, y \triangleright x); (x, y \triangleright x); \Gamma \vdash \Delta}\, A$$

There is no obvious way to make this case admissible, and this is the reason we have a special case of the rule A, namely A_C. In the rule A_C, contraction is absorbed so that there is only one principal relational atom. The new derivation is as follows.

$$\frac{\begin{array}{c}\Pi'\\ (x, w \triangleright x); (y, y \triangleright w); (x, y \triangleright x); \Gamma \vdash \Delta\end{array}}{(x, y \triangleright x); \Gamma \vdash \Delta}\, A_C$$

For Eq_1 and Eq_2, as the principal relational atom is carried to the premise (although some labels may be changed), so admissibility of contraction on those relational atoms is obvious. □

The admissibility of contraction on formulae is straightforward. Most of the cases are analogous to the ones in Negri's labelled calculus for modal logic [72].

Lemma 4.2.9. *For all structures Γ, Δ, and labelled formula $w : A$, the following holds in LS_{BBI}:*

1. *If there is a cut-free derivation Π of $\Gamma; w : A; w : A \vdash \Delta$, then there is a cut-free derivation Π' of $\Gamma; w : A \vdash \Delta$ with $ht(\Pi') \leq ht(\Pi)$.*

2. *If there is a cut-free derivation Π of $\Gamma \vdash w : A; w : A; \Delta$, then there is a cut-free derivation Π' of $\Gamma \vdash w : A; \Delta$ with $ht(\Pi') \leq ht(\Pi)$.*

Proof. By simultaneous induction on the height of derivations for the left and right contraction. Let $n = ht(\Pi)$.
(Base case) If $n = 0$, the premise is one of the conclusions of id, $\bot L$, $\top R$ and $\top^* R$, then the contracted sequent is also the conclusion of the corresponding rules.
(Inductive case) If $n > 0$, consider the last rule applied to the premise.

(i) If the contracted formula is not principal in the last rule, then we can apply the induction hypothesis on the premise(s) of the last rule, then use the same rule to get the derivation.

(ii) If the contracted formula is the principal formula of the last rule, there are several cases. For the additive rules the cases are reduced to contraction on smaller formulae. The resultant sequent has a shorter derivation tree, so we can use the induction hypothesis to obtain contraction admissibility. See [73] for details.

For $\top^* L$, we have the following derivation:

$$\dfrac{\begin{array}{c}\Pi \\ \Gamma[\epsilon/x]; \epsilon : \top^* \vdash \Delta[\epsilon/x]\end{array}}{\Gamma; x : \top^*; x : \top^* \vdash \Delta}\ \top^* L$$

Note that the only case where $\top^*$ is useful on the left hand side is when it is labelled with a world other than ϵ. Since the substitution $[\epsilon/\epsilon]$ does not do anything to the sequent, Π can also be the derivation for $\Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]$, cf. Lemma 4.2.3, which leads to $\Gamma; x : \top^* \vdash \Delta$.

For $*R$ and $-\!\!* L$, we can apply the induction hypothesis directly on the premise of the corresponding rule since the rules carry the principal formula into the premise(s).

For $*L$, we have a derivation as follows.

$$\dfrac{\begin{array}{c}\Pi \\ (x, y \rhd z); \Gamma; z : A * B; x : A; y : B \vdash \Delta\end{array}}{\Gamma; z : A * B; z : A * B \vdash \Delta}\ *L$$

Apply the Invertibility Lemma 4.2.7 on the premise of $*L$, we have:

$$\begin{array}{c}\Pi' \\ (x, y \rhd z); (x', y' \rhd z); \Gamma; x' : A; y' : B; x : A; y : B \vdash \Delta\end{array}$$

The Substitution Lemma 4.2.2 yields a derivation for

$$(x, y \triangleright z); (x, y \triangleright z); \Gamma; x : A; y : B; x : A; y : B \vdash \Delta.$$

Apply the induction hypothesis twice and admissibility of contraction on relational atoms on this sequent, we get a derivation for $(x, y \triangleright z); \Gamma; x : A; y : B \vdash \Delta$. Then apply $*L$ on this sequent to get $\Gamma; z : A * B \vdash \Delta$.

The case for $-\!\!* R$ follows similarly. We have a derivation as follows.

$$\frac{\begin{array}{c}\Pi\\ (x, y \triangleright z); \Gamma; x : A \vdash z : B; y : A -\!\!* B; \Delta\end{array}}{\Gamma \vdash y : A -\!\!* B; y : A -\!\!* B; \Delta} -\!\!* R$$

The Invertibility of $-\!\!* R$ in the premise yields:

$$\begin{array}{c}\Pi\\ (x, y \triangleright z); (x', y \triangleright z'); \Gamma; x : A; x' : A \vdash z : B; z' : B; \Delta\end{array}$$

We obtain $(x, y \triangleright z); (x, y \triangleright z); \Gamma; x : A; x : A \vdash z : B; z : B; \Delta$ by the Substitution Lemma 4.2.2. Apply the induction hypothesis twice, and the admissibility of contraction on relations (Lemma 4.2.8) on this sequent to get $(x, y \triangleright z); \Gamma; x : A \vdash z : B; \Delta$. Finally, apply $-\!\!* R$ to derive $\Gamma \vdash y : A -\!\!* B; \Delta$ in the nth step. □

We can now state the admissibility of contraction in LS_{BBI}, which is a direct result of the above lemmas.

Lemma 4.2.10 (Contraction admissibility)**.** *If* $\Gamma; \Gamma \vdash \Delta; \Delta$ *is derivable in* LS_{BBI}*, then* $\Gamma \vdash \Delta$ *is derivable with the same height in* LS_{BBI}*.*

Cut Elimination Theorem

We define the *cut rank* of an application of the *cut* rule as the pair $(|f|, ht(\Pi_1) + ht(\Pi_2))$, where $|f|$ denotes the size of the cut formula (i.e., the number of connectives in the formula), and $ht(\Pi_1)$, $ht(\Pi_2)$ are the heights of the derivations above the *cut* rule, the sum of them is call the *cut height*. Cut ranks are ordered lexicographically, where each component of the ranks is ordered according to the ordering $>$ on $\mathcal{N}$.

Theorem 4.2.11 (Cut-admissibility)**.** *If* $\Gamma \vdash \Delta$ *is derivable in* LS_{BBI}*, then it is also derivable without using the cut rule.*

Proof. By induction on the cut ranks of the proof in LS_{BBI}. We show that each application of *cut* can either be eliminated, or be replaced by one or more *cut* rules of smaller cut ranks. The argument for termination is similar to the cut-elimination proof for *G3ip* [73]. We start to eliminate the topmost *cut* first, and repeat this procedure until there is no *cut* in the derivation. We first show that *cut* can be eliminated when the *cut height* is the lowest, i.e., at least one premise is of height 1. Then we show that the *cut height* is reduced in all cases in which the cut formula is not principal in both premises

of cut. If the cut formula is principal in both premises, then the *cut* is reduced to one or more *cut*s on smaller formulae or shorter derivations. Since atoms cannot be principal in logical rules, finally we can either reduce all *cut*s to the case where the cut formula is not principal in both premises, or reduce those *cut*s on compound formulae until their *cut height*s are minimal and then eliminate those *cut*s.
(Base case) If at least one premise of the *cut* rule is *id*, $\bot L$, $\top R$, or $\top^* R$, we consider the following cases:

1. The left premise of *cut* is an application of *id*, and the cut formula is not principal, then the derivation is transformed as follows.

$$\dfrac{\dfrac{}{\Gamma; y : B \vdash y : B; x : A; \Delta}\,id \qquad \begin{matrix}\Pi\\ \Gamma'; x : A \vdash \Delta'\end{matrix}}{\Gamma; \Gamma'; y : B \vdash y : B; \Delta; \Delta'}\,cut \rightsquigarrow$$

$$\dfrac{}{\Gamma; \Gamma'; y : B \vdash y : B; \Delta; \Delta'}\,id$$

   The same transformation works for $\bot L$, $\top R$, $\top^* R$ in this case.

2. The left premise of *cut* is an application of *id*, and the cut formula is principal, then the derivation is transformed as follows.

$$\dfrac{\dfrac{}{\Gamma; x : A \vdash x : A; \Delta}\,id \qquad \begin{matrix}\Pi\\ \Gamma'; x : A \vdash \Delta'\end{matrix}}{\Gamma; \Gamma'; x : A \vdash \Delta; \Delta'}\,cut \rightsquigarrow$$

$$\dfrac{\begin{matrix}\Pi\\ \Gamma'; x : A \vdash \Delta'\end{matrix}}{\Gamma; \Gamma'; x : A \vdash \Delta; \Delta'}\,\text{Weakening Lemma 4.2.6}$$

3. The left premise of *cut* is an application of $\top R$, and the cut formula is principal, then the derivation is transformed as follows.

$$\dfrac{\dfrac{}{\Gamma \vdash x : \top; \Delta}\,\top R \qquad \begin{matrix}\Pi\\ \Gamma'; x : \top \vdash \Delta'\end{matrix}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\,cut \rightsquigarrow$$

   As $x : \top$ cannot be a principal formula in the antecedent, by Weakening Lemma 4.2.4 there is a derivation Π' of $\Gamma' \vdash \Delta'$. The above derivation is transformed to:

$$\dfrac{\begin{matrix}\Pi'\\ \Gamma' \vdash \Delta'\end{matrix}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\,\text{Weakening Lemma 4.2.6}$$

   The same holds for $\top^* R$.

4. The right premise of *cut* is an application of *id*, $\bot L$, $\top R$ or $\top^* R$, and the cut formula is not principal. This case is similar to case 1.

5. The right premise of *cut* is an application of *id*, and the cut formula is principal. This case is similar to case 2.

6. The right premise of *cut* is an application of $\bot L$, and the cut formula is principal. This case is similar to case 3.

(Inductive case) If both premises are not in one of the base cases, we distinguish three cases here: the cut formula is not principal in the left premises; the cut formula is only principal in the left premise; and the cut formula is principal in both premises.

1. The cut formula is not principal in the left premise. Suppose the left premise ends with a rule r. We permute the *cut* instance up in the derivation tree to reduce the cut height.

   (a) If r is $\top^* L$, w.l.o.g. we assume the label of the principal formula is y (which might be equal to x). The original derivation is as follows.

   $$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \Gamma[\epsilon/y] \vdash x : A; \Delta[\epsilon/y]\end{matrix}}{\Gamma; y : \top^* \vdash x : A; \Delta}\,\top^* L \qquad \begin{matrix}\Pi_2\\ \Gamma'; x : A \vdash \Delta'\end{matrix}}{\Gamma; \Gamma'; y : \top^* \vdash \Delta; \Delta'}\,cut$$

   By the Substitution Lemma 4.2.2, there is a derivation Π_2' of $\Gamma'[\epsilon/y]; x : A \vdash \Delta'[\epsilon/y]$. Thus we can transform the derivation into the following:

   $$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \Gamma[\epsilon/y] \vdash x : A; \Delta[\epsilon/y]\end{matrix} \qquad \begin{matrix}\Pi_2'\\ \Gamma'[\epsilon/y]; x : A \vdash \Delta'[\epsilon/y]\end{matrix}}{\Gamma[\epsilon/y]; \Gamma'[\epsilon/y] \vdash \Delta[\epsilon/y]; \Delta'[\epsilon/y]}\,cut}{\Gamma; \Gamma'; y : \top^* \vdash \Delta; \Delta'}\,\top^* L$$

   If $x = y$ in the original derivation, then the new derivation cuts on $\epsilon : A$ instead. As substitution is height preserving, the cut height in this case is reduced as well.

   (b) If r is Eq_1, and the label x of the principal formula is not equal to w', the original derivation is as follows.

   $$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ (\epsilon, w \rhd w); \Gamma[w/w'] \vdash x : A; \Delta[w/w']\end{matrix}}{(\epsilon, w' \rhd w); \Gamma \vdash x : A; \Delta}\,Eq_1 \qquad \begin{matrix}\Pi_2\\ \Gamma'; x : A \vdash \Delta'\end{matrix}}{(\epsilon, w' \rhd w); \Gamma; \Gamma' \vdash \Delta; \Delta'}\,cut$$

   This *cut* is reduced in the same way as the $\top^* L$ case, where we get Π_2' from the Substitution Lemma 4.2.2:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ (\epsilon, w \rhd w); \Gamma[w/w'] \vdash x : A; \Delta[w/w']\end{array} \qquad \begin{array}{c}\Pi_2'\\ \Gamma'[w/w']; x : A \vdash \Delta'[w/w']\end{array}}{(\epsilon, w \rhd w); \Gamma[w/w']; \Gamma'[w/w'] \vdash \Delta[w/w']; \Delta'[w/w']}\, cut}{(\epsilon, w' \rhd w); \Gamma; \Gamma' \vdash \Delta; \Delta'}\, Eq_1$$

If $x = w'$, then we cut on $w : A$ instead in the reduced version.

(c) If r is Eq_2, the procedure follows similarly as the case for Eq_1 above.

(d) If r is a unary inference except for $\top^* L$, Eq_1, and Eq_2, then the original derivation is as follows.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \Gamma_1 \vdash x : A; \Delta_1\end{array}}{\Gamma \vdash x : A; \Delta}\, r \qquad \begin{array}{c}\Pi_2\\ \Gamma'; x : A \vdash \Delta'\end{array}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\, cut$$

Then we can permute the application of *cut* upwards as follows.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \Gamma_1 \vdash x : A; \Delta_1\end{array} \qquad \begin{array}{c}\Pi_2\\ \Gamma'; x : A \vdash \Delta'\end{array}}{\Gamma_1; \Gamma' \vdash \Delta_1; \Delta'}\, cut}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\, r$$

Note that as all our rules except $\top^* L$, Eq_1, and Eq_2 do not modify side structures, Γ' and Δ' in the premise of r are not changed. The cut rank of the original *cut* is $(|x : A|, |\Pi_1| + 1 + |\Pi_2|)$, whereas the cut rank of the new *cut* is $(|x : A|, |\Pi_1| + |\Pi_2|)$, so the *cut height* reduces.

(e) If r is a binary inference, we can transform the derivation similarly.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \Gamma_1 \vdash x : A; \Delta_1\end{array} \qquad \begin{array}{c}\Pi_2\\ \Gamma_2 \vdash x : A; \Delta_2\end{array}}{\Gamma \vdash x : A; \Delta}\, r \qquad \begin{array}{c}\Pi_3\\ \Gamma'; x : A \vdash \Delta'\end{array}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\, cut \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \Gamma_1 \vdash x : A; \Delta_1\end{array} \qquad \begin{array}{c}\Pi_3\\ \Gamma'; x : A \vdash \Delta'\end{array}}{\Gamma_1; \Gamma' \vdash \Delta_1; \Delta'}\, cut \qquad \dfrac{\begin{array}{c}\Pi_2\\ \Gamma_2 \vdash x : A; \Delta_2\end{array} \qquad \begin{array}{c}\Pi_3\\ \Gamma'; x : A \vdash \Delta'\end{array}}{\Gamma_2; \Gamma' \vdash \Delta_2; \Delta'}\, cut}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\, r$$

The cut rank of the original *cut* is $(|x : A|, max(|\Pi_1|, |\Pi_2|) + 1 + |\Pi_3|)$, and that of the two new *cut*s are $(|x : A|, |\Pi_1| + |\Pi_3|)$ and $(|x : A|, |\Pi_2| + |\Pi_3|)$ respectively. Thus the cut heights are reduced.

2. The cut formula is only principal in the left premise. We only consider the last rule in the right branch. The proof of this case is symmetric to that in Case 1. The cut height is reduced.

3. The cut formula is principal in both premises. We do a case analysis on the main connective of the cut formula. If the main connective is additive, then there is no need to substitute any labels.

   For $\wedge$,

$$\dfrac{\dfrac{\dfrac{\Pi_1}{\Gamma \vdash x : A; \Delta} \quad \dfrac{\Pi_2}{\Gamma \vdash x : B; \Delta}}{\Gamma \vdash x : A \wedge B; \Delta}\,{\wedge R} \quad \dfrac{\dfrac{\Pi_3}{\Gamma'; x : A; x : B \vdash \Delta'}}{\Gamma'; x : A \wedge B \vdash \Delta'}\,{\wedge L}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\,{cut} \rightsquigarrow$$

$$\dfrac{\dfrac{\Pi_1}{\Gamma \vdash x : A; \Delta} \quad \dfrac{\dfrac{\Pi_2}{\Gamma \vdash x : B; \Delta} \quad \dfrac{\Pi_3}{\Gamma'; x : A; x : B \vdash \Delta'}}{\Gamma; \Gamma'; x : A \vdash \Delta; \Delta'}\,{cut}}{\Gamma; \Gamma; \Gamma' \vdash \Delta; \Delta; \Delta'}\,{cut}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\ \text{Contraction Lemma 4.2.10}$$

   For $\rightarrow$,

$$\dfrac{\dfrac{\dfrac{\Pi_1}{\Gamma'; x : A \vdash x : B; \Delta'}}{\Gamma' \vdash x : A \rightarrow B; \Delta'}\,{\rightarrow R} \quad \dfrac{\dfrac{\Pi_2}{\Gamma \vdash x : A; \Delta} \quad \dfrac{\Pi_3}{\Gamma; x : B \vdash \Delta}}{\Gamma; x : A \rightarrow B \vdash \Delta}\,{\rightarrow L}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\,{cut} \rightsquigarrow$$

$$\dfrac{\dfrac{\Pi_2}{\Gamma \vdash x : A; \Delta} \quad \dfrac{\dfrac{\Pi_1}{\Gamma'; x : A \vdash x : B; \Delta'} \quad \dfrac{\Pi_3}{\Gamma; x : B \vdash \Delta}}{\Gamma; \Gamma'; x : A \vdash \Delta; \Delta'}\,{cut}}{\Gamma; \Gamma; \Gamma' \vdash \Delta; \Delta; \Delta'}\,{cut}}{\Gamma; \Gamma' \vdash \Delta; \Delta'}\ \text{Contraction Lemma 4.2.10}$$

   For both $\wedge$ and $\rightarrow$, *cut* is reduced to applications on smaller formulae, therefore the cut rank of the *cut* reduces. The cases for $\vee$ rules are symmetric to $\wedge$ rules.

   There is an asymmetry in the rules for $\top^*$. That is, the left rule for $\top^*$ requires that the label w of $\top^*$ cannot be ϵ, whereas the right rule for $\top^*$ restricts the label of $\top^*$ to be ϵ only. As a consequence, when the cut formula is $\top^*$, it cannot be the principal formula of both premises at the same time. Therefore the cases for $\top^*$ are covered in the proof above.

   When the main connective of the cut formula is $*$ or $-\!\!*$, the case is more complicated. For $*$, we have the following two derivations as the premises of *cut*:

$$\dfrac{\dfrac{\Pi_1}{(x, y \triangleright z); \Gamma \vdash x : A; z : A * B; \Delta} \quad \dfrac{\Pi_2}{(x, y \triangleright z); \Gamma \vdash y : B; z : A * B; \Delta}}{(x, y \triangleright z); \Gamma \vdash z : A * B; \Delta}\,{*R}$$

and

$$\dfrac{\begin{array}{c}\Pi_3\\ (x', y' \rhd z); \Gamma'; x' : A; y' : B \vdash \Delta'\end{array}}{\Gamma'; z : A * B \vdash \Delta'} *L$$

And the *cut* rule gives the end sequent $(x, y \rhd z); \Gamma; \Gamma' \vdash \Delta; \Delta'$. The cut rank of this *cut* is $(|A * B|, 1 + |\Pi_3| + 1 + max(|\Pi_1|, |\Pi_2|))$.

We use several *cut*s with smaller ranks to derive $(x, y \rhd z); \Gamma; \Gamma' \vdash \Delta; \Delta'$ as follows.

Firstly,

$$\dfrac{\begin{array}{c}\Pi_1\\ (x, y \rhd z); \Gamma \vdash x : A; z : A * B; \Delta\end{array} \qquad \dfrac{\begin{array}{c}\Pi_3\\ (x', y' \rhd z); \Gamma'; x' : A; y' : B \vdash \Delta'\end{array}}{\Gamma'; z : A * B \vdash \Delta'} *L}{(x, y \rhd z); \Gamma; \Gamma' \vdash x : A; \Delta; \Delta'} cut$$

The cut rank of this *cut* is $(|A * B|, |\Pi_1| + |\Pi_3| + 1))$, thus is less than that of the original *cut*.

The second *cut* works similarly.

$$\dfrac{\begin{array}{c}\Pi_2\\ (x, y \rhd z); \Gamma \vdash y : B; z : A * B; \Delta\end{array} \qquad \dfrac{\begin{array}{c}\Pi_3\\ (x', y' \rhd z); \Gamma'; x' : A; y' : B \vdash \Delta'\end{array}}{\Gamma'; z : A * B \vdash \Delta'} *L}{(x, y \rhd z); \Gamma; \Gamma' \vdash y : B; \Delta; \Delta'} cut$$

The third *cut* works on a smaller formula.

$$\dfrac{\begin{array}{c}\Pi'_3\\ (x, y \rhd z); \Gamma; \Gamma' \vdash x : A; \Delta; \Delta' \qquad (x, y \rhd z); \Gamma'; x : A; y : B \vdash \Delta'\end{array}}{(x, y \rhd z); (x, y \rhd z); \Gamma; \Gamma'; \Gamma'; y : B \vdash \Delta; \Delta'; \Delta'} cut$$

The cut formula is $x : A$, thus the cut rank of this *cut* is less regardless of the height of the derivations.

Note that in the Π_3 branch, the $*L$ rule requires that the relation $(x', y' \rhd z)$ is newly created, so x' and y' cannot be ϵ and they cannot be in Γ' or Δ'. Therefore we are allowed to use the Substitution Lemma 4.2.2 to get a derivation Π'_3 of $(x, y \rhd z); \Gamma'; x : A; y : B \vdash \Delta'$ by just substituting x' for x and y' for y.

Finally we cut on another smaller formula $y : B$.

$$\dfrac{\begin{array}{c}(x, y \rhd z); \Gamma; \Gamma' \vdash y : B; \Delta; \Delta' \qquad (x, y \rhd z); (x, y \rhd z); \Gamma; \Gamma'; \Gamma'; y : B \vdash \Delta; \Delta'; \Delta'\end{array}}{(x, y \rhd z); (x, y \rhd z); (x, y \rhd z); \Gamma; \Gamma; \Gamma'; \Gamma'; \Gamma' \vdash \Delta; \Delta; \Delta'; \Delta'; \Delta'} cut$$

The cut rank of this *cut* is less than the original *cut*. We then apply the admissibility of contraction (Lemma 4.2.10) to derive $(x, y \triangleright z); \Gamma; \Gamma' \vdash \Delta; \Delta'$.

The case for $-\!\!*$ is similar. The two premises in the original *cut* are as follows.

$$\dfrac{\begin{array}{c}\Pi_1\\ (x', y \triangleright z'); \Gamma'; x' : A \vdash z' : B; \Delta'\end{array}}{\Gamma' \vdash y : A -\!\!* B; \Delta'} -\!\!* R$$

and

$$\dfrac{\begin{array}{cc}\Pi_2 & \Pi_3\\ (x, y \triangleright z); \Gamma; y : A -\!\!* B \vdash x : A; \Delta & (x, y \triangleright z); \Gamma; y : A -\!\!* B; z : B \vdash \Delta\end{array}}{(x, y \triangleright z); \Gamma; y : A -\!\!* B \vdash \Delta} -\!\!* L$$

And the *cut* rule yields the end sequent $(x, y \triangleright z); \Gamma; \Gamma' \vdash \Delta; \Delta'$. We use two cuts on the same formula, but with smaller derivation height.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ (x', y \triangleright z'); \Gamma'; x' : A \vdash z' : B; \Delta'\end{array}}{\Gamma' \vdash y : A -\!\!* B; \Delta'} -\!\!* R \qquad \begin{array}{c}\Pi_2\\ (x, y \triangleright z); \Gamma; y : A -\!\!* B \vdash x : A; \Delta\end{array}}{(x, y \triangleright z); \Gamma; \Gamma' \vdash x : A; \Delta; \Delta'} cut$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ (x', y \triangleright z'); \Gamma'; x' : A \vdash z' : B; \Delta'\end{array}}{\Gamma' \vdash y : A -\!\!* B; \Delta'} -\!\!* R \qquad \begin{array}{c}\Pi_3\\ (x, y \triangleright z); \Gamma; y : A -\!\!* B; z : B \vdash \Delta\end{array}}{(x, y \triangleright z); \Gamma; \Gamma'; z : B \vdash \Delta; \Delta'} cut$$

Then we cut on a smaller formula $x : A$.

$$\dfrac{\begin{array}{cc}\ & \Pi_1'\\ (x, y \triangleright z); \Gamma; \Gamma' \vdash x : A; \Delta; \Delta' & (x, y \triangleright z); \Gamma'; x : A \vdash z : B; \Delta'\end{array}}{(x, y \triangleright z); (x, y \triangleright z); \Gamma; \Gamma'; \Gamma' \vdash z : B; \Delta; \Delta'; \Delta'} cut$$

Again, in the original derivation, x' and z' are fresh in the premise of $-\!\!* R$ rule, thus by the Substitution Lemma 4.2.2 we can have a derivation Π_1' of the sequent $(x, y \triangleright z); \Gamma'; x : A \vdash z : B; \Delta'$, with x' substituted to x and z' substituted to z.

Then we cut on $z : B$.

$$\dfrac{\begin{array}{cc}(x, y \triangleright z); (x, y \triangleright z); \Gamma; \Gamma'; \Gamma' \vdash z : B; \Delta; \Delta'; \Delta' & (x, y \triangleright z); \Gamma; \Gamma'; z : B \vdash \Delta; \Delta'\end{array}}{(x, y \triangleright z); (x, y \triangleright z); (x, y \triangleright z); \Gamma; \Gamma; \Gamma'; \Gamma'; \Gamma' \vdash \Delta; \Delta; \Delta'; \Delta'; \Delta'} cut$$

In the end we use Lemma 4.2.10 (admissibility of contraction) to obtain the required sequent $(x, y \triangleright z); \Gamma; \Gamma' \vdash \Delta; \Delta'$. □

4.3 Localising Structural Rules

As a first step towards designing an effective proof search procedure for LS_{BBI}, we need to restrict the use of structural rules so that their applications in proof search are controlled by logical rules, thereby reducing the non-determinism in proof search. From now on we will assume that the sequents in LS_{BBI} consists of **sets** rather than **multisets**, since we have shown the admissibility of contraction for LS_{BBI}.

Suppose r_1, r_2 are two rule applications in a derivation where r_1 is directly below r_2. We say r_1 can permute upwards over r_2 when we can switch the order of these two rule applications. That is, in backwards proof search, we can first apply r_2 then apply r_1 on all premises of r_2 to obtain a new derivation.

We note the fact that the structural rule applications in LS_{BBI} can permute upwards over all other rule applications except for id, $\top^* R$, $*R$, and $-\!\!* L$. We refer to these four rules as *positive rules*, and refer to the other logical rules in LS_{BBI} as *negative rules*. The main reason is, all negative rules do not rely on relational atoms.

Lemma 4.3.1. *In any LS_{BBI} derivation, the structural rules can permute upwards through negative rules.*

Proof. To prove this lemma, we need to show that if a derivation involves structural rules, we can always apply them (in the same order) exactly below $*R$ and $-\!\!* L$, or below zero-premise rules. We show this by an induction on the height of the derivation. Since we do not permute structural rules through zero-premise rules, the proof in the base case and the inductive step are essentially the same. Assuming the lemma holds up to any derivation of height $n-1$, consider a derivation of height n.

1. Permute the application of Eq_1 or Eq_2 through non-zero-premise logical rules except for $*R$ and $-\!\!* L$.

   (a) Permute Eq_1/Eq_2 through additive logical rules is trivial. This is exemplified by $\wedge L$, assuming the label of the principal formula is modified by the Eq_2 application. The case for Eq_1 being used is similar. Suppose the original derivation is as follows.

$$\dfrac{\dfrac{\begin{matrix}\Pi\\ (\epsilon, w' \rhd w'); \Gamma[w'/w]; w' : A; w' : B \vdash \Delta[w'/w]\end{matrix}}{(\epsilon, w' \rhd w'); \Gamma[w'/w]; w' : A \wedge B \vdash \Delta[w'/w]}\,\wedge L}{(\epsilon, w' \rhd w); \Gamma; w : A \wedge B \vdash \Delta}\,Eq_2$$

   The derivation is changed to the following:

$$\dfrac{\dfrac{\begin{matrix}\Pi\\ (\epsilon, w' \rhd w'); \Gamma[w'/w]; w' : A; w' : B \vdash \Delta[w'/w]\end{matrix}}{(\epsilon, w' \rhd w); \Gamma; w : A; w : B \vdash \Delta}\,Eq_2}{(\epsilon, w' \rhd w); \Gamma; w : A \wedge B \vdash \Delta}\,\wedge L$$

We give another example using Eq_2 that is applied below $\wedge R$. The original derivation runs as below:

$$\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ (\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash w' : A; \Delta[w'/w] & (\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash w' : B; \Delta[w'/w]\end{array}}{(\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash w' : A \wedge B; \Delta[w'/w]}\,\wedge R}{(\epsilon, w' \rhd w); \Gamma \vdash w : A \wedge B; \Delta}\,Eq_2$$

We can use the following derivation instead:

$$\dfrac{\dfrac{\Pi_1}{(\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash w' : A; \Delta[w'/w]}\,Eq_2 \quad \dfrac{\Pi_2}{(\epsilon, w' \rhd w'); \Gamma[w'/w] \vdash w' : B; \Delta[w'/w]}\,Eq_2}{(\epsilon, w' \rhd w); \Gamma \vdash w : A; \Delta \qquad (\epsilon, w' \rhd w); \Gamma \vdash w : B; \Delta}\,\wedge R$$

$$(\epsilon, w' \rhd w); \Gamma \vdash w : A \wedge B; \Delta$$

The cases for $\rightarrow L$ and $\rightarrow R$ are similar.

(b) Permute Eq_1/Eq_2 through $\top^* L$, assuming the label of the principal formula is w. Here we give a derivation for the case of Eq_1 is as follows. The case for Eq_2 is similar.

$$\dfrac{\dfrac{\dfrac{\Pi}{(\epsilon, \epsilon \rhd \epsilon); \Gamma[w/w'][\epsilon/w] \vdash \Delta[w/w'][\epsilon/w]}\,\top^* L}{(\epsilon, w \rhd w); \Gamma[w/w']; w : \top^* \vdash \Delta[w/w']}\,Eq_1}{(\epsilon, w' \rhd w); \Gamma; w' : \top^* \vdash \Delta}$$

We modify the derivation as follows.

$$\dfrac{\dfrac{\dfrac{\Pi}{(\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/w'][\epsilon/w] \vdash \Delta[\epsilon/w'][\epsilon/w]}\,Eq_2}{(\epsilon, \epsilon \rhd w); \Gamma[\epsilon/w'] \vdash \Delta[\epsilon/w']}\,\top^* L}{(\epsilon, w' \rhd w); \Gamma; w' : \top^* \vdash \Delta}$$

Notice that the premises of the two derivations below Π are exactly the same. The application of Eq_1 in the original derivation is changed to an application of Eq_2 in the modified derivation. However, this does not break the proof, as the induction hypothesis ensures that either of them can be permuted upwards.

(c) Permute Eq_1/Eq_2 through $*L$ or $-\!\!* R$. We give here an example of permuting Eq_2 through $*L$, assuming the label of principal formula is z, and it is modified by the Eq_2 application. The other cases are similar.

$$\dfrac{\dfrac{\dfrac{\Pi}{(x, y \rhd \epsilon); (\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/z]; x : A; y : B \vdash \Delta[\epsilon/z]}\,*L}{(\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/z]; \epsilon : A * B \vdash \Delta[\epsilon/z]}\,Eq_2}{(\epsilon, \epsilon \rhd z); \Gamma; z : A * B \vdash \Delta}$$

Since x and y are fresh labels, they will not be affected by Eq_2. Thus the derivation can be changed to the following:

$$\dfrac{\dfrac{\begin{array}{c}\Pi\\ (x, y \rhd \epsilon); (\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/z]; x : A; y : B \vdash \Delta[\epsilon/z]\end{array}}{(x, y \rhd z); (\epsilon, \epsilon \rhd z); \Gamma; x : A; y : B \vdash \Delta}\, Eq_2}{(\epsilon, \epsilon \rhd y); \Gamma; z : A * B \vdash \Delta}\, *L$$

Since Eq_1 and Eq_2 only globally replace labels, their action can be safely delayed through all the rules other than $*R$ and $-\!\!* L$. The applications of these two rules after the last $*R$ or $-\!\!* L$ will be delayed until the zero-premise rule is necessary.

2. Permute the applications of E, U, A, or A_C through non-zero premise logical rules other than $*R$ and $-\!\!* L$. Again, we give some examples, the rest are similar.

   (a) Permute E through $\top^* L$, assuming the label of the principal formula is y. The original derivation runs as follows.

   $$\dfrac{\dfrac{\begin{array}{c}\Pi\\ (\epsilon, x \rhd z); (x, \epsilon \rhd z); \Gamma[\epsilon/y] \vdash \Delta[\epsilon/y]\end{array}}{(y, x \rhd z); (x, y \rhd z); \Gamma; y : \top^* \vdash \Delta}\, E}{(x, y \rhd z); \Gamma; y : \top^* \vdash \Delta}\, \top^* L$$

   The new derivation is as follows.

   $$\dfrac{\dfrac{\begin{array}{c}\Pi\\ (\epsilon, x \rhd z); (x, \epsilon \rhd z); \Gamma[\epsilon/y] \vdash \Delta[\epsilon/y]\end{array}}{(x, \epsilon \rhd z); \Gamma[\epsilon/y] \vdash \Delta[\epsilon/y]}\, E}{(x, y \rhd z); \Gamma; y : \top^* \vdash \Delta}\, \top^* L$$

   This shows that if the logical rule only does substitution, delaying the structural rule applications below it makes no difference.

   (b) Permute U through $*L$, assuming the label of the principal formula is z. The original derivation is as follows.

   $$\dfrac{\dfrac{\begin{array}{c}\Pi\\ (x, y \rhd z); (z, \epsilon \rhd z); \Gamma; x : A; y : B \vdash \Delta\end{array}}{(z, \epsilon \rhd z); \Gamma; z : A * B \vdash \Delta}\, *L}{\Gamma; z : A * B \vdash \Delta}\, U$$

   The new derivation is as follows.

   $$\dfrac{\dfrac{\begin{array}{c}\Pi\\ (z, \epsilon \rhd z); (x, y \rhd z); \Gamma; x : A; y : B \vdash \Delta\end{array}}{(x, y \rhd z); \Gamma; z : A * B \vdash \Delta}\, U}{\Gamma; z : A * B \vdash \Delta}\, *L$$

Since the labels x and y are all fresh labels, it is safe to change the order of rule applications as above.

Additive logical rules are totally independent of the relational atoms, so those cases are similar as the one shown above, except that those rules do not add relational atoms to the sequent. □

Now we design a more compact proof system where applications of structural rules are separated into a special entailment relation for relational atoms and equivalence of labels. Such a special entailment can be seen as a "condition" in a rule application. To be specific, since all the structural rule applications can be permuted upwards through negative rule applications, we would apply negative rules, backwards, as much as possible in the proof search, and only apply structural rules, backwards, when they are required by positive rules. In this sense, structural rule applications are encapsulated in positive rule applications as "conditions" of the form "to apply this positive rule, some structural rules have to be applied first". We shall see in the next section that proof search in this proof system can be separated into two phases: guessing the shape of the proof tree, and deriving the relational atoms needed. The latter will be phrased using a constraint system.

In this section we localise the structural rules in two steps: we first deal with Eq_1 and Eq_2, and then the other structural rules.

4.3.1 Localising Eq_1 and Eq_2

Allowing substitutions in a proof rule simplifies the cut-elimination proof for LS_{BBI}. However, for proof search, this creates a problem as Eq_1 and Eq_2 do not permute over certain rules that require matching of two labels (e.g., $*R$ or $-\!\!* L$). Our first intermediate proof system LS^{e}_{BBI} aims to remove substitutions from LS_{BBI}. Instead, the equality between labels is captured via a special entailment relation. To define its inference rules, we first need a few preliminary definitions.

A structural rule r can be seen as defining a relation $\mathcal{R}$ as follows: $(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2) \in \mathcal{R}$ iff there is an instance of r such that

- the set of principal relational atoms (cf. Definition 4.1.1) of the instance is $\mathcal{G}_1$;
- the substitution applied to the premise of the instance is θ;
- the set of fresh labels created in the premise of the instance is $\mathcal{V}$; and
- the set of new relational atoms in the premise of the instance is $\mathcal{G}_2$.

In the following, we shall write $r(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2)$ to denote that $(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2) \in \mathcal{R}$ as defined above. For example, we have both $U(\{\}, [\,], \{x\}, \{(x, \epsilon \triangleright x)\})$ and $U(\{\}, [\,], \{\}, \{(x, \epsilon \triangleright x)\})$, which are justified respectively by the following instances of U:

$$\frac{(x, \epsilon \triangleright x); (a, b \triangleright c) \vdash a : F}{(a, b \triangleright c) \vdash a : F}\, U \qquad \frac{(x, \epsilon \triangleright x); (x, y \triangleright z) \vdash y : G}{(x, y \triangleright z) \vdash y : G}\, U$$

The rule U does not restrict x to be among the labels occuring in the conclusion, so one can create a new label. Similarly, we have $A(\{(x, y \rhd z), (u, v \rhd x)\}, [\,], \{w\}, \{(u, w \rhd z), (y, v \rhd w)\})$ which is justified by, e.g., the following instance of A:

$$\frac{(x, y \rhd z); (u, v \rhd x); (u, w \rhd z), (y, v \rhd w); x : F \vdash y : G}{(x, y \rhd z); (u, v \rhd x); x : F \vdash y : G}\, A$$

and $Eq_1(\{(e, w \rhd w')\}, [w'/w], \{\,\}, \{\,\})$, which is justified by, e.g.,

$$\frac{(\epsilon, w' \rhd w'); w' : F \vdash w' : G}{(\epsilon, w \rhd w'); w : F \vdash w' : G}\, Eq_1$$

We call $r(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2)$ an *abstract instance* of the rule r. The set of abstract instances of structural rules are ranged over by $\mathbf{r}$.

Given a set of relational atoms $\mathcal{G}$, we denote with $LV(\mathcal{G})$ the set of label variables in $\mathcal{G}$. Let σ be a sequence (list) of abstract instances of structural rules $[\mathbf{r}_1; \cdots; \mathbf{r}_n]$. Given a set of relational atoms $\mathcal{G}$, the result of the application of σ to $\mathcal{G}$, denoted by $\mathcal{S}(\mathcal{G}, \sigma)$, is defined inductively as follows where @ stands for sequence (list) concatenation:

$$\mathcal{S}(\mathcal{G}, \sigma) \;=\; \begin{cases} \mathcal{G} & \text{if} \quad \sigma = [\,] \\ \mathcal{S}(\mathcal{G}\theta \cup \mathcal{G}_2, \sigma') & \text{if} \quad \mathcal{G}_1 \subseteq \mathcal{G}, \sigma = [r(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2)]@\sigma' \text{ and } LV(\mathcal{G}) \cap \mathcal{V} = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

Given a sequence $\sigma = [r_1(\mathcal{G}_1, \theta_1, \mathcal{V}_1, \mathcal{G}_1'); \cdots; r_n(\mathcal{G}_n, \theta_n, \mathcal{V}_n, \mathcal{G}_n')]$ of structural rule applications, we denote with $subst(\sigma)$ the composite substitution $\theta_1 \circ \cdots \circ \theta_n$, where $t(\theta_1 \circ \theta_2)$ means $(t\theta_1)\theta_2$. The intuition is that $\mathcal{S}(\mathcal{G}, \sigma)$ is the set of relational atoms after applying a sequence σ of structural rules on a set $\mathcal{G}$ of relational atoms. Since $\mathcal{S}(\mathcal{G}, \sigma)$ is defined based on rule applications in derivations, it is easy to observe that given a sequent $\mathcal{G}; \Gamma \vdash \Delta$ in a derivation and a series σ of structural rule applications above this sequent, $\mathcal{S}(\mathcal{G}, \sigma)$ must be defined.

Definition 4.3.1. *Let $\mathcal{G}$ be a set of relational atoms. The entailment relation $\mathcal{G} \vdash_E (u = v)$ holds iff there exists a sequence σ of abstract instances of Eq_1 or Eq_2 such that $\mathcal{S}(\mathcal{G}, \sigma)$ is defined, and $u\theta = v\theta$, where $\theta = subst(\sigma)$.*

One can equally define the entailment relation $\vdash_E$ as a separated proof system. However, our emphasis here is to drive all the rule applications by logical rules. Absorbing some rules in $\vdash_E$ and only exposing the substitutions involved in the applications are handy in later parts of this work.

We now define the proof system LS^e_{BBI} as $LS_{BBI} \setminus \{Eq_1, Eq_2\}$ (i.e., LS_{BBI} without rules Eq_1, Eq_2) where certain rules are modified according to Figure 4.3. Note that the new $\top^* L$ rule does not modify any labels, instead, the relational atom $(\epsilon, w \rhd \epsilon)$ in the premise ensures that the derivability of $(w = \epsilon)$ is preserved. The point of this intermediate step is to avoid label substitutions in the proof system.

The following theorem states the soundness of LS^e_{BBI} with respect to LS_{BBI}.

$$\frac{\mathcal{G} \vdash_E (w_1 = w_2)}{\Gamma; w_1 : P \vdash w_2 : P; \Delta}\ id \qquad \frac{(\epsilon, w \triangleright \epsilon); \Gamma \vdash \Delta}{\Gamma; w : \top^* \vdash \Delta}\ \top^* L \qquad \frac{\mathcal{G} \vdash_E (w = \epsilon)}{\Gamma \vdash w : \top^*; \Delta}\ \top^* R$$

$$\frac{(x, y \triangleright z'); \Gamma \vdash x : A; z : A * B; \Delta \qquad (x, y \triangleright z'); \Gamma \vdash y : B; z : A * B; \Delta \qquad \mathcal{G} \vdash_E (z = z')}{(x, y \triangleright z'); \Gamma \vdash z : A * B; \Delta}\ *R$$

$$\frac{(x, y' \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad (x, y' \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B; z : B \vdash \Delta \qquad \mathcal{G} \vdash_E (y = y')}{(x, y' \triangleright z); \Gamma; y : A \mathrel{-\!\!*} B \vdash \Delta}\ {-\!\!*} L$$

$$\frac{(u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta \qquad \mathcal{G} \vdash_E (x = x')}{(x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta}\ A$$

$$\frac{(x, w \triangleright x'); (y, y \triangleright w); (x, y \triangleright x'); \Gamma \vdash \Delta \qquad \mathcal{G} \vdash_E (x = x')}{(x, y \triangleright x'); \Gamma \vdash \Delta}\ A_C$$

$\mathcal{G}$ is the set of relational atoms in the left hand side of the conclusion sequent.
In each rule, the entailment $\vdash_E$ is not a premise, but a condition on the rule.

Figure 4.3: The changed rules in LS^e_{BBI}.

Theorem 4.3.2. *If there is a derivation* Π *for a sequent* $\Gamma \vdash \Delta$ *in* LS^e_{BBI}*, then there is a derivation* Π' *for the same sequent in* LS_{BBI}*.*

Proof. By induction on the height n of Π.

1. Base case: $n = 1$. In this case the only rule must be a zero-premise rule. If the rule is $\bot L$ or $\top R$, then we can use the same rule in LS_{BBI}, since they are the same. Otherwise, suppose the rule is *id*, then Π reads as follows.

$$\frac{\mathcal{G} \vdash_E (w_1 = w_2)}{\Gamma; w_1 : P \vdash w_2 : P; \Delta}\ id$$

Since $\mathcal{G} \vdash_E (w_1 = w_2)$ is true, there is a sequence σ of Eq_1, Eq_2 applications such that $\mathcal{S}(\mathcal{G}, \sigma)$ is defined and $w_1\theta = w_2\theta$, where $\theta = subst(\sigma)$. Therefore we can construct Π' as follows:

$$\frac{}{\Gamma\theta; w_1\theta : P \vdash w_2\theta : P; \Delta\theta}\ id$$

$$\vdots\ \sigma$$

$$\Gamma; w_1 : P \vdash w_2 : P; \Delta$$

If the rule is $\top^* R$, suppose Π is:

$$\frac{\mathcal{G} \vdash_E (w = \epsilon)}{\Gamma \vdash w : \top^*; \Delta}\ \top^* R$$

We construct Π' similarly, as $w\theta = \epsilon$ after the application of σ.

$$\dfrac{}{\Gamma\theta \vdash w\theta : \top^*; \Delta\theta}\ {\top^*}R$$

$$\vdots\ \sigma$$
$$\Gamma \vdash w : \top^*; \Delta$$

2. Inductive cases: suppose every sequent that is derivable in LS^e_{BBI} with height less than n is also derivable in LS_{BBI}, consider a LS^e_{BBI} derivation of height n. We do a case analysis on the bottom rule in the derivation.

   (a) If the rule is $\wedge L$, $\wedge R$, $\to L$, $\to R$, $*L$, $*R$, E or U, we can use the same rule in LS_{BBI}, since nothing is changed.

   (b) If the rule is $\top^* L$, then Π must be the following:

$$\dfrac{\begin{array}{c}\Pi_1\\ (\epsilon, w \rhd \epsilon); \Gamma \vdash \Delta\end{array}}{\Gamma; w : \top^* \vdash \Delta}\ {\top^*}L$$

   By the induction hypothesis, $(\epsilon, w \rhd \epsilon); \Gamma \vdash \Delta$ is derivable in LS_{BBI}. Applying Lemma 4.2.2 (substitution for labels in LS_{BBI}) with $[\epsilon/w]$, we obtain $(\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]$. Thus we construct Π' as follows.

$$\dfrac{\dfrac{\begin{array}{c}\Pi'_1\\ (\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]\end{array}}{(w, \epsilon \rhd w); \Gamma; w : \top^* \vdash \Delta}\ {\top^*}L}{\Gamma; w : \top^* \vdash \Delta}\ U$$

   (c) If the rule is $*R$, the derivation Π runs as follows.

$$\dfrac{\begin{array}{c}\Pi_1\\ (x, y \rhd z'); \Gamma \vdash x : A; z : A * B; \Delta\end{array} \qquad \begin{array}{c}\Pi_2\\ (x, y \rhd z'); \Gamma \vdash y : B; z : A * B; \Delta\end{array}}{(x, y \rhd z'); \Gamma \vdash z : A * B; \Delta}\ *R$$

   The condition on the $*R$ rule is $\mathcal{G} \vdash_E (z = z')$. Let σ be the sequence of Eq_1, Eq_2 applications such that $\mathcal{S}(\mathcal{G}, \sigma)$ is defined and, $z\theta = z'\theta$ holds, where $\theta = subst(\sigma)$. Also, by applying the induction hypothesis on Π_1 and Π_2, we obtain the LS_{BBI} derivations for each branch respectively. Then with the help of the substitution lemma (Lem 4.2.2), we get two derivations as below. Note that we use dotted lines to indicate applications of lemmas.

$$\dfrac{\begin{array}{c}\Pi'_1\\ (x, y \rhd z'); \Gamma \vdash x : A; z : A * B; \Delta\end{array}}{(x\theta, y\theta \rhd z'\theta); \Gamma\theta \vdash x\theta : A; z\theta : A * B; \Delta\theta}\ \text{Lemma 4.2.2}$$

   and

$$\dfrac{\begin{array}{c}\Pi'_2\\ (x, y \rhd z'); \Gamma \vdash y : B; z : A * B; \Delta\end{array}}{(x\theta, y\theta \rhd z'\theta); \Gamma\theta \vdash y\theta : B; z\theta : A * B; \Delta\theta}\ \text{Lemma 4.2.2}$$

Then we can apply $*R$ and obtain $(x\theta, y\theta \rhd z'\theta); \Gamma \vdash z\theta : A * B; \Delta\theta$, and by applying σ we obtain the end sequent as follows.

$$\begin{array}{c}(x\theta, y\theta \rhd z'\theta); \Gamma \vdash z\theta : A * B; \Delta\theta \\ \vdots \sigma \\ (x, y \rhd z'); \Gamma \vdash z : A * B; \Delta\end{array}$$

The case for $-\!\!* L$ is treated similarly.

(d) If the rule is A, the treatment for the equality entailment is the same. Π is of the following form:

$$\dfrac{\begin{array}{c}\Pi_1 \\ (u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x'); \Gamma \vdash \Delta \end{array} \qquad \mathcal{G} \vdash_E (x = x')}{(x, y \rhd z); (u, v \rhd x'); \Gamma \vdash \Delta}\, A$$

Let $\mathcal{S}(\mathcal{G}, \sigma)$ yield $x\theta = x'\theta$, where $\theta = subst(\sigma)$, we obtain Π' as follows.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1' \\ (u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x'); \Gamma \vdash \Delta \end{array}}{(u\theta, w\theta \rhd z\theta); (y\theta, v\theta \rhd w\theta); (x\theta, y\theta \rhd z\theta); (u\theta, v\theta \rhd x'\theta); \Gamma\theta \vdash \Delta\theta}\, \text{Lemma 4.2.2}}{(x\theta, y\theta \rhd z\theta); (u\theta, v\theta \rhd x'\theta); \Gamma\theta \vdash \Delta\theta}\, A$$

$$\begin{array}{c}\vdots \sigma \\ (x, y \rhd z); (u, v \rhd x'); \Gamma \vdash \Delta\end{array}$$

The case for A_C is similar. □

To prove the completeness of LS^e_{BBI}, firstly we add Eq_1 and Eq_2 in LS^e_{BBI} and show that the resultant system has the same power as LS_{BBI}. Then we prove the admissibility of Eq_1 and Eq_2 in LS^e_{BBI}.

Lemma 4.3.3. *If a sequent $\Gamma \vdash \Delta$ is derivable in LS_{BBI}, then it is derivable in $LS^e_{BBI} + \{Eq_1, Eq_2\}$.*

Proof. By induction on the height of the LS_{BBI} derivation. Since with Eq_1 and Eq_2, most of the other rules become identical, the only non-trivial case is $\top^* L$.

In LS_{BBI}, the derivation runs as follows.

$$\dfrac{\begin{array}{c}\Pi \\ \Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]\end{array}}{\Gamma; w : \top^* \vdash \Delta}\, \top^* L$$

By the induction hypothesis, there is a derivation for $\Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]$ in $LS^e_{BBI} + \{Eq_1, Eq_2\}$. Therefore we construct the derivation as follows.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi' \\ \Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]\end{array}}{(\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/w] \vdash \Delta[\epsilon/w]}\, \text{Lemma 4.2.6}}{(\epsilon, w \rhd \epsilon); \Gamma \vdash \Delta}\, Eq_1}{\Gamma; w : \top^* \vdash \Delta}\, \top^* L$$

□

The following lemma is a simple application of the Eq_1 rule to show the correspondence between the Eq_1 rule and the equivalence relation $\vdash_E$.

Lemma 4.3.4. *If $\mathcal{G}[x/y]; (\epsilon, x \triangleright x) \vdash_E (w_1[x/y] = w_2[x/y])$ holds then $\mathcal{G}; (\epsilon, y \triangleright x) \vdash_E (w_1 = w_2)$ holds. If $\mathcal{G}[y/x]; (\epsilon, y \triangleright y) \vdash_E (w_1[y/x] = w_2[y/x])$ holds then $\mathcal{G}; (\epsilon, y \triangleright x) \vdash_E (w_1 = w_2)$ holds.*

Proof. Let $\mathcal{G}' = \mathcal{G}; (\epsilon, y \triangleright x)$ and $\mathcal{S}(\mathcal{G}'[x/y], \sigma)$ yield $(w_1[x/y]\theta = w_2[x/y]\theta)$, we show that $\mathcal{G}' \vdash_E (x = y)$ by the following:

$$\dfrac{\begin{array}{c}\mathcal{G}'[x/y]\theta \vdash_E (w_1[x/y]\theta = w_2[x/y]\theta) \\ \vdots\, \sigma \\ \mathcal{G}'[x/y] \vdash_E (w_1[x/y] = w_2[x/y])\end{array}}{\mathcal{G}; (\epsilon, y \triangleright x) \vdash_E (x = y)}\, Eq_1$$

Although to prove the lemma it suffices to only use the last step. The other case is symmetric using the rule Eq_2. □

Now we show that Eq_1 is admissible in LS^e_{BBI}.

Lemma 4.3.5. *If $(\epsilon, x \triangleright x); \Gamma[x/y] \vdash \Delta[x/y]$ is derivable in LS^e_{BBI}, then $(\epsilon, y \triangleright x); \Gamma \vdash \Delta$ is derivable in LS^e_{BBI}. If $(\epsilon, y \triangleright y); \Gamma[y/x] \vdash \Delta[y/x]$ is derivable in LS^e_{BBI}, then $(\epsilon, y \triangleright x); \Gamma \vdash \Delta$ is derivable in LS^e_{BBI}.*

Proof. We show that Eq_1 can always permute up through all the rules in LS^e_{BBI} except for Eq_2, and eventually disappear when it hits the zero-premise rule. Since Lemma 4.3.1 is sufficient to show the permutations through nagative rules, here we particularly show the cases for positive rules.

1. First let us show the cases for the zero-premise rules. $\bot L$ and $\top R$ are trivial, as they are applicable for an arbitrary label. The permutation for *id* runs as follows, where $\mathcal{G}$ is the set of relational atoms in $(\epsilon, y \triangleright x); \Gamma$:

$$\dfrac{\dfrac{\mathcal{G}[x/y] \vdash_E (w_1[x/y] = w_2[x/y])}{(\epsilon, x \triangleright x); \Gamma[x/y]; w_1[x/y] : P \vdash w_2[x/y] : P; \Delta}\, id}{(\epsilon, y \triangleright x); \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, Eq_1$$

   By Lemma 4.3.4, if $\mathcal{G}[x/y] \vdash_E (w_1[x/y] = w_2[x/y])$ then $\mathcal{G} \vdash_E (w_1 = w_2)$ (note that this is because $(\epsilon, y \triangleright x) \in \mathcal{G}$). Therefore we can apply *id* directly on the bottom sequent, and eliminate the Eq_1 application.

   The case for $\top^* R$ is treated similarly.

2. Permute Eq_1 through E, assuming the label being replaced is y. The original derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{(w,x,\triangleright z);(x,w\triangleright z);(\epsilon,w\triangleright w);\Gamma[w/y]\vdash\Delta[w/y]}\,E}{(x,w\triangleright z);(\epsilon,w\triangleright w);\Gamma[w/y]\vdash\Delta[w/y]}\,Eq_1}{(x,y\triangleright z);(\epsilon,y\triangleright w);\Gamma\vdash\Delta}$$

The permuted derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{(w,x,\triangleright z);(x,w\triangleright z);(\epsilon,w\triangleright w);\Gamma[w/y]\vdash\Delta[w/y]}\,Eq_1}{(y,x\triangleright z);(x,y\triangleright z);(\epsilon,y\triangleright w);\Gamma\vdash\Delta}\,E}{(x,y\triangleright z);(\epsilon,y\triangleright w);\Gamma\vdash\Delta}$$

3. Premute Eq_1 through U, assuming the replaced label is x. Then the derivation runs as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{(w,\epsilon\triangleright w);(\epsilon,w\triangleright w);\Gamma[w/x]\vdash\Delta[w/x]}\,U}{(\epsilon,w\triangleright w);\Gamma[w/x]\vdash\Delta[w/x]}\,Eq_1}{(\epsilon,x\triangleright w);\Gamma\vdash\Delta}$$

We modify the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\Pi}{(w,\epsilon\triangleright w);(\epsilon,w\triangleright w);\Gamma[w/x]\vdash\Delta[w/x]}\,Eq_1}{(x,\epsilon\triangleright x);(\epsilon,x\triangleright w);\Gamma\vdash\Delta}\,U}{(\epsilon,x\triangleright w);\Gamma\vdash\Delta}$$

Note that we can also generate $(w,\epsilon\triangleright w)$ directly using the U rule, but the effect is the same.

4. Permute Eq_1 through $*R$. Suppose the principal relational atom of Eq_1 is not the same as the one used in $*R$, let $\mathcal{G}$ be the set of relational atoms in $(\epsilon,w\triangleright w')(x,y\triangleright z');\Gamma$, the derivation runs as follows. Here we write $(\Gamma\vdash\Delta)[x/y]$ to mean replacing every y by x in the entire sequent. The equality entailment is $\mathcal{G}[w'/w]\vdash_E (z[w'/w]=z'[w'/w])$ (to save space, we do not write the constraint in the derivation).

$$\dfrac{\dfrac{}{((\epsilon,w'\triangleright w');(x,y\triangleright z');\Gamma\vdash z:A*B;\Delta)[w'/w]}\,{*R}}{(\epsilon,w\triangleright w');(x,y\triangleright z');\Gamma\vdash z:A*B;\Delta}\,Eq_1$$

The two premises of the $*R$ rule application are listed below.

$$((\epsilon,w'\triangleright w');(x,y\triangleright z');\Gamma\vdash x:A;z:A*B;\Delta)[w'/w]$$
$$((\epsilon,w'\triangleright w');(x,y\triangleright z');\Gamma\vdash y:B;z:A*B;\Delta)[w'/w]$$

By Lemma 4.3.4, since $\mathcal{G}[w'/w] \vdash_E (z[w'/w] = z'[w'/w])$, and $(\epsilon, w \triangleright w') \in \mathcal{G}$, $\mathcal{G} \vdash_E (z = z')$ holds. Therefore we have the following two derivations:

$$\frac{((\epsilon, w' \triangleright w'); (x, y \triangleright z'); \Gamma \vdash x : A; z : A * B; \Delta)[w'/w]}{(\epsilon, w \triangleright w'); (x, y \triangleright z'); \Gamma \vdash x : A; z : A * B; \Delta} \; Eq_1$$

and

$$\frac{((\epsilon, w' \triangleright w'); (x, y \triangleright z'); \Gamma \vdash y : B; z : A * B; \Delta)[w'/w]}{(\epsilon, w \triangleright w'); (x, y \triangleright z'); \Gamma \vdash y : B; z : A * B; \Delta} \; Eq_1$$

then we use the $*R$ rule, where the equality entailment is $\mathcal{G} \vdash_E (z = z')$, to obtain the end sequent $(\epsilon, w \triangleright w'); (x, y \triangleright z'); \Gamma \vdash z : A * B; \Delta$.

If the principal relational atom is used in the $*R$ rule, the permutation is analogous. The permutation through $-\!\!* L$ is similar.

5. Permutation through A. We show the case where the principal relational atom in Eq_1 is not in A. The other cases are similar. The original derivation is as follows:

$$\frac{\dfrac{((\epsilon, w \triangleright w); (u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta)[w/w']}{((\epsilon, w \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta)[w/w']} \; Eq_1}{(\epsilon, w' \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta} \; A$$

The condition on the A rule is $\mathcal{G}[w/w'] \vdash_E (x[w/w'] = x'[w/w'])$. By Lemma 4.3.4, $\mathcal{G} \vdash_E (x = x')$ holds. Therefore the derivation is transformed into the following:

$$\frac{\dfrac{((\epsilon, w' \triangleright w); (u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta)[w/w']}{(\epsilon, w' \triangleright w); (u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta} \; A}{(\epsilon, w' \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \Gamma \vdash \Delta} \; Eq_1$$

The condition on the A rule is $\mathcal{G} \vdash_E (x = x')$. The rule A_C is treated similarly.

The case for Eq_2 is symmetric.

Although we have not showed that Eq_1 and Eq_2 can permute through each other, the above proof suffices to show that they can permute through all other rule applications in LS^e_{BBI}, thus we can eliminate the topmost Eq_1 or Eq_2 application one by one, eventually eliminate all Eq_1 and Eq_2 applications. □

Combining the soundness proof (Theorem 4.3.2) and the completeness proof (Lemma 4.3.3 to Lemma 4.3.5), we obtain the following correspondence between LS_{BBI} and LS^e_{BBI}:

Theorem 4.3.6. *A sequent $\Gamma \vdash \Delta$ is derivable in LS_{BBI} if and only if it is derivable in LS^e_{BBI}.*

Now we prove the substitution lemma for the intermediate system LS^{e}_{BBI}, as this will be used in some proofs.

Lemma 4.3.7. *If* $\mathcal{G} \vdash_E (x = y)$ *then for any substitution* $[s/t]$*, where* $t \neq \epsilon$*,* $\mathcal{G}[s/t] \vdash_E (x[s/t] = y[s/t])$*.*

Proof. Let σ be the solution to $\mathcal{G} \vdash_E (x = y)$ with substitutions ϕ. We prove this lemma by induction on the length of σ.

1. Base case, σ is an empty sequence. In this case, the sequence of substitutions ϕ is also empty, therefore $x = y$. As a result, it must be the case that $x[s/t] = y[s/t]$, so $\mathcal{G}[s/t] \vdash_E (x[s/t] = y[s/t])$ trivially holds.

2. Inductive case, assume $|\sigma| = n$. Let us look at the bottom rule application in σ. Assume this rule is Eq_1 (the case for Eq_2 is symmetric), and the principal relational atom is $(\epsilon, u \triangleright v)$, then σ is as follows:

$$\dfrac{\begin{array}{c}\mathcal{G}\phi \vdash_E (x\phi = y\phi) \\ \vdots\, \sigma' \\ \mathcal{G}'[v/u]; (\epsilon, v \triangleright v) \vdash_E (x[v/u] = y[v/u])\end{array}}{\mathcal{G}'; (\epsilon, u \triangleright v) \vdash_E (x = y)}\, Eq_1$$

   (a) If $u = t$ and $v = s$, then the premise of the last rule application is already what we need.

   (b) If $u = t$ and $v \neq s$, we obtain the desired entailment as follows ($IH[x/y]$ stands for applying the induction hypothesis with the substitution $[x/y]$, we use double line to mean that the premise and the conclusion are equivalent).

$$\dfrac{\dfrac{\begin{array}{c}IH[v/s] \\ \mathcal{G}'[v/u][v/s]; (\epsilon, v \triangleright v) \vdash_E (x[v/u][v/s] = y[v/u][v/s])\end{array}}{\mathcal{G}'[s/u][v/s]; (\epsilon, v \triangleright v) \vdash_E (x[s/u][v/s] = y[s/u][v/s])}}{\mathcal{G}'[s/u]; (\epsilon, s \triangleright v) \vdash_E (x[s/u] = y[s/u])}\, Eq_1$$

   (c) If $u = s$, we prove the substituted entailment as follows:

$$\dfrac{\dfrac{\begin{array}{c}IH[v/t] \\ \mathcal{G}'[v/u][v/t]; (\epsilon, v \triangleright v) \vdash_E (x[v/u][v/t] = y[v/u][v/t])\end{array}}{\mathcal{G}'[u/t][v/u]; (\epsilon, v \triangleright v) \vdash_E (x[u/t][v/u] = y[u/t][v/u])}}{\mathcal{G}'[u/t]; (\epsilon, u \triangleright v) \vdash_E (x[u/t] = y[u/t])}\, Eq_1$$

   In this case if $v = t$, the proof is just a special case of the one above.

   (d) If $v = t$, the case is shown below.

$$\dfrac{\dfrac{\dfrac{IH[s/v]}{\mathcal{G}'[v/u][s/v];(\epsilon, s \rhd s) \vdash_E (x[v/u][s/v] = y[v/u][s/v])}}{\mathcal{G}'[s/v][s/u];(\epsilon, s \rhd s) \vdash_E (x[s/v][s/u] = y[s/v][s/u])}\, Eq_1}{\mathcal{G}'[s/v];(\epsilon, u \rhd s) \vdash_E (x[s/v] = y[s/v])}$$

(e) If $v = s$, the proof is as follows:

$$\dfrac{\dfrac{\dfrac{IH[v/t]}{\mathcal{G}'[v/u][v/t];(\epsilon, v \rhd v) \vdash_E (x[v/u][v/t] = y[v/u][v/t])}}{\mathcal{G}'[v/t][v/u];(\epsilon, v \rhd v) \vdash_E (x[v/t][v/u] = y[v/t][v/u])}\, Eq_1}{\mathcal{G}'[v/t];(\epsilon, u \rhd v) \vdash_E (x[v/t] = y[v/t])}$$

(f) If $[s/t]$ and $[u/v]$ are independent, then we can switch the order of substitution, and derive the entailment as follows:

$$\dfrac{\dfrac{\dfrac{IH[s/t]}{\mathcal{G}'[v/u][s/t];(\epsilon, v \rhd v) \vdash_E (x[v/u][s/t] = y[v/u][s/t])}}{\mathcal{G}'[s/t][v/u];(\epsilon, v \rhd v) \vdash_E (x[s/t][v/u] = y[s/t][v/u])}\, Eq_1}{\mathcal{G}'[s/t];(\epsilon, u \rhd v) \vdash_E (x[s/t] = y[s/t])}$$

□

Since substitution does not break the equality entailment, we can show a substitution lemma for the system LS^e_{BBI}.

Lemma 4.3.8 (Substitution in LS^e_{BBI})**.** *If there is a derivation for the sequent $\Gamma \vdash \Delta$ in LS^e_{BBI} then there is a derivation of the same height for the sequent $\Gamma[y/x] \vdash \Delta[y/x]$ in LS^e_{BBI}, where every occurrence of label variable $x \in LVar$ is replaced by label y.*

Proof. The proof is basically the same as the one for LS_{BBI}, since there are a lot of common rules. For the rules that are changed, the case for $\top^* L$ is similar to those cases for additive rules. The proof for the rest of the changed rules are straightforward with the help of Lemma 4.3.7. □

4.3.2 Localising the Rest of the Structural Rules

As a second step, we isolate the other structural rules into a separate entailment relation, as we did with Eq_1 and Eq_2, giving another intermediate system LS^{sf}_{BBI}.

Definition 4.3.2 (Relation Entailment $\vdash_R$)**.** *The entailment relation $\vdash_R$ has the following two forms:*

1. *$\mathcal{G} \vdash_R (w_1 = w_2)$ is true iff there is a sequence σ of E, U, A, A_C applications so that $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_2)$.*

2. $\mathcal{G} \vdash_R (w_1, w_2 \triangleright w_3)$ *is true iff there is a sequence* σ *of* E, U, A, A_C *applications so that* $(w_1', w_2' \triangleright w_3') \in \mathcal{S}(\mathcal{G}, \sigma)$ *and the following hold:* $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_1')$, $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_2 = w_2')$, *and* $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_3 = w_3')$.

In each case, we say that σ *is a derivation of the entailment relation involved.*

The entailment $\vdash_R$ is stronger than $\vdash_E$. For example, if $\mathcal{G}$ only contains $(x, \epsilon \triangleright y)$, then $\mathcal{G} \nvdash_E (x = y)$; but $\mathcal{G} \vdash_R (x = y)$ by applying E to obtain $(\epsilon, x \triangleright y)$, then apply Eq_1 or Eq_2 on the new relational atom.

$$\frac{\mathcal{G} \vdash_R (w_1 = w_2)}{\mathcal{G}||\Gamma; w_1 : P \vdash w_2 : P; \Delta}\ id \qquad \frac{\mathcal{G} \vdash_R (w = \epsilon)}{\mathcal{G}||\Gamma \vdash w : \top^*; \Delta}\ \top^* R$$

$$\frac{\mathcal{S}(\mathcal{G}, \sigma)||\Gamma \vdash x : A; w : A * B; \Delta \qquad \mathcal{S}(\mathcal{G}, \sigma)||\Gamma \vdash y : B; w : A * B; \Delta \qquad \mathcal{G} \vdash_R (x, y \triangleright w)}{\mathcal{G}||\Gamma \vdash w : A * B; \Delta}\ *R^{\dagger}$$

$$\frac{\mathcal{S}(\mathcal{G}, \sigma)||\Gamma; w : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad \mathcal{S}(\mathcal{G}, \sigma)||\Gamma; w : A \mathrel{-\!\!*} B; z : B \vdash \Delta \qquad \mathcal{G} \vdash_R (x, w \triangleright z)}{\mathcal{G}||\Gamma; w : A \mathrel{-\!\!*} B \vdash \Delta}\ \mathrel{-\!\!*} L^{\ddagger}$$

†: σ is the derivation of $\mathcal{G} \vdash_R (x, y \triangleright w)$ ‡: σ is the derivation of $\mathcal{G} \vdash_R (x, w \triangleright z)$

In each rule, the entailment $\vdash_R$ is not a premise, but a condition on the rule.

Figure 4.4: Changed rules in LS^{sf}_{BBI}.

The changed rules in the second intermediate system LS^{sf}_{BBI} are given in Figure 4.4 where we use a slightly different notation for sequents. We write $\mathcal{G}||\Gamma \vdash \Delta$ to emphasize that the left hand side of a sequent is partitioned into two parts: $\mathcal{G}$, which contains only relational atoms; and Γ, which contains only labelled formulae.

We first show the soundness of LS^{sf}_{BBI} with respect to LS^{e}_{BBI}.

Theorem 4.3.9. *If there is a derivation* Π *for a sequent* $\mathcal{G}||\Gamma \vdash \Delta$ *in* LS^{sf}_{BBI}, *then there is a derivation* Π' *for the sequent* $\mathcal{G}; \Gamma \vdash \Delta$ *in* LS^{e}_{BBI}.

Proof. The soundness proof for this system is rather straightforward. To prove this, we show that each rule in LS^{sf}_{BBI} is derivable in LS^{e}_{BBI}. To do this, one just needs to unfold the structural rule applications into the derivation. For instance, we can simulate the *id* rule in LS^{sf}_{BBI} by using the following rules in LS^{e}_{BBI}:

$$\frac{\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_2)}{\mathcal{S}(\mathcal{G}, \sigma); \Gamma; w_1 : P \vdash w_2 : P; \Delta}\ id$$

$$\vdots\ \sigma$$

$$\mathcal{G}; \Gamma; w_1 : P \vdash w_2 : P; \Delta$$

The above works because the *id* rule in LS^{sf}_{BBI} requires $\mathcal{G} \vdash_R (w_1 = w_2)$, which by definition ensures that $\mathcal{S}(\mathcal{G}, \sigma) \vdash_E (w_1 = w_2)$ holds. The case for $\top^* R$ works

similarly. One thing to notice is that the structural rules only add relational atoms into the current set, so except for the fact that $\mathcal{G}$ is becoming a bigger set, all the other structures in the sequent remain the same after the sequence σ of applications. Let us examine the simulation of $*R$ in LS^{e}_{BBI}.

$$\dfrac{\dfrac{\mathcal{S}(\mathcal{G},\sigma);\Gamma \vdash x' : A; w : A * B; \Delta \qquad \mathcal{S}(\mathcal{G},\sigma);\Gamma \vdash y' : B; w : A * B; \Delta}{\mathcal{S}(\mathcal{G},\sigma);\Gamma \vdash w : A * B; \Delta}\, {*R}}{\begin{array}{c}\vdots\, \sigma \\ \mathcal{G};\Gamma \vdash w : A * B; \Delta\end{array}}$$

The condition of the $*R$ rule is $\mathcal{S}(\mathcal{G},\sigma) \vdash_E (w = w')$. Since the LS^{sf}_{BBI} rule requires $\mathcal{G} \vdash_R (x, y \triangleright w)$, which by definition ensures that there is a structural rule derivation σ such that $(x', y' \triangleright w') \in \mathcal{S}(\mathcal{G},\sigma)$, we have:

$$\begin{array}{c}\mathcal{S}(\mathcal{G},\sigma) \vdash_E (x = x') \\ \mathcal{S}(\mathcal{G},\sigma) \vdash_E (y = y') \\ \mathcal{S}(\mathcal{G},\sigma) \vdash_E (w = w')\end{array}$$

The last relation entailment is enough to guarantee that the $*R$ rule is applicable. To restore each branch, we need Lemma 4.3.8 (Substitution lemma for LS^{e}_{BBI}). Let us look at the left branch. By the first relation entailment, there is a sequence σ' of Eq_1, Eq_2 applications so that $x\theta = x'\theta$. So we can construct a proof for the left branch as follows.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{S}(\mathcal{G},\sigma);\Gamma \vdash x : A; w : A * B; \Delta \\ \hdotsfor{1} \\ \mathcal{S}(\mathcal{G},\sigma)\theta;\Gamma\theta \vdash x\theta : A; w\theta : A * B; \Delta\theta\end{array}\, \text{Substitution Lemma 4.3.8}}{\mathcal{S}(\mathcal{G},\sigma)\theta;\Gamma\theta \vdash x'\theta : A; w\theta : A * B; \Delta\theta}}{\begin{array}{c}\vdots\, \sigma' \\ \mathcal{S}(\mathcal{G},\sigma);\Gamma \vdash x' : A; w : A * B; \Delta\end{array}}$$

Now the top sequent is the left premise of the $*R$ rule in LS^{sf}_{BBI}. The case for the right premise is symmetric. Therefore we can simulate the $*R$ rule in LS^{sf}_{BBI} by using a derivation in LS^{e}_{BBI}.

The case for $-\!\!* L$ is analogous. The other rules are the same as in LS^{e}_{BBI}, thus we conclude that the rules in LS^{sf}_{BBI} are sound. □

The completeness proof runs the same as in LS^{e}_{BBI}: if we add the structural rules E, U, A, A_C in LS^{sf}_{BBI}, then it becomes a superset of LS^{e}_{BBI}. Then we prove that these rules are admissible in LS^{sf}_{BBI} by showing that they can permute through the rules $*R$, $-\!\!* L$, id, and $\top^* R$.

First of all, let us show that when we add E, U, A, A_C (from LS^{e}_{BBI}) to LS^{sf}_{BBI}, its rules can simulate those ones in LS^{e}_{BBI}. As most of the rules are identical, the key part

is to show the relation entailment is as powerful as the equality entailment. This is "built-in" the definition, so there is no surprise.

Lemma 4.3.10. *If* $\mathcal{G} \vdash_E (w_1 = w_2)$, *then* $\mathcal{G} \vdash_R (w_1 = w_2)$.

Proof. Let σ be an empty list of E, U, A, A_C rule applications, then $\mathcal{S}(\mathcal{G}, \sigma) = \mathcal{G}$ and $\mathcal{G} \vdash_E (w_1 = w_2)$. Therefore by definition $\mathcal{G} \vdash_R (w_1 = w_2)$. □

If we change $\vdash_R$ to $\vdash_E$ in LS^{sf}_{BBI}, every rule is the same as the one in LS^{e}_{BBI}. Therefore $LS^{sf}_{BBI} + E + U + A + A_C$ is at least as powerful as LS^{e}_{BBI}.

Lemma 4.3.11. *The rules* E, U, A, *and* A_C *are admissible in* LS^{sf}_{BBI}.

Proof. We show that the said rules can permute upwards through *id*, $\top^* R$, $*R$ and $-\!\!* L$, the other cases are cover by Lemma 4.3.1. We only give some examples here, the others are similar. The heart of the argument is that the application of structural rules are hidden inside the relation entailment, so we do not have to apply them explicitly.

Permute E through *id*, the suppose the original derivation runs as follows.

$$\dfrac{\dfrac{\mathcal{G}; (y, x \rhd z); (x, y \rhd z) \vdash_R (w_1 = w_2)}{\mathcal{G}; (y, x \rhd z); (x, y \rhd z) || \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, id}{\mathcal{G}; (x, y \rhd z) || \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, E$$

The permuted derivation is:

$$\dfrac{\mathcal{G}; (x, y \rhd z) \vdash_R (w_1 = w_2)}{\mathcal{G}; (x, y \rhd z) || \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, id$$

Assume that $\mathcal{G}; (y, x \rhd z); (x, y \rhd z) \vdash_R (w_1 = w_2)$ is derived by applying a sequence σ of structural rules. Then $\mathcal{S}((\mathcal{G}; (x, y \rhd z)), \sigma')$ can prove $\mathcal{G}; (x, y \rhd z) \vdash_R (w_1 = w_2)$, where σ' is $E(\{(x, y \rhd z)\}, \emptyset)$ followed by σ.

Permuting A upwards through *id* is similar. The original derivation is:

$$\dfrac{\dfrac{\mathcal{G}; (u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x') \vdash_R (w_1 = w_2)}{\mathcal{G}; (u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x') || \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, id}{\mathcal{G}; (x, y \rhd z); (u, v \rhd x') || \Gamma; w_1 : P \vdash w_2 : P; \Delta}\, A$$

The condition on the rule A is $\mathcal{G}; (x, y \rhd z); (u, v \rhd x') \vdash_E (x = x')$. Then we can omit the application of A, since $\mathcal{G}; (u, w \rhd z); (y, v \rhd w); (x, y \rhd z); (u, v \rhd x') \vdash_R (w_1 = w_2)$ implies $\mathcal{G}; (x, y \rhd z); (u, v \rhd x') \vdash_R (w_1 = w_2)$, one just need to add the A application ahead to the sequence of structural rules that derives the former relation entailment to get a new sequence of rules to derive the latter one.

Finally, we show a case where E is applied below $*R$. The derivation runs as below:

$$\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \mathcal{G}; (x, y \rhd z); (y, x \rhd z) || \Gamma \vdash y : A; z : A * B; \Delta & \mathcal{G}; (x, y \rhd z); (y, x \rhd z) || \Gamma \vdash x : B; z : A * B; \Delta\end{array}}{\mathcal{G}; (x, y \rhd z); (y, x \rhd z) || \Gamma \vdash z : A * B; \Delta}\, *R}{\mathcal{G}; (x, y \rhd z) || \Gamma \vdash z : A * B; \Delta}\, E$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdots \vdash_E (w_2 = w_2)}{\cdots ; w_2 : r \vdash w_2 : r; \cdots}\, id \quad \dfrac{(\epsilon, w_3 \rhd \epsilon); (w_3, w_4 \rhd w_1); \cdots \vdash_E (w_4 = w_1)}{\cdots ; w_4 : q \vdash w_1 : q; \cdots}\, id}{(w_2, w_1 \rhd w_0); \cdots ; w_3 : p; w_3 : \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, *R}{(\epsilon, w_3 \rhd \epsilon); (w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0); w_3 : p; w_3 : \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, E}{(w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0); w_3 : p; w_3 : \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, \top^* L}{(w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0); w_3 : p \wedge \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, \wedge L}{w_0 : ((p \wedge \top^*) * q) * r \vdash w_0 : r * q}\, *L \times 2}{\vdash w_0 : (((p \wedge \top^*) * q) * r) \to (r * q)}\, \to R$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdots \vdash_R (w_2 = w_2)}{(w_2, w_1 \rhd w_0); \cdots ; || \cdots ; w_2 : r \vdash w_2 : r; \cdots}\, id \quad \dfrac{(\epsilon, w_3 \rhd \epsilon); (w_3, w_4 \rhd w_1); \cdots \vdash_R (w_4 = w_1)}{(w_2, w_1 \rhd w_0); \cdots || \cdots ; w_4 : q \vdash w_1 : q; \cdots}\, id \quad C}{(\epsilon, w_3 \rhd \epsilon); (w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0) || w_3 : p; w_3 : \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, *R}{(w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0) || w_3 : p; w_3 : \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, \top^* L}{(w_3, w_4 \rhd w_1); (w_1, w_2 \rhd w_0) || w_3 : p \wedge \top^*; w_4 : q; w_2 : r \vdash w_0 : r * q}\, \wedge L}{w_0 : ((p \wedge \top^*) * q) * r \vdash w_0 : r * q}\, *L \times 2}{\vdash w_0 : (((p \wedge \top^*) * q) * r) \to (r * q)}\, \to R$$

where $C ::= (w_1, w_2 \rhd w_0); \cdots \vdash_R (w_2, w_1 \rhd w_0)$

Figure 4.5: Example derivations in LS^{e}_{BBI} (top) and LS^{sf}_{BBI} (bottom).

Apparently the structural rules, including E, can be absorbed into the notion of $\vdash_R$. In this case, the sequence of structural rule applications $\sigma = E$ ensures that $(y, x \rhd z) \in \mathcal{S}(\mathcal{G} \cup \{(x, y \rhd z)\}, \sigma)$. Thus $\mathcal{G} \cup \{(x, y \rhd z)\} \vdash_R (y, x \rhd z)$, thus we have the following new derivation in LS^{sf}_{BBI}:

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \mathcal{S}(\mathcal{G} \cup \{(x, y \rhd z)\}, \sigma) || \Gamma \vdash y : A; z : A * B; \Delta & \mathcal{S}(\mathcal{G} \cup \{(x, y \rhd z)\}, \sigma) || \Gamma \vdash x : B; z : A * B; \Delta\end{array}}{\mathcal{G}; (x, y \rhd z) || \Gamma \vdash z : A * B; \Delta}\, *R$$

where $\mathcal{S}(\mathcal{G} \cup \{(x, y \rhd z)\}, \sigma) = \mathcal{G} \cup \{(x, y \rhd z), (y, x \rhd z)\}$. Recall that we have shifted the notation for sequents so that they consist of sets, the premises in the new derivation is equivalent to the premises in the old derivation. The other cases can be shown in the same way. That is, the structural rules do not disappear in the derivation, but are simply absorbed in negative rules. □

The completeness of LS^{sf}_{BBI} is then an immediate result of Lemma 4.3.11.

Theorem 4.3.12. *A sequent $\Gamma \vdash \Delta$ is derivable in LS^{e}_{BBI} if and only if it is derivable in LS^{sf}_{BBI}.*

As a consequence of Theorem 4.3.6 and 4.3.12, we can also obtain the equivalence between LS_{BBI} and LS^{sf}_{BBI}. The latter will be used in the next section. Figure 4.5 shows example derivations in the intermediate systems LS^{e}_{BBI} and LS^{sf}_{BBI} respectively in contrast to the derivation in Figure 4.2.

4.4 Proof Search using Free Variables

We now consider a proof search strategy for a system based on LS^{sf}_{BBI}. As we have isolated all the structural rules into the entailment relation $\vdash_R$, proof search in LS^{sf}_{BBI} consists of guessing the shape of the derivation tree, and then checking that each entailment $\vdash_R$ can be proved. The latter involves guessing a splitting of labels in the $*R$ and $-\!\!* L$ rules which also satisfies the equality constraints in the *id* and $\top^* R$ rules. We formalise this via a symbolic proof system, where splitting and equality are handled lazily, via the introduction of *free variables* which are essentially existential variables (or logic variables) that must be instantiated to concrete labels satisfying all the entailment constraints in the proof tree, for a derivation to be sound. The idea of using free variables can be found in the literature, e.g., [4], in which a benefit is that "the use of free variables generates a smaller search space". We shall see in the following sections that our free variable system, although different from existing ones in certain aspects, can also narrow down the search space when zero-premise rules can give exact equality constraints so that the constraints generated by logical rules can be solved based on those generated by zero-premise rules. This means that the applications of structural rules (hidden in the logical rules) are not only driven by logical rules, but also by zero-premise rules.

Free variables are denoted by $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. We use $\mathbf{u}, \mathbf{v}, \mathbf{w}$ to denote either labels or free variables, and a, b, c are ordinary labels. A *symbolic sequent* is just a sequent but possibly with occurrences of free variables in place of labels. We shall sometimes refer to the normal (non-symbolic) sequent as a *ground sequent* to emphasise the fact that it contains no free variables. The symbolic proof system $FVLS_{BBI}$ is given in Figure 4.6. The rules are mostly similar to LS^{sf}_{BBI}, but lacking the entailment relations $\vdash_R$. Instead, new free variables are introduced when applying $*R$ and $-\!\!* L$ backwards. Notice also that in $FVLS_{BBI}$, the $*R$ and $-\!\!* L$ rules do not compute the set $\mathcal{S}(\mathcal{G}, \sigma)$. So the relational atoms in $FVLS_{BBI}$ are only those that are created by $*L, -\!\!* R, \top^* L$. In the following, given a derivation in $FVLS_{BBI}$, we shall assume that the free variables that are created in different branches of the derivation are pairwise distinct. We shall sometimes refer to a derivation in $FVLS_{BBI}$ simply as a *symbolic derivation*.

An *equality constraint* is an expression of the form $\mathcal{G} \vdash^?_R (\mathbf{u} = \mathbf{v})$, and a *relational constraint* is an expression of the form $\mathcal{G} \vdash^?_R (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$. In both cases, we refer to $\mathcal{G}$ as the left hand side of the constraints, and $(\mathbf{u} = \mathbf{v})$ and $(\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$ as the right hand side. Constraints are ranged over by $\mathfrak{c}, \mathfrak{c}', \mathfrak{c}_1, \mathfrak{c}_2$, etc. Given a constraint $\mathfrak{c}$, we write $\mathcal{G}(\mathfrak{c})$ for the left hand side of $\mathfrak{c}$. A *constraint system* is just a set of constraints. We write $\mathcal{G} \vdash^?_R C$ for either an equality or relational constraint. We use $fv(\mathfrak{c})$ to denote the set of free variables in $\mathfrak{c}$, and $fv(\mathcal{C})$ to denote the set of free variables in a set of constraints $\mathcal{C}$.

Definition 4.4.1 (Constraint systems)**.** *A* constraint system *is a pair $(\mathcal{C}, \preceq)$ of a multiset of constraints and a partial order on elements of $\mathcal{C}$ satisfying:*

Monotonicity: $\mathfrak{c}_1 \preceq \mathfrak{c}_2$ *implies* $\mathcal{G}(\mathfrak{c}_1) \subseteq \mathcal{G}(\mathfrak{c}_2)$.

Initial Sequent:

$$\frac{}{\mathcal{G}||\Gamma; \mathbf{w}_1 : P \vdash \mathbf{w}_2 : P; \Delta}\ id$$

Logical Rules:

$$\frac{}{\mathcal{G}||\Gamma; \mathbf{w} : \bot \vdash \Delta}\ \bot L \qquad \frac{}{\mathcal{G}||\Gamma \vdash \mathbf{w} : \top; \Delta}\ \top R$$

$$\frac{\mathcal{G}; (\epsilon, \mathbf{w} \triangleright \epsilon)||\Gamma \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : \top^* \vdash \Delta}\ \top^* L \qquad \frac{}{\mathcal{G}||\Gamma \vdash \mathbf{w} : \top^*; \Delta}\ \top^* R$$

$$\frac{\mathcal{G}||\Gamma; \mathbf{w} : A; \mathbf{w} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \wedge B \vdash \Delta}\ \wedge L \qquad \frac{\mathcal{G}||\Gamma \vdash \mathbf{w} : A; \Delta \qquad \mathcal{G}||\Gamma \vdash \mathbf{w} : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \wedge B; \Delta}\ \wedge R$$

$$\frac{\mathcal{G}||\Gamma \vdash \mathbf{w} : A; \Delta \qquad \mathcal{G}||\Gamma; \mathbf{w} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \to B \vdash \Delta}\ \to L \qquad \frac{\mathcal{G}||\Gamma; \mathbf{w} : A \vdash \mathbf{w} : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \to B; \Delta}\ \to R$$

$$\frac{\mathcal{G}; (a, b \triangleright \mathbf{w})||\Gamma; a : A; b : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A * B \vdash \Delta}\ *L^{\dagger} \qquad \frac{\mathcal{G}; (a, \mathbf{w} \triangleright c)||\Gamma; a : A \vdash c : B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A \mathrel{-\!\!*} B; \Delta}\ -\!\!* R^{\ddagger}$$

$$\frac{\mathcal{G}||\Gamma \vdash \mathbf{x} : A; \mathbf{w} : A * B; \Delta \qquad \mathcal{G}||\Gamma \vdash \mathbf{y} : B; \mathbf{w} : A * B; \Delta}{\mathcal{G}||\Gamma \vdash \mathbf{w} : A * B; \Delta}\ *R^{\sharp}$$

$$\frac{\mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B \vdash \mathbf{x} : A; \Delta \qquad \mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B; \mathbf{z} : B \vdash \Delta}{\mathcal{G}||\Gamma; \mathbf{w} : A \mathrel{-\!\!*} B \vdash \Delta}\ -\!\!* L^{\natural}$$

$\dagger$: a and b must be fresh in $*L$ $\quad$ $\ddagger$: a and c must be fresh in $-\!\!* R$
$\sharp$: $\mathbf{x}$ and $\mathbf{y}$ are new free variables in $*R$ $\quad$ $\natural$: $\mathbf{x}$ and $\mathbf{z}$ are new free variables in $-\!\!* L$

Figure 4.6: Labelled Sequent Calculus $FVLS_{BBI}$ for Boolean BI.

A constraint system is well-formed *if it also satisfies the condition below.*

Unique variable origin: $\forall \mathbf{x}$ *in* $\mathcal{C}$*, there exists a unique constraint occurrence* $\mathfrak{c}(\mathbf{x}) = \mathcal{G}_x \vdash^?_R (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$ *s.t.* $\mathbf{x}$ *occurs in* $(\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$*, but not in* $\mathcal{G}_x$*, and* $\mathbf{x}$ *does not occur in any* $\mathfrak{c}'$ *where* $\mathfrak{c}' \neq \mathfrak{c}(\mathbf{x})$ *and* $\mathfrak{c}' \preceq \mathfrak{c}(\mathbf{x})$*. Such a* $\mathfrak{c}(\mathbf{x})$ *is the origin of* $\mathbf{x}$*. Furthermore, for any free variable* $\mathbf{x}$ *and any constraint* $\mathfrak{c}'$*, if* $\mathbf{x}$ *occurs in* $\mathcal{G}(\mathfrak{c}')$*, then* $\mathfrak{c}(\mathbf{x}) \preceq \mathfrak{c}'$*.*

Notice that $\mathcal{C}$ is a multiset, rather than a set. We could have defined $\mathcal{C}$ as a set, but the definition of composition of constraint systems (Definition 4.4.8) would be more complicated. Thus in $\mathcal{C}$ there can be more than one occurrence of the same constraint $\mathfrak{c}$.[3] We write $\mathcal{C}_1 \uplus \mathcal{C}_2$ to denote the multiset union of $\mathcal{C}_1$ and $\mathcal{C}_2$. We shall use the same symbol for a constraint and its occurrences. We shall often refer to a constraint occurrence as simply a constraint when it is clear from the context of discussions that we are referring to an occurrence rather than a constraint.

In a well-formed constraint system $(\mathcal{C}, \preceq)$, in every minimum constraint $\mathfrak{c}$, with respect to $\preceq$, $\mathcal{G}(\mathfrak{c})$ must be ground, i.e., has no free variables (see Lemma 4.4.2). The

[3] Another way of looking at this is to consider a set of pairs of the form $(i, \mathfrak{c})$ where i is an identifier (e.g., a natural number) and $\mathfrak{c}$ is a constraint. Multisets offer a more convenient abstraction.

existence of such a $\mathfrak{c}$ is important in the definition of solutions for a well-formed constraint system, and in the proof that the symbolic proof system is sound with respect to its concrete counterpart (i.e., the derivation for the same formula in LS^{sf}_{BBI}).

From now on, we shall denote with $\mathfrak{c}(\mathbf{x})$ the constraint occurrence from which $\mathbf{x}$ originates, as defined in the above definition. We use the letter $\mathbb{C}$ to range over constraint systems.

We write $\mathfrak{c}_i \prec \mathfrak{c}_j$ when $\mathfrak{c}_i \preceq \mathfrak{c}_j$ and $\mathfrak{c}_i \neq \mathfrak{c}_j$. Further, we define a direct successor relation $\lessdot$ as follows: $\mathfrak{c}_i \lessdot \mathfrak{c}_j$ iff $\mathfrak{c}_i \prec \mathfrak{c}_j$ and there does not exist any $\mathfrak{c}_k$ such that $\mathfrak{c}_i \prec \mathfrak{c}_k \prec \mathfrak{c}_j$.

In proof search, associated constraints are generated as follows.

Definition 4.4.2. *To a given symbolic derivation* Π*, we define a multiset of constraints* $\mathcal{C}(\Pi)$ *by structural induction on* Π*. We shall assume that variables introduced in instances of* $*R$ *and* $-\!\!* L$ *in* Π *are pairwise distinct. In the following, for each instance of the rules, we use the same naming schemes for labels and variables as in Figure 4.6. We distinguish several (base/inductive) cases based on the lowest rule of* Π*:*

id	$\mathcal{C}(\Pi) = \{\mathcal{G} \vdash^?_R (\mathbf{w}_1 = \mathbf{w}_2)\}$
$\top^* R$	$\mathcal{C}(\Pi) = \{\mathcal{G} \vdash^?_R (\mathbf{w} = \epsilon)\}$
$*R$	$\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \uplus \mathcal{C}(\Pi_2) \uplus \{\mathcal{G} \vdash^?_R (\mathbf{x}, \mathbf{y} \triangleright \mathbf{w})\}$ *where the left premise derivation is* Π_1 *and the right-premise derivation is* Π_2
$-\!\!* L$	$\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \uplus \mathcal{C}(\Pi_2) \uplus \{\mathcal{G} \vdash^?_R (\mathbf{x}, \mathbf{w} \triangleright \mathbf{y})\}$ *where the left premise derivation is* Π_1 *and the right-premise derivation is* Π_2
–	*If* Π *ends with any other rule, with premise derivations* $\{\Pi_1, \ldots, \Pi_n\}$*, then* $\mathcal{C}(\Pi) = \mathcal{C}(\Pi_1) \uplus \cdots \uplus \mathcal{C}(\Pi_n)$.

Proof search in the free variable system is a refutation procedure. The ternary relational atoms generated by $\top^* L$, $*L$, and $-\!\!* R$ are the base knowledge about the monoidal semantics; the constraints generated by $*R$ and $-\!\!* L$ give hints on what are further needed to falsify the end sequent. These constraints needs to be "solved" by grounding the free variables to labels. Zero-premise rule applications refute that the ternary relational atoms generated along a branch cannot form a counter-model. The constraints generated by *id* and $\top^* R$ are the conditions that can close a branch whenever they are satisfied.

Each constraint $\mathfrak{c} \in \mathcal{C}(\Pi)$ corresponds to a rule instance $r(\mathfrak{c})$ in Π where $\mathfrak{c}$ is generated. The ordering of the rules in the derivation tree of Π naturally induces a partial order on $\mathcal{C}(\Pi)$. That is, let $\preceq^\Pi$ be an ordering on $\mathcal{C}(\Pi)$ defined as follows: $\mathfrak{c}_1 \preceq^\Pi \mathfrak{c}_2$ iff the conclusion of $r(\mathfrak{c}_1)$ appears in the path from the root sequent to the conclusion of $r(\mathfrak{c}_2)$. Then obviously $\preceq^\Pi$ is a partial order.

Lemma 4.4.1. *Let* Π *be a symbolic derivation. Then* $(\mathcal{C}(\Pi), \preceq^\Pi)$ *is a constraint system. Moreover, if the root sequent is ground, then* $(\mathcal{C}(\Pi), \preceq^\Pi)$ *is well-formed.*

Proof. Each constraint in $\mathcal{C}(\Pi)$ is associated with an instance of a rule in Π; this induces a partial order on the constraints as follows: $c_1 \preceq^\Pi c_2$ iff the rule instance where

c_1 originates from appears in the path from the root to the rule instance where c_2 originates from, in the derivation tree of Π. It is easy to see that this gives us a partial order. The monotonicity of $\preceq^{\Pi}$ is immediate. The unique variable origin property of $\mathcal{C}(\Pi)$ is also satisfied by the fact that new variables can only be created at $*R$ and $-\!\!* L$. So the minimum constraint for each variable is the constraint generated by the $*R$ or $-\!\!* L$ rule instance where these variables are created. □

Given a symbolic derivation Π, define $\mathbb{C}(\Pi)$ as the constraint system $(\mathcal{C}(\Pi), \preceq^{\Pi})$. A consequence of Lemma 4.4.1 is that if $\mathbb{C}(\Pi) \neq \{\ \}$, then there exists a minimum constraint $\mathfrak{c}$, w.r.t. the partial order $\preceq^{\Pi}$, such that $\mathcal{G}(\mathfrak{c})$ is ground.

We now define what it means for a constraint system to be solvable. The complication arises when we need to capture that (ternary) relational atoms created by the solution need to be accumulated along each branch in the proof search in order to guarantee the soundness of $FVLS_{BBI}$. A *free-variable substitution* θ is a total mapping with finite domain[4] from free variables to free variables or labels. We denote with $dom(\theta)$ the domain of θ. Given θ, θ' with $dom(\theta') \subseteq dom(\theta)$, and a set V of free variables, $\theta \uparrow V$ is the substitution obtained from θ by restricting the domain to V; and $\theta \setminus \theta'$ is the result of θ "subtract" θ'; formally:

$$\mathbf{x}(\theta \uparrow V) = \begin{cases} \mathbf{x}\theta & \text{if } \mathbf{x} \in V \\ \mathbf{x} & \text{otherwise.} \end{cases} \qquad \mathbf{x}(\theta \setminus \theta') = \begin{cases} \mathbf{x}\theta & \text{if } \mathbf{x} \notin dom(\theta') \\ \mathbf{x} & \text{otherwise.} \end{cases}$$

Definition 4.4.3 (Simple constraints and their solutions)**.** *A constraint* $\mathfrak{c}$ *is* simple *if its left hand side* $\mathcal{G}(\mathfrak{c})$ *contains no free variables. A* solution (θ, σ) *to a simple constraint* $\mathfrak{c}$ *is a substitution* θ *and a sequence* σ *of structural rules such that:*

- *If* $\mathfrak{c}$ *is* $\mathcal{G} \vdash_R^? (\mathbf{u} = \mathbf{v})$ *then* σ *is a derivation of* $\mathcal{G} \vdash_R (\mathbf{u}\theta = \mathbf{v}\theta)$.
- *If* $\mathfrak{c}$ *is* $\mathcal{G} \vdash_R^? (\mathbf{u}, \mathbf{v} \triangleright \mathbf{w})$ *then* σ *is a derivation of* $\mathcal{G} \vdash_R (\mathbf{u}\theta, \mathbf{v}\theta \triangleright \mathbf{w}\theta)$.

Lemma 4.4.2. *In a well-formed constraint system* $(\mathcal{C}, \preceq)$*, the minimum constraints in* $\mathcal{C}$ *are simple constraints.*

Proof. Suppose otherwise, i.e., there exists a minimum $\mathfrak{c} \in \mathcal{C}$ such that $\mathcal{G}(\mathfrak{c})$ contains a free variable $\mathbf{x}$. By the unique variable origin property (Definition 4.4.1), there exists $\mathfrak{c}(\mathbf{x})$ such that $\mathbf{x}$ is not free in $\mathcal{G}(\mathfrak{c}(x))$ and $\mathfrak{c}(\mathbf{x}) \preceq \mathfrak{c}$. The former rules out the case that $\mathfrak{c}(x) = \mathfrak{c}$, so we have $\mathfrak{c}(\mathbf{x}) \prec \mathfrak{c}$, which contradicts the minimality of $\mathfrak{c}$. Therefore $\mathcal{G}(\mathfrak{c})$ must be ground. □

From the above definition, a simple constraint $\mathcal{G} \vdash_R^? (\mathbf{u} = \mathbf{v})$ is solvable if there is a series of structural rule applications on $\mathcal{G}$ and a series of free variable substitutions θ, such that $\mathbf{u}\theta = \mathbf{v}\theta$. Since $\mathcal{G}$ only contains labels, the structural rule applications in this case are usually not needed. For example, the simple constraint

$$(w_1, w_2 \triangleright w_3) \vdash_R^? (w_1 = \mathbf{x})$$

[4]Here the substitution θ can be viewed as a *set* of substitutions.

is solvable by applying no structural rules and only assigning $\mathbf{x}$ to w_1 in the free variable substitution θ. The case for a ternary relation on the r.h.s. of $\vdash^?_R$ is defined similarly, but this case often involves structural rule applications. For example, the simple constraint

$$(w_1, w_2 \rhd w_0); (w_3, w_4 \rhd w_1) \vdash^?_R (w_4, w_2 \rhd \mathbf{x})$$

can be solved by letting σ be A followed by E as in the following structural rule applications:

$$\dfrac{\dfrac{(w_4, w_2 \rhd w_5); \cdots}{(w_3, w_5 \rhd w_0); (w_2, w_4 \rhd w_5); \cdots}\,E}{(w_1, w_2 \rhd w_0); (w_3, w_4 \rhd w_1); \cdots}\,A$$

And the free variable substitution θ would be $[w_5/\mathbf{x}]$.

Definition 4.4.4 (Restricting a constraint system)**.** *Let $\mathbb{C} = (\mathcal{C}, \preceq)$ be a well-formed constraint system, and $\mathfrak{c}$ be a minimum (simple) constraint occurrence in $\mathbb{C}$. Let (θ, σ) be a solution to $\mathfrak{c}$ and $\mathcal{G}' = \mathcal{S}(\mathcal{G}(\mathfrak{c}), \sigma)$. Define a function f on constraints:*

$$f(\mathfrak{c}') = \begin{cases} (\mathcal{G}' \cup \mathcal{G}\theta \vdash^?_R C\theta) & \text{if } \mathfrak{c}' = (\mathcal{G} \vdash^?_R C) \in \mathcal{C} \setminus \{\mathfrak{c}\} \text{ and } \mathfrak{c} \preceq \mathfrak{c}', \\ \mathfrak{c}' & \text{otherwise.} \end{cases}$$

The restriction of $\mathbb{C}$ *by* $(\mathfrak{c}, \theta, \sigma)$*, written* $\mathbb{C} \uparrow (\mathfrak{c}, \theta, \sigma)$*, is the pair* $(\mathcal{C}', \preceq')$*, where*

1. $\mathcal{C}' = \{f(\mathfrak{c}') \mid \mathfrak{c}' \in \mathcal{C} \setminus \{\mathfrak{c}\}\}$ *and*
2. $f(\mathfrak{c}_1) \preceq' f(\mathfrak{c}_2)$ *iff* $\mathfrak{c}_1 \preceq \mathfrak{c}_2$.

The proof of the following lemma is straightforward from Definition 4.4.4.

Lemma 4.4.3. *The restricted constraint system $\mathbb{C} \uparrow (\mathfrak{c}, \theta, \sigma)$ as defined in Definition 4.4.4 is a well-formed constraint system.*

Proof. It is easy to see that monotonicity is preserved by the restriction, since $\mathcal{G}'$ is accumulated on every constraint $\mathfrak{c}'$ in the restricted system with $\mathfrak{c} \preceq \mathfrak{c}'$, and the constraint system $\mathbb{C}$ is originally well-formed. For the unique variable origin property, first we notice that the free variables that originate from $\mathfrak{c}$ do not occur in the restricted system, because the solution (θ, σ) to $\mathfrak{c}$ must have instantiated those free variables to labels (since the l.h.s. of $\mathfrak{c}$ only contains labels), and the substitution θ is applied on every constraint where those free variables may occur, i.e., every $\mathfrak{c}'$ such that $\mathfrak{c} \preceq \mathfrak{c}'$. For the other free variables, their origins remain unchanged in the restricted system. □

Definition 4.4.5 (Solution to well-formed constraint systems)**.** *Let $\mathbb{C} = (\{\mathfrak{c}_1, \ldots, \mathfrak{c}_n\}, \preceq)$ be a well-formed constraint system. A* solution $(\theta, \{\sigma_1, \ldots, \sigma_n\})$ *to $\mathbb{C}$ is a substitution and a set of sequences of structural rules, such that:*

If $n = 0$ *then* $([], \{\})$ *is trivially a solution.*

Let $\mathcal{G} = \{(a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1)\}$ and $\Gamma_1 := \{a_2 : r \ ; \ a_3 : p\}$ and $\Gamma_2 := \{a_3 : p \ ; \ a_4 : q\}$ in

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{G}||a_2 : r; a_3 : p; a_4 : q \vdash_5 \mathbf{x}_5 : p}\, id \quad \dfrac{\dfrac{}{\mathcal{G}||\Gamma_1; a_4 : q \vdash_4 \mathbf{x}_7 : q}\, id \quad \dfrac{}{\mathcal{G}||a_2 : r; \Gamma_2 \vdash_3 \mathbf{x}_8 : r}\, id}{\mathcal{G}||a_2 : r; a_3 : p; a_4 : q \vdash_2 \mathbf{x}_6 : q * r}\, *R}{\mathcal{G}||a_2 : r; a_3 : p; a_4 : q \vdash_1 a_0 : p * (q * r)}\, *R}{(a_1, a_2 \triangleright a_0)||a_1 : p * q; a_2 : r \vdash a_0 : p * (q * r)}\, *L}{a_0 : (p * q) * r \vdash a_0 : p * (q * r)}\, *L}{\vdash a_0 : ((p * q) * r) \to (p * (q * r))} \to R$$

Figure 4.7: A symbolic derivation for $((p * q) * r) \to (p * (q * r))$.

If $n \geq 1$ *then there must exist some minimum (simple) constraint in* $\mathbb{C}$. *For any minimum constraint* $\mathfrak{c}_i$, *let* $\theta_i = \theta \uparrow fv(\mathfrak{c}_i)$, *then* (θ_i, σ_i) *is a solution to* $\mathfrak{c}_i$, *and* $(\theta \setminus \theta_i, \{\sigma_1, \ldots, \sigma_n\} \setminus \sigma_i)$ *is a solution to* $\mathbb{C} \uparrow (\mathfrak{c}_i, \theta_i, \sigma_i)$.

If a constraint system $\mathbb{C} = (\{\mathfrak{c}_1, \cdots, \mathfrak{c}_n\}, \preceq)$ has a solution $(\theta, \{\sigma_1, \cdots, \sigma_n\})$, then there is a solution to each $\mathfrak{c}_i$, which is computed recursively from the minimum constraints in $\mathbb{C}$, as defined in Definition 4.4.5. It is easy to see that every $\mathfrak{c} \in \{\mathfrak{c}_1, \cdots, \mathfrak{c}_n\}$ has a solution of the form $(\theta_{\mathfrak{c}}, \sigma)$, for some $\theta_{\mathfrak{c}}$, where $\sigma \in \{\sigma_1, \ldots, \sigma_n\}$. Moreover, σ is uniquely related to $\mathfrak{c}$. In the following, given such a constraint $\mathfrak{c}$, we shall write $dev(\mathfrak{c})$ to refer to that sequence of rules σ in its solution.

Example 4.4.4. *We now give an example of how to prove a formula using the free variable system. Suppose we want to prove* $((p * q) * r) \to (p * (q * r))$, *where* p, q *and* r *are propositional variables. Using* $FVLS_{BBI}$, *we build a symbolic derivation as in Figure 4.7. Note that this derivation is generated automatically from our theorem prover which uses* $a_0, a_1, \cdots$ *for label variables, hence the end-sequent is of the form* $a_0 : F$ *where* a_0 *stands for the label variable* w *and some of the indices of variables may not be contiguous. We subscript* $\vdash$ *with a number* i *to indicate that the rule applied on this sequent generates the constraint* $\mathfrak{c}_i$. *The set of constraints* $\mathcal{C} = \{\mathfrak{c}_1, \cdots, \mathfrak{c}_5\}$ *are generated from this derivation as below:*

$$\begin{array}{lllll}
\mathfrak{c}_5: & (a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1) & \vdash_R & (a_3 = \mathbf{x}_5) \\
\mathfrak{c}_4: & (a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1) & \vdash_R & (a_4 = \mathbf{x}_7) \\
\mathfrak{c}_3: & (a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1) & \vdash_R & (a_2 = \mathbf{x}_8) \\
\mathfrak{c}_2: & (a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1) & \vdash_R & (\mathbf{x}_7, \mathbf{x}_8 \triangleright \mathbf{x}_6) \\
\mathfrak{c}_1: & (a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1) & \vdash_R & (\mathbf{x}_5, \mathbf{x}_6 \triangleright a_0).
\end{array}$$

The partial order $\preceq$ *on these constraints is based on the order of rule applications in the symbolic derivation in Figure 4.7. Thus we obtain that* $\mathfrak{c}_1 \preceq \mathfrak{c}_2$, $\mathfrak{c}_1 \preceq \mathfrak{c}_5$, $\mathfrak{c}_2 \preceq \mathfrak{c}_3$, *and* $\mathfrak{c}_2 \preceq \mathfrak{c}_4$. *By Lemma 4.4.1, the constraint system* $(\mathcal{C}, \preceq)$ *is well-formed.*

The naive way to solve a constraint system would start by solving the minimum constraint (i.e., $\mathfrak{c}_1$*), then solve the other constraints in the partial order* $\preceq$. *Finally, if the constraints generated by zero-premise rules can be solved based on the existing free variable substitutions, then we obtain a solution to the constraint system; otherwise we have to backtrack and try to solve previous constraints using different structural rule applications and/or free variable*

substitutions.

We have not introduced a method to solve the constraints yet, the next section will give a heuristic method. For now let us just non-deterministically guess the solutions, starting from the minimum constraint $\mathfrak{c}_1$*. If we were to solve* $\mathfrak{c}_1$ *with the solution* $(\{\mathbf{x}_5 \mapsto a_0, \mathbf{x}_6 \mapsto a_1 1\}, \{\})$*, then we would have trouble when solving the constraint* $\mathfrak{c}_5$*, because* $(a_3 = a_0)$ *cannot be derived by any structural rule applications. Backtracking to* $\mathfrak{c}_1$*, suppose the oracle says we should apply structural rules on* $\mathcal{G}(\mathfrak{c}_1)$ *as below:*

$$\frac{(a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1)}{(a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1)}\ {\scriptstyle A}$$

where w *is a fresh label created by the* A *application. Then* $\mathfrak{c}_1$ *can be solved by the solution:* $(\{\mathbf{x}_5 \mapsto a_3, \mathbf{x}_6 \mapsto w\}, \{A(\{(a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1)\}, [], \{w\}, \{(a_3, w \rhd a_0); (a_2, a_4 \rhd w)\})\})$*. By Def. 4.4.4, this solution restricts the constraints system as follows:*

$$\begin{array}{lllll}
\mathfrak{c}_5: & (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_3 = a_3) \\
\mathfrak{c}_4: & (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_4 = \mathbf{x}_7) \\
\mathfrak{c}_3: & (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_2 = \mathbf{x}_8) \\
\mathfrak{c}_2: & (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (\mathbf{x}_7, \mathbf{x}_8 \rhd w).
\end{array}$$

Next step is to use the rule E *to create* $(a_4, a_2 \rhd w)$ *from* $(a_2, a_4 \rhd w)$*. The constraint* $\mathfrak{c}_2$ *can then be solved by the solution:*

$$(\{\mathbf{x}_7 \mapsto a_4, \mathbf{x}_8 \mapsto a_2\}, \{E(\{(a_2, a_4 \rhd w)\}, [], \{\}, \{(a_4, a_2 \rhd w)\})\}).$$

The constraint system is now:

$$\begin{array}{lllll}
\mathfrak{c}_5: & (a_4, a_2 \rhd w); (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_3 = a_3) \\
\mathfrak{c}_4: & (a_4, a_2 \rhd w); (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_4 = a_4) \\
\mathfrak{c}_3: & (a_4, a_2 \rhd w); (a_3, w \rhd a_0); (a_2, a_4 \rhd w); (a_1, a_2 \rhd a_0); (a_3, a_4 \rhd a_1) & \vdash_R & (a_2 = a_2).
\end{array}$$

The remaining constraints can be trivially solved by the solution $(\{\}, \{\})$*. Since the constraint system is solvable, the symbolic derivation in Figure 4.7 corresponds to a valid proof of the formula by the coming soundness and completeness theorems. The reader can check that the above solutions satisfy the conditions in Def. 4.4.5.*

Theorem 4.4.5 (Soundness)**.** *Let* Π *be a symbolic derivation of a ground sequent* $\mathcal{G}||\Gamma \vdash \Delta$*. If* $\mathcal{C}(\Pi)$ *is solvable, then* $\mathcal{G}||\Gamma \vdash \Delta$ *is derivable in* LS^{sf}_{BBI}*.*

Proof. By induction on the height n of derivation Π. The basic idea of the proof is that one progressively "grounds" a symbolic derivation, starting from the root of the derivation. At each inductive step we show that grounding the premises corresponds to restricting the constraint system induced by the symbolic derivation.

Base case: $n = 1$. In this case, we can only use a zero-premise rule to prove the sequent. Since the sequent is ground, there are no free variables. Thus the constraint generated by the rule application is a simple constraint of the form $\mathcal{G} \vdash^?_R (a = b)$ or $\mathcal{G} \vdash^?_R (a = \epsilon)$. A solution of this constraint is simply a derivation σ of $\mathcal{G} \vdash_R (a = b)$

(resp. $\mathcal{G} \vdash (a = \epsilon)$. In either case, this translates straightforwardly into a derivation in LS^{sf}_{BBI} with the same rule.
Inductive case: $n > 1$. This can be done by a case analysis of the last rule application in Π. We demonstrate the case for $*R$, where a constraint is generated. The case for $-\!\!* L$ is analogous, and the other cases are easy since we can use the induction hypothesis directly. Suppose Π runs as follows.

$$\frac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \mathcal{G}||\Gamma \vdash \mathbf{x} : A; w : A * B; \Delta & \mathcal{G}||\Gamma \vdash \mathbf{y} : B; w : A * B; \Delta\end{array}}{\mathcal{G}||\Gamma \vdash w : A * B; \Delta}\ {}_{*R}$$

Assume that $\mathbb{C}(\Pi) = (\{\mathfrak{c}_1, \ldots, \mathfrak{c}_k\}, \preceq)$, for some $k \geq 1$, and the constraint generated by this rule application is $\mathcal{G} \vdash^{?}_{R} (\mathbf{x}, \mathbf{y} \rhd w)$ and it corresponds to $\mathfrak{c}_i$ for some $i \in \{1, \ldots, k\}$. By the assumption, there is a solution $(\theta, \{\sigma_1, \cdots, \sigma_k\})$ for the constraint system $\mathbb{C} = (\mathcal{C}(\Pi), \preceq^{\Pi})$. Now $\mathfrak{c}_i$ must be a simple constraint in $\mathbb{C}$, as the end sequent is ground. Let (θ_i, σ_i) be the solution to $\mathfrak{c}_i$, where θ_i is a restriction to θ containing $\mathbf{x}$ and $\mathbf{y}$, and $\sigma_i \in \{\sigma_1, \cdots, \sigma_k\}$. By definition of the solution to a simple constraint, σ_i is a derivation of $\mathcal{G} \vdash_R (\mathbf{x}\theta_i, \mathbf{y}\theta_i \rhd w)$. Therefore in LS^{sf}_{BBI}, to derive the end sequent, we apply $*R$ backwards:

$$\frac{\mathcal{S}(\mathcal{G}, \sigma_i)||\Gamma \vdash \mathbf{x}\theta_i : A; w : A * B; \Delta \qquad \mathcal{S}(\mathcal{G}, \sigma_i)||\Gamma \vdash \mathbf{y}\theta_i : B; w : A * B; \Delta}{\mathcal{G}||\Gamma \vdash w : A * B; \Delta}\ {}_{*R}$$

The condition on this rule is $\mathcal{G} \vdash_R (\mathbf{x}\theta_i, \mathbf{y}\theta_i \rhd w)$. Now we construct the derivation for both branches in the following way. Firstly we substitute $\mathbf{x}$ and $\mathbf{y}$ with $\mathbf{x}\theta_i$ and $\mathbf{y}\theta_i$ respectively in Π_1 and Π_2, making the end sequents in the two derivations ground. Let us refer to the modified derivations as Π'_1 and Π'_2 respectively. Then we add $\mathcal{S}(\mathcal{G}, \sigma_i)$ to the left hand side of each sequent in Π'_1 and Π'_2 and each constraint in $\mathcal{C}(\Pi'_1) \uplus \mathcal{C}(\Pi'_2)$. Let the resultant derivations be Π''_1 and Π''_2 respectively. Now the end sequents of Π''_1 and Π''_2 are respectively just the same as the two branches we created in the LS^{sf}_{BBI} derivation. Moreover, each constraint in $\mathcal{C}(\Pi''_1) \uplus \mathcal{C}(\Pi''_2)$ is in the restricted constraint system $\mathbb{C}' = (\mathcal{C}', \preceq') = (\mathcal{C}(\Pi), \preceq^{\Pi}) \uparrow (\mathfrak{c}_i, \theta_i, \sigma_i)$, which has a solution $(\theta \setminus \theta_i, \{\sigma_1, \cdots, \sigma_k\} \setminus \sigma_i)$, and obeys the partial order $\preceq'$. Further, as Π''_1 (resp. Π''_2) uses the same rule applications as in Π_1 (resp. Π_2), the order of constraints is preserved. That is, in the constraints system $\mathbb{C}_1 = (\mathcal{C}(\Pi''_1), \preceq^{\Pi''_1})$ (resp. $\mathbb{C}_2 = (\mathcal{C}(\Pi''_2), \preceq^{\Pi''_2})$), if $\mathfrak{c} \preceq^{\Pi''_1} \mathfrak{c}'$ (resp. $\mathfrak{c} \preceq^{\Pi''_2} \mathfrak{c}'$) then $\mathfrak{c} \preceq' \mathfrak{c}'$ in $\mathbb{C}'$. Therefore we can construct the solution (θ''_1, Σ_1) to $\mathbb{C}_1$ (and analogously to $\mathbb{C}_2$) as follows.

$$\theta''_1 = (\theta \setminus \theta_i) \uparrow fv(\mathcal{C}(\Pi''_1))$$
$$\Sigma_1 = \{\sigma \mid \mathfrak{c} \in \mathcal{C}(\Pi''_1), \sigma \in \{\sigma_1, \cdots, \sigma_k\} \setminus \sigma_i, \textit{ and } \sigma = dev(\mathfrak{c})\}.$$

By the induction hypothesis, we can obtain a LS^{sf}_{BBI} derivation for each branch. □

To prove the completeness of $FVLS_{BBI}$, we show that for every cut-free derivation Π of a (ground) sequent in LS^{sf}_{BBI}, there is a symbolic derivation Π' of the same sequent such that $\mathcal{C}(\Pi')$ is solvable. It is quite obvious that Π' should have exactly the same rule applications as Π; the only difference is that some relational atoms are omitted in the derivation, but instead are accumulated in the constraint system. Additionally, some (new) labels are replaced with free variables. This is formalised as below.

Definition 4.4.6. *Given a sequent in a LS^{sf}_{BBI} derivation, let $\mathcal{G}$ be the set of its relational atoms. We define $\mathcal{G}_E$ as the subset of $\mathcal{G}$ that contains those ternary relational atoms created by $*L$, $-\!\!* R$, and $\top^* L$. We define $\mathcal{G}_S = \mathcal{G} \setminus \mathcal{G}_E$. We refer to $\mathcal{G}_E$ as the **essential** subset of $\mathcal{G}$, and $\mathcal{G}_S$ as the **supplementary** subset of $\mathcal{G}$.*

For a list L, we denote by $head(L)$ the first element in the list L and $tail(L)$ the list of L without the first element, and $end(L)$ the last element in L. We denote by $L_1@L_2$ the concatenation of two lists L_1 and L_2, and $pre(x)$ the predecessor of x in a list L, and $suc(x)$ the successor of x in L.

Given a well-formed constraint system $(\mathcal{C}, \preceq)$, we can define a partial order $\preceq^v$ on free variables of $\mathcal{C}$ as follows: $\mathbf{x} \preceq^v \mathbf{y}$ iff $\mathfrak{c}(\mathbf{x}) \preceq \mathfrak{c}(\mathbf{y})$. That is, free variables are ordered according to their origins. The relations $\prec^v$ and $\lessdot^v$ are defined analogously to $\prec$ and $\lessdot$, i.e., as the non-reflexive subset of $\preceq^v$ and the successor relation.

Definition 4.4.7 (A thread of variables)**.** *Let $\mathbb{C} = (\mathcal{C}, \preceq)$ be a well-formed constraint system, and let X be a list of free variables $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where $n \geq 0$. Let $\preceq^v$ be the partial order on variables, derived from $\preceq$. We say X is a* thread of free variables of $\mathbb{C}$ *(or simply a* thread *of $\mathbb{C}$) iff it satisfies the following conditions:*

1. $\forall \mathbf{x} \in X$, $\mathbf{x} \in fv(\mathcal{C})$
2. *For every* $i \in \{1, \ldots, n-1\}$, $\mathbf{x}_i \lessdot^v \mathbf{x}_{i+1}$
3. *If $n \geq 1$, then $\mathbf{x}_1$ is a minimum element and $\mathbf{x}_n$ is a maximum element of $\preceq^v$*
4. *If $n \geq 1$, then $\mathfrak{c}(\mathbf{x}_1)$ is a minimum constraint in $\mathcal{C}$.*

A thread is effectively those variables that are generated along a certain branch in a $FVLS_{BBI}$ symbolic derivation. It is not hard to verify that in a valid symbolic derivation in $FVLS_{BBI}$ of a ground sequent, the set of free variables in any symbolic sequent in the derivation can be linearly ordered as a thread.

Definition 4.4.8. *Let $\mathbb{C} = (\mathcal{C}_1, \preceq_1)$ be a well-formed constraint system, let X be a thread of $\mathbb{C}_1$, and $\mathbb{C}_2 = (\mathcal{C}_2, \preceq_2)$ be a constraint system that satisfies monotonicity but may not be well-formed, and that X consists of free variables in $fv(\mathcal{C}_1) \cap fv(\mathcal{C}_2)$. Assume that every variable $\mathbf{x}$ in $\mathcal{C}_2$ but not in X satisfies the unique variable origin property, i.e., $\mathbf{x}$ originates from a constraint in $\mathcal{C}_2$; while every variable $\mathbf{y}$ in $\mathcal{C}_2$ and in X does not have an origin in $\mathcal{C}_2$. The* composition *of $\mathbb{C}_1$ and $\mathbb{C}_2$ along the thread X, written $\mathbb{C}_1 \circ^X \mathbb{C}_2$, is the constraint system $(\mathcal{C}, \preceq)$ such that:*

- $\mathcal{C} = \mathcal{C}_1 \uplus \mathcal{C}_2$*; and*

- *Define a relation $\mathcal{R}$ as follows: for $\mathfrak{c}_1, \mathfrak{c}_2 \in \mathcal{C}$, $\mathfrak{c}_1 \mathcal{R} \mathfrak{c}_2$ iff either one of the following holds:*
  - $\mathfrak{c}_1 \preceq_1 \mathfrak{c}_2$,
  - $\mathfrak{c}_1 \preceq_2 \mathfrak{c}_2$, *or*
  - *X is non-empty, $\mathbf{y} = end(X)$, $\mathfrak{c}_1 = \mathfrak{c}(\mathbf{y})$, $\mathfrak{c}_2 \in \mathcal{C}_2$, and the left hand side of $\mathfrak{c}_1$ is a subset of the left hand side of $\mathfrak{c}_2$.*

  Then define $\preceq$ to be the transitive closure of $\mathcal{R}$.

This definition basically says that the composition of $\mathbb{C}_1$ and $\mathbb{C}_2$ along X is obtained by simply ordering the constraints so that all constraints $\mathcal{C}_2$ are greater than $\mathfrak{c}(\mathbf{y})$, where $\mathbf{y}$ is the last variable in X. If X is empty, then $\mathcal{C}_1$ and $\mathcal{C}_2$ are independent, and $\preceq$ is simply the union of $\preceq_1$ and $\preceq_2$.

The next two lemmas follow from the definitions above.

Lemma 4.4.6. *Let $(\mathcal{C}, \preceq)$ be as defined in Definition 4.4.8. Then $(\mathcal{C}, \preceq)$ is well-formed.*

Proof. The case when X is empty is trivial, because in that case $\mathbb{C}_1$ and $\mathbb{C}_2$ are both well-formed and independent, so their union satisfies monotonicity and unique variable origin property. If X is non-empty, monotonicity holds because whenever $\mathfrak{c}_1 \preceq \mathfrak{c}_2$ and $\mathfrak{c}_1 \in \mathcal{C}_1$ and $\mathfrak{c}_2 \in \mathcal{C}_2$, the left hand side of $\mathfrak{c}_1$ is a subset of the left hand side of $\mathfrak{c}_2$. Unique variable origin also holds because (1) all the free variables in $\mathcal{C}_1$, including those in X, have unique origins in $\mathcal{C}_1$; (2) every free variable in $\mathcal{C}_2$ and also in X does not have an origin in $\mathcal{C}_2$; and finally, (3) every free variable in $\mathcal{C}_2$ but is not in X has a unique origin in $\mathcal{C}_2$. □

Lemma 4.4.7. *Let $\mathbb{C} = (\mathcal{C}, \preceq)$ be a well-formed constraint system and let X be a thread of $\mathbb{C}$. Let Π be a symbolic derivation such that the free variables in its end sequent are exactly those in X. Then $\mathbb{C} \circ^X \mathbb{C}(\Pi)$ is well-formed.*

Proof. Since Π is a symbolic derivation, $\mathbb{C}(\Pi)$ apparently satisfies monotonicity. The free variables occurring in the end sequent of Π obviously do not have origins in $\mathbb{C}(\Pi)$, and these free variables' origins are only in $\mathcal{C}$. Every free variable in Π but not in X should have a unique origin in $\mathbb{C}(\Pi)$ because Π is a symbolic derivation. Thus by Lemma 4.4.6, $\mathbb{C} \circ^X \mathbb{C}(\Pi)$ is well-formed. □

Definition 4.4.9. *Let $\mathbb{C} = (\mathcal{C}, \preceq)$ be a well-formed constraint system and let $S = (\theta, \{\vec{\sigma}\})$ be its solution. Let X be a thread of $\mathbb{C}$. Define a set of relational atoms $\mathcal{S}^*(\mathbb{C}, S, X)$ inductively by the length n of X as follows:*

- *If $n = 0$ then $\mathcal{S}^*(\mathbb{C}, S, []) = \emptyset$*
- *Suppose $n > 0$. Let $head(X) = \mathbf{x}$. Then $\mathfrak{c}(\mathbf{x}) \in \mathcal{C}$ is a minimum constraint of $\mathbb{C}$, and there exists $\sigma_{\mathbf{x}} \in \{\vec{\sigma}\}$ such that $(\theta_{\mathbf{x}}, \sigma_{\mathbf{x}})$ is a solution to $\mathfrak{c}(\mathbf{x})$, where $\theta_{\mathbf{x}} = \theta \uparrow fv(\mathfrak{c}(\mathbf{x}))$. In this case, $\mathcal{S}^*(\mathbb{C}, S, X)$ is defined as follows.*

$$\mathcal{S}^*(\mathbb{C}, S, X) = \mathcal{S}(\mathcal{G}(\mathfrak{c}(\mathbf{x})), \sigma_{\mathbf{x}}) \cup \mathcal{S}^*(\mathbb{C} \uparrow (\mathfrak{c}(\mathbf{x}), \theta_{\mathbf{x}}, \sigma_{\mathbf{x}}), S', tail(X))$$

where $S' = (\theta \setminus \theta_{\mathbf{x}}, \{\vec{\sigma}\} \setminus \{\sigma_{\mathbf{x}}\})$.

Notice that by the definition of restriction to a constraint system, every time a minimum constraint $\mathfrak{c}_{\mathbf{x}}$ is eliminated in the second clause in the above definition, $\mathcal{S}(\mathcal{G}(\mathfrak{c}_{\mathbf{x}}), \sigma_{\mathbf{x}})$ is also added to the left hand side of every successor constraints of $\mathfrak{c}_{\mathbf{x}}$ in $\mathcal{C}$. Therefore it is straightforward that the following proposition holds.

Proposition 4.4.8. *Let* $\mathbb{C} = (\mathcal{C}, \preceq)$ *be a well-formed constraint system. Let* $\mathcal{G} = \mathcal{S}^*(\mathbb{C}, S, X)$, *for some thread* X *of* $\mathbb{C}$, *let* $\mathbf{x}_e = end(X)$ *and let* $S = (\theta, \{\overrightarrow{\sigma}\})$ *be a solution to* $\mathbb{C}$. *Let* $\mathfrak{c} = \mathcal{G}_{\mathfrak{c}} \vdash^?_R C_{\mathfrak{c}}$ *be a constraint not in* $\mathcal{C}$, *the l.h.s. of* $\mathfrak{c}(\mathbf{x}_e)$ *is a subset of* $\mathcal{G}_{\mathfrak{c}}$, *and* $\mathcal{G}_{\mathfrak{c}}$ *only contains free variables that occur in* $\mathbb{C}$. *Let* $\mathbf{x}$ *be a new variable occurring only on the right hand side of* $\mathfrak{c}$. *Let* $\mathbb{C}' = (\mathcal{C}', \preceq')$ *be the following constraint system:*

- $\mathcal{C}' = \mathcal{C} \uplus \{\mathfrak{c}\}$;
- $\preceq'$ *is the smallest extension of* $\preceq$ *such that* $\mathfrak{c}(\mathbf{x}_e) \lessdot \mathfrak{c}$.

Let (θ_x, σ_x) *be the solution to* $\mathfrak{c}' = \mathcal{G} \cup \mathcal{G}_{\mathfrak{c}}\theta \vdash^?_R C_{\mathfrak{c}}\theta$, $S' = (\theta \cup \theta_x, \{\overrightarrow{\sigma}, \sigma_x\})$, *and* $X' = X@[\mathbf{x}]$. *Then* $\mathcal{S}^*(\mathbb{C}', S', X') = \mathcal{S}(\mathcal{G} \cup \mathcal{G}_{\mathfrak{c}}\theta, \sigma_x)$.

Observe first that $\mathbb{C}'$ is a well-formed constraint system: it satisfies monotonicity because the l.h.s. of $\mathfrak{c}(\mathbf{x}_e)$ is a subset of $\mathcal{G}_{\mathfrak{c}}$; it satisfies the unique variable origin property because every free variable in $\mathcal{G}_{\mathfrak{c}}$ has a unique origin in $\mathcal{C}$, and any free variable in $\mathfrak{c}$ that does not occur in $\mathcal{C}$ (which must occur in $C_{\mathfrak{c}}$) has the unique origin $\mathfrak{c}$. This proposition says that when we have eliminated (solved) all the constraints but the last one $\mathfrak{c}$ in a constraint system $\mathbb{C}'$, we have the current accumulated and substituted l.h.s. $\mathcal{G}$ of eliminated constraints, a set θ of free variable substitutions, and structural rule applications $\overrightarrow{\sigma}$. The only remaining things are to apply θ on $\mathcal{G}_{\mathfrak{c}}$ to eliminate previously solved free variables, take the union $\mathcal{G} \cup \mathcal{G}_{\mathfrak{c}}\theta$, and apply the remaining structural rule applications σ_x to obtain the final set of relational atoms, which by definition is $\mathcal{S}^*(\mathbb{C}', S', X')$.

Finally, we present the completeness proof for the free variable system $FVLS_{BBI}$.

Theorem 4.4.9. *Let* Π *be a derivation of a sequent in* LS^{sf}_{BBI}. *Then there exists a symbolic derivation* Π' *of the same sequent such that* $\mathcal{C}(\Pi')$ *is solvable.*

Proof. We describe the construction from a LS^{sf}_{BBI} derivation Π to a $FVLS_{BBI}$ derivation Π'. We need to prove a stronger invariant: for each sequent $\mathcal{G}_E; \mathcal{G}_S || \Gamma \vdash \Delta$ in Π, if there exists a triple consisting of:

(1) a symbolic sequent $\mathcal{G}'_E || \Gamma' \vdash \Delta'$,
(2) a well-formed constraint system $\mathbb{C} = (\mathcal{C}, \preceq)$,
(3) and a solution $S = (\theta, \{\overrightarrow{\sigma}\})$ to $\mathbb{C}$

such that

(I) there exists a thread X of $\mathbb{C}$ consisting of $fv(\mathcal{G}'_E||\Gamma' \vdash \Delta')$,
(II) $\mathcal{G}'_E\theta = \mathcal{G}_E$, $\Gamma'\theta = \Gamma$, $\Delta'\theta = \Delta$ and
(III) $\mathcal{G}_E \cup \mathcal{G}_S = \mathcal{S}^*(\mathbb{C}, S, X)$,

then there is a symbolic derivation Ψ of $\mathcal{G}'_E||\Gamma' \vdash \Delta'$ such that $\mathbb{C} \circ^X \mathbb{C}(\Psi)$ is well-formed and solvable.

First of all, by Lemma 4.4.7, since the end sequent in Ψ only contains the free variables occurring in X, the composition $\mathbb{C} \circ^X \mathbb{C}(\Psi)$ must be well-formed. Further, any minimum constraint $\mathfrak{c}$ in Ψ follows $end(X)$ in the partial order $\preceq$ of the composed constraint system since the l.h.s. of $\mathfrak{c}$ must include the l.h.s. of $end(X)$. Thus we only need to show that there is a solution to the composed constraint system. We prove this by a case analysis on the last rule in Π, and show that in each case, for each premise of the rule, one can find a triple satisfying the above property, such that the symbolic sequent(s) in the premise(s), together with the one in the conclusion form a valid inference in $FVLS_{BBI}$. We illustrate here with a non-trivial case when Π ends with $*R$. Suppose the derivation Π runs as:

$$\frac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \mathcal{S}((\mathcal{G}_E;\mathcal{G}_S),\sigma)||\Gamma \vdash w_1 : A; w : A * B; \Delta & \mathcal{S}((\mathcal{G}_E;\mathcal{G}_S),\sigma)||\Gamma \vdash w_2 : B; w : A * B; \Delta\end{array}}{\mathcal{G}_E;\mathcal{G}_S||\Gamma \vdash w : A * B; \Delta}\ *R$$

and the relational entailment is $\mathcal{G}_E;\mathcal{G}_S \vdash_R (w_1, w_2 \triangleright w)$. Suppose that the relational atoms in the premises are derived via σ. Suppose further that we can find a triple consisting of (1) a symbolic sequent $\mathcal{G}'_E||\Gamma' \vdash \mathbf{w} : A * B; \Delta'$, (2) a well-formed constraint system $\mathbb{C} = (\mathcal{C}, \preceq)$, and (3) a solution $S = (\theta, \{\sigma_1, \ldots, \sigma_n\})$ to $\mathbb{C}$, satisfying the following: (I) X is a thread of $\mathbb{C}$ consisting of $fv(\mathcal{G}'_E||\Gamma' \vdash \mathbf{w} : A * B; \Delta')$, (II) $\mathcal{G}'_E\theta = \mathcal{G}_E$, $\Gamma'\theta = \Gamma$, $\Delta'\theta = \Delta$, $w = \mathbf{w}\theta$, and (III) $\mathcal{G}_E \cup \mathcal{G}_S = \mathcal{S}^*(\mathbb{C}, S, X)$. We need to show that we can find such triples for the premises, and more importantly, the symbolic sequents in the premises are related to the symbolic sequent in the conclusion via $*R$. In this case, the symbolic sequents are simply the following:

1. $\mathcal{G}'_E||\Gamma' \vdash \mathbf{x} : A; \mathbf{w} : A * B; \Delta'$, for the left premise,
2. $\mathcal{G}'_E||\Gamma' \vdash \mathbf{y} : A; \mathbf{w} : A * B; \Delta'$, for the right premise.

The constraint systems are: $\mathbb{C}' = (\mathcal{C} \uplus \{\mathfrak{c}_j\}, \preceq')$ for both premises, where $\mathfrak{c}_j = \mathcal{G}'_E \vdash^?_R (\mathbf{x}, \mathbf{y} \triangleright \mathbf{w})$ and $\preceq'$ is $\preceq$ extended with $\mathfrak{c}(end(X)) \preceq' \mathfrak{c}_j$. The solutions, for both premises, are the tuple $S' = (\theta', \Sigma)$ where $\theta' = \theta \cup \{\mathbf{x} \mapsto w_1,\ \mathbf{y} \mapsto w_2\}$ and $\Sigma = \{\sigma_1, \ldots, \sigma_n, \sigma\}$. It is guaranteed that θ' is enough to make both premises grounded, as $\mathbf{x}$ and $\mathbf{y}$ are the only two new free variables. The threads of free variables X_1 and X_2 for the two premises are naturally $X@[\mathbf{x}]$ and $X@[\mathbf{y}]$ respectively. By Proposition 4.4.8, in each premise, the following holds:

$$\mathcal{G}_E \cup \mathcal{G}'_S = \mathcal{S}(\mathcal{G}_E \cup \mathcal{G}_S, \sigma) = \mathcal{S}(\mathcal{G}_E \cup \mathcal{G}_S \cup \mathcal{G}'_E\theta, \sigma) = \mathcal{S}^*(\mathbb{C}', S', X_1) = \mathcal{S}^*(\mathbb{C}', S', X_2).$$

So by the induction hypothesis we have a symbolic derivation Π'_1 for sequent (1) and a symbolic derivation Π'_2 for sequent (2), such that $\mathbb{C}_{\beta 1} = \mathbb{C}' \circ^{X_1} \mathbb{C}(\Pi'_1)$ and $\mathbb{C}_{\beta 2} = \mathbb{C}' \circ^{X_2} \mathbb{C}(\Pi'_2)$ are both solvable. Suppose the solutions are respectively $(\theta' \cup \theta_1, \Sigma \cup \Sigma_1)$ and $(\theta' \cup \theta_2, \Sigma \cup \Sigma_2)$. Then construct Π' by applying the $*R$ rule to Π'_1 and Π'_2. Note that the variables created in Π'_1 are Π'_2 are distinct so their constraints are independent of each other. So we can construct $\mathbb{C}_p = \mathbb{C}(\Pi'_1) \circ^{\emptyset} \mathbb{C}(\Pi'_2) = (\mathcal{C}_p, \preceq_p)$, along an empty thread $\emptyset$. Now $\mathbb{C}(\Pi')$ is obtained as $(\mathcal{C}_p \uplus \{\mathfrak{c}_j\}, \preceq^{\Pi'})$, where $\preceq^{\Pi'}$ is derived as follows.

- If $\mathfrak{c} \preceq_p \mathfrak{c}'$ in $\mathbb{C}_p$, then $\mathfrak{c} \preceq^{\Pi'} \mathfrak{c}'$ in $\mathbb{C}(\Pi')$
- For any minimum constraint $\mathfrak{c}_m$ in $\mathbb{C}_p$, $\mathfrak{c}_j \preceq^{\Pi'} \mathfrak{c}_m$ in $\mathbb{C}(\Pi')$.

The solution to $\mathbb{C}_\alpha = \mathbb{C} \circ^X \mathbb{C}(\Pi')$ is constructed as the combination of the solutions to $\mathbb{C}_{\beta 1}$ and $\mathbb{C}_{\beta 2}$: $(\theta' \cup \theta_1 \cup \theta_2, \Sigma \cup \Sigma_1 \cup \Sigma_2)$. This construction of the solution is indeed valid, because the symbolic derivation that gives $\mathbb{C}_\alpha$ also yields exactly $\mathbb{C}_{\beta 1}$ and $\mathbb{C}_{\beta 2}$ (respectively on its two branches created by the $*R$ rule). □

4.5 A Heuristic Method for Proof Search

In this section, we first give an example of deriving a formula and solving the generated constraints in $FVLS_{BBI}$ based on a heuristic method, in which constraints generated by zero-premise rules are solved first, then the constraints from logical rules are solved based on what we have gained in the solved constraints. We then extend this idea and formalise it in the remainder of this section.

Consider again the constraint system in Example 4.4.4, which is generated from the symbolic derivation in Figure 4.7. Our heuristic constraint solving method differs from the naive method in two aspects. Firstly, we start by solving the constraints generated by zero-premise rules. Since the constraints $\mathfrak{c}_3, \mathfrak{c}_4, \mathfrak{c}_5$ are required by the *id* rule, we must accept them by assigning $\mathbf{x}_5, \mathbf{x}_7, \mathbf{x}_8$ to a_3, a_4, a_2 respectively. Then we are only left with the constraints $\mathfrak{c}_1$ and $\mathfrak{c}_2$. In the following, we shall write $(a_1, a_2 \triangleright a_0); (a_3, a_4 \triangleright a_1)$ as $\mathcal{G}_1$, and $(a_3, \mathbf{x}_6 \triangleright a_0); (a_2, a_4 \triangleright \mathbf{x}_6)$ as $\mathcal{G}_2$. Now $\mathbf{x}_6$ is the only remaining free variable. We can apply the rule A (upwards) on $\mathcal{G}_1$ to obtain $(a_3, w \triangleright a_0); (a_2, a_4 \triangleright w)$, where w is a new label. Then apply the rule E (upwards) to obtain $(a_4, a_2 \triangleright w)$. The two constraints can be solved by the above derivation and assigning w to $\mathbf{x}_6$.

The second novelty of our heuristic method is that we view a set of relational atoms as trees, and in certain cases, we can solve the constraints by only looking at the root and the leaves, ignoring the internal structure of the trees. For the running example, $\mathcal{G}_1$ is a tree tr_1 with root a_0 and leaves $\{a_2, a_3, a_4\}$, which are exactly the same as the tree tr_2 of $\mathcal{G}_2$, although the internal structures of tr_1 and tr_2 are different. We will show in Lemma 4.5.2 that, since every internal node in tr_2 is a unique free variable, and there are no ϵ labels in tr_2, it is guaranteed that there exists a sequence of structural rule applications on $\mathcal{G}_1$ to obtain a tree tr'_1, which has the same structure as tr_2, but only differing in the labels of internal nodes. Hence we can assign the labels of internal nodes in tr'_1 to the corresponding free variables in tr_2. The labels of nodes in tr'_1 may

be existing labels occurring in tr_1, or fresh labels created by A, A_C applications. We will first try to match the free variables in tr_2 with existing labels in tr_1, if this is not possible, we will assign the free variable to a fresh label. In the example, $\mathbf{x}_6$ cannot be matched to any existing label, so we can globally replace $\mathbf{x}_6$ with a fresh label w, and add $\mathcal{G}_2$ to the left hand side of the successor constraints of $\mathfrak{c}_1$ in the partial order $\preceq$. The advantage of this process is that we do not care about the structural rule applications to obtain tr_1' at all, but we know that the fresh label w as in the previous paragraph must exist and can substitute $\mathbf{x}_6$.

We can extend this method to a chain of multiple relational atoms which form a labelled binary tree.

We define a labelled binary tree as a binary tree where each node is associated with a label. Each node in a labelled binary tree has a left child and a right child. The minimum labelled binary tree has a root and two leaves, which corresponds to a single relational atom. We define the following function inductively from a labelled binary tree to a set of relational atoms.

Definition 4.5.1. *Let tr be a labelled binary tree, the set $Rel(tr)$ of relational atoms w.r.t. tr is defined as follows.*

- *(Base case): tr only contains a root node labelled with r and two leaves labelled with a, b respectively. Then $Rel(tr) = \{(a, b \triangleright r)\}$*
- *(Inductive case): tr contains a root node labelled with r and its left and right children labelled with a and b respectively. Then $Rel(tr) = Rel(tr_a) \cup Rel(tr_b) \cup \{(a, b \triangleright r)\}$, where tr_a and tr_b are the subtrees rooted at, respectively, the left child and the right child of the root node of tr.*

The *width* of a labelled binary tree is defined as the number of leaves in the tree. A labelled binary tree is a *variant* of another labelled binary tree if either they are exactly the same, or they differ only in the labels of the internal nodes.

We say that a set $\mathcal{R}$ of relational atoms *forms a labelled binary tree tr* when $\mathcal{R} = Rel(tr)$. In this case, the leaves in tr are actually a "splitting" of the root node. Commutativity and associativity guarantee that we can split a node arbitrarily, as long as the leaves in the tree are the same. Moreover, since all internal nodes are free variables, we can assign them to either existing labels or fresh labels (created by A, A_C) without clashing with existing relational atoms.

In the following proofs we use the tree representation of a set of relational atoms. Given a labelled binary tree tr as defined above, we say another labelled binary tree tr' is a permutation of tr if they have the same root and same multiset of leaves. A permutation on tr is generally done by applying the rules E, A on $Rel(tr)$. Figure 4.8 gives some examples on tree permutations. In Figure 4.8, (b) is permuted from (a) by using E on $(d, e \triangleright b)$, whereas (c) is permuted from (a) by using A on the two relational atoms in the original tree.

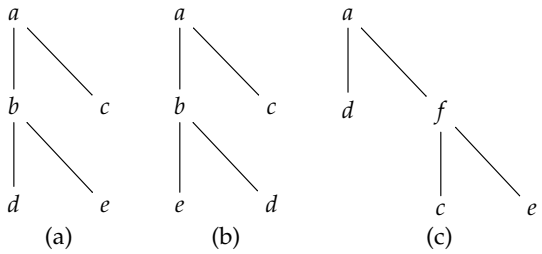

Figure 4.8: Examples of tree permutations.

Lemma 4.5.1. *Let tr be a labelled binary tree with a root labelled with r and a multiset L of labels for the leaves. If there is a labelled binary tree tr' with the same root and leaf labels respectively, then there is a variant tr'' of tr' and a sequence σ of E, A rule applications such that $Rel(tr'') \subseteq \mathcal{S}(Rel(tr), \sigma)$.*

Proof. Prove by induction on the width of the tree tr. We show that any distinct permutation (i.e., they are not variants of each other) of a tree can be achieved by using the rules E and A. Base case is when there are only two leaves in tr. In this case, there is only one relational atom in $Rel(tr)$, thus clearly there is only one distinct permutation of tr, which can be obtained by applying E on $Rel(tr)$.

The next case is when there are 3 leaves in the tree, meaning $Rel(tr)$ contains two relational atoms. In this case, it can be easily checked that there are 12 distinct permutations of tr, all of which can be derived by using E and A.

Inductive case, suppose the lemma holds for all trees with width less than n, consider a tree tr with width n. Suppose further that the root label of tr is r, it's two children are in the relational atom $(w_1, w_2 \triangleright r)$, and the multisets of leaves labels for the subtrees of w_1 and w_2 are L_1, L_2 respectively. Let tr' be a permutation of tr with the same root label and leaves labels, and in tr' the two children of the root label are in the relational atom $(w_3, w_4 \triangleright r)$. Suppose the multisets of leaf labels for the subtrees of w_3, w_4 are L_3, L_4 respectively. Apparently, since $L_1 \cup L_2 = L_3 \cup L_4 = L$, every label in L_3 is either in L_1 or in L_2. Let $L' = L_1 \cap L_3$ and $L'' = L_2 \cap L_3$, then $L' \cup L'' = L_3$ and $(L_1 \setminus L') \cup (L_2 \setminus L'') = L_4$. By the induction hypothesis on the subtrees of w_1 and w_2, there exist w_5, w_6, w_7, w_8 s.t. $(w_5, w_6 \triangleright w_1), (w_7, w_8 \triangleright w_2)$ hold, and the subtrees of w_5, w_6, w_7, w_8 give the multisets of leaves $L', (L_1 \setminus L'), L'', (L_2 \setminus L'')$ respectively. Then we use the following derivation to permute the tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(w'', w''' \rhd r); (w_6, w_8 \rhd w'''); (w_5, w_7 \rhd w''); \cdots}{(w', w_6 \rhd r); (w'', w_8 \rhd w'); (w_5, w_7 \rhd w''); \cdots}\,A}{(w_6, w' \rhd r); (w_8, w'' \rhd w'); (w_5, w_7 \rhd w''); \cdots}\,E\times 2}{(w_6, w' \rhd r); (w_2, w_5 \rhd w'); (w_8, w_7 \rhd w_2); \cdots}\,A}{(w_6, w' \rhd r); (w_2, w_5 \rhd w'); (w_7, w_8 \rhd w_2); \cdots}\,E}{(w_6, w_5 \rhd w_1); (w_7, w_8 \rhd w_2); (w_1, w_2 \rhd r); \cdots}\,A}{(w_5, w_6 \rhd w_1); (w_7, w_8 \rhd w_2); (w_1, w_2 \rhd r); \cdots}\,E$$

Now the subtrees of w'' and w''' has the same multisets of leaves as w_3 and w_4 respectively. Again by the induction hypothesis on the subtrees of w'' and w''', we obtain a tree tr'' which is a variant of tr'. □

Our heuristic method is proved correct as below.

Lemma 4.5.2. *Given constraints $\mathfrak{c}_1 \lessdot \cdots \lessdot \mathfrak{c}_n$ with $\mathcal{G} = \mathcal{G}(\mathfrak{c}_1) = \cdots = \mathcal{G}(\mathfrak{c}_n)$ where the r.h.s. of these constraints gives the set $\mathcal{R}$ of relational atoms, the constraints $\mathfrak{c}_1, \cdots, \mathfrak{c}_n$ are solvable if the following hold:*

1. *$\mathcal{R} = Rel(tr)$, for some labelled binary tree tr where every internal node label is a free variable $\mathbf{x}$ which only occurs once in tr, and $\mathfrak{c}_1 \preceq \mathfrak{c}(\mathbf{x})$.*
2. *The other node labels in tr are non-ϵ labels.*
3. *There exist $\mathcal{G}' \subseteq \mathcal{G}$ and tr' such that $\mathcal{G}' = Rel(tr')$ and tr' has the same root and leaves as tr.*

Proof. The lemma restricts the labels of internal nodes to be free variables that are created after all the labels on the left hand side. Additionally, each free variable is only allowed to occur once in a tree. Therefore given a set $\mathcal{G}$ of relational atoms as the left hand side of those constraints, and any sequence σ of structural rule applications, the free variable labels for internal nodes can be assigned to any labels occur in $\mathcal{S}(\mathcal{G}, \sigma)$. By Lemma 4.5.1, there exists a sequence σ of E, A applications which converts the tree on the left hand side to a tree which is a variant of the one on the right hand side, thus those constraints can be solved by assigning the free variables to the corresponding labels in the variant tree. □

Although incomplete, our heuristic method can solve constraints quickly in many cases, as will be demonstrated in the next section.

4.6 Experiments

We used a Dell Optiplex 790 desktop with Intel CORE i7 2600 @ 3.4 GHz CPU and 8GB memory as the platform, and tested the following provers on the formulae from Park et al. [77]:

BBeye: the OCaml prover from Park et al. based upon nested sequents [77];

Formula	BBeye (opt)	Naive (Vamp)	$FVLS_{BBI}$ Heuristic
$(P \mathrel{-\!\!*} Q) \wedge (\top * (\top^* \wedge P)) \to Q$	d(2) 0	0.003	0.001
$(\top^* \mathrel{-\!\!*} \neg(\neg P * \top^*)) \to P$	d(2) 0	0.003	0.000
$\neg((P \mathrel{-\!\!*} \neg(P * Q)) \wedge ((\neg P \mathrel{-\!\!*} \neg Q) \wedge Q))$	d(2) 0	0.004	0.001
$\top^* \to ((P \mathrel{-\!\!*} (Q \mathrel{-\!\!*} R)) \mathrel{-\!\!*} ((P * Q) \mathrel{-\!\!*} R))$	d(2) 0.015	0.017	0.001
$\top^* \to ((P * (Q * R)) \mathrel{-\!\!*} ((P * Q) * R))$	d(2) 0.036	0.006	0.000
$\top^* \to ((P * ((Q \mathrel{-\!\!*} V) * R)) \mathrel{-\!\!*} ((P * (Q \mathrel{-\!\!*} V)) * R))$	d(2) 0.07	0.019	0.001
$\neg((P \mathrel{-\!\!*} \neg(\neg(U \mathrel{-\!\!*} \neg(P * (R * Q))) * P)) \wedge R * (U \wedge (P * Q)))$	d(2) 0.036	0.037	0.001
$\neg((R * (U * V)) \wedge B)$ where	d(2) 0.016	0.075	0.039
$B := ((P \mathrel{-\!\!*} \neg(\neg(Q \mathrel{-\!\!*} \neg(U * (V * R))) * P)) * (Q \wedge (P * \top)))$			
$\neg(C * (U \wedge (P * (Q * V))))$ where	d(3) 96.639	0.089	0.038
$C := ((P \mathrel{-\!\!*} \neg(\neg(U \mathrel{-\!\!*} \neg((R * V) * (Q * P))) * P)) \wedge R)$			
$(P * (Q * (R * U))) \to (U * (R * (Q * P)))$	d(2) 0.009	0.048	0.001
$(P * (Q * (R * U))) \to (U * (Q * (R * P)))$	d(3) 0.03	0.07	0.001
$(P * (Q * (R * (U * V)))) \to (V * (U * (P * (Q * R))))$	d(3) 1.625	1.912	0.001
$(P * (Q * (R * (U * V)))) \to (V * (Q * (P * (R * U))))$	d(4) 20.829	0.333	0.001
$\top^* \to (P * ((Q \mathrel{-\!\!*} V) * (R * U)) \mathrel{-\!\!*} ((P * U) * (R * (Q \mathrel{-\!\!*} V))))$	d(3) 6.258	0.152	0.007

Table 4.1: Initial experimental results.

Naive (Vamp): translates a BBI formula into a first-order formula based on the Kripke semantics of BBI. Then uses Vampire 2.6 [50] to solve it;

$FVLS_{BBI}$ **Heuristic:** backward proof search in $FVLS_{BBI}$, using the heuristic-based method to solve the set of constraints, implemented in *OCaml*.

The results are shown in Table 4.1. The BBeye (opt) column shows the results from Park et al's prover where the d() indicates the depth of proof search. The other two columns are for the two methods stated above. We see that naive translation is comparable with BBeye in most cases, but the latter is not stable. When the tested formulae involves more interaction between structural rules, BBeye runs significantly slower. The heuristic method outperforms all other methods in the tested cases.

Nonetheless, our prover is slower than BBeye for formulae which contain many occurrences of the same atomic formulae, giving (id) instances such as:

$$\Gamma; w_1 : P; w_2 : P; \cdots; w_n : P \vdash \mathbf{x} : P; \Delta$$

We have to choose some w_i to match with $\mathbf{x}$ without knowing which choice satisfies other constraints. In the worst case, we have to try each using backtracking. Multiple branches of this form lead to a combinatorial explosion. Determinising the concrete labels (worlds) for formulae in proof search in LS_{BBI} or BBeye [77] avoids this problem. Further work is needed to solve this in $FVLS_{BBI}$.

Even though we do not claim the completeness of our heuristics method, it appears to be a fast way to solve certain problems. Completeness can be restored by fully implementing LS_{BBI} or $FVLS_{BBI}$. The derivations in LS_{BBI} are generally shorter than those in the Display Calculus or Nested Sequent Calculus for BBI. The optimisations of the implementation, however, is out of the scope of this chapter.

4.7 Discussion and Related Work

The main contribution of this chapter is a labelled sequent calculus for BBI_{ND} that is sound, complete, and enjoys cut-elimination. There are no explicit contraction rules in LS_{BBI} and all structural rules can be restricted so that proof search is entirely driven by logical rules. We further propose a free variable system to restrict the proof search space so that some applications of $*R$, $-\!\!* L$ rules can be guided by zero-premise rules. Although we can structure proof search to be more manageable compared to the unrestricted (labelled or display) calculus, the undecidability of BBI implies that there is no terminating proof search strategy for a sound and complete system. The essence of proof search resides in guessing which relational atoms to use in the $*R$ and $-\!\!* L$ rules and whether they need to be applied more than once to a formula. Nevertheless, our initial experimental results raise the hope that a more efficient proof search strategy can be developed based on our calculus.

In the literature, the purely syntactic proof theory of BBI comes in three flavours: Hilbert calculi [84, 36] (cf. Section 2.2), display calculi [16] (cf. Section 2.3) and nested sequent calculi [77] (cf. Section 3.4). All are sound and complete w.r.t. the ND-semantics.

In between the relational semantics and the purely syntactic proof theory are the labelled tableaux of Larchey-Wendling and Galmiche which are sound and complete w.r.t. the PD-semantics [61, 60]. They remark that "the adaptation of this tableau system to BBI_{TD} should be straightforward (contrary to BBI_{ND})" [63]. Their labelled tableau calculus can be converted into a labelled sequent calculus that has the same set of logical rules as that of LS_{BBI}. The difference lies in the rules for capturing the semantics. Our structural rules directly encode the properties of the non-deterministic monoidal semantics of BBI (i.e., identity, commutativity, and associativity) by explicitly using ternary relational atoms. In comparison, Larchey-Wendling and Galmiche's tableau calculus indirectly captures the partial-deterministic semantics via a set of rules for *Partial Monoidal Equivalences* (PMEs). Their rules do not employ ternary relations, but treat a combination of worlds as a string, building in the partial-determinism reading. This could be part of the reason why their tableau calculus can be further specialised to capture other properties in separation theories, but it is hard to be generalised to capture the non-deterministic semantics. The tableau method also differs from ours in that its completeness is proved via a counter-model construction, whereas our completeness is proved by simulating the Hilbert system for BBI. We shall see later that there does not exist a Hilbert system for BBI_{PD}, thus their counter-model construction is necessary.

The display postulates and other structural rules of display calculi, especially the contraction rules on structures, are impractical for backward proof search. Display postulates shuffle structures in a sequent, whereas contraction rules copy structures from the conclusion to the premise. These rules are applicable at any stage of the proof search, and they can easily generate redundant derivations when one is not

clear which inference rule to use. Hence it is hard to give a systematic proof search procedure without controlling these rules. Nested sequents usually face similar problems with the contraction rules and propagation rules, and although Park et al. [77] showed the admissibility of contraction in an improved nested sequent calculus, that calculus contains other rules that explicitly contract structures. Their iterative deepening automated theorem prover for BBI based on nested sequents is terminating and incomplete for bounded depths, but complete and potentially non-terminating for an unbounded depth [77]. The labelled tableaux of Larchey-Wendling and Galmiche compile all structural rules into PD-monoidal constraints, and are cut-fee complete for BBI_{PD} using a potentially infinite counter-model construction [60]. But effective proof search is only a "perspective" and is left as further work [60, page 2].

Our labelled sequent calculus LS_{BBI} for BBI adopts some features from existing labelled tableaux for BBI [61] and existing labelled sequent calculi for modal logics [72]. Unlike these calculi, some LS_{BBI}-rules contain substitutions on labels. From a proof-search perspective, labelled calculi are no better than display calculi since they require extra-logical rules to explicitly encode the frame conditions of the underlying (Kripke) semantics. Such rules, which we refer to simply as structural rules, are just as bad as display postulates for proof search since we may be forced to explore all potential models and our structural rules may generate unnecessary relational atoms in proof search. As a step towards our goal, we showed that the applications of these structural rules can be localised around logical rules. Thus these structural rules are only triggered by applications of logical rules, leading to a purely syntax-driven proof search procedure for LS_{BBI} in which all the rule applications in proof search are decided by the logical connectives (the syntax), and redundant derivations caused by structural rules are reduced.

Our work is novel from two perspectives. Compared to the labelled tableaux of Larchey-Wendling and Galmiche, we deal with the non-deterministic semantics of BBI, which they have flagged as a difficulty, and we obtain a constructive cut-elimination procedure. Compared to the nested sequent calculus of Park et al., our calculus has much simpler structural rules. Some of Park et al.'s structural rules and traverse rules involve copying structures. When made contraction free, these rules (especially EA_C) are extraordinarily long and complicated. Our structural rules directly capture the semantics using ternary relational atoms, thus they are very intuitive and easy to read. As a result, our calculus generally gives much shorter derivations for the same formula than Park et al.'s calculus. Note that Park et al. actually gave a labelled variant of their nested sequent calculus, with the same logical rules as ours. However, their structural rules are still just notational variants of the original ones, which are lengthy and do not use ternary relations.

An immediate task is to find a complete and terminating (if possible) constraint solving strategy. We will return to this issue in Chapter 8.

Another interesting topic is to extend our calculus to handle some semantics other

$$\frac{(a, b \rhd c)[c/d]; \Gamma[c/d] \vdash \Delta[c/d]}{(a, b \rhd c); (a, b \rhd d); \Gamma \vdash \Delta}\ P \qquad \frac{(a, b \rhd c); \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ T$$

$$\frac{(\epsilon, \epsilon \rhd \epsilon); \Gamma[\epsilon/a][\epsilon/b] \vdash \Delta[\epsilon/a][\epsilon/b]}{(a, b \rhd \epsilon); \Gamma \vdash \Delta}\ IU \qquad \frac{(a, b \rhd c)[b/d]; \Gamma[b/d] \vdash \Delta[b/d]}{(a, b \rhd c); (a, d \rhd c); \Gamma \vdash \Delta}\ C$$

In T, a, b do occur in the conclusion but c does not
In all substitutions $[y/x]$, $x \neq \epsilon$

Figure 4.9: Some auxiliary structural rules.

than the non-deterministic monoidal ones. Our design of the structural rules in LS_{BBI} can be generalised as follows. If there is a semantic condition of the form $(w_{11}, w_{12} \rhd w_{13}) \wedge \cdots \wedge (w_{i1}, w_{i2} \rhd w_{i3}) \Rightarrow (w'_{11}, w'_{12} \rhd w'_{13}) \wedge \cdots \wedge (w'_{j1}, w'_{j2} \rhd w'_{j3}) \wedge (x_{11} = x_{12}) \wedge \cdots \wedge (x_{k1} = x_{k2})$, we create a rule:

$$\frac{(w'_{11}, w'_{12} \rhd w'_{13}); \cdots; (w'_{j1}, w'_{j2} \rhd w'_{j3}); (w_{11}, w_{12} \rhd w_{13}); \cdots; (w_{i1}, w_{i2} \rhd w_{i3}); \Gamma \vdash \Delta}{(w_{11}, w_{12} \rhd w_{13}); \cdots; (w_{i1}, w_{i2} \rhd w_{i3}); \Gamma \vdash \Delta}\ r$$

And apply substitutions $[x_{12}/x_{11}] \cdots [x_{k2}/x_{k1}]$ globally on the premise, where ϵ is not substituted. Many additional features can be added in this way. We summarise the following desirable ones:

PD-semantics: the composition of two elements is either the empty set or a singleton, i.e., $(a, b \rhd c) \wedge (a, b \rhd d) \Rightarrow (c = d)$;
TD-semantics: the composition of any two elements is always defined as a singleton, i.e., $\forall a, b, \exists c$ s.t. $(a, b \rhd c)$;
Indivisible unit: $(a, b \rhd \epsilon) \Rightarrow (a = \epsilon) \wedge (b = \epsilon)$;
Cancellativity: if $w \circ w'$ is defined and $w \circ w' = w \circ w''$, then $w' = w''$, i.e., $(a, b \rhd c) \wedge (a, d \rhd c) \Rightarrow (b = d)$.

Note that TD-semantics are in addition to PD-semantics, so is our definition of cancellativity. The above are formalised in the rules P, T, IU, C respectively in Figure 4.9.

The formula $(F * F) \rightarrow F$, where $F = \neg(\top \mathrel{-\!\!*} \neg\top^*)$, differentiates BBI_{ND} and BBI_{PD} [62] and is provable using $LS_{BBI} + P$. Using $LS_{BBI} + P + T$, we can prove $(\neg\top^* \mathrel{-\!\!*} \bot) \rightarrow \top^*$ and $(\top^* \wedge ((p * q) \mathrel{-\!\!*} \bot)) \rightarrow ((p \mathrel{-\!\!*} \bot) \vee (q \mathrel{-\!\!*} \bot))$, which are valid in BBI_{TD} but not in BBI_{PD} [62, 21]. These additional rules do not break cut-elimination. The derivations for the formulae above are shown below.

1. To prove the formula $(F * F) \rightarrow F$, where $F = \neg(\top \mathrel{-\!\!*} \neg\top^*)$, we use the following derivation in LS_{BBI}:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(w', w'' \rhd \epsilon); (b', c' \rhd w''); (b, c \rhd w'); (b, c \rhd a); \cdots}A}{(w', c' \rhd w); (w, b' \rhd \epsilon); \cdots}E^2}{(c', w' \rhd w); (b, c \rhd w'); (b', w \rhd \epsilon); \cdots}A}{(b', w \rhd \epsilon); (\epsilon, b \rhd w); (c', c \rhd \epsilon); \cdots}A}{(b, c \rhd a); (b', b \rhd \epsilon); (c', c \rhd \epsilon); (\epsilon, \epsilon \rhd \epsilon); \cdots}U}{(b, c \rhd a); (b', b \rhd \epsilon); (c', c \rhd \epsilon); a : \top \mathrel{-\!\!*} \neg\top^*; b' : \top, c' : \top \vdash}\top^*L^2}{(b, c \rhd a); (b', b \rhd b''); (c', c \rhd c''); a : \top \mathrel{-\!\!*} \neg\top^*; b' : \top, c' : \top; b'' : \top^*; c'' : \top^* \vdash}\neg R^2}{(b, c \rhd a); (b', b \rhd b''); (c', c \rhd c''); a : \top \mathrel{-\!\!*} \neg\top^*; b' : \top, c' : \top \vdash b'' : \neg\top^*; c'' : \neg\top^*}\mathrel{-\!\!*} R^2}{(b, c \rhd a); a : \top \mathrel{-\!\!*} \neg\top^* \vdash b : \top \mathrel{-\!\!*} \neg\top^*; c : \top \mathrel{-\!\!*} \neg\top^*}\neg L^2; \neg R}{(b, c \rhd a); b : \neg(\top \mathrel{-\!\!*} \neg\top^*); c : \neg(\top \mathrel{-\!\!*} \neg\top^*) \vdash a : \neg(\top \mathrel{-\!\!*} \neg\top^*)}*L}{a : F * F \vdash a : F}\to R}{\vdash a : (F * F) \to F}$$

The top sequent above the A rule instance contains only one non-atomic formula: $a : \top \mathrel{-\!\!*} \neg\top^*$ on the left hand side. The correct relational atom that is required to split $a : \top \mathrel{-\!\!*} \neg\top^*$ is $(w'', a \rhd \epsilon)$. However, in the labelled sequent calculus we can only obtain $(w'', w' \rhd \epsilon)$. Although w' and a both have exactly the same children, but the non-deterministic monoid allows the composition $b \circ c$ to be multiple elements, or even $\emptyset$ in $\mathcal{M}$. Thus we cannot conclude that $w' = a$. This can be solved by using P to replace w' by a, then use E to obtain $(w'', a \rhd \epsilon)$ on the left hand side of the sequent, then the derivation can go through:

$$\dfrac{\dfrac{\dfrac{}{(w'', a \rhd \epsilon); \cdots; \vdash \epsilon : \top^*}\top^*R}{(w'', a \rhd \epsilon); \cdots; \epsilon : \neg\top^* \vdash}\neg L \qquad \dfrac{}{(w'', a \rhd \epsilon); \cdots \vdash w'' : \top}\top R}{(w'', a \rhd \epsilon); \cdots; a : \top \mathrel{-\!\!*} \neg\top^*; b' : \top, c' : \top \vdash}\mathrel{-\!\!*} L$$

2. The trick to prove $(\neg\top^* \mathrel{-\!\!*} \bot) \to \top^*$ is to create a relational atom $(w, w \rhd w')$, as shown below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon, \epsilon \rhd w'); \cdots \vdash \epsilon : \top^*}\top^*R}{(w, w \rhd w'); \cdots; w : \top^* \vdash w : \top^*}\top^*L}{(w, w \rhd w'); \cdots \vdash w : \neg\top^*; w : \top^*}\neg R \qquad \dfrac{}{(w, w \rhd w'); \cdots; w' : \bot \vdash w : \top^*}\bot L}{(w, w \rhd w'); w : \neg\top^* \mathrel{-\!\!*} \bot \vdash w : \top^*}\mathrel{-\!\!*} L}{w : \neg\top^* \mathrel{-\!\!*} \bot \vdash w : \top^*}T}{\vdash w : (\neg\top^* \mathrel{-\!\!*} \bot) \to \top^*}\to R$$

3. The proof for $(\top^* \wedge ((p * q) \mathrel{-\!\!*} \bot)) \to ((p \mathrel{-\!\!*} \bot) \vee (q \mathrel{-\!\!*} \bot))$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots;c:q\vdash c:q;\cdots}\,id \quad \dfrac{}{\cdots;a:p\vdash a:p;\cdots}\,id}{(a,c\rhd e);\cdots;a:p;c:q\vdash e:p*q;\cdots}\,*R \quad \dfrac{}{\cdots e:\bot\vdash\cdots}\,\bot L}{(e,\epsilon\rhd e);(a,c\rhd e);(a,\epsilon\rhd b);(c,\epsilon\rhd d);\epsilon:(p*q)\multimap\bot;a:p;c:q\vdash b:\bot;d:\bot}\,\multimap L}{(a,c\rhd e);(a,\epsilon\rhd b);(c,\epsilon\rhd d);\epsilon:(p*q)\multimap\bot;a:p;c:q\vdash b:\bot;d:\bot}\,U}{(a,\epsilon\rhd b);(c,\epsilon\rhd d);\epsilon:(p*q)\multimap\bot;a:p;c:q\vdash b:\bot;d:\bot}\,T}{\epsilon:(p*q)\multimap\bot\vdash\epsilon:p\multimap\bot;\epsilon:q\multimap\bot}\,\multimap R^2}{w:\top^*;w:(p*q)\multimap\bot\vdash w:p\multimap\bot;w:q\multimap\bot}\,\top^*L}{w:\top^*\wedge((p*q)\multimap\bot)\vdash w:(p\multimap\bot)\vee(q\multimap\bot)}\,\wedge L;\vee R}{\vdash w:(\top^*\wedge((p*q)\multimap\bot))\to((p\multimap\bot)\vee(q\multimap\bot))}\,\to R$$

We will continue to discuss how to handle the above properties in Chapter 6.

Oddly, the formula $\neg(\top^*\wedge A\wedge(B*\neg(C\multimap(\top^*\to A))))$, which is valid in BBI_{ND}, is very hard to prove in the display calculus and Park et al.'s method. We ran this formula using Park et al.'s prover for a week on a CORE i7 2600 processor, without success. Very short proofs of this formula exist in LS_{BBI} or Larchey-Wendling and Galmiche's labelled tableaux (this formula must also be valid in BBI_{PD}), as below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(c,b\rhd\epsilon);(a,b\rhd\epsilon);\epsilon:A;a:B;c:C\vdash\epsilon:A}\,id}{(c,b\rhd d);(a,b\rhd\epsilon);\epsilon:A;a:B;c:C;d:\top^*\vdash d:A}\,\top^*L}{(c,b\rhd d);(a,b\rhd\epsilon);\epsilon:A;a:B;c:C\vdash d:\top^*\to A}\,\to R}{(a,b\rhd\epsilon);\epsilon:A;a:B\vdash b:C\multimap(\top^*\to A)}\,\multimap R}{(a,b\rhd\epsilon);\epsilon:A;a:B;b:\neg(C\multimap(\top^*\to A))\vdash}\,\neg L}{\epsilon:A;\epsilon:B*\neg(C\multimap(\top^*\to A))\vdash}\,*L}{w:\top^*;w:A;w:B*\neg(C\multimap(\top^*\to A))\vdash}\,\top^*L}{w:\top^*\wedge A\wedge(B*\neg(C\multimap(\top^*\to A)))\vdash}\,\wedge L}{\vdash w:\neg(\top^*\wedge A\wedge(B*\neg(C\multimap(\top^*\to A))))}\,\neg R$$

We are not sure what caused the above phenomenon. It might be that a proof for this formula exists in Park et al.'s nested sequent calculus and the display calculus and the Hilbert system, but the proof is too complicated to be found in days by a computer. Another possibility is that there is a bug in Park et al.'s implementation, or in their completeness proof. The worst situation, which was discussed through our private communication with Park at POPL14, is that Park et al.'s nested sequent calculus, Brotherston's display calculus, and the Hilbert system are all sound and complete with respect to each other, but the non-deterministic semantics do not correspond to these proof theories of BBI. Nonetheless, we shall focus on our own problems in this dissertation and leave this as future work.

Chapter 5

Separation Logic

Having discussed some proof methods for BBI in the previous chapters, we can now move on to consider one of the most successful applications of BBI: separation logic (SL). Reynolds, Ishtiaq, O'Hearn, and Yang etc. introduced separation logic as an extension of Hoare logic, which is a widely used method to reason about computer programs with its well-known *Hoare triple*:

$$\{P\}C\{Q\}$$

where P and Q are called the precondition and the postcondition respectively, both of which are assertions typically formulated in first-order logic, and C is a piece of program code. Separation logic further employs the multiplicative connectives from BBI in its assertion language to reason about resources such as memory addresses. It also supports program commands to read and write memory addresses, thus it is able to deal with long-standing troubles in program verification such as reasoning about pointers and aliasing. Within years of research, separation logic quickly became a hot topic and well-developed branch in program verification. There have been numerous proof methods and automated reasoning tools for the assertion language of SL. But existing tools, although some support complicated features such as arbitrary inductive predicates, do not consider the full spectrum of logical connectives. Therefore in the following chapters we will fill in this blank by extending our labelled sequent calculus for BBI to reason about the assertion language of separation logic.

Here is the story of separation logic from the beginning of Chapter 2 retold, but now you may have a better understanding of this logic. The early version of separation logic is formalised as an extension of Hoare logic to reason about shared mutable data structures intuitionistically [87] via assertions from BI and the "points-to" predicate

$$e_1 \mapsto e_2$$

to express that the value of expression e_1 is a memory address that holds the value of expression e_2 as the content. Ishtiaq and O'Hearn then gave a classical version based on BBI that is more powerful and is able to express storage deallocation [54]. This idea

later went through several evolutions [76] and finally was formalised as separation logic [88].

The story, of course, did not end there. The reader may have observed that we have talked about more than one flavour of separation logic. Even the classical flavour introduced by Ishtiaq [54], O'Hearn [76], and Reynolds [88] all have slight differences. The growing interest in separation logic later raised a vast array of papers using separation logic to verify programs. Many of those papers either added new features or made certain modifications to separation logic to fit in their scenario. As a consequence, when we say "separation logic" now, it may not ring a bell for a particular logic, but more often it refers to "a zoo of logics" [55]. Finding the right core principles and providing abstract specifications of separation logics then became a trend [79]. Calcagno, O'Hearn and Yang started a thread of abstract separation logic, which does not have the $\mapsto$ predicate to describe concrete heaps, but still uses BBI as the core of the assertion logic with two extra properties: partial-determinism and cancellativity [24]. Dockins et al. extended these properties with some other properties that are found useful in various settings [32], these properties were then summarised by Brotherston and Villard as a separation theory [22].

This chapter first introduces Reynolds's separation logic in Section 5.1 with a focus on its assertion logic. In this dissertation we shall not cover the full details of Reynolds's assertion logic, but only concentrate on a fragment with all logical connectives, but without address arithmetic and arbitrary predicates. Our dedicated fragment is given in Section 5.1.2.2, in which we also review various fragments of the assertion logic that are automated in the literature, followed by a discussion of the computability issue of these fragments. Section 5.2 presents Calcagno et al.'s abstract separation logic and some popular properties in separation theories.

5.1 Reynolds's Separation Logic

Separation logic has appeared in the literature in many forms, but in general it is a combination of a programming language, an assertion logic, and a specification logic for reasoning about Hoare triples [55]. By tradition, the assertion logic of separation logic is also referred to as separation logic, this might be confusing, but it is the assertion logic that we are interested in. So in the remaining chapters, when we say separation logic without any explanation, we mean the assertion logic of a separation logic. In the following we give the definitions of these components as in Reynolds's 2002 LICS paper [88].

5.1.1 Programming Language

The programming language in Reynolds's separation logic is a simple imperative language that extends the one axiomatised by Hoare [48] with new commands to manip-

ulate mutable data structures:

$$
\begin{array}{rll}
C ::= & v := e & \text{assignment} \\
& \mid C; C & \text{sequencing} \\
& \mid \textbf{if } b \textbf{ then } C \textbf{ else } C & \text{conditional} \\
& \mid \textbf{while } b \textbf{ do } C & \text{loop} \\
& \mid v := \textbf{cons}(e, \cdots, e) & \text{allocation} \\
& \mid v := [e] & \text{lookup} \\
& \mid [e] := e & \text{mutation} \\
& \mid \textbf{dispose } e & \text{deallocation}
\end{array}
$$

Here we use C for a piece of program code, v for a program variable, e for an ordinary expression which may involve program variables, atoms, integers, and basic arithmetic operators such as $+, -, \times$, and use b for a Boolean expression that may involve binary relations such as $=, <, >$ between ordinary expressions, logical constants $\top, \bot$, and logical connectives $\neg, \rightarrow$, etc..

Reynolds defined all *values* as *integers*, an infinite number of which are *addresses*. *Atoms*, containing nil, form a subset of values that is disjoint from addresses. *Heaps* are finite partial functions from addresses to values, and *stores* are total functions from finite sets of *program variables* to values. These are formalised as below:

$$
\begin{array}{ll}
Value = Int & Atoms \cup Addr \subseteq Int \\
\text{nil} \in Atoms & Atoms \cap Addr = \emptyset \\
H = Addr \rightharpoonup_{fin} Value & S = Var \rightarrow Value
\end{array}
$$

A state in separation logic not only talks about the status of variables (the store) but also the contents of memory addresses (the heap). This will be formalised in the next subsection. We write $[\![e]\!]_s$ to denote the valuation of an expression e by store s, and fix that $[\![\text{nil}]\!]_s = \text{nil}$. We often use nil to denote an invalid memory address such as -1. When computing $[\![e]\!]_s$, we look up the value of each variable x in e from the store s, then compute the value of e using arithmetic. The store valuates an ordinary expression to integers, and valuates a Boolean expression to either $\top$ or $\bot$, where $\top = 1$ and $\bot = 0$.

$$[\![b]\!]_S \in \{\top, \bot\} \qquad [\![e]\!]_S \in Int$$

The command $v := \textbf{cons}(e_1, \cdots e_n)$ can be read as: from the memory address at the value of v, allocate n continuous memory addresses with contents e_1 to e_n respectively so that the content of address $S(v)$ is $[\![e_1]\!]_s$. The command $v := [e]$ looks up the content of the address $[\![e]\!]_s$ and assigns that content to variable v. The mutation command $[e_1] := e_2$ updates the content of the address $[\![e_1]\!]_s$ to the value $[\![e_2]\!]_s$. We say that each of the commands $v := \textbf{cons}(\cdots)$ and $v := [e]$ and $v := e$ *modify* v. Note that the command $[v] := e$, for example, does not change the value of v, but only changes the content of memory address $[\![v]\!]_s$.

$s, h \Vdash \top^*$ iff $h = \emptyset$
$s, h \Vdash A * B$ iff $\exists h_1, h_2.(h_1 \circ h_2 = h$ and $s, h_1 \Vdash A$ and $s, h_2 \Vdash B)$
$s, h \Vdash e \mapsto e'$ iff $dom(h) = \{[\![e]\!]_s\}$ and $h([\![e]\!]_s) = [\![e']\!]_s$
$s, h \Vdash A -\!\!* B$ iff $\forall h_1, h_2.(h_1 \circ h = h_2$ and $s, h_1 \Vdash A)$ implies $s, h_2 \Vdash B)$

Table 5.1: The semantics of the new formulae in separation logic.

5.1.2 Assertion Logic

This subsection first gives the full assertion logic defined by Reynolds, then introduces a fragment that is compliant with Reynolds's semantics and will be used in Chapter 7. Other widely-used fragments for SL and computability issues are discussed at the end of this subsection.

5.1.2.1 Reynolds's Full Assertion Logic

The assertion language of separation logic that Reynolds proposed is an extension of classical first-order logic with the multiplicative connectives from BBI and a special predicate $\mapsto$ called "points-to" for singleton heaps. Thus the syntax extends that of first-order logic (cf. Section 1.1.1) with the following:

$$F ::= \ e \mapsto e' \mid \top^* \mid F * F \mid F -\!\!* F$$

where e stands for an expression and F is a SL formula. We prefer to write $\top^*$ for the empty heap constant *emp* in the literature, as the former is used in the prior work for BBI and PASL [53, 51]. The points-to predicate $e \mapsto e'$ denotes a singleton heap sending the value of e to the value of e'. The connectives $*$ and $-\!\!*$ are taken from BBI, which, in separation logic, denote heap composition and heap extension respectively. These two connectives are interpreted with the binary operator $\circ$ defined as follows, where h_1, h_2 are heaps [20]:

$$h_1 \circ h_2 = \begin{cases} h_1 \cup h_2 & \text{if } h_1, h_2 \text{ have disjoint domains} \\ \text{undefined} & \text{otherwise} \end{cases}$$

That is, two heaps h_1 and h_2 can only be composed when $dom(h_1) \cap dom(h_2) = \emptyset$. A *state* is a pair (s, h) of a store s and a heap h. Given two stores s_1 and s_2, the operator $\circ$ can be extended to states as below:

$$(s_1, h_1) \circ (s_2, h_2) = \begin{cases} (s_1, h_1 \circ h_2) & \text{if } s_1 = s_2 \text{ and} \\ & \quad h_1 \circ h_2 \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

A *separation logic model* is a pair (S, H) of stores and heaps. Both are non-empty as

defined previously. The forcing relation between a state and a formula in one of the four new forms is formally defined in Table 5.1.

A formula F is true at the state (s,h) if $(s,h) \Vdash F$, and it is *valid* if $(s,h) \Vdash F$ for every $s \in S, h \in H$.

The literature contains the following useful abbreviations:

$$e \mapsto \_ \equiv \exists x.e \mapsto x \qquad e_1 \doteq e_2 \equiv (e_1 = e_2) \wedge \top^* \qquad e \hookrightarrow e' \equiv e \mapsto e' * \top$$
$$e \mapsto e_1, \cdots, e_n \equiv (e \mapsto e_1) * (e+1 \mapsto e_2) * \cdots * (e+n-1 \mapsto e_n)$$

The multi-field points-to predicate $e \mapsto e_1, \cdots, e_n$ has different interpretations in the literature. In Reynolds's notation, the formula $e \mapsto e_1, e_2$ is equivalent to $(e \mapsto e_1) * (e+1 \mapsto e_2)$, thus it is a heap of size two. However, in other versions of separation logic, the set of heaps may be defined as finite partial functions from addresses to pairs of values [26, 18]:

$$H = Addr \rightharpoonup_{fin} Value \times Value$$

In this setting the formula $e \mapsto e_1, e_2$ is a singleton heap. A more general case can be found in the definition of symbolic heap [8], where heaps are defined as[1]

$$H = Addr \rightharpoonup_{fin} (Fields \to Value)$$

Now one can define as (finitely) many fields in the points-to predicate as one wishes. But this also creates a dilemma for us, since on the one hand we want to use Reynolds's semantics and notations, on the other hand we also want to compare our method with the prevalent tools for symbolic heap, such as Smallfoot [7]. We will use the interpretation of Smallfoot for the multi-field points-to predicate when comparing our work with Smallfoot. Moreover, we will also provide a slightly different tool that obeys Reynolds's semantics.

5.1.2.2 The Assertion Logic in This Dissertation

Reynolds allows arithmetic expressions denoting values, but in the remaining of this dissertation, we shall only focus on expressions which are either program variables or the constant nil. Variables in our assertion logic are ranged over by x, y, z and expressions by e, possibly with subscripts and primes. We also do not consider arbitrary predicates, terms and propositions from first-order logic. Our syntax for formulae is thus restricted as:

$$F ::= e = e' \mid e \mapsto e' \mid e \mapsto e', e'' \mid \bot \mid F \to F \mid \top^* \mid F * F \mid F -\!\!* F \mid \exists x.F$$

The only atomic formulae are $\bot$, $\top^*$, $(e = e')$, $(e \mapsto e')$, and $(e \mapsto e', e'')$. We will later develop a branch of our method that considers the arbitrary multi-field $\mapsto$

[1] We made a slight modification since in the original symbolic heap, addresses and values are disjoint.

predicate from the symbolic heap semantics. There are no propositional variables. The domain of the quantifier is the set of values. We assume the usual notion of free and bound variables in formulae. We can also define $\top$, $\neg$, $\wedge$, $\vee$ and $\forall$ based on the above connectives as in first-order logic. The two field points-to predicate $e \mapsto e', e''$ is similar to the one field version, but also specifies that the next address contains the value of e''.

The semantics of the formulae in our fragment are the same as in Reynolds's definition, Table 5.2 completes Table 5.1 for the semantics of our logic. We write $s[x \mapsto v]$ to denote a stack that is identical to s, except possibly on the valuation of x, i.e., $s[x \mapsto v](x) = v$ and $s[x \mapsto v](y) = s(y)$ for $y \neq x$.

$s, h \Vdash \bot$ iff never
$s, h \Vdash e = e'$ iff $[\![e]\!]_s = [\![e']\!]_s$
$s, h \Vdash A \to B$ iff $s, h \Vdash A$ implies $s, h \Vdash B$
$s, h \Vdash \exists x.A$ iff $\exists v \in \mathit{Value}$ such that $s[x \mapsto v], h \Vdash A$

Table 5.2: The semantics of some formulae in our assertion logic.

5.1.2.3 Other Fragments of SL and Computability Issues

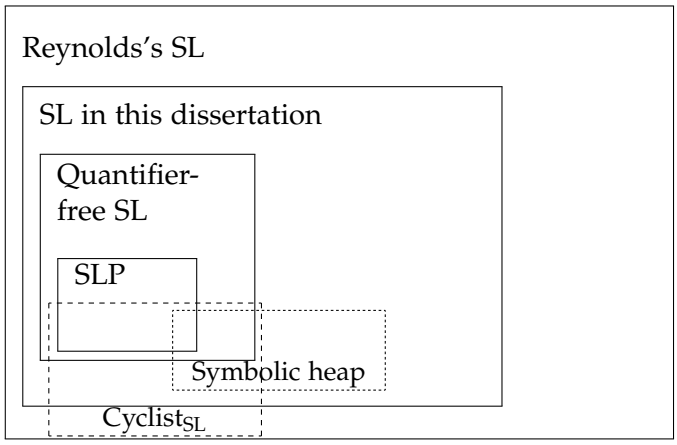

Figure 5.1: Syntactical expressivity of some SL fragments with Reynolds's semantics.

The expressive power of some popular fragments of SL are visualised in Figure 5.1. The syntax of our fragment introduced in the previous section is more expressive than the popular *symbolic heap* fragment [8], whose syntax is restricted to the following:

$$P ::= e = e' \mid \neg P \qquad \Pi ::= \top \mid P \mid \Pi \wedge \Pi$$
$$S ::= e \mapsto [f : e] \qquad \Sigma ::= \top^* \mid S \mid \Sigma * \Sigma$$

The $\mapsto$ predicate in symbolic heap supports a list $[f_1 : e_1, \cdots, f_n : e_n]$ of fields, where f is the name of a field and e is the content of the field. Symbolic heaps are pairs $\Pi \wedge \Sigma$.

The entailment of symbolic heaps is written as $\Pi \wedge \Sigma \vdash \Pi' \wedge \Sigma'$, and the corresponding formula is $(\Pi \wedge \Sigma) \to (\Pi' \wedge \Sigma')$. The symbolic heap fragment also allows formulae of the form $e \mapsto \_$ which does not specify the content of the heap.

It is known that $-\!\!*$ can encode $*$, but not the other way around [14], thus symbolic heap cannot express extending the current heap with a new heap. Even in the absence of $-\!\!*$, there are many interesting formulae that cannot be expressed in the symbolic heap fragment. For example, symbolic heap cannot express $(F * F) \to F$ where F is $\neg(\top \to \neg\top^*)$, the formula that is only valid when partial-determinism holds in abstract separation logics [62]. Symbolic heap also cannot describe the axiom $(\top^* \wedge (A * B)) \to A$ of indivisible unit, because having more than one splitting of a heap is not allowed. For some examples with the $\mapsto$ predicate and lists, see the experimental section of [91]. However, a majority of previous work on automated reasoning for SL focused on the symbolic heap fragment, because it is decidable [6]. As a result, sound, complete and terminating decision procedures for this fragment can be developed, e.g., Smallfoot [7].

Galmiche and Méry's resource graph tableau method considers a decidable fragment called SLP[2] [37], which does not consider first-order quantifiers, equality of expressions, the one-field $\mapsto$ predicate, and quantified variables from our syntax, and only allows l in $(l \mapsto e', e'')$ to be an address. Without equality, SLP cannot express data structures such as lists and trees. The authors further extended the tableau method to handle quantifiers and equality, although completeness is impossible. Their tableau method relies on a complicated back-end theory to check the consistency of heaps, thus automating proof search using their tableau method is likely to be non-trivial.

The quantifier-free fragment of SL is decidable [26], but so far there is no proof method that is sound and complete for this fragment, except for a translation method to first-order logic on an empty signature [23].

Our assertion logic introduced in Section 5.1.2.2 does not support complicated features such as address and expression arithmetic and arbitrarily defined predicate etc., but does allow arbitrary combinations of all the logical connectives, which is not supported by existing automated reasoning tools for SL. If one allows the combination of all connectives, the assertion logic with the two-field $\mapsto$ predicate is not recursively enumerable [26], and the logic with the one-field $\mapsto$ predicate is equivalent to second order logic [14]. Our logic is not recursively enumerable either, thus it has no finitary proof system that is both sound and complete. But a sound proof system may still be useful if it can prove a wide range of formulae in a reasonable time. Developing such a system is the main objective in Chapter 7.

BBI can be faithfully translated to first-order logic, although proving translated formulae using first-order solvers is not very efficient [53]. By contrast, we could not use a naive translation from our assertion logic to first-order logic, because the naive translation encodes heaps as functions, and connectives $*, -\!\!*$ build quantifiers over

[2]Do not confuse this with the solver SLP by Navarro Pérez et al. [70]

functions. More sophisticated incomplete translation may be possible, but that is not in our scope.

Cyclist$_{\text{SL}}$ [19] implements a fragment of SL that includes connectives $\top^*$, $\vee$, $*$, and $\exists$, predicates $\mapsto$, $=$, and $\neq$, as well as arbitrary inductively defined predicates. As for the symbolic heap fragment, the fragment considered by Cyclist$_{\text{SL}}$ excludes $-\!\!*$, so it cannot express heap extension. However, it supports arbitrary inductive definitions, whereas we will only consider two specific instances, i.e., lists and trees (see Section 7.3), and unlike Cyclist$_{\text{SL}}$, we do not support direct inductive reasoning over these definitions.

There are also closely related fragments of separation logic that are recently identified and do not yet have mechanisations. Demri and Deters [14] sharpened the previous undecidability results on separation logic with the 1 field points-to predicate, they further showed that this logic, when restricted to only two quantified variables and the magic wand (i.e., no $*$), is as expressive as the version without these restrictions. Thus their restricted fragment is also undecidable [29]. On the other hand, Demri et al. showed that separation logic with the 1 field points-to predicate, when restricted to only one quantified variable, but with both $*$ and $-\!\!*$, is PSPACE-complete [28]. The authors also showed that when the number of program variables are bounded, the satisfiability problem can be solved in polynomial time. Hence automated reasoning in these fragments is an interesting problem.

5.1.3 Specification Logic

The notion of program specification of separation logic is similar to that of Hoare logic. Reynolds discussed both partial and total correctness, here we only revisit the partial correctness part using Hoare triples of the form $\{P\}C\{Q\}$, which means that if the precondition P holds in the prestate before the code C is executed, and C terminates in a poststate, then the postcondition Q holds in the poststate. We now give the inference rules for Hoare triples à la Reynolds [88].

Most inference rules in Hoare logic still work in separation logic, for example, the consequence rule that combines precondition strengthening and postcondition weakening runs as follows:

$$\frac{P' \to P \qquad \{P\}C\{Q\} \qquad Q \to Q'}{\{P'\}C\{Q'\}}\ \text{Consequence}$$

Auxiliary variable elimination, where v is a variable not free in C:

$$\frac{\{P\}C\{Q\}}{\{\exists v.P\}C\{\exists v.Q\}}\ \text{Variable Elimination}$$

And the substitution rule where $v_1, \cdots, v_n$ are variables free in P, C, Q and if v_i, where $1 \leq i \leq n$, is modified by C, then e_i is a variable that does not occur free in any other e_j, where $1 \leq j \leq n, j \neq i$. The rule is presented below with the notation of substitution identical to the one used in our labelled sequent calculus.

$$\frac{\{P\}C\{Q\}}{(\{P\}C\{Q\})[e_1/v_1,\cdots,e_n/v_n]}\ \text{Substitution}$$

The rules for program commands in Hoare logic can also be used in separation logic, those are presented below.

$$\frac{}{\{Q[e/v]\}v := e\{Q\}}\ \text{Assignment} \qquad \frac{\{P\}C_1\{Q\} \qquad \{Q\}C_2\{R\}}{\{P\}C_1;C_2\{R\}}\ \text{Sequencing}$$

$$\frac{\{P \wedge b\}C_1\{Q\} \qquad \{P \wedge \neg b\}C_2\{Q\}}{\{P\}\textbf{if}\ \ b\ \ \textbf{then}\ \ C_1\ \ \textbf{else}\ \ C_2\{Q\}}\ \text{Conditional} \qquad \frac{\{P \wedge b\}C\{Q\}}{\{P\}\textbf{while}\ \ b\ \ \textbf{do}\ \ C\{Q \wedge \neg b\}}\ \text{Loop}$$

Note that the assignment rule in Hoare logic works backwards, i.e., we copy the post-condition Q to the precondition and perform the substitution. Some literature presents the forward reading assignment rule, but they turn out to be equivalent [41].

The rule of consistency, however, is not sound in Hoare logic and separation logic:

$$\frac{\{P\}C\{Q\}}{\{P \wedge R\}C\{Q \wedge R\}}\ \text{Consistency}$$

For example, $\{\top^*\}v := \textbf{cons}(1)\{v \mapsto 1\}$ holds, but $\{\top^* \wedge \top^*\}v := \textbf{cons}(1)\{(v \mapsto 1) \wedge \top^*\}$ is not true since $(v \mapsto 1) \wedge \top^*$ is unsatisfiable.

A key feature of separation logic is that it supports local reasoning, which only uses the part of memory (heap) used in the program code. When composing programs in a proof, local reasoning often requires the frame rule that extends the specification to talk about a part of memory that is not used in the program, as shown below.

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}\ \text{Frame}$$

where no variable occurring free in R is modified by C.

Instead of showing Reynolds's local and global rules for the mutation, deallocation, allocation, and lookup commands, we simply present the *Small Axioms* of O'Hearn et al. [76]:

$$\{e_1 \mapsto \_\}[e_1] := e_2\{e_1 \mapsto e_2\} \qquad \{e \mapsto \_\}\textbf{dispose}(e)\{\top^*\}$$
$$\{v \doteq m\}v := \textbf{cons}(e_1,\cdots,e_n)\{v \mapsto e_1[m/v],\cdots,e_n[m/v]\}$$
$$\{e \mapsto n \wedge v = m\}v := [e]\{v = n \wedge e[m/v] \mapsto n\}$$

Reynolds also gave backward reasoning rules for the above commands. Those backward rules give complete weakest preconditions that may involve $-\!\!*$ and are less frequently used in automated tools since few existing tools support $-\!\!*$. The door to such weakest preconditions will be open when we have a theorem prover for separation logic that can deal with $-\!\!*$.

5.2 Abstract Separation Logic and DHA Separation Theory

In this section we move back to the abstract setting and consider separation logics that do not have the $\mapsto$ predicate and the elements in the programming language, but instead use propositions. These abstract logics are based on BBI, but they further satisfy various useful properties abstracted from concrete heap models etc.. Therefore these abstract logics are useful in a wide spectrum of situations, although they may not be applied directly.

We define the *separation algebra* semantics of Calcagno et al [24] for Propositional Abstract Separation Logic (PASL), discuss some properties in separation theories, and present examples of these semantics summarised by Clouston.

Separation logic variants in abstract semantics are undecidable [20, 62] even at the propositional level, thus there can only be semi-decision procedures for the logics in this subsection.

5.2.1 Propositional Abstract Separation Logic

The formulae of PASL are defined inductively as follows, where p ranges over some set *PVar* of propositional variables:

$$A ::= p \mid \top \mid \bot \mid \neg A \mid A \vee A \mid A \wedge A \mid A \to A \mid \top^* \mid A * A \mid A -\!\!* A$$

PASL-formulae are interpreted with respect to the following semantics, where we use H as the set of worlds instead of $\mathcal{M}$ because separation algebra is inspired by the heap model, thus each world can be seen as a heap.

Definition 5.2.1. *A* separation algebra*, or partial cancellative commutative monoid, is a triple $(H, \circ, \epsilon)$ where H is a non-empty set, $\circ$ is a partial binary function $H \times H \rightharpoonup H$ written infix, and $\epsilon \in H$, satisfying the following conditions, where '$=$' is interpreted as 'both sides undefined, or both sides defined and equal':*

identity: $\forall h \in H.\, h \circ \epsilon = h$
commutativity: $\forall h_1, h_2 \in H.\, h_1 \circ h_2 = h_2 \circ h_1$
associativity: $\forall h_1, h_2, h_3 \in H.\, h_1 \circ (h_2 \circ h_3) = (h_1 \circ h_2) \circ h_3$
cancellativity: $\forall h_1, h_2, h_3, h_4 \in H.$ *if* $h_1 \circ h_2 = h_3$ *and* $h_1 \circ h_4 = h_3$ *then* $h_2 = h_4$

Note that *partial-determinism* of the monoid is assumed since $\circ$ is a partial function: for any $h_1, h_2, h_3, h_4 \in H$, if $h_1 \circ h_2 = h_3$ and $h_1 \circ h_2 = h_4$ then $h_3 = h_4$.

The following are some example applications of the above properties in the semantics summarised by Clouston.

Example 5.2.1. *The paradigmatic example of a separation algebra is the set of* heaps *[88]: finite partial functions from an infinite set of* addresses *to a set of* values*. Then $h_1 \circ h_2 = h_1 \cup h_2$ if h_1, h_2 have disjoint domains, and is undefined otherwise. ϵ is the empty function.*

Example 5.2.2. *A* partial commutative semigroup *[11], also known as a* permission algebra *[24].*[3]*, is a set V equipped with an associative commutative partial binary operator $\star$, written infix.*

Given an infinite set Addr of addresses, we define two finite partial functions h_1, h_2 from Addr to V to be compatible *iff for all l in the intersection of their domains, $h_1(l) \star h_2(l)$ is defined (we give definitions of $\star$ below for various situations). We then define the binary operation $\circ$ on partial functions h_1, h_2 as undefined if they are not compatible, otherwise when they are compatible, $(h_1 \circ h_2)(l)$ is defined as:*

$$(h_1 \circ h_2)(l) = \begin{cases} h_1(l) \star h_2(l) & l \in dom(h_1) \cap dom(h_2) \\ h_1(l) & l \in dom(h_1) \setminus dom(h_2) \\ h_2(l) & l \in dom(h_2) \setminus dom(h_1) \\ undefined & l \notin dom(h_1) \cup dom(h_2) \end{cases}$$

Setting ϵ as the empty function, many examples of concrete separation algebras have this form, with the $\star$ operation, where defined, intuitively corresponding to some notion of shared *resource. The following are some example definitions of the $\star$ operation with resource reading:*

- *Heaps: let V be the set of values, and $\star$ be undefined everywhere.*
- *Fractional permissions [13]: let V be the set of pairs of values (denoted by v, w) and (real or rational) numbers (denoted by i, j) in the interval $(0, 1]$, and*

$$(v, i) \star (w, j) = \begin{cases} (v, i + j) & v = w \text{ and } i + j \leq 1 \\ undefined & otherwise \end{cases}$$

- *Named permissions [78]: given a set $\mathbb{P}$ of* permission names*, let V be the set of pairs of values (denoted by v, w) and non-empty subsets (denoted by P, Q) of $\mathbb{P}$, and*

$$(v, P) \star (w, Q) = \begin{cases} (v, P \cup Q) & v = w \text{ and } P \cap Q = \emptyset \\ undefined & otherwise \end{cases}$$

- *Counting permissions [11]: let V be the set of pairs of values (denoted by v, w) and integers (denoted by i, j). Here 0 is interpreted as* total permission*, negative integers as* read permissions*, and positive integers as counters of the number of permissions taken. Let*

$$(v, i) \star (w, j) = \begin{cases} (v, i + j) & v = w \text{ and } i < 0 \text{ and } j < 0 \\ (v, i + j) & v = w \text{ and } i + j \geq 0 \text{ and } (\, i < 0 \text{ or } j < 0 \,) \\ undefined & otherwise \end{cases}$$

[3] We prefer the former term, as many interesting examples have little to do with permissions, and the definition of permissions algebra seems somewhat up for grabs - compare [24, 96].

- *Binary Tree Share Model [32]: Consider the set of finite non-empty binary trees whose leaves are labelled true ($\top$) or false ($\bot$), modulo the smallest congruence such that $\top \cong \widehat{\top\top}$ and $\bot \cong \widehat{\bot\bot}$. Let $\vee$ (resp. $\wedge$) be the pointwise disjunction (resp. conjunction) of representative trees of the same shape. Then let V be the pairs of values (denoted by v, w) and equivalence classes of trees (denoted by t, u) so defined, with*

$$(v,t) \star (w,u) = \begin{cases} (v, t \vee u) & v = w \text{ and } t \wedge u = [\bot] \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Note that the construction above employing partial commutative semigroups does not in general guarantee cancellativity; for this we need to require further that $(V, \star)$ is cancellative and has no idempotent elements (where $v \star v = v$). As we will see later, some interesting concrete models fail this requirement, so we will generalise the results of this dissertation to drop cancellativity in Section 6.4.

Example 5.2.3. *Other concrete separation algebras resemble the construction of Example 5.2.2 without fitting it precisely:*

- *Finite set of addresses: The concrete memory model of a 32-bit machine [56] has as its addresses the set of integers* $[0 \ldots 2^{32})$.
- *Total functions: Markings of petri nets [69] without capacity constraints are simply multisets. They may be considered as separation algebras [24] by taking Addr to be Places and $(V, \star)$ to be the set of natural numbers with addition, then considering the set of* total *functions Places $\to \mathbb{N}$, with $\circ$ defined as usual (hence, as multiset union), and ϵ as the constant 0 function. If there is a global capacity constraint κ then we let $i \star j$ be undefined if $i + j > \kappa$, and hence $\circ$ becomes undefined also in the usual way.*

  Note that this example can only be made to exactly fit the construction of Example 5.2.2 if we restrict ourselves to markings of infinite Petri nets with finite numbers of tokens*. In this case we would consider a place without tokens to have an undefined map, rather than map to 0, and set V to be the positive integers.*
- *Constraints on functions: The endpoint heaps of [99] are only those partial functions that are* dual, irreflexive *and* injective *(we refer to the citation for the definition of these properties). Similarly, if the places of a petri net comes equipped with a capacity constraint function κ : Places $\to \mathbb{N}$, we consider only those functions compatible with those constraints.*

There are many other examples of separation logics with concrete semantics, which inspired us to study the abstract properties that are shared by those concrete semantics. We begin with PASL, which is based on the separation algebra defined in Def. 5.2.1. In this dissertation we prefer to express PASL semantics in the style of *ternary relations* to maintain consistency with the earlier work on BBI [53]; it is easy to see that the definition below is a trivial notational variant of Def. 5.2.1.

$h \Vdash p$	iff	$p \in PVar$ and $h \in v(p)$	$h \Vdash \top^*$	iff	$h = \epsilon$
$h \Vdash A \wedge B$	iff	$h \Vdash A$ and $h \Vdash B$	$h \Vdash \top$	iff	always
$h \Vdash A \to B$	iff	$h \nVdash A$ or $h \Vdash B$	$h \Vdash \bot$	iff	never
$h \Vdash A \vee B$	iff	$h \Vdash A$ or $h \Vdash B$	$h \Vdash \neg A$	iff	$h \nVdash A$
$h \Vdash A * B$	iff	$\exists h_1, h_2.(R(h_1, h_2, h)$ and $h_1 \Vdash A$ and $h_2 \Vdash B)$			
$h \Vdash A -\!\!* B$	iff	$\forall h_1, h_2.((R(h, h_1, h_2)$ and $h_1 \Vdash A)$ implies $h_2 \Vdash B)$			

Table 5.3: Semantics of PASL.

Definition 5.2.2. *A* PASL Kripke relational frame *is a triple* (H, R, ϵ)*, where* H *is a non-empty set of* worlds*,* $R \subseteq H \times H \times H$*, and* $\epsilon \in H$*, satisfying the following conditions for all* h_1, h_2, h_3, h_4, h_5 *in* H*:*

identity: $R(h_1, \epsilon, h_2) \Leftrightarrow h_1 = h_2$
commutativity: $R(h_1, h_2, h_3) \Leftrightarrow R(h_2, h_1, h_3)$
associativity: $(R(h_1, h_5, h_4) \& R(h_2, h_3, h_5)) \Rightarrow \exists h_6.(R(h_6, h_3, h_4) \& R(h_1, h_2, h_6))$
cancellativity: $(R(h_1, h_2, h_3) \& R(h_1, h_4, h_3)) \Rightarrow h_2 = h_4$
partial-determinism: $(R(h_1, h_2, h_3) \& R(h_1, h_2, h_4)) \Rightarrow h_3 = h_4$.

A *PASL Kripke relational model* is a tuple (H, R, ϵ, v) of a PASL Kripke relational frame (H, R, ϵ) and a *valuation* function $v : PVar \to \mathcal{P}(H)$ (where $\mathcal{P}(H)$ is the power set of H). The forcing relation $\Vdash$ between a world $h \in \mathcal{M}$ and a formula is defined in Table 5.3, where we write $h \nVdash A$ for the negation of $h \Vdash A$. Given a model $\mathcal{M} = (H, R, \epsilon, v)$, a formula is *true at (world)* h iff $\mathcal{M}, h \Vdash A$. The formula A is *valid* iff it is true at all worlds of all models.

5.2.2 DHA Separation Theory

As shown by some examples in the previous subsection, even the notion of separation algebra is not universally agreed, for example, in some applications we do not demand cancellativity. On the other hand, Dockins et al. examined several additional properties that may also be useful, including indivisible unit, disjointness, splittability and cross-split [32][4], the first of which was proposed by Brotherston and Kanovich [20]. We have discussed some of these properties at the end of Chapter 4. In fact, one can freely choose these properties and the ones presented previously to obtain an abstract separation logic. Thus Brotherston and Villard proposed the term *separation theory* [22] to mean any combination of the properties like those proposed by Dockins et al.. Here we call the set of properties: partial-determinism, cancellativity, indivisible unit, disjointness, splittability, and cross-split as *DHA separation theory*, following the initials of Dockins, Hobor, and Appel. We now discuss the additional properties with examples provided by Clouston.

[4]Their work also distinguishes models with single unit and multiple units. However, it turns out these two properties give the same set of valid BBI-formulae (cf. [64]), so we assume single unit here.

Indivisible Unit The unit ϵ in a commutative monoid $(H, \circ, \epsilon)$ is *indivisible* iff the following holds for any $h_1, h_2 \in H$:

$$\text{if } h_1 \circ h_2 = \epsilon \text{ then } h_1 = \epsilon.$$

Relationally, this corresponds to the first-order condition:

$$\forall h_1, h_2 \in H.\ \text{if } R(h_1, h_2, \epsilon) \text{ then } h_1 = \epsilon.$$

This also means that $h_2 = \epsilon$ whenever $h_1 \circ h_2 = \epsilon$. Most memory models in the literature obey indivisible unit [20], so this property seems appropriate for reasoning about concrete applications of separation logic. Indivisible unit can be axiomatised by the formula [22]:

$$\top^* \wedge (A * B) \to A.$$

Example 5.2.4. *It is trivial to confirm that all the concrete separation algebras surveyed in Section 5.2.1 satisfy indivisible unit; we are not aware of any separation algebras with applications to program verification that fail to do so.*

Disjointness The separating conjunction $*$ in separation logic requires that the two combined heaps have disjoint domains [88]. Without concrete semantics that describe the "points-to" predicate $\mapsto$, we cannot express the domain of a heap. However, the abstract semantics do support a special case where two heaps have a common sub-heap. In a separation algebra $(H, \circ, \epsilon)$, *disjointness* is defined by the following additional requirement:

$$\forall h_1, h_2 \in H.\ \text{if } h_1 \circ h_1 = h_2 \text{ then } h_1 = \epsilon.$$

The above can be expressed relationally:

$$\forall h_1, h_2 \in H.\ \text{if } R(h_1, h_1, h_2) \text{ then } h_1 = \epsilon.$$

Notice also the subtlety between "intersecting heaps" and "heaps with intersecting domains". In Reynolds' semantics, we can find two heaps, for example, $h_1 = \{(1 \mapsto 3), (2 \mapsto 4)\}$ and $h_2 = \{(1 \mapsto 2), (5 \mapsto 7)\}$, that do not have an intersection (a common sub-heap), but their domains intersect, i.e., both contain 1. These two heaps cannot be combined either, but this idea cannot be expressed by the disjointness property in the abstract semantics. We will return to this issue in Section 7.2.

Disjointness implies indivisible unit (but not vice versa), as shown by Dockins et al. [32]. However, when partial-determinism is assumed, disjointness and indivisible unit collapse [64].

Example 5.2.5. *In the cases of Example 5.2.2, where separation algebras are defined via a partial commutative semigroup $(V, \star)$, the disjointness property holds iff there exist no $v \in V$*

such that $v \star v$ is defined. This is the case for heaps, named permissions, and the binary tree share model. On the other hand, disjointness fails to holds for fractional permissions (where $(v,i) \star (v,i)$ is defined so long as $i \leq 0.5$) and counting permissions (for example $(v,-1) \star (v,-1) = (v,-2)$).

Disjointness fails in general for markings of petri nets, as a marking can be combined with itself by doubling its number of tokens at all places. However disjointness holds in the presence of a global capacity constraint $\kappa = 1$.

Splittability The property of infinite splittability is sometimes useful when reasoning about the kinds of resource sharing that occur in divide-and-conquer style computations [32]. A separation algebra $(H, \circ, \epsilon)$ has *splittability* if

$$\forall h_0 \in H \setminus \{\epsilon\}, \exists h_1, h_2 \in H \setminus \{\epsilon\} \text{ such that } h_1 \circ h_2 = h_0.$$

Relationally, this corresponds to:

$$\text{if } h_0 \neq \epsilon \text{ then } \exists h_1 \neq \epsilon, h_2 \neq \epsilon \text{ such that } R(h_1, h_2, h_0).$$

This property can be axiomatised by the formula $\neg\top^* \to (\neg\top^* * \neg\top^*)$ [22].

Example 5.2.6. *In the case of separation algebras defined via a partial commutative semigroup $(V, \star)$, splittability holds iff all $v \in V$ are in the image of $\star$. This holds for fractional permissions, as each (v,i) is $(v, \frac{i}{2}) \star (v, \frac{i}{2})$. The binary tree share model also enjoys this property; see [32] for details.*

On the other hand, splittability does not hold for heaps (for which the image of $\star$ is empty), for named permissions (singletons cannot be split), or for counting permissions (where $(v,-1)$ is not in the image of $\star$). Splittability also fails for petri nets, as the marking assigning one token to one place, with all other places empty, cannot be split.

Cross-split specifies that if a heap can be split in two different ways, then there should be intersections of these splittings. Formally, in a separation algebra $(H, \circ, \epsilon)$, if $h_1 \circ h_2 = h_0$ and $h_3 \circ h_4 = h_0$, then there should be four elements $h_{13}, h_{14}, h_{23}, h_{24}$, informally representing the intersections $h_1 \cap h_3$, $h_1 \cap h_4$, $h_2 \cap h_3$ and $h_2 \cap h_4$ respectively, such that $h_{13} \circ h_{14} = h_1$, $h_{23} \circ h_{24} = h_2$, $h_{13} \circ h_{23} = h_3$, and $h_{14} \circ h_{24} = h_4$. The corresponding condition on Kripke relational frames is:

> $\forall h_0, h_1, h_2, h_3, h_4 \in H$, if $R(h_1, h_2, h_0)$ and $R(h_3, h_4, h_0)$ hold then
> $\exists h_5, h_6, h_7, h_8 \in H$ such that $R(h_5, h_6, h_1)$, $R(h_7, h_8, h_2)$, $R(h_5, h_7, h_3)$
> and $R(h_6, h_8, h_4)$ hold.

Example 5.2.7. *All examples of separation algebras presented in Section 5.2.1 satisfy cross-split; we are not aware of any separation algebras with applications to program verification that fail to do so. In the case of heaps, for example, cross-splits are simply defined as intersections,*

Concrete model	*IU*	*D*	*S*	*CS*	*C*
Heaps	✓	✓	×	✓	✓
Fractional permissions	✓	×	✓	✓	✓
Named permissions	✓	✓	×	✓	✓
Counting permissions	✓	×	×	✓	✓
Binary trees	✓	✓	✓	✓	✓
Petri nets	✓	×	×	✓	✓
Petri nets with capacity 1	✓	✓	×	✓	✓
Monotonic counter	✓	×	✓	✓	×
Logical heaps	✓	×	✓	✓	×

Table 5.4: Some concrete separation algebras and their abstract properties.

but the situation becomes more complex in the case that sharing is possible, and the sub-splittings $h_{13}, h_{23}, \ldots$ need not be uniquely defined.

Take the concrete case of counting permissions. The most difficult case of the examples in Section 5.2.1. Let $h = \{l \mapsto (v, 1)\}$ for some address l and value v. We will abuse notation by writing this as $h = 1$, since the identity of the address and value are not important here. Let (h_1, h_2, h_3, h_4) be $(-2, 3, -3, 4)$ respectively. Then the values of $(h_{13}, h_{14}, h_{23}, h_{24})$ may be $(-2, \mathit{undefined}, -1, 4)$, or $(\mathit{undefined}, -2, -3, 6)$, or $(-1, -1, 2, 5)$.

However the definition of CS does not require uniqueness; it is sufficient to describe a method by which a valid cross-split may be defined. We consider each address $l \in dom(h)$ in turn. The difficult case has that l is in the domain of all of h_1, h_2, h_3, h_4, and that the two splittings are not identical on l. By definition at least one of $h_1(l), h_2(l)$ is negative, and similarly $h_3(l), h_4(l)$. Without loss of generality say $h_1(l)$ is the strictly largest *negative number of the four. Then we may set $(h_{13}(l), h_{14}(l), h_{23}(l), h_{24}(l))$ to be $(h_1(l), \mathit{undefined}, h_3(l) - h_1(l), h_4(l))$. Routine calculation confirms that this defines a cross-split.*

On the other hand, there are concrete models that drop some of the properties in separation theories. For example, the fictional separation logic mentioned below does not require cancellativity.

Example 5.2.8. *The partial commutative semigroup construction of Example 5.2.2, as noted after that example, need not yield a cancellative structure. In particular, if there exists an idempotent element $v \star v = v$, then $\{l \mapsto v\} \circ \{l \mapsto v\} = \{l \mapsto v\} = \{l \mapsto v\} \circ \epsilon$. We give two examples:*

- *Monotonic Counter for Fictional Separation Logic [57]: fictional separation logic is a program verification framework where every module is associated with its own notion of resource. We here note only the example of a monotonic counter, for which the partial commutative semigroup is the integers with a bottom element, with max as operation. Clearly every element is idempotent.*

- *Logical Heaps for Relaxed Separation Logic [97]: we refer to the citation for the rather involved definition and an example of an idempotent element.*

Both of the examples above satisfy indivisible unit, splittability, and cross-split, but fail to satisfy disjointness.

This section presented examples of concrete separation algebras from a range of different program verification applications, and discussed their relation to the various spatial properties considered for abstract separation logic. Table 5.4 summarises this information, in which there is no entry for partial-determinism because it holds in all the examined models. We use P, C, IU, D, S, CS to denote the properties partial-determinism, cancellativity, indivisible unit, disjointness, splittability, and cross-split respectively.

Chapter 6

Labelled Sequent Calculi for Propositional Abstract Separation Logics

Separation logic has proved fruitful for a range of memory (or, more generally, resource) models, some quite different from the original heap model. We have seen examples drawn from [13, 78, 11, 24, 32, 99, 57, 56, 97] in the previous chapter, but this list is far from exhaustive. Each such model has its own notion of separation and sharing of resources, and hence formally gives rise to a new logic with respect to the BI connectives, let alone any special-purpose predicates which might be taken as fundamental. The connections between these separation logics are usually not formal, so new metatheory and tool support must be substantially reconstructed for each case. This has led to a subgenre of papers highlighting the need for organisation and generalisation across these logics [10, 24, 79, 55].

In this chapter we work with the *Abstract Separation Logic* (ASL) of [24]. Taking a cue from the well-known algebraic semantics for substructural logics [35], ASL introduces the abstract semantics of partial cancellative monoids, or *separation algebras* (cf. Section 5.2.1). These semantics allow interpretation of $*$, $\top^*$ and $-\!\!*$, although the latter is not considered by [24]. On the other hand, points-to ($\mapsto$) is not a first class citizen of ASL; it may be introduced as a predicate only if an appropriate concrete separation algebra is fixed. ASL is appropriate to reasoning about certain relations of memory heaps, but not the addresses and their contents. We further consider the properties in DHA separation theory (cf. Section 5.2.2) and the resultant variants of abstract separation logics.

Since [88] the term *separation logic* has been used "both for the extension of predicate calculus with the separation operators and the resulting extension of Hoare logic". In this chapter we consider the former style of logic with respect to abstract separation logic. As we will not consider quantifiers, we call this logic Propositional Abstract Separation Logic (PASL). Reasoning for the assertion language of ASL is the key to

support precondition strengthening and postcondition weakening for the Hoare-style (specification) logic, but proof search and structural proof theory for PASL and its variations has received little attention until now. It is known the added expressive power of the multiplicative connectives comes at a price, since these (propositional) abstract separation logics are in general undecidable [20, 62]. Given their wide applicability, a semi-decision procedure for these logics is then of vital importance in program verification, and that is our goal in this Chapter.

We first give our labelled sequent calculus in Section 6.1, in which we also discuss the cut-elimination theorem for our calculus. Section 6.2 presents the completeness proof for our calculus using a counter-model construction method, which is also extensible to handle some other variants of PASL. Section 6.3 introduces labelled sequent rules for other properties in DHA separation theory, followed by Section 6.4 where we discuss our labelled systems for variants of PASL, completing our labelled calculi framework. We then demonstrate experimental results in Section 6.5 and discuss related work in Section 6.6.

This chapter is based on a published paper by Hóu, Clouston, Goré, and Tiu [51].

6.1 LS_{PASL}: A Labelled Sequent Calculus for PASL

Separation algebras are a restriction of *non-deterministic monoids*, which are known to give sound and complete semantics for BBI [36]. In this sense PASL (cf. Section 5.2.1) is a refinement of BBI, differing only by the addition of the semantic properties of *partial-determinism* and *cancellativity*[1] natural to ask if proof techniques for BBI can be adapted to PASL. We show that this is possible, extending the labelled sequent calculus LS_{BBI} for BBI in Chapter 4 by adding explicit sequent rules that capture partial-determinism and cancellativity, we call the resultant calculus LS_{PASL}.

The definition of labelled formula, relational atom, sequent, substitution, label mapping, sequent falsifiability etc. in LS_{PASL} are the same as those in LS_{BBI} (cf. Section 4.1.1). Here we only slightly change the presentation of a sequent as $\mathcal{G}; \Gamma \vdash \Delta$, where $\mathcal{G}$ is a multiset of relational atoms, and Γ, Δ are multisets of formulae. Moreover, the interpretation of the logical connectives of PASL are the same as those for BBI. Thus we may obtain a labelled sequent calculus LS_{PASL} for PASL by adding the rules P (partial-determinism) and C (cancellativity) to LS_{BBI} [53]. The rules for LS_{PASL} are presented in Fig. 6.1, where p is a propositional variable, A, B are formulae, and w, x, y, z are labels.

Unlike the rules Eq_1 and Eq_2, both of which are needed as explained in Section 4.1.1, we do not need two variants of the rule P or C, since the order of relational atoms in the sequent does not matter. That is, the rule P may also be used if we want

[1] Recent work [64] shows that validity of partial-deterministic BBI-models coincides with validity of cancellative partial-deterministic BBI-models. We nonetheless treat cancellativity as a first class property for reasons we will address in Section 6.6.

Identity and Cut:

$$\frac{}{\mathcal{G};\Gamma;w:p\vdash w:p;\Delta}\ id \qquad \frac{\mathcal{G};\Gamma\vdash x:A;\Delta \qquad \mathcal{G}';\Gamma';x:A\vdash\Delta'}{\mathcal{G};\mathcal{G}';\Gamma;\Gamma'\vdash\Delta;\Delta'}\ cut$$

Logical Rules:

$$\frac{}{\mathcal{G};\Gamma;w:\bot\vdash\Delta}\ \bot L \qquad \frac{(\epsilon,w\triangleright\epsilon);\mathcal{G};\Gamma\vdash\Delta}{\mathcal{G};\Gamma;w:\top^*\vdash\Delta}\ \top^* L \qquad \frac{}{\mathcal{G};\Gamma\vdash w:\top;\Delta}\ \top R \qquad \frac{}{\mathcal{G};\Gamma\vdash\epsilon:\top^*;\Delta}\ \top^* R$$

$$\frac{\mathcal{G};\Gamma;w:A;w:B\vdash\Delta}{\mathcal{G};\Gamma;w:A\wedge B\vdash\Delta}\ \wedge L \qquad \frac{\mathcal{G};\Gamma\vdash w:A;\Delta \qquad \mathcal{G};\Gamma\vdash w:B;\Delta}{\mathcal{G};\Gamma\vdash w:A\wedge B;\Delta}\ \wedge R$$

$$\frac{\mathcal{G};\Gamma\vdash w:A;\Delta \qquad \mathcal{G};\Gamma;w:B\vdash\Delta}{\mathcal{G};\Gamma;w:A\to B\vdash\Delta}\ \to L \qquad \frac{\mathcal{G};\Gamma;w:A\vdash w:B;\Delta}{\mathcal{G};\Gamma\vdash w:A\to B;\Delta}\ \to R$$

$$\frac{(x,y\triangleright z);\mathcal{G};\Gamma;x:A;y:B\vdash\Delta}{\mathcal{G};\Gamma;z:A*B\vdash\Delta}\ *L \qquad \frac{(x,z\triangleright y);\mathcal{G};\Gamma;x:A\vdash y:B;\Delta}{\mathcal{G};\Gamma\vdash z:A\mathrel{-\!\!*}B;\Delta}\ -\!\!* R$$

$$\frac{(x,y\triangleright z);\mathcal{G};\Gamma\vdash x:A;z:A*B;\Delta \qquad (x,y\triangleright z);\mathcal{G};\Gamma\vdash y:B;z:A*B;\Delta}{(x,y\triangleright z);\mathcal{G};\Gamma\vdash z:A*B;\Delta}\ *R$$

$$\frac{(x,y\triangleright z);\mathcal{G};\Gamma;y:A\mathrel{-\!\!*}B\vdash x:A;\Delta \qquad (x,y\triangleright z);\mathcal{G};\Gamma;y:A\mathrel{-\!\!*}B;z:B\vdash\Delta}{(x,y\triangleright z);\mathcal{G};\Gamma;y:A\mathrel{-\!\!*}B\vdash\Delta}\ -\!\!* L$$

Structural Rules:

$$\frac{(y,x\triangleright z);(x,y\triangleright z);\mathcal{G};\Gamma\vdash\Delta}{(x,y\triangleright z);\mathcal{G};\Gamma\vdash\Delta}\ E \qquad \frac{(u,w\triangleright z);(y,v\triangleright w);(x,y\triangleright z);(u,v\triangleright x);\mathcal{G};\Gamma\vdash\Delta}{(x,y\triangleright z);(u,v\triangleright x);\mathcal{G};\Gamma\vdash\Delta}\ A$$

$$\frac{(x,\epsilon\triangleright x);\mathcal{G};\Gamma\vdash\Delta}{\mathcal{G};\Gamma\vdash\Delta}\ U \qquad \frac{(x,w\triangleright x);(y,y\triangleright w);(x,y\triangleright x);\mathcal{G};\Gamma\vdash\Delta}{(x,y\triangleright x);\mathcal{G};\Gamma\vdash\Delta}\ A_C$$

$$\frac{(\epsilon,w'\triangleright w');\mathcal{G}[w'/w];\Gamma[w'/w]\vdash\Delta[w'/w]}{(\epsilon,w\triangleright w');\mathcal{G};\Gamma\vdash\Delta}\ Eq_1 \qquad \frac{(\epsilon,w'\triangleright w');\mathcal{G}[w'/w];\Gamma[w'/w]\vdash\Delta[w'/w]}{(\epsilon,w'\triangleright w);\mathcal{G};\Gamma\vdash\Delta}\ Eq_2$$

$$\frac{(x,y\triangleright z)[z/w];\mathcal{G}[z/w];\Gamma[z/w]\vdash\Delta[z/w]}{(x,y\triangleright z);(x,y\triangleright w);\mathcal{G};\Gamma\vdash\Delta}\ P \qquad \frac{(x,y\triangleright z)[y/w];\mathcal{G}[y/w];\Gamma[y/w]\vdash\Delta[y/w]}{(x,y\triangleright z);(x,w\triangleright z);\mathcal{G};\Gamma\vdash\Delta}\ C$$

Side conditions:
Only label variables (not ϵ) may be substituted for.
In $*L$ and $-\!\!* R$, the labels x and y do not occur in the conclusion.
In the rules A, A_C, the label w does not occur in the conclusion.

Figure 6.1: The labelled sequent calculus LS_{PASL} for PASL.

to globally replace z by w, when $z \neq \epsilon$.

To show that a formula A is *valid* in LS_{PASL}, we prove $\vdash w : A$ for an arbitrary label w. See Section 4.7 for an example derivation of formula $(F * F) \to F$, where $F = \neg(\top \mathrel{-\!\!*} \neg\top^*)$. The use of the rule P is essential in proving this formula.

Theorem 6.1.1 (Soundness). *For any formula A, and any label $w \in LVar$, if the labelled*

sequent $\vdash w : A$ *is derivable in* LS_{PASL} *then* A *is valid in PASL.*

Proof. We prove that the rules of LS_{PASL} preserve falsifiability upwards. For all the rules except for P, C, we refer the interested reader to the soundness proof for LS_{BBI} (Theorem 4.1.1), since the semantics of PASL is just the semantics of BBI plus partial determinism and cancellativity, those rules are sound by the same argument.

For the rule P, suppose the conclusion is falsifiable in some model (H, R, ϵ, v) and label mapping ρ, which means $R(\rho(x), \rho(y), \rho(z))$ and $R(\rho(x), \rho(y), \rho(w))$ both hold in the model. By partial-determinism, $\rho(z) = \rho(w)$, thus the premise is falsifiable. That is, the substitution preserves the falsifiability of the sequent in the same model under the same label mapping.

The case for the rule C is similar. Suppose the conclusion is falsifiable in a model (H, R, ϵ, v) and a label mapping ρ. We know that the relations $R(\rho(x), \rho(y), \rho(z))$ and $R(\rho(x), \rho(w), \rho(z))$ both hold. By cancellativity, $\rho(y) = \rho(w)$, therefore replacing every w with y preserves falsifiability of the premise. □

6.1.1 Cut-elimination for LS_{PASL}

The only differences between LS_{PASL} and LS_{BBI} are the additions of the structural rules P and C, so we may prove cut-elimination by the same route, which in turn follows from the usual cut-elimination procedure for labelled sequent calculi for modal logics [73]. Recall that we use $ht(\Pi)$ to denote the height of the derivation Π. This will be used in the lemmas below.

We first show the substitution lemma for LS_{PASL}, most of the details are the same as for LS_{BBI}, so we only present the part about the new rules.

Lemma 6.1.2 (Substitution)**.** *For any label* $x \neq \epsilon$ *and any label* y*, if* Π *is an* LS_{PASL} *derivation for the sequent* $\mathcal{G}; \Gamma \vdash \Delta$ *then there is an* LS_{PASL} *derivation* Π' *of the sequent* $\mathcal{G}[y/x]; \Gamma[y/x] \vdash \Delta[y/x]$ *such that* $ht(\Pi') \leq ht(\Pi)$*.*

Proof. By induction on $ht(\Pi)$. We do a case analysis on the last rule applied in the derivation. There are three sub-cases: (1) neither x nor y is the label of the principal formula; (2) y is the label of the principal formula; and (3) x is the label of the principal formula. Most of the rules can be proved as in LS_{BBI}. Here we only illustrate the new rules P and C. Apparently they both fall into the first sub-case, as they are structural rules and there is no principal formula for them.

If the last rule in Π is P, which generally runs as below,

$$\frac{(\mathcal{G}; (a, b \rhd c); \Gamma \vdash \Delta)[c/d]}{\mathcal{G}; (a, b \rhd c); (a, b \rhd d); \Gamma \vdash \Delta}\ {}^{P}$$

we further distinguish three cases: (1) $x \neq d$ and $x \neq c$; (2) $x = d$; and (3) $x = c$.

1. If $x \neq d$ and $x \neq c$, we need to consider three sub-cases:

(a) If $y \neq d$ and $y \neq c$, then the two substitutions $[y/x]$ and $[c/d]$ do not interfere with each other, thus we can use the induction hypothesis to substitute $[y/x]$ and reorder the substitutions to obtain the desired derivation.

$$\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[c/d][y/x]}}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[y/x][c/d]}}{(\mathcal{G}; (a, b \triangleright c); (a, b \triangleright d); \Gamma \vdash \Delta)[y/x]}\ P$$

(b) If $y = d$ we first use the induction hypothesis, substituting $[c/x]$, then obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[c/d][c/x]}}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[c/x][c/d]}}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[d/x][c/d]}}{(\mathcal{G}; (a, b \triangleright c); (a, b \triangleright d); \Gamma \vdash \Delta)[d/x]}\ P$$

(c) If $y = c$, the proof is similar to above, without the second last step.

2. If $x = d$, we consider three sub-cases:

(a) If $y \neq c$ and $y \neq \epsilon$, we use the induction hypothesis to substitute $[c/y]$, and obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[c/d][c/y]}}{(\mathcal{G}; (a, b \triangleright c); \Gamma \vdash \Delta)[y/d][c/y]}}{(\mathcal{G}; (a, b \triangleright c); (a, b \triangleright y); \Gamma \vdash \Delta)[y/d]}\ P$$

(b) If $y \neq c$ but $y = \epsilon$, then we use induction hypothesis to substitute $[\epsilon/c]$, and obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G}; (a, b \triangleright \epsilon); \Gamma \vdash \Delta)[c/d][\epsilon/c]}}{(\mathcal{G}; (a, b \triangleright \epsilon); \Gamma \vdash \Delta)[\epsilon/d][\epsilon/c]}}{(\mathcal{G}; (a, b \triangleright c); (a, b \triangleright \epsilon); \Gamma \vdash \Delta)[\epsilon/d]}\ P$$

(c) If $y = c$, then the case is reduced to admissibility of weakening on relational atoms.

3. If $x = c$, the cases are similar to those for $x = d$.

If the last rule in Π is C, the proof is analogous to the proof for P. □

The rules P and C do not affect the result of weakening, therefore we state the lemma without proof.

Lemma 6.1.3 (Admissibility of weakening). *If $\mathcal{G};\Gamma \vdash \Delta$ is derivable in LS_{PASL}, then for any multiset $\mathcal{G}'$ of relational atoms, any multiset Γ' and Δ' of labelled formulae, the sequent $\mathcal{G};\mathcal{G}';\Gamma;\Gamma' \vdash \Delta;\Delta'$ is derivable with the same height in LS_{PASL}.*

Invertibility needs more care. We give the parts of proof related to the new rules.

Lemma 6.1.4 (Invertibility). *If Π is a cut-free LS_{PASL} derivation of the conclusion of a rule, then there is a cut-free LS_{PASL} derivation for each premise, with height at most $ht(\Pi)$.*

Proof. The rules P, C themselves are trivially invertible, since the inverted versions can be proved by using Lemma 6.1.2. The invertibility of all other rules except for $\top^* L$ can be proved similarly as in LS_{BBI}. Here we show the proof for $\top^* L$ in LS_{PASL}. We do an induction on the height of the derivation. Base case is the same as the proof for LS_{BBI}. For the inductive case, we illustrate the cases where the last rule in the derivation is P or C. Assume w.l.o.g. that the principal formula for the rule $\top^* L$ is $x : \top^*$.

1. If the last rule is P, which runs as below.

$$\frac{(\mathcal{G};(a,b \rhd c);\Gamma;x:\top^* \vdash \Delta)[c/d]}{\mathcal{G};(a,b \rhd c);(a,b \rhd d);\Gamma;x:\top^* \vdash \Delta}\ P$$

   we distinguish three sub-cases:

   (a) If $x \neq d$ and $x \neq c$, then the substitutions $[\epsilon/x]$ and $[c/d]$ are independent. Thus we can use the induction hypothesis and apply the rule $\top^* L$ (meanwhile switch the order of substitutions) to obtain the desired derivation.

   (b) If $x = d$, the original derivation is as follows.

$$\frac{\begin{array}{c}\Pi\\ (\mathcal{G};(a,b \rhd c);c:\top^*;\Gamma \vdash \Delta)[c/d]\end{array}}{\mathcal{G};(a,b \rhd c);(a,b \rhd d);d:\top^*;\Gamma \vdash \Delta}\ P$$

   We apply the induction hypothesis on the premise, then apply P to obtain the following derivation:

$$\frac{\frac{\begin{array}{c}\Pi'\\ (\mathcal{G};(a,b \rhd \epsilon);\Gamma \vdash \Delta)[c/d][\epsilon/c]\end{array}}{(\mathcal{G};(a,b \rhd \epsilon);\Gamma \vdash \Delta)[\epsilon/d][\epsilon/c]}}{(\mathcal{G};(a,b \rhd c);(a,b \rhd \epsilon);\Gamma \vdash \Delta)[\epsilon/d]}\ P$$

   (c) If $x = c$, the case is similar.

2. The case where the last rule is C is similar to above. □

Admissibility of Contraction is also easy to see. The lemma is shown below.

Lemma 6.1.5 (Admissibility of contraction). *If $\mathcal{G};\mathcal{G};\Gamma;\Gamma \vdash \Delta;\Delta$ is derivable in LS_{PASL}, then $\mathcal{G};\Gamma \vdash \Delta$ is derivable with the same height in LS_{PASL}.*

To show this lemma we need to go through two cases: contraction on relational atoms and labelled formulae. As the new rules P, C only involve global substitutions, the proofs for Lemma 4.2.8, 4.2.9 also work for LS_{PASL}. The case where a relational atom is contracted, then the pair of identical relational atoms is used in a P (resp. C) application does not pose a problem, since in this case the substitution does not change the conclusion at all, so the contraction and the rule P (resp. C) applied on the conclusion is admissible.

The lemmas in this section are presented in an order such that the next lemma depends on the previous ones. For example, the admissibility of contraction relies on invertibility of certain rules, which in turn requires admissibility of weakening. These details can be found in Section 4.2. Finally, we give the cut-elimination theorem, which can be proven the same way as for LS_{BBI}.

Theorem 6.1.6 (Cut-elimination)**.** *If $\mathcal{G}; \Gamma \vdash \Delta$ is derivable in LS_{PASL} then it is derivable without using the cut rule.*

Proof. The proof follows the same structure as that for LS_{BBI} (Theorem 4.2.11), utilising the lemmas above. The additional cases we need to consider are those involving the rules P and C; their treatment is similar to that for Eq_1 in the proof for LS_{BBI}. We show the case for P here. The case for C is analogous. These cases both belong to the subcase "the cut formula is not principal in the left premise" of Theorem 4.2.11. Suppose the original derivation runs as below:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ ((x, y \rhd z); \mathcal{G}; \Gamma \vdash x : A; \Delta)[z/w]\end{array}}{(x, y \rhd z); (x, y \rhd w); \mathcal{G}; \Gamma \vdash x : A; \Delta}\, P \qquad \begin{array}{c}\Pi_2\\ \mathcal{G}'; \Gamma'; x : A \vdash \Delta'\end{array}}{(x, y \rhd z); (x, y \rhd w); \mathcal{G}; \mathcal{G}'; \Gamma; \Gamma' \vdash \Delta; \Delta'}\, cut$$

We can permute this application of *cut* upwards through the P application as follows:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ ((x, y \rhd z); \mathcal{G}; \Gamma \vdash x : A; \Delta)[z/w]\end{array} \qquad \begin{array}{c}\Pi_2'\\ (\mathcal{G}'; \Gamma'; x : A \vdash \Delta')[z/w]\end{array}}{((x, y \rhd z); \mathcal{G}; \mathcal{G}'; \Gamma; \Gamma' \vdash \Delta; \Delta')[z/w]}\, cut}{(x, y \rhd z); (x, y \rhd w); \mathcal{G}; \mathcal{G}'; \Gamma; \Gamma' \vdash \Delta; \Delta'}\, P$$

That is, the new *cut* application cuts on the formula $x : A[z/w]$ instead, which has the same cut complexity as the original *cut* application. The derivation Π_2' for the right premise is obtained by applying the substitution lemma on the derivation Π_2. □

Reflecting on the above proofs, the rules P and C do not break the nice properties related to cut-elimination, and the proofs share the same idea as the one presented for LS_{BBI}. This is partly because the two new rules only involve global substitutions that work similarly to the old rules Eq_1 and Eq_2. We will further exploit rules that only involve substitutions in later sections.

On the other hand, since partial-determinism and cancellativity are not axiomatisable in BBI [22], there does not exist a sound and complete Hilbert system for PASL that we can simulate using LS_{PASL}, and cut-elimination does not immediately yield the completeness of LS_{PASL}; we prove completeness of our calculus in the next section.

6.2 Completeness of LS_{PASL}

We prove the completeness of LS_{PASL} with respect to the Kripke relational semantics by a counter-model construction. A standard way to construct a counter-model for an unprovable sequent is to show that it can be saturated by repeatedly applying all applicable inference rules to reach a limit sequent where a counter-model can be constructed. In adopting such a counter-model construction strategy to LS_{PASL} we encounter difficulty in formulating the saturation conditions for rules involving label substitutions. We therefore adopt the approach in Section 4.3, using an intermediate system without explicit use of label substitutions, but where equivalences between labels are captured via an entailment $\vdash_E$. Again, in the sequel, we assume that labelled sequents such as $\mathcal{G}; \Gamma \vdash \Delta$ are built from **sets** $\mathcal{G}, \Gamma, \Delta$ rather than **multisets**. This is harmless since contraction is admissible in LS_{PASL}.

6.2.1 The Intermediate System ILS_{PASL}

We introduce an intermediate system where rules with substitutions (Eq_1, Eq_2, P, C) are isolated into an equivalence entailment $\vdash_E$, so that the resultant calculus does not involve explicit substitutions.

We recall the definition of the function $\mathcal{S}(\mathcal{G}, \sigma)$ at the beginning of Section 4.3.1 here. Given a set of relational atoms $\mathcal{G}$, we denote with $LV(\mathcal{G})$ the set of label variables in $\mathcal{G}$. Let σ be a sequence (list) of abstract instances of structural rules $[\mathbf{r}_1; \cdots; \mathbf{r}_n]$. Given a set of relational atoms $\mathcal{G}$, the result of the application of σ to $\mathcal{G}$, denoted by $\mathcal{S}(\mathcal{G}, \sigma)$, is defined inductively as follows:

$$\mathcal{S}(\mathcal{G}, \sigma) \quad = \quad \begin{cases} \mathcal{G} & \text{if} \quad \sigma = [\,] \\ \mathcal{S}(\mathcal{G}\theta \cup \mathcal{G}_2, \sigma') & \text{if} \quad \mathcal{G}_1 \subseteq \mathcal{G}, \sigma = [r(\mathcal{G}_1, \theta, \mathcal{V}, \mathcal{G}_2)]@\sigma' \text{ and } LV(\mathcal{G}) \cap \mathcal{V} = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

Given a sequence $\sigma = [r_1(\mathcal{G}_1, \theta_1, \mathcal{V}_1, \mathcal{G}_1'); \cdots; r_n(\mathcal{G}_n, \theta_n, \mathcal{V}_n, \mathcal{G}_n')]$ of structural rule applications, we denote with $subst(\sigma)$ the composite substitution $\theta_1 \circ \cdots \circ \theta_n$, where $t(\theta_1 \circ \theta_2)$ means $(t\theta_1)\theta_2$.

Definition 6.2.1 (Equivalence entailment)**.** *Let $\mathcal{G}$ be a set of relational atoms. The entailment $\mathcal{G} \vdash_E (a = b)$ holds iff there exists a sequence σ of Eq_1, Eq_2, P, C applications s.t. $\mathcal{S}(\mathcal{G}, \sigma)$ is defined, and $a\theta = b\theta$, where $\theta = subst(\sigma)$.*

Since substitution is no longer in the calculus, some inference rules that involve matching two equal labels need to be changed. We define the intermediate system ILS_{PASL} as LS_{PASL} minus $\{Eq_1, Eq_2, P, C\}$, with certain rules changed following Fig. 6.2. The equivalence entailment $\vdash_E$ is not a premise, but rather a condition of the rules. One can readily notice that the rules in Figure 6.2 are just notational variants of the rules in Figure 4.3, as the two systems LS_{BBI} and LS_{PASL} have the same rules when substitutions are absorbed into $\vdash_E$.

$$\frac{\mathcal{G} \vdash_E (w_1 = w_2)}{\mathcal{G}; \Gamma; w_1 : p \vdash w_2 : p; \Delta}\ id \qquad \frac{\mathcal{G} \vdash_E (w = \epsilon)}{\mathcal{G}; \Gamma \vdash w : \top^*; \Delta}\ \top^* R$$

$$\frac{(x, w \triangleright x'); (y, y \triangleright w); (x, y \triangleright x'); \mathcal{G}; \Gamma \vdash \Delta}{(x, y \triangleright x'); \mathcal{G}; \Gamma \vdash \Delta}\ A_C$$

$$\frac{(u, w \triangleright z); (y, v \triangleright w); (x, y \triangleright z); (u, v \triangleright x'); \mathcal{G}; \Gamma \vdash \Delta}{(x, y \triangleright z); (u, v \triangleright x'); \mathcal{G}; \Gamma \vdash \Delta}\ A$$

$$\frac{(x, y \triangleright w'); \mathcal{G}; \Gamma \vdash x : A; w : A * B; \Delta \qquad (x, y \triangleright w'); \mathcal{G}; \Gamma \vdash y : B; w : A * B; \Delta}{(x, y \triangleright w'); \mathcal{G}; \Gamma \vdash w : A * B; \Delta}\ *R$$

$$\frac{(x, w' \triangleright z); \mathcal{G}; \Gamma; w : A \mathrel{-\!\!*} B \vdash x : A; \Delta \qquad (x, w' \triangleright z); \mathcal{G}; \Gamma; w : A \mathrel{-\!\!*} B; z : B \vdash \Delta}{(x, w' \triangleright z); \mathcal{G}; \Gamma; w : A \mathrel{-\!\!*} B \vdash \Delta}\ \mathrel{-\!\!*} L$$

Side conditions:

In A, A_C, the label w does not occur in the conclusion.
In A_C, $(x, y \triangleright x'); \mathcal{G} \vdash_E (x = x')$ In A, $(x, y \triangleright z); (u, v \triangleright x'); \mathcal{G} \vdash_E (x = x')$
In $*R$, $(x, y \triangleright w'); \mathcal{G} \vdash_E (w = w')$ In $\mathrel{-\!\!*}\ L$, $(x, w' \triangleright z); \mathcal{G} \vdash_E (w = w')$

Figure 6.2: Changed rules in the intermediate system ILS_{PASL}.

Given a set of relational atoms $\mathcal{G}$, we define the relation $=_{\mathcal{G}}$ as follows: $a =_{\mathcal{G}} b$ iff $\mathcal{G} \vdash_E (a = b)$. We show next that $=_{\mathcal{G}}$ is in fact an equivalence relation. This equivalence relation will be useful in our counter-model construction later. Recall from Section 4.3 that we use σ to stand for a sequence of structural rule applications.

Lemma 6.2.1. *Let $\mathcal{G}$ be a set of relational atoms. If $\mathcal{G} \vdash_E (a = b)$ by applying σ_1 and $\mathcal{G} \vdash_E (c = d)$ by applying σ_2, then $\exists \sigma_3$ s.t. $\mathcal{S}(\mathcal{G}, \sigma_1) \vdash_E (c\theta = d\theta)$ by σ_3, where $\theta = subst(\sigma_1)$.*

Proof. Note that $\mathcal{S}(\mathcal{G}, \sigma_1) = \mathcal{G}\theta$. So essentially we need to show that if $\mathcal{G} \vdash_E (c = d)$, then $\mathcal{G}\theta \vdash_E (c\theta = d\theta)$. This is a consequence of the substitution Lemma 6.1.2. □

Lemma 6.2.2. *Given a set of relational atoms $\mathcal{G}$, the relation $=_{\mathcal{G}}$ is an equivalence relation on the set of labels.*

Proof. We show that $\vdash_E$ satisfies the following conditions:

Reflexivity: for any label a, we have $\mathcal{G} \vdash_E (a = a)$ by applying an empty sequence of Eq_1, Eq_2, P, C rules.

Symmetry: if $\mathcal{G} \vdash_E (x = y)$, via a sequence σ of Eq_1, Eq_2, P, C applications. Let $\theta = subst(\sigma)$, then by definition, $x\theta$ and $y\theta$ are syntactically identical. Thus $y\theta \equiv x\theta$. We obtain that $\mathcal{G} \vdash_E (y = x)$.

Note that if one of x and y is not in $\mathcal{G}$, any structural rule application on $\mathcal{G}$ will not derive any relation between x and y unless x and y are the same label. If x and y are the same label, symmetry is immediate.

Transitivity: if $\mathcal{G} \vdash_E (x = y)$ and $\mathcal{G} \vdash_E (y = z)$, then by Lemma 6.2.1 we obtain a sequence σ of Eq_1, Eq_2, P, C applications, with $\theta = subst(\sigma)$. Then $x\theta \equiv y\theta \equiv z\theta$. Thus $\mathcal{G} \vdash_E (x = z)$.

Again, if one of x, y and z is not in $\mathcal{G}$, transitivity is immediate. □

The intermediate system ILS_{PASL} is equivalent to LS_{PASL}. This connection is easy to make, as is shown in Section 4.3. We just need to additionally consider that the rules P, C are captured by the equivalence entailment $\vdash_E$. This is straightforward from the definition. Properties such as contraction admissibility, closure under substitution etc. also hold for ILS_{PASL}.

Lemma 6.2.3. *The intermediate labelled calculus ILS_{PASL} is equivalent to LS_{PASL}. That is, every sequent provable in ILS_{PASL} is also provable in LS_{PASL}, and vice versa.*

6.2.2 Counter-model Construction

We now give a counter-model construction procedure for ILS_{PASL} which, by Lemma 6.2.3, applies to LS_{PASL} as well.

As the counter-model construction involves infinite sets and sequents, we extend the definition of $\vdash_E$ appropriately as below.

Definition 6.2.2. *A (possibly infinite) set $\mathcal{G}$ of relational atoms satisfies $\mathcal{G} \vdash_E (x = y)$ iff $\mathcal{G}_f \vdash_E (x = y)$ for some finite $\mathcal{G}_f \subseteq \mathcal{G}$.*

Given a set of relational atoms $\mathcal{G}$, the equivalence relation $=_{\mathcal{G}}$ partitions the set $\mathcal{L}$ of labels into equivalence classes $[a]_{\mathcal{G}}$ for each label $a \in \mathcal{L}$:

$$[a]_{\mathcal{G}} = \{a' \in \mathcal{L} \mid a =_{\mathcal{G}} a'\}.$$

The counter-model construction is essentially a procedure to saturate a sequent by applying all applicable rules repeatedly. The aim is to obtain a possibly infinite saturated sequent from which a counter-model can be extracted. We first define a list of desired properties of such a saturated sequent, by tradition is called the *Hintikka sequent*, which would allow the counter-model construction.

Definition 6.2.3 (Hintikka sequent)**.** *A labelled sequent $\mathcal{G}; \Gamma \vdash \Delta$ is a* Hintikka sequent *if it satisfies the following conditions for any formulae A, B and any labels a, a', b, c, d, e, z:*

1. *If* $a : A \in \Gamma$ *and* $b : A \in \Delta$ *then* $a \neq_{\mathcal{G}} b$.
2. *If* $a : A \wedge B \in \Gamma$ *then* $a : A \in \Gamma$ *and* $a : B \in \Gamma$.
3. *If* $a : A \wedge B \in \Delta$ *then* $a : A \in \Delta$ *or* $a : B \in \Delta$.
4. *If* $a : A \to B \in \Gamma$ *then* $a : A \in \Delta$ *or* $a : B \in \Gamma$.
5. *If* $a : A \to B \in \Delta$ *then* $a : A \in \Gamma$ *and* $a : B \in \Delta$.
6. *If* $a : \top^* \in \Gamma$ *then* $a =_{\mathcal{G}} \epsilon$.
7. *If* $a : \top^* \in \Delta$ *then* $a \neq_{\mathcal{G}} \epsilon$.
8. *If* $z : A * B \in \Gamma$ *then* $\exists x, y, z' \in \mathcal{L}$ *s.t.* $(x, y \rhd z') \in \mathcal{G}$, $z =_{\mathcal{G}} z'$, $x : A \in \Gamma$ *and* $y : B \in \Gamma$.
9. *If* $z : A * B \in \Delta$ *then* $\forall x, y, z' \in \mathcal{L}$ *if* $(x, y \rhd z') \in \mathcal{G}$ *and* $z =_{\mathcal{G}} z'$ *then* $x : A \in \Delta$ *or* $y : B \in \Delta$.
10. *If* $z : A \mathrel{-\!\!*} B \in \Gamma$ *then* $\forall x, y, z' \in \mathcal{L}$ *if* $(x, z' \rhd y) \in \mathcal{G}$ *and* $z =_{\mathcal{G}} z'$, *then* $x : A \in \Delta$ *or* $y : B \in \Gamma$.
11. *If* $z : A \mathrel{-\!\!*} B \in \Delta$ *then* $\exists x, y, z' \in \mathcal{L}$ *s.t.* $(x, z' \rhd y) \in \mathcal{G}$, $z =_{\mathcal{G}} z'$, $x : A \in \Gamma$ *and* $y : B \in \Delta$.
12. *For any label* $m \in \mathcal{L}$, $(m, \epsilon \rhd m) \in \mathcal{G}$.
13. *If* $(a, b \rhd c) \in \mathcal{G}$ *then* $(b, a \rhd c) \in \mathcal{G}$.
14. *If* $\{(a, b \rhd c), (d, e \rhd a')\} \subseteq \mathcal{G}$ *and* $a =_{\mathcal{G}} a'$, *then* $\exists f, f' \in \mathcal{L}$ *s.t.* $\{(d, f \rhd c), (b, e \rhd f')\} \subseteq \mathcal{G}$ *and* $f =_{\mathcal{G}} f'$.
15. $a : \bot \notin \Gamma$ *and* $a : \top \notin \Delta$.

The next lemma shows that a Hintikka sequent gives a PASL Kripke relational frame which is a (counter-)model of the sequent. Given a PASL model (H, R, ϵ, ν), we define an extended model as $(H, R, \epsilon, \nu, \rho)$ where ρ is a mapping from labels to members of H.

Lemma 6.2.4. *Every Hintikka sequent is falsifiable.*

Proof. Let $\mathcal{G}; \Gamma \vdash \Delta$ be a Hintikka sequent. We construct an extended model $\mathcal{M} = (H, \rhd_{\mathcal{G}}, \epsilon_{\mathcal{G}}, \nu, \rho)$ as follows:

- $H = \{[a]_{\mathcal{G}} \mid a \in \mathcal{L}\}$
- $\rhd_{\mathcal{G}}([a]_{\mathcal{G}}, [b]_{\mathcal{G}}, [c]_{\mathcal{G}})$ iff $\exists a', b', c'.(a', b' \rhd c') \in \mathcal{G}$, $a =_{\mathcal{G}} a'$, $b =_{\mathcal{G}} b'$, $c =_{\mathcal{G}} c'$
- $\epsilon_{\mathcal{G}} = [\epsilon]_{\mathcal{G}}$
- $\nu(p) = \{[a]_{\mathcal{G}} \mid a : p \in \Gamma\}$ for every $p \in PVar$
- $\rho(a) = [a]_{\mathcal{G}}$ for every $a \in \mathcal{L}$

To reduce clutter, we shall drop the subscript $\mathcal{G}$ in $[a]_{\mathcal{G}}$ and write $[a], [b] \rhd_{\mathcal{G}} [c]$ instead of $\rhd_{\mathcal{G}}([a], [b], [c])$.

We first show that $\mathcal{F} = (H, \rhd_{\mathcal{G}}, \epsilon_{\mathcal{G}})$ is a PASL Kripke relational frame.

identity: for each $[a] \in H$, by definition, there must be a label $a' \in \mathcal{L}$ such that $[a] = [a']$. It follows from condition 12 in Def. 6.2.3 that $(a', \epsilon \rhd a') \in \mathcal{G}$, thus $[a], [\epsilon] \rhd_{\mathcal{G}} [a]$ holds. By condition 13, we have $[\epsilon], [a] \rhd_{\mathcal{G}} [a]$. On the other hand, assume there is

some $[\epsilon], [a] \triangleright_{\mathcal{G}} [b]$. There must be some a', b' such that $[a] = [a']$, $[b] = [b']$, and $(\epsilon, a' \triangleright b') \in \mathcal{G}$. By the rules Eq_1 and Eq_2, we obtain that $\mathcal{G} \vdash_E (a' = b')$. Thus $[a] = [a'] = [b'] = [b]$.

commutativity: if $[a], [b] \triangleright_{\mathcal{G}} [c]$ holds, there must be some $(a', b' \triangleright c') \in \mathcal{G}$ s.t. $[a] = [a'], [b] = [b'], [c] = [c']$. Then by condition 13 in Def. 6.2.3, $(b', a' \triangleright c') \in \mathcal{G}$, therefore $[b], [a] \triangleright_{\mathcal{G}} [c]$ holds.

associativity: if $[a], [b] \triangleright_{\mathcal{G}} [c]$ and $[d], [e] \triangleright_{\mathcal{G}} [a]$ holds, then there exist some $(a', b' \triangleright c') \in \mathcal{G}$ and $(d', e' \triangleright a'') \in \mathcal{G}$ s.t. $[a] = [a'] = [a'']$, $[b] = [b']$, $[c] = [c']$, $[d] = [d']$, $[e] = [e']$. Then by condition 14 in Def. 6.2.3, there also exist labels f, f' s.t. $(d', f \triangleright c') \in \mathcal{G}$ and $(b', e' \triangleright f') \in \mathcal{G}$ and $[f] = [f']$. Thus we can find $[f]$ s.t. $[d], [f] \triangleright_{\mathcal{G}} [c]$ and $[b], [e] \triangleright_{\mathcal{G}} [f]$ hold.

Partial-determinism: If $[a], [b] \triangleright_{\mathcal{G}} [c]$ and $[a], [b] \triangleright_{\mathcal{G}} [d]$ hold, then there exists some $(a', b' \triangleright c') \in \mathcal{G}$ and $(a'', b'' \triangleright d') \in \mathcal{G}$ s.t. $[a] = [a'] = [a'']$, $[b] = [b'] = [b'']$, $[c] = [c']$, $[d] = [d']$. So there are some σ_1, σ_2 such that $\mathcal{G} \vdash_E (a' = a'')$ via σ_1 and $\mathcal{G} \vdash_E (b' = b'')$ via σ_2. Let θ_1 be $subst(\sigma_1)$. By Lemma 6.2.1, there is some σ_3 such that $\mathcal{S}(\mathcal{G}, \sigma_1) \vdash_E (b'\theta_1 = b''\theta_1)$. Let θ_2 be $subst(\sigma_3)$. We have $a'\theta_1\theta_2 \equiv a''\theta_1\theta_2$ and $b'\theta_1\theta_2 \equiv b''\theta_1\theta_2$ in $\mathcal{S}(\mathcal{S}(\mathcal{G}, \sigma_1), \sigma_3)$. So we can apply the rule P on $\mathcal{S}(\mathcal{S}(\mathcal{G}, \sigma_1), \sigma_3)$ to unify $c'\theta_1\theta_2$ and $d'\theta_1\theta_2$, obtaining $[c] = [c'] = [d] = [d']$.

Cancellativity: if $[a], [b] \triangleright_{\mathcal{G}} [c]$ and $[a], [d] \triangleright_{\mathcal{G}} [c]$ hold, then we can find some $(a', b' \triangleright c') \in \mathcal{G}$ and $(a'', d' \triangleright c'') \in \mathcal{G}$ s.t. $[a] = [a'] = [a'']$, $[c] = [c'] = [c'']$, $[b] = [b']$, $[d] = [d']$. By a similar reasoning as the above case, we obtain that $[b] = [b'] = [c] = [c']$.

So $\mathcal{M}$ is indeed a model based on a PASL Kripke relational frame. We prove next that $\mathcal{G}; \Gamma \vdash \Delta$ is false in $\mathcal{M}$. We need to show the following (where $\rho(m) = [m]$):

(1) If $(a, b \triangleright c) \in \mathcal{G}$ then $([a], [b] \triangleright_{\mathcal{G}} [c])$.
(2) If $m : A \in \Gamma$ then $\rho(m) \Vdash A$.
(3) If $m : A \in \Delta$ then $\rho(m) \nVdash A$.

Item (1) follows from the definition of $\triangleright_{\mathcal{G}}$. We prove (2) and (3) simultaneously by induction on the size of A.

Base cases: when A is an atomic proposition p.

- If $m : p \in \Gamma$ then $[m] \in \nu(p)$ by definition of ν, so $[m] \Vdash p$.
- Suppose $m : p \in \Delta$, but $[m] \Vdash p$. Then $m' : p \in \Gamma$, for some m' s.t. $m' =_{\mathcal{G}} m$. This violates condition 1 in Def. 6.2.3. Thus $[m] \nVdash p$.

Inductive cases: when A is a compound formula. We do a case analysis on the main connective of A.

- If $m : A \wedge B \in \Gamma$, by condition 2 in Def. 6.2.3, $m : A \in \Gamma$ and $m : B \in \Gamma$. By the induction hypothesis, $[m] \Vdash A$ and $[m] \Vdash B$, thus $[m] \Vdash A \wedge B$.
- If $m : A \wedge B \in \Delta$, by condition 3 in Def. 6.2.3, $m : A \in \Delta$ or $m : B \in \Delta$. By the induction hypothesis, $[m] \nVdash A$ or $[m] \nVdash B$, thus $[m] \nVdash A \wedge B$.
- If $m : A \to B \in \Gamma$, by condition 4 in Def. 6.2.3, $m : A \in \Delta$ or $m : B \in \Gamma$. By the induction hypothesis, $[m] \nVdash A$ or $[m] \Vdash B$, thus $[m] \Vdash A \to B$.
- If $m : A \to B \in \Delta$, by condition 5 in Def. 6.2.3, $m : A \in \Gamma$ and $m : B \in \Delta$. By the induction hypothesis, $[m] \Vdash A$ and $[m] \nVdash B$, thus $[m] \nVdash A \to B$.
- If $m : \top^* \in \Gamma$ then $[m] = [\epsilon]$ by condition 6 in Def. 6.2.3. Since $[\epsilon] \Vdash \top^*$, we obtain $[m] \Vdash \top^*$.
- If $m : \top^* \in \Delta$, by condition 7 in Def. 6.2.3, $[m] \neq [\epsilon]$ and then $[m] \nVdash \top^*$.
- If $m : A * B \in \Gamma$, by condition 8 in Def. 6.2.3, $\exists a, b, m'$ s.t. $(a, b \rhd m') \in \mathcal{G}$ and $[m] = [m']$ and $a : A \in \Gamma$ and $b : B \in \Gamma$. By the induction hypothesis, $[a] \Vdash A$ and $[b] \Vdash B$. Thus $[a], [b] \rhd_{\mathcal{G}} [m]$ holds and $[m] \Vdash A * B$.
- If $m : A * B \in \Delta$, by condition 9 in Def. 6.2.3, $\forall a, b, m'$ if $(a, b \rhd m') \in \mathcal{G}$ and $[m] = [m']$, then $a : A \in \Delta$ or $b : B \in \Delta$. By the induction hypothesis, if such a, b exist, then $[a] \nVdash A$ or $[b] \nVdash B$. For any $[a], [b] \rhd_{\mathcal{G}} [m]$, there must be some $(a', b' \rhd m'') \in \mathcal{G}$ s.t. $[a] = [a'], [b] = [b'], [m] = [m'']$. Then $[a] \nVdash A$ or $[b] \nVdash B$ therefore $[m] \nVdash A * B$.
- If $m : A \mathrel{-\!\!*} B \in \Gamma$, by condition 10 in Def. 6.2.3, $\forall a, b, m'$ if $(a, m' \rhd b) \in \mathcal{G}$ and $[m] = [m']$, then $a : A \in \Delta$ or $b : B \in \Gamma$. By the induction hypothesis, if such a, b exists, then $[a] \nVdash A$ or $[b] \Vdash B$. Consider any $[a], [m] \rhd_{\mathcal{G}} [b]$. There must be some $(a', m'' \rhd b') \in \mathcal{G}$ s.t. $[a] = [a']$, $[m''] = [m]$, and $[b] = [b']$. So $[a] \nVdash A$ or $[b] \Vdash B$, thus $[m] \Vdash A \mathrel{-\!\!*} B$.
- If $m : A \mathrel{-\!\!*} B \in \Delta$, by condition 11 in Def. 6.2.3, $\exists a, b, m'$ s.t. $(a, m' \rhd b) \in \mathcal{G}$ and $[m] = [m']$ and $a : A \in \Gamma$ and $b : B \in \Delta$. By the induction hypothesis, $[a] \Vdash A$ and $[b] \nVdash B$ and $[a], [m] \rhd_{\mathcal{G}} [b]$ holds, thus $[m] \nVdash A \mathrel{-\!\!*} B$. □

To prove the completeness of ILS_{PASL}, we have to show that any given unprovable sequent can be extended to a Hintikka sequent. To do so we need a way to enumerate all possible applicable rules in a fair way so that every rule will be chosen infinitely often. Traditionally, this is achieved via a fair enumeration strategy of every principal formula of every rule. Since our calculus contains structural rules with no principal formulas, we need to include them in the enumeration strategy as well. For this purpose, we define a notion of *extended formulae*, given by the grammar:

$$ExF ::= F \mid \mathbb{U} \mid \mathbb{E} \mid \mathbb{A} \mid \mathbb{A}_C$$

where F is a formula, and $\mathbb{U}, \mathbb{E}, \mathbb{A}, \mathbb{A}_C$ are constants that are used as "dummy" principal formulae for the structural rules U, E, A, and A_C, respectively. A schedule enumerates each combination of left or right of turnstile, a label, an extended formula and at most two relational atoms infinitely often.

Definition 6.2.4 (Schedule ϕ)**.** *A* rule instance *is a tuple* (O, m, ExF, R)*, where* O *is either* 0 *(left) or* 1 *(right),* m *is a label,* ExF *is an extended formula and* R *is a set of relational atoms such that* $|R| \leq 2$*. Let* $\mathcal{S}$ *denote the set of all rule instances. A* schedule *is a function from natural numbers* $\mathbb{N}$ *to* $\mathcal{S}$*. A schedule* ϕ *is* fair *if for every rule instance* S*, the set* $\{i \mid \phi(i) = S\}$ *is infinite.*

Lemma 6.2.5. *There exists a fair schedule.*

Proof. Our proof is similar to the proof of *fair strategy* of Larchey-Wendling [60]. To adapt their proof, we need to show that the set $\mathcal{S}$ is countable. This follows from the fact that $\mathcal{S}$ is a finite product of countable sets. □

From now on, we shall fix a fair schedule, which we call ϕ. We assume that the set of labels $\mathcal{L}$ is totally ordered, and its elements can be enumerated as $a_0, a_1, a_2, \ldots$ where $a_0 = \epsilon$. This indexing is used to select fresh labels in our construction of Hintikka sequents. We say the formula F is not cut-free provable in ILS_{PASL} if the sequent $\vdash w : F$ is not cut-free derivable in ILS_{PASL} for any label $w \neq \epsilon$. Since we shall be concerned only with cut-free provability, in the following when we mention derivation, we mean cut-free derivation.

Definition 6.2.5. *Let* F *be a formula which is not provable in* ILS_{PASL}*. We construct a series of finite sequents* $\langle \mathcal{G}_i; \Gamma_i \vdash \Delta_i \rangle_{i \in \mathcal{N}}$ *from* F *where* $\mathcal{G}_1 = \Gamma_1 = \emptyset$ *and* $\Delta_1 = a_1 : F$*.*

Assuming that $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ *has been defined, we define* $\mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$ *as follows. Suppose* $\phi(i) = (O_i, m_i, ExF_i, R_i)$*.*

- *If* $O_i = 0$*,* ExF_i *is a PASL formula* C_i *and* $m_i : C_i \in \Gamma_i$*:*
  - *If* $C_i = F_1 \wedge F_2$*, then* $\mathcal{G}_{i+1} = \mathcal{G}_i$*,* $\Gamma_{i+1} = \Gamma_i \cup \{m_i : F_1, m_i : F_2\}$*,* $\Delta_{i+1} = \Delta_i$*.*
  - *If* $C_i = F_1 \to F_2$*. If there is no derivation for* $\mathcal{G}_i; \Gamma_i \vdash m_i : F_1; \Delta_i$ *then* $\Gamma_{i+1} = \Gamma_i$*,* $\Delta_{i+1} = \Delta_i \cup \{m_i : F_1\}$*. Otherwise* $\Gamma_{i+1} = \Gamma_i \cup \{m_i : F_2\}$*,* $\Delta_{i+1} = \Delta_i$*. In both cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$*.*
  - *If* $C_i = \top^*$*, then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(\epsilon, m_i \triangleright \epsilon)\}$*,* $\Gamma_{i+1} = \Gamma_i$*,* $\Delta_{i+1} = \Delta_i$*.*
  - *If* $C_i = F_1 * F_2$*, then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{2i}, a_{2i+1} \triangleright m_i)\}$*,* $\Gamma_{i+1} = \Gamma_i \cup \{a_{2i} : F_1, a_{2i+1} : F_2\}$*,* $\Delta_{i+1} = \Delta_i$*.*
  - *If* $C_i = F_1 -\!\!* F_2$ *and* $R_i = \{(x, m \triangleright y)\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (m = m_i)$*. If* $\mathcal{G}_i; \Gamma_i \vdash x : F_1; \Delta_i$ *has no derivation, then* $\Gamma_{i+1} = \Gamma_i$*,* $\Delta_{i+1} = \Delta_i \cup \{x : F_1\}$*. Otherwise* $\Gamma_{i+1} = \Gamma_i \cup \{y : F_2\}$*,* $\Delta_{i+1} = \Delta_i$*. In both cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$*.*
- *If* $O_i = 1$*,* ExF_i *is a PASL formula* C_i*, and* $m_i : C_i \in \Delta$*:*
  - *If* $C_i = F_1 \wedge F_2$*. If there is no derivation for* $\mathcal{G}_i; \Gamma_i \vdash m_i : F_1; \Delta_i$ *then* $\Delta_{i+1} = \Delta_i \cup \{m_i : F_1\}$*. Otherwise* $\Delta_{i+1} = \Delta_i \cup \{m_i : F_2\}$*. In both cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$ *and* $\Gamma_{i+1} = \Gamma_i$*.*

- *If* $C_i = F_1 \to F_2$, *then* $\Gamma_{i+1} = \Gamma \cup \{m_i : F_1\}$, $\Delta_{i+1} = \Delta_i \cup \{m_i : F_2\}$, *and* $\mathcal{G}_{i+1} = \mathcal{G}_i$.
- $C_i = F_1 * F_2$ *and* $R_i = \{(x, y \rhd m)\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (m_i = m)$. *If* $\mathcal{G}_i; \Gamma_i \vdash x : F_1; \Delta_i$ *has no derivation, then* $\Delta_{i+1} = \Delta_i \cup \{x : F_1\}$. *Otherwise* $\Delta_{i+1} = \Delta_i \cup \{y : F_2\}$. *In both cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$ *and* $\Gamma_{i+1} = \Gamma_i$.
- *If* $C_i = F_1 -\!\!* F_2$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{2i}, m_i \rhd a_{2i+1})\}$, $\Gamma_{i+1} = \Gamma_i \cup \{a_{2i} : F_1\}$, *and* $\Delta_{i+1} = \Delta_i \cup \{a_{2i+1} : F_2\}$.

- *If* $ExF_i \in \{\mathbb{U}, \mathbb{E}, \mathbb{A}, \mathbb{A}_C\}$, *we proceed as follows:*
  - *If* $ExF_i = \mathbb{U}$, $R_i = \{(a_n, \epsilon \rhd a_n)\}$, *where* $n \leq 2i + 1$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_n, \epsilon \rhd a_n)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{E}$, $R_i = \{(x, y \rhd z)\} \subseteq \mathcal{G}_i$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(y, x \rhd z)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{A}$, $R_i = \{(x, y \rhd z); (u, v \rhd x')\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (x = x')$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(u, a_{2i} \rhd z), (y, v \rhd a_{2i})\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{A}_C$, $R_i = \{(x, y \rhd x')\} \subseteq \mathcal{G}_i$, *and* $\mathcal{G}_i \vdash_E (x = x')$ *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(x, a_{2i} \rhd x), (y, y \rhd a_{2i})\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.

- *In all other cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$, $\Gamma_{i+1} = \Gamma_i$ *and* $\Delta_{i+1} = \Delta_i$.

Intuitively, each tuple (O_i, m_i, ExF_i, R_i) corresponds to a potential rule application. If the components of the rule application are in the current sequent, we apply the corresponding rule to these components. The indexing of labels guarantees that the choice of a_{2i} and a_{2i+1} are always fresh for the sequent $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$. The construction in Def. 6.2.5 non-trivially extends a similar construction of Hintikka CSS due to Larchey-Wendling [60], in addition to which we have to consider the cases for structural rules. We also borrow the notions of consistency and finite-consistency from Larchey-Wendling's work [60].

We say $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq \mathcal{G}; \Gamma \vdash \Delta$ iff $\mathcal{G}' \subseteq \mathcal{G}$, $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$. A labelled sequent $\mathcal{G}; \Gamma \vdash \Delta$ is *finite* if $\mathcal{G}, \Gamma, \Delta$ are finite sets. Define $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq_f \mathcal{G}; \Gamma \vdash \Delta$ iff $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq \mathcal{G}; \Gamma \vdash \Delta$ and $\mathcal{G}'; \Gamma' \vdash \Delta'$ is finite. If $\mathcal{G}; \Gamma \vdash \Delta$ is a finite sequent, it is *consistent* iff it does not have a derivation in ILS_{PASL}. A (possibly infinite) sequent $\mathcal{G}; \Gamma \vdash \Delta$ is *finitely-consistent* iff every $\mathcal{G}'; \Gamma' \vdash \Delta' \subseteq_f \mathcal{G}; \Gamma \vdash \Delta$ is consistent.

We write $\mathcal{L}_i$ for the set of labels occurring in the sequent $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$. Thus $\mathcal{L}_1 = \{a_1\}$. The following lemma states some properties of the construction of the sequents $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$, e.g., the labels a_{2i}, a_{2i+1} are always fresh for $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$. This can be proved by an induction on i.

Lemma 6.2.6. *For any* $i \in \mathcal{N}$, *the following properties hold:*

1. $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ *has no derivation*
2. $\mathcal{L}_i \subseteq \{a_0, a_1, \cdots, a_{2i-1}\}$

3. $\mathcal{G}_i; \Gamma_i \vdash \Delta_i \subseteq_f \mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$

Proof. Item 1 is based on the fact that the inference rules preserves falsifiability upwards, and we always choose the branch with no derivation. To show item 2, we do an induction on i. Base case, $i = 1$, $\mathcal{L}_1 \subseteq \{a_0, a_1\}$ (recall that $a_0 = \epsilon$). Inductive cases: suppose item 2 holds for any $i \leq n$, for $n + 1$, we consider five cases depending on which rule is applied on $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$.

1. If $*L$ is applied, then $\mathcal{L}_{i+1} = \mathcal{L}_i \cup \{a_{2i}, a_{2i+1}\} \subseteq \{a_1, \cdots, a_{2i+1}\}$.
2. If $-\!\!* R$ is applied, same as above.
3. If U is applied, which generates $(a_n, \epsilon \triangleright a_n)$, then $n \leq 2i + 1$, thus $\mathcal{L}_{i+1} = \mathcal{L}_i \cup \{a_n\} \subseteq \{a_1, \cdots, a_{2i+1}\}$.
4. If A is applied, the fresh label in the premise is a_{2i}. Thus $\mathcal{L}_{i+1} = \mathcal{L}_i \cup \{a_{2i}\} \subseteq \{a_1, \cdots, a_{2i+1}\}$.
5. Otherwise, $\mathcal{L}_{i+1} = \mathcal{L}_i \subseteq \{a_1, \cdots, a_{2i+1}\}$.

Item 3 is obvious from the construction of $\mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$. □

Given the construction of the series of sequents we have just seen above, we define a notion of a *limit sequent* as the union of every sequent in the series.

Definition 6.2.6 (Limit sequent)**.** *Let F be a formula unprovable in ILS_{PASL}. The* limit sequent for F *is the sequent $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ where $\mathcal{G}^\omega = \bigcup_{i \in \mathcal{N}} \mathcal{G}_i$ and $\Gamma^\omega = \bigcup_{i \in \mathcal{N}} \Gamma_i$ and $\Delta^\omega = \bigcup_{i \in \mathcal{N}} \Delta_i$ and where $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ is as defined in Def.6.2.5.*

The following lemma shows that the limit sequent defined above is indeed a Hintikka sequent. Thus we can use it to extract a counter-model.

Lemma 6.2.7. *If F is a formula unprovable in ILS_{PASL}, then the limit labelled sequent for F is a Hintikka sequent.*

Proof. Let $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ be the limit sequent. First we show that $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ is finitely-consistent. Consider any $\mathcal{G}; \Gamma \vdash \Delta \subseteq_f \mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$, we show that $\mathcal{G}; \Gamma \vdash \Delta$ has no derivation. Since $\mathcal{G}, \Gamma, \Delta$ are finite sets, there exists $i \in \mathcal{N}$ s.t. $\mathcal{G} \subseteq \mathcal{G}_i$, $\Gamma \subseteq \Gamma_i$, and $\Delta \subseteq \Delta_i$. Moreover, $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ is not provable in ILS_{PASL}. Since weakening is admissible in ILS_{PASL}, $\mathcal{G}; \Gamma \vdash \Delta \subseteq_f \mathcal{G}_i; \Gamma_i \vdash \Delta_i$ cannot be provable either. So condition 1, 7, and 15 in Definition 6.2.3 hold for the limit sequent, for otherwise we would be able to construct a provable finite labelled sequent from the limit sequent. We show the proofs that the other conditions in Definition 6.2.3 are also satisfied by the limit sequent. The following cases are numbered according to items in Definition 6.2.3.

2. If $m : F_1 \wedge F_2 \in \Gamma^\omega$, then it is in some Γ_i, where $i \in \mathcal{N}$. Since ϕ select the formula infinitely often, there is $j > i$ such that $\phi(j) = (0, m, F_1 \wedge F_2, R)$. Then by construction $\{m : F_1, m : F_2\} \subseteq \Gamma_{j+1} \subseteq \Gamma^\omega$.

3. If $m : F_1 \wedge F_2 \in \Delta^\omega$, then it is in some Δ_i, where $i \in \mathcal{N}$. Since ϕ select the formula infinitely often, there is $j > i$ such that $\phi(j) = (1, m, F_1 \wedge F_2, R)$. Then by construction $m : F_n \in \Delta_{j+1} \subseteq \Delta^\omega$, where $n \in \{1, 2\}$ and $\mathcal{G}_j; \Gamma_j \vdash m : F_n; \Delta_j$ does not have a derivation.

4. If $m : F_1 \to F_2 \in \Gamma^\omega$, similar to case 3.

5. If $m : F_1 \to F_2 \in \Delta^\omega$, similar to case 2.

6. If $m : \top^* \in \Gamma^\omega$, then $m : \top^* \in \Gamma_i$, for some $i \in \mathcal{N}$, since each labelled formula from Γ^ω must appear somewhere in the sequence. Then there exists $j > i$ such that $\phi(j) = (0, m, \top^*, R)$ where this formula becomes principal. By construction, $(\epsilon, m \triangleright \epsilon) \in \mathcal{G}_{j+1} \subseteq \mathcal{G}^\omega$. Then $\mathcal{G}^\omega \vdash_E (m = \epsilon)$ because $\mathcal{G}_{j+1} \vdash_E (m = \epsilon)$. So we deduce that $m =_{\mathcal{G}^\omega} \epsilon$.

8. If $m : F_1 * F_2 \in \Gamma^\omega$, then it is in some Γ_i, where $i \in \mathcal{N}$. Then there exists $j > i$ such that $\phi(j) = (0, m, F_1 * F_2, R)$. By construction $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(a_{2j}, a_{2j+1} \triangleright m)\} \subseteq \mathcal{G}^\omega$, and $\Gamma_{j+1} = \Gamma_j \cup \{a_{2j} : F_1, a_{2j+1} : F_2\} \subseteq \Gamma^\omega$.

9. If $m : F_1 * F_2 \in \Delta^\omega$, then it is in some Δ_i, where $i \in \mathcal{N}$. For any $(x, y \triangleright m') \in \mathcal{G}^\omega$ such that $\mathcal{G}^\omega \vdash_E (m = m')$, there exists $j > i$ such that $(x, y \triangleright m') \in \mathcal{G}_j$ and $\mathcal{G}_j \vdash_E (m = m')$. Also, there exists $k > j$ such that $\phi(k) = (1, m, F_1 * F_2, \{(x, y \triangleright m')\})$ where the labelled formula becomes principal. Since $(x, y \triangleright m') \in \mathcal{G}_k$ and $\mathcal{G}_k \vdash (m = m')$, we have either $x : F_1 \in \Delta_{k+1} \subseteq \Delta^\omega$ or $y : F_2 \in \Delta_{k+1} \subseteq \Delta^\omega$.

10. If $m : F_1 -\!\!* F_2 \in \Gamma^\omega$, similar to case 8.

11. If $m : F_1 -\!\!* F_2 \in \Delta^\omega$, similar to case 9.

12. For each $a_n \in \mathcal{L}$, there is a $j \geq n$ such that $\phi(j) = (O, m, \mathbb{U}, \{(a_n, \epsilon \triangleright a_n)\})$ where U is applied to a_n. Then $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(a_n, \epsilon \triangleright a_n)\} \subseteq \mathcal{G}^\omega$, because $n \leq 2j + 1$.

13. If $(x, y \triangleright z) \in \mathcal{G}^\omega$, then it is in some $\mathcal{G}_i$, where $i \in \mathcal{N}$. Since the schedule is fair, there is a $j > i$ such that $\phi(j) = (O, m, \mathbb{E}, \{(x, y \triangleright z)\})$ where E is applied. Then $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(y, x \triangleright z)\} \subseteq \mathcal{G}^\omega$.

14. If $(x, y \triangleright z) \in \mathcal{G}^\omega$, $(u, v \triangleright x') \in \mathcal{G}^\omega$, and $x =_{\mathcal{G}^\omega} x'$, then there is some $\mathcal{G}_i$, $i \in \mathcal{N}$ such that $\{(x, y \triangleright z), (u, v \triangleright x')\} \subseteq \mathcal{G}_i$ and $\mathcal{G}_i \vdash_E (x = x')$. There are two cases to consider, depending on whether $(x, y \triangleright z)$ and $(u, v \triangleright x')$ are the same relational atoms or not. Suppose they are distinct. Then there must be some $j > i$ such that $\phi(j) = (O, m, \mathbb{A}, \{(x, y \triangleright z), (u, v \triangleright x')\})$. Then $\{(x, y \triangleright z), (u, v \triangleright x')\} \in \mathcal{G}_j$ and $\mathcal{G}_j \vdash_E (x = x')$. By construction we obtain that $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(u, a_{2j} \triangleright z), (y, v \triangleright a_{2j})\} \subseteq \mathcal{G}^\omega$. If $(x, y \triangleright z)$ and $(u, v \triangleright x')$ are the same relational atom, then a similar argument can be applied, but in this case the rule instance to choose is one which selects $\mathbb{A}_C$ rather than $\mathbb{A}$. □

Finally we can state the completeness theorem: whenever a formula has no derivation in ILS_{PASL}, there is an infinite counter-model based on the limit sequent and the Kripke relational frame.

Theorem 6.2.8 (Completeness)**.** *Every formula F unprovable in ILS_{PASL} is not valid (in PASL relational Kripke models).*

Proof. We construct a limit sequent $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ for F following Def. 6.2.6. Note that by the construction of the limit sequent, we have $a_1 : F \in \Delta^\omega$. By Lemma 6.2.7, this limit sequent is a Hintikka sequent, and therefore by Lemma 6.2.4, $\mathcal{G}^\omega; \Gamma^\omega \vdash \Delta^\omega$ is falsifiable. This means there exists a model $(\mathcal{F}, \nu, \rho)$ that satisfies $\mathcal{G}^\omega$ and Γ^ω and falsifies every element of Δ^ω, including $a_1 : F$, which means that F is false at world $\rho(a_1)$. Thus F is not valid. □

Corollary 6.2.9. *If formula F is unprovable in LS_{PASL} then F is not valid (in PASL relational Kripke models).*

Proof. By Lemma 6.2.3 and Theorem 6.2.8. □

6.3 $LS_{ST_{DHA}}$: Labelled Rules for DHA Separation Theory

We now consider some extensions of PASL obtained by imposing additional properties on the semantics, as suggested by Dockins et al. [32] and covered in Section 5.2.2. We show that sound rules for *indivisible unit* and the stronger property of *disjointness* can be added to our labelled sequent calculus without jeopardising our completeness proof, but that the more exotic properties of *splittability* and *cross-split* are not fully compatible with our current framework, they require non-trivial changes to the proofs in Section 6.2.

Indivisible unit is captured by the following sound rule:

$$\frac{(\epsilon, y \triangleright \epsilon)[\epsilon/x]; \mathcal{G}[\epsilon/x]; \Gamma[\epsilon/x] \vdash \Delta[\epsilon/x]}{(x, y \triangleright \epsilon); \mathcal{G}; \Gamma \vdash \Delta}\ IU$$

Note that even if y is not x, we can still instantiate the label y to ϵ by applying Eq_1 upwards. Recall that the sequent calculus LS_{BBI} [53] is just the sequent calculus LS_{PASL} minus the rules C and P, and that the formula corresponds to frames with indivisible unit is $\top^* \wedge (A * B) \to A$. The following proposition states that the axiom of indivisible unit is provable in LS_{BBI} extended with IU.

Proposition 6.3.1. *The formula $\top^* \wedge (A * B) \to A$ is provable in $LS_{BBI} + IU$.*

We give a derivation for this proposition below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon, a_2 \rhd \epsilon); \epsilon : A; a_2 : B \vdash \epsilon : A}\,id}{(a_1, a_2 \rhd \epsilon); a_1 : A; a_2 : B \vdash \epsilon : A}\,IU}{(a_1, a_2 \rhd a_0); a_0 : \top^*; a_1 : A; a_2 : B \vdash a_0 : A}\,\top^* L}{; a_0 : \top^*; a_0 : A * B \vdash a_0 : A}\,{*L}}{; a_0 : \top^* \wedge (A * B) \vdash a_0 : A}\,\wedge L}{; \vdash a_0 : (\top^* \wedge (A * B)) \to A}\,\to R$$

Therefore the completeness of $LS_{BBI} + IU$ for the BBI semantics plus indivisible unit is obvious. However, showing that the axiom for indivisible unit can be proved is not sufficient for proving the completeness of $LS_{PASL} + IU$. Again, we would have to prove via a counter-model construction similar to the one presented in Section 6.2.2. Luckily, the rule IU can be easily incorporated into the definition of $\vdash_E$ as IU only involves a global label substitution, so the main proof remains the same. Nice properties such as cut-elimination are also preserved.

Theorem 6.3.2. *$LS_{PASL} + IU$ is sound and cut-free complete with respect to the class of PASL Kripke relational frames (and separation algebras) with indivisible unit.*

Proof. Soundness is straightforward as the rule IU essentially just encodes the semantics into the labelled sequent calculus.

Cut-elimination follows by checking each lemma in Section 6.1.1. Specifically, we show the details of Lemma 6.1.2 (substitution) and Lemma 6.1.4 (invertibility) here.

Substitution: Prove by induction on the height of the derivation. Here we examine the case where IU is the last rule in the derivation. The rule IU is a structural rule, thus it does not have a principal formula and belongs to the case where neither x nor y in the substitution $[y/x]$ is the label of the principal formula. We consider the sub-cases of x, y being a or not respectively, where the IU application is shown below.

$$\dfrac{\begin{array}{c}\Pi\\ (\mathcal{G}; (\epsilon, b \rhd \epsilon); \Gamma \vdash \Delta)[\epsilon/a]\end{array}}{\mathcal{G}; (a, b \rhd \epsilon); \Gamma \vdash \Delta}\,IU$$

1. If $x \neq a$ we consider two sub-cases.
   (a) If $y \neq a$ then the substitutions $[y/x], [\epsilon/a]$ are independent, thus we can easily use the induction hypothesis to substitute $[y/x]$ and switch the order of substitutions to obtain the desired derivation. If $x = b$ and $y = \epsilon$, the IU application is reduced to Eq_1 and E applications as follows, where Π' is obtained by using the induction hypothesis to substitute $[\epsilon/b]$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G};(\epsilon,\epsilon\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/a][\epsilon/b]}}{(\mathcal{G};(\epsilon,\epsilon\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/b][\epsilon/a]}\,Eq_1}{(\mathcal{G};(a,\epsilon\rhd\epsilon);(\epsilon,a\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/b]}\,E}{(\mathcal{G};(a,\epsilon\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/b]}$$

(b) If $y = a$, we use the induction hypothesis to substitute $[\epsilon/x]$, then use IU to obtain the derivation.

$$\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G};(\epsilon,b\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/a][\epsilon/x]}}{(\mathcal{G};(\epsilon,b\rhd\epsilon);\Gamma\vdash\Delta)[a/x][\epsilon/a]}\,IU}{(\mathcal{G};(a,b\rhd\epsilon);\Gamma\vdash\Delta)[a/x]}$$

A special case where $x = b$ can be shown similarly.

2. If $x = a$, again, we consider two sub-cases:

(a) If $y \neq \epsilon$, we use the induction hypothesis to substitute $[\epsilon/y]$, and then use IU to obtain the derivation.

$$\dfrac{\dfrac{\dfrac{\Pi'}{(\mathcal{G};(\epsilon,b\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/a][\epsilon/y]}}{(\mathcal{G};(\epsilon,b\rhd\epsilon);\Gamma\vdash\Delta)[y/a][\epsilon/y]}\,IU}{(\mathcal{G};(y,b\rhd\epsilon);\Gamma\vdash\Delta)[y/a]}$$

(b) If $y = \epsilon$, then the substitution gives $\mathcal{G}[\epsilon/a];(\epsilon,b\rhd\epsilon);\Gamma[\epsilon/a]\vdash\Delta[\epsilon/a]$, which is known to be derivable by using Π.

Invertibility: The rule IU is trivially invertible, as can be proved by using the substitution lemma. We show here (by induction on the height of the derivation) the case for the rule $\top^*L$, where the last rule in the derivation is IU. The other rules can be proved similarly as in [53]. The last rule IU runs as below.

$$\dfrac{\dfrac{\Pi}{(\mathcal{G};(\epsilon,b\rhd\epsilon);\Gamma;x:\top^*\vdash\Delta)[\epsilon/a]}\,IU}{\mathcal{G};(a,b\rhd\epsilon);\Gamma;x:\top^*\vdash\Delta}$$

we consider three sub-cases:

1. If $x \neq a$ and $x \neq b$, then we can safely apply the induction hypothesis on the premise, switch the order of substitutions, and apply IU to obtain $(\mathcal{G};(a,b\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/x]$.
2. If $x = a$, then the premise of the IU application is what we need to derive (with a dummy $\epsilon:\top^*$ on the left hand side of the sequent).
3. If $x = b$, then $(\mathcal{G};(a,\epsilon\rhd\epsilon);\Gamma\vdash\Delta)[\epsilon/b]$ can be derived by applying E and Eq_1 backwards then use the induction hypothesis to obtain the derivation. The details are the same as the derivation shown in 1(a) of the proof for substitution.

Completeness can be proved via the same counter-model construction for LS_{PASL} (Corollary 6.2.9). That is, we first define an intermediate calculus $ILS_{PASL}+IU$ that is equivalent to $LS_{PASL}+IU$, and do counter-model construction in $ILS_{PASL}+IU$. Since the IU rule involves substitution, the rule will be localised into the entailment relation $\vdash_E$, so the definition of $\vdash_E$ in Definition 6.2.1 is modified to include IU in addition to Eq_1, Eq_2, P and C. Thus the rules of $ILS_{PASL}+IU$ are exactly the same as ILS_{PASL}, and the only change is in the definition of $\vdash_E$. The equivalence between $LS_{PASL}+IU$ and $ILS_{PASL}+IU$ can be proved as in Lemma 6.2.3.

Then we only need to show that a Hintikka sequent yields a Kripke relational frame that corresponds to a separation algebra with indivisible unit. In particular, no additional clauses are needed in the definition of Hintikka sequent since it is parametric on the entailment relation $\vdash_E$.

For a Hintikka sequent $\mathcal{G};\Gamma\vdash\Delta$, suppose $(H,\triangleright_{\mathcal{G}},[\epsilon])$ is the PASL Kripke relational frame generated by $\mathcal{G}$. Given any $[a],[b]\triangleright_{\mathcal{G}}[\epsilon]$, we can find a $(a',b'\triangleright c')\in\mathcal{G}$ such that $[a]=[a'],[b]=[b'],[\epsilon]=[c']$. Also, we can use the rule IU to derive $\mathcal{G}\vdash_E(a'=\epsilon)$. Thus by Lemma 6.2.1, we obtain $[a]=[a']=[\epsilon]$. So the structure $(H,\triangleright_{\mathcal{G}},[\epsilon])$ generated by $\mathcal{G}$ is indeed a PASL Kripke relational frame that obeys indivisible unit.

The saturation with logical rules and structural rules E,U,A,A_C is then the same as in Section 6.2. □

Disjointness is captured by the following rule:

$$\frac{(\epsilon,\epsilon\triangleright y)[\epsilon/x];\mathcal{G}[\epsilon/x];\Gamma[\epsilon/x]\vdash\Delta[\epsilon/x]}{(x,x\triangleright y);\mathcal{G};\Gamma\vdash\Delta}\ {}_D$$

Clearly the label y, if not the same as x, can be unified to ϵ by applying Eq_2 on the premise. Since associativity and commutativity for the ternary relation R are assumed, given three worlds w_1,w_2,w_0 that satisfy $R(w_1,w_2,w_0)$, if w_1,w_2 share a common child w_3, then there must be some worlds w,w'' such that $R(w,w'',w_0)$ and $R(w_3,w_3,w)$ hold. Therefore by disjointness we get $w_3=\epsilon$ as well as $w=\epsilon$. If we view each world as a heap, this corresponds to forbidding two heaps that can be combined to have a non-empty intersection (sub-heap).

We can prove the axiom for indivisible unit by using $LS_{BBI}+D$.

Proposition 6.3.3. *The formula $\top^*\wedge(A*B)\to A$ is provable in $LS_{BBI}+D$.*

The following is a derivation for the above proposition. We highlight the principal relational atoms where they are not obvious.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon,\epsilon\triangleright\epsilon);\cdots;\epsilon:A;\epsilon:B\vdash\epsilon:A}\,id}}{(\epsilon,a_1\triangleright\epsilon);\cdots;a_1:A;\epsilon:B\vdash\epsilon:A}\,Eq_1}}{(a_1,w_2\triangleright w_1);(\epsilon,\epsilon\triangleright w_2);\boxed{(a_1,\epsilon\triangleright\epsilon)};\cdots;a_1:A;\epsilon:B\vdash\epsilon:A}\,E}}{(a_1,w_2\triangleright w_1);\boxed{(a_2,a_2\triangleright w_2)};(a_1,a_2\triangleright\epsilon);\cdots;a_1:A;a_2:B\vdash\epsilon:A}\,D}}{(a_1,w_1\triangleright\epsilon);\boxed{(\epsilon,a_2\triangleright w_1);(a_1,a_2\triangleright\epsilon)};\cdots;a_1:A;a_2:B\vdash\epsilon:A}\,A}}{(\epsilon,\epsilon\triangleright\epsilon);(a_1,a_2\triangleright\epsilon);a_1:A;a_2:B\vdash\epsilon:A}\,A}}{(a_1,a_2\triangleright\epsilon);a_1:A;a_2:B\vdash\epsilon:A}\,U}}{(a_1,a_2\triangleright a_0);a_0:\top^*;a_1:A;a_2:B\vdash a_0:A}\,\top^*L}}{;a_0:\top^*;a_0:A*B\vdash a_0:A}\,*L}}{;a_0:\top^*\wedge(A*B)\vdash a_0:A}\,\wedge L}}{;\ \vdash a_0:\top^*\wedge(A*B)\to A}\,\to R$$

Similarly to the rule IU, the rule D for disjointness only involves global label substitution, therefore we can prove the completeness of $LS_{PASL}+D$ by the same routine. But unlike indivisible unit, disjointness is not axiomatisable [22], so the completeness proof for $LS_{BBI}+D$ has to follow the counter-model construction procedure as well.

Theorem 6.3.4. *$LS_{PASL}+D$ is sound and cut-free complete with respect to the class of PASL Kripke relational frames (and separation algebras) with disjointness.*

Proof. Similar to the proof for Theorem 6.3.2. □

Splittability and cross-split are harder to capture, in the sense that the labelled rules for them cannot be directly incorporated in the counter-model construction in Section 6.2. Instead, we need rather non-trivial changes in the proof to accommodate them. Therefore we present these two properties together, as well as how the rules for them are handled in the completeness proof.

We give the following rules for splittability:

$$\dfrac{(x,y\triangleright z);(x\neq\epsilon);(y\neq\epsilon);(z\neq\epsilon);\mathcal{G};\Gamma\vdash\Delta}{(z\neq\epsilon);\mathcal{G};\Gamma\vdash\Delta}\,S \qquad \dfrac{}{(w\neq w);\mathcal{G};\Gamma\vdash\Delta}\,{\neq}L$$

$$\dfrac{(w\neq\epsilon);\mathcal{G};\Gamma\vdash\Delta \qquad (\epsilon,w\triangleright\epsilon);\mathcal{G};\Gamma\vdash\Delta}{\mathcal{G};\Gamma\vdash\Delta}\,EM$$

We add a new type of structure, namely *inequality* of labels, to our calculus. This should not be confused with the equality predicate for expressions or values in separation logic with concrete heap model semantics. We can view $(x\neq y)$ as a negated $(x,\epsilon\triangleright y)$, which is equivalent to allowing $(x,\epsilon\triangleright y)$ in the succedent. The inequality expressions are grouped with relational atoms in $\mathcal{G}$. The rule S directly encodes the semantics of splittability. We then need another rule $\neq L$ to conclude that $(w\neq w)$, for any label w, cannot be valid. Finally, the rule EM, named as the law of excluded middle for $(w=\epsilon)\vee(w\neq\epsilon)$, is essentially a cut on $w:\top^*$. This rule is needed because our *cut* rule does not cut on relational atoms.

The advantage of formulating the above as structural rules, in contrast to the corresponding rules presented in [51], is that these structural rules do not require extra arguments in the cut-elimination proof.

The following sound rule naturally captures cross-split, where $p, q, s, t, u, v, x, y, z$ are labels:

$$\frac{(p, q \rhd x); (p, s \rhd u); (s, t \rhd y); (q, t \rhd v); (x, y \rhd z); (u, v \rhd z); \mathcal{G}; \Gamma \vdash \Delta}{(x, y \rhd z); (u, v \rhd z); \mathcal{G}; \Gamma \vdash \Delta}\ CS$$

The labels p, q, s, t do not occur in the conclusion

However, to ensure contraction admissibility, we need the following special case for this rule where the two principal relational atoms are the same.

$$\frac{(p, q \rhd x); (p, s \rhd x); (s, t \rhd y); (q, t \rhd y); (x, y \rhd z); \mathcal{G}; \Gamma \vdash \Delta}{(x, y \rhd z); \mathcal{G}; \Gamma \vdash \Delta}\ CS_C$$

The labels p, q, s, t do not occur in the conclusion

Soundness for these new rules are easy to check. They obviously preserve falsifiability upwards.

Proposition 6.3.5. *The rules $S, \neq L, EM, CS, CS_C$ are sound.*

The axiom of splittability can be proved in LS_{BBI} with the extension of rules for this property, thus the extended system is complete for BBI plus splittability.

Proposition 6.3.6. *The axiom $\neg\top^* \to (\neg\top^* * \neg\top^*)$ for splittability is provable in $LS_{BBI} + \{S, \neq L, EM\}$.*

Proof. We start from $\neg\top^* \to (\neg\top^* * \neg\top^*)$, and obtain the following derivation backwards:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon, \epsilon \rhd \epsilon);\vdash \epsilon : \top^*; \epsilon : \neg\top^* * \neg\top^*}\ \top^*R}{(\epsilon, w \rhd \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ Eq_1 \qquad \begin{array}{c}\Pi\\ (w \neq \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*\end{array}}{;\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ EM}{;w : \neg\top^* \vdash w : \neg\top^* * \neg\top^*}\ \neg L}{;\vdash w : \neg\top^* \to (\neg\top^* * \neg\top^*)}\ \to R$$

Where Π is the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon, \epsilon \rhd \epsilon); (w \neq \epsilon); (\epsilon, y \rhd w); (\epsilon \neq \epsilon); (y \neq \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ \neq L}{(\epsilon, x \rhd \epsilon); (w \neq \epsilon); (x, y \rhd w); (x \neq \epsilon); (y \neq \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ Eq_1}{(w \neq \epsilon); (x, y \rhd w); (x \neq \epsilon); (y \neq \epsilon); x : \top^* \vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ \top^*L}{(w \neq \epsilon); (x, y \rhd w); (x \neq \epsilon); (y \neq \epsilon);\vdash x : \neg\top^*; w : \top^*; w : \neg\top^* * \neg\top^*}\ \neg R \qquad \Pi'}{(w \neq \epsilon); (x, y \rhd w); (x \neq \epsilon); (y \neq \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ *R}{(w \neq \epsilon);\vdash w : \top^*; w : \neg\top^* * \neg\top^*}\ S$$

and Π' is a symmetric derivation of the left branch of the $*R$ application. □

It is easy to check that the rules $S, \neq L, EM, CS, CS_C$ do not break cut-elimination. Incorporating splittability and cross-split both involve modifications of our previous counter-model construction method. We present these treatments together in the completeness proof for $LS_{PASL} + \{S, \neq L, EM, CS, CS_C\}$.

Theorem 6.3.7. *$LS_{PASL} + \{S, \neq L, EM, CS, CS_C\}$ is sound and cut-free complete with respect to the class of PASL Kripke relational frames (and separation algebras) with splittability and cross-split.*

In the following we give the proof for the above theorem. The definition of the *equivalence entailment* $\vdash_E$ is the same as Definition 6.2.1. We then obtain the intermediate system $ILS_{PASL}2$ as ILS_{PASL} plus S, EM, CS_C, with the following modifications:

$$\frac{\mathcal{G} \vdash_E (w = w')}{(w \neq w'); \mathcal{G}; \Gamma \vdash \Delta} \neq L$$

$$\frac{(p, q \rhd x); (p, s \rhd u); (s, t \rhd y); (q, t \rhd v); (x, y \rhd z); (u, v \rhd z'); \mathcal{G}; \Gamma \vdash \Delta}{(x, y \rhd z); (u, v \rhd z'); \mathcal{G}; \Gamma \vdash \Delta} CS$$
$$(x, y \rhd z); (u, v \rhd z'); \mathcal{G} \vdash_E (z = z')$$
The labels p, q, s, t do not occur in the conclusion

Note that, although technically being a side condition and a premise are the same, we say $\vdash_E$ is a side condition because it does not appear in a derivation tree.

The system $ILS_{PASL}2$ is equivalent to $LS_{PASL} + \{S, \neq L, EM, CS, CS_C\}$, which is straightforward to show. Thus in what follows we give a counter-model construction procedure for $ILS_{PASL}2$, then obtain the completeness result for both systems. As $ILS_{PASL}2$ is just an extension of ILS_{PASL}, we only give the additional definitions and proofs, and the parts where modifications are made.

Definition 6.3.1 (Hintikka sequent)**.** *A labelled sequent $\mathcal{G}; \Gamma \vdash \Delta$ is a* Hintikka sequent *if it satisfies the conditions in Definition 6.2.3 and the following, for any formulae A, B and any labels a, a', b, c, d, e, z, z':*

16. *For any label $m \in \mathcal{L}$, either $(m \neq \epsilon) \in \mathcal{G}$ or $m =_{\mathcal{G}} \epsilon$.*
17. *If $(z \neq \epsilon) \in \mathcal{G}$, then $\exists x, y$, s.t. $(x, y \rhd z) \in \mathcal{G}$, and $(x \neq \epsilon) \in \mathcal{G}$, and $(y \neq \epsilon) \in \mathcal{G}$.*
18. *It is not the case that $(a \neq a') \in \mathcal{G}$ and $a =_{\mathcal{G}} a'$.*
19. *If $(a, b \rhd z) \in \mathcal{G}$ and $(c, d \rhd z') \in \mathcal{G}$ and $z =_{\mathcal{G}} z'$, then $\exists ac, bc, ad, bd$ s.t. $(ad, ac \rhd a) \in \mathcal{G}$, $(ad, bd \rhd d) \in \mathcal{G}$, $(ac, bc \rhd c) \in \mathcal{G}$, and $(bc, bd \rhd b) \in \mathcal{G}$.*

We extend the proof for Lemma 6.2.4 for the additional items in the above definition. That is, we show that every Hintikka sequent is satisfiable by additionally showing that the constructed model $(H, \rhd_{\mathcal{G}}, \epsilon_{\mathcal{G}}, \nu, \rho)$ from the Hintikka sequent satisfies splittability and cross-split.

Proof. The following belongs to the first part of the proof for Lemma 6.2.4.

Splittability: for each $[a] \in H$, there is some $a' \in \mathcal{L}$ s.t. $[a] = [a']$. By condition 16 in Definition 6.3.1, either (1) $(a' \neq \epsilon) \in \mathcal{G}$ or (2) $a' =_{\mathcal{G}} \epsilon$. If (1) holds, by condition 17 in Definition 6.3.1, there exist x, y s.t. $(x, y \triangleright a') \in \mathcal{G}$, $(x \neq \epsilon) \in \mathcal{G}$, and $(y \neq \epsilon) \in \mathcal{G}$ hold. Thus we can find $[x], [y]$ s.t. $[x], [y] \triangleright_{\mathcal{G}} [a]$ holds and $[x] \neq [\epsilon]$, $[y] \neq [\epsilon]$. If (2) holds, splittability trivially holds.

Cross-split: if $[a], [b] \triangleright_{\mathcal{G}} [z]$ and $[c], [d] \triangleright_{\mathcal{G}} [z]$ hold, then we can fine some $(a', b' \triangleright z') \in \mathcal{G}$ and $(c', d' \triangleright z'') \in \mathcal{G}$ s.t. $[a] = [a']$, $[b] = [b']$, $[c] = [c']$, $[d] = [d']$, and $[z] = [z'] = [z'']$. This implies that $z' =_{\mathcal{G}} z''$. Then by condition 19 in Definition 6.3.1, there are ad, ac, bc, bd, s.t. $(ad, ac \triangleright a') \in \mathcal{G}$, $(ad, bd \triangleright d') \in \mathcal{G}$, $(ac, bc \triangleright c') \in \mathcal{G}$, and $(bc, bd \triangleright b') \in \mathcal{G}$. Therefore we can find $[ad]$, $[ac]$, $[bc]$, $[bd]$, s.t. $[ad], [ac] \triangleright_{\mathcal{G}} [a]$, $[ad], [bd] \triangleright_{\mathcal{G}} [d]$, $[ac], [bc] \triangleright_{\mathcal{G}} [c]$, and $[bc], [bd] \triangleright_{\mathcal{G}} [b]$.

For the second part of the proof for Lemma 6.2.4, we need to further show that

4. If $(a \neq b) \in \mathcal{G}$, then $[a] \neq [b]$.

This is a direct consequence of Condition 18 in Definition 6.3.1. □

To deal with the new rules we added, we now define the *extended formulae* as

$$ExF ::= F \mid \mathbb{U} \mid \mathbb{E} \mid \mathbb{A} \mid \mathbb{A}_C \mid \mathbb{S} \mid \mathbb{EM} \mid \mathbb{CS} \mid \mathbb{CS}_C$$

where F is a BBI formula, and the others are constants. We also need to redefine the rule instance to handle the inequality structure. Thus Definition 6.2.4 is modified as below.

Definition 6.3.2 (Schedule ϕ)**.** *A* rule instance *is a tuple (O, m, ExF, R, I), where O is either 0 (left) or 1 (right), m is a label, ExF is an extended formula, R is a set of relational atoms such that $|R| \leq 2$, and I is a singleton inequality. Let $\mathcal{S}$ denote the set of all rule instances. A* schedule *is a function from the set of natural numbers $\mathcal{N}$ to $\mathcal{S}$. A schedule ϕ is* fair *if for every rule instance $S \in \mathcal{S}$, the set $\{i \mid \phi(i) = S\}$ is infinite.*

It follows from the same reason that there exists a fair schedule. The new component I in the schedule is ignored in all the cases in Definition 6.2.5. However, we have to slightly extend the definition to accommodate splittability and cross-split. The original definition is rewritten as follows.

Definition 6.3.3. *Let F be a formula which is not provable in $ILS_{PASL}2$. We construct a series of finite sequents $\{\mathcal{G}_i; \Gamma_i \vdash \Delta_i\}_{i \in \mathcal{N}}$ from F where $\mathcal{G}_1 = \Gamma_1 = \emptyset$ and $\Delta_1 = a_1 : F$.*

Assuming that $\mathcal{G}_i; \Gamma_i \vdash \Delta_i$ has been defined, we define $\mathcal{G}_{i+1}; \Gamma_{i+1} \vdash \Delta_{i+1}$ as follows. Suppose $\phi(i) = (O_i, m_i, ExF_i, R_i, I_i)$.

- *If $O_i = 0$, ExF_i is a PASL formula C_i and $m_i : C_i \in \Gamma_i$:*
  - *If $C_i = F_1 \wedge F_2$, same as original def..*

  - *If* $C_i = F_1 \to F_2$, *same as original def..*
  - *If* $C_i = \top^*$, *same as original def..*
  - *If* $C_i = F_1 * F_2$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{4i}, a_{4i+1} \rhd m_i)\}$, $\Gamma_{i+1} = \Gamma_i \cup \{a_{4i} : F_1, a_{4i+1} : F_2\}$, $\Delta_{i+1} = \Delta_i$.
  - *If* $C_i = F_1 -\!\!* F_2$ *and* $R_i = \{(x, m \rhd y)\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (m = m_i)$, *same as original def..*

- *If* $O_i = 1$, ExF_i *is a PASL formula* C_i, *and* $m_i : C_i \in \Delta$:
  - *If* $C_i = F_1 \wedge F_2$, *same as original def..*
  - *If* $C_i = F_1 \to F_2$, *same as original def..*
  - $C_i = F_1 * F_2$ *and* $R_i = \{(x, y \rhd m)\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (m_i = m)$, *same as original def..*
  - *If* $C_i = F_1 -\!\!* F_2$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{4i}, m_i \rhd a_{4i+1})\}$, $\Gamma_{i+1} = \Gamma_i \cup \{a_{4i} : F_1\}$, *and* $\Delta_{i+1} = \Delta_i \cup \{a_{4i+1} : F_2\}$.

- *If* $ExF_i \in \{\mathbb{U}, \mathbb{E}, \mathbb{A}, \mathbb{A}_C, \mathbb{S}, \mathbb{EM}, \mathbb{CS}, \mathbb{CS}_C\}$, *we proceed as follows:*
  - *If* $ExF_i = \mathbb{U}$, $R_i = \{(a_n, \epsilon \rhd a_n)\}$, *where* $n \leq 4i + 3$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_n, \epsilon \rhd a_n)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{E}$, *same as original def..*
  - *If* $ExF_i = \mathbb{A}$, $R_i = \{(x, y \rhd z); (u, v \rhd x')\} \subseteq \mathcal{G}_i$ *and* $\mathcal{G}_i \vdash_E (x = x')$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(u, a_{4i} \rhd z), (y, v \rhd a_{4i})\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{A}_C$, $R_i = \{(x, y \rhd x')\} \subseteq \mathcal{G}_i$, *and* $\mathcal{G}_i \vdash_E (x = x')$ *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(x, a_{4i} \rhd x), (y, y \rhd a_{4i})\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{S}$ *and* $I_i = \{(w \neq \epsilon)\} \subseteq \mathcal{G}_i$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{4i}, a_{4i+1} \rhd w), (a_{4i} \neq \epsilon), (a_{4i+1} \neq \epsilon)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{EM}$ *and* $R_i = \{(\epsilon, a_n \rhd \epsilon)\}$, *where* $n \leq 4i + 3$. *If there is no derivation for* $(\epsilon, a_n \rhd \epsilon); \mathcal{G}_i; \Gamma_i \vdash \Delta_i$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(\epsilon, a_n \rhd \epsilon)\}$, *otherwise* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_n \neq \epsilon)\}$. *In both cases,* $\Gamma_{i+1} = \mathcal{G}_i$ *and* $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{CS}$, $R_i = \{(x, y \rhd z), (u, v \rhd z')\} \subseteq \mathcal{G}_i$, *and* $\mathcal{G}_i \vdash_E (z = z')$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{4i}, a_{4i+1} \rhd x), (a_{4i}, a_{4i+2} \rhd u), (a_{4i+2}, a_{4i+3} \rhd y), (a_{4i+1}, a_{4i+3} \rhd v)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.
  - *If* $ExF_i = \mathbb{CS}_C$ *and* $R_i = \{(x, y \rhd z)\} \subseteq \mathcal{G}_i$, *then* $\mathcal{G}_{i+1} = \mathcal{G}_i \cup \{(a_{4i}, a_{4i+1} \rhd x), (a_{4i}, a_{4i+2} \rhd x), (a_{4i+2}, a_{4i+3} \rhd y), (a_{4i+1}, a_{4i+3} \rhd y)\}$, $\Gamma_{i+1} = \Gamma_i$, $\Delta_{i+1} = \Delta_i$.

- *In all other cases,* $\mathcal{G}_{i+1} = \mathcal{G}_i$, $\Gamma_{i+1} = \Gamma_i$ *and* $\Delta_{i+1} = \Delta_i$.

Lemma 6.2.6 is easy to show for the new definitions. The second item in the lemma should now be stated as $\mathcal{L} \subseteq \{a_0, a_1, \cdots, a_{4i-1}\}$. We are ready to prove Lemma 6.2.7 for the new conditions in the Hintikka sequent.

Proof. Condition 18 holds because the limit sequent is finitely-consistent. We show the cases for conditions 16, 17, 19 as follows.

16. For each $a_n \in \mathcal{L}$, there is some natural number $j \geq n$ s.t. $\phi(j) = (O, m, \mathbb{EM}, \{(\epsilon, a_n \rhd \epsilon)\}, I)$, where EM is applied to a_n. Then either (1) $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(\epsilon, a_n \rhd \epsilon)\}$ or (2) $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(a_n \neq \epsilon)\}$, depending on which choice gives a finitely-consistent sequent $\mathcal{G}_{j+1}; \Gamma_{j+1} \vdash \Delta_{j+1}$. If (1) holds, then $(\epsilon, a_n \rhd \epsilon) \in \mathcal{G}_{j+1} \subseteq \mathcal{G}^\omega$, and $\mathcal{G}^\omega \vdash_E (a_n = \epsilon)$ by an Eq_1 application, giving $a_n =_{\mathcal{G}^\omega} \epsilon$. If (2) holds, then $(a_n \neq \epsilon) \in \mathcal{G}_{j+1} \subseteq \mathcal{G}^\omega$.

17. If $(z \neq \epsilon) \in \mathcal{G}^\omega$, then $(z \neq \epsilon) \in \mathcal{G}_i$, for some $i \in \mathcal{N}$. Then there exists $j > i$ s.t. $\phi(j) = (O, m, \mathbb{S}, R, \{(z \neq \epsilon)\})$. Then $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(a_{4j}, a_{4j+1} \rhd z), (a_{4j} \neq \epsilon), (a_{4j+1} \neq \epsilon)\} \subseteq \mathcal{G}^\omega$.

19. If $(x, y \rhd z) \in \mathcal{G}^\omega$ and $(u, v \rhd z') \in \mathcal{G}^\omega$ and $z =_{\mathcal{G}^\omega} z'$. There must be some $i \in \mathcal{N}$ s.t. $\{(x, y \rhd z), (u, v \rhd z')\} \subseteq \mathcal{G}_i$ and $\mathcal{G}_i \vdash_E (z = z')$. Suppose $(x, y \rhd z)$ and $(u, v \rhd z')$ are distinct, then there exists $j > i$ s.t. $\phi(j) = (O, m, \mathbb{CS}, \{(x, y \rhd z), (u, v \rhd z')\}, I)$, and $\{(x, y \rhd z), (u, v \rhd z')\} \subseteq \mathcal{G}_j$, $\mathcal{G}_j \vdash_E (z = z')$ hold. By construction, $\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{(a_{4j}, a_{4j+1} \rhd x), (a_{4j}, a_{4j+2} \rhd u), (a_{4j+2}, a_{4j+3} \rhd y), (a_{4j+1}, a_{4j+3} \rhd v)\} \subseteq \mathcal{G}^\omega$. If $(x, y \rhd z)$ and $(u, v \rhd z')$ are the same, a similar argument can be applied for $\mathbb{CS}_C$.

The other cases are similar to the original proof. Note that the subscript of new labels needs to be adjusted accordingly. □

Therefore the limit sequent in the new definition is indeed an Hintikka sequent, and we can extract an infinite counter-model from it. The completeness result stated by Theorem 6.3.7 follows.

Following the previous work [32, 22], we refer to the set of rules $C, P, IU, D, S, \neq L, EM, CS, CS_C$ as $LS_{ST_{DHA}}$ for labelled sequent rules for separation theories. By a proof that appropriately incorporates the treatments for these properties in this section, we obtain the following result:

Theorem 6.3.8. *$LS_{BBI} + LS_{ST_{DHA}}$ is sound and cut-free complete with respect to the class of BBI Kripke relational frames with DHA separation theory.*

Subsystems of $LS_{BBI} + LS_{ST_{DHA}}$ are also easy to obtain. In our current setting, checking the completeness for additional properties is a long and tedious process. It is possible to come up with a more general proof theory for these logics so that the completeness of new properties would be immediate. This will be a future work. We discuss some examples of subsystems in the next section.

6.4 Calculi for BBI with Subsets of DHA Separation Theory

We now consider various labelled calculi obtained by extending LS_{BBI} with one or more structural rules that correspond to the properties in DHA separation theory.

Most of the results in this section either directly follow from the proofs in previous sections, or are easy adaptations. As those conditions for monoids are often given in a modular way, e.g., in [32, 22], it is not surprising that our structural rules can also be added modularly to LS_{BBI}, since they just simulate those conditions directly and individually in the labelled sequent calculus.

Calculi without Cancellativity There exist resource semantics which would be separation algebras were it not for the failure of cancellativity; we have given two examples in Example 5.2.8. This has led to the suggestion, first in [45], that cancellativity should be omitted from the definition of separation algebra. In any case, it is clear that dropping the rule C in LS_{PASL} gives an interesting system. The proofs in Sec. 6.2 still work if we just insist on a partial commutative monoid, and drop C in $\vdash_E$.

Theorem 6.4.1. *The labelled sequent calculus $LS_{BBI} + P$ is sound and cut-free complete with respect to the partial commutative monoidal semantics for BBI.*

As a result, it is easy to obtain the following sound and complete labelled calculi for the corresponding semantics: $LS_{BBI} + P + IU$ and $LS_{BBI} + P + D$. The proofs are similar to that for Theorem 6.3.2. Larchey-Wendling and Galmiche's work showed that separation algebras with partial-determinism and those additionally with cancellativity have the same set of valid formulae. Their result implies that $LS_{BBI} + P + IU$ and $LS_{BBI} + P + D$ are respectively equivalent to $LS_{BBI} + P + C + IU$ and $LS_{BBI} + P + C + D$. However, in separation logic with concrete memory model semantics, assuming partial-determinism, there might be formulae that can distinguish cancellative models and non-cancellative models. Thus we list these calculi with cancellativity as standalone systems.

Calculi without Partial-determinism and Cancellativity The labelled calculus $LS_{BBI} + IU$ is sound and complete by Prop. 6.3.1, and cut-elimination holds.

Theorem 6.4.2. *The labelled sequent calculus $LS_{BBI} + IU$ is sound and cut-free complete with respect to the commutative monoidal semantics for BBI with indivisible unit.*

To prove the completeness of the calculus $LS_{BBI} + D$, we need to go through the counter-model construction proof, since disjointness is not axiomatisable. It is easy to check that the proofs in Section 6.2 do not break when we define $\vdash_E$ by using Eq_1, Eq_2, D only, and the Hintikka sequent then gives the BBI Kripke relational frame that obeys disjointness. The other proofs remain the same.

Theorem 6.4.3. *The labelled sequent calculus $LS_{BBI} + D$ is sound and cut-free complete with respect to the commutative monoidal semantics for BBI with disjointness.*

To summarise, the proofs given in this chapter offer a sound and cut-free complete calculus for the extension of BBI with every combination of the properties P, C, IU, D,

except that we assume a system with cancellativity also has partial-determinism. The case where none of the properties hold, i.e. regular BBI, have already been solved in Section 3.4 (by Park et al.) and Chapter 4. Omitting the cases covered by the implication of IU by D, this provides us with the following seven labelled calculi:

$$\begin{array}{ll} LS_{BBI} + IU & LS_{BBI} + D \\ LS_{BBI} + P = LS_{PASL} (= LS_{BBI} + P + C) & \\ LS_{BBI} + P + IU & LS_{PASL} + IU \\ LS_{BBI} + P + D & LS_{PASL} + D \end{array}$$

Additionally, sound and cut-free complete subsystems containing rules for splittability or cross-split can be obtained by incorporating the treatments in the proof for Theorem. 6.3.7. The proofs for these subsystems may involve changes in the definition of Hintikka sequent and construction of the limit sequent etc., but the general idea remains the same. We give the following theorem without showing the full proof:

Theorem 6.4.4. *For each subset of DHA separation theory coupled with PASL, there is a subsystem of $LS_{PASL} + LS_{ST_{DHA}}$ that is sound and cut-free complete for it.*

6.5 Experiments

The logic PASL with disjointness and cross-split seems to be a natural candidate for capturing the heap model of separation logic. However, we could not find any PASL formula that requires cross-split to prove. On the other hand, we found many separation logic formulae that require cross-split to prove. For the purpose of the experiments in this section, we thus target PASL with disjointness (denoted as PASL_D) instead. We discuss here an implementation of the proof system $LS_{PASL} + D$ for PASL_D. It turns out that the A_C rule is admissible in this system; in fact it is admissible in the subsystem LS_{PASL}, as shown next. So we do not implement the A_C rule.

Proposition 6.5.1. *The A_C rule is admissible in LS_{PASL}.*

Proof. We show that every derivation in LS_{PASL} can be transformed into one with no applications of A_C. It is sufficient to show that we can eliminate a single application of A_C; then we can eliminate all A_C in a derivation successively starting from the topmost A_C application. So suppose we have a derivation in LS_{PASL} of the form:

$$\dfrac{\begin{array}{c}\Pi \\ (x, w \triangleright x); (y, y \triangleright w); (x, y \triangleright x); \mathcal{G}; \Gamma \vdash \Delta\end{array}}{(x, y \triangleright x); \mathcal{G}; \Gamma \vdash \Delta}\, A_C$$

where w is a new label not in the root sequent. This is transformed into the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi'}{(x, \epsilon \rhd x); (\epsilon, \epsilon \rhd \epsilon); \mathcal{G}[\epsilon/y]; \Gamma[\epsilon/y] \vdash \Delta[\epsilon/y]}\, u}{(x, \epsilon \rhd x); \mathcal{G}[\epsilon/y]; \Gamma[\epsilon/y] \vdash \Delta[\epsilon/y]}\, c}{(x, \epsilon \rhd x); (x, y \rhd x); \mathcal{G}; \Gamma \vdash \Delta}\, u$$

$$(x, y \rhd x); \mathcal{G}; \Gamma \vdash \Delta$$

where Π' is obtained by applying the substitutions $[\epsilon/y]$ and $[\epsilon/w]$ to Π (by using Lemma 6.1.2). Note that since w does not occur in the root sequent, $\mathcal{G}[\epsilon/y][\epsilon/w] = \mathcal{G}[\epsilon/y]$, $\Gamma[\epsilon/y][\epsilon/w] = \Gamma[\epsilon/y]$ and $\Delta[\epsilon/y][\epsilon/w] = \Delta[\epsilon/y]$. These substitutions do not introduce new instances of A_C. □

On the other hand, we restrict the rule U to create the identity relational atom $(w, \epsilon \rhd w)$ only if w occurs in the conclusion (denoted as U' below). This does not reduce the power of LS_{PASL}, as will be shown next.

Lemma 6.5.2. *If $\mathcal{G}; \Gamma \vdash \Delta$ is derivable in LS_{PASL}, then it is derivable in $LS_{PASL} - U + U'$.*

Proof. The original U rule can be separated into two cases:

- U', with the restriction as described above and
- U'', where the created relational atom $(x, \epsilon \rhd x)$ satisfies that x does not occur in the conclusion.

We show that the rule U'' is admissible, therefore using the rule U' alone does not affect provability.

This is proved by a simple induction on the height n of the derivation for $\mathcal{G}; \Gamma \vdash \Delta$. Suppose U'' is the last rule in the derivation, with a premise:

$$(x, \epsilon \rhd x); \mathcal{G}; \Gamma \vdash \Delta$$

where x is a fresh label. Assume that the conclusion is not an empty sequent, hence there must be some label w that occurs in conclusion. By Lemma 6.1.2, replacing x by w, we obtain that

$$(w, \epsilon \rhd w); \mathcal{G}; \Gamma \vdash \Delta$$

is derivable in $n - 1$ steps. By the induction hypothesis, there is a U''-free derivation of this sequent, which leads to the derivation of $\mathcal{G}; \Gamma \vdash \Delta$ by applying the restricted rule U'. □

Our implementation applies the first applicable item in the following list:

1. Try to close the branch by rules id, $\bot L$, $\top^* R$, $\top^* R$.
2. Apply all possible Eq_1, Eq_2, P, C, IU, D rule applications to unify labels [2].
3. Apply invertible rules $\wedge L$, $\wedge R$, $\to L$, $\to R$, $* L$, $-\!\!* R$, $\top^* L$ in all possible ways.

[2] Although IU is admissible, we keep it because it simplifies proof search.

4. Try $*R$ or $-\!\!* L$ by choosing from an existing relational atom that has not been used in the same rule application.
5. Apply structural rules on the set $\mathcal{G}_0$ of relational atoms in the sequent as follows.
   (a) Use E to generate all commutative variants of existing relational atoms in $\mathcal{G}_0$, giving a set $\mathcal{G}_1$.
   (b) Apply A for each applicable pair in $\mathcal{G}_1$, generating a set $\mathcal{G}_2$.
   (c) Use U' to generate all identity relational atoms for each label in $\mathcal{G}_2$, giving the set $\mathcal{G}_3$.
6. If none of above is applicable, fail.

Step (2) is terminating, because each substitution eliminates a label, and we only have finitely many labels. Step (5) is not applicable when $\mathcal{G}_3 = \mathcal{G}_0$, it is also clear that step (5) is terminating. Note that we forbid applications of the rule A to the pair $\{(x, y \rhd z), (u, v \rhd x)\}$ of relational atoms when $\{(u, w \rhd z), (y, v \rhd w)\}$, for some label w, (or any commutative variants of this pair, e.g., $\{(w, u \rhd z); (v, y \rhd w)\}$) is already in the sequent. This is because the created relational atoms in such an A application can be unified to existing ones by using rules P, C.

In the implementation we view Γ, Δ in a sequent $\mathcal{G}; \Gamma \vdash \Delta$ as lists, and each time a logical rule is applied backwards, we place the created new labelled formulae in the front of the list. Thus our proof search has a "focusing flavour" that always tries to decompose the subformulae of a principal formula if possible. To guarantee completeness, each time we apply a $*R$ or $-\!\!* L$ rule, the principal formula is moved to the end of the list, so that each principal formula for non-deterministic rules $*R, -\!\!* L$ is considered fairly, i.e., applied in turn.

We incorporate a number of optimisations in the proof search.

- Back jumping [3] is used to collect the "unsatisfiable core" along each branch. When one premise of a binary rule has a derivation, we try to derive the other premise only when the unsatisfiable core is not included in that premise.

- A search strategy discussed by Park et al [77] is also adopted. For $*R$ and $-\!\!* L$ applications, we forbid the search to consider applying the rule twice with the same pair of principal formula and principal relational atom, since the effect is the same as contraction, which is admissible.

- Previous work on theorem proving for BBI has shown that associativity of $*$ is a source of inefficiency in proof search [77, 53]. We utilise our idea of the heuristic method presented in Section 4.5 to quickly solve certain associativity instances. When we detect $z : A * B$ on the right hand side of a sequent, we try to search for possible worlds (labels) for the subformulae of A, B in the sequent, and construct a binary tree using these labels. For example, if we can find $x : A$ and $y : B$ in the sequent, we will take x, y as the children of z. When we can build such a binary

	Formula	BBeye (opt)	$FVLS_{BBI}$ (heuristic)	Separata (PASL)
(1)	$(a \mathrel{-\!\!*} b) \wedge (\top * (\top^* \wedge a)) \to b$	0.076	0.002	0.002
(2)	$(\top^* \mathrel{-\!\!*} \neg(\neg a * \top^*)) \to a$	0.080	0.004	0.002
(3)	$\neg((a \mathrel{-\!\!*} \neg(a * b)) \wedge ((\neg a \mathrel{-\!\!*} \neg b) \wedge b))$	0.064	0.003	0.002
(4)	$\top^* \to ((a \mathrel{-\!\!*} (b \mathrel{-\!\!*} c)) \mathrel{-\!\!*} ((a * b) \mathrel{-\!\!*} c))$	0.060	0.003	0.002
(5)	$\top^* \to ((a * (b * c)) \mathrel{-\!\!*} ((a * b) * c))$	0.071	0.002	0.004
(6)	$\top^* \to ((a * ((b \mathrel{-\!\!*} e) * c)) \mathrel{-\!\!*} ((a * (b \mathrel{-\!\!*} e)) * c))$	0.107	0.004	0.008
(7)	$\neg((a \mathrel{-\!\!*} \neg(\neg(d \mathrel{-\!\!*} \neg(a * (c * b))) * a)) \wedge c * (d \wedge (a * b)))$	0.058	0.002	0.006
(8)	$\neg((c * (d * e)) \wedge B)$ where $B := ((a \mathrel{-\!\!*} \neg(\neg(b \mathrel{-\!\!*} \neg(d * (e * c))) * a)) * (b \wedge (a * \top)))$	0.047	0.002	0.013
(9)	$\neg(C * (d \wedge (a * (b * e))))$ where $C := ((a \mathrel{-\!\!*} \neg(\neg(d \mathrel{-\!\!*} \neg((c * e) * (b * a))) * a)) \wedge c)$	94.230	0.003	0.053
(10)	$(a * (b * (c * d))) \to (d * (c * (b * a)))$	0.030	0.004	0.002
(11)	$(a * (b * (c * d))) \to (d * (b * (c * a)))$	0.173	0.002	0.002
(12)	$(a * (b * (c * (d * e)))) \to (e * (d * (a * (b * c))))$	1.810	0.003	0.002
(13)	$(a * (b * (c * (d * e)))) \to (e * (b * (a * (c * d))))$	144.802	0.003	0.002
(14)	$\top^* \to (a * ((b \mathrel{-\!\!*} e) * (c * d)) \mathrel{-\!\!*} ((a * d) * (c * (b \mathrel{-\!\!*} e))))$	6.445	0.003	0.044
(15)	$\neg(\top^* \wedge (a \wedge (b * \neg(c \mathrel{-\!\!*} (\top^* \to a)))))$	T/O	0.003	0.003
(16)	$((D \to (E \mathrel{-\!\!*} (D * E))) \to (b \mathrel{-\!\!*} ((D \to (E \mathrel{-\!\!*} ((D * a) * a))) * b)))$, where $D := \top^* \to a$ and $E := a * a$	0.039	0.005	8.772
(17)	$((\top^* \to (a \mathrel{-\!\!*} (((a * (a \mathrel{-\!\!*} b)) * \neg b) \mathrel{-\!\!*} (a * (a * ((a \mathrel{-\!\!*} b) * \neg b)))))) \to ((((\top^* * a) * (a * ((a \mathrel{-\!\!*} b) * \neg b))) \to (((a * a) * (a \mathrel{-\!\!*} b)) * \neg b)) * \top^*))$	T/O	fail	49.584
(18)	$(F * F) \to F$, where $F := \neg(\top \mathrel{-\!\!*} \neg\top^*)$	invalid	invalid	0.004
(19)	$(\top^* \wedge (a * b)) \to a$	invalid	invalid	0.003

Table 6.1: Experiment 1 results.

tree of labels, the corresponding relational atoms given by the binary tree will be used (if they are in the sequent) as the prioritised ones when decomposing $z : A * B$ and its subformulae. Without a free-variable system, our handling of this heuristic method is just a special case of the original one, but this approach can speed up the search in certain cases.

The experiments in this section are conducted on a Dell Optiplex 790 desktop with Intel CORE i7 2600 @ 3.4 GHz CPU and 8GB memory, running Ubuntu 13.04. The theorem provers are written in OCaml.

Experiment on Formulae in the Literature We tested our prover Separata for the logic $LS_{PASL} + D$ on the formulae listed in Table 6.1; the times displayed are in seconds. We compare the results with provers for BBI, BBeye [77] and the incomplete heuristic-based $FVLS_{BBI}$ [53], when the formula is valid in BBI. We run BBeye in an iterative deepening way, and the time counted for BBeye is the total time it spends. The timeout for each prover is 1000 seconds, we write "T/O" in the table if a prover times out on a formula. The prover based on $FVLS_{BBI}$ and the heuristic constraint solving

method is not complete, we use "fail" to indicate that the prover terminates but fails to find a proof. We write "invalid" for those formulae that are not valid in BBI, we do not test them on BBI provers. Formulae (1-14) are used by Park et al. to test their prover BBeye for BBI [77]. We can see that for formulae (1-14) the performance of Separata is comparable with the heuristic based prover for $FVLS_{BBI}$. Both provers are generally faster than BBeye. Formula (15) is one that BBeye had trouble with [53], but Separata handles it trivially. However, there are cases where BBeye is faster than Separata, for example, formula (16) found from a set of testings on randomly generated BBI theorems. Formula (17) is a converse example where a randomly generated BBI theorem causes BBeye to time out and $FVLS_{BBI}$ with heuristics to terminate within the timeout but without finding a proof due to its incompleteness. Formula (18) is valid only when the monoid is partial [62], and formula (19) is the axiom of indivisible unit. Both Formula (18) and (19) cannot be proved by BBeye and the heuristic-based $FVLS_{BBI}$ prover within the timeout.

Experiment on Randomly Generated Formulae In this experiment we use randomly generated BBI theorems. Of course, these randomly generated theorems are not truly random. There could be some proof search mechanism to take advantage our random generation method. We generate theorems that are "random" to some extend. As mentioned before, there does not exist a Hilbert system for PASL because the logic is not axiomatisable. BBI theorems are a subset of theorems in PASL_D, thus our prover based on $LS_{PASL} + D$ should be able to prove BBI theorems. This test may not show the full power of our prover Separata, but we can see how Separata scales compared to the BBI prover BBeye [77]. Recall the BBI Hilbert system from Section 2.2 in Figure 6.3. We will use these axioms and deduction rules to generate BBI theorems in this test.

Axioms:

$A \to (B \to A)$ $\quad A \to A \vee B$ $\quad B \to A \vee B$

$A \wedge B \to A$ $\quad A \wedge B \to B$ $\quad A \to (B \to (A \wedge B))$

$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$ $\quad \bot \to A$

$(A \to C) \to ((B \to C) \to (A \vee B \to C))$ $\quad \neg\neg A \to A$

$A \to (\top^* * A)$ $\quad (\top^* * A) \to A$

$(A * B) \to (B * A)$ $\quad (A * (B * C)) \to ((A * B) * C)$

Deduction Rules:

$$\frac{\vdash A \qquad \vdash A \to B}{\vdash B}\, MP \qquad \frac{\vdash A \to C \qquad \vdash B \to D}{\vdash (A * B) \to (C * D)}\, *$$

$$\frac{\vdash A \to (B \mathrel{-\!\!*} C)}{\vdash (A * B) \to C} \mathrel{-\!\!*} 1 \qquad \frac{\vdash (A * B) \to C}{\vdash A \to (B \mathrel{-\!\!*} C)} \mathrel{-\!\!*} 2$$

Figure 6.3: The Hilbert system for BBI.

A common way to generate random theorems is to simply globally replace a sub-

Test	n	i	BBeye proved	BBeye avg. time	Separata proved	Separata avg. time
1	10	20	74%	0.27s	78%	0.19s
2	15	30	72%	4.63s	66%	2.40s
3	20	30	61%	8.88s	56%	8.76s
4	20	50	55%	12.39s	53%	0.88s
5	30	50	49%	11.81s	50%	6.26s
6	50	50	31%	11.82s	31%	3.79s

Table 6.2: Experiment 2 results.

formula in a theorem by a longer random formula. We avoid this method on purpose, since we can easily "cheat" by forcing our prover not to expand those sub-formulae until necessary. For example, we can globally replace A by $p \wedge q$, and replace B by $r \vee t$ in the BBI theorem $(A * B) \to (B * A)$, obtaining a longer theorem $((p \wedge q) * (r \vee t)) \to ((r \vee t) * (p \wedge q))$. But then we can build in a mechanism in our prover that given a formula F, it searches for any sub-formula F' that occurs multiple times in F, and replaces F' by a "pseudo-proposition", which is only allowed to be decomposed when the prover cannot find a derivation. In this way, no matter how large the generated theorem is, the prover uses constant time to prove it. The only difference would be the time used to search for sub-formulae.

We create random BBI theorems by first generating some random formulae (not necessarily theorems) of length n, and perform global substitution of these formulae to certain places in a BBI axiom schema; then we use the deduction rules $-\!\!*\,1$ and $-\!\!*\,2$ in Figure 2.3 to mutate the resultant formula in random places, repeat this step by i iterations, resulting in the final theorem.

When generating random theorems, our procedure randomly chooses axioms (and deduction rules), but is biased to use the axioms in Figure 2.3 more often rather than the classical axioms. Given the parameter n (resp. i) as in the previous paragraph, our procedure chooses a random number ranging from 1 to n (resp. i) and proceeds as above. The mutation step is vital to generate theorems with $-\!\!*$, since BBI axioms do not involve $-\!\!*$ at all. Moreover, the "cheat" mechanism would more often fail when we use the deduction rules to mutate the formula, because the internal structure of the formula is changed. Our random theorem generation does not create theorems of a fixed length, but the length grows as n increases.

We compare the performance of Separata with Park et al.'s BBI prover BBeye against our randomly generated theorem suites in Table 6.2. As said previously, the parameter n is the maximum length of random formulae to be substituted into a BBI axiom, i is the maximum iteration of mutation. Each test suite contains 100 BBI theorems, the "proved" column for each prover gives the percentage of successfully proved formulae within the time out, and the "avg. time" column is the average time used when a formula is proved. We set the time out as 200 seconds. If the prover cannot

prove a formula within 200 seconds, the time is not counted in the average time. To give a rough idea of how long the tested formulae are, the formulae in test suite 1 have 20 to 30 binary connectives on average. These numbers are about 100 to 150 for the formulae in test suite 6. The two provers have similar successful rates in these tests, average time used on successful attempts increase for BBeye, but fluctuates for Separata. In fact, both provers spent less than 1 second on most successful attempts (even for test suite 6), but there are some "difficult" formulae that took them over 100 seconds to prove. The reason "avg. time" for Separata is slightly shorter is that there are fewer formulae that are "difficult" for it. But in test suites 2,3,4, Separata proved fewer formulae in time. It seems that the theorems we generate are either too easy or too difficult to prove, there are very few medium difficulty ones. As another fact, although the success rates for both provers are similar, the proved formulae are sometimes different. The two provers may be good at proving different theorems, a phenomenon that was also shown in previous work for BBI [53]. Lastly, we emphasise that Separata and BBeye are designed for different logics, thus comparing their performance may not be fair, we do so only because there are no other provers for PASL to compete with. For example, indivisible unit and disjointness may make Separata prove some formulae much faster, because they unify a lot of labels to ϵ, and significantly simplify the set of ternary relational atoms.

6.6 Discussion and Related Work

Although several propositional abstract separation logics and their relationships with BBI had been discussed in the literature, two important aspects were missing: there was no uniform proof method; and there were no automated reasoning tools for these abstract separation logics. This chapter solves both of these open problems by developing a modular proof method for various propositional abstract separation logics using labelled sequent calculi [73], and showing that the resulting sequent calculus can be used for effective backward proof search, thereby giving a modular semi-decision procedure for all of these logics using labelled sequents. Our implementation is the first automated theorem prover that can handle many of these propositional abstract separation logics.

Sequent calculi are amenable to backward proof-search only if the cut rule is redundant. We have shown in Section 4.2 an explicit cut-elimination theorem for LS_{BBI} to show that any sequent that is derivable with cut is also derivable without using cut. We then obtain the completeness of LS_{BBI} by mimicking derivations in the Hilbert axiomatisation of BBI, but this avenue is no longer viable for LS_{PASL} because partial-determinism and cancellativity are not axiomatisable in the language of PASL (which is precisely the language of BBI), as proved by Brotherston and Villard [22]; that paper goes on to give a sound and complete Hilbert axiomatisation of these properties by extending BBI with techniques from hybrid logic. As a consequence, there cannot be

a Hilbert system in the language of PASL that is sound and complete for partially-deterministic and cancellative separation algebras.

Instead, we proved cut-free completeness of our labelled sequent calculi via a *counter-model construction* procedure which shows that if a formula is not cut-free derivable in our sequent calculus then it is falsifiable in some PASL-model. The redundancy of the cut rule then follows. A novelty of our counter-model construction is that it can be modularly extended to handle extensions and sublogics of PASL. Labelled sequent calculi, with their explicitly semantics-based rules, provide good support for this modularity, as rules for the various properties can be added and removed as required. We showed how the abstract properties suggested for separation algebras by various authors [24, 32, 45] can be combined or omitted without sacrificing our cut-free completeness result.

We have also implemented proof search using our calculus. As stated previously, no decision procedure for PASL is possible [16], so our prover realises a semi-decision procedure. Experimental results show that our prover is usually faster than other provers for BBI when testing against the same benchmarks of BBI formulae, partly because the extra properties (indivisible unit, disjointness, etc.) we consider sometimes simplify the proof search.

To our knowledge this is the first proof to be presented for the cut-free completeness of a modular proof system for PASL and some variants with the properties mentioned previously. Our implementation is the first automated theorem prover for PASL and its neighbours.

While we work with abstract models of separation logics, the reasoning principles behind our proof-theoretic methods should be applicable to concrete models also, so in the next chapter we investigate how concrete predicates such as $\mapsto$ for the heap model might be integrated into our approach. Proof search strategies that come out of our proof-theoretic analysis could also potentially be applied to guide proof search in various encodings of separation logics [2, 94, 68] in proof assistants: that is, they can guide the constructions of proof tactics needed to automate the reasoning tasks in those embeddings.

There are many more automated tools, formalisations, and logical embeddings for separation logics than can reasonably be surveyed within the scope of this dissertation, almost all are not directly comparable to this chapter because they deal with separation logic for some *concrete* semantics.

One exception to this concrete approach is Holfoot [95], a HOL mechanisation with support for automated reasoning about the 'shape' of SL specifications – exactly those aspects captured by abstract separation logic. However, unlike Separata, Holfoot does not support magic wand. This is a common restriction when developing automated reasoning tools for SL, because $-\!\!*$ is a source of undecidability [14]. Conversely, the mechanisations and embeddings that do incorporate magic wand tend to give little thought to (semi-) decision procedures. An important exception to this is the tableaux

of Galmiche and Méry [37], which are designed for the decidable fragment of the assertion language of concrete separation logic with $-\!\!*$, but may also be extendable to capture $=$ and quantifiers. These methods have not been implemented, and given the difficulty of the development we expect that practical implementation would be non-trivial. Another partial exception to the trend to omit $-\!\!*$ is SmallfootRG [25], which supports automation yet includes *septraction* [98]. However SmallfootRG does not support additive negation nor implication, and so $-\!\!*$ cannot be recovered; indeed in this setting septraction is mere syntactic sugar that can be eliminated.

The denigration of magic wand is not without cost, as the connective, while surely less useful than $*$, has found applications. A non-exhaustive list follows: generating weakest preconditions via backward reasoning [54]; specifying iterators [78, 59, 46]; reasoning about parallelism [33]; and various applications of septraction, such as the specification of iterators and buffers [27]. For a particularly deeply developed example, see the correctness proof for the Schorr-Waite Graph Marking Algorithm of Yang [100], which involves non-trivial inferences involving $-\!\!*$ (Lems. 78 and 79). These examples provide ample motivation to build proof calculi and tool support that include magic wand. Undecidability, which in any case is pervasive in program proof, should not deter us from seeking practically useful automation.

Also related, but so far not implemented, are the tableaux for partial-deterministic (PD) BBI of Larchey-Wendling and Galmiche [61, 60], which, as mentioned in Section 4.7, was claimed to be extendable with cancellativity to attain PASL via a "rather involved" proof. The authors recently showed that when partial-deterministic is assumed, no BBI-formula can distinguish between a cancellative model and a non-cancellative one [64]. This means that their tableau method for PD-BBI is actually complete for PASL. They also stated that the rule C in our system was admissible. Nevertheless, we intended to keep our rule C in the system because experiences in separation logic with concrete semantics show that cancellativity may not be admissible for more complicated semantics, see Example 5.2.8 for instances, or more specifically, see Gotsman et al.'s discussion [45]:

> It is well-known that if $*$ is cancellative, then for a precise q in $\mathcal{P}(\Sigma)$ and any p_1, p_2 in $\mathcal{P}(\Sigma)$, we have $(p_1 \wedge p_2) * q = (p_1 * q) \wedge (p_2 * q)$

where a predicate is "precise" if it "unambiguously carves out an area of the heap". The differences made in the above examples would not exist if cancellativity does not add new valid formulae to partial-determinism semantics. So our C rule may indeed play an important role when we extend our method to handle concrete semantics. It is unknown how Larchey-Wendling and Galmiche's tableau method can be extended to handle concrete semantics without an explicit rule for cancellativity. Moreover, their latest result does not include any treatment for non-deterministic BBI, nor for properties such as splittability and cross-split. In contrast, the relative ease with which certain properties can be added or removed from labelled sequent calculi is an important benefit of our approach; this advantage comes from structural rules which directly

capture the conditions on Kripke relational frames and handle the equality of worlds by explicit global substitutions.

Chapter 7

From Abstract to Concrete Separation Logic

Following the work on propositional abstract separation logics in the previous chapter, one natural (although orthogonal) question to ask is, can we apply our method to separation logics with concrete semantics such as the heap model etc.? This question is hard to answer because there are too many separation logics. For instance, separation logic for higher-order store [86], bunched typing [9], concurrency [15], owned variables [12, 80], rely/guarantee reasoning [98], abstract data types [57], amongst many. In this chapter we choose the separation logic with Reynolds's original semantics, but without features such as address arithmetic and arbitrary predicates. The logic we consider is foreshadowed in Section 5.1.2.2. Since most separation logic variants are based on the original one, Reynolds's heap model semantics is widely used and respected in many other variants, so a "case study" on Reynolds's separation logic may also be useful for other separation logics. Although not as powerful as Reynolds's original logic, our fragment does support all the logical connectives, including the "almighty" magic wand $-\!\!*$ [14], which is excluded in almost all automated tools we can compare with. As a result, the logic we consider is not even recursively enumerable [26], so no finite, sound, and complete proof system can be devised. This chapter relaxes the demand for completeness and focuses on realising a sound proof method that is useful in real world program verification problems.

Our labelled sequent calculus for separation logic extends $LS_{PASL} + D + CS$ with rules to handle the $\mapsto$ predicate, $=$, and quantifiers, as will be shown in Section 7.1. We then discuss several deficiencies of existing work in Section 7.2 and show how our system overcomes these problems. Capturing data structures is essential for program verification, thus we further extend our method to handle singly linked lists and binary trees in Section 7.3. Proof search and implementation are discussed in Section 7.4, where we also show that with slight adaptations, our proof calculus is sound, complete, and terminating for the most popular fragment of SL called symbolic heap [8]. Although termination is not practical in our implementation when the formulae are too large, our prover deals with a much more expressive language. Experiments are

shown in Section 7.5 and Section 7.6 concludes this chapter.

This chapter extends a paper by Hóu, Goré, and Tiu [52].

7.1 LS_{SL}: A Labelled Sequent Calculus for Separation Logic

This section presents our proof system for a fragment of Reynolds's separation logic. Recall Reynolds's original assertion logic in Section 5.1.2.1 and our fragment in Section 5.1.2.2. Our proof system is based on LS_{PASL} (cf. Section 6.1), which in turn employs the definitions of labels, labelled formulae, relational atoms, label substitution, etc. in LS_{BBI} (cf. Section 4.1). In the following we only give the definitions that are different or new.

Labels in our proof system represent heaps in the separation logic model, so we naturally use lower case h, possibly decorated by a subscript or a superscript, to denote label variables in $LVar$. Moreover, the formula $e \mapsto e'$ in SL represents a singleton heap, so we may loosely call a label h or a $\mapsto$ atomic formula a heap in the sequel. The label constant ϵ corresponds to the empty heap in the semantics. We explicitly use ternary relational atoms such as $(h_1, h_2 \triangleright h_3)$ in our proof system to indicate that the composition of the heaps represented by h_1, h_2 gives the heap represented by h_3, i.e., $h_1 \circ h_2 = h_3$. We shall use $e, e', e_1, e_2, \cdots$ to range over expressions, which can be program variables (variables defined in computer programs) or arithmetic expressions that involve program variables. However, we shall only consider expressions as atomic objects in our language. We have established the admissibility of weakening and contraction for our labelled calculi in the previous chapters. The former has always been built in our zero-premise rules, now we build in the latter by defining the sequent as a combination of sets instead of multisets. A *sequent* takes the form

$$\mathcal{G}; \Gamma \vdash \Delta$$

where $\mathcal{G}$ is a set of ternary relational atoms, Γ, Δ are sets of labelled formulae, and ; denotes set union. For example, $\Gamma; h : A$ is the union of Γ and $\{h : A\}$.

The labelled sequent calculus LS_{SL} consists of inference rules taken from $LS_{PASL} + D + CS$ [51] with the addition of a special id_2 rule, a cut rule for $=$, and the general rules for $\exists$ and $=$, as shown in Figure 7.1, and the rules for the $\mapsto$ predicate, as shown in Figure 7.2, which are new to this Chapter. The inference rules for the points-to predicate with two fields are analogous to the rules in Figure 7.2, as given in Figure 7.3. Note that the rules in Figure 7.3 are for the semantics where points-to with multiple fields is deemed as a singleton heap. In the semantics where $e \mapsto e_1, e_2$ is a heap of size two, the rules in Figure 7.3 are unsound and unnecessary. The side conditions for pointer rules mainly are based on the semantic reading of those rules. That is, whenever we want to express "there exists", we create a fresh expression or heap in the premise. In these figures we write A, B for formulae. We use $[h'_1/h_1, \ldots, h'_n/h_n]$ to

Identity and cut:

$$\frac{}{\mathcal{G};\Gamma;h:e_1\mapsto e_2\vdash h:e_1\mapsto e_2;\Delta}\ id \qquad \frac{}{\mathcal{G};\Gamma;h:e_1\mapsto e_2,e_3\vdash h:e_1\mapsto e_2,e_3;\Delta}\ id_2$$

$$\frac{\mathcal{G};\Gamma[e_1/e_2]\vdash\Delta[e_1/e_2] \qquad \mathcal{G};\Gamma\vdash h:e_1=e_2;\Delta}{\mathcal{G};\Gamma\vdash\Delta}\ cut_=$$

Logical Rules:

$$\frac{}{\mathcal{G};\Gamma;h:\bot\vdash\Delta}\ \bot L \qquad \frac{\mathcal{G}[\epsilon/h];\Gamma[\epsilon/h]\vdash\Delta[\epsilon/h]}{\mathcal{G};\Gamma;h:\top^*\vdash\Delta}\ \top^* L \qquad \frac{}{\mathcal{G};\Gamma\vdash\epsilon:\top^*;\Delta}\ \top^* R \qquad \frac{\mathcal{G};\Gamma;h:A[y/x]\vdash\Delta}{\mathcal{G};\Gamma;h:\exists x.A\vdash\Delta}\ \exists L$$

$$\frac{\mathcal{G};\Gamma\vdash h:A;\Delta \qquad \mathcal{G};\Gamma;h:B\vdash\Delta}{\mathcal{G};\Gamma;h:A\to B\vdash\Delta}\ \to L \qquad \frac{\mathcal{G};\Gamma;h:A\vdash h:B;\Delta}{\mathcal{G};\Gamma\vdash h:A\to B;\Delta}\ \to R \qquad \frac{\mathcal{G};\Gamma\vdash h:A[e/x];h:\exists x.A;\Delta}{\mathcal{G};\Gamma\vdash h:\exists x.A;\Delta}\ \exists R$$

$$\frac{(h_1,h_2\rhd h_0);\mathcal{G};\Gamma;h_1:A;h_2:B\vdash\Delta}{\mathcal{G};\Gamma;h_0:A*B\vdash\Delta}\ *L \qquad \frac{(h_1,h_0\rhd h_2);\mathcal{G};\Gamma;h_1:A\vdash h_2:B;\Delta}{\mathcal{G};\Gamma\vdash h_0:A\mathrel{-\!\!*}B;\Delta}\ \mathrel{-\!\!*}R \qquad \frac{\mathcal{G};\Gamma\theta\vdash\Delta\theta}{\mathcal{G};\Gamma;h:e_1=e_2\vdash\Delta}\ =L$$

$$\frac{(h_1,h_2\rhd h_0);\mathcal{G};\Gamma\vdash h_1:A;h_0:A*B;\Delta \qquad (h_1,h_2\rhd h_0);\mathcal{G};\Gamma\vdash h_2:B;h_0:A*B;\Delta}{(h_1,h_2\rhd h_0);\mathcal{G};\Gamma\vdash h_0:A*B;\Delta}\ *R \qquad \frac{}{\mathcal{G};\Gamma\vdash h:e=e;\Delta}\ =R$$

$$\frac{(h_1,h_0\rhd h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B\vdash h_1:A;\Delta \qquad (h_1,h_0\rhd h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B;h_2:B\vdash\Delta}{(h_1,h_0\rhd h_2);\mathcal{G};\Gamma;h_0:A\mathrel{-\!\!*}B\vdash\Delta}\ \mathrel{-\!\!*}L$$

Structural Rules:

$$\frac{(h,\epsilon\rhd h);\mathcal{G};\Gamma\vdash\Delta}{\mathcal{G};\Gamma\vdash\Delta}\ U \qquad \frac{(h_3,h_5\rhd h_0);(h_2,h_4\rhd h_5);(h_1,h_2\rhd h_0);(h_3,h_4\rhd h_1);\mathcal{G};\Gamma\vdash\Delta}{(h_1,h_2\rhd h_0);(h_3,h_4\rhd h_1);\mathcal{G};\Gamma\vdash\Delta}\ A$$

$$\frac{(h_2,h_1\rhd h_0);(h_1,h_2\rhd h_0);\mathcal{G};\Gamma\vdash\Delta}{(h_1,h_2\rhd h_0);\mathcal{G};\Gamma\vdash\Delta}\ E \qquad \frac{(\epsilon,\epsilon\rhd h_2);\mathcal{G}[\epsilon/h_1];\Gamma[\epsilon/h_1]\vdash\Delta[\epsilon/h_1]}{(h_1,h_1\rhd h_2);\mathcal{G};\Gamma\vdash\Delta}\ D$$

$$\frac{(\epsilon,h_2\rhd h_2);\mathcal{G}[h_2/h_1];\Gamma[h_2/h_1]\vdash\Delta[h_2/h_1]}{(\epsilon,h_1\rhd h_2);\mathcal{G};\Gamma\vdash\Delta}\ Eq_1 \qquad \frac{(\epsilon,h_1\rhd h_1);\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2]\vdash\Delta[h_1/h_2]}{(\epsilon,h_1\rhd h_2);\mathcal{G};\Gamma\vdash\Delta}\ Eq_2$$

$$\frac{(h_1,h_2\rhd h_0)[h_0/h_3];\mathcal{G}[h_0/h_3];\Gamma[h_0/h_3]\vdash\Delta[h_0/h_3]}{(h_1,h_2\rhd h_0);(h_1,h_2\rhd h_3);\mathcal{G};\Gamma\vdash\Delta}\ P \qquad \frac{(h_1,h_2\rhd h_0)[h_2/h_3];\mathcal{G}[h_2/h_3];\Gamma[h_2/h_3]\vdash\Delta[h_2/h_3]}{(h_1,h_2\rhd h_0);(h_1,h_3\rhd h_0);\mathcal{G};\Gamma\vdash\Delta}\ C$$

$$\frac{(h_5,h_6\rhd h_1);(h_7,h_8\rhd h_2);(h_5,h_7\rhd h_3);(h_6,h_8\rhd h_4);(h_1,h_2\rhd h_0);(h_3,h_4\rhd h_0);\mathcal{G};\Gamma\vdash\Delta}{(h_1,h_2\rhd h_0);(h_3,h_4\rhd h_0);\mathcal{G};\Gamma\vdash\Delta}\ CS$$

Side conditions:
Each label being substituted cannot be ϵ, each expression being substituted cannot be nil.
In $=L$, $\theta = mgu(\{(e_1,e_2)\})$.
In $*L$ and $\mathrel{-\!\!*}R$, the labels h_1 and h_2 do not occur in the conclusion.
In $\exists L$, y is not free in the conclusion.
In A, the label h_5 does not occur in the conclusion.
In CS, the labels h_5, h_6, h_7, h_8 do not occur in the conclusion.

Figure 7.1: Logical rules and structural rules in LS_{SL}.

$$\frac{}{\mathcal{G};\Gamma;\epsilon : e_1 \mapsto e_2 \vdash \Delta}\ {\scriptstyle \mapsto L_1} \qquad \frac{(h_1, h_0 \triangleright h_2);\mathcal{G};\Gamma;h_1 : e_1 \mapsto e_2 \vdash \Delta}{\mathcal{G};\Gamma \vdash \Delta}\ {\scriptstyle HE}$$

$$\frac{\begin{array}{c}(\epsilon, h_0 \triangleright h_0);\mathcal{G}[\epsilon/h_1, h_0/h_2];\Gamma[\epsilon/h_1, h_0/h_2];h_0 : e_1 \mapsto e_2 \vdash \Delta[\epsilon/h_1, h_0/h_2] \\ (h_0, \epsilon \triangleright h_0);\mathcal{G}[\epsilon/h_2, h_0/h_1];\Gamma[\epsilon/h_2, h_0/h_1];h_0 : e_1 \mapsto e_2 \vdash \Delta[\epsilon/h_2, h_0/h_1]\end{array}}{(h_1, h_2 \triangleright h_0);\mathcal{G};\Gamma;h_0 : e_1 \mapsto e_2 \vdash \Delta}\ {\scriptstyle \mapsto L_2}$$

$$\frac{}{(h_1, h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : e \mapsto e_1;h_2 : e \mapsto e_2 \vdash \Delta}\ {\scriptstyle \mapsto L_3} \qquad \frac{\mathcal{G};\Gamma\theta;h : e_1\theta \mapsto e_2\theta \vdash \Delta\theta}{\mathcal{G};\Gamma;h : e_1 \mapsto e_2;h : e_3 \mapsto e_4 \vdash \Delta}\ {\scriptstyle \mapsto L_4}$$

$$\frac{\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h_1 : e_1 \mapsto e_2 \vdash \Delta[h_1/h_2]}{\mathcal{G};\Gamma;h_1 : e_1 \mapsto e_2;h_2 : e_1 \mapsto e_2 \vdash \Delta}\ {\scriptstyle \mapsto L_5} \qquad \frac{}{\mathcal{G};\Gamma;h : \text{nil} \mapsto e \vdash \Delta}\ {\scriptstyle NIL}$$

$$\frac{(h_3, h_4 \triangleright h_1);(h_5, h_6 \triangleright h_2);\mathcal{G};\Gamma;h_3 : e_1 \mapsto e_2;h_5 : e_1 \mapsto e_3 \vdash \Delta \qquad (h_1, h_2 \triangleright h_0);\mathcal{G};\Gamma \vdash \Delta}{\mathcal{G};\Gamma \vdash \Delta}\ {\scriptstyle HC}$$

Side conditions:
Each label being substituted cannot be ϵ, each expression being substituted cannot be nil.
In $\mapsto L_4$, $\theta = mgu(\{(e_1, e_3), (e_2, e_4)\})$.
In *HE*, h_0 occurs in conclusion, h_1, h_2, e_1 are fresh.
In *HC*, h_1, h_2 occur in the conclusion, $h_0, h_3, h_4, h_5, h_6, e_1, e_2, e_3$ are fresh in the premise.

Figure 7.2: Pointer rules in LS_{SL}.

$$\frac{}{\mathcal{G};\Gamma;\epsilon : e_1 \mapsto e_2, e_3 \vdash \Delta} \qquad \frac{}{(h_1, h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : e \mapsto e_1, e_3;h_2 : e \mapsto e_2 \vdash \Delta}$$

$$\frac{}{\mathcal{G};\Gamma;h : \text{nil} \mapsto e, e' \vdash \Delta} \qquad \frac{}{(h_1, h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : e \mapsto e_1, e_3;h_2 : e \mapsto e_2, e_4 \vdash \Delta}$$

$$\frac{}{\mathcal{G};\Gamma;h : e_1 \mapsto e_2, e_5;h : e_3 \mapsto e_4 \vdash \Delta} \qquad \frac{\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h_1 : e_1 \mapsto e_2, e_3 \vdash \Delta[h_1/h_2]}{\mathcal{G};\Gamma;h_1 : e_1 \mapsto e_2, e_3;h_2 : e_1 \mapsto e_2, e_3 \vdash \Delta}$$

$$\frac{\mathcal{G};\Gamma\theta;h : e_1 \mapsto e_2, e_5 \vdash \Delta\theta}{\mathcal{G};\Gamma;h : e_1 \mapsto e_2, e_5;h : e_3 \mapsto e_4, e_6 \vdash \Delta}$$

Side conditions:
Each label being substituted cannot be ϵ, each expression being substituted cannot be nil.
In the last rule, $\theta = mgu(\{(e_1, e_3), (e_2, e_4), (e_5, e_6)\})$.

Figure 7.3: Inference rules for $\mapsto$ with two fields.

denote a label substitution which maps h_i to h'_i where $1 \leq i \leq n$. Label substitutions are extended to mappings between labelled formulae and labelled sequents in the obvious way. An *expression substitution* is defined similarly, where the domain is the union of program variables and expressions, and the range is the set of expressions. We use θ (possibly with subscripts) to range over expression substitutions, and we write $e\theta$ to denote the result of applying θ to e. Given a set of pairs of expressions $E = \{(e_1, e'_1), \ldots, (e_n, e'_n)\}$, a *unifier* for E is an expression substitution θ such that $e_i\theta = e'_i\theta$, for all $1 \leq i \leq n$. We assume the usual notion of *most general unifier* (mgu) from logic programming. We denote with $mgu(E)$ the most general unifier of E when it exists.

Logical and structural rules in LS_{SL} are quite straightforward as the assertion logic naturally has the following properties in DHA separation theory: identity, commutativity, associativity, partial-determinism, cancellativity, indivisible unit, disjointness, and cross-split. These properties can be defined as first-order formulae, from which one can then derive the corresponding structural rules; see the discussion in Section 4.7. Previously we have shown that the structural rules in Figure 7.1 are complete for DHA separation theory with the said properties. However, that result only holds in the abstract semantics, not in the concrete heap semantics. Also shown in our previous work is the advantage of our formulation of structural rules: if someone requires a different separation logic with respect to some other separation theory, we can simply add or remove some structural rules to obtain a proof system for that separation logic. As mentioned before, here we only concentrate on a specific separation logic rather than a range of separation logics, as the latter has already been studied in Chapter 6.

We give some intuitions about the pointer rules here. The rules $\mapsto L_1, \mapsto L_2$ specify that $e_1 \mapsto e_2$ is a singleton heap, so it cannot be empty, nor a composite heap. The rule $\mapsto L_4$ ensures that the singleton heap is a valid function that sends exactly one address to one value. These are all forecast in the future work section of a previous paper on PASL [51]. However, the $\mapsto L_3$ rule proposed in that paper, which says that any two singleton heaps with the same address are identical, is unsound for the considered concrete semantics. The corresponding $\mapsto L_3$ rule in Figure 7.2 is correct, which states that it is fine to have two singleton heaps with the same address, but they cannot be combined to form another heap. The rule $\mapsto L_5$ is also new. It and $\mapsto L_4$ state that singleton heaps are uniquely determined by the $\mapsto$ relation. The rule *NIL* states that nil cannot be in the domain of a heap.

The situation becomes much more complicated when we consider composite heaps. Since the set of addresses is infinite, it is guaranteed that we can extend any existing heaps with fresh addresses, thus we have the rule *HE*. To capture the heap composition operator $\circ$ in the semantics, we use the rule *HC*, which says that given any two heaps h_1, h_2, either they can be combined, giving the right premise; or they cannot be combined, from which we deduce that the domains of h_1 and h_2 intersect, hence there is some expression e_1 whose value is in both the domains of h_1 and h_2, yielding the left

premise. To our knowledge, proof systems for SL in the literature do not have rules similar to HE and HC, which enable us to prove many formulae that no other systems can prove.

Finally, we note that the $cut_=$ rule is admissible in LS_{SL}. However, $cut_=$ is needed when we add rules for data structures (Section 7.3).

Lemma 7.1.1. *The $cut_=$ rule is admissible in LS_{SL}.*

Proof. (Outline.) Note that there is only one rule in LS_{SL} that uses $=$ of expressions on the right hand side: $= R$. Therefore if the right premise of $cut\ =$ is not the conclusion of $= R$, we can permute $cut\ =$ upwards over that rule. We can repeat this until the $cut\ =$ instance permutes over a zero-premise rule, making the $cut\ =$ instance admissible.

If the right premise of $cut\ =$ is the conclusion of $= R$, we distinguish two cases: (1) If the cut formula is the principal formula of $= R$, then the cut formula must be $h : e = e$ for some h and e. This means that the global substitution on the left premise is the identity substitution. Thus the derivation for the left premise is enough to derive the end sequent. (2) If the cut formula is not the principal formula of $= R$, we must be able to apply $= R$ directly on the end sequent. Therefore the $cut\ =$ instance is eliminated. □

Similar to the previous chapters, a formula F is *provable* or *derivable* if there is a derivation of the sequent $\vdash h : F$ for some arbitrary label h.

A *label mapping* ρ is a function $\mathcal{L} \to H$ such that $\rho(\epsilon) = \emptyset$. Recall that a separation logic model is defined as a pair (S, H) of stores and heaps. We define an *extended separation logic model* (S, H, s, ρ) as a separation logic model plus a store $s \in S$ and a label mapping ρ. Falsifiability of a sequent is then defined as below.

Definition 7.1.1 (Sequent falsifiability)**.** *A sequent $\mathcal{G}; \Gamma \vdash \Delta$ is* falsifiable *if there is an extended separation logic model (S, H, s, ρ) such that $\rho(h_1) \circ \rho(h_2) = \rho(h_3)$ for every $(h_1, h_2 \triangleright h_3) \in \mathcal{G}$; $s, \rho(h) \Vdash A$ for every $h : A \in \Gamma$; and $s, \rho(h') \not\Vdash B$ for every $h' : B \in \Delta$.*

As in previous work for BBI and PASL, we prove the soundness of LS_{SL} by showing that each rule preserves falsifiability upwards: assuming the conclusion of a rule is falsifiable, we show that each premise is also falsifiable. Then a simple proof by contradiction gives the following theorem:

Theorem 7.1.2 (Soundness)**.** *For any formula F, and for an arbitrary label h, if the sequent $\vdash h : F$ is derivable in LS_{SL}, then F is valid in Reynolds's semantics.*

Proof. The cases for most logical rules and structural rules are straightforward adaptations from the soundness proofs in previous chapters (Theorem 4.1.1 and Theorem 6.1.1), here we show the cases for the new rules.

For $cut_=$: suppose the conclusion is falsifiable. The formula $h : e_1 = e_2$ is either true or false, if it is true, then we can safely substitute e_2 with e_1, as in the left premise; otherwise $h : e_1 = e_2$ is false, thus at least one of the premises is falsifiable.

For $\exists L$, if the conclusion is falsifiable, $h : \exists x.A$ must be true, thus there is a value v for x that makes A true in the heap $\rho(h)$. Let y be valuated as v in the store. Since y is fresh, this does not affect other value mappings in the store. Now locally replacing x with y in A preserves falsifiability.

For $\exists R$, if the conclusion is falsifiable, then $h : \exists x.A$ is false. That is, any value for x occurring in A does not make A true in the heap $\rho(h)$. So it is safe to replace x by any expression e in A, and $A[e/x]$ must be false in the heap $\rho(h)$, so the premise is also falsifiable.

For $= L$, assume that the conclusion is falsifiable, then $h : e_1 = e_2$ must be true, thus replacing e_2 with e_1, as used in θ, preserves falsifiability.

For $= R$, if the conclusion is falsifiable, then $h : e = e$ is false, but $e = e$ is universally true in every heap, so we obtain a contradiction. Therefore the conclusion cannot be falsifiable, and this zero-premise rule is sound.

For $\mapsto L_1$, the conclusion cannot be falsifiable as $e_1 \mapsto e_2$ represents a singleton heap, so it cannot be true in $\rho(\epsilon) = \emptyset$, which is an empty heap.

For $\mapsto L_2$, if the conclusion is falsifiable, then $h_0 : e_1 \mapsto e_2$ is true and $\rho(h_1) \circ \rho(h_2) = \rho(h_0)$. But $\rho(h_0)$ is a singleton heap, thus either $\rho(h_1)$ is empty, giving the first premise, or $\rho(h_2)$ is empty, giving the second premise. So the rule preserves falsifiability upwards.

For $\mapsto L_3$, the conclusion is not falsifiable, as $\rho(h_1)$ and $\rho(h_2)$ are two singleton heaps with the same domain, so they cannot be combined to form $\rho(h_0)$.

For $\mapsto L_4$, if the conclusion is falsifiable, then both $h : e_1 \mapsto e_2$ and $h : e_3 \mapsto e_4$ are true, so $e_1 \mapsto e_2$ and $e_3 \mapsto e_4$ represent the same singleton heap, hence $e_1 = e_3$ and $e_2 = e_4$, therefore the substitution on the premise preserves falsifiability.

For $\mapsto L_5$, if the conclusion is falsifiable, then both $h_1 : e_1 \mapsto e_2$ and $h_2 : e_1 \mapsto e_2$ are true. Thus h_1 and h_2 are actually the same mapping from address to value, so by definition they are the same heap, therefore $\rho(h_1) = \rho(h_2)$. The substitution $[h_1/h_2]$ hence preserves falsifiability.

The rule NIL is obviously sound, as we do not allow nil to be valuated as an address, so no heaps can make nil $\mapsto e$ true.

In HE, for any heap $\rho(h_0)$, since it has a finite domain, but the set of addresses is infinite, we must be able to find some addresses that are not in the domain of $\rho(h_0)$. Therefore we can let the store map the fresh expression e_1 to be an address disjoint from $\rho(h_0)$, then $\rho(h_1)$ must be disjoint from $\rho(h_0)$, thus can be combined with $\rho(h_0)$ and obtain a new heap $\rho(h_2)$. So we have found mappings to make $(h_1, h_0 \triangleright h_2)$ and $h_1 : e_1 \mapsto e_2$ true. Therefore HE preserves falsifiability upwards.

In HC, for any two heaps $\rho(h_1)$ and $\rho(h_2)$, either they can be combined or they cannot be combined. In the former case, we can obtain a combination of the two

heaps, and map h_0 to the combined heap, giving the right premise. In the latter case, there must be an intersection in the domain of $\rho(h_1)$ and $\rho(h_2)$. So we can find some expression e_1 whose value is in the intersection. This means that there is a subheap of $\rho(h_1)$, namely $\rho(h_3)$, whose domain only contains the value of e_1. Similarly, we can find some subheap $\rho(h_5)$ of $\rho(h_2)$ such that the domain of $\rho(h_5)$ is also the value of e_1. Now it is obvious that $(h_3, h_4 \triangleright h_1); (h_5, h_6 \triangleright h_2)$; $h_3 : e_1 \mapsto e_2$; $h_5 : e_1 \mapsto e_3$ are true in this setting. So if the conclusion is falsifiable, at least one of the premises is falsifiable.

The rules for $\mapsto$ with two fields can be proved similarly. □

7.2 Comparison with Existing Proof Calculi

This section serves two purposes. Firstly, we give some example derivations to show how our labelled calculus works, especially for the pointer rules. Secondly, we compare and contrast our calculus with existing proof calculi for "separation logics" and point out some subtleties in the literature.

Derivations in this section may involve some derived rules from LS_{SL}, for example, the derived rules for $\neg$, $\wedge$, $\vee$, $\top$ are just the same as those rules for LS_{BBI}, cf. Section 4.1.1 and 4.1.3. If it is obvious, we may use $r \times n$ to indicate that the rule r is applied n times on a sequent. We may also leave out some non-essential parts in a derivation to make the derivations more presentable.

Formula 7.1 is inspired by Vafeiadis and Parkinson's work on septraction [98]:

$$\top^* \to \neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \neg(e_1 \mapsto e_2)) \tag{7.1}$$

This formula states that if the current heap is the empty heap, then it is impossible that the combination of the empty heap and a heap $(e_1 \mapsto e_2)$ is not $(e_1 \mapsto e_2)$. The above is valid if one assumes that every value is an address, as in Galmiche and Méry's resource graph tableaux [37], Lee and Park's labelled system [65], and of course, Vafeiadis and Parkinson's combination of rely/guarantee and separation logic [98]. To prove Formula 7.1, one needs to create a singleton heap $e_1 \mapsto e_2$, even though it may not have appeared in backward proof search. This is captured by the following rule:

$$\frac{\mathcal{G}; \Gamma; h : e_1 \mapsto e_2 \vdash \Delta}{\mathcal{G}; \Gamma \vdash \Delta}\ {}_{H} \qquad \text{where } h \text{ is a fresh label.}$$

There are no conditions on expressions e_1, e_2, so they can be fresh or existing ones. If every value is an address, then the value of e_1 must be an address, too. So $e_1 \mapsto e_2$ must be a legitimate heap in Reynolds's semantics. We cannot prove Formula 7.1 using Lee and Park's labelled system nor the original resource graph tableaux, although both claimed to be complete. But the latter can be modified to do so[1], and it is not hard for the former to include a new rule like H. In our setting, however, Formula 7.1 is

[1] Confirmed via private communication.

not valid and the rule H is unsound, because we allow a set $Atoms$ of unaddressible values, it is possible that the value of e_1 is in $Atoms$, thus $e_1 \mapsto e_2$ does not represent a legitimate heap in Reynolds's semantics.

Following the above reasoning, if $e_1 \mapsto e_2$ and $e_3 \mapsto e_4$ represent two legitimate and disjoint heaps, i.e., e_1, e_3 denote distinct addresses, then we should be able to prove

$$\top^* \to \neg(((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) * (e_3 \mapsto e_4))).$$

Formula 7.2 captures this idea:

$$\begin{aligned}(\top^* \wedge \neg(e_1 = e_3)) \to (((e_1 \mapsto e_2) \mathrel{-\!\!*} \bot) \vee ((e_3 \mapsto e_4) \mathrel{-\!\!*} \bot) \vee \\ \neg(((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) * (e_3 \mapsto e_4))))\end{aligned} \quad (7.2)$$

To prove Formula 7.2, we have to combine two heaps $e_1 \mapsto e_2$ and $e_3 \mapsto e_4$ to create a larger heap, as achieved by our rule HC. We are not aware of any other proof methods for separation logic that have explicit rules like HC. The resource graph tableaux [37] could be modified to achieve a similar effect. However, unlike our calculus which has explicit inference rules, the resource graph tableau only has rules for logical connectives and points-to. The other aspects of the logic are captured via a background theory outside the logical system. As a consequence, their proofs are hard to automate. A partial derivation for Formula 7.2 is given in Figure 7.4. From the top sequent

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\epsilon, h_1 \triangleright h_1); (\epsilon, h_3 \triangleright h_3); \cdots; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4); \epsilon : B \vdash \cdots; \epsilon : (e_1 = e_3)}{(\epsilon, h_1 \triangleright h_2); (\epsilon, h_3 \triangleright h_4); \cdots; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4); \epsilon : B \vdash \cdots; \epsilon : (e_1 = e_3)}\, Eq_2 \times 2}{(h_1, \epsilon \triangleright h_2); (h_3, \epsilon \triangleright h_4); h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4); \epsilon : B \vdash h_2 : \bot; h_4 : \bot; \epsilon : (e_1 = e_3)}\, E \times 2}{; \epsilon : B \vdash \epsilon : (e_1 = e_3); \epsilon : (e_1 \mapsto e_2) \mathrel{-\!\!*} \bot; \epsilon : (e_3 \mapsto e_4) \mathrel{-\!\!*} \bot}\, \mathrel{-\!\!*} R \times 2}{; \vdash \epsilon : (e_1 = e_3); \epsilon : (e_1 \mapsto e_2) \mathrel{-\!\!*} \bot; \epsilon : (e_3 \mapsto e_4) \mathrel{-\!\!*} \bot; \epsilon : \neg B}\, \neg R}{; \vdash \epsilon : (e_1 = e_3); \epsilon : A}\, \vee R \times 2}{; \epsilon : \neg(e_1 = e_3) \vdash \epsilon : A}\, \neg L}{; h_0 : \top^*; h_0 : \neg(e_1 = e_3) \vdash h_0 : A}\, \top^* L}{; h_0 : (\top^* \wedge \neg(e_1 = e_3)) \vdash h_0 : A}\, \wedge L}{; \vdash h_0 : (\top^* \wedge \neg(e_1 = e_3)) \to A}\, \to R$$

$$\begin{aligned} A ::= & (((e_1 \mapsto e_2) \mathrel{-\!\!*} \bot) \vee ((e_3 \mapsto e_4) \mathrel{-\!\!*} \bot) \vee \\ & \neg(((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) * (e_3 \mapsto e_4)))) \\ B ::= & ((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \end{aligned}$$

Figure 7.4: A partial derivation for Formula 7.2.

in Figure 7.4, we apply HC on h_1 and h_3, obtaining two branches:

1. $(h_6, h_7 \triangleright h_1); (h_8, h_9 \triangleright h_3); \cdots; h_6 : (e_5 \mapsto e_6); h_8 : (e_5 \mapsto e_7); h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4); \cdots \vdash \cdots; \epsilon : (e_1 = e_3)$

2. $(h_1, h_3 \rhd h_5); (\epsilon, h_1 \rhd h_1); (\epsilon, h_3 \rhd h_3); \cdots; h_1 : (e_1 \mapsto e_2); h_3 : (e_3 \mapsto e_4); \epsilon : ((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) * (e_3 \mapsto e_4)) \vdash \cdots; \epsilon : (e_1 = e_3)$

To close branch 1, we apply $\mapsto L_2$ on h_1, further obtaining two branches respectively with substitutions (1.1) $[\epsilon/h_6]$ and (1.2) $[\epsilon/h_7]$. Branch (1.1) is closed by applying $\mapsto L_1$ on $\epsilon : (e_5 \mapsto e_6)$. For branch (1.2), we unify e_5 with e_1 by using $\mapsto L_4$, then we apply $\mapsto L_2$ again on h_3, and get two branches respectively with substitutions (1.2.1) $[\epsilon/h_8]$ and (1.2.2) $[\epsilon/h_9]$. Branch (1.2.1) can be closed by $\mapsto L_1$. For branch (1.2.2), we use $\mapsto L_4$ to unify e_5 with e_3. Now e_1, e_3, e_5 are all the same. Then we close the branch by apply $= R$ on $\epsilon : (e_1 = e_3)$. On branch 2, we use U to obtain $(h_5, \epsilon \rhd h_5)$, then use this ternary relational atom and the (only) $\mathrel{-\!\!*}$ formula to apply $\mathrel{-\!\!*} L$, obtaining two branches, both of which can be easily shown to have $h_5 : (e_1 \mapsto e_2) * (e_3 \mapsto e_4)$ in the succedent. After an application of $*R$ on this $*$ formula and $(h_1, h_3 \rhd h_5)$, the *id* rule can be used to close all of the remaining branches.

In the literature, the heaps in separation logic are often restricted to have finite domains, and the set of addresses is usually infinite. This is to guarantee that heap allocation can always succeed [55]. So for any heap h, no matter how large its domain is, we should always be able to combine it with some other heaps, because we can always find addresses that are not in the domain of h. Formula 7.3 says that any heap can be combined with a composite heap:

$$\neg(((\neg\top^*) * (\neg\top^*)) \mathrel{-\!\!*} \bot) \tag{7.3}$$

The key to prove this formula is to extend a heap with a fresh address. To our knowledge, current proof systems for separation logic lack this kind of mechanism. It is possible to prove this formula by changing or adding some rules in resource graph tableaux [37], but their proof relies on the fact that their language is restricted to a fragment in which every l occurring in $(l \mapsto e)$ is an address. Thus their method cannot be used in a more general situation like ours. Our derivation utilises the rule HE, as shown in Figure 7.5.

Formula 7.4 is another interesting example, which is not valid in Reynolds's separation logic and not provable in LS_{SL}, but is provable in Lee and Park's system [65].

$$(((e_1 \mapsto e_2) * \top) \mathrel{-\!\!*} \bot) \vee (((e_1 \mapsto e_3) * \top) \mathrel{-\!\!*} \neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \bot)) \vee (e_2 = e_3) \tag{7.4}$$

The meaning of the above formula may not be straightforward, but we can construct a counter-model in Reynolds's semantics for Formula 7.4 by trying to derive it in LS_{SL}. The following sequent (we do not show $\top$ in the antecedent and $\bot$ in the succedent) will occur in the backward proof search for Formula 7.4:

$$\begin{gathered}(h_5, h_6 \rhd h_1); (h_7, h_8 \rhd h_3); (h_1, h_0 \rhd h_2); (h_3, h_0 \rhd h_4); \\ h_5 : (e_1 \mapsto e_2); h_7 : (e_1 \mapsto e_3); h_4 : (e_1 \mapsto e_2) \mathrel{-\!\!*} \bot \vdash h_0 : (e_2 = e_3)\end{gathered}$$

It is easy to find a mapping ρ from each label to a legitimate heap, so that everything

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\epsilon,\epsilon\triangleright\epsilon);\cdots;\cdots h_1:e_1\mapsto e_2;\epsilon:e_3\mapsto e_4\vdash\cdots}\mapsto L_1}{(\epsilon,h_3\triangleright\epsilon);\cdots;\cdots h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash\cdots}Eq_1}{(h_3,h_1\triangleright h_5);\cdots;\cdots h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4;h_3:\top^*\vdash\cdots}\top^*L}{(h_3,h_1\triangleright h_5);\cdots;\cdots h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash h_3:\neg\top^*;\cdots}\neg R}{(h_3,h_1\triangleright h_5);\cdots;\cdots h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash h_5:((\neg\top^*)*(\neg\top^*))}*R}{(h_5,h_0\triangleright h_4);(h_3,h_1\triangleright h_5);\cdots;h_0:((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot;h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash}-\!\!*L$$

Π

$$\dfrac{\dfrac{}{\cdots;\cdots h_4:\bot\vdash}\bot L}{}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(h_0,h_5\triangleright h_4);(h_3,h_1\triangleright h_5);\cdots;h_0:((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot;h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash}{(h_2,h_3\triangleright h_4);(h_0,h_1\triangleright h_2);h_0:((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot;h_1:e_1\mapsto e_2;h_3:e_3\mapsto e_4\vdash}HE}{(h_0,h_1\triangleright h_2);h_0:((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot;h_1:e_1\mapsto e_2\vdash}HE}{;h_0:((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot\vdash}\neg R}{;\vdash h_0:\neg(((\neg\top^*)*(\neg\top^*))\mathrel{-\!\!*}\bot)}\neg R$$

N.B. The sub-derivation Π is similar to the other branch of the $*R$ rule application

Figure 7.5: A derivation for Formula 7.3 in LS_{SL}.

in the antecedent is true and everything in the succedent is false. In particular, the mapped heap $\rho(h_4)$ makes $(e_1 \mapsto e_2)\mathrel{-\!\!*} \bot$ true because the domain of $\rho(h_4)$ already contains the value of expression e_1, so $\rho(h_4)$ cannot be combined with the singleton heap $e_1 \mapsto e_2$. In Lee and Park's system, their rule $Disj\mathrel{-\!\!*}$ plays an important role in proving Formula 7.4. With this rule, we can obtain the following relational atoms:

$$(h_{10}, h_4 \triangleright h_9); (h_2, h_{12} \triangleright h_9); (h_{10}, h_{11} \triangleright h_1); (h_{11}, h_{12}, \triangleright h_3)$$

Informally, since $h_5 : (e_1 \mapsto e_2)$ is a part of the heap h_1, the heap h_5 is either in h_{10} or in h_{11}. But h_1 cannot be in h_{10}, because $h_7 : (e_1 \mapsto e_3)$ is in h_3, which is in h_4; and h_{10} can be combined with h_4 to form h_9. The heaps h_5 and h_7 certainly cannot be combined. So h_1 must be in h_{11}. Symmetrically, h_7 must also be in h_{11}. But again, h_5 and h_7 cannot co-exist in a heap. We obtain a contradiction (derivation).

The rule $Disj\mathrel{-\!\!*}$, however, is not sound in Reynolds's semantics, because when assuming two heaps can be combined, their rule only ensures that the two heaps do not have a common subheap, but it is possible for two heaps to have intersecting domains without having a common subheap in Reynolds's semantics.

The rule $\mapsto L_5$ in LS_{SL} is needed when proving Formula 7.5:

$$((e_1 \mapsto e_2)\mathrel{-\!\!*} \neg((e_3 \mapsto e_4) * \top)) \vee ((e_1 \mapsto e_2)\mathrel{-\!\!*} \neg((e_3 \mapsto e_5) * \top)) \vee (e_4 = e_5) \quad (7.5)$$

This formula says that if the current heap can be extended with two heaps, both of which satisfy $(e_1 \mapsto e_2)$, and the result of both extension contains $(e_3 \mapsto e_4)$ and $(e_3 \mapsto e_5)$ respectively, then $(e_4 = e_5)$. Proof search for this formula requires us to identify that the two added heaps satisfying $(e_1 \mapsto e_2)$ are indeed the same heap. This

can be done with $\mapsto L_5$, which is neglected in the set of putative rules from previous work [51]. We give a partial derivation for this formula in Figure 7.6.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(h_5, h_6 \rhd h_2); (h_7, h_8 \rhd h_2); (h_1, h_0 \rhd h_2); h_1 : (e_1 \mapsto e_2); h_5 : A; h_7 : B \vdash h_0 : (e_4 = e_5)}{(h_5, h_6 \rhd h_2); (h_7, h_8 \rhd h_4); (h_1, h_0 \rhd h_2); (h_1, h_0 \rhd h_4); h_1 : (e_1 \mapsto e_2); h_5 : A; h_7 : B \vdash h_0 : (e_4 = e_5)}\, P}{(h_5, h_6 \rhd h_2); (h_7, h_8 \rhd h_4); \cdots; h_1 : (e_1 \mapsto e_2); h_3 : (e_1 \mapsto e_2); h_5 : A; h_7 : B \vdash h_0 : (e_4 = e_5)}\, \mapsto L_5}{(h_1, h_0 \rhd h_2); (h_3, h_0 \rhd h_4); h_1 : (e_1 \mapsto e_2); h_3 : (e_1 \mapsto e_2); h_2 : A * \top; h_4 : B * \top \vdash h_0 : (e_4 = e_5)}\, *L \times 2}{(h_1, h_0 \rhd h_2); (h_3, h_0 \rhd h_4); h_1 : (e_1 \mapsto e_2); h_3 : (e_1 \mapsto e_2) \vdash h_2 : \neg(A * \top); h_4 : \neg(B * \top); h_0 : (e_4 = e_5)}\, \neg R \times 2}{; \vdash h_0 : ((e_1 \mapsto e_2) -\!\!* \neg(A * \top)); h_0 : ((e_1 \mapsto e_2) -\!\!* \neg(B * \top)); h_0 : (e_4 = e_5)}\, -\!\!* R \times 2}{; \vdash h_0 : ((e_1 \mapsto e_2) -\!\!* \neg(A * \top)) \vee ((e_1 \mapsto e_2) -\!\!* \neg(B * \top)) \vee (e_4 = e_5)}\, \vee L \times 2$$

$$A ::= (e_3 \mapsto e_4)$$
$$B ::= (e_3 \mapsto e_5)$$

Figure 7.6: A partial derivation for Formula 7.5 in LS_{SL}.

From the top sequent in Figure 7.6, we apply CS on $(h_5, h_6 \rhd h_2); (h_7, h_8 \rhd h_2)$, generating $(h_9, h_{10} \rhd h_5); (h_{11}, h_{12} \rhd h_6); (h_9, h_{11} \rhd h_7); (h_{10}, h_{12} \rhd h_8)$. Since h_5 is the label of $(e_3 \mapsto e_4)$, we apply $\mapsto L_2$ on $(h_9, h_{10} \rhd h_5)$, obtaining two branches with substitutions (1) $[\epsilon / h_9]$, (2) $[\epsilon / h_{10}]$ respectively. For branch (1), we apply $\mapsto L_2$ again on $(h_9, h_{11} \rhd h_7)$, further obtaining two branches with substitutions (1.1) $[\epsilon / h_9]$, (1.2) $[\epsilon / h_{11}]$ respectively. On branch (1.1), we apply Eq rules to unify $[h_5 / h_{10}]$ and $[h_7 / h_{11}]$, followed by E, A applications to generate a ternary relation of the form $(h_7, h_5 \rhd \cdot)$, then close this branch by $\mapsto L_3$. On the branch (1.2), we unify ϵ and h_7 by applying Eq rules, then use $\mapsto L_1$ to close this branch. The derivation for branch (2) is similar.

The derivation for Formula 7.5 requires the cross-split rule CS. So do Formula 7.6 and 7.7, both are crafted quite artificially. We present them as below.

$$(((e1 \mapsto e2) * (\neg((e3 \mapsto e4) * \top))) \wedge ((e3 \mapsto e4) * \top)) \rightarrow (e1 = e3) \quad (7.6)$$

$$(((e1 \mapsto e2) * \top) \wedge ((e3 \mapsto e4) * \top)) \rightarrow ((e1 = e3) \vee (((e1 \mapsto e2) * (e3 \mapsto e4)) * \top)) \quad (7.7)$$

As mentioned before, our proof system is not complete. For a valid example that cannot be proved by LS_{SL}, consider Formula 7.8:

$$\top^* \vee (\exists e1, e2.(e1 \mapsto e2)) \vee ((\neg\top^*) * (\neg\top^*)) \quad (7.8)$$

This formula is valid because any heap can only be in one of the following forms: either (1) it is the empty heap, or (2) it is a singleton heap, or (3) it is a composite heap. Recall that LS_{SL} does not have any $\mapsto$ right rules, but this formula requires one to analyse what may happen when a $\mapsto$ predicate occurs in the succedent. Thus LS_{SL}

cannot prove this formula. To prove Formula 7.8, we can add a $\mapsto R$ rule with four premises:

$$\frac{\begin{array}{c}(h_1, h_2 \triangleright h); (h_1 \neq \epsilon); (h_2 \neq \epsilon); \mathcal{G}; \Gamma \vdash h : e_1 \mapsto e_2; \Delta \\ \mathcal{G}; \Gamma; h : e_1 \mapsto e_3 \vdash h : e_2 = e_3; h : e_1 \mapsto e_2; \Delta \\ \mathcal{G}; \Gamma; h : e_3 \mapsto e_4 \vdash h : e_1 = e_3; h : e_1 \mapsto e_2; \Delta \\ \mathcal{G}[\epsilon/h]; \Gamma[\epsilon/h] \vdash \epsilon : e_1 \mapsto e_2; \Delta[\epsilon/h]\end{array}}{\mathcal{G}; \Gamma \vdash h : e_1 \mapsto e_2; \Delta} \mapsto R$$

The rule $\mapsto R$ essentially negates the semantics for $\mapsto$, obtaining four possibilities when $e_1 \mapsto e_2$ is false at a heap h: (1) h is a composite heap, so it is possible to split it into two non-empty heaps; (2) h is a singleton heap, its address is the value of e_1, but it does not map this address to the value of e_2; (3) h is a singleton heap, but its address is not the value of e_1; (4) h is the empty heap. We will also need a new type of structure, namely inequality of labels as shown in the first premise. This structure was also introduced in the rules for Splittability in Section 6.3. Now Formula 7.8 is provable, but there would be other formulae that require us to add more rules, and one can never give enough rules for this logic. For efficiency reasons we do not include the rule $\mapsto R$ into our labelled calculus. Our method, however, is able to prove simpler properties such as "every heap is either empty or non-empty", which can be formulated as $\top^* \vee \neg\top^*$ and can be proved trivially.

7.3 Inference Rules for Data Structures

Many data structures can be defined inductively by using separation logic's assertion language [18]. Numerous original tools for separation logic, such as Smallfoot, use hard-coded inductive prediaces because it is possible to do so and still obtain a complete and decidable theory. However, more recent tools, including $Cyclist_{SL}$ [19], provide facilities for arbitrary user-defined inductive predicates. In such setting, the logic is known not to be decidable any more [1]. The performance of tools not specifically designed for decidable fragments is generally worse on those fragments than those that are so specifically designed. In this section we take the hard-coded approach and focus on two widely used data structures: singly linked lists and binary trees. We take several rules from Berdine et al.'s method for symbolic heap entailment [8] and give corresponding labelled versions, then we extend these rules with new labelled rules for separation logic in general.

Linked lists are defined in several ways in the literature. Here we use the definition in provers for symbolic heap [18, 8], given below, to facilitate comparison between our prover and those provers.

$$ls(e_1, e_2) \iff (e_1 = e_2 \wedge \top^*) \vee (e_1 \neq e_2 \wedge \exists x.(e_1 \mapsto x * ls(x, e_2)))$$

In this definition, a linked list (segment) $ls(e_1, e_n)$ is in the form

$$(e_1 \mapsto e_2) * (e_2 \mapsto e_3) * \cdots * (e_{n-2} \mapsto e_{n-1}) * (e_{n-1} \mapsto e_n)$$

where $e_1, \cdots e_n$ are all mapped to distinct values by the store. A complete list that starts with e is defined as $ls(e, \text{nil})$. Cycles are not allowed in this definition, since any list denoted by $ls(e, e)$ or $ls(\text{nil}, \text{nil})$ is an empty list, and any heaps with intersecting domains are not allowed to be combined. More specifically, for a non-empty list segment $ls(e_1, e_n)$, suppose there is a cycle from e_i to e_i, with at least one singleton heap in between, there are two possibilities:

1. If the last occurrence of e_i is not at the end of the list, i.e., $e_i \neq e_n$, then it must be the address of a singleton heap, otherwise the next singleton heap also has address e_i.
   (a) If the first occurrence of e_i is the content of a singleton heap, then the next singleton heap has address e_i, and this singleton heap cannot be the one where e_i occurs last, because they are disjoint, we obtain a contradiction because two singleton heaps with the same address are combined.
   (b) Otherwise the first occurrence of e_i is the address of a singleton heap, we obtain the same contradiction.
2. If the last occurrence of e_i is the end of the list, then by the inductive definition, the list $ls(e_1, e_i)$ will be parsed to have a tail $ls(e_i, e_i)$, which is equivalent to an empty list, rather than having a series of singleton heaps with a cycle.

To represent binary trees we need the pointer predicate with two fields, i.e., $(e \mapsto e_1, e_2)$. The two fields are for the left subtree and right subtree respectively, the empty tree is given by nil. The binary tree structure is defined as:

$$tr(e) \iff ((e = \text{nil}) \wedge \top^*) \vee (\exists x, y.((e \mapsto x, y) * tr(x) * tr(y)))$$

The $*$ connective ensures that there is no sharing and cycles between the two subtrees.

One can also define doubly linked lists, reverse linked lists etc., and give inference rules for these data structures. But as they are less frequently used, we leave them as future work.

We give the first part of the inference rules for data structures in Figure 7.7. Some of these rules are just labelled versions of the rules in the entailment checking system in Smallfoot [8], but we also extend their rules with some new ones for non-symbolic heaps. For example, the rules LS_6 and LS_7 are for overlaid data structures that cannot be expressed in symbolic heap fragment. The abbreviation $ds(e, e')$ stands for a data structure where the first address is the value of e and the last content is the value of e'. Examples of $ds(e, e')$ are $(e \mapsto e')$ and $ls(e, e')$. We use $ad(e)$ for a data structure that *may* contain the address of value of e, i.e., $(e \mapsto e')$, $(e \mapsto e', e'')$, $ls(e, e')$, or $tr(e)$,

$$\frac{\mathcal{G};\Gamma[e_1/e_2] \vdash \Delta[e_1/e_2]}{\mathcal{G};\Gamma;\epsilon : ls(e_1,e_2) \vdash \Delta}\ LS_1 \qquad \frac{}{\mathcal{G};\Gamma \vdash \epsilon : ls(e,e);\Delta}\ LS_2 \qquad \frac{\mathcal{G};\Gamma;h : \top^* \vdash \Delta}{\mathcal{G};\Gamma;h : ls(e,e) \vdash \Delta}\ LS_3$$

$$\frac{\mathcal{G};\Gamma[\text{nil}/e];h : \top^* \vdash \Delta[\text{nil}/e]}{\mathcal{G};\Gamma;h : ls(\text{nil},e) \vdash \Delta}\ LS_4 \qquad \frac{}{\mathcal{G};\Gamma;h : A \vdash h : A;\Delta}\ id_a$$

$$\frac{\mathcal{G};\Gamma\theta_1;h : \top^* \vdash \Delta\theta_1 \qquad \mathcal{G};\Gamma\theta_2;h : ls(e_1\theta_2,e_2\theta_2) \vdash \Delta\theta_2}{\mathcal{G};\Gamma;h : ls(e_1,e_2);h : ls(e_3,e_4) \vdash \Delta}\ LS_5$$

$$\frac{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : ds(e_1,e_2);h_0 : ls(e_1,e_3);h_2 : ls(e_2,e_3) \vdash \Delta}{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : ds(e_1,e_2);h_0 : ls(e_1,e_3) \vdash \Delta}\ LS_6$$

$$\frac{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : ds(e_2,e_3);h_0 : ls(e_1,e_3);h_2 : ls(e_1,e_2) \vdash \Delta}{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : ds(e_2,e_3);h_0 : ls(e_1,e_3) \vdash \Delta}\ LS_7$$

$$\frac{(h_1,h_2 \triangleright h_0);(h_1,h_3 \triangleright h_4);\mathcal{G};\Gamma;h_1 : ds(e_1,e_2);h_3 : ad(e_3) \vdash h_2 : ls(e_2,e_3);h_0 : ls(e_1,e_3);h : G(ad(e_3));\Delta}{(h_1,h_2 \triangleright h_0);(h_1,h_3 \triangleright h_4);\mathcal{G};\Gamma;h_1 : ds(e_1,e_2);h_3 : ad(e_3) \vdash h_0 : ls(e_1,e_3);h : G(ad(e_3));\Delta}\ LS_8$$

$$\frac{}{(h_1,h_2 \triangleright h_0);\mathcal{G};\Gamma;h_1 : ad(e_1);h_2 : ad(e_1)' \vdash h_3 : G(ad(e_1));h_4 : G(ad(e_1)');\Delta}\ IC$$

Abbreviations and side conditions:
$ds(e,e')$ is either $(e \mapsto e')$ or $ls(e,e')$.
$ad(e)$ stands for one of $(e \mapsto e'), (e \mapsto e',e''), ls(e,e')$, or $tr(e)$, for some e',e''. Similarly for $ad(e)'$.
$G(ad(e))$ is defined as $G(e \mapsto e') \equiv G(e \mapsto e',e'') \equiv \bot, G(ls(e,e')) \equiv (e = e'), G(tr(e)) \equiv (e = \text{nil})$.
In LS_5, $\theta_1 = mgu(\{(e_1,e_2),(e_3,e_4)\})$ and $\theta_2 = mgu(\{(e_1,e_3),(e_2,e_4)\})$.
In LS_8, if e_3 is nil, then $(h_1,h_3 \triangleright h_4), h_3 : ad(e_3)$ and $h : G(ad(e_3))$ in the conclusion are optional.
In LS_8, if $ds(e_1,e_2)$ is $(e_1 \mapsto e_2)$, then $(h_1,h_3 \triangleright h_4), h_3 : ad(e_3)$ and $h : G(ad(e_3))$ in the conclusion are optional, on the condition that $h' : (e_1 = e_3)$ occurs in the RHS of the conclusion, for some h'.

Figure 7.7: Data structure rules part 1.

for some e',e''. In the case that $e = e'$, the data structure $ls(e,e')$ actually is empty and does not contain e. We use $G(ad(e))$ on the right hand side of the sequent to ensure that the data structure $ad(e)$ is non-empty. Since the pointer predicate is guaranteed to be non-empty by the semantics, $G(e \mapsto e') \equiv G(e \mapsto e,e'') \equiv \bot$. By the definition of lists and trees, $G(ls(e,e')) \equiv (e = e')$, and $G(tr(e)) \equiv (e = \text{nil})$. For example, if $ls(e,e')$ is empty, then $G(ls(e,e')) \equiv (e = e')$ on the right hand side will immediately close the branch. Here we assume that $\bot$ is false everywhere, so $h : \bot \in \Delta$ for every label h, although this is not shown explicitly.

While other rules in Figure 7.7 are intuitive, the rule LS_8 is extraordinarily complicated and needs some explanation. For simplicity, in the following we will say an expression e is an "address" without saying the fact that it needs to be valuated by the store. Suppose the heap h_1 is a data structure from address e_1 to e_2, and h_3 is a data structure that talks about address e_3. By $G(ad(e_3))$ in the succedent, we know that h_3 is non-empty and indeed contains the address e_3. Since $(h_1,h_3 \triangleright h_4)$ holds, we know that the address e_3 is not in the domain of h_1. The labelled formula $h_0 : ls(e_1,e_3)$ in the

succedent now tells us that h_0 should also make $ds(e_1, e_2) * ls(e_2, e_3)$ false, no matter in which form the $ds(e_1, e_2)$ in the antecedent is. Thus by an $*R$ application on this formula using $(h_1, h_2 \triangleright h_0)$, the branch with $h_1 : ds(e_1, e_2)$ in the succedent can be closed, and we only have the other branch with $h_2 : ls(e_2, e_3)$ in the succedent. There are two special cases as indicated by the side conditions. First, if e_3 is nil, then we do not need to worry about it appearing in the addresses of $ds(e_1, e_2)$, because nil can never be an address. So we do not need $(h_1, h_3 \triangleright h_4)$, $h_3 : ad(e_3)$ and $h : G(ad(e_3))$ in the conclusion. Second, if $ds(e_1, e_2)$ is a singleton heap $(e_1 \mapsto e_2)$, then we only require that e_3 does not have the same value as e_1, thus $(h_1, h_3 \triangleright h_4)$, $h_3 : ad(e_3)$ and $h : G(ad(e_3))$ can be neglected as long as $(e_1 = e_3)$ occurs in the succedent.

The rules IC and id_a are respectively the generalised versions of $\mapsto L_3$ and id. That is, IC captures that two data structures that contain the same address cannot be composed by $*$, and id_a simply forbids a heap to make a formula both true and false.

$$\frac{\mathcal{G}; \Gamma[\text{nil}/e] \vdash \Delta[\text{nil}/e]}{\mathcal{G}; \Gamma; \epsilon : tr(e) \vdash \Delta}\ TR_1 \qquad \frac{\mathcal{G}; \Gamma[e_1/e_2]; h : tr(e_1) \vdash \Delta[e_1/e_2]}{\mathcal{G}; \Gamma; h : tr(e_1); h : tr(e_2) \vdash \Delta}\ TR_4$$

$$\frac{}{\mathcal{G}; \Gamma \vdash \epsilon : tr(\text{nil}); \Delta}\ TR_2 \qquad \frac{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma; h_1 : e \mapsto e_1, e_2; h_0 : tr(e); h_2 : tr(e_1) * tr(e_2) \vdash \Delta}{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma; h_1 : e \mapsto e_1, e_2; h_0 : tr(e) \vdash \Delta}\ TR_5$$

$$\frac{\mathcal{G}; \Gamma; h : \top^* \vdash \Delta}{\mathcal{G}; \Gamma; h : tr(\text{nil}) \vdash \Delta}\ TR_3 \qquad \frac{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma; h_1 : e \mapsto e_1, e_2 \vdash h_2 : tr(e_1) * tr(e_2); h_0 : tr(e); \Delta}{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma; h_1 : e \mapsto e_1, e_2 \vdash h_0 : tr(e); \Delta}\ TR_6$$

Figure 7.8: Data structure rules part 2.

The second collection of rules are for binary trees, as shown in Figure 7.8. Similarly, the rules TR_1, TR_2, TR_3, TR_6 can be found in similar forms in the Smallfoot system [8], but TR_4 and TR_5 are for non-symbolic heaps, respectively are analogous to LS_5 and LS_6 for linked lists. Note that TR_4 only needs one premise, because the case where two trees are empty ($e_1 = e_2 = \text{nil}$) implies the case that the two trees are the same ($e_1 = e_2$), so only the latter case is used as the premise. In the rule TR_6, we implicitly unfold $tr(e)$ to $(e \mapsto e_1, e_2) * tr(e_1) * tr(e_2)$ as is done by Berdine et al. [8], and apply $*R$ on this formula, the branch with $(e \mapsto e_1, e_2)$ is immediately closed, leaving only the branch with $tr(e_1) * tr(e_2)$, thus the formula $tr(e)$ is also copied to the premise by the $*R$ application.

Note also that the rule $cut_=$ in LS_{SL} is required when we consider data structures. For example, when proving the following formula which says that the current heap consists of a list $ls(e_2, e_3)$ and a list $ls(e_1, e_2)$, and the latter can be extended with an non-empty list $ls(e_3, e_4)$, then the current heap forms a list $ls(e_1, e_3)$:

$$((ls(e_1, e_2) \wedge \neg((ls(e_3, e_4) \wedge \neg\top^*) -\!\!* \bot)) * ls(e_2, e_3)) \rightarrow ls(e_1, e_3) \tag{7.9}$$

The above formula would not be valid if we do not have the subformula $\neg\top^*$, which ensures that the list $ls(e_3, e_4)$ is non-empty and the address e_3 is indeed disjoint from

$$\frac{}{\mathcal{G};\Gamma;h:e\mapsto\omega,\omega'\vdash h:e\mapsto\omega;\Delta} \qquad \frac{\mathcal{G};\Gamma\theta;h:e_1\mapsto\omega\vdash\Delta\theta}{\mathcal{G};\Gamma;h:e_1\mapsto\omega;h:e_2\mapsto\omega'\vdash\Delta}$$

θ is the *mgu* that unifies e_1 with e_2 and ω with ω'.

$$\frac{}{\mathcal{G};\Gamma;\epsilon:e\mapsto\omega\vdash\Delta} \qquad \frac{}{\mathcal{G};\Gamma;h:\text{nil}\mapsto\omega\vdash\Delta}$$

$$\frac{\begin{array}{c}(\epsilon,h_0\triangleright h_0);\mathcal{G}[\epsilon/h_1][h_0/h_2];\Gamma[\epsilon/h_1][h_0/h_2];h_0:e_1\mapsto\omega\vdash\Delta[\epsilon/h_1][h_0/h_2]\\(h_0,\epsilon\triangleright h_0);\mathcal{G}[\epsilon/h_2][h_0/h_1];\Gamma[\epsilon/h_2][h_0/h_1];h_0:e_1\mapsto\omega\vdash\Delta[\epsilon/h_2][h_0/h_1]\end{array}}{(h_1,h_2\triangleright h_0);\mathcal{G};\Gamma;h_0:e_1\mapsto\omega\vdash\Delta}$$

$$\frac{\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h_1:e_1\mapsto\omega,\omega';h_1:e_1\mapsto\omega\vdash\Delta[h_1/h_2]}{\mathcal{G};\Gamma;h_1:e_1\mapsto\omega,\omega';h_2:e_1\mapsto\omega\vdash\Delta}$$

Figure 7.9: Generalised rules for $\mapsto$ with arbitrary fields in non-Reynolds's semantics.

the heap that denotes $ls(e_1,e_2)$. It is essential to use $cut_=$ to split the case of $e_3=e_4$ and $e_3\neq e_4$ in the proof of this formula.

We refer to the labelled system LS_{SL} plus the rules introduced in Figure 7.7 and 7.8 as $LS_{SL}+DS$. The soundness of $LS_{SL}+DS$ for Reynolds's semantics can be proved in the same way as previously showed for BBI.

Before we discuss the completeness w.r.t. the symbolic heap fragment, let us recall from the end of Section 5.1.2.1 that symbolic heap employs slightly different semantics for the multi-field points-to predicate, and treat it as a singleton heap. This reading would not make sense in our setting because our logic is based on Reynolds's semantics. Here we develop a branch of our system by compromising both kinds of semantics and viewing $(e_1\mapsto e_2,e_3)$ as a singleton heap that maps the value of e_1 to the value of e_2, and the next address contains the value of e_3. Thus we need the following rule to match up with Smallfoot:

$$\frac{}{\mathcal{G};\Gamma;h:e_1\mapsto e_2,e_3\vdash h:e_1\mapsto e_2;\Delta}\ id_3$$

This rule, however, is not sound in Reynolds's semantics in which $e_1\mapsto e_2,e_3$ is a heap of size two thus is not contradictory when $e_1\mapsto e_2$ is not true at that heap.

We can even generalise the above to the case for the points-to predicate with arbitrary number of fields. We give the generalised $\mapsto$ rules for non-Reynolds's semantics in Figure 7.9 where ω,ω' denote any number of fields.

In the non-Reynolds's semantics, the rules in Figure 7.7 need to be adjusted so that $ds(e1,e2)$ now considers $(e_1\mapsto e_2,\omega)$ and $ad(e)$ considers $(e\mapsto\omega)$. Similarly, the $h_1:e\mapsto e_1,e_2$ in Figure 7.8 should be changed to $h_1:e\mapsto e_1,e_2,\omega$. We refer to the variant of $LS_{SL}+DS$ with these changes, the addition of rules in Figure 7.9 and the exclusion of rules in Figure 7.3 as $LS'_{SL}+DS$. In the following we list some important lemmas for proving the completeness of $LS'_{SL}+DS$ for the symbolic heap fragment.

Lemma 7.3.1 (Global substitution of labels)**.** *If a sequent $\mathcal{G};\Gamma \vdash \Delta$ is derivable in $LS'_{SL} + DS$, then $\mathcal{G}[h'/h];\Gamma[h'/h] \vdash \Delta[h'/h]$ is also derivable in $LS'_{SL} + DS$ with at most the same height, where h, h' are two labels and $h \neq \epsilon$.*

Proof. The proof is similar to Lemma 4.2.2. We do an induction on the height of the derivation for the original sequent. The base case is when the original sequent can be derived in 1 step, i.e., by a zero-premise rule. It is easy to check that we can also apply the corresponding zero-premise rule on the substituted sequent.

For the inductive case, consider the last (bottom) rule application in the derivation for the original sequent. The rules with only substitutions in $LS'_{SL} + DS$ can be shown by similar arguments for Eq_1, Eq_2 in Lemma 4.2.2. The rules that create fresh labels (HE and HC) can be shown by using the argument for $*L, -\!\!* R$ in Lemma 4.2.2. Most other rules are easy to show, we give an example here for LS_5, which not only involves substitutions, but also creates new labelled formulae in the left premise. Suppose the derivation for the original sequent is as below, where $e_2 \neq e_4$ and $e_3 \neq e_4$ (the case when $e_3 = e_4$ is similar, but we need to use $[e_3/e_1][e_3/e_2]$ instead on the right premise; when $e_2 = e_4$, use $[e_2/e_1][e_2/e_3]$ instead on the left premise):

$$\frac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \mathcal{G};\Gamma[e_1/e_2][e_3/e_4];h:\top^* \vdash \Delta[e_1/e_2][e_3/e_4] & \mathcal{G};\Gamma[e_1/e_3][e_2/e_4];h:ls(e_1,e_2) \vdash \Delta[e_1/e_3][e_2/e_4]\end{array}}{\mathcal{G};\Gamma;h:ls(e_1,e_2);h:ls(e_3,e_4) \vdash \Delta}\ LS_5$$

Our objective is to derive $\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h:ls(e_1,e_2)[h_1/h_2];h:ls(e_3,e_4)[h_1/h_2] \vdash \Delta[h_1/h_2]$, for some labels h_1, h_2. We consider the following cases:

1. If neither h_1 nor h_2 is h, then we can apply LS_5 on the substituted sequent, getting two premises:

   - $\mathcal{G};\Gamma[h_1/h_2][e_1/e_2][e_3/e_4];h:\top^* \vdash \Delta[h_1/h_2][e_1/e_2][e_3/e_4]$
   - $\mathcal{G};\Gamma[h_1/h_2][e_1/e_3][e_2/e_4];h:ls(e_1,e_2) \vdash \Delta[h_1/h_2][e_1/e_3][e_2/e_4]$

   Since label substitutions and expression substitutions are independent, these two premises can be derived by using the induction hypothesis on Π_1 and Π_2 respectively with $[h_1/h_2]$.

2. If $h_1 = h$ and $h_2 \neq h$, same as the above case.

3. If $h_2 = h$ and $h_1 \neq h$, then we need to derive $\mathcal{G}[h_1/h_2];\Gamma[h_1/h_2];h_1:ls(e_1,e_2);h_1:ls(e_3,e_4) \vdash \Delta[h_1/h_2]$. Similar as before, we apply LS_5 backwards on this sequent, and obtain two premises that are exactly the same as those in case 1. Again, we use the induction hypothesis to apply $[h_1/h_2]$ on Π_1 and Π_2 to obtain respectively the derivations for the above two premises.

4. If $h_1 = h_2 = h$, the substitution $[h_1/h_2]$ is the identity substitution, so it does not change the conclusion at all. □

Global substitution of expressions is similar to global substitution of labels, but in this case we do not need to consider the set $\mathcal{G}$ of relational atoms in a sequent, because it does not contain any expressions.

Lemma 7.3.2 (Global substitution of expressions)**.** *If a sequent $\mathcal{G};\Gamma \vdash \Delta$ is derivable in $LS'_{SL} + DS$, then $\mathcal{G};\Gamma[e'/e] \vdash \Delta[e'/e]$ is also derivable in $LS'_{SL} + DS$ with at most the same height, where e, e' are two expressions and $e \neq nil$.*

Proof. This proof is similar to the substitution for labels, which is an induction on the height of the derivation for the original sequent. Both the base case and the inductive case are easy to check. We only give an example for $= L$ here. Suppose the derivation for the original sequent is

$$\dfrac{\begin{array}{c}\Pi\\ \mathcal{G};\Gamma[e_2/e_1] \vdash \Delta[e_2/e_1]\end{array}}{\mathcal{G};\Gamma;h : e_1 = e_2 \vdash \Delta}\,{=}L$$

We need a derivation for the sequent $\mathcal{G};\Gamma[e'/e];h : e_1 = e_2[e'/e] \vdash \Delta[e'/e]$. A case analysis as follows suffices.

1. If e, e' and e_1, e_2 are pairwise distinct. Then we only need to apply the induction hypothesis on Π, swap the two substitutions, and apply $= R$ to derive the required sequent.

2. If $e = e_1$ and e' is not e_1 nor e_2, then the derivation is transformed as below:

$$\dfrac{\begin{array}{c}\Pi'\\ \mathcal{G};\Gamma[e'/e][e_2/e'] \vdash \Delta[e'/e][e_2/e']\end{array}}{\mathcal{G};\Gamma[e'/e];h : e' = e_2 \vdash \Delta[e'/e]}\,{=}L$$

   where Π' is obtained by the induction hypothesis on Π with $[e_2/e']$.

3. If $e = e_2$, e' is not e_1 nor e_2, this case is similar to the first case.

4. If $e' = e_1$, e is not e_1 nor e_2, this case is similar to the second case.

5. If $e' = e_2$, e is not e_1 nor e_2, this case is similar to the first case.

6. If $e = e_1$, $e' = e_2$, we can directly use Π to derive the required sequent.

7. if $e = e_2$, $e' = e_1$, we can apply the induction hypothesis on Π with $[e_1/e_2]$ to get the derivation for the required sequent. □

Weakening is built in our zero-premise rules and is easy to check, thus we only state the lemma as below.

Lemma 7.3.3. *If $\mathcal{G};\Gamma \vdash \Delta$ is derivable in $LS'_{SL} + DS$, then for any set $\mathcal{G}'$ of relational atoms, any set Γ' and Δ' of labelled formulae, the sequent $\mathcal{G};\mathcal{G}';\Gamma;\Gamma' \vdash \Delta;\Delta'$ is derivable with the same height in $LS'_{SL} + DS$.*

Most non-logical rules in $LS'_{SL} + DS$ are either about global substitutions, in which case the above two lemmas guarantee invertibility, or are trivially invertible, i.e., the premise(s) includes the conclusion. So the following lemma is not surprising.

Lemma 7.3.4 (Invertibility of rules)**.** *If the conclusion of a rule in $LS'_{SL} + DS$ is derivable in $LS'_{SL} + DS$, then each premise is also derivable in $LS'_{SL} + DS$ with at most the same height.*

Proof. Most of rules in $LS'_{SL} + DS$ can be shown invertible by similar (but longer) arguments for the same rules in LS_{BBI}. The pointer rules in Figure 7.2 are trivially invertible, which can be shown by either weakening or substitution lemmas (in case of zero-premise rules, nothing is required). The same argument holds for the following rules for data structures: LS_1, LS_2, LS_6, LS_7, LS_8, TR_1, TR_2, TR_4, TR_5, TR_6.

For LS_3, we prove by induction on the height of the derivation for the conclusion

$$\mathcal{G}; \Gamma; h : ls(e, e) \vdash \Delta.$$

If $h : ls(e, e)$ is not principal in the last rule application, then we can apply that rule backwards on $\mathcal{G}; \Gamma; h : \top^* \vdash \Delta$, too. If $h : ls(e, e)$ is principal in the last rule application, we need to look at the last rule application in more details. If the last rule is id_a, then we know that $h : ls(e, e)$ is in Δ. The required sequent can be derived by applying $\top^* L$ backwards, unifying h with ϵ, then use $\top^* R$ to close the branch. If the last rule is LS_1 or LS_4, it is trivial to show. If the last rule is LS_5, which applies on, e.g., $\mathcal{G}; \Gamma; h : ls(e, e); h : ls(e_3, e_4) \vdash \Delta$, and gives the derivations for the sequents

$$\Pi_1 : \ (\mathcal{G}; \Gamma; h : \top^* \vdash \Delta)[e_3/e_4]$$

$$\Pi_2 : \ (\mathcal{G}; \Gamma; h : ls(e, e) \vdash \Delta)[e/e_3][e/e_4]$$

Now we need to derive $\mathcal{G}; \Gamma; h : \top^*; h : ls(e_3, e_4) \vdash \Delta$. We use the following derivation:

$$\dfrac{\dfrac{(\mathcal{G}; \Gamma \vdash \Delta)[\epsilon/h][e_3/e_4]}{(\mathcal{G}; \Gamma; \epsilon : ls(e_3, e_4) \vdash \Delta)[\epsilon/h]}\ {\scriptstyle LS_1}}{\mathcal{G}; \Gamma; h : \top^*; h : ls(e_3, e_4) \vdash \Delta}\ {\scriptstyle \top^* L}$$

Note that the rule $\top^* L$ is also invertible, thus by applying the induction hypothesis on Π_1, we obtain a derivation Π'_1 for $(\mathcal{G}; \Gamma \vdash \Delta)[e_3/e_4][\epsilon/h]$, which is what we need.

If the last rule application is LS_6, LS_7, LS_8, we can simply use the induction hypothesis to derive the required sequent. Finally, if the last rule application is IC, the guard formula $G(ad(e_1))$ and $G(ad(e_1)')$ ensures that if $h : ls(e, e)$ is principal, which means $e_1 = e$, and $(e_1 = e_1)$ occurs in the succedent, we can use $= R$ to close the branch. The above includes all possible cases for $h : ls(e, e)$ being a principal formula.

The invertibility of LS_4 is similar to that for LS_3.

For LS_5, if the conclusion is derivable, the left premise is derivable by Lemma 7.3.2 and invertibility of LS_3; the right premise is derivable directly by Lemma 7.3.2.

The case for TR_3 is similar to that for LS_3. □

Since we assume that our sequents consist of sets rather than multisets, contraction of labelled formulae and relational atoms are built in. To show the completeness w.r.t. the symbolic heap fragment (defined in Section 5.1.2.3), we only need to give a translation from the entailment checking system in Smallfoot to our labelled system, and show that every rule in the former system can be mimicked in the latter system. Since Smallfoot is complete for the symbolic heap fragment [8], we obtain the completeness of our system for the symbolic heap fragment.

But before we proceed to prove the completeness for $LS'_{SL} + DS$, we also need the following lemma:

Lemma 7.3.5. *Let $\Pi \wedge \Sigma$ and $\Pi' \wedge \Sigma'$ be two arbitrary symbolic heaps, then $h_1 : \Pi; h_2 : \Sigma \vdash h_3 : \Pi'; h_4 : \Sigma'$ is derivable in $LS'_{SL} + DS$ iff $h'_1 : \Pi; h_2 : \Sigma \vdash h'_3 : \Pi'; h_4 : \Sigma'$ is derivable in $LS'_{SL} + DS$, for any labels $h_1, h_2, h_3, h_4, h'_1, h'_3$.*

The proof for this lemma is an easy induction on the height of the derivation for the former sequent. Since Π and Π' only contain (negated) equality atomic formulae and $\top$ combined by $\wedge$, the labels for Π and Π' play no role in proof search at all, so we can replace them with any other labels.

We call $*R$, $-\!\!* L$, $\exists R$ *positive* rules, and the other rules in $LS'_{SL} + DS$ *negative* rules. The idea is, negative rules can be applied whenever applicable, but positive rules need to be applied with the "right" choice (a wrong choice does not make any progress in proof search). Ideally we would always apply negative rules, and apply positive rules only if no negative rules are applicable. It is easy to check that applying negative rules (with the restriction mentioned in Section 7.5) is a terminating procedure. We call a sequent a *negative closure* when no negatives rules are applicable to it.

In the following, we shall restrict the $*R$ rule in LS'_{SL} so that the principal formula is not copied to each premise. We call the resulting system $LS''_{SL} + DS$, in which the $*R$ rule is not invertible any more. But as we shall see, the invertibility of $*R$ is not used in the completeness proof for $LS''_{SL} + DS$, which will then induce the completeness of the stronger system $LS'_{SL} + DS$. In the following, we say certain rules are "saturated" when these rules are not applicable any more.

Lemma 7.3.6. *Let $\mathcal{G}; \Gamma \vdash \Delta$ be a $LS''_{SL} + DS$ derivable sequent in the proof search for a symbolic heap formula. Then either*

1. *it can be derived by only using negative rules, or*

2. *for each open branch after the negative rules are saturated, let $\mathcal{G}'; \Gamma' \vdash \Delta'$ be the negative closure of the top sequent, there is some $h : \Sigma' * \Sigma'' \in \Delta'$ and $(h_1, h_2 \triangleright h) \in \mathcal{G}'$ such that*
   (I) $\mathcal{G}'; \Gamma' \vdash h_1 : \Sigma'; \Delta''$ and
   (II) $\mathcal{G}'; \Gamma' \vdash h_2 : \Sigma''; \Delta''$
   are both derivable in $LS''_{SL} + DS$, where Δ'' is $\Delta' \setminus \{h : \Sigma' * \Sigma''\}$.

Proof. If (1) does not hold, then by the nature of our (negative) inference rules, the structure Δ' consists of only atomic labelled formulae and $*$ formulae, and Γ' consists of only atomic labelled formulae. Since we assume that no negative rules are applicable on the sequent

(S:) $\mathcal{G}'; \Gamma' \vdash \Delta'$,

this sequent can only be derived by first applying $*R$ backwards. So there must be some $(h_1, h_2 \triangleright h) \in \mathcal{G}'$ which is the "right" choice for this $*R$ application on some $h : \Sigma' * \Sigma'' \in \Delta'$ that leads to a derivation, so

(I) $\mathcal{G}'; \Gamma' \vdash h_1 : \Sigma'; \Delta'$ and
(II) $\mathcal{G}'; \Gamma' \vdash h_2 : \Sigma''; \Delta'$

are derivable. Note that from now on each branch in the derivation for (I) and (II) can only be closed by id, id_2, id_s, $\top^* R$, LS_2, or TR_2, because if any other zero-premise rules are applicable, they are applicable on S, too. □

Theorem 7.3.7 (Soundness and Completeness w.r.t. symbolic heap)**.** *Any symbolic heap formula provable in $LS'_{SL} + DS$ is valid, and any valid symbolic heap formula is provable in $LS'_{SL} + DS$.*

Proof. The soundness of the rules in and 7.8 $LS'_{SL} + DS$ are easy to check, by arguing that each rule preserves falsifiability upwards. The cases for many rules for linked lists and binary trees are obvious since they are based on the rules in Smallfoot. We give the cases for LS_6, LS_7, TR_4, TR_5 below. From the reading of a sequent, we may loosely say a labelled formula is "true" when it appears on the left hand side of a sequent, and say it is "false" if it appears on the right hand side.

LS_6: suppose the conclusion is falsifiable, so $(h_1, h_2 \triangleright h_0)$, $h_1 : ds(e_1, e_2)$ and $h_0 : ls(e_1, e_2)$ are true. Hence $\rho(h_1) \circ \rho(h_2) = \rho(h_0)$. There are two cases for $ds(e_1, e_2)$: either (1) $(e_1 \mapsto e_2, \omega)$ or (2) $ls(e_1, e_2)$. For the first case, h_1 must be a singleton heap (which means h_0 must be non-empty), so h_2 would be the subtraction of h_1 from h_0, which is a list $ls(e_2, e_3)$. For the second case, (2.1) if $\rho(h_0)$ is empty, then both $\rho(h_1)$ and $\rho(h_2)$ are empty by indivisible unit. So by definition of lists $e_1 = e_2 = e_3$. Therefore $h_2 : ls(e_2 = e_3)$ is true. (2.2) If $\rho(h_0)$ is non-empty but $\rho(h_1)$ is empty, then $\rho(h_2) = \rho(h_0)$, and $e_1 = e_2$. So $h_2 : ls(e_2, e_3)$ is true. (2.3) If both $\rho(h_0)$ and $\rho(h_1)$ are non-empty, then $\rho(h_2)$, should it be empty or not, would still make $ls(e_2, e_3)$ true. Thus the premise is falsifiable as well.

LS_7: Similar to LS_6.

TR_4: Suppose the conclusion is falsifiable, then $h : tr(e_1); h : tr(e_2)$ are true. If $\rho(h)$ is an empty heap, then $e_1 = e_2 = \text{nil}$. If $\rho(h)$ is non-empty, then again $e_1 = e_2$. So $[e_1/e_2]$ on the premise preserves falsifiability.

TR_5: Suppose the conclusion is falsifiable, then $(h_1, h_2 \triangleright h_0)$, $h_1 : (e \mapsto e_1, e_2, \omega)$ and $h_0 : tr(e)$ are true. Since $\rho(h_1) \circ \rho(h_2) = \rho(h_0)$ and $\rho(h_1)$ is a singleton heap (from non-Reynolds's semantics), $\rho(h_0)$ must also be non-empty. So $\rho(h_2)$, whether empty or not, would make $tr(e_1) * tr(e_2)$ true. Thus the premise is also falsifiable.

For the completeness proof, we show that each rule in the entailment checking system of Smallfoot can be mimicked by the rules in $LS''_{SL} + DS$. Since the entailment checking system of Smallfoot is complete w.r.t. the symbolic heap fragment [8], we deduce that our system $LS''_{SL} + DS$ is also complete w.r.t. the symbolic heap fragment. The unrestricted system $LS'_{SL} + DS$ is then complete w.r.t. the same fragment, too.

First, we translate a symbolic heap entailment into our labelled sequent as follows:

$$\tau(\Pi \wedge \Sigma \vdash \Pi' \wedge \Sigma') = h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge \Sigma'$$

It is easy to see that this translation is faithful. That is, given two symbolic heaps $\Pi \wedge \Sigma$ and $\Pi' \wedge \Sigma'$, $\Pi \wedge \Sigma \rightarrow \Pi' \wedge \Sigma'$ is a valid separation logic formula iff $\Pi \wedge \Sigma \vdash \Pi' \wedge \Sigma'$ is a valid entailment. By the $\rightarrow R$ rule in LS'_{SL}, this also implies that $h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge \Sigma'$ is the labelled sequent we need to derive. Now for each rule in the entailment checking system of Smallfoot of the form

$$\frac{P_1 \cdots \qquad \cdots P_n}{C}\ {}_r$$

where n is 1 or 2. We show that if all the translated premises $\tau(P_1)$ to $\tau(P_n)$ are derivable in $LS''_{SL} + DS$, then the translated conclusion $\tau(C)$ is also derivable in $LS''_{SL} + DS$. Most of the rules are easy to show, but the cases for the $*$-introduction rule and the last four rules (Table 3 in [8]) in the entailment system of Smallfoot are non-trivial. We first give an example for the easy case here, consider the following rule in Smallfoot:

$$\frac{\Pi \wedge \Sigma \vdash \Pi' \wedge \Sigma'}{\Pi \wedge \Sigma \vdash \Pi' \wedge (tr(\text{nil}) * \Sigma')}$$

We need to show that if there is a derivation for $h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge \Sigma'$ in $LS''_{SL} + DS$, then $h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge tr(\text{nil}) * \Sigma'$ is also derivable in $LS''_{SL} + DS$. Let us start from the translated conclusion

$$; h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge (tr(\text{nil}) * \Sigma')$$

and apply $\wedge R$ backwards. We obtain two branches. The first one is

$$; h : \Pi \wedge \Sigma \vdash h : \Pi'$$

and the second branch is a derivation as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{;h:\Pi\wedge\Sigma\vdash h:\Sigma'}{(\epsilon,h\rhd h);(h,\epsilon\rhd h);h:\Pi\wedge\Sigma\vdash h:\Sigma';h:tr(\mathrm{nil})*\Sigma'}\,\text{Lem 7.3.3} \quad \dfrac{}{\cdots;\cdots\vdash\epsilon:tr(\mathrm{nil});\cdots}\,TR_2}{(\epsilon,h\rhd h);(h,\epsilon\rhd h);h:\Pi\wedge\Sigma\vdash h:tr(\mathrm{nil})*\Sigma'}\,{*R}}{(h,\epsilon\rhd h);h:\Pi\wedge\Sigma\vdash h:tr(\mathrm{nil})*\Sigma'}\,E}{;h:\Pi\wedge\Sigma\vdash h:tr(\mathrm{nil})*\Sigma'}\,U$$

For the two open branches, invertibility of $\wedge R$ says that we only need to derive $h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge \Sigma'$, which by assumption is derivable.

Now let us consider the $*$-introduction rule in Smallfoot:

$$\dfrac{S \vdash S' \qquad \Pi \wedge \Sigma \vdash \Pi' \wedge \Sigma'}{\Pi \wedge (S * \Sigma) \vdash \Pi' \wedge (S' * \Sigma')}$$

Assume that there are derivations D_1 and D_2 in $LS''_{SL} + DS$ respectively for $h_1 : S \vdash h_1 : S'$ and $h_2 : \Pi \wedge \Sigma \vdash h_2 : \Pi' \wedge \Sigma'$, for some labels h_1, h_2. We first use the following backward proof search from the translated conclusion:

$$\dfrac{\dfrac{h:\Pi;h:(S*\Sigma)\vdash h:\Pi' \qquad h:\Pi;h:(S*\Sigma)\vdash h:S'*\Sigma'}{h:\Pi;h:(S*\Sigma)\vdash h:\Pi'\wedge(S'*\Sigma')}\,{\wedge R}}{h:\Pi\wedge(S*\Sigma)\vdash h:\Pi'\wedge(S'*\Sigma')}\,{\wedge L}$$

We give the left branch the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{h_2:\Pi\wedge\Sigma\vdash h_2:\Pi'}{h_2:\Pi;h_2:\Sigma\vdash h_2:\Pi'}\,\text{Lem 7.3.4}}{h_4:\Pi;h_4:\Sigma\vdash h_4:\Pi'}\,\text{Lem 7.3.1}}{h:\Pi;h_4:\Sigma\vdash h:\Pi'}\,\text{Lem 7.3.5}}{(h_3,h_4\rhd h);h:\Pi;h_3:S;h_4:\Sigma\vdash h:\Pi'}\,\text{Lem 7.3.3}}{h:\Pi;h:(S*\Sigma)\vdash h:\Pi'}\,{*L}$$

And the right branch is derived as below:

$$\dfrac{\dfrac{\dfrac{D_1}{h_1:S\vdash h_1:S'}\,\text{Lem 7.3.1}}{h_3:S\vdash h_3:S'}\,\text{Lem 7.3.3} \quad \dfrac{\dfrac{\dfrac{\dfrac{h_2:\Pi\wedge\Sigma\vdash h_2:\Sigma'}{h_2:\Pi;h_2\Sigma\vdash h_2:\Sigma'}\,\text{Lem 7.3.4}}{h_4:\Pi;h_4:\Sigma\vdash h_4:\Sigma'}\,\text{Lem 7.3.1}}{h:\Pi;h_4:\Sigma\vdash h_4:\Sigma'}\,\text{Lem 7.3.5}}{\cdots;h:\Pi;h_4:\Sigma\vdash h_4:\Sigma';\cdots}\,\text{Lem 7.3.3}}{\cdots;h_3:S\vdash h_3:S';\cdots}\,{*R}}{(h_3,h_4\rhd h);h:\Pi;h_3:S;h_4:\Sigma\vdash h:S'*\Sigma'}\,{*L}}{h:\Pi;h:(S*\Sigma)\vdash h:S'*\Sigma'}$$

The derivations for the two open branches are obtained by using invertibility of $\wedge R$ on the end sequent of D_2.

Finally, we show the case for the following rule in Smallfoot. The other non-trivial cases can be argued similarly.

$$\frac{\Pi \wedge (ls(e_1, e_2) * \Sigma) \vdash \Pi' \wedge (ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma')}{\Pi \wedge (ls(e_1, e_2) * \Sigma) \vdash \Pi' \wedge (ls(e_1, \text{nil}) * \Sigma')}$$

We apply the rules in $LS''_{SL} + DS$ backwards on the translated premise, obtaining the following derivation:

$$\frac{\dfrac{h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : \Pi' \qquad h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma'}{h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : \Pi' \wedge (ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma')}\,\wedge R}{h : \Pi \wedge (ls(e_1, e_2) * \Sigma) \vdash h : \Pi' \wedge (ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma')}\,\wedge L$$

By the invertibility of $\wedge L$ and $\wedge R$, we know that the top sequent in both branches are derivable in $LS''_{SL} + DS$. Now let us do backward proof search on the translated conclusion, giving the following derivation:

$$\frac{\dfrac{h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : \Pi' \qquad h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : ls(e_1, \text{nil}) * \Sigma'}{h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : \Pi' \wedge (ls(e_1, \text{nil}) * \Sigma')}\,\wedge R}{h : \Pi \wedge (ls(e_1, e_2) * \Sigma) \vdash h : \Pi' \wedge (ls(e_1, \text{nil}) * \Sigma')}\,\wedge L$$

The left premise is derivable as we deduced above, so we only need to show that if
(S_1) $h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma'$
is derivable in $LS''_{SL} + DS$ then
(S_2) $h : \Pi; h : ls(e_1, e_2) * \Sigma \vdash h : ls(e_1, \text{nil}) * \Sigma'$
is also derivable in $LS''_{SL} + DS$.

Assuming S_1 is derivable, by Lemma 7.3.6, one possibility is that S_1 is derivable by only using negative rules, in which case S_2 would be derivable by using the same rule applications. If the above does not hold, for each open branch, let the negative closure of the top sequent be $\mathcal{G}; \Gamma \vdash h : ls(e_1, e_2) * ls(e_2, \text{nil}) * \Sigma'\theta; \Delta$, where θ is the series of substitutions in the proof search for negative rules. Since the process of applying negative rules in $LS''_{SL} + DS$ is terminating, the negative closure of any sequent is finite. Again by (multiple applications of) Lemma 7.3.6 and the invertibility of the negative rules in $LS''_{SL} + DS$, there are some labels h_1, h_2, h_3 such that $(h_1, h_2 \triangleright h_4), (h_4, h_3 \triangleright h\theta)$ occur in $\mathcal{G}$ such that the following are derivable:

1. $\mathcal{G}; \Gamma \vdash h_1 : ls(e_1, e_2)\theta; \Delta$
2. $\mathcal{G}; \Gamma \vdash h_2 : ls(e_2, \text{nil})\theta; \Delta$
3. $\mathcal{G}; \Gamma \vdash h_3 : \Sigma'\theta; \Delta$

Apparently h_1 should be the same as the label of $ls(e_1, e_2)\theta$ in Γ to match the right hand side: if the branch (1) is closed by id_s, this is obvious; otherwise the branch (1) can only be closed by LS_2, which means $h_1 = \epsilon$ and $e_1 = e_2$ (syntactically), and the label for $ls(e_1, e_2)\theta$ is ϵ on both sides. It is easy to see that the negative closure of the corresponding sequent in the proof search for S_2 is $\mathcal{G}; \Gamma \vdash h : ls(e_1, \text{nil}) * \Sigma'\theta; \Delta$. We then apply $*R$ backwards on this sequent, obtaining two premises $\mathcal{G}; \Gamma \vdash h_4 :$

$ls(e_1, \text{nil})\theta; \cdots; \Delta$ and $\mathcal{G}; \Gamma \vdash h_3 : \Sigma'\theta; \cdots; \Delta$. The latter sequent is derivable because (3) is derivable and weakening is built in. We apply LS_8 on the former to obtain

$$\mathcal{G}; \Gamma \vdash h_2 : ls(e_2, \text{nil})\theta; h_4 : ls(e_1, \text{nil})\theta; \cdots; \Delta$$

which is also derivable because of weakening and the derivation for (2). Thus we conclude that S_2 is derivable in $LS''_{SL} + DS$. □

Coming back to LS_{SL}, which is not complete w.r.t. Reynolds's SL, it is certainly incomplete when data structures like linked lists and binary trees are considered. The following valid formula, for example, is not provable in $LS_{SL} + DS$:

$$\neg(ls(e_1, e_2) \wedge ((e_3 \mapsto e_3) * \top)) \tag{7.10}$$

This formula says that it is not possible that the current heap is a list from the value of e_1 to the value of e_2, and the current heap also contains the singleton heap $e_3 \mapsto e_3$. Suppose the current heap is empty, then it certainly cannot contain a singleton heap. Otherwise the list $ls(e_1, e_2)$ must be non-empty, but then $e_3 \mapsto e_3$ cannot be a subheap in the list, because a list is required to be acyclic. One can certainly add rules to handle this situation. For example, the following rule would help give a derivation of Formula 7.10:

$$\frac{}{(h_1, h_2 \triangleright h_0); \mathcal{G}; \Gamma; h_0 : ls(e1, e2); h_1 : (e \mapsto e) \vdash \Delta}$$

But adding this rule cannot possibly give us completeness for a non-recursively enumerable logic. It is again a design decision whether to use this rule or not.

7.4 Proof Search and Optimisations

In this section we describe proof search and automated reasoning based on our labelled system $LS'_{SL} + DS$. These tactics can also be used for the variant $LS_{SL} + DS$.

7.4.1 Proof Search and Implementation

We have implemented our labelled calculus $LS'_{SL} + DS$ as a prover called Separata+, in which several restrictions for the logical and structural rules are incorporated without sacrificing provability. See Figure 7.1 for the related inference rules in LS_{SL}. Some of these restrictions are also used in our prover Separata for LS_{PASL} (cf. Section 6.5). The rule U only creates identity relations for existing labels. The rule A is only applicable when the following holds: if the principal relational atoms are $(h_1, h_2 \triangleright h_0)$ and $(h_3, h_4 \triangleright h_1)$, then the conclusion does not contain $(h_3, h \triangleright h_0)$ and $(h_2, h_4 \triangleright h)$, or any commutative variants of them, for any label h. These forbidden cases for the restricted U, A rules can easily be shown admissible for the unrestricted versions, since we have rules P, C.

In formulating the heuristic proof search, it is useful to define a notion of height for labels. Given a set $\mathcal{G}$ of ternary relational atoms, for two distinct labels h, h' that occur in $\mathcal{G}$, we write $h <_{\mathcal{G}} h'$ if $(h, h'' \triangleright h') \in \mathcal{G}$ for some label h'' such that $h'' \neq \epsilon$ and $h'' \neq h'$, and say that h is a *child* of h' and h' is a *parent* of h. A label without any child is called a *leaf* label, a label that is not the child of any other labels is called a *root* label. We have seen similar notions that view ternary relational atoms as a tree structure in Section 4.5. When the relation $<_{\mathcal{G}}$ is well-founded (i.e., there is no cycle), we define the *height* of a label as the longest distance from the label to a leaf label, where distance is measured by the number of labels in the chain of the $<_{\mathcal{G}}$ relation.

Lee and Park [65] give a detailed explanation of how cross-split should be applied. Here we use a different approach to simply enforce that each time we apply the rule CS, we choose the principal relational atoms such that the parent label has the lowest height. Our approach does not strictly follow Lee and Park's method but employs similar ideas.

Calcagno et al. [26] provide a way to deal with $-\!\!*$ formulae in the quantifier-free fragment, but we do not know whether their results hold for the separation logic in our discussion. Nevertheless, inspired by their result, the rules HE, HC in our prover are driven by $-\!\!*$ formulae in the antecedent. Given a labelled formula $h : A -\!\!* B$ in the antecedent of a sequent, we first compute the *size* of this formula as below:

$$
\begin{array}{ll}
|e \mapsto e'| = |e \mapsto e', e''| = 1 & |e = e'| = 0 \\
|\bot| = 0 & |\top^*| = 1 \\
|A \to B| = max(|A|, |B|) & |\exists x.A| = |A| \\
|A * B| = |A| + |B| & |A -\!\!* B| = |B|
\end{array}
$$

We allow to use the HE rule to extend h for at most $max(|A|, |B|)/2 + 1$ times instead of $max(|A|, |B|)$ as indicated in [26], because we do not worry about completeness w.r.t. SL here. The HC rule is restricted to only combine three types of heaps: any singleton heaps that occur as subformulae of $A -\!\!* B$; any heaps created by HE for $A -\!\!* B$; and any compositions of the previous two.

The atomic formula $e \mapsto \_$ in the symbolic heap fragment is translated to $\exists x.(e \mapsto x)$ in our language. This type of formulae is the only one in the symbolic heap fragment that require our quantifier rules. Since nested quantifiers are forbidden in the reasoning for symbolic heap, we demand that when proving a symbolic heap formula, the $\exists R$ rule only instantiates the quantified variable to an existing expression or the constant nil. We call this restricted version $\exists R'$. Also, when proving a symbolic heap formula, we disable the rules $-\!\!* L$, $-\!\!* R$, CS, HE, and HC, because these rules are never used in the proof search for symbolic heap. Although not explicitly allowed in the symbolic heap fragment nor in our assertion logic, some symbolic heap provers can recognise numbers, which is useful when verifying programs. To match them, we check when a rule wants to globally replace a number (expression) by another number, and close the branch immediately because two distinct numbers should not be

made equal. The rule $cut_=$ is restricted to only apply on existing expressions and the constant nil.

Overall, we forbid rule applications that do not create new labelled formulae. Substitutions are always welcomed, though.

Our proof search procedure for $LS'_{SL} + DS$ builds in the above restrictions, and applies the first applicable rule in the following order:

1. Any zero-premise rule.
2. Any unary rule that involves global substitutions.
3. Any other unary non-structural rule except $\exists R$.
4. Any binary rule that involves global substitutions except $cut_=$.
5. $\rightarrow L$.
6. $*R$, $-\!\!* L$ and $\exists R'$.
7. U, E, A, CS.
8. $cut_=$ and $\exists R$.

Theorem 7.4.1 (Termination for symbolic heap)**.** *The proof search procedure for $LS'_{SL} + DS$ is complete and terminating for the symbolic heap fragment.*

Proof. First we show that the various restrictions in our proof search procedure does not affect completeness w.r.t. the symbolic heap fragment. The restriction for rules A, U obviously preserves completeness, as already shown in Section 6.5. Our method to apply CS has no effect on the symbolic heap fragment, since the situation of cross-split is never encountered when proving a symbolic heap formula. Similarly, the way we build HE, HC into $-\!\!* L$ is irrelevant. The restriction on $cut_=$ to only apply on existing labels does not reduce provability of our system. Suppose there is a $cut_=$ application that runs as follows:

$$\frac{\mathcal{G}; \Gamma[e_1/e_2] \vdash \Delta[e_1/e_2] \qquad \mathcal{G}; \Gamma \vdash h : e_1 = e_2; \Delta}{\mathcal{G}; \Gamma \vdash \Delta}\, cut_=$$

Suppose one of e_1, e_2 does not occur in the conclusion, then $h : e_1 = e_2$ in the succedent will never be used in proof search. This $cut_=$ application is admissible, because the derivation for the right premise can be used to derive the conclusion. Our last restriction concerns the $\exists R$ rule, which is only used when an atomic formula of the form $(e \mapsto \_)$ occurs in the succedent. We argue that instantiating a variable to a fresh expression does not help the proof search for a symbolic heap formula. Suppose the $\exists R$ rule does create a new formula $h : (e \mapsto e')$ such that e' does not occur in the conclusion. By inspection on the rules in $LS'_{SL} + DS$, if the end sequent consists of symbolic heaps, then any formula in the antecedent of any sequent in proof search only involves those expressions that occur in the end sequent, or the fresh ones created by $\exists L$, or nil. None of these expressions can be equal to e'. Furthermore, the only rules that may use $\mapsto$ atomic formula in the succedent are id, id_2, and id_s. So $(e \mapsto e')$ in the succedent can not be used in proof search for the symbolic heap fragment at

all. From this we deduce that our restriction on $\exists R$ preserves completeness for the symbolic heap fragment.

Now we show that the proof search procedure described in Section 7.5 is terminating. Let $h : \Pi \wedge \Sigma \vdash h : \Pi' \wedge \Sigma'$ be a symbolic heap labelled sequent. It is easy to see that we can only apply $*L$ on this sequent for finitely many times, creating only finitely many labels. Besides $*L$, the only applicable rule that is able to create new labels is A. Since Σ consists of atomic $\mapsto$, list, or tree formula connected by just $*$, the $*L$ applications result in a binary tree structure of ternary relational atoms. It is obvious that the structural rules $E, A, U, Eq_1, Eq_2, D, P, C$ can only be applied finitely many times on these ternary relational atoms. More specifically, each E, A rule application creates a variant of the binary tree structure of ternary relational atoms. Each variant represents a different way to bracket and commute the same set of leaves, so there can only be finitely many variants. This means that $*R$ can only be applied finitely many times. As a result, we will reach a point where no more fresh labels can be introduced. Since we restrict the way to use $\exists R$, the only rule that is able to create fresh expressions is $\exists L$. Since there are only finitely many subformulae of the form $(e \mapsto \_)$ in Σ, $\exists L$ can only be applied finitely many times, generating finitely many fresh expressions. As a result, proof search for any formula in the symbolic heap fragment only involves finitely many labels and finitely many expressions. Then it is easy to see that the rules LS_8, TR_6 can only create finitely many new labelled formulae in proof search, thus these two rules can only be applied finitely many times. For the other rules, either they are not used in proof search for the symbolic heap fragment (such as $-\!\!* L$, $-\!\!* R$, HE, HC, LS_5, LS_6, LS_7, TR_4, TR_5), or they only involve substitutions, which reduce the number of labels or expressions, and can only be applied finitely many times. We then conclude that in the proof search for the symbolic heap fragment, any (allowed) rules can only be applied finitely many times, thus the proof search is terminating. □

Our system can be extended with rules to prove formulae 7.8 and 7.10 as mentioned previously, the resulting experimental prover can prove the former formula in 0.5 second, and the latter formula in 0.01 second. However, these rules will not be used when comparing our work with other provers in the next section.

7.4.2 Optimisations

The prior work on theorem proving for abstract separation logics by Park et al. [77, 65] and our work in Chapter 4 and 6 has raised the question of how efficiently can those methods deal with multiplicative connectives ($\top^*$, $*$ and $-\!\!*$). To solve problems with multiplicative connectives faster, we proposed a heuristic method for our free-variable system $FVLS_{BBI}$ for BBI (cf. Section 4.4 and 4.5). As we do not use free-variables here, the heuristic method for $FVLS_{BBI}$ cannot be directly used in Separata+, but we can still utilise the idea and use some features dedicated to separation logic semantics to give a fast $*R$ rule described as below.

By the nature of our proof search procedure, when we try to apply the rule $*R$ on $h : A * B$ in a sequent

$$\mathcal{G}; \Gamma \vdash h : A * B; \Delta$$

the antecedent Γ only contains atomic formulae and $-\!\!*$ formulae, the succedent Δ only contains atomic formulae and $*$ formulae. We first search for all the atomic or $-\!\!*$ subformulae of A (resp. B), obtaining a set Sub_A (resp. Sub_B). We then try to find a set L_A (resp. L_B) of labels from Γ for each formula in Sub_A (resp. Sub_B). That is, if $h' : A' \in \Gamma$ and $A' \in Sub_A$, then we say h' is the label[2] for A'. If we can find the labels for every formula in Sub_A and Sub_B, then we will proceed as the heuristic method for BBI suggests: we try to find a binary tree structure of ternary relational atoms in $\mathcal{G}$ that has exactly the set of leaves $L_A \cup L_B$ and root h, then create the ternary relational atom $(h_1, h_2 \triangleright h)$, and build a subtree for h_1 (resp. h_2) that has leaves L_A (resp. L_B). This is guaranteed to be sound by the rules A and E. Then we apply $*R$ on $h : A * B$ and $(h_1, h_2 \triangleright h)$. If we can only find the labels for every formula in Sub_A (symmetrically for Sub_B), then we search for ternary relational atoms in $\mathcal{G}$ that form a binary tree with root h and leaves L such that $L_A \subseteq L$. If found, we deduce that labels in L_A represent heaps that are disjoint from each other. Then it is sound to build a subtree for h_1 with leaves L_A, and a subtree for h_2 with leaves $L \setminus L_A$, and $(h_1, h_2 \triangleright h)$. Again, we apply $*R$ on $h : A * B$ and $(h_1, h_2 \triangleright h)$. In this way, the structural rules E, A are driven by the fast $*$ rule applications.

Analysing the subformulae in $A -\!\!* B$ is a different matter, since the heaps that subformula A talks about may be disjoint from all existing heaps. Nonetheless, some existing work on program verification with $-\!\!*$ relies on the simple principle $A * (A -\!\!* B) \to B$ [67], which indicates that the following rule may help:

$$\frac{(h_1, h_0 \triangleright h_2); \mathcal{G}; \Gamma; h_1 : A; h_0 : A -\!\!* B; h_2 : B \vdash \Delta}{(h_1, h_0 \triangleright h_2); \mathcal{G}; \Gamma; h_1 : A; h_0 : A -\!\!* B \vdash \Delta} \; {}^{-\!\!* L'}$$

In addition, we also simplify the formula after parsing it using the following rules:

$$ls(e, e) \equiv \top^* \qquad tr(\text{nil}) \equiv \top^*$$
$$\top^* * A \equiv A * \top^* \equiv A \qquad \top^* -\!\!* A \equiv A$$

7.5 Experiments

The experiments in this section were run on a machine with a Core i7 2600 3.4GHz processor and 8GB memory, in Ubuntu 14.04. The code is written in OCaml, no concurrent computation is used.

In the previous sections we have developed two branches of systems. $LS_{SL} + DS$ is consistent with Reynolds's semantics, while $LS'_{SL} + DS$ mimics the interpretation of multi-field points-to predicate from symbolic heap fragment. We have implemented

[2] It is possible that a formula in Γ occurs with different labels, we only need one of them.

provers for both types of systems, one is an easy modification of the other. In the following, for the purpose of comparing our prover with other provers for symbolic heap, we use the prover based on $LS'_{SL} + DS$.

Experiment on Benchmarks in the Literature Our first experiment compares our prover with the current state-of-the-art provers for symbolic heap. We choose the Clones benchmark from Navarro and Rybalchenko [70]. The formulae generated by this benchmark come from "real life" list manipulating programs and specifications involved in verification. We filter out problems that contain a data structure that we do not consider in this chapter, the remaining set consists of 164 valid formulae and 39 invalid formulae. Each Clones test set has the same type of formulae, but the length (number of copies of subformulae) of formulae increases from Clones 1 to Clones 10.

Several provers for separation logic have been compared in the literature [70, 71, 90], including jStar [31], Smallfoot [7], VeriStar [90], SLP [70][3], and Asterix [71], most of which are for symbolic heap. From these comparisons we can conclude that Asterix (an improved version of SLP) is significantly faster than others in terms of solving symbolic heap formulae, VeriStar and Smallfoot can outperform each other in different situations, jStar often comes in the last position. Here we compare our prover with Asterix, Smallfoot, as well as Brotherston et al.'s prover $\text{Cyclist}_{\text{SL}}$ [19]. It is not easy to put these competitors in the same race, since they are designed for different fragments of separation logic. For example, $\text{Cyclist}_{\text{SL}}$ cannot recognise numbers, and there are 17 formulae in each Clones test set that cannot be parsed by it (counted as not proved). We also want to emphasise that although Smallfoot and Asterix are designed for the symbolic heap fragment, their implementations use slightly different syntax and semantics. For example, both provers only support $*$ as the only conjunction, and force that formulae of the form $e = e'$ can only be true at the empty heap. This means that the following formula, which is valid in separation logic and provable by our prover, but not expressible in the symbolic heap fragment, is a legitimate formula that Smallfoot and Asterix can parse, but is actually not provable by these two provers, nor can it be proved by $\text{Cyclist}_{\text{SL}}$:

$$(e \mapsto e') \to (e = e) \tag{7.11}$$

Table 7.1 shows the results of our first experiment. Time out is set as 50 seconds (but $\text{Cyclist}_{\text{SL}}$ has a timeout mechanism built in, which seems to be 30 seconds). The proved column for each prover shows the number of formulae the prover completed proving/disproving within the time out, the avg. time column shows the average time used when successfully proving a formula. Unsuccessful attempts (those that were timed out) are not counted in average time. Asterix outperformed all compared provers. $\text{Cyclist}_{\text{SL}}$ is not complete, so it might terminate without giving a proof. It also cannot determine if a formula is invalid. Separata+ and Smallfoot have similar

[3]The SLP in this section is the solver by Navarro et al., not the SL fragment by Galmiche and Méry also called SLP.

Test suite with 164 valid formulae						
Test suite	Separata+		$\text{Cyclist}_{\text{SL}}$		Smallfoot	
	proved	avg. time	proved	avg. time	proved	avg. time
Clones 1	164	0.01	147	0.04	164	0.00
Clones 2	160	0.02	137	0.17	164	0.00
Clones 3	159	0.07	126	0.48	164	0.01
Clones 4	159	0.30	117	0.11	164	0.03
Clones 5	158	0.03	115	0.13	164	0.15
Clones 6	158	0.08	114	0.29	164	0.65
Clones 7	158	0.18	106	0.01	162	0.75
Clones 8	158	0.42	106	0.01	160	0.83
Clones 9	158	0.89	106	0.01	157	0.36
Clones 10	157	1.19	106	0.01	157	0.83

Test suite with 39 invalid formulae						
Test suite	Separata+		$\text{Cyclist}_{\text{SL}}$		Smallfoot	
	dis-proved	avg. time	dis-proved	avg. time	dis-proved	avg. time
Clones 1	39	0.09	0	-	39	0.00
Clones 2	23	3.37	0	-	39	0.00
Clones 3	9	1.78	0	-	39	0.01
Clones 4	6	7.89	0	-	39	0.02
Clones 5	2	0.52	0	-	39	0.10
Clones 6	2	20.10	0	-	39	0.40
Clones 7	0	-	0	-	39	0.00
Clones 8	0	-	0	-	38	2.10
Clones 9	0	-	0	-	38	5.37
Clones 10	0	-	0	-	32	3.54

Asterix proved every test set with an average of 0.01s and 100% successful rate.

Table 7.1: Experiment 1: the Clones benchmark. Times are in seconds.

performance on valid formulae, but Separata+ is not efficient on invalid formulae. The Clone 1 test set contains the original formulae extracted from program verification, both Separata+ and Smallfoot can easily prove/disprove these formulae. This means that, although our prover is not as fast as dedicated provers such as Asterix, and not as fast as Smallfoot when disproving large artificially made invalid formulae, our prover is reasonably fast when dealing with problems in real world applications.

Our procedure was tuned towards proving validity rather than counter-model generation. If soundness, completeness and termination are achievable, proof search is usually turned into a counter-model construction. However, our proof procedure is not complete in general (although it is complete for the symbolic heap fragment), so

	Formula
7.12	$ls(e_1, e_2) \wedge \top^* \wedge \neg(e_1 = e_2)$
7.13	$\neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \neg\top) \wedge ((e_1 \mapsto e_2) * \top)$
7.14	$(ls(e_1, e_2) * \neg ls(e_2, e_3)) \wedge ls(e_1, e_3)$
7.15	$ls(e_1, e_2) \wedge ls(e_1, e_3) \wedge \neg\top^* \wedge \neg(e_2 = e_3)$
7.16	$\neg(ls(e_1, e_2) \mathrel{-\!\!*} \neg ls(e_1, e_2)) \wedge \neg\top^*$
7.17	$\neg((e_3 \mapsto e_4) \mathrel{-\!\!*} \neg ls(e_1, e_4)) \wedge ((e_3 = e_4) \vee \neg ls(e_1, e_3))$
7.18	$\neg(\neg((e_2 \mapsto e_3) \mathrel{-\!\!*} \neg ls(e_2, e_4)) \mathrel{-\!\!*} \neg ls(e_1, e_4)) \wedge \neg ls(e_1, e_3)$
7.19	$\neg(\neg((e_2 \mapsto e_3) \mathrel{-\!\!*} \neg ls(e_2, e_4)) \mathrel{-\!\!*} \neg ls(e_3, e_1)) \wedge (e_2 = e_4)$
7.20	$\neg((e_1 \mapsto e_2) \mathrel{-\!\!*} \neg ls(e_1, e_3)) \wedge (\neg ls(e_2, e_3) \vee ((\top \wedge ((e_1 \mapsto e_4) * \top)) \vee (e_1 = e_3)))$
7.21	$\neg((ls(e_1, e_2) \wedge \neg(e_1 = e_2)) \mathrel{-\!\!*} \neg ls(e_3, e_4)) \wedge \neg(e_3 = e_1) \wedge (e_4 = e_2) \wedge \neg ls(e_3, e_1)$
7.22	$\neg(e_3 = e_4) \wedge \neg(ls(e_3, e_4) \mathrel{-\!\!*} \neg ls(e_1, e_2)) \wedge (e_4 = e_2) \wedge \neg ls(e_1, e_3)$
7.23	$\neg((ls(e_1, e_2) \wedge \neg(e_1 = e_2)) \mathrel{-\!\!*} \neg ls(e_3, e_4)) \wedge \neg(e_3 = e_2) \wedge (e_3 = e_1) \wedge \neg ls(e_2, e_4)$
7.24	$\neg(\neg((e_2 \mapsto e_3) \mathrel{-\!\!*} \neg ls(e_2, e_4)) \mathrel{-\!\!*} \neg ls(e_3, e_1)) \wedge (\neg ls(e_4, e_1) \vee (e_2 = e_4))$

Separata+ proved the negation of each listed formula within 0.01 second.

Table 7.2: Translated versions (via $A \mathrel{-\!\!\circledast} B \equiv \neg(A \mathrel{-\!\!*} \neg B)$) of selected formulae from Thakur et al. [91, Table 3].

failure of proof search does not give a counter-model. A redesign would be required to find counter-models for this logic, it is definitely interesting as future work.

Experiment on Examples with Magic Wand Our second experiment features some formulae outside the symbolic heap fragment, consequently we cannot find other provers to compare with, except for a recent work by Thakur, Breck, and Reps [91]. However, their prover does not employ Reynolds's semantics of separation logic, but instead restricts that heaps must be acyclic. For example, $(e_1 \mapsto e_2) * (e_2 \mapsto e_1)$ is a satisfiable formula in Reynolds's semantics, but is unsatisfiable in Thakur et al's semantics. The fragment of separation logic they consider has "septraction" $A \mathrel{-\!\!\circledast} B$, defined as $\neg(A \mathrel{-\!\!*} \neg B)$, and only allows classical negation on atomic formulae. This syntax is able to express overlapping data structures and some formulae used in rely/guarantee reasoning and program verification with fine-grained concurrency, but our syntax strictly contains theirs.

Formulae 7.3, 7.2, 7.5, 7.6, 7.7, 7.9, in earlier sections can all be proved by Separata+ within 0.01s, we are not aware of any existing automated tools that can deal with these formulae. Moreover, Thakur et al. gave a collection of unsatisfiable formulae [91, Table 3]. Our Table 7.2 shows a part of these formulae translated by the definition of septraction. Separata+ can prove the negation of each formula in our Table 7.2 within 0.01s. We ignore the other formulae from [91, Table 3], not because they are hard, but because they are not unsatisfiable in Reynolds's semantics, thus their negations are not valid. Formula 7.14 to 7.24 are identified as "beyond the scope of existing tools" by Thakur et al.. More specifically, Formula 7.12, 7.14 and 7.15 describe overlapping data structures; the other formulae in Table 7.2 demonstrate the use of list and septraction.

E.g., formula 7.17 is an instance of an elimination rule for $-\circledast$ and linked list [25].

Again, it is difficult to compare the performance between Separata+ and the tool by Thakur et al. because they are made for different logics (different semantics of separation logic). But it is worth noting that Thakur et al.'s tool proved the formulae in Table 7.2 on the order of 5 seconds on a 2-processor 2.27GHz Xeon machine (although only 1 core is used), which indicates that proving formulae in a larger fragment than symbolic heap could be much harder, but Separata+ runs on the order of just 0.01 second for these formulae, using a Core i7 2600 processor.

Maeda, Sato, and Yonezawa [67] provide more examples that use $-\!\!*$ in program verification. Many of their inferences, e.g., those in Section 3.1 of [67], can be easily proved by Separata+ if their syntax is carefully translated into ours. We give an example of an extracted formula below,

$$\begin{aligned}(ls(e_0,\mathrm{nil}) -\!\!* (ls(e_0,\mathrm{nil}) * (ls(e_0,\mathrm{nil}) -\!\!* ((ls(e_1,\mathrm{nil}) -\!\!* ls(e_2,\mathrm{nil})) * (e_1 \mapsto e_3) * \\ ls(e_0,\mathrm{nil}))) * (ls(e_0,\mathrm{nil}) -\!\!* ls(e_3,\mathrm{nil})))) \to (ls(e_0,\mathrm{nil}) -\!\!* (((ls(e_1,\mathrm{nil}) -\!\!* ls(e_2,\mathrm{nil})) \\ * (e_1 \mapsto e_3) * ls(e_0,\mathrm{nil})) * (ls(e_0,\mathrm{nil}) -\!\!* ls(e_3,\mathrm{nil})))) \quad (7.25)\end{aligned}$$

To challenge our prover more, we further test on larger formulae generated from Table 7.2 and the other formulae we mentioned in this section. Our generation method is inspired by the "clone" benchmark. We first convert each formula to an equivalent formula in the form $A \to B$. Then we make multiple copies of the l.h.s. and the r.h.s., giving a formula of the form

$$A_1 * A_2 * \cdots * A_n \to B_1 * B_2 \cdots * B_n$$

where the expressions in each pair of A_i, B_i are systematically renamed based on the original formula. It is known that provers for BBI based logics are vulnerable to commutativity and associativity of $*$ [53]. To make the formulae even more difficult, we mutate each formula by randomly switching the order of starred subformulae. It will cause our prover to apply more structural rules to discover the right splitting of heaps. We call these test suites "MClones", where M stands for mutated. The test results are shown in Table 7.3.

The MClones 1 set contains the original formulae mentioned in this section, while the MClones 10 set contains formulae that are 10 times larger than the original ones. We set the timeout as 50 seconds, and count the number of successfully proved formulae within the timeout, as well as the average time on successful attempts. Our prover can easily handle the original formulae, but the successful rate drops as the number of cloned subformulae increases. The average time used to prove a formulae, however, only fluctuates. The reason is that the cloned formulae are randomly mutated. If a mutated formula requires more associativity applications, then our prover will spend more time proving it; if the formula only requires commutativity applications, then it

Test suite	Separata+	
	proved	avg. time
MClones 1	13/13	0.003s
MClones 2	13/13	1.017s
MClones 3	13/13	0.030s
MClones 4	12/13	0.064s
MClones 5	11/13	0.894s
MClones 6	10/13	3.341s
MClones 7	10/13	0.685s
MClones 8	9/13	1.876s
MClones 9	8/13	0.746s
MClones 10	8/13	1.070s

Table 7.3: Mutated clones benchmark for formulae in Table 7.2.

is easier for our prover. For the MClones 10 set, our prover can still prove the majority of formulae in 1 second. The phenomenon that tested formulae are either too easy or too hard for provers seems to be prevalent in BBI and separation logic benchmarks. The reason may be that the formulae in Table 7.2, although are good examples to show what our prover is capable of, are not ideal benchmarks for testing the scalability of our prover. However, our previous experiment in Section 6.5 shows that for the same series of benchmarks, when the formulae become larger, Park et al.'s BBeye performs worse in terms of both successful rate and average time; while our prover Separata's average time fluctuates, only the successful rate drops. Thus this phenomenon may just be the characteristic of our provers.

7.6 Discussion and Related Work

Existing tools for Reynolds's semantics SL, such as Smallfoot [7], jStar [31], VeriStar [90], SLP [70], and Asterix [71], are all restricted to small fragments, most notably, the symbolic heap fragment proposed by Berdine et al. [8]. Other fragments include Brotherston et al.'s fragment with arbitrary inductive predicates [19], and more recently, Schwerhoff and Summers' fragment with $*, -\!\!*$ and $\rightarrow$ [89]. Besides these methods, Galmiche and Méry's resource graph based tableaux [37] deal with a fragment of SL that does not contain quantifiers and equality, but the details of proof search and automation have not been addressed. On the other hand, there are also existing tools that handle larger fragments than symbolic heap, but for non-Reynolds semantics, e.g., Lee and Park's theorem prover [65], and Thakur et al.'s unsatisfiability checker [91], cf. Section 7.2 and 7.5 for the differences in their semantics.

Besides the above fragments of SL, there are many other fragments and variants in the literature. For example, the "natural proofs" proposed by Pek et al. [81] is designed for a logic called *DRYAD*, which includes a number of binary relations such as $=, \neq, <, \leq, \subset, \subseteq$ on its terms, and includes the logical connectives/constants $\top, \bot$,

$\top^*, \wedge, \vee, *$. This fragment is a reasonably expressive one, although it does not have $\neg$, $-\!\!*$ and quantifiers. A remotely related work is Piskac et al.'s automated method for trees [82], whose language is a fragment of first-order logic that is decidable in NP. Their syntax, however, does not resemble much of Reynolds's separation logic, excluding both the points-to predicate $\mapsto$ and connectives/constants for heaps such as $\top^*$, $*$, and $-\!\!*$. Although they do have special predicates for trees, read and write actions etc., their method and ours cannot be compared.

There is a growing demand from the verification community to move beyond symbolic heap and to deal with $-\!\!*$, although this connective is ignored in the widely used symbolic heap fragment. Having $-\!\!*$ is a desirable feature, since many algorithms/programs are verified using this connective, especially when expressing tail-recursive operations [67], iterators [59], septraction in rely/guarantee [98] etc.. Moreover, $-\!\!*$ is very useful when verifying programs using weakest preconditions, which introduces $-\!\!*$ "in each statement in the program being analysed" [66]. See the introduction of [65] and [91] for other examples requiring $-\!\!*$. In addition to $-\!\!*$, allowing arbitrary combinations of logical connectives is also useful when describing overlapping data structures [49], properties such as cross-split can be useful in proof search in this setting [32]. Supporting quantifiers is also a step forward to inductively defined data structures, which are the centre of the new features brought by SL. Nevertheless, existing tools for SL with Reynolds's semantics do not support the reasoning for all logical connectives. Thus, an important area of research is to obtain a practical proof system for SL with all connectives.

However, as we have seen, supporting all the logical connectives in SL is not easy. As shown by Calcagno et al. [26] and Brochenin et al. [14], SL is not recursively enumerable in general, which means we cannot give a finite, sound, and complete proof system for this logic. Here we sacrificed completeness in order to give a proof method that is compliant with Reynolds's semantics and also useful in real applications from program verification. We built upon the previous labelled sequent calculi for propositional abstract separation logics (PASLs, cf. Chapter 6) by adding inference rules for quantifiers, equality, and the $\mapsto$ predicate. This is not trivial at all, as the latter involves heaps and stores in the semantics in a very subtle way, making this study error-prone. For example, during our investigation, we found that the resource graph tableaux [37], claimed to have complete rules for a fragment of separation logic, cannot prove some valid formulae using their original rules, although their rules can be modified to prove these formulae. Another example is Lee and Park's proof system, originally claimed to be complete for the quantifier-free fragment of SL [65], but which turned out to be unsound and incomplete w.r.t. Reynolds's semantics since their actual semantics is non-standard. These are discussed in Section 7.2.

Capturing data structures is also a desired feature since they are frequently used in program verification. We extended our proof system with treatments for singly linked lists and binary trees based on the entailment checking rules for Smallfoot [8],

but we also moved beyond symbolic heap to consider situations where overlapping data structures occur. Since the symbolic heap fragment is very popular, we showed that our proof method is complete w.r.t. this fragment. With certain restrictions on inference rules integrated, we gave a proof search procedure that is sound, complete, and terminating when proving a symbolic heap formula, therefore our proof method is strictly stronger than existing symbolic heap proof methods. Our experiments show that our implementation is competitive with Smallfoot on valid formulae when testing against a set of benchmarks extracted from real world program verification problems, but not so for invalid formulae. In addition, we demonstrated that our prover can deal with a much wider range of formulae than existing tools, thus it handles the largest fragment of SL (with the standard Reynolds's semantics) to date in terms of logical connectives and paves the way to more sophisticated program verification using Reynolds's SL.

Chapter 8

Conclusion and Future Work

We have discussed the conclusion and related work for each of Chapter 4, 6, and 7 in their last sections. Here we first summarise all the technical contributions in this dissertation, then we propose some future directions stemming from our line of research.

8.1 Conclusion

Our main objective was to give proof methods for solving the validity problem for separation logics. Our work began in late 2011, when many aspects of bunched logics and separation logics were still undiscovered.

We started by looking at bunched logics as they are the core of separation logic. Our initial plan was to give a nested sequent calculus for an extension of CBI with dual connectives, following Postniece's approach [83]: first convert a display calculus into a nested sequent calculus with shallow inference, then move to a deep inference system and consider proof search. The nesting of our sequents interleaves additive (nested) sequents and multiplicative (nested) sequents. We hoped that we could give a set of "propagation rules" to transfer information between additive and multiplicative sequents. This, however, was unsuccessful. By the nature of bunched logics, the additive structures and the multiplicative structures are mostly independent. There are very few cases where they can talk to each other. As a consequence, we moved on to formalise proof search using labelled sequents.

The advantage of using labels is that one can easily encode the semantics into the calculus. For this reason, some people do not consider labelled sequent calculi as proof theory. But in the case of bunched logics, labelled sequents saved our day and also became popular in Park et al.'s work [77, 65]. Interestingly, Park et al., like us, initially gave a nested sequent calculus for BBI, but later on changed to use labels. This coincidence hints that proof search using labels may be a better way for bunched logics, and that more work is needed to understand the proof theory of bunched logics. At the same time, we developed our labelled sequent calculus LS_{BBI} for BBI which is sound, complete, and enjoys cut-elimination. Unlike Park et al.'s labelled calculus for BBI, our structural rules directly encode the semantical properties in the

non-deterministic monoidal semantics of BBI. Compared to traditional labelled calculi, our calculus utilises global label substitutions to capture the equality of labels. These features make our method much easier to be extended to handle similar logics: we only need to add/remove some rules to capture the semantics. We also developed a free-variable system $FVLS_{BBI}$ to separate the proof search into two phases: giving a "pseudo-derivation" where there are free variables as undecided place holders for labels, and solving a constraint system to assign each free variable an actual label. With some heuristic proof search, our incomplete BBI prover using free variables is much faster than Park et al's prover when proving formulae that involve complicated multiplicative connective combinations, while their prover outperforms ours in some other cases.

Moving on, we captured, in a modular way, a range of abstract separation logics, all of which are extensions of BBI. The labelled calculi for these logics are easy to obtain, but the difficulty is in proving that our calculi are complete for the corresponding logics. For LS_{BBI}, completeness is proved by showing that the Hilbert system for BBI can be mimicked by LS_{BBI} using cut, and then showing cut-elimination. This method is not possible for most abstract separation logics because some properties in the semantics are shown to be inexpressible using BBI formulae. Inspired by Larchey-Wendling's work [60], we proved the completeness via a counter-model construction. The novelty in our proof is that we absorb the structural rules with substitutions into an equivalence relation, which allows us to prove the completeness for different logics modularly by just plugging in or removing some structural rules – the main structure of the proof does not need to be changed. With this advantage, we showed that our labelled method can handle every propositional abstract separation logic generated by any subset of DHA separation theory. The properties cross-split and splittability, however, require more work in our current setting. We plan to investigate new ways to formulate the proof system so that complicated properties can be captured easier.

Finally, we presented a labelled sequent calculus LS_{SL} for Reynolds's SL with the concrete heap model. The syntax allows all the logical connectives in SL including $*, -\!\!*$, quantifiers, the predicate $\mapsto$ and equality. It is impossible to obtain a finite, sound and complete sequent system for this logic [26], so we focused on soundness, usefulness, and efficiency. With the extension to handle linked lists and binary trees, our proof method is sound and complete w.r.t. the widely used symbolic heap fragment. Our proof search procedure, when properly restricted, is complete and terminating when proving symbolic heap formulae. We evaluated our implementation by first comparing it with provers dedicated to symbolic heap, testing against a set of formulae extracted from program verification. Our prover Separata+ showed comparable results as that from Smallfoot on proving valid formulae, although Separata+ does not perform well when the formula is invalid, which may be due to our inference rules having to cover a larger fragment. Then we showed that Separata+ is more versatile than any existing automated tools by illustrating many formulae that, to our knowl-

edge, no other provers for Reynolds's SL can prove, but Separata+ can easily prove. Some of these formulae are taken from existing (manual) proofs to verify algorithms/programs. These indicate that our method would be useful, at least as a part of the tool chain, for program verification with more sophisticated use of separation logic.

8.2 Future Work

Although we have solved some open problems about bunched logics and separation logics, there are still many unsolved problems and possible ways to improve existing methods. We gave a free variable system for BBI, but we have not implemented a complete prover based on the free variable method, nor have we extended the free variable method to handle other abstract separation logics. In fact, Alwen Tiu was very keen on giving a complete strategy for solving the constraint system in the free variable method, but I hesitated to go that direction because at that time we were not clear whether BBI should be our final target – we were not aware of Dockins et al's paper until mid 2012 when Ranald Clouston joined us – and for each separation theory, the way to solve the constraint system in the free variable method could be very different. Looking back, the complication of the free variable system for BBI arises from the fact that we have to keep track of the relational atoms in each sequent. For example, if we use associativity to generate $(u, w \rhd z); (v, y \rhd w)$ from $(x, y \rhd z); (u, v, \rhd x)$, where w is fresh, we have to accumulate the generated relational atoms upwards in the proof search. Otherwise if we generate similar relational atoms $(u, w' \rhd z); (v, y \rhd w')$ later, we cannot guarantee that w' acts the same way as the deleted w, thus some information is lost. With partial-determinism (and cancellativity), however, the above is no longer a problem, since this property ensures that w and w' are the same. This is exactly the reason why Larchey-Wendling and Galmiche's tableau method [61] for BBI_{PD} does not have to accumulate relational atoms in proof search. Therefore a free variable system for abstract separation logic with partial-determinism should be much cleaner than the one we gave for BBI_{ND}. Allowing indivisible unit and disjointness also simplifies the ternary relation in many cases. The once hard decision of which abstract logic should we look at depends on what concrete heap model we are interested in, which now clearly is Reynolds's model – so $BBI + P + C + IU + D + CS$ appears to be a promising angle. Although a complete constraint solving strategy for this logic may still be a challenge, this problem is now more interesting than we first thought it to be.

To study the properties in separation theories, Brotherston and Villard developed $HyBBI$ [22], which is BBI combined with hybrid logic. All the properties in DHA separation theory except for cross-split can be defined as $HyBBI$ formulae, the extended hybrid logic $HyBBI(\downarrow)$ is able to capture cross-split as a formula. Therefore they solved the axiomatisation problem of subsets of DHA separation theory by giving a Hilbert system in $HyBBI(\downarrow)$. However, there is not a proof search friendly calculus for $HyBBI(\downarrow)$ yet. The binder @ in hybrid logic attaches a world to a formula, while

the binder $\downarrow$ gives access to the world in which a formula is true. These are closely related to the presentation of our labelled sequent calculi, thus it is natural to extend our labelled method to handle hybrid BBI logics.

Our work on separation logic with Reynolds's semantics opens up many possibilities for future work as well. Handling arbitrary (inductively defined) predicates and address arithmetic is our next step. But incorporating them is highly non-trivial, especially for the former, which may require very complicated proofs for the correctness of integrating cyclic proofs in our framework and may lower the efficiency of our implementation. Hard-wired subtraction rules for data structures often yield shorter and simpler proofs with fewer side conditions to check, although they are not as flexible as a general theory that deals with predicates and inductive definitions [19]. Nevertheless, it would be interesting to see how our work could be incorporated into Brotherston et al.'s cyclic proof framework. Integrating other data structures to enrich the language our prover can handle is also future work. We have integrated some optimisations in our prover, but more can be done in terms of driving structural rule applications by logical rules. For example, one could inspect the subformulae of $A -\!\!* B$ and guess that the current heap should be the heap for B minus the heap for A. However, determining the heaps for A and B could be as hard as the proof search itself, so we leave this aspect as future work.

Another interesting direction would be integrating our prover into existing tool chains for program verification using separation logic. Currently most of the existing work concentrates on the symbolic heap fragment, but our prover can greatly enlarge the set of formulae existing tools can handle. Moreover, although our prover is not complete for our SL, it is complete for the symbolic heap fragment. So when it terminates on proving a symbolic heap formula without giving a closed derivation, we know that this formula is invalid. Therefore we can collect the information from the open branches and build a counter-model in separation logic. This may help program verifiers to better understand the problem in the verification. Similarly, even if our prover fails when proving a non-symbolic heap formula, the collected information from the open branches, although not necessarily a counter-model, may still help to pinpoint whether the formula is valid, whether more rules are required, etc.. Finally, with $-\!\!*$ in our language, our work in Chapter 7 enables reasoning with weakest preconditions for separation logic. This is an interesting topic that has not been investigated in the literature due to the lack of support for $-\!\!*$. A semi-automated tool for weakest preconditions in separation logic would be a future direction.